Rigid-Body Simulation-Based Probabilistic Analysis of Drone Crash Impact Points and Its Embedded System Application

Mu-Won Lee
Chungbuk Science High School, Cheongju, Republic of Korea
lmw.hpc@gmail.com

Abstract This paper presents an offline-to-embedded workflow for estimating the displacement of an unmanned aerial vehicle after total thrust loss. A C#/.NET simulator integrates rigid-body motion through BepuPhysics2 and uses Monte Carlo trials with randomized steady wind and stochastic turbulence. A grid sweep over initial altitude, horizontal speed, and vertical speed produces crash-location samples that are exported to CSV and summarized by a C99 lookup-table library. The originally reported experiment contains 62,912 trials over 64 altitude–horizontal-speed combinations and reports an overall mean displacement of 20.33 m and a sample 95th percentile of 49.02 m. The embedded library uses static buffers, grouped sample statistics, and trilinear interpolation to return estimated offsets and a radius proxy without running the physics simulator on the flight controller. Inspection of the published implementation reveals important distinctions between an empirical percentile and the normal-approximation radius used by the C library, as well as assumptions concerning interpolation, out-of-grid inputs, and simulation fidelity. The findings therefore demonstrate a computationally lightweight, simulation-based prediction pipeline, while the reported radii should not be interpreted as field-validated safety guarantees.

Keywords Drone crash prediction; embedded systems; ground impact probability; Monte Carlo method; rigid-body simulation.

1 Introduction

When a crash occurs, existing drones provide operators with GPS data right up to the moment of signal loss, and operators use this data to locate the drone. However, drones do not always fall vertically downward. Drones can crash in various locations due to the influence of wind, previous speed, and direction. Even if an operator attempts a search considering these factors, the lack of knowledge regarding where and with what probability the drone might be located can cause difficulties in a rapid search. If this environment is constructed through simulation and the drone's impact point is expressed in the form of a probability distribution to be provided to the operator upon a crash, it would be of great assistance in locating the drone. For this reason, this study was initiated with the idea that dropping a drone in a simulated environment, expressing the impact points as a probability distribution, and developing it into a C library would enhance convenience when applied to drone firmware development.

1.1 Problem Formulation and Scope

Let the failure state be $\mathbf{s} = (h, v_h, v_v)$, where h is altitude above a locally flat ground plane, $v_h \geq 0$ denotes horizontal speed, and v_v denotes signed vertical speed. Given a vehicle model $\boldsymbol{\theta}$ and uncertain environment $\boldsymbol{\xi}$, the simulator defines a forward mapping

$$\mathbf{p} = f(\mathbf{s}, \boldsymbol{\theta}, \boldsymbol{\xi}), \quad \mathbf{p} = (x_{\text{hit}}, y_{\text{hit}})^{\mathsf{T}}. \tag{1}$$

Rather than identifying a single deterministic landing point, the objective is to approximate conditional summaries of $\mathbf{p}$ for a specified flight state and set of modeled environmental uncertainties. The implementation uses an offline collection of simulations and an online interpolator. The intended operational use is prioritization of an aircraft search area after a presumed loss of thrust; it is not a validated third-party casualty-risk model or a demonstrated onboard crash detector.

1.2 Contributions and Evidence Boundaries

The contributions documented by the provided manuscript and public implementation are: (i) a BepuPhysics2-based simulator coupling rigid-body integration with stochastic wind; (ii) a configurable grid-sweep pipeline that exports individual crash samples; (iii) a fixed-capacity C99 statistical lookup table with trilinear interpolation; and (iv) reported simulation and holdout-interpolation summaries. Descriptions of implementation behavior below are based on inspection of the publicly available repository [10]; reported numerical outcomes are inherited from the experiment described in the original manuscript, not regenerated in the present revision. Hardware integration is discussed as a design pathway rather than a completed flight validation.

2 Related Work

As the operation of Unmanned Aerial Vehicles (UAVs) rapidly expands into commercial and civilian sectors, ground impact caused by system failures during flight

is emerging as a major factor threatening third-party safety. Dalamagkidis [1] systematically outlined the necessity of safety assessments for integrating UAVs into the national airspace, which became the starting point for subsequent research on UAV crash risk assessment.

Research aiming to quantitatively derive the probability distribution of crash impact points was initiated in earnest by La Cour-Harbo [2]. This study analytically derived the probability density function of the ground impact point in a ballistic descent situation by combining a second-order drag model with probabilistic assumptions of flight parameters. It is characterized by considering both the influence of wind and the uncertainty of the drag coefficient. In a subsequent study by the same author [3], this was expanded into a model that quantifies the fatality rate per unit of flight time based on the derived probability distribution.

