OpenUAS Version 1.0*

Chris Johannsen[1], Marcella Anderson[1], William Burken[3], Ellie Diersen[2], John Edgren[2], Colton Glick[1], Stephanie Jou[2], Adhyaksh Kumar[2], John Levandowski[2], Evelyn Moyer[2], Taylor Roquet[2], Alexander VandeLoo[2], and Kristin Yvonne Rozier[1,2]

***Abstract*— The future of fixed-wing autonomous aircraft operations depends on the availability of an appropriate UAS testbed. The testbed must be accessible: relatively inexpensive and easy to come by, without requiring long shipping delays from small international companies. It must be reconfigurable to accommodate the vast range of different sensor packages, payloads, use-cases, and flight characteristics needed to accommodate a broad variety of autonomy operations. It must be easy to fix or find replacement parts that will inevitably break during rigorous testing and it must accommodate safety analysis on-board with a deep understanding of the aircraft's design. Yet no previous fixed-wing UAS meets this need. Therefore, we contribute the first completely open-source (in both hardware and software) fixed-wing UAS, designed for reconfigurability and accessibility of broad audiences from researchers to high school students.**

I. INTRODUCTION

Fixed-wing electric aircraft offer many advantages for AI and autonomous operations such as surveillance and mapping tasks; they are a particularly nice testbed for operations involving safe recovery and resilience. In order to design, test, and fly increasingly autonomous operations, we need a fixed-wing UAS testbed that is accessible in terms of both availability and cost, configurable to host the sensor packages and on-board computing payloads required for autonomy, and easy to repair since flights will inevitably result in occasional loose or broken parts.

However, no currently-available fixed-wing UAS meets these needs. Larger aircraft, like the 13-foot wingspan NASA Swift UAS [1] make great testbeds for AI-enabled operations, e.g., [2], [3] but are too large and expensive for many research labs and academic uses. Smaller platforms, like NASA's 1.14 m wingspan DragonEye UAS [4], [5], are difficult to re-configure for missions requiring different sensor suites, have payload capacities that are too oddly shaped and cramped to facilitate the instrumentation required for real-time monitoring, restrict internal airflow in a way that causes AI components like FPGAs or Raspberry Pis to overheat, have battery lives that are too short to enable rigorous experimental field evaluation in complex environments, and are too restricted or expensive for use in academic environments [3]. If a battery ages on a DragonEye, it cannot be replaced since it is fused into the fuselage. If a wire comes loose during a flight test, it cannot be re-secured without carefully drilling through the fuselage.

Hobbyist options like the 1.5m wingspan E-flite Apprentice solve the cost accessibility challenge with retail prices in the hundreds of dollars, but are short on room for additional sensors or payloads needed to host AI operations and are not reconfigurable [6]. When additional parts can be crammed in they suffer the same overheating problems as the Dragon Eye and are also difficult to repair. Perhaps the closest UAS to meeting the need for an autonomy testbed is the 3 m wingspan Applied Aeronautics Albatross [7], advertised as "the first affordable long-range drone." The albatross has many desirable characteristics including a 4-hour battery life, 68 km/h cruise speed, and an easily-accessible, roomy fuselage that allows for experimental payloads. However, it is not inexpensive (ready-to-fly systems start at $6,600 USD) and the beautifully-crafted shell is not easy to repair.

A major problem with purchasing, even "open-source," UAS is the frequency with which their designs change, making it difficult to impossible to get replacement parts. The design for the Albatross changed in the middle of our order shipment, causing us to receive some parts from two different versions of this aircraft; we had to kludge together the initial craft from these two versions with long wait periods for new parts and incomplete documentation as to how they fit together. The E-flight Apprentice has already been discontinued [8]. The software for these and other UAS may be open, but there is still no end-to-end release of the CAD models and specifications for each component, or the system integration plan as to how they all fit together (and how changes in any one component affect the rest of the design).[1] To make a truly fixable, reconfigurable UAS testbed, we need open-sourcing of *how* the UAS is designed, including full specifications for manufacturing *all* parts (not just software), so replacements or alternatives can always be quickly manufactured from the open designs.

