A Data Management Framework & UAV Simulation Testbed for the Study of System-level Prognostics Technologies

Timothy Darrah[1], Jeremy Frank[2], Marcos Quiñones-Grueiro[3], Gautam Biswas[4]

[1, 3, 4] *Vanderbilt University, Nashville TN, 37209 USA*
timothy.s.darrah@vanderbilt.edu
marcos.quinones.grueiro@vanderbilt.edu
gautam.biswas@vanderbilt.edu

[2] *NASA Ames Research Center, 1 Moffett Field, CA 94035 USA*
jeremy.d.frank@nasa.gov

Abstract

Prognostics-enabled technologies have emerged over the last few years primarily for predictive maintenance activites such as condition based maintenance (CBM), or its successor, CBM+, that accounts for the entire network of support elements required to execute a CBM program. However, due to the challenges that arise from real-world systems and safety concerns, they have not been adopted for operational decision making based on system *end of life* estimates. It is typically cost-prohibitive or highly unsafe to run a system to complete failure and, therefore, engineers turn to simulation studies for analyzing system performance. Prognostics research has matured to a point where we can start putting pieces together to be deployed on real systems, but this reveals new problems. First, a lack of standardization exists within this body of research that hinders our ability to compose various technologies or study their joint interactions when used together. The second hindrance lies in data management and creates hurdles when trying to reproduce results for validation or use the data as input to machine learning algorithms. We propose an end-to-end object-oriented data management framework & simulation testbed that can be used for a wide variety of applications. We describe the requirements, design, and implementation of the framework and provide a use-case application involving a stochastic data collection experiment that demonstrates how the framework can be used.

1. Introduction

The use of Unmanned Aerial Vehicles (UAVs) has rapidly grown over the last few years across a wide variety of applications that include aerial photography, surveillance, package delivery, cartography, agriculture, military missions, and more. As adoption and use of these vehicles increase, so does the risk of collisions and crashes that can result in a loss of money, time, productivity, and most importantly, human lives. As the health of these systems degrade over time, so does the risk of failure in flight and that is why safety is such an active area of research. System-wide safety[1], one of NASA's primary thrust in the Urban Air Mobility (UAM) program, deals with all aspects of safety within the operational context of the UAV and is a driving force in this area of research.

There is an abundance of research on the technical aspects of UAV systems: their design & implementation (Osmić et al., 2016); stability & risk analysis (Quiñones-Grueiro et al., 2021); decision making (Darrah et al., 2021); degradation (Gorospe et al., 2017; Darrah et al., 2020); and fault diagnostics (Moir & Seabridge, 2012). Prognostics and health management (PHM) technologies are of greater interest to us, which specifically addresses fault diagnosis and remaining useful life estimation in an effort to improve system reliability, safety, and maintainability. PHM technologies will play a critical role in the future of the next generation airspace.

NASA recently commissioned the National Academy of Sciences to conduct an in-depth study of the benefits and challenges of Advanced Air Mobility (National Academies of Sciences & Medicine, 2020). A key finding related to safety was that the **current state of the art in simulation technology is not adequate**. The report discusses the need for better tools to address both simulation and testing. Development of new technologies typically start with simulation and experimentation, and therefore a systematic approach to designing end-to-end simulation systems is needed.

In a similar vein, using a mix of tools and frameworks, along with custom software written in various languages, researchers

[1]https://www.nasa.gov/aeroresearch/programs/aosp/sws

around the world are developing prognostics applications with great results. However, (Li et al., 2020) summarize quite succinctly,

> *"existing literature addresses aspects of PHM design methodology and provides PHM architecture formulations. However, a systematic methodology towards a consistent definition of PHM architectures, i.e., one that spans the conceptual and application level, has not been well established."*

A complete "bits-to-batteries" approach to simulation that allows for the seamless composition of component models into a fully fledged dynamical system with data management and interoperability in mind is the thesis of this paper. We specifically focus on model and data management as the foundation for building PHM applications. Our contribution lies in an end-to-end simulation framework with a object-oriented model & data management design pattern. This allows for a more comprehensive study of degradation, failure, and remaining useful life; the generation of curated datasets for the development of machine learning models; and flexibility to simulate a multitude of vehicles (not just constrainted to UAVs) in a wide variety of environments and trajectories.

Paper Organization

The rest of the paper is organized as follows: Section 2 discusses the motivation behind our work; section 3 describes the testbed requirements; section 4 discusses the data management design pattern; section 5 details the simulation environment; section 6 details the framework implementation; section 7 provides the results and discussion; and section 8 is the conclusion with a brief overview of future work.

2. MOTIVATION

The origin of this work came from the need to generate data for deep learning and reinforcement learning based prognostics applications. We realized that the creation and curation of the data used to generate machine learning models was a bigger challenge than anticipated, and, this especially applies to the evaluation of PHM technology in general. Since data is generated from the simulations, it is important to account for how the data is generated, stored, and later retrieved. Many system-level prognosics experiments utilize some form of Monte Carlo simulation and take a considerable amount of time to finish. The amount of data can be quite substantial and a lot of time is spent organizing it *after-the-fact*. Often times the source code contains changes that are not reflected in the accompanying dataset, making it difficult to reproduce results. Different component models or degradation processes could be used in different experiments and these differences manifest in the resultant data. When this type of metadata is missing or the dataset lacks an accurate description, validation exercises are typically unsuccessful.

Tracking components, degradation models, environment models, and other "software artifacts" that generate data for the entire system and the environment is a critical piece that has been found missing in the PHM literature. System performance and overall state of health is affected by all of these factors and a robust PHM architecture should track them as well. In the context of safety, it can be disastrous if a prognostics algorithm is built off data generated from different components than those that are in the system it is operating in. System level prognosis is all about understanding how degrading system performance is a result of the joint interactions among all of these factors, and a **whole system-simulation framework** that supports this does not exist. This is where the shortcomings in currently available tools and frameworks come to light.

Simulation tools such as Simulink, LabVIEW or Modelica are industry standard for modeling components, designing systems, and performing simulations. Gazebo is a tool that focuses on environmental simulation and visualization, has a powerful physics engine and is great for simulating interactions with the environment. Robot Operating System (ROS) is a framework for developing robotics applications, and, can integrate with other tools. However, these are *open-ended* tools, not prognostics frameworks. The Generic Software Architecture for Prognostics (GSAP) (Teubert et al., 2017) **is** a framework for developing prognostic applications. This is an excellent software package developed by researchers at NASA with well defined interfaces and is easily extended. GSAP focuses on implementing common functionality across PHM applications, not so much on data management. The data management framework presented here can interoperate with any of the aforementioned tools and example usage is provided with Simulink.

3. REQUIREMENTS

Design decisions are driven by requirements, which are themselves driven by higher level goals that are to be achieved. In this case, the goal is to simplify and facilitate PHM research, and four key requirements to reach this goal are

Reproducibility. Being able to reproduce experiments allows for better peer-review of research, and provides a known starting place for other experiments (data, meta data, asset organization).

Explainability. Explaining what causes a particular configuration or algorithm to perform a certain way requires tracking all influences from models to algorithms, and tracing their influence through the process of data generation to models that control the vehicle.

Extensibility. When the existing codebase does not need to be significantly changed to accommodate new experiments, technologies and techniques can be developed and evaluated at a faster pace

Maintainability This is a measure of how easy it is to keep a consistent codebase or data storage, track bugs, and fix errors. Maintaining a consistent and organized codebase is an essential step in using data for machine learning models.

Capabilities

These requirements establish fundamental attributes that the data management framework should possess in order to address the motivating issues and support a wide variety of use cases. Five key applications include

- support different component & degradation models
- support multiple UAV types & configurations
- support multiple missions with different trajectories, tasks, risks, and rewards
- support Monte Carlo simulations
- support data logging with automatic validation[1] and a common interface for storing and retrieving data

We demonstrate the use of this framework with a Monte Carlo simulation study and through the discussion show how these requirements and capabilities are met. First, a brief overview of the system used is provided in the next section.

4. DATA MANAGEMENT DESIGN PATTERN

Object oriented design methodology is perhaps the most successful approach to planning and designing software. Advantages include the ability to reuse code, improved maintainability, reduce mistakes, and improve testing or debugging tasks (among many others). We propose an object oriented *design pattern* for *asset, process and data management* to record and organize high fidelity simulation data for the development, testing, and evaluation of PHM applications.

The foundations of the framework are **assets** (i.e. component models such as a motor, battery, sensor, etc), **processes** (degradation models and environmental factors such as wind), and **data** (such as flight telemetry or aggregate statistics). We provide a complete simulation environment as reference implementation of this framework for a UAV in MATLAB® and Simulink®. [2] [3]

Assets

Assets are the tangible components that make up the UAV. The asset acts as a *first class object* and is the archetype model that all components of the UAV inherit from. This includes the airframe, motors, battery, as well as any number and type of other components that might be used on the UAV. All physical components are assets, including the UAV. This a central point for the paper and key to interoperability among components, systems, environmental effects, and degradation models. This built-in modularity addresses two of the four requirements, extensibility and maintainability.

All assets have an associated *asset type* (shown in table 1), which holds meta-data about the asset. This table name is

[1]Data constraints set in the table definitions are checked and enforced by the database, not user code.
[2]A complete simulation package that can serve as a stand alone application or as an example to build one using this framework can be found at https://github.com/darrahts/uavTestbed.
[3]an API in Python is currently under development.

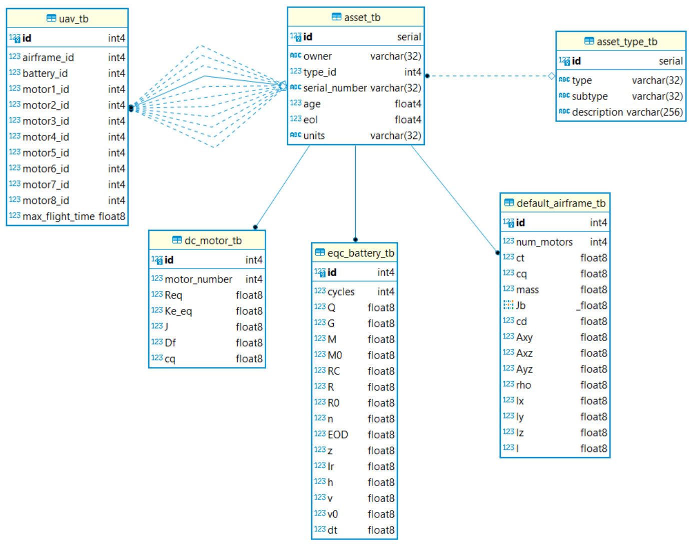

Figure 1. Asset Relationship Diagram

`asset_type_tb`, and it contains the fields `id`, `type`, `subtype`, and `description`. The `type` property is the high level component type such as an airframe or battery. The `subtype` property is used for further specification such as whether or not the battery is a *discrete-eqc* (discrete equivalent circuit) or a *continuous-ec* (continuous electro-chemistry). The `description` property contains information regarding the source of the *model* that is used by the *asset*. DC motor dynamics are well understood and a simple models like this can be considered `generic`. It may be more appropriate to cite an author an year for more complex models such as `plett_2015` for the battery.

id	type	subtype	description
1	airframe	octorotor	osmic_2016
2	battery	discrete-eqc	plett_2015
3	motor	dc	generic
4	uav	-	-

Table 1. Asset Types

Other asset types might include *electronics* or *sensors*, or even *equipment* could be a type that might include a communication device and a grappling mechanism for transporting goods. These are just a few examples and not meant to serve as an exhaustive list. The point for a robust PHM architecture is that if it generates data, degrades, or affects system performance, then it should be tracked.

Assets can be created after the asset types. The `asset_tb` contains the fields `id`, `owner`, `type_id`, and `serial_number`. The relationship between these two tables is depicted in Figure 1, where the `type_id` field of the `asset_tb` table is a *foreign key reference* to the `id` field of the `asset_type_tb` table. Four other tables are depicted in Figure 1 that store the model parameters for the asset types listed in table 1. Each table stores the parameters of the component model and is specifically related to that component model type. There is a one-to-one relationship between a given model class and an entry in the asset type table, while there can be any number of model instances of the same model class. Enforcing this constraint (and others) using a database management system is very straightforward and reliably works the way it is implemented. Such constraints ensure data integrity and directly address the other two high level requirements, explainability and maintainability.

Each table (with the exception of `uav_tb`) presented in Figure 1 comes with default parameters that are provided from the information in the `description` field of the `asset_type_tb`. Each of these tables is linked to the asset table via foreign key relationships on the `id` fields, whereby the asset table entry must exist before the derived component model entry is created. This decoupling between model parameters and the object representation they hold when used in simulation is another aspect that supports a robust data generation process and improves traceability.

Finally there is the `uav` type (table 1), a special type of asset that serves as a *container* that simply stores asset IDs of the installed components and links to the process and environmental models. This is one of many concepts applied that provide modularity and data organization, and this serves as the common interface among all components. Correctly storing and extracting data generated from different sources and experiments can be very tedious and prone to errors. This is addressed through the use of an underlying database management system with a well defined data management architecture. The metadata provided by the asset makes it easy to track the components with all other factors and sources of data within the simulation. This is a subtle, but very critical piece of the framework that is repeated with not just assets like motors and batteries, but with *processes*, like degradation or wind.

Processes

Each component has different degradation profiles and failure modes separate from the operation and health of the overall system. Much like tracking components as *assets* having *asset types*, degradation is tracked as a *process*, having *process types* (table 2). The relationship between processes and assets is shown in Figure 2. Each process has a foreign key relation `type_id` on the `process_type_tb` `id` field, which has a foreign key relation `asset_type_id` on the `asset_type_tb` `id` field.

With these relationships and constraints established, the joint interactions between various processes and whole system performance is captured in the database *by design.*

id	type	subtype1	subtype2	asset_type_id
1	degradation	battery	capacitance	3
2	degradation	battery	internal resistance	3
3	degradation	motor	internal resistance	2
4	environment	wind	gust	1
5	environment	wind	constant	1

Table 2. Process Type Table

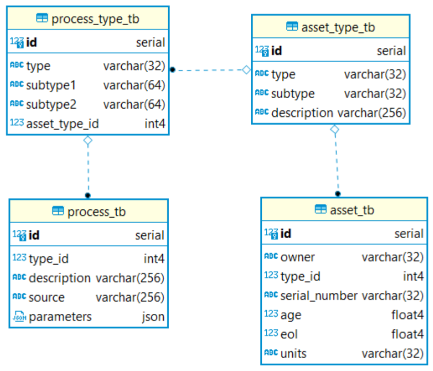

Figure 2. Process Relationship Diagram

Data Management

Experimental data is influenced by a multitude of factors and generated from a wide variety of sources within the simulation environment. In the context of prognostics applications, we are especially concerned with components, degradation models, environmental models, and other internal / external events that generate information. *Metadata* is just as important to capture as the raw data as well. Ensuring these complex interactions captured and all relevant data is logged and associated with the metadata of the UAV, components processes, and the environment, is the cornerstone of system-wide PHM applications.

The data relationship diagram can be seen in Figure 3 where three types of data are considered: *degradation* data, *summary* data, and *telemetry* data. The degradation and telemetry tables have a foreign key relation on the `flight_summary_tb`, which contains aggregate data from each mission such as its ending state of charge (`z_end`), average position error (`avg_pos_err`), or `distance` travelled [1].

All data that is generated is inherently organized correctly when this framework is used in conjunction with a simulation environment, regardless of whether it is a UAV, car, spacecraft, or other system. In this manner *the entire process of data generation, storage, retrieval, and analysis among flights with different components can be executed with the same code.* It is left to the individual practitioner to implement the dynamical models of their system, this framework handles everything else. We apply this framework to a UAV system in an urban environment as discussed in the following section.

[1]for a complete description of the table schemas see https://github.com/darrahts/uavTestbed/blob/main/sql/table_schema.sql

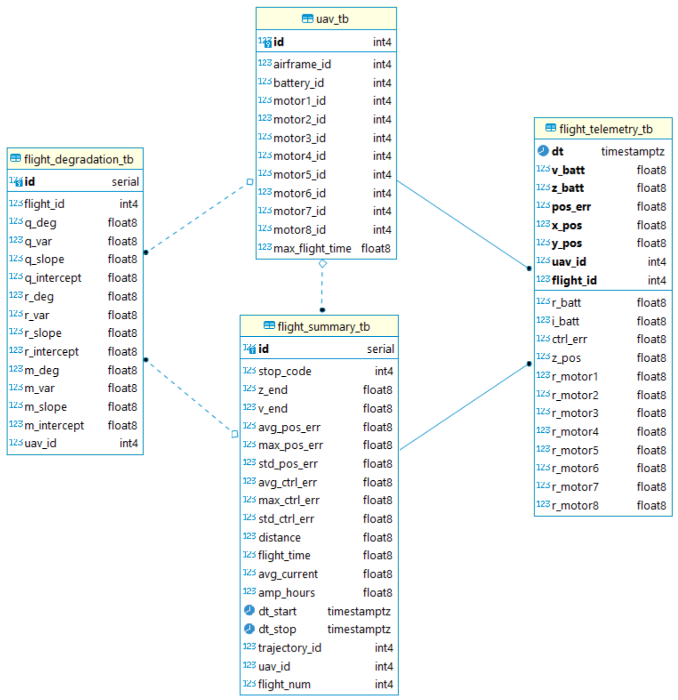

Figure 3. Data Relationship Diagram

5. SIMULATION ENVIRONMENT

System failure is not a function of individual component failure, but of (1) the compete interactivity among all the components within the system as well as (2) between the system and the environment. The simulation environment needs to have enough fidelity to realistically simulate the joint interactions and nuances among all the components and the environmental conditions the system is operating in. This level of fidelity is critical to evaluating PHM applications, e.g. comparing Remaining Useful Life (RUL) algorithms to each other, or in different environments or with different vehicles. The same considerations apply to machine learning, especially for PHM applications. Data driven models are only as good as the data they are trained on, and therefore the nonlinearities of system-level degradation with respect to individual component degradation, operational load, and environmental factors must manifest in the data. The coupling of the environmental conditions, the system, components, and degradation models (potentially more, this is not an exhaustive list), naturally lends itself to higher quality data, and therefore, higher quality data-driven models. Simulation to failure or capturing hard to understand edge cases are other benefits of end-to-end simulation infrastructures, and the system itself is only part of the simulation.