Liu et al. [4] proposed an approach that explicitly deals with multi-uncertainty. Fluctuations in the drag coefficient, velocity at the time of the crash, and local wind effects were all treated as random variables, and by combining them, the distribution of the ground impact point was estimated.

As for applied research in urban environments, the ground risk map study by Primatesta et al. [5] is representative. This study proposed a method to generate a 2D risk map by combining the impact distribution model proposed by La Cour-Harbo [2] with population density, sheltering factors, and no-fly zones in urban areas, and utilizing this for path planning.

Standard methodologies for 6-Degrees of Freedom (6DOF) rigid-body dynamics simulations of UAVs are systematically organized in the books by Beard and McLain [6] and by Stevens et al. [11], and the simulator implementation in this study is based on the equations of motion and coordinate system definitions presented in this literature. As an atmospheric turbulence model, the Dryden model is widely used in the field of aviation simulation and is standardized in the U.S. Military Handbook MIL-HDBK-1797 [7]. Hakim and Arifianto [8] presented a case of implementing the Dryden continuous turbulence model in a simulation environment.

UAV crash analysis based on Monte Carlo simulations was first conducted by Rudnick-Cohen et al. [9]. This study calculated the statistical characteristics of the crash probability distribution by applying probabilistic inputs to a deterministic dynamics model and repeating multiple trials.

The differentiating factors of this study are as follows. While previous studies mainly adopted a method of deriving the probability density function of the impact point by directly applying probabilistic assumptions to an analytical drag model, this study combines 6DOF rigid-body physics simulations with the Monte Carlo method so that the empirical distribution emerges from the data solely through deterministic equations of motion and stochastic disturbance inputs, without prior assumptions about the distribution. This is based on the statistical convergence guaranteed by the Law of Large Numbers. Additionally, porting the derived distribution characteristics into a lightweight C library to estimate crash risk zones in real-time on an embedded flight controller is another point that differentiates this study from previous research. Table 1 summarizes how this study differs from representative prior work along these dimensions.

Table 1: Comparison with Representative Prior Work

Study	**Modeling Approach**	**Uncertainty Treatment**	**Embedded Deployment**
La Cour-Harbo [2]	Analytical 2nd-order drag model	Probabilistic assumptions on parameters	No
Liu et al. [4]	Analytical drag model	Multiple random variables combined	No
Primatesta et al. [5]	Analytical impact model + GIS	Inherited from [2]	No
Rudnick-Cohen et al. [9]	Deterministic dynamics + probabilistic inputs	Monte Carlo	No
This work	6DOF rigid-body simulation	Monte Carlo (no prior distributional assumption)	Yes (C99 LUT library)

3 System Design

The implemented system consists of two principal components: a PC simulator and an embedded C library. The PC simulator generates impact point data by repeatedly simulating drone crashes under various flight conditions, and the C library embeds this data in the form of a LUT to utilize it for real-time prediction. The overall system architecture, dividing the offline PC simulation environment and the real-time embedded system application, is shown in Fig. 1.

3.1 Drone Model Abstraction

The drone is abstracted into three physical parameters: mass, drag coefficient (C_d), and reference area (A). By specifying only these three values, the same simulator can be applied regardless of the aircraft's shape or size, ensuring versatility.

The flight state at the time of the crash is represented as FlightState, with altitude (h), horizontal speed (v_h), and vertical speed (v_v), which can be directly acquired via GPS/IMU, set as the main independent variables. Considering that attitude and angular velocity right before a crash are difficult to specify, the published grid-sweep implementation fixes the initial attitude to the identity orientation and the angular velocity to zero. The results therefore do not sample uncertainty in initial attitude or angular motion; averaging over environmental trials cannot by itself establish that such omitted effects vanish.

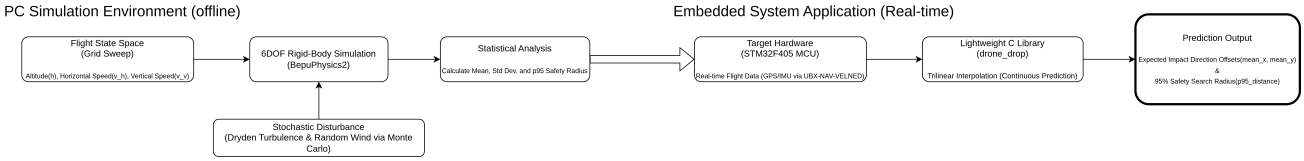


Figure 1: Overall system architecture diagram dividing the offline PC simulation environment and the real-time embedded system application.

3.2 Environment Model

The environment model consists of atmospheric conditions and a wind field. The standard-atmosphere implementation calculates density from altitude using a temperature lapse rate and a pressure ratio; 1.225 kg/m^3 is the reference sea-level density, not a density imposed at every height in the current source code.

Wind is expressed as the sum of a steady wind component defined by average wind speed and wind direction, and a stochastic turbulence component generated by the Dryden turbulence model. In each Monte Carlo trial, wind speed is independently sampled from a uniform distribution of $[0,\ v_{max}]$, and wind direction from a uniform distribution of $[0°,\ 360°]$ to reflect the uncertainty of the atmospheric environment.

The coordinate system uses domain coordinates (X = East, Y = North, Z = Up), and the transformation to/from the BepuPhysics2 internal coordinates (X = East, Y = Up, Z = North) is handled in the Physics layer.

3.3 Crash Scenario

The crash scenario assumes total thrust loss. From the moment the crash is detected, the motor thrust is immediately set to 0, and thereafter the drone falls under the influence of only gravity and air resistance.

The flight state space is exhaustively searched using a Grid Sweep method. The ranges for altitude, horizontal speed, and vertical speed are each divided into a grid, and for each grid combination, 983 Monte Carlo simulations are performed to estimate the impact point distribution. The results of each trial are saved as a CSV file, which is later used to construct the LUT for the C library.

4 Physics Simulation

4.1 Rigid-Body Dynamics

The physics simulation utilizes the rigid-body integrator of BepuPhysics2 v2.4.0. The drone is modeled as a rigid body with mass and an inertia tensor, and external forces are summed at each time step to numerically integrate the 6-DOF equations of motion.

The simulation terminates the moment the drone reaches the ground ($Z = 0$), and the (X, Y) coordinates at this point are recorded as the crash point. The crash point is expressed in relative coordinates with the crash start location as the origin.

4.2 Aerodynamic Model

Drag is calculated as shown in (2).

$$\mathbf{F}_d = -\frac{1}{2}\rho C_d A \left|\mathbf{v}_{rel}\right| \mathbf{v}_{rel} \tag{2}$$

Here, ρ is the air density, C_d is the drag coefficient, A is the reference area, and $\mathbf{v}_{rel}$ is the relative velocity, which is the difference between the drone's velocity and the wind velocity. The direction of the drag is opposite to the relative velocity, and its magnitude is proportional to the square of the relative velocity.

In this study, an isotropic drag model is applied. This assumes that the drag coefficient C_d and reference area A are constant regardless of the drone's attitude, based on the assumption that differences in drag by direction due to irregular attitude changes of the drone during a crash are averaged out during the Monte Carlo iterations.

4.3 Numerical Integration and Contact Criterion

The code applies force and torque impulses and advances the BepuPhysics2 simulation once per timestep. The general configuration uses $\Delta t = 0.01$ s, whereas the grid-sweep runner explicitly selects a faster batch configuration with $\Delta t = 0.04$ s and a maximum duration of 120 s [10]. After each integration step, terrain elevation is evaluated and the trial stops once the body position is at or below the terrain surface. In the published sweep, the environment uses a flat terrain. The terminal horizontal coordinates of the last simulated sample are recorded as the crash point. Because the implementation checks terrain after the timestep rather than solving for an exact time-of-impact root, a timestep-dependent positional discretization error is possible. No convergence analysis against smaller timesteps is reported in the supplied results.

4.4 Monte Carlo Sampling and Reproducibility

For each fixed combination of altitude, horizontal speed, and vertical speed, the sweep creates a new pseudo-random seed, samples steady wind speed uniformly between zero and a configurable maximum, and samples wind bearing uniformly over a full revolution. The initial horizontal velocity is aligned with the simulator's local X axis, the lateral velocity is zero, and vertical velocity follows the selected sweep level. Thus, the

outputs are naturally expressed relative to the initial horizontal travel direction; geographic bearings require an additional frame transformation. Individual sampled winds and terminal points are written to the CSV, supporting subsequent reconstruction of per-condition statistics. However, because the sweep generator is seeded from system time, identical numerical reruns require the relevant seed and configuration metadata to be retained. The manuscript's numerical tables alone do not specify enough information to reconstruct every random draw.

4.5 Turbulence Model

Atmospheric turbulence is implemented based on the Dryden turbulence model [7]. The Dryden model generates a time-domain turbulence velocity signal by passing band-limited white noise through a first-order IIR forming filter.

The 3-axis components of turbulence (u, v, w) are generated independently, and to eliminate correlations between trials, an independent random seed is used for each Monte Carlo trial. The generated turbulence velocity is added to the steady wind vector at each time step to constitute the instantaneous wind velocity acting on the drone.