We contribute the first truly open-source (in both hardware and software) design for a fixed-wing UAS. This includes opening CAD models, manufacturing instructions, and all other data and instructions needed to easily, quickly, and cheaply manufacture or modify every part of the OpenUAS,

*This work is partially supported by NSF CAREER Award CNS-1664356 and Iowa Space Grant Consortium (ISGC) Award No. 80NSSC20M0107. Full datasets and documentation are available: http://temporallogic.org/research/ICUAS21/

[1]Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, 50011 USA {chrisj17, thing1, crglick}@iastate.edu

[2] Department of Aerospace Engineering, Iowa State University, Ames, Iowa, 50011 USA {ediersen, edgrenj, johnl, ssjou, adhyk, elmoyer, troquet, aavande, kyrozier}@iastate.edu

[3] Department of Mechanical Engineering, Iowa State University, Ames, Iowa, 50011 USA {weburken}@iastate.edu

[1]Some 3D printable UAS parts are available online for a fee [9].

in addition to all software and procedures for flight testing and safety. We even add specifications for on-board runtime verification (RV) via the R2U2 open-source RV engine.[2] We analyze the results from recent flight testing and examine our design and manufacturing choices to enable others to specialize and build upon our design.

This paper is organized as follows. Section II overviews the OpenUAS design, including CAD modeling, parameters for the wings, vertical, and horizontal tails, stability analysis, CFD analysis, landing gear, and materials considerations. We detail the manufacturing process in Section III, including the materials choices, manufacturing process, and lessons learned. To automate the OpenUAS, Section IV lays out the electronics and software schematics, including details of the electronic hardware, motors, propellers, batteries, and avenues for control. We include discussions of simulations and support software. Accounts of our flight testing and evaluation appear in Section V, including flight procedures, and our setup for on-board runtime verification. Section VI discusses impacts and designs for future work.

II. DESIGN

The OpenUAS is a fixed-wing aircraft with a 6 ft wingspan and a distance from nose to tail of 38 inches. The OpenUAS uses a tractor styled propulsion system. Figures 1 and 2 show an isometric view and a right view of the OpenUAS's CAD model created through Solidworks.

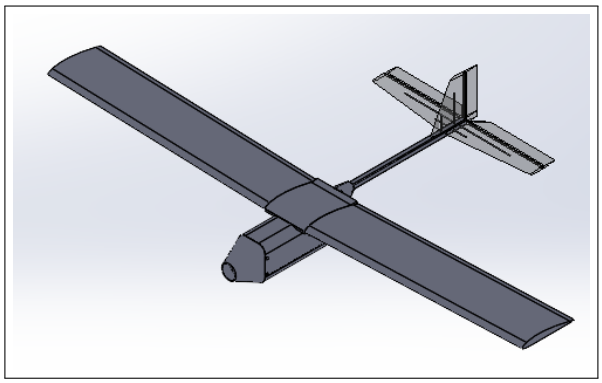

Fig. 1. Isometric View of OpenUAS in Solidworks; 6 ft wingspan

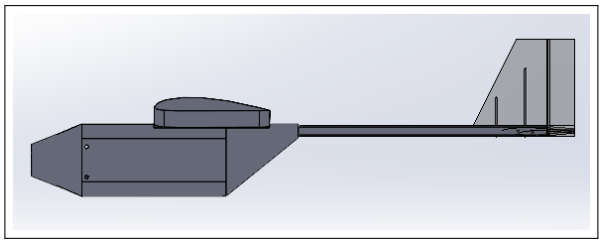

Fig. 2. Right View of OpenUAS in Solidworks; 38 inchs nose-to-tail

A. CAD Modeling

We modeled the OpenUAS in Solidworks, creating full CAD models of each of the OpenUAS model's components. This approach leads to comprehensive documentation of the UAS and each of its components.

1) Wings and Airfoil Selection: For this design, four airfoils were studied for comparison. These include the AG35, NACA 4512, S1223, and Clark Y. The data for each airfoil was extracted from the UIUC Airfoil Database [10]. The NACA 4512 was the preliminary choice of airfoil due to familiarity from an initial study by the team using this airfoil. We analyzed the airfoils using the program XFLR5 [11] under the following conditions: a Reynolds number of 172,000 corresponding to a speed of 20 m/s, chord width of 0.122 m, and kinematic viscosity of 1.4207×10^{-5} m^2/s. This corresponds to a Mach number of 0.0059.

The following graphs were analyzed with XFLR5: C_l vs α, C_l/C_d vs α, C_l vs C_d. With the graphs, the information for the $C_{l,0}$, $C_{l,max}$, stall angle, $C_{d,min}$, and maximum efficiency E_{max} was extracted. An analysis using a modified version of the Weighted Scoring Method shown in [12] was used. Each parameter has a multiplier and points between 1-4 based on how the parameters for each airfoil compare to each other.

Table I shows the weighted results between the NACA 4512 airfoil and the Clark Y airfoil chosen for the design. Additional documentation of our airfoil selection process can be found in [13]. The wings in the model have a 2° dihedral angle that provides more stability on the OpenUAS.

Parameter	NACA 4512	Clark Y	Multiplier
$C_{l,0}$	2	3	1.2
$C_{l,max}$	3	1	1.25
α_{stall}	1	3	1.15
$C_{d_{min}}$	2	3	1.15
E_{max}	4	3	1.25
Total Points	**14.6**	**15.5**	

TABLE I

AIRFOIL WEIGHTED RESULTS USING MODIFIED WEIGHTED SCORING METHOD AND POINTS BETWEEN 1-4 BASED ON AIRFOIL COMPARISON OF EACH PARAMETER

2) Vertical and Horizontal Tails: We selected the NACA 0012 airfoil for the tail section due to its symmetry and because of its typical usage for tail sections in RC planes. The horizontal tail has a span of 24 inches with 35% of its chord allocated for the elevators. The vertical tail has a height of 6 inches with 25% of its chord allocated for the rudder. Details of the tail sizing can be found on the supplemental website [14]. The horizontal and vertical tails are 24 and 22 inches behind the leading edge of the wing at the root chord respectively.

B. Stabilty Analysis

After completing the CAD model of the OpenUAS, we conducted a stability analysis using AVL [15]. We use the airfoil and geometric specifications of the OpenUAS to model the wing and tail sections. The center of gravity is calculated using the CAD model and used with the normal point value from AVL to get the static margin of the OpenUAS. Figure 3 shows the AVL geometry.

[2] http://r2u2.temporallogic.org/

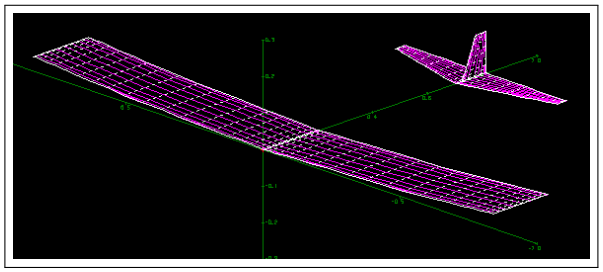

Fig. 3. AVL Geometry showing Wing-Tail Relation, 2° Dihedral Angle, and 24-inch distance between Horizontal Tail and Wing

Running the simulation, the neutral point obtained is located 0.10672 m behind the wings' leading edge. Using this value with the center of gravity located at 30% of the wing chord (0.06096 m from leading edge) and the 8-inch chord value, the static margin is 22.5%.

We analyzed the static stability by looking for the following stability derivatives: the pitching moment coefficient with respect to the angle of attack ($C_{m\alpha}$), the rolling moment coefficient with respect to sideslip angle ($C_{l\beta}$), and the yawing moment coefficient with respect to sideslip angle ($C_{n\beta}$). Recall that for static stability, $C_{m\alpha}$ and $C_{l\beta}$ should be negative while $C_{n\beta}$ should be positive. From the analysis done in AVL, the 3 stability derivatives are in the expected ranges. This confirms the model is statically stable. The results for the stability derivatives appear in Table II.

Stability Derivative	Value Returned
$C_{m\alpha}$	-1.0458
$C_{l\beta}$	-0.0429
$C_{n\beta}$	0.0166

TABLE II
STABILITY DERIVATIVES AVL RESULTS

The OpenUAS has an electronics bay holding the electronic hardware in Table IV. In this bay, the battery is placed in line with the 30% wing chord location. The battery may also be shifted slightly forward or backward, thus enabling the adjustment of the center of gravity location as needed.

C. CFD Analysis

We conducted CFD analysis of the OpenUAS model using StarCCM+ [16]. We created and ran simulations using take-off data in Ames, Iowa. The simulations were run at angles of attack ranging between 0 and 18 degrees with increases of 3 degrees. Each simulation was run for 4000 iterations and used over 12 million cells. From the simulations, we obtain the values for the lift and drag coefficients as well as the lift and drag generated. We also generated four performance curves: C_l vs α, C_d vs α, C_m vs α, and L/D vs α. From these curves, we optimized the design until the lift generated was higher than the weight of the OpenUAS and the C_m vs α curve showed a negative slope.

D. Additional Design Work

1) Landing Gear: For the initial flight, the OpenUAS had no landing gear, but this was added shortly after. This allows the OpenUAS to take off and land on unimproved runways, reducing the impact from a belly landing. Two options were considered based on off the shelf products: conventional (tailwheel) and tricycle (nose wheel). After some analysis, the tailwheel option was selected, designed in Solidworks, and added to the model. The landing gear was bolted to the lower surface of the fuselage and the tailwheel was mounted below the vertical stabilizer. Figure 4 shows the model with the addition of the conventional landing gear.

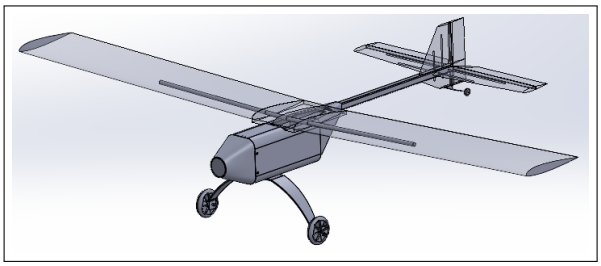

Fig. 4. OpenUAS Model in Solidworks with Conventional Landing Gear and Material Considerations

2) Material Considerations: The current wings and fuselage are designed in carbon fiber. Given the importance of accessibility and ease of manufacturing for the design, we understand that completing carbon fiber layups for the wing and tail pieces may not always be a viable option for all interested parties. The materials to complete this process are expensive and not everyone is familiar with the handling of carbon fiber or composites. As such, the team is creating design variations that remove the need of doing the carbon fiber layup by including carbon fiber rods inside the wings and tail sections that would add strength to the components and reduce the weight added with the fiber and epoxy. Further tests will be completed to finalize the manufacturing process for future variations of the OpenUAS.

III. MANUFACTURING

A. Material Choices

Materials are chosen based on three main characteristics: low weight, high strength, and endurance. Therefore, the bulk of the OpenUAS is constructed using carbon fiber for its combination of high strength and low density (270 ksi and 1.75 lb/yard). The layups are done on XPS 25 pink foam molds. They are finished with a variety of dremels, hand sanders, and files used with P95 masks to mitigate the effects of the airborne carbon fiber particles.

The wing support is manufactured out of fiberglass to ensure the structural integrity of the connection between the wings and fuselage. The electronics bay is made out of posterboard to keep the overall weight low without sacrificing structural integrity of integral parts.

The rest of the parts are purchased ready-to-use from a variety of sources, including the landing gear, various fasteners, and tail boom. These parts differentiate themselves from the others due to their complexity, small tolerances, and purchasing accessibility.

To launch the OpenUAS, we designed a launch rail consisting of $\frac{1}{2}$ inch bolts and $1\frac{1}{2}$ inch square aluminum tubing.

Component	Material	Price
Fuselage, Nose Cone, Wings, Tail construction	Toray T300 Carbon fiber layout on XPS 25 pink foam using 250 F epoxy resin	$250
Electronics Bay	Poster board (substituted for fiber-glass)	$5.00
Wing Support	Fiberglass	$18.00
Front Landing Gear	DuBro Super Strength Landing Gear (.35-.5) paired with DuBro 3" Super Lite Wheels	$20.00
Rear Landing Gear	DuBro .4 Plane Tailwheel Bracket paired with DuBro 3/4" Tailwheel	$ 3.40
Wing Fasteners	Great Planes Nylon Wing Bolts 1/4 - 20x2	$6.50
Epoxy	JB Weld 5 Minute Epoxy	$6.98
Launch Rail Frame	1.5" aluminum square tubing (39ft)	$110
Launch Rail Interface	80/20 1010 1" Rails (12ft)	$60
Launch Rail Fasteners	1/2" Steel Bolts	$10
Launch Rail Interface Hardware	Standard 1" Rail Buttons	$6.70
Launch Rail Carrier	3D printed PLA	$ .50
	Total	**$497.08**

TABLE III

OPENUAS V1.0 MANUFACTURING MATERIALS PARTS LIST

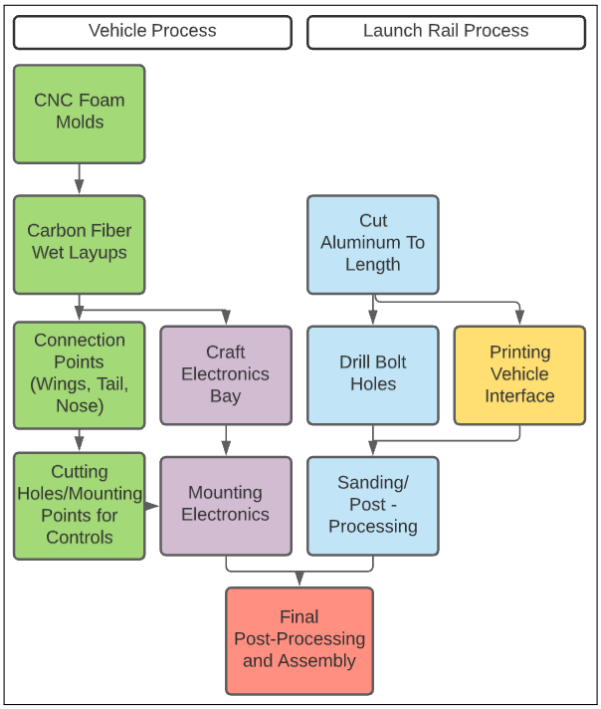

Fig. 5. Manufacturing Process Flow Chart. Green processes represent those working with carbon fiber. Purple processes represent the electronics and thir interfaces. Blue processes represent aluminum cutting and drilling. Yellow represents 3D printed processes. Red represents the combination of all. The OpenUAS (left) and its launch rail (right) represent parallel processes.

We chose a bungee cord as the propulsion mechanism: this allows for manipulation of the force exerted by controlling the length of the cord and the number of lengths used to launch the craft. Additionally, we 3D printed a craft carrier. 3D printing permitted the carrier to be manufactured in one piece with complicated geometry. This process is avoided for parts on the OpenUAS because the final pieces produced with PLA (polyactic acid) filament are too dense and heavy for the performance of the material provided.

B. Manufacturing Process

The high-level manufacturing process of the OpenUAS appears in Figure 5. We start by creating male molds that are used to create the carbon fiber exterior. The parts are created using a CNC machine with XPS pink foam material. Each component is made using a wet carbon fiber layup process and cured for 24 hours within a vacuum bag to apply pressure. Once cured, each part is sanded using a basic palm sander and trimmed using a dremel to fit the final specs of the design. The nose cone and the body require more extensive post-processing since the foam mold needs to be chipped out of the interiors (using hand tools such as a flathead screwdriver and X-Acto knife) to make space for the electronic components used to control the vehicle. We do not remove the foam mold from any other parts, such as the wings and tail struts, due to the added stiffness and low weight of the foam.

We then construct the electronics bay and the connection points between the different sections of the vehicle. The pieces of the electronics bay are cut out of posterboard with an X-acto knife and epoxied together using five-minute epoxy. The electronics bay is then glued to the nose cone to make them into one piece/assembly. Bolt holes are drilled into the body, wings, and nose cone. Nuts are then epoxied on the back side because threads can not be made carbon fiber or fiberglass. The tail boom is mounted into the body using an extra piece of XPS foam and five-minute epoxy. Small holes are then drilled into the tail boom to hold the support bars for the tail struts, and the tail struts are mounted once this was completed.

The basic design of the launch rail was to mirror a large crossbow that had been lifted above the ground. To do this, the launch rail system is made out of cost-effective aluminum extruded stock and designed to be cut, drilled, and assembled as easily as possible.

Manufacturing the rail requires only a low-torque hand drill, a Dremel, and a basic hacksaw. We start by cutting the launch rail down to size using the hacksaw then we use a dremel for smaller parts that must be finished or cut in more intricate shapes than a hacksaw allows. The bolt holes for the launch rail are then drilled out. We mark these holes by indenting their centers, drilling a spot hole with an $\frac{1}{8}$ inch drill bit, and then drilling the final $\frac{1}{2}$ inch hole. To ensure proper alignment of these bolt holes, not all are drilled at first and instead are drilled once the rail system was partially assembled and the other holes could line up over them. The launch rail is then finished using a hand file to ensure there are no sharp edges or rough patches.

Another important piece of the launch rail is the interface between the rail itself and the OpenUAS. This piece (the "Launch Carrier") was printed out of PLA plastic filament on a LulzBot Taz 6. The design is made to be light as possible, modular, and able to fit the OpenUAS in its basket. We partitioned the design into six subsections due to the print

space of the LulzBot and the size of this component. These pieces are then glued together using JB Weld five-minute epoxy. The launch carrier is interfaced into the launch rail system using 1010 rail buttons that are typically used in the model rocketry community, which allows for a smooth and reliable interface that keeps the launch carrier on track until the OpenUAS takes off.

Once these tasks are completed, the electronics can be mounted. Holes are cut to allow wiring and control points to exit the body and to mount servos into place. Final assembly and testing commences once all electronics are in place.

C. Manufacturing Lessons Learned

As the manufacturing process of the current OpenUAS progressed, the team noticed a few difficulties that came with the use of the carbon fiber. The process of shaping carbon fiber in the lab was difficult and outweighed the benefits of the materials strength and durability. One major issue was the difficulty of preventing the material from wrinkling while the epoxy was curing in the vacuum bag. The initial solution was to sand the material down to form a smooth surface after the epoxy had completely cured. This, however, resulted in visible weak spots that were located along the edges of the craft and on a few of the flat surfaces. Further, access to precision tools needed to create access points and mounting holes was an issue. With limited space for proper clamping and a low-torque hand drill, issues arose with mounting holes having to be redrilled due to them being angled.

Initially, the electronics bay was designed to be manufactured out of fiberglass to ