An Environmentally Complex Requirement for Safe Separation Distance between UAVs

Pedro Antonio Alarcon Granadeno, Arturo Miguel Russell Bernal, Md Nafee Al Islam, Jane Cleland-Huang
Computer Science and Engineering
University of Notre Dame
South Bend, USA
{palarcon, arussel8, mislam2, janehuang}@nd.edu

***Abstract*—Cyber-Physical Systems (CPS) interact closely with their surroundings. They are directly impacted by their physical and operational environment, adjacent systems, user interactions, regulatory codes, and the underlying development process. Both the requirements and design are highly dependent upon assumptions made about the surrounding world, and therefore environmental assumptions must be carefully documented, and their correctness validated as part of the iterative requirements and design process. Prior work exploring environmental assumptions has focused on projects adopting formal methods or building safety assurance cases. However, we emphasize the important role of environmental assumptions in a less formal software development process, characterized by natural language requirements, iterative design, and robust testing, where formal methods are either absent or used for only parts of the specification. In this paper, we present a preliminary case study for dynamically computing the safe minimum separation distance between two small Uncrewed Aerial Systems based on drone characteristics and environmental conditions. In contrast to prior community case studies, such as the mine pump problem, patient monitoring system, and train control system, we provide several concrete examples of environmental assumptions, and then show how they are iteratively validated at various stages of the requirements and design process, using a combination of simulations, field-collected data, and runtime monitoring.**

***Index Terms*—requirements, environmental assumptions, small unmanned aerial systems, UAV, case-study**

I. Introduction

Cyber-Physical Systems (CPS) are strongly dependent upon their surrounding environments, including their concurrently executing software components, regulatory dependencies, user interactions, and physical devices, such as sensors [1], [2], [3]. It is therefore important for developers to clearly document their *assumptions* about the operating environment [4]. Failure to do so can lead to unwanted behaviors, with numerous prior incident reports pointing to the role of missing or inadequate assumptions as key contributing factors [2]. For example, the U.S. National Research Council cited an example of the 1993 Airbus incident in which the plane failed to brake upon landing, at least partially due to the invalid assumption that "lack of compression always accompanies being airborne" [3]. In this case, the plane hydroplaned on the icy runway, compression never occurred, and as a result the system mistakenly believed the plane to be airborne, and failed to apply the brakes.

Environmental assumptions are generally classified into six groups [5]. These include (1) *physical environment* assumptions, which are expected to hold invariantly (e.g., "If the drone is 'flying' it is not 'on the ground'.); (2) *operational environment* assumptions that describe the operational environment surrounding the system, (e.g., "There is no interference from other wireless devices in the area."); (3) *adjacent system* assumptions describing behavior of interacting adjacent systems, (e.g., "The aircraft is controlled with the mRo Control zero autopilot at a rate of 480 Hz.'); (4) *user interface* assumptions describing users and their behavior (e.g., "The operator will place the RC transmitter's throttle in the neutral position prior to take-off"; (5) *regulatory* assumptions describing the way regulations, laws, or standards affect the system or related components (e.g., "All flights must be operated under FAA Part 107 regulations.") and finally (6) *development process* assumptions describing policies or procedures impacting the development and/or operation of the system, (e.g., "All features are validated in simulation prior to queuing them for field tests").

In some scenarios, a single requirement may depend on multiple environmental assumptions, creating a complex space for specifying, analyzing, and validating both the requirements and their underlying assumptions [6]. We therefore introduce the term 'environmentally complex requirement' to describe a requirement that relies on multiple, inter-dependent environmental assumptions. Unlike basic safety requirements, which might depend on relatively stable and predictable environmental conditions, environmentally complex requirements involve a broader range of variables that interact in complex ways. These requirements must be analyzed and validated in ways that accommodate increased complexity and inter-dependencies, so that they can operate reliably under a variety of unpredictable environmental conditions.

In constructing the case study, we followed a robust engineering approach, supported by our domain experience, to systematically reason about factors that impact separation distance between small Uncrewed Arial Systems (sUAS), therefore deriving the requirements and assumptions laid out in this paper. As depicted in Figure 1, starting

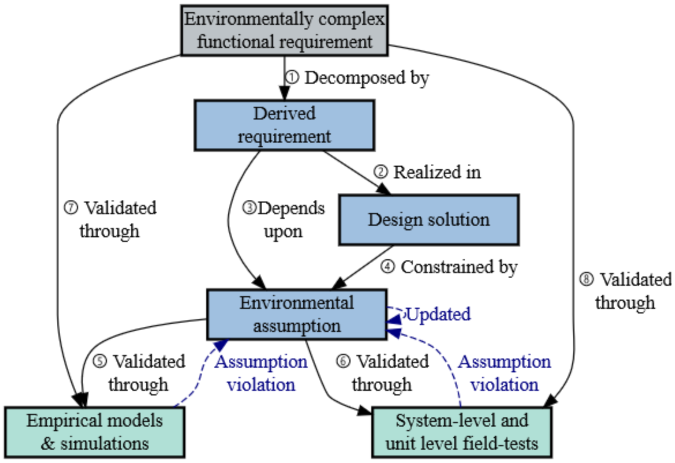


Fig. 1. Steps for deconstructing an environmentally complex requirement into derived requirements and their associated environmental assumptions, and subsequently for validating assumptions and requirements.

with an environmentally complex requirement we followed standard requirements engineering practices to refine it into derived requirements (Task 1), explore design solutions (Task 2), and simultaneously identify and document environmental assumptions (Task 3) and dependencies of the design on those assumptions (Task 4). As these dependencies were recognized, we investigated the assumptions through a combination of techniques that included use of empirical models, simulations, and unit-level field tests (Tasks 5-6), with the goal of reducing uncertainty in order to design an effective solution that encompassed appropriate tolerance levels. Finally we validated the system level requirements through simulations (Task 7) and monitored field deployments (Task 8) to build confidence that despite any remaining uncertainties, our solution would maintain a safe separation distance between sUAS at all times and that no assumptions were violated.

The remainder of the paper is structured as follows. Section II describes the multi-vehicle sUAS platform that provides the context for our case study. It also describes a single environmentally complex requirement and its derivatives which serve as the focus of our paper. Section III describes associated environmental assumptions and their preliminary validation, while Section IV takes a more holistic view to validating the high-level requirement within the context of the broader set of assumptions. Sections V and VI then discuss threats to validity and related research, while Section VII summarizes the paper's contributions and discusses future work.

II. Case Example: Maintaining Minimum Separation Distance between sUAS

Our case example focuses upon computing the safe minimum separation distance between sUAS at runtime, considering individual drone characteristics and current environmental conditions.

A. DroneResponse Platform

The context of our example, is our own real-world DroneResponse system [7], [8] which provides a robust, multi-vehicle, platform for sUAS that operate PX4 autopilot firmware [9] using the MAVROS communication protocol [10]. The DroneResponse platform, shown in action in Figure 2, includes a ground control station (GCS), hosting microservices and Graphical User Interfaces (GUIs), while each sUAS has its own onboard compute capabilities hosted on a Jetson Xavier NX running Drone Response's proprietary control software called 'DR-Onboard'. All communication between drones and GCS components is via MQTT over mesh-radio.

In this multi-vehicle application, it is critical that sUAS flying in a shared airspace do not collide with one another. There are many different approaches for maintaining minimum separation. Our current approach, delegates this responsibility to a centralized air-leasing system, hosted as a Microservice on the GCS. The air-leaser authorizes requests for the use of the airspace. It maintains a global minimum separation distance between all pairs of sUAS and is designed to accommodate worst-case aerodynamics (e.g., increasing stopping distance), extreme wind conditions, slow reflexes, and worst-case geolocation, based on GPS and other sensor data. However, this approach means that drones are separated by large distances and often suffer from lengthy delays as they wait for their desired airspace to open up.

B. An Environmentally Complex Requirement & Derivatives

For convenience we refer to each sUAS as a drone throughout the remainder of the paper, and start by specifying a single system-level, functional requirement.

> *Functional Requirement (R1):* When in flight, drone d shall continually compute the **minimum_separation_distance** with each neighboring drone d' within its Region of Interest (ROI).

The minimum separation distance required between two drones must account for intrinsic and extrinsic uncertain-

Fig. 2. Using the DroneResponse platform, multiple sUAS fly autonomously in close vicinity to one another, requiring safe minimum separation distances to be maintained at all times.

ties related to each drone's navigation and positioning. We have identified four factors as primary determinants for establishing the size of an operational buffer. These include (a) geolocation uncertainty, (b) stopping distance, (c) wind conditions, and (d) the projected distance that a drone might travel between status messages. This leads to the following definition:

$$d_{\text{buffer}} = d_{\text{geo}} + d_{\text{stop}} + d_{\text{wind}} + d_{\text{proj}} \tag{1}$$

Here, d_{geo} encompasses the margins required to address geolocation uncertainties, d_{stop} represents the distance necessary for drone d to come to a complete stop in non-windy conditions, d_{wind} reflects the additional distance needed to compensate for worst-case wind-induced deviations, and d_{proj} represents the distance that a drone is projected to travel during interval P.

Throughout the remainder of this paper we also refer to drone d', which we treat as a black box representing either a drone using the DroneResponse platform or an external drone. In either case, d' is expected to comply with global airspace regulations concerning behaviors such as broadcasting current status and subsequent mitigation efforts in event of airspace breach. The minimum safety distance required for safe operations between drones d and d', as perceived by drone d, is thus the sum of their individual operational buffers, adjusted by a comfort tolerance factor. This is expressed as:

$$d_{\text{safe}} = d_{\text{buffer}}(d, P) + d_{\text{buffer}}(d', L) + \text{tolerance} \tag{2}$$

where $d_{\text{buffer}}(d, P)$ is the operation buffer for drone d, factoring in the standard broadcast period P, and $d_{\text{buffer}}(d', L)$ is the operational buffer for drone d' computed by d, given the data broadcast by d' and adjusted for $L = P + T_{\text{com}}$. Here, L denotes the total effective interval that accounts for both the broadcast period and communication latency T_{com}. This adjustment to the equation acknowledges that the broadcasted information from drone d' is rendered stale by an additional time factor of T_{com}. Moreover, in this context, drones d and d' symbolize the sets of parameters—such as velocity and position—outlined in our broadcasting requirement.

Given these factors we decompose our top level requirement (R1) into a set of derived requirements (DR1-DR5). The first requirement addresses the geolocation of the current drone d and the geolocation of d' from the perspective of d.

> *(DR1):* For each drone d, the autopilot shall periodically ascertain its `geolocation` and the `geolocation` of each neighboring drone d', ensuring at least a 99% confidence level that each drone is positioned within a specified 3D region around its computed coordinates.

We next specify a requirement associated with the drone's required stopping distance in non-windy conditions as:

> *(DR2):* For each drone d, the autopilot shall periodically compute its `maximum stopping distance` and the `maximum stopping distance` of each neighboring drone d', in non-windy conditions, given their current velocity.

For computing additional stopping distances due to wind conditions we consider the tradeoffs associated with the challenge of measuring wind velocity and direction during flight at various altitudes and locations, the volatility of the wind gusts, complexity of computing relative wind direction versus the drone's direction of travel, and limited onboard computational resources. Therefore, we assume worst case scenario of the projected maximum wind-gusts and a tailwind leading to the following:

> *(DR3):* For each drone d, and each neighboring drone d' of d, the autopilot shall consider the drone's additional stopping distance introduced by the maximum projected tailwind.

Additionally, we account for any change in geolocation during the status update interval, considering communication latency from a neighboring drone's status updates as depicted in Eq. 2.

> *(DR4):* For each drone d, the autopilot shall compute its `distance projected to travel`, and the `distance projected to travel` of each neighboring drone d', between status updates, given their current velocity.

Finally, as discussed in Section IV, an absolute minimum separation distance, which we refer to as the *comfort distance* between each drone must be assured even in worst case scenarios.

> *(DR5):* For each drone d, and each neighboring drone d', there shall exist a minimum separation distance that must never be violated.

III. Investigating Environmental Assumptions

Given the dependencies that requirements exhibit upon environmental assumptions [11], [12] , our next step involves identifying and documenting a set of relevant assumptions labeled as A1 - AN. Further, because incorrect assumptions can lead to unsafe solutions; we evaluate the evidence supporting each assumption and its associated degree of certainty or uncertainty. For assumptions with high degrees of uncertainty, we aim to systematically reduce the uncertainty using techniques such as examining manufacture's claims and their supporting documentation, creating empirical models, running simulations, and/or collecting and analyzing field data.

A. System-wide Operational Regulations

We start by considering regulations that impact the overall operation of the sUAS platform, as these may have a cross-cutting impact upon multiple requirements. The first two operational regulations serve a dual role, first as requirements for drones operating on the DroneResponse platform and therefore under our own control, and second as regulatory assumptions for all other drones operating in the airspace. These regulations specify the timing for broadcasting messages between drones and for computing minimum separation distances. By consensus, any drone entering the shared airspace is required to comply with these regulations.

REG1 / A1. Regulatory: Every drone d operating in the shared airspace broadcasts a status message at fixed intervals, denoted as period P. The status message includes the drone's current position, velocity, heading, acceleration, and accuracy in its geolocation.

REG2 / A2. Regulatory: Each drone d computes its required safety distance from drone d' upon receipt of the status message sent from d'.

An additional environmental assumption impacts the freshness of the status message at the time it is broadcasted by drone d. We document it as follows:

A3. Adjacent System: The onboard software, which broadcasts status messages, receives updates from its associated flight controller sufficiently frequently that the delay between obtaining a status update and broadcasting to other drones is negligible.

Finally, we document one further assumption related to the health of the drone between consecutive broadcasts. This is documented as follows:

A4. Adjacent System: Drones maintain operational safety between consecutive broadcasts and thus all minimum safety calculations are valid until the following status update.

While it is inevitable that drones will occasionally experience failures, these failures are handled by raising an alert that triggers a failsafe mechanism. This scenario is outside the scope of the current paper; however, we document an assumption based on the underlying premise that the broadcasting period is sufficiently small, meaning that failures can be ignored until such time as an alert is raised and the drone transitions to a failsafe mode.

These assumptions describe communication protocols for the shared airspace and set the stage for each drone to quantify a safety buffer between itself and drone d' at interval P triggered by the receipt of a status update from d'. We now examine four of our five derived requirements (DR1-DR4), each of which is associated with a unique factor described in Eq. 1 for computing safe distance. As discussed in Section IV, we refer to derived requirement DR5 as an *emergent requirement*, which is expected to be satisfied as a result of meeting requirements DR1-DR4 and adding a sufficient tolerance level as depicted in Eq. 2.

B. Geolocation Uncertainty (DR1)

We start by considering the factors associated with geolocation uncertainty. Inaccuracies in a drone's geolocation arise due to errors in satellite signal delays, atmospheric conditions, multipath effects, and signal interference, alongside challenges posed by dynamic environmental factors and hardware limitations. Therefore, in order to reflect a drone's position inaccuracies, we represent the drone's location as being within a 3D bounded region of space rather than at a pinpoint location. This region provides a three-dimensional 'container' that reflects the uncertainty of the drone's position. To systematically analyze geolocation uncertainty, we start with the following environmental assumption, supported by the MAVLink documentation for interfacing with PX4 [13].

Most sUAS, including our PX4-based flight controllers utilize Extended Kalman Filters (EKF) to aggregate data into a fused global data structure for both horizontal and vertical positioning. They also report $pos\_horiz\_accuracy$ defined as 'Horizontal position 1-STD accuracy relative to the EKF local origin' [14], which indicates the estimated accuracy of the drone's horizontal position as calculated by the EKF, with a 68% confidence level that the drone's true horizontal position is within this reported range from the EKF's local origin point. By multiplying the pos_horiz_accuracy by the Z-score corresponding to 99% confidence in a normal distribution (≈ 2.576), we effectively scale the horizontal error margin. This adjusts the base of a cylinder on the horizontal plane, within which we can be 99% confident of the drone's location. Similarly, vertical accuracy can be scaled to define the vertical height of the cylinder, providing a three-dimensional spatial region for the drone's probable location. Thus to account for geolocation uncertainty in our minimum safety distance calculation, we define the following equation:

$$d_{\text{geo}} = 2.576 \times \sqrt{\sigma_{\text{horiz}}^2 + \sigma_{\text{vert}}^2} \tag{3}$$

where σ_{horiz} and σ_{vert} are the horizontal and vertical position 1-STD accuracy relative to the EKF origin, respectively. Three related environmental assumptions, supported by MAVLink [15] and PX4 [9] documentation are as follows:

A5. Adjacent System: The EKF reports the estimated latitude and longitude of the drone, along with a distance representing a 68% probability that the drone's actual location is within this distance from the estimated coordinates.

A6. Adjacent System: The EKF reports the estimated altitude of the drone, along with a height representing a 68% probability that the drone's actual altitude is within this distance from the estimated altitude.

A7. Adjacent System: Errors in the horizontal position (latitude & longitude) and vertical position (altitude) as computed by the flight-controller, follow a normal distribution where Z-score corresponds to 99% = 2.576

We initially trust these assumptions, but recognize that they need to be validated in the field through designing and executing a series of tests with specific physical drones.

C. Stopping Distance without wind (DR2)

We define the stopping distance of a drone as the total distance the drone travels from the moment it decides to stop, until the point where it comes to a complete halt. Excluding environmental perturbations like wind, this distance consists of two main constituents: the reaction distance d_{reaction} and the braking distance d_{braking}. We therefore consider each of these in turn.

- *Reaction Distance:* The reaction distance, given by d_{reaction}, represents the distance the drone covers during reaction time. As previously stated in Regulation REG1 (and Assumption A1), each drone reports its status at intervals marked by $P = t_1 - t_0$. During these time intervals, the drone is unlikely to be stationary, and therefore we need to consider the upper-bound on its potential movement. Specifically, we consider the velocity increase over period P as $v_1 = v_0 + a_{\max} \times P$, where $a_{\max}$ represents the drone's maximum acceleration capacity and v_0 is the velocity at the beginning of the interval. Therefore, the reaction distance is defined as $d_{\text{reaction}} = v_1 \times T_{\text{reaction}}$ where T_{reaction} is the interval between receiving a stop signal and initiating deceleration. Under upper-bound considerations, we assume T_{reaction} immediately trails P, hence $T_{\text{reaction}} = t_2 - t_1$. Further, we specify one additional assumption:

A8. Operational Environment: All drones flying in the shared airspace operate autonomously.

Given this assumption we do not need to include operator response time, and therefore T_{reaction} is solely determined by system latency.

In Eq. 2, we showed that the safe separation distance included two buffer calculations. In the case of drone d the period over which the buffer is computed is p, whereas, for drone d', as discussed in Section II-B, the period is effectively L to account for staleness in the reported parameters. Communication latency, T_{com} can often be determined from the specifications provided in the documentation of communication devices. In our case study, we utilize Doodle Labs Smart Radio – RM-2450, which uses an ultra-reliable Low Latency Channel. Messages are currently transmitted between drones via the GCS, thus, we document the following assumptions related to latency:

A9. Adjacent System: Status messages are transmitted over Doodle Labs Smart Radio – RM-2450 using an ultra-reliable Low Latency Channel with message latency of 3-30 ms. Messages take two hops creating a total transmission latency of 6-60 ms.

A10. Adjacent System: Environmental factors such as signal interference or communication delays beyond the established latency T_{com} are negligible or within controllable limits.

- *Braking Distance:* At t_2, the drone initiates deceleration. In this phase, we must consider the drone's velocity at the onset of braking, $v_2 = v_1 + a_{\max} \times T_{\text{reaction}}$, which incorporates the maximal potential increase in speed during the reaction interval. Hence, the braking distance, defined as the distance traversed from deceleration to a complete halt, is expressed as $d_{\text{braking}} = \frac{v_2^2}{2a_{\text{dec}}}$, where a_{dec} represents the drone's deceleration capacity. Therefore, the total stopping distance can be modeled by the equation:

$$d_{\text{stop}} = (v_1 \times T_{\text{reaction}}) + \left(\frac{v_2^2}{2a_{\text{dec}}}\right) \tag{4}$$

In these calculations we have treated system latency and the deceleration rate as fixed constants. While reaction time might vary according to the current processing load of the sUAS, our system prioritizes critical processes (e.g. stop commands) over other secondary processes (i.e. computer vision, analytics), thereby reducing variation in reaction times for prioritized processes such as emergency braking.

Finally, as the drone's maximum acceleration capacity is bounded by the drone's maximum velocity capacity, the practical formulation of $a_{\max}$ is accounted in the equation:

$$a_{\max} = min\left(a_{\max}, \left(\frac{v_{\max} - v}{T}\right)\right) \tag{5}$$

where $v_{\max}$ corresponds to the drone's maximum velocity capacity, v corresponds to the drone's velocity of reference and T to the time interval in question.

D. The Impact of Wind (DR3)

Computing the effect of wind speed and direction, especially with transient wind gusts, versus the drone's heading and velocity is highly complex due to the unpredictable, often chaotic nature of wind behavior, countered by the drone's PID (Proportional-Integral-Derivative) control system's response to these external forces.

We therefore start by making two simplifying assumptions. First, we assume that drones are operated within manufacturers specifications, and therefore are only flown in weather conditions they can handle. This means that we can ignore extreme weather conditions that are beyond the capabilities of the PID.

A11. Process: Operators do not launch the drone when wind conditions are outside operational specifications or forecasted to be outside operational specifications within the planned flight window.

Further, due to the processing resources that would be required to continually compute the current direction of the wind versus the drone's heading, we make the following assumptions that allow us to focus on the worst-case scenario of tailwinds only.

A12. Physical Environment: When wind conditions are within operational specifications, regardless of wind direction, sideways drift of the drone is minimal and remains within an acceptable drift_margin.

A13. Physical Environment: Tailwinds extend drone stopping distances more than other relative wind directions.

Given that the PID system's capacity to mitigate wind conditions hinges on its precise tuning, which can vary across different drones and operational conditions, we adopted an empirical approach for predicting the impact of wind conditions. We started by utilizing the Gazebo simulation environment [16], [17] to methodically assess trajectory deviations under a range of wind conditions. However, any trust we place in Gazebo for generating correct trajectory deviations caused by wind, implies the following assumption:

A14. Process: The Gazebo simulation environment closely approximates the aerodynamic and physical interactions of a *typical drone* under varied wind conditions, reflecting real-world behavior with an acceptable degree of fidelity.

In an effort to validate this assumption, we inspected the source code and associated documentation of the underlying Gazebo physics model, which indicates that wind is modeled using a nonlinear quadratic approximation for thrust calculations under various wind conditions. The documentation further claims that wind behavior has been validated by the developers in a physical wind tunnel [16]; however, this assumption has relatively high degrees of uncertainty associated with it, and is highly dependent upon individual drone characteristics. All three of these assumptions are supported by limited data that we previously collected from a series of physical flight tests [18]; however, additional data is needed for specific drone models. Based on these findings, we focus our analysis on tailwinds, which represent the worst case scenario with respect to stopping distance.

For purposes of this paper, we report results obtained from our experimentation in the Gazebo simulation in which we investigated the stopping distance caused by a constant tailwind aligned with the intended flight path on a representative drone using the Gazebo simulation environment. We conducted an experiment in which drones executed a predetermined straight-line flight between two waypoints, where total distances were measured. We varied wind speeds while keeping a control case (zero wind) for baseline comparisons. The deviation in stopping distance caused by the wind was quantified by comparing the drone's travel distance in wind-affected trials versus their non-wind controlled distance as reported in Figure 3.

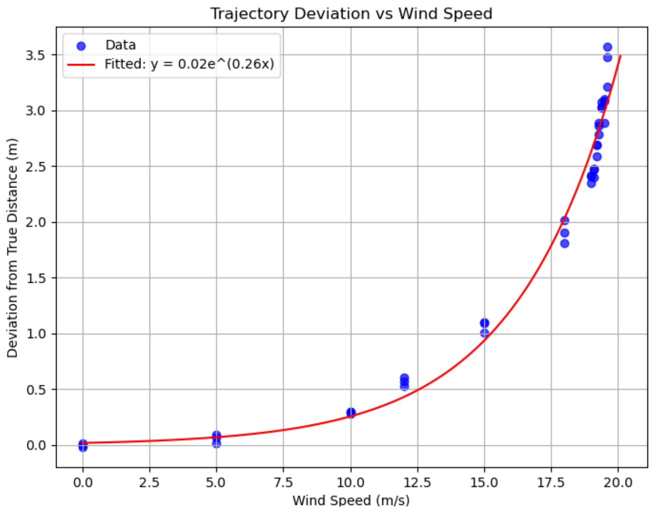


Fig. 3. Nonlinear regression analyses on data collected in Gazebo.

We performed nonlinear regression analysis to model the relationship between wind speed and the deviation of the drone's trajectory from its intended path. An exponential model of the form $y = ae^{bx}$ was fitted to the data using the least squares method, implemented via the curve_fit function from the SciPy library in Python. This model was chosen based on the observed exponential increase in trajectory deviation with wind speed. The results confirm the intrinsic exponential relationship claimed by the Gazebo documentation, and thus we define d_{wind} as:

$$d_{\text{wind}} = 0.02e^{0.26v_{\text{wind}}} \tag{6}$$

where v_{wind} is the speed of a tailwind upon contact with the drone.

Given the inherent variability of wind patterns encountered during flight, we employed a Monte Carlo [19] simulation approach to evaluate the uncertainties affecting tailwind disturbances under a wide range of conditions to provide a robust framework for understanding the probabilistic impacts of wind on drone trajectories. To model the probability distribution of wind speeds during a particular flight session, we employed the Weibull distribution. The use of the Weibull distribution is supported by its extensive validation for accurately modeling wind speed variations across various environments [20], [21].

Given a mean wind speed, μ_{wind} and a predefined shape parameter, the scale parameter λ can be estimated using $\lambda = \frac{\mu_{\text{wind}}}{\Gamma(1+\frac{1}{k})}$. This allows us to construct a tailored wind profile for each flight session. Furthermore, since the Weibull distribution generally denotes speeds at 10 meters above ground level, adjustments for drone altitudes are made using the wind power law, expressed as:

$$v(z) = v_{\text{ref}} \left(\frac{z}{z_{\text{ref}}} \right)^{\alpha} \tag{7}$$

Here, $v(z)$ signifies wind speed at altitude z, v_{ref} denotes reference wind speed at reference height z_{ref}, and α represents the wind shear exponent, which depends on surface roughness and stability of the atmosphere. This law enables refined altitude-specific wind speed estimations that can be tailored to each drone's operational necessities.

By integrating these elements, along with our empirical Equation 6, we can calculate trajectory deviations for randomly generated wind profiles within the Monte Carlo simulation. For conservative safety buffer estimations, we consider the disturbance corresponding to wind speeds situated two standard deviations from the mean.

E. Projected Distance during time interval 'P' (DR4)

Projected distance, d_{proj}, forecasts the drone's displacement based on current operational conditions. For a drone with current speed v and subjected to a maximum acceleration $a_{\max}$, the projected distance over a time interval P can be articulated through the kinematic equation:

$$d_{\text{proj}} = v \times P + \frac{1}{2} a_{\max} \times P^2 \tag{8}$$

In operational terms, the projected distance incorporates the drone's current motion state and its capability to increase speed within the specific time frame. We adopt a conservative operative approach by factoring in $a_{\max}$ as the upper limit of potential acceleration when computing the projected distance. As discussed in Section II-B, for drone d', the period P is effectively L to account for communication latency.

F. Tolerances

Eq. 2 includes a tolerance variable, which is designed to accommodate additional factors, uncertainties, and d_{comfort}, representing a distance between drones that must be maintained at all times to prevent scenarios in which drones pass within inches of each other. As an example, the tolerance currently includes the physical dimensions of drones d and d' for which we specify one additional assumption.

> *A15. Operational Environment*: Every drone knows its physical dimensions, broadcasts this information upon startup, and notifies new drones upon their entry to the airspace.

IV. Validating Environmentally Complex Requirements

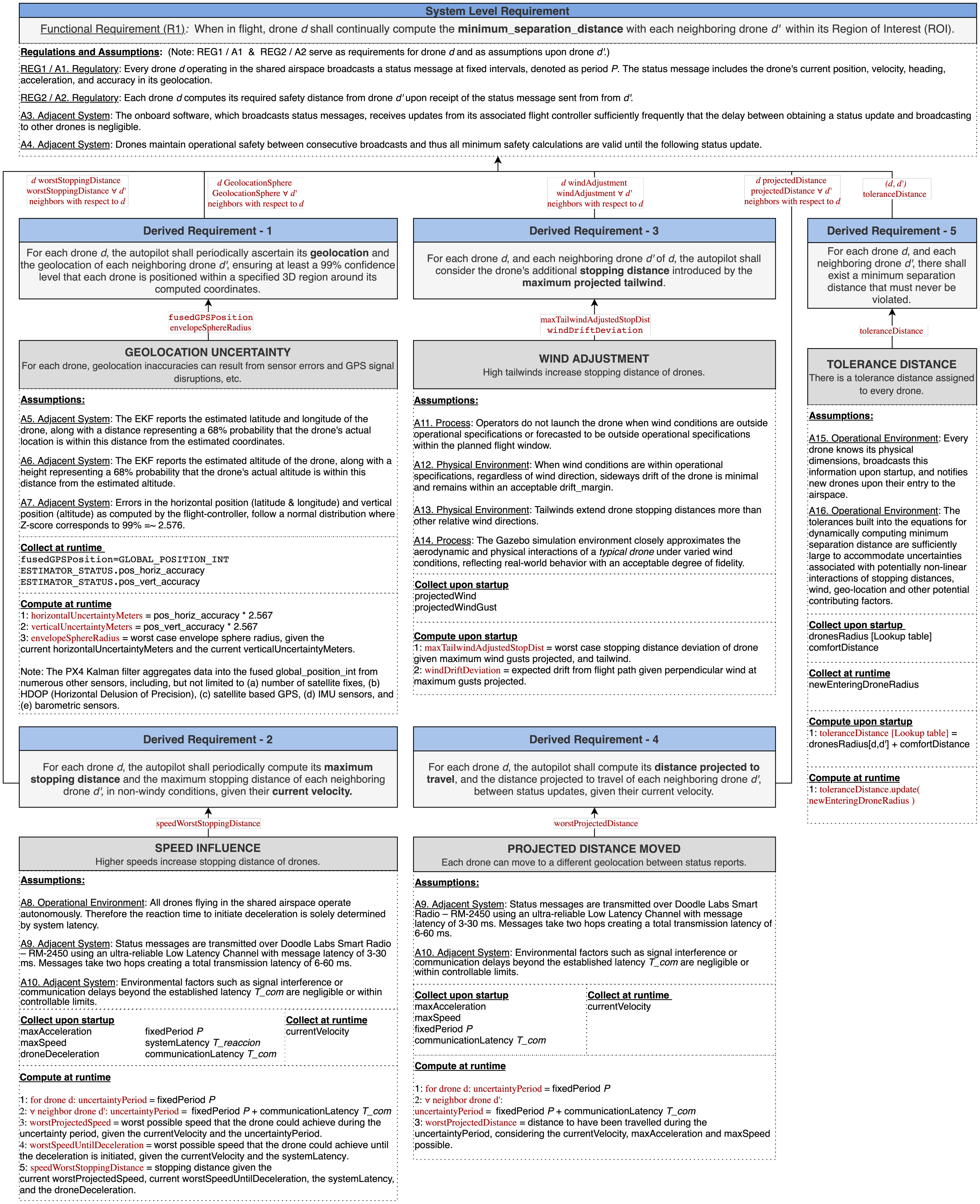
So far we have documented 15 environmental assumptions and validated them to various degrees using a combination of techniques. However, to simplify our reasoning about this complex space, we have assumed a linear dependency between wind speed, drone trajectory deviation, and geolocation accuracy, as depicted in Eq. 1, even though the interactions between the factors may lead to complex, nonlinear outcomes, and therefore may not be fully predictable using linear models or under assumptions of component independence. However, we deliberately built tolerances into Eq. 2 to accommodate unaccounted interactions, and additional factors that we were either unaware of, chose not to explicitly model, or for which unknown degrees of uncertainty could impact our assumptions. This leads to the following assumption:

> *A16. Operational Environment*: The tolerances built into the equations for dynamically computing minimum separation distance are sufficiently large to accommodate uncertainties associated with potentially non-linear interactions of stopping distances, wind, geolocation and other potential contributing factors.

It is particularly important to validate this assumption in order to provide assurance that an absolute safe separation distance is maintained between drones at all times, even if all other factors combine into a worst-case scenario. For example, consider some value for d_{safe} representing the safe separation distance that drone d needs to maintain from other drones under current operating conditions, and then assume that the drone needs to perform an emergency stop when the timing of status messages, wind conditions, and geolocation accuracy join together to create a worst case scenario. Further, following the emergency stop, there is a d_{comfort} distance that must still be maintained, for example, to ensure that drones pass each other with only inches to spare. This d_{comfort} is already accounted for in the tolerance buffers of in Eq. 2. It addresses derived requirement DR5 which established the need for an absolute minimum separation distance to never be violated.

To validate that drones consistently maintain the necessary d_{comfort} distance under all operational conditions, including worst-case scenarios, we have planned, but not yet executed a Monte Carlo, using the Gazebo simulation framework. These simulations will introduce variables such as GPS variance, diverse drone velocities and maneuvers, and varying wind profiles. By systematically testing against a broad spectrum of conditions, we aim to ensure that drones adhere to the predefined d_{comfort} separation. If simulations confirm that drones uphold the d_{comfort} distance, this would affirm that the tolerances integrated into Equation 2 are sufficiently robust to encapsulate potential non-linearities and any overlooked factors present in the operational model.

Furthermore, given potential uncertainties in the fidelity of the simulation models versus the physical world, we also need to collect data from diverse physical world flights to build confidence in our results. We can collect this data using runtime monitoring [22] or through post-mortem



System Level Requirement

Functional Requirement (R1): When in flight, drone *d* shall continually compute the **minimum_separation_distance** with each neighboring drone *d′* within its Region of Interest (ROI).

Regulations and Assumptions: (Note: REG1 / A1 & REG2 / A2 serve as requirements for drone *d* and as assumptions upon drone *d′*.)

REG1 / A1. Regulatory: Every drone *d* operating in the shared airspace broadcasts a status message at fixed intervals, denoted as period *P*. The status message includes the drone's current position, velocity, heading, acceleration, and accuracy in its geolocation.

REG2 / A2. Regulatory: Each drone *d* computes its required safety distance from drone *d′* upon receipt of the status message sent from from *d′*.

A3. Adjacent System: The onboard software, which broadcasts status messages, receives updates from its associated flight controller sufficiently frequently that the delay between obtaining a status update and broadcasting to other drones is negligible.

A4. Adjacent System: Drones maintain operational safety between consecutive broadcasts and thus all minimum safety calculations are valid until the following status update.

Derived Requirement - 1

For each drone *d*, the autopilot shall periodically ascertain its **geolocation** and the geolocation of each neighboring drone *d′*, ensuring at least a 99% confidence level that each drone is positioned within a specified 3D region around its computed coordinates.

GEOLOCATION UNCERTAINTY

For each drone, geolocation inaccuracies can result from sensor errors and GPS signal disruptions, etc.

Assumptions:

A5. Adjacent System: The EKF reports the estimated latitude and longitude of the drone, along with a distance representing a 68% probability that the drone's actual location is within this distance from the estimated coordinates.

A6. Adjacent System: The EKF reports the estimated altitude of the drone, along with a height representing a 68% probability that the drone's actual altitude is within this distance from the estimated altitude.

A7. Adjacent System: Errors in the horizontal position (latitude & longitude) and vertical position (altitude) as computed by the flight-controller, follow a normal distribution where Z-score corresponds to 99% =~ 2.576.

Collect at runtime

`fusedGPSPosition=GLOBAL_POSITION_INT`
`ESTIMATOR_STATUS`.pos_horiz_accuracy
`ESTIMATOR_STATUS`.pos_vert_accuracy

Compute at runtime

1: horizontalUncertaintyMeters = pos_horiz_accuracy * 2.567
2: verticalUncertaintyMeters = pos_vert_accuracy * 2.567
3: envelopeSphereRadius = worst case envelope sphere radius, given the current horizontalUncertaintyMeters and the current verticalUncertaintyMeters.

Note: The PX4 Kalman filter aggregates data into the fused global_position_int from numerous other sensors, including, but not limited to (a) number of satellite fixes, (b) HDOP (Horizontal Delusion of Precision), (c) satellite based GPS, (d) IMU sensors, and (e) barometric sensors.

Derived Requirement - 2

For each drone *d*, the autopilot shall periodically compute its **maximum stopping distance** and the maximum stopping distance of each neighboring drone *d′*, in non-windy conditions, given their **current velocity.**

SPEED INFLUENCE

Higher speeds increase stopping distance of drones.

Assumptions:

A8. Operational Environment: All drones flying in the shared airspace operate autonomously. Therefore the reaction time to initiate deceleration is solely determined by system latency.

A9. Adjacent System: Status messages are transmitted over Doodle Labs Smart Radio – RM-2450 using an ultra-reliable Low Latency Channel with message latency of 3-30 ms. Messages take two hops creating a total transmission latency of 6-60 ms.

A10. Adjacent System: Environmental factors such as signal interference or communication delays beyond the established latency *T_com* are negligible or within controllable limits.

Collect upon startup

maxAcceleration
maxSpeed
droneDeceleration
fixedPeriod *P*
systemLatency *T_reaccion*
communicationLatency *T_com*

Collect at runtime

currentVelocity

Compute at runtime

1: for drone d: uncertaintyPeriod = fixedPeriod *P*
2: ∀ neighbor drone d': uncertaintyPeriod = fixedPeriod *P* + communicationLatency *T_com*
3: worstProjectedSpeed = worst possible speed that the drone could achieve during the uncertainty period, given the currentVelocity and the uncertaintyPeriod.
4: worstSpeedUntilDeceleration = worst possible speed that the drone could achieve until the deceleration is initiated, given the currentVelocity and the systemLatency.
5: speedWorstStoppingDistance = stopping distance given the current worstProjectedSpeed, current worstSpeedUntilDeceleration, the systemLatency, and the droneDeceleration.

Derived Requirement - 3

For each drone *d*, and each neighboring drone *d′* of *d*, the autopilot shall consider the drone's additional **stopping distance** introduced by the **maximum projected tailwind**.

WIND ADJUSTMENT

High tailwinds increase stopping distance of drones.

Assumptions:

A11. Process: Operators do not launch the drone when wind conditions are outside operational specifications or forecasted to be outside operational specifications within the planned flight window.

A12. Physical Environment: When wind conditions are within operational specifications, regardless of wind direction, sideways drift of the drone is minimal and remains within an acceptable drift_margin.

A13. Physical Environment: Tailwinds extend drone stopping distances more than other relative wind directions.

A14. Process: The Gazebo simulation environment closely approximates the aerodynamic and physical interactions of a *typical drone* under varied wind conditions, reflecting real-world behavior with an acceptable degree of fidelity.

Collect upon startup

projectedWind
projectedWindGust

Compute upon startup

1: maxTailwindAdjustedStopDist = worst case stopping distance deviation of drone given maximum wind gusts projected, and tailwind.
2: windDriftDeviation = expected drift from flight path given perpendicular wind at maximum gusts projected.

Derived Requirement - 4

For each drone *d*, the autopilot shall compute its **distance projected to travel**, and the distance projected to travel of each neighboring drone *d′*, between status updates, given their current velocity.

PROJECTED DISTANCE MOVED

Each drone can move to a different geolocation between status reports.

Assumptions:

A9. Adjacent System: Status messages are transmitted over Doodle Labs Smart Radio – RM-2450 using an ultra-reliable Low Latency Channel with message latency of 3-30 ms. Messages take two hops creating a total transmission latency of 6-60 ms.

A10. Adjacent System: Environmental factors such as signal interference or communication delays beyond the established latency *T_com* are negligible or within controllable limits.

Collect upon startup

maxAcceleration
maxSpeed
fixedPeriod *P*
communicationLatency *T_com*

Collect at runtime

currentVelocity

Compute at runtime

1: for drone d: uncertaintyPeriod = fixedPeriod *P*
2: ∀ neighbor drone d': uncertaintyPeriod = fixedPeriod *P* + communicationLatency *T_com*
3: worstProjectedDistance = distance to have been travelled during the uncertaintyPeriod, considering the currentVelocity, maxAcceleration and maxSpeed possible.

Derived Requirement - 5

For each drone *d*, and each neighboring drone *d′*, there shall exist a minimum separation distance that must never be violated.

TOLERANCE DISTANCE

There is a tolerance distance assigned to every drone.

Assumptions:

A15. Operational Environment: Every drone knows its physical dimensions, broadcasts this information upon startup, and notifies new drones upon their entry to the airspace.

A16. Operational Environment: The tolerances built into the equations for dynamically computing minimum separation distance are sufficiently large to accommodate uncertainties associated with potentially non-linear interactions of stopping distances, wind, geo-location and other potential contributing factors.

Collect upon startup

dronesRadius [Lookup table]
comfortDistance

Collect at runtime

newEnteringDroneRadius

Compute upon startup

1: toleranceDistance [Lookup table] = dronesRadius[d,d'] + comfortDistance

Compute at runtime

1: toleranceDistance.update(newEnteringDroneRadius)

Fig. 4. An environmentally complex requirement is decomposed into derived requirements and environmental assumptions. At runtime, sensor data is continually collected as specified in the requirements to compute minimum separation distances, and to monitor that sUAS maintain safe distances between each other at all times.

flight log analysis [23]. Finding any case in which d_{comfort} is violated, could indicate that we have missed an important factor (e.g., humidity) or an inter-feature interaction (e.g., cold weather impacting the responsiveness of the flight controller) that is neither explicitly modeled nor accommodated in our tolerance factor and therefore needs to be modelled independently.

V. Threats to Validity

Our reliance on simulation results with limited field data introduces a potential gap between the controlled, simulated environment and the real-world conditions under which the drones will operate. Further, we used the Gazebo simulator and its underlying physics engines, without modifying the properties to match those of our physical drones. This creates a risk that the simulation-based findings will not generalize to physical environmental conditions. We have partially mitigated this problem through validating some of our simulation results, especially the wind-related analysis, against real-world data, and have performed preliminary stopping distance tests. Further, we have used this simulator as part of our devops environment for several years, and have observed high (but not perfect) fidelity with the real world. The field-data that we have collected so far supports our assumptions that differences between simulation and physical world outcomes are covered by tolerances built into our formulas.

In addition, as architects of DroneResponse, we are intimately aware of its requirements; however, it is possible that we have missed key factors for computing the minimum separation distance dynamically, and that these missing factors exceed the tolerances built into our formulas. The final Monte Carlo simulation and proposal for field-based runtime monitoring is designed to detect near violations of minimum-separation distance, triggering reconsideration of the key factors and our inbuilt tolerance levels.

VI. Related Work

In Michael Jackson's world-machine model [24], [25], the 'world' refers to the application domain or the specific problem context that the software aims to address, while the 'machine' is the software solution or system being developed to interact with this world. The 'specification' acts as a bridge between the world and the machine, describing the behavior the machine must exhibit at its interfaces to satisfy user requirements. Environmental assumptions are integral to the world-machine model as they define domain knowledge from the perspective of developers' beliefs about the external context and conditions within which the machine operates. However, the majority of work on environmental assumptions has focused on formal requirements modeling [1], [2] and safety assurance cases [26], [27]. In contrast, our case study has emphasized the role of environmental assumptions in a less formal software development process, characterized by natural language requirements, iterative design, and robust testing, without the use of formal methods. Several researchers have highlighted the central role of environmental assumptions across diverse areas such as automotive, aviation, and medical domains [12]. Leveson discussed approaches for avoiding assumption violations [11], [28] while Rahimi et al., discussed problems associated with missing and outdated assumptions [5]. Finally, Piccardi et al., [29] and Chowdhury et al., [30] described the role of well validated assumptions in safety assurance arguments.

Case studies have played an important role in requirements engineering for research associated with requirements synthesis, goal modeling, model checking, and architectural design. While recent discussions have attempted to narrow the definition of a case study [31], this term has traditionally been used to describe this type of in-depth example, and we therefore continue to use it in this paper. Commonly referenced case studies include the mine pump problem [32], a patient monitoring system [33], and the automated train control system [34]. However, the focus has been primarily on requirements, with less emphasis on environmental assumptions beyond stating that they must be identified and validated (e.g., [35]). In contrast, our case study emphasizes the role of the assumptions in the requirements specification and validation process.

VII. Conclusions

The case study presented in this paper provides an initial description of requirements and environmental assumptions for addressing the problem of dynamically maintaining safe separation distance between sUAS. The paper provides only a high-level discussion of the process we followed to identify assumptions, and does not propose or compare alternative modeling approaches based on techniques such as formal methods or goal-oriented approaches. This is by design, as our primary intention is to provide an in-depth example from the sUAS domain to serve as a case-study for other researchers interested in exploring diverse modeling and synthesis approaches.

In ongoing work we will conduct the currently unimplemented integration tests described in Section IV. Second we will extend our case study through more robust real-world testing, including evaluating the impact of additional environmental factors such as drone characteristics, and additional weather conditions, such as temperature which is known to impact flight performance. We will determine whether to explicitly model these additional characteristics. We will also investigate other environmentally complex requirements from our DroneResponse system, including various failure cases which are currently not fully addressed.

As this workshop paper represents an ongoing effort to specify requirements associated with challenging aspects of our DroneResponse project, we have established a github repository for reporting further results and for facilitating community discussion. The link can be found at https://github.com/SAREC-Lab/UAV-MinimumSeparation.

Acknowledgements

This work was primarily funded by grants SHF-1909007, CPS-1931962 from the USA National Science Foundation (NSF), and by grant 80NSSC21M0185 from the USA National Aeronautics and Space Administration (NASA).

References

[1] P. Zave and M. Jackson, "Four dark corners of requirements engineering," ACM Trans. Softw. Eng. Methodol., vol. 6, no. 1, pp. 1–30, 1997. [Online]. Available: http://doi.acm.org/10.1145/237432.237434

[2] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.

[3] D.Jackson, M. Thomas, and L.I.Millet, in Software for Dependable Systems: Sufficient Evidence?, National Research Council, 2007.

[4] T. T. Tun, R. R. Lutz, B. Nakayama, Y. Yu, D. Mathur, and B. Nuseibeh, "The role of environmental assumptions in failures of DNA nanosystems," in 1st IEEE/ACM International Workshop on Complex Faults and Failures in Large Software Systems, COUFLESS, 2015, pp. 27–33. [Online]. Available: http://dx.doi.org/10.1109/COUFLESS.2015.12

[5] M. Rahimi, W. Xiong, J. Cleland-Huang, and R. R. Lutz, "Diagnosing assumption problems in safety-critical products," in Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017. IEEE Computer Society, 2017, pp. 473–484. [Online]. Available: https://doi.org/10.1109/ASE.2017.8115659

[6] SAREC Lab, "A primer for environmental assumptions," https://github.com/SAREC-Lab/sUAS-UseCases/blob/master/use-case-minsepdistance/primer.md, accessed: [Insert date here].

[7] Drone Response Systems, "Drone response search and detect demo," https://youtu.be/DyKqxkesgg0, 2023, accessed: 10/11/2023.

[8] A. Agrawal, S. J. Abraham, B. Burger, C. Christine, L. Fraser, J. M. Hoeksema, S. Hwang, E. Travnik, S. Kumar, W. J. Scheirer, J. Cleland-Huang, M. Vierhauser, R. Bauer, and S. Cox, "The next generation of human-drone partnerships: Co-designing an emergency response system," in CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020. ACM, 2020, pp. 1–13. [Online]. Available: https://doi.org/10.1145/3313831.3376825

[9] PX4, "Open Source Flight Controller," https://px4.io, 2021, [Last accessed 01-06-2021].

[10] V. Ermakov, "MAVROS: A MAVLink extendable communication node for ROS for integrating with MAVLink-based drones," 2024, accessed on 2024-04-07. [Online]. Available: https://github.com/mavlink/mavros

[11] N. G. Leveson, Safeware, System Safety and Computers. Addison Wesley, 1995.

[12] S. Nair, J. L. de la Vara, M. Sabetzadeh, and L. C. Briand, "An extended systematic literature review on provision of evidence for safety certification," Information & Software Technology, vol. 56, no. 7, pp. 689–717, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2014.03.001

[13] "Estimator status flags," https://mavlink.io/en/messages/common.html#ESTIMATOR_STATUS, accessed: 2024-04-06.

[14] MAVLink Development Team, "Messages (common)," https://mavlink.io/en/messages/common.html, n.d., accessed: [03/20/24].

[15] DroneCode, "Mavlink - developer guide," https://mavlink.io/en, [Last accessed 01-11-2021].

[16] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in Proceedings of the Third International Conference on Simulation, Modeling, and Programming for Autonomous Robots, ser. SIMPAR'12. Berlin, Heidelberg: Springer-Verlag, 2012, p. 400–411. [Online]. Available: https://doi.org/10.1007/978-3-642-34327-8_36

[17] PX4 Development Team, "Gazebo simulation," https://docs.px4.io/v1.12/en/simulation/gazebo.html, 2021, accessed: 2024-03-28.

[18] M. N. A. Islam, M. T. Chowdhury, P. Alarcon, J. Cleland-Huang, and L. Spirkovska, "Towards an annotated all-weather dataset of flight logs for small uncrewed aerial systems," in AIAA AVIATION 2023 Forum, 2023, p. 3856.

[19] N. Metropolis and S. Ulam, "The monte carlo method," Journal of the American Statistical Association, vol. 44, no. 247, pp. 335–341, 1949. [Online]. Available: http://www.jstor.org/stable/2280232

[20] J. Yingni, Y. Xiuling, C. Xiaojun, and P. Xiaoyun, "Wind potential assessment using the weibull model at the inner mongolia of china," Energy Exploration & Exploitation, vol. 24, no. 3, pp. 211–221, 2006. [Online]. Available: https://doi.org/10.1260/014459806779367509

[21] Z. R. Shu and M. Jesson, "Estimation of Weibull parameters for wind energy analysis across the UK," Journal of Renewable and Sustainable Energy, vol. 13, no. 2, p. 023303, 03 2021. [Online]. Available: https://doi.org/10.1063/5.0038001

[22] M. Vierhauser, R. Wohlrab, M. Stadler, and J. Cleland-Huang, "Amon: A domain-specific language and framework for adaptive monitoring of cyber-physical systems," J. Syst. Softw., vol. 195, p. 111507, 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2022.111507

[23] M. N. A. Islam, Y. Ma, P. A. Granadeno, N. V. Chawla, and J. Cleland-Huang, "RESAM: requirements elicitation and specification for deep-learning anomaly models with applications to UAV flight controllers," in Proc. of the 30th IEEE International Requirements Engineering Conference. IEEE, 2022, pp. 153–165.

[24] M. Jackson, "The world and the machine," in 17th International Conference on Software Engineering, Seattle, Washington, USA, April 23-30, 1995, Proceedings, D. E. Perry, R. Jeffery, and D. Notkin, Eds. ACM, 1995, pp. 283–292. [Online]. Available: https://doi.org/10.1145/225014.225041

[25] ——, "System behaviours and problem frames: Concepts, concerns and the role of formalisms in the development of cyber-physical systems," in Dependable Software Systems Engineering, M. Irlbeck, D. A. Peled, and A. Pretschner, Eds. IOS Press, 2015, vol. 40, pp. 79–104. [Online]. Available: https://doi.org/10.3233/978-1-61499-495-4-79

[26] A. Bondavalli and F. D. Giandomenico, Eds., Computer Safety, Reliability, and Security- 33rd International Conference, SAFECOMP, ser. Lecture Notes in Computer Science, vol. 8666. Springer, 2014. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10506-2

[27] A. L. de Oliveira, R. T. V. Braga, P. C. Masiero, Y. Papadopoulos, I. Habli, and T. Kelly, "Supporting the automated generation of modular product line safety cases," in Int'l Conf on Dependability and Complex Systems DepCoS-RELCOMEX, 2015, pp. 319–330. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-19216-1_30

[28] N. G. Leveson, Engineering a Safer World: Systems Thinking Applied to Safety. MIT Press, 2012.

[29] C. Picardi, C. Paterson, R. Hawkins, R. Calinescu, and I. Habli, "Assurance argument patterns and processes for machine learning in safety-related systems," in SafeAI@AAAI, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:212419532

[30] T. Chowdhury, A. Wassyng, R. F. Paige, and M. Lawford, "Criteria to systematically evaluate (safety) assurance cases," in 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), 2019, pp. 380–390.

[31] C. Wohlin, "Do we publish credible evidence?" in 2023 IEEE 31st International Requirements Engineering Conference (RE). Los Alamitos, CA, USA: IEEE Computer Society, sep 2023, pp. 1–1. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/RE57278.2023.00008

[32] M. Joseph, Real-Time Systems: Specification, Verification and Analysis. Prentice Hall Intl., 1996.

[33] W. Stevens, G. Myers, and L. Constantine, "Structured design," IBM Systems Journal, vol. 13, no. 2, pp. 115–139, 1974.

[34] V. Winter, R. Berg, and J. Ringland, "Bay Area Rapid Transit District, Advance Automated Train Control System: Case Study Description," Sandia National Labs, Tech. Rep., 1999. [Online]. Available: http://www.sandia.gov/ast/papers/BART_case_study.pdf

[35] A. Murugesan, M. Whalen, S. Rayadurgam, and M. Heimdahl, "Compositional verification of a medical device system," vol. 33, 11 2013, pp. 51–64.