4.6 Implemented Turbulence Recursion

The published `DrydenTurbulenceModel` uses three independently updated, first-order Gauss–Markov states. For each component, a discretized recursion in the code takes the form

$$\eta_{k+1} = \left(1 - \frac{\Delta t_t V}{L}\right)\eta_k + \sigma\sqrt{\frac{2\Delta t_t V}{L}}\, z_k, \quad z_k \sim \mathcal{N}(0,1), \tag{3}$$

where the turbulence update uses $\Delta t_t = 0.01$ s internally, V is bounded below by 1 m/s, L is a length scale, and σ is a component intensity. The source sets horizontal intensity to 1.5 m/s, vertical intensity to 1.0 m/s, horizontal length scale to 200 m, and vertical length scale to 50 m. The update is a simplified stochastic turbulence surrogate. In particular, the 0.01 s recursion value is distinct from the 0.04 s batch-physics timestep; this distinction should be preserved when describing implementation fidelity and comparing against fully specified Dryden spectral models. It should not be taken as a verification that the discrete process reproduces the complete Dryden spectrum.

5 Experiments and Results

5.1 Simulation Setup

The Grid Sweep consists of combinations of 8 altitude levels (10.4 120.3 m) and 8 cruise speed levels (0.0 20.0 m/s), and for each grid combination, approximately 983 Monte Carlo trials were performed. The entire dataset consists of 62,912 trials. The drone parameters

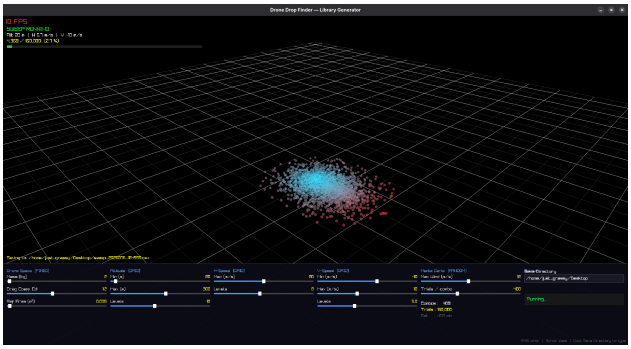


Figure 2: Execution screen of the developed BepuPhysics2-based Monte Carlo crash simulator showing generated impact point distribution.

Table 2: Simulation Parameters

Parameter	Value
Drone mass	0.6 kg
Drag coefficient C_d	1.1
Reference area A	0.03 m^2
Altitude range	10.4–120.3 m (8 levels)
Cruise speed range	0.0–20.0 m/s (8 levels)
Number of trials	Approx. 983 trials/combination
Total trials	62,912 trials

used in the simulation are shown in Table 2. The developed simulation environment and the generated empirical impact point distribution are visualized in Fig. 2.

5.2 Data Products and Statistical Estimators

The sweep writer exports the following ten columns in order: `altitude_m`, `h_speed_mps`, `v_speed_mps`, `trial_index`, `wind_speed_mps`, `wind_bearing_deg`, `crash_x_m`, `crash_y_m`, `crash_distance_m`, and `crash_bearing_deg`. For N impact points, the radial displacement is $d_i = \sqrt{x_i^2 + y_i^2}$ and the sample mean is

$$\bar{d} = \frac{1}{N}\sum_{i=1}^{N} d_i, \qquad \overline{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{p}_i. \tag{4}$$

The simulator's `DistributionEstimator` computes sample-coordinate variances with denominator $N-1$, covariance, and interpolated empirical quantiles of the ordered radial distances. With ordered distances $d_{(0)} \le \cdots \le d_{(N-1)}$, its percentile index is $q(N-1)$ and the code linearly interpolates between adjacent order statistics. Consequently, the empirical 95th radial percentile is an explicitly sample-defined statistic rather than a Gaussian assumption. This differs from the subsequently embedded approximation, as detailed in Sec. 6.8.

5.3 Impact Point Distribution

The main statistics for the fall distance distribution across all trials are as follows. The overall average fall

Table 3: Fall Distance Statistics by Altitude

Altitude (m)	Mean (m)	p95 (m)
10.4	8.76	20.88
26.1	13.37	31.25
41.8	16.81	38.53
57.5	19.72	45.20
73.2	22.00	48.77
88.9	24.67	54.10
104.6	27.27	58.30
120.3	30.03	62.80

distance is 20.33 m, with a standard deviation of 15.0 m, and the 95th percentile (p95) is 49.02 m.

The effect of altitude on the fall distance is summarized in Table 3. As altitude increases, both the fall distance and the p95 safety radius tend to monotonically increase. This is because the longer the fall time, the more the effects of wind and turbulence accumulate.

The effect of cruise speed is also evident. The average fall distance at a cruise speed of 0 m/s is only 9.73 m, but at 20 m/s, it increases to 33.93 m, an increase of approximately 3.5 times. The p95 safety radius also expands from 30.38 m to 65.70 m. This indicates that the horizontal momentum at the time of the crash has a dominant effect on the impact point distribution.

5.4 Interpretation of the Original Aggregates

The aggregate figures reported below combine flight states rather than describing one conditional distribution for a specific aircraft state. They are therefore useful for evaluating broad simulation trends, but their pooled percentiles should not be substituted for a state-conditional search radius. In particular, a radial percentile computed across altitude and speed settings is not the same as a percentile conditional on the observed altitude and velocity at the failure event. Furthermore, monotonic differences across a selected grid do not independently validate performance outside that grid or under unmodeled wind distributions.

5.5 Verification of the Law of Large Numbers

According to the Law of Large Numbers, the sample mean of independent trials converges to the population mean as the number of trials N increases. To verify this, the cumulative sample mean was observed while incrementally increasing N under the conditions of altitude 57.5 m and cruise speed 8.57 m/s (983 trials) (Table 4).

At $N = 10$, the sample mean is 17.727 m with high variance, but as N increases, it shows a tendency to converge around 17.26 m. For $N \geq 400$, the fluctuation range of the sample mean stabilizes within 0.2 m, confirming the statistical convergence of the Monte Carlo simulation. This provides a descriptive convergence check for the sample mean at the selected state; it does not establish that 400 trials are sufficient to calibrate a 95th percentile uniformly across all states. Tail quantiles, especially when distributions are skewed or multimodal, require separate uncertainty and convergence assessments.

Table 4: Convergence of Sample Mean by Number of Trials N (Altitude 57.5 m, Cruise Speed 8.57 m/s)

N	Sample mean (m)	Sample std. dev. (m)
10	17.727	9.373
30	16.706	6.918
50	16.381	7.141
100	16.893	7.072
200	17.486	7.475
400	17.421	7.374
983	17.264	7.344

6 Embedded System Application

6.1 Offline-to-Embedded Architecture

The intended pipeline has a clear boundary. The PC-side C#/.NET 8 application computes individual trajectories and writes sweep CSV records. At startup, the C99 library reads the CSV via standard file I/O, groups records by their three flight-state coordinates, accumulates statistical moments, and creates a fixed-capacity indexed table. Thereafter the online call performs interpolation only. This division avoids shipping the .NET runtime or BepuPhysics2 to a microcontroller and decouples model-data generation from prediction-time computation. The source repository separates simulator projects into domain, physics, application, I/O, visualization, and application-shell components, with additional source-controlled tests [10]. These software artifacts demonstrate implementation structure; they do not alone establish timing on a physical STM32 board.

6.2 Empirical Regression Model

To utilize the impact point distribution derived from the simulation results in real-time on an embedded system, a 3D Lookup Table (LUT) approach was adopted. The LUT consists of a 3D grid with axes for altitude (h), cruise speed (v_c), and vertical speed (v_v), and each grid point stores distribution statistics such as mean fall distance, standard deviation, and p95 safety radius for that specific flight state.

The LUT-based approach does not require real-time computation of complex numerical integration or probability models, making it suitable for Microcontroller Units (MCUs) with limited computational resources. Values between grid points are estimated using trilinear interpolation, providing continuous predictions even for input values finer than the grid resolution.

Trilinear interpolation sequentially applies linear interpolation in three axial directions and calculates a predicted value as a weighted average of eight adja-

Table 5: Leave-One-Out Interpolation Validation at Altitude 51.11 m ($v_h = 0$, $v_v = 0$)

Field	Interpolated	Actual	Abs. Error
Mean distance (m)	7.77	7.46	0.31
p95 radius (m)	15.98	15.04	0.94
Std. x (m)	6.50	6.36	0.14
Std. y (m)	6.54	6.01	0.52

cent grid points for any given input (h, v_c, v_v). Since the prediction function is activated simply by loading the CSV file once without any additional learning or computation, it can be immediately applied to various aircraft specifications just by replacing sweep.csv.

6.3 Interpolation Mathematics and Boundary Handling

Let $u_a, u_h, u_v \in [0, 1]$ denote the normalized positions of a query state within the enclosing altitude, horizontal-speed, and vertical-speed grid cell. For a stored scalar statistic g, trilinear interpolation gives

$$\hat{g}(\mathbf{s}) = \sum_{i,j,k \in \{0,1\}} w_i(u_a) w_j(u_h) w_k(u_v) g_{ijk}, \qquad (5)$$
$$w_0(u) = 1 - u, \qquad w_1(u) = u.$$

The C implementation first locates the two grid levels bracketing each input, then applies linear interpolation across eight table corners to each output field independently. Inputs outside a grid's range are clamped to the nearest endpoint: the library does not extrapolate and does not return an out-of-range status. Moreover, when an expected corner is missing from the index map, the published source substitutes the first LUT entry. This fallback can silently alter estimates for incomplete grids; a robust deployment should reject incomplete grids or explicitly flag them instead. These are properties of the inspected software, not claims that extrapolated or missing-corner queries have been validated.

6.4 Interpolation Accuracy Validation

To assess the accuracy of the trilinear interpolation scheme independently of the production LUT, a leave-one-out validation was performed on a separately generated three-level altitude grid (20.00 m, 51.11 m, and 82.22 m) at zero horizontal and vertical speed, with 400 Monte Carlo trials per level. This validation grid is distinct from the 62,912-trial production grid described in Section 5 and was generated specifically for this accuracy check. The middle level (51.11 m) was withheld from the LUT, and its distribution statistics were predicted by linearly interpolating between the 20.00 m and 82.22 m levels; the interpolated prediction was then compared against the statistics computed directly from the withheld level's own 400 trials. Table 5 summarizes the result.

The interpolated p95 safety radius deviates from the directly simulated value by 0.94 m (6.3%), and the mean fall distance by 0.31 m (4.2%). The mean offset components (mean_x, mean_y) are close to zero under this symmetric hover condition, so their percentage errors are not a meaningful metric; their absolute errors (0.25 m and 0.83 m, respectively) remain small relative to the resulting p95 radius. These results indicate that linear interpolation between adjacent grid points introduces only a modest error relative to the overall magnitude of the predicted safety radius, supporting the use of a moderately spaced grid for the embedded LUT.

6.5 Limitations of the Interpolation Experiment

The holdout uses three altitude levels at zero initial horizontal and vertical speed. Because two axes remain fixed, the numerical experiment primarily exercises one-dimensional interpolation along altitude. It cannot by itself validate the complete trilinear interpolation behavior at arbitrary nonzero speed combinations. The reported absolute errors are observations for one selected held-out state, not a distribution of errors across the state space. A broader validation would withhold multiple interior cells, compare predictions against independent Monte Carlo replications, and report error quantiles separately for means, component spreads, and radius estimates.

6.6 C Library Design

The embedded C library (`drone_drop.h` / `drone_drop.c`) complies with the C99 standard and has no external dependencies other than linking `-lm`. The internal memory solely uses static allocation and does not use heap allocation at all, allowing it to operate safely in both RTOS environments and bare-metal firmware. The memory footprint for the LUT data is approximately 117 KB BSS.

The public API of the library is as follows.

```
DroneDrop_Status drone_drop_init(
    const char *csv_path);

DroneDrop_Status drone_drop_predict(
    float altitude_m,
    float h_speed_mps,
    float v_speed_mps,
    DroneDrop_Prediction *out);

void drone_drop_free(void);
int  drone_drop_lut_size(void);
```

`drone_drop_init()` is called once during boot to parse the CSV sequentially and build the LUT. Afterwards, passing the altitude, horizontal speed, and vertical speed acquired via GPS/IMU to `drone_drop_predict()` returns the prediction results in the `DroneDrop_Prediction` struct. The main fields of the output struct are shown in Table 6.

Table 6: DroneDrop_Prediction Output Fields

Field	Description
`mean_x_m`	Forward mean offset (m)
`mean_y_m`	Lateral mean offset (m)
`std_x_m`	Forward standard deviation (m)
`std_y_m`	Lateral standard deviation (m)
`mean_distance_m`	Mean fall distance (m)
`p95_distance_m`	95% safety radius (m)

The library output labelled as a 95% radius is implemented by the following moment-based normal approximation, whose probability calibration is discussed in Sec. 6.8.

$$r_{95} = \mu_d + 1.6449 \cdot \sigma_d \tag{6}$$

Here, μ_d is the mean fall distance, and σ_d is the standard deviation of the fall distance. The coefficient 1.6449 is the z-score corresponding to the one-sided 95th percentile of a normal distribution.

6.7 Memory Complexity and Portability

The default maximum grid dimensions in `drone_drop.h` are $20 \times 15 \times 10 = 3000$ combinations. Each `LutEntry` holds nine 32-bit floating-point fields, giving about $3000 \times 36 = 108{,}000$ bytes in the table; an additional array of 3000 integers normally contributes about 12,000 bytes on 32-bit systems, plus small level arrays and bookkeeping. The aggregate static allocation is therefore approximately 120 kB (about 117 KiB), consistent with the repository's approximate memory figure. This is a compile-time maximum allocation, not a measurement of data loaded in the original 64-cell experiment. No dynamic allocation appears in the C library; however, actual integration requires a suitable filesystem/standard-I/O adapter for `fopen`, `fgets`, and `fclose`. The code uses `strtod` during initialization and math-library functions. The portability claim is therefore conditional on available RAM, C library support, filesystem access, and sufficient stack/headroom in the target firmware.

6.8 Radius Estimation and Probability Calibration

The embedded code stores a radial-radius proxy computed from radial moments. Its normal-approximation formula is

$$\tilde{r}_{.95} = \bar{d} + z_{.95}\sqrt{N^{-1}\sum_{i=1}^{N}(d_i - \bar{d})^2}, \qquad z_{.95} \approx 1.6449. \tag{7}$$

In contrast, the simulator can compute the empirical percentile $\widehat{Q}_{.95}(d)$ directly from ordered d_i. Equation (7) gives a true 95th percentile only under an appropriate distributional approximation for nonnegative radial distance. The radial distance of a bivariate position distribution need not be normally distributed: even isotropic Gaussian coordinate offsets yield a Rayleigh-type radial distribution when centered. Therefore $\tilde{r}_{.95}$ must not be equated with a validated 95% coverage probability without calibration. Interpolating several individual $\tilde{r}_{.95}$ values adds another potential source of calibration error. The numerical examples from the original manuscript are retained as reported radius estimates, not certified containment probabilities. A defensible probability claim would compare predicted radii against independently simulated or field-observed exceedance frequencies across a representative range of states.

6.9 MCU Application

Because this library complies with the C99 standard and has no external dependencies, it operates on any MCU that supports a C99 compiler. As long as a means to read files (FatFS, SPIFFS, LittleFS, etc.) is secured, it can be integrated regardless of the platform.

The target hardware assumed in this study is the STM32F405. A u-blox NEO-M9N is used as the GPS receiver, and the 3-axis velocity (`velN`, `velE`, `velD`) is directly acquired through the UBX-NAV-VELNED message. Horizontal speed is calculated as the composite of the North and East velocities, and vertical speed uses the downward component. It is assumed that the LUT data file (`sweep.csv`) is stored on an SPI Micro SD card and accessed via FatFS.

The firmware integration flow is as follows. Upon boot, `drone_drop_init()` is called once to load the LUT. If a crash is detected during flight, the current altitude, horizontal speed, and vertical speed are read from the GPS to call `drone_drop_predict()`. The returned `p95_distance_m` is provided to the operator as the safety search radius, and `mean_x_m` and `mean_y_m` as the expected impact direction offsets.

According to the simulation results of this study, under the conditions of an altitude of 57.5 m and a cruise speed of 8.57 m/s, the p95 safety radius is estimated to be 30.73 m, and this is interpreted as a nominal 95% search-radius estimate under the model approximation, not a verified coverage probability. Utilizing this, the operator can set a 30.73 m radius from the last received GPS point as the priority search area immediately following a crash.

7 Discussion

7.1 Physical Interpretation of the Sweep Results

The originally reported simulation shows increasing horizontal displacement with both initial horizontal speed and altitude. A greater initial horizontal speed supplies more horizontal momentum at the onset of the simulated fall, whereas greater altitude permits longer exposure to wind and drag before ground contact. These explanations are consistent with the model

mechanics. Their relative importance is nevertheless conditional on the selected ranges and wind sampling parameters. The reported ratios are not universal scaling laws and should not be extrapolated to different drone geometries, velocities, or weather regimes.

7.2 Model Fidelity and Structural Uncertainty

The simulator uses a rigid-body integrator, yet the adopted scalar isotropic aerodynamic drag model removes attitude-dependent aerodynamic area and coefficient variations. Starting every sweep trial with identity attitude and zero angular velocity further suppresses possible effects of actual tumbling or asymmetric failures. Consequently, a 6-DOF numerical engine should not be confused with a fully identified six-degree-of-freedom aircraft aerodynamic model. Additional unmodeled mechanisms include partial propulsion loss, actively spinning propellers, structural breakup, impact with obstacles or uneven terrain, and mismatches in the vehicle's mass and aerodynamic parameters. The current results pertain to a total-thrust-loss scenario over a flat surface.

The stochastic disturbance model introduces variability but is not equivalent to experimental identification of the real atmospheric field. The source implements a first-order Gauss–Markov surrogate with prescribed intensities and scales; its numerical update interval also differs from the batch integration step. Sensitivity analyses spanning wind intensity, length scale, time step, aerodynamic parameters, and initial attitude would be needed to determine which assumptions dominate prediction error. Without those analyses, it cannot be asserted that omitted attitude effects change only higher-order distribution moments or that the normal-approximation radius is conservative.

7.3 Statistical and Numerical Uncertainty

Monte Carlo sampling uncertainty must be distinguished from physical-model uncertainty. Increasing the number of samples can improve the precision of statistics within the specified simulator, but it cannot correct biased aerodynamics, an unrealistic turbulence process, or incorrect initial-state sensing. The mean-convergence table establishes only a preliminary stability observation; it is not a confidence interval or a bound on the radial tail. Because the simulation writes individual terminal positions, the empirical conditional quantile and its uncertainty can in principle be studied by independent repetitions or bootstrapping. Those further analyses are not included among the numerical findings reported here.

7.4 Embedded-System Implications

Moving costly simulation offline makes the online numerical workload small and deterministic in shape: the prediction path performs grid bracketing, corner indexing, and interpolation. However, no hardware execution-time benchmark or fully instrumented, flight-tested STM32 deployment is supplied. In addition, the published library assumes an ordered, contiguous group of CSV rows for each flight-state combination, clamps out-of-range inputs, and substitutes a default LUT entry when a corner is absent. These behaviors may be acceptable for controlled demonstrations but should be addressed before treating the library as a dependable safety-critical component. Altitude above ground level, velocity frame conversion, data freshness, crash-detection latency, and communication of the estimate are responsibilities of the surrounding system.

7.5 Suggested Validation Protocol

A rigorous independent validation would use a frozen model and a separate set of flight states spanning the interior and boundaries of the LUT. For each held-out state, Monte Carlo replications should yield an independent empirical 95th percentile and coverage estimates for both the empirical-quantile LUT and the current moment-based approximation. Results should report mean absolute displacement error, vector-centroid error, radius error, undercoverage frequency, and confidence intervals, with particular attention to high winds and states near the grid boundary. A separate numerical sensitivity study should vary the batch time step and turbulence update interval. Physical drop or controlled-failure experiments, when appropriate and conducted under applicable safety procedures, would be required before asserting real-world coverage.

8 Conclusion

This work documents a simulation-to-embedded architecture for generating and querying drone crash-location distributions. The original study reports 62,912 Monte Carlo realizations across 64 altitude–horizontal-speed combinations, a pooled mean horizontal displacement of 20.33 m, and a pooled sample 95th percentile of 49.02 m. The software exports trial data and uses a statically allocated C99 lookup table to return displacement statistics by trilinear interpolation. An additional altitude-only holdout experiment reports a 0.31 m mean-distance interpolation error and a 0.94 m radius-estimate error at its selected validation point.

Inspection of the public implementation clarifies the interpretation and limitations of these findings. The online radius uses a normal-approximation formula rather than the empirical quantile calculated by the simulator, and therefore does not provide a demonstrated 95% containment guarantee. The numerical physics, environmental assumptions, interpolation boundaries, and embedded-platform dependencies likewise limit direct generalization. The present contribution is an implemented and reproducible-in-principle prediction

workflow with reported simulation results; a quantitative reliability claim requires independent distribution-calibration testing, broader numerical validation, and real-world evaluation.

Code Availability

The simulator and embedded C library are publicly available at `https://github.com/lmwmason/DDF`. The implementation descriptions in this revision were checked against the repository's `main` branch as accessed in November 2026. Numeric experiments quoted in the manuscript are those reported in the original manuscript and have not been independently rerun for this expanded version.

Acknowledgment

Claude Code (Anthropic) was used to assist in the implementation of all code in this study.

References

[1] K. Dalamagkidis, K. P. Valavanis, and L. A. Piegl, *On Integrating Unmanned Aircraft Systems into the National Airspace System*. Dordrecht, The Netherlands: Springer, 2008.

[2] A. La Cour-Harbo, "Ground impact probability distribution for small unmanned aircraft in ballistic descent," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, 2020, pp. 1442–1451.

[3] A. La Cour-Harbo, "Quantifying risk of ground impact fatalities for small unmanned aircraft," *J. Intell. Robot. Syst.*, vol. 93, no. 1–2, pp. 367–384, 2019.

[4] X. Liu, X. Li, and J. Zhang, "Ground risk assessment of UAV operations based on horizontal distance estimation under uncertain conditions," *Math. Probl. Eng.*, vol. 2021, Art. no. 3384870, 2021.

[5] S. Primatesta, A. Rizzo, and A. La Cour-Harbo, "Ground risk map for unmanned aircraft in urban environments," *J. Intell. Robot. Syst.*, vol. 97, pp. 489–509, 2020.

[6] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton, NJ, USA: Princeton Univ. Press, 2012.

[7] *Flying Qualities of Piloted Aircraft*, U.S. Dept. Defense, MIL-HDBK-1797, 1997.

[8] T. M. I. Hakim and O. Arifianto, "Implementation of Dryden continuous turbulence model into Simulink for LSA-02 flight test simulation," *J. Phys.: Conf. Ser.*, vol. 1005, Art. no. 012017, 2018.

[9] E. Rudnick-Cohen, J. W. Herrmann, and S. Azarm, "Risk-based path planning optimization methods for unmanned aerial vehicles over inhabited areas," *J. Comput. Inf. Sci. Eng.*, vol. 16, no. 2, 2016.

[10] M.-W. Lee, "Drone Drop Finder: Simulator and embedded C library," source repository, 2026. [Online]. Available: `https://github.com/lmwmason/DDF`

[11] B. L. Stevens, F. L. Lewis, and E. N. Johnson, *Aircraft Control and Simulation*, 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, 2015.