System Description

The system used in our experiments is a generic octorotor modelled with parameters taken from (Osmić et al., 2016). In previous work (Darrah et al., 2020) & (Quiñones-Grueiro et al., 2021), a DJI Mavic Pro and DJI S-1000 were used, respectively. Detailed modeling and implementation can be found in those publications, however a brief overview is provided below.

The octocopter airframe dynamics model is based on Newton-Euler equations of motion for a rigid body (Valavanis & Vachtsevanos, 2015). The derived body forces are

$$F_b = F_M + F_D + mgR_{IB}\mathbf{e}_z, \tag{1}$$

where $F_b \in \Re^3$ is the resulting force acting on the body frame, $F_M \in \Re^3$ is the resultant force generated by the motors, $F_D \in \Re^3$ is the drag force resulting due to the movement of a UAV through the air, m is the mass of the UAV, g is the gravity acceleration, $R_{IB} \in \Re^{3\times3}$ is the rotation matrix from the inertial frame to the body frame, and $\mathbf{e}_z = [0\ 0\ 1]^T$. The motors are simulated with a generic dc-motor model by (Schacht-Rodriguez et al., 2018) in equations 2 and 3:

$$\dot{\omega}_i = \frac{1}{J_m}(K_e i_c - T_{load} - D_f\omega - T_f), \tag{2}$$

$$i_c = \frac{1}{R_e q}(v_{DC} - K_e\omega_i), \tag{3}$$

$R_e q = \frac{2}{3}\sum_{j=1}^{3} R_j$ is the equivalent electric resistance of the coils, K_e is the back electromotive force constant, ω_i is the angular velocity of the ith motor, T_f is the static friction torque, D_f is the viscous damping coefficient that allows to estimate the dynamic friction torque ($D_f\omega$), J_m is the inertia of the motor, v_{DC} is the input voltage control signal, i_c is the current demanded from the battery pack, and T_{load} represents the torque load generated by the propellers. The battery is modeled using an equivalent circuit representation (Plett, 2015) with dynamic equations that characterize the battery behavior given by:

$$\dot{V}_{SoC} = -\frac{i_c}{Q} \tag{4}$$

$$\dot{i}_d = \frac{i_c}{R_d C_d} - \frac{i_d}{R_d C_d} \tag{5}$$

$$V_{out} = V_{ocv} - R_d i_d - R_0 i_c, \tag{6}$$

where Q represents the total capacity of the battery, i_c the input current, R_d and C_d are the diffusion resistance and capacitance, i_d the current going through the diffusion resistance, R_0 represents the internal resistance, V_{ocv} is the open circuit voltage, and V_{out} is the output voltage of the battery. A listing of the system parameters can be found in the appendix.

Degradation Models

Component degradation models (see Figure 4 of (Darrah et al., 2021)) are just as important to track as the components themselves, if a high degree of confidence in data quality is desired. Degradation profiles for three parameters considered were derived empirically through run to failure experiments as described in (Saha & Goebel, 2007; Jackey et al., 2009) [battery], or (Xuan et al., 2017) [motor]. These parameters degrade non-linearly over time (although they can be linearly approximated) and are dependent on several factors such as load demand or external disturbances. The three parameters considered in this work are

battery charge capacitance. This is the amount of charge that can be extracted from the battery in a fully charged state. Internal chemical processes within the battery and environmental factors such as temperature affect this parameter. Over time this value decreases. Degradation is a function of current draw, both instantaneously (fast discharges increase aging) and cumulatively (the total amount of current drawn from the battery over time).

battery internal resistance. This is a resistance to charge flowing out and is considered a separate degradation factor than charge capacitance. Over time this value increases and results in the battery reaching end of discharge (EOD) faster. Lithium corrosion, plating, and electrolyte layer formation affect this parameter (Daigle & Kulkarni, 2013).

motor #2 coil resistance. This is used as a proxy for modeling loss of performance due to different factors such as exposure to adverse weather conditions or general use. As the coil resistance increases, the amount of current drawn from the battery for the same throttle demand will be reduced, which translates into a reduction in thrust.

External Influences

External influences are environmental factors that affect the operation or performance of the system. These can include (but are not limited to)

temperature. Temperature is known to have an effect on both flight and battery performance. In high temperatures, flight performance is reduced due to the aerodynamic affects of increased molecular air speed and lift. In (Ma et al., 2018), the authors show that battery aging increases in high temperature, and battery charge capacitance decreases in low temperatures.

wind. In (Wang et al., 2019), the authors provide a comprehensive analysis of different types of wind, wind modelling, and their effects on flight. We implement a simple *turbulent* wind model via sampling of magnitude and direction values from a normal and uniform distribution, respectively. Wind effects create load on the airframe which puts stress on both the motors and the battery as the controller will attempted to compensate to maintain stability.

obstacles. Obstacles are either static, that is their location in the global coordinate frame (time and space) does not change, or they are dynamic, in which their temporal-spatial location does change. Static obstacle influence system behavior during the trajectory planning phase. Obstacles are part of the map and all trajectories are a minimum of 3 meters from any static obstacle.

We do not go into detail discussing *how* these external influences affect UAV performance (see previous references), but show that they are part of the whole simulation architecture.

Trajectories

The trajectories a UAV flies also plays a role in its overall performance and degradation of its components, as well as risk of failure or safety violation during flight. To be able to study these effects in depth, the trajectories must be tracked as well and linked to the data generated from them. In general, a trajectory consists of a timeseries multidimensional array whereby each axis represents the *x,y,z* plane in space and each triplet is a specific point in space at a specific time. There are 37 different trajectories that can be chosen from based on distance, estimated flight time, risk, or reward. Trajectories are formed in a two step process using probabilistic roadmaps (PRM) followed by B-spline smoothing.

PRMs (Kavraki et al., 1996) were introduced as a method of generating collision-free trajectories for robots in static environments (i.e. the environment is known ahead of time). This method consists of a *learning* phase and a *query* phase. In the learning phase, the algorithm generates a graph of the environment *"who's nodes correspond to collision-free configurations and who's edges correspond to feasible paths between these configurations."* In the query phase, any two locations are added to the graph, and a path is returned that connects these two points.

A thorough discussion on the application of B-spline smoothing to motion planning for robots can be found in (Magid et al., 2006). The basic concept is to smooth the transition of two intersecting lines about their intersection where the radius of the circle generated by the curvature of the smoothed intersection can be parameterized..

6. IMPLEMENTATION

The simulation environment is implemented with MATLAB® and Simulink and utilizes the design pattern described above for data management. The database is implemented with PostgreSQL, an open-source relational database management system. There are many benefits to this database engine such as being free, community driven, and widely supported. Using a database with an API will directly solve many issues faced when it comes to managing data generated from simulation experiments as detailed in this paper. In our example data collection experiment, all possible variables have been fixed - i.e. the parameters for different random variables (such as the battery charge capacitance degradation parameter, etc...) do not change from flight to flight. We are just asking the question, *"at what point will this UAV no longer safely fly this trajectory, given these degradation profiles and environmental conditions?"*

The simulation block diagram is shown below in Figure 4. In the **initialize workspace** block, system performance thresholds are defined, control and estimation parameters are loaded,

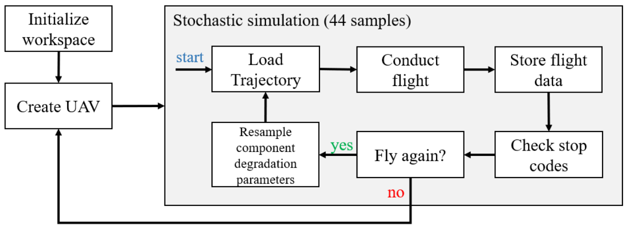

Figure 4. Simulation Experiment Block Diagram

and degradation profiles are loaded from the database. All the necessary degradation information can be retrieved with the following sql statement (listing 1) that joins three of the tables shown in Figure 2.

```
select ptt.id as process_type_id,
   ptt.subtype1 as process_subtype1,
   ptt.subtype2 as process_subtype2,
   ast.id as asset_type_id,
   ast."type" as asset_type,
   ast.subtype as asset_subtype,
   ast.description as model_source,
   pst.id as process_id,
   pst.description as process_description,
   pst."source" as degradation_model_source,
   pst.parameters
from process_type_tb ptt
inner join process_tb pst on pst.type_id = ptt.id
inner join asset_type_tb ast
on ptt.asset_type_id = ast.id;
```

SQL, like any language, has its own syntax and semantics that are easily picked up with regular use. For example, in lines 5 and 10, the words *type* and *source* are in quotes - this is because they are *keywords* in PostgreSQL. The data returned from this query can be viewed in table 7 of the appendix.

In the **create uav** block of Figure 4, a new UAV is created in the database and returned as a *struct* object. All data generated by the system is linked to the UAV; aggregation and other analysis methods use the UAV's `id` property to get all data generated by that UAV. In this experiment a UAV flight and degredation processes are simulated over and over again until the UAV reaches end of life (EOL). This is depicted in the **stocahstic simulation** grouping.

First, a trajectory is loaded from the database, then the flight is simulated within the simulation environment. The three sources of data described above, telemetry, summary, and degradation, are stored in the database. Next, the stop codes received from the simulation are checked in a logic block that informs us when the UAV has *violated a system performance parameter threshold.* These stop codes are stored in the `stop_code_tb` and are related to the `stop_code` field of the `flight_summary_tb`. Only valid stop codes are allowed to be entered as flight summary data. This is another of many constraints in place to ensure that data integrity is maintained. The decision to not fly again is made when the stop code is not *arrival success* for a count of 10 times (does not have to be sequential). This allows us to also catch data from failed flights during the stages of degradation that result in violating a system performance parameter. If the UAV does fly again, the degradation parameters are resampled from their distributions and the UAV flies again.

id	description
1	low soc
2	low voltage
3	position error
4	arrival success
5	average position error of entire flight
6	low soh (battery)

Table 3. Stop Codes

The data collected from 44 run-to-failure (RTF) samples totaling 3,624 flights is presented in the following section.

7. RESULTS & DISCUSSION

The first thing to look at is basic telemetry data from a typical flight when the system is healthy in figures 5, 6, 7, 8. The **telemetry** data used to generate these plots is comprised of a single flight only sampled at 4 hz and depicts the UAVs reference trajectory and actual position (Figure 5); position error (Figure 6); voltage and SOC sensor readings (Figure 7); as well as total current demand (Figure 8).

The `flight_telemetry_tb` contains all sensor data that comes from the UAV and all data that was used to generate these plots. Adding new sources of sensor data to the existing framework is quite easy with the correct PostgreSQL syntax and following a consistent grammatical naming convention.

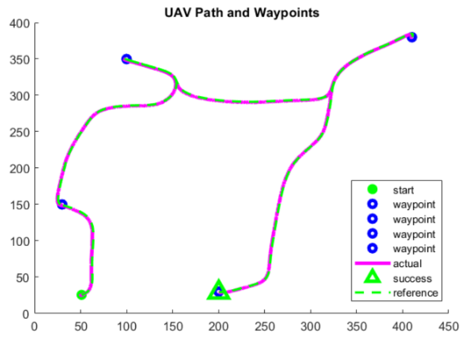

Figure 5. Trajectory # 3 Reference & Actual Position

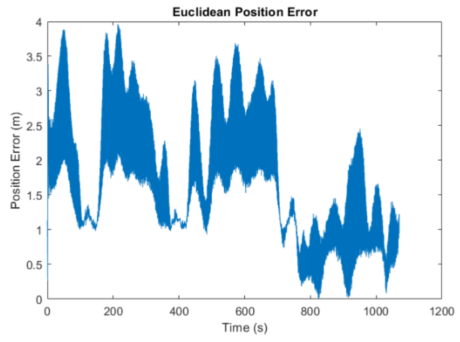

Figure 6. Position Error

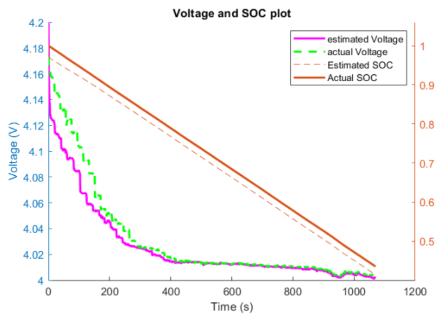

Figure 7. Voltage & State of Charge (SOC)

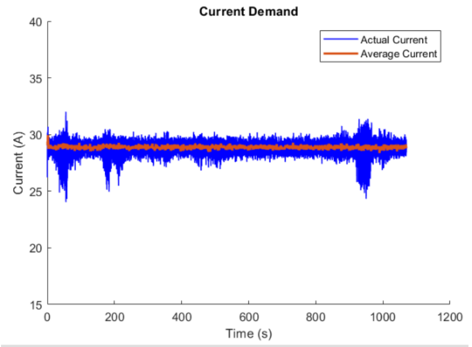

Figure 8. Current Demand

Next to look at is data from all of the flights contained in the `flight_summary_tb`, starting with the distribution of failure times (i.e. flight number) across all UAVs in the study. Figure 9 (top) is a bar graph of the number of flights before faliure for each UAV, and Figure 9 (bottom) is the failure distribution generated from this data.

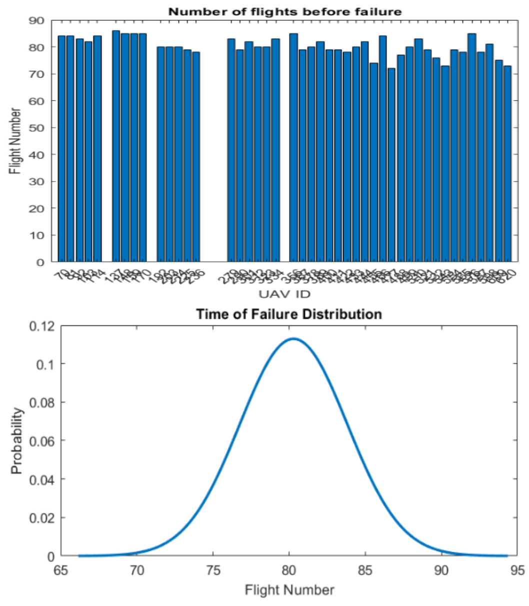

Figure 9. Time of failure distribution.

What these results show is that this specific UAV, components, component degradation profiles, external influences, and trajectory has typically violated a system performance parameter threshold 10 times (10 was chosen to ensure enough failure data is collected) by around the 80th flight. Retrieving the UAVs and the flight data for these plots can be done with the following *select* statement

```
select count(fst.*) as num_flights,
    fst.uav_id,
    ast.serial_number from flight_summary_tb fst
join asset_tb ast on fst.uav_id = ast.id
group by fst.uav_id, ast.serial_number
order by fst.uav_id asc;
```

Each `uav_id` can be further used as an index to query other data tables. Using a `uav_id` from the derived uav table, the system summary and degradation data for that and only that UAV across an entire simulation epoch (also referred to as a run-to-failure sample) can be returned.

The system performance parameters battery state of charge (top left), output voltage (top right), average position error (bottom left), and position error standard deviation (bottom right) are shown in Figure 10.

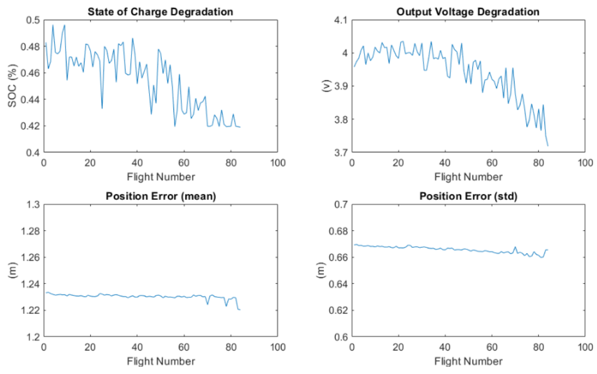

Figure 10. System Performance Parameters

It can be seen that each flight is indeed stochastic and that this particular UAV likely suffered from battery failure, as opposed to motor failure, even though both components are undergoing degradation. It is shown in the two position plots at the bottom of Figure 10 that system performance in terms of position error remained relatively stable until near the end of the simulation epoch.

Component degradation for this UAV is depicted in Figure 11. The capacitance degraded by roughly 14%, the battery internal resistance degraded by 1000%, and motor coil resistance degraded by about 23%. How would the performance have been different if the internal resistance degradation wasn't so extreme?

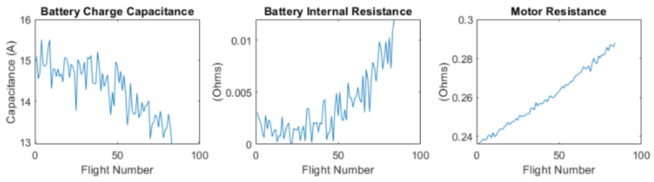

Figure 11. Component Degradation Parameters

In Figure 12 the stop codes for each flight taken by the UAV is shown. The majority of the time the UAV successfully reaches all waypoints of the trajectory. A few times around the 60th flight the stop code is low SOC, meaning that the flight had to be stopped because the UAV's battery SOC went below the minimum SOC threshold (a parameter set in the *initialize workspace* block described above). Then around the 75th flight the same stop code was repeatedly triggered, and ultimately the sample was terminated.

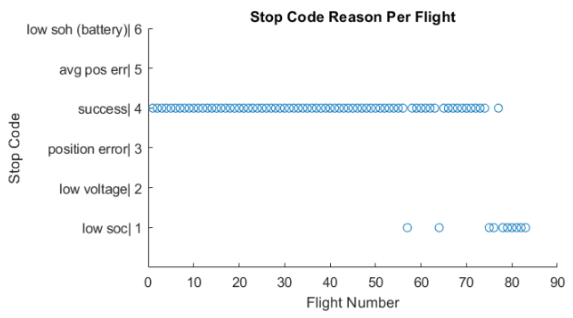

Figure 12. Stop Codes by Flight

Next, the evolution of the system performance parameter distribution over time is depicted in Figure 13 for the battery SOC, and in Figure 14 for the position error standard deviation. The battery state of charge appears to progress fairly linearly across the first 50 flights and then in the last couple of flights the distribution tightens up quite a bit. This suggests that the majority of the failures occured when there was 42% remaining state of charge at mission completion.

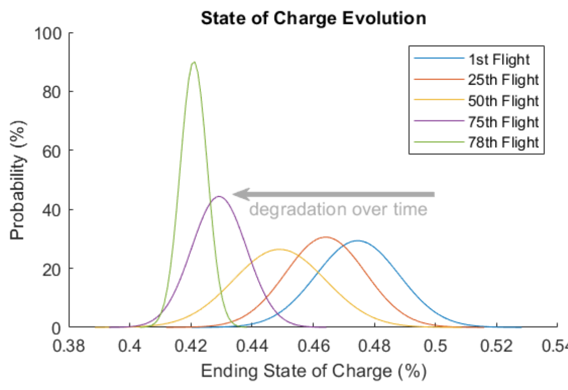

Figure 13. State of Charge Degradation Distributions

The position error standard deviation was chosen as a system performance parameter because it is a measure of *stability* and *volatility* that offers a different view of what is happening than just looking at raw position error data. Here it is shown that the distributions actually *grows*, indicative of a *decrease in stability*. Given that the UAV had zero stop codes related to motor degradation, position, stability, or control, one observation from this result is that battery degradation plays a bigger role in overall UAV flight stability than is given credit for. It is also seen that this growth is on a very small scale, so this remains inconclusive. With a tightly coupled simulation environment and data management framework, this can easily be studied further.

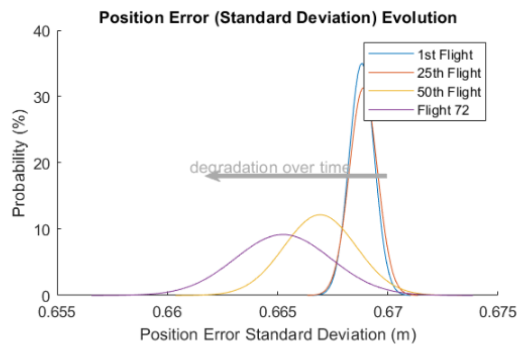

Figure 14. Position Error Standard Deviation Distributions

These results were generated using the framework described in this paper via standard SQL queries to the database. The common interface provided by this framework ensures the different components' data is automatically registered with the database and allows for repeatability among experiments and reuse of code, regardless of the vehicle under study or application.

We now describe how the design meets our requirements. UAV component and degradation models are easily declared and composed, as are combinations of maps and trajectories. High-quality data is generated by ensuring the environment and degradation models propagate component-level health into system-level behavior. Metadata from the UAV components, environment and data models is automatically recorded. Monte-Carlo simulations generate per-flight data; because of the data-logging requirements imposed on assets, this data is **inherently organized** and structured.

The organization of the data imposed by the asset-process-data paradigm ensures it can easily be used to evaluate PHM applications and research questions. For example, instead of ending the sample after 10 failures, what if we *change the battery*? How many battery changes will it take to notice the effects of motor degradation? How does the position error evolve over time after multiple battery changes and continuous motor degradation? Or, perhaps consider the evolution of battery SOC degradation in Figure 13. How does this difference continue to grow as we fly other trajectories or change batteries? Or in different wind or temperature environments? What if we use different degradation models? These are all great research questions that are of high interest to the PHM community.

This architecture for data management will enable researchers to dig deeper into these questions with greater ease, and with a high degree of confidence that their results are valid. It also facilitates using the data as input to machine learning tasks such as model development for remaining useful life (RUL) prediction, or decision making making, among many other uses. This framework simplifies the entire simulation process takes care of the tedious and error prone tasks of data management.

8. CONCLUSION & FUTURE WORK

In this paper, an asset, process, and data management framework for the research and development of PHM applications has been proposed. This work was motivated by the lack of a comprehensive simulation environment and data management architecture that addresses requirements specific to PHM research. Therefore, a requirements analysis into the specific needs of PHM research and development was done and a new framework has been developed from the ground up. The framework was demonstrated with an end-to-end simulation environment implemented in MATLAB®. Using this framework, simulation changes are easily tracked, generated data is inherently organized, and data integrity is guaranteed. Collaboration is also facilitated when different researchers are using the same framework, making it easier to share code, reproduce results, and build off others work. We demonstrate the use of this framework with a data generation experiment and are left with a couple of different open ended questions that, with this simulation architecture will soon be studied.

Part of the continuation of research will be directed at improving the fidelity of simulation environment with better degradation and process models (i.e. implementing a Dryden or Von Karman wind gust model as opposed to stochastic sampling of magnitude and direction values), and better simulation of take-off and landing which have different power demand profiles than hovering or cruising at altitude. Then there is also improvements with incorporating risk and reward information for decision making applications, among countless other extensions and improvements that could be made.

Of more interest to the general PHM community is the future work on the framework itself. Does it generalize to other types of PHM simulation use cases, vehicles, or systems? Does it scale to dozens, hundreds, or even thousands of vehicles? Does the framework behave well in real-world operation, and, are there time constraints that need to be considered? Can this framework be used to automate the evaluation and testing of different prognostics algorithms on the same system? There is certainly no shortage of research questions to address, using this framework.

ACKNOWLEDGMENT

- NASA grant 80NSSC19M0166 from the NASA Shared Services Center
- NASA OSTEM Fellowship 20-0154.

REFERENCES

Daigle, M., & Kulkarni, C. (2013). Electrochemestry-based battery modeling for prognostics. In *Annual conference of the prognostics and health management society.*

Darrah, T., Kulkarni, C. S., & Biswas, G. (2020). The effects of component degradation on system-level prognostics for the electric powertrain system of uavs. In *Aiaa scitech 2020 forum.* doi: 10.2514/6.2020-1626

Darrah, T., Quiñones-Grueiro, M., Biswas, G., & Kulkarni, C. (2021). Prognostics based decision making for safe and optimal uav operations. In *Aiaa scitech 2021 forum.*

Gorospe, G. E., Kulkarni, C. S., Hogge, E., Hsu, A., & Ownby, N. (2017). A study of the degradation of electronic speed controllers for brushless dc motors..

Jackey, R., Plett, G., & Klein, M. (2009). Parameterization of a battery simulation model using numerical optimization methods. *SAE International.*

Kavraki, L., Svestka, P., Latombe, J.-C., & Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation.*

Li, R., Verhagen, W. J., & Curran, R. (2020). A systematic methodology for prognostic and health management system architecture definition. *Reliability Engineering System Safety, 193.*

Ma, S., Jiang, M., Tao, P., Song, C., Wu, J., Wang, J., ... Shang, W. (2018). Temperature effect and thermal impact in lithium-ion batteries: A review. *Progress in Natural Science: Materials International, 28.*

Magid, E., Keren, D., Rivlin, E., & Yavneh, I. (2006). Spline-based robot navigation..

Moir, I., & Seabridge, A. (2012). *Design and development of aircraft systems* (Vol. 67). John Wiley & Sons.

National Academies of Sciences, E., & Medicine. (2020). *Advancing aerial mobility: A national blueprint.*

Osmić, N., Kurić, M., & Petrović, I. (2016). Detailed octorotor modeling and pd control. In *2016 ieee international conference on systems, man, and cybernetics (smc).*

Plett, G. L. (2015). *Battery Management Systems Volume 2: Equivalent-Circuit Methods.* Artech House.

Quiñones-Grueiro, M., Biswas, G., Darrah, I. A. T., & Kulkarni, C. (2021). Online decision making and path planning framework for safe operation of unmanned aerial vehicles in urban scenarios. *Aerospace Journal (to appear).*

Saha, B., & Goebel, K. (2007). Battery data set. *NASA Ames Prognostics Data Repository.*

Schacht-Rodriguez, R., Ponsart, J., Garcia-Beltran, C., Astorga-Zaragoza, C., Theilliol, D., & Zhang, Y. (2018). Path planning generation algorithm for a class of uav multirotor based on state of health of lithium polymer battery. *Journal of Intelligent and Robotic Systems, 91:115-131.*

Teubert, C., Daigle, M. J., Sankararaman, S., Goebel, K., & Watkins, J. (2017). A generic software architecture for prognostics (gsap). *International journal of prognostics and health management.*

Valavanis, K. P., & Vachtsevanos, G. J. (2015). In *Handbook of unmanned aerial vehicles* (chap. Quadcopter Kinematics and Dynamics). Springer.

Wang, B., Wang, D., Ali, Z., Ting, B., & Wang, H. (2019). An overview of various kinds of wind effects on unmanned aerial vehicle. *Measurement and Control*.

Xuan, J., Wang, X., Lu, D., & Wang, L. (2017). Research on the safety assessment of the brushless dc motor based on the gray model. *Advances in Mechanical Engineering*.

APPENDIX

Parameter	**Desc**	**Value**
ct	rotor thrust coef.	$8.5486x10^{-6}$
cq	rotor drag coef.	$1.3678x10^{-7}$
cd	translational drag coef.	1.0
mass	total mass	1.8kg
Jb	inertia matrix	$\begin{bmatrix} .0429 & & \\ & .0429 & \\ & & .0748 \end{bmatrix} kg - m^2$
A_{xy}	cross sectional area	.9m
A_{xz}	cross sectional area	.5m
A_{yz}	cross sectional area	.5m
l	arm length	0.45m

Table 4. Airframe Parameters

Parameter	**Desc**	**Value**
$R*$	coil resistance	$.8\Omega$
Ke	back EMF	.0068
Tf	friction torque	$1e^{-8}$
Df	viscous dampening	$1.6e^{-7}$
d	drag constant	$1.71e^{-8}$
j	inertia	$4.9e^{-6}$

** degradation parameter*

Table 5. Motor Parameters

Parameter	**Value**
$Q*$	$15000mAh$
η	.9929
γ	.1199
M_0	$1e^{-4}$
M	$1e^{-6}$
R_0	$.0112\Omega$
R_1	$2.83e^{-4}\Omega$
C_1	$12.93\mu F$
V_0	$4.2v$

** degradation parameter*

Table 6. Battery Parameters

process_type_id	**process_subtype1**	**process_subtype2**	**asset_type_id**	**asset_type**	**asset_subtype**
1	battery	capacitance	3	battery	discrete-eqc
2	battery	internal resistance	3	battery	discrete-eqc
3	motor	internal resistance	2	motor	dc
4	wind	gust	1	airframe	octorotor

model_source	**process_id**	**process_description**	**degradation_model_source**	**parameters**
plett_2015	1	discrete cycle based	NASA prognostics dataset	{”qdeg”: [15.0, 14.99 ...]}
plett_2015	2	discrete cycle based	NASA prognostics dataset	{”rdeg”: [.0011, .0011 ...]}
generic	3	discrete, mission-based	artificial nonlinear profile	{”mdeg”: [.2371, .2372 ...]}
osmic_2016	4	simulated wind gusts	trial & error	{”x”: [1.0, .25], ”y”: ...}

Table 7. Process data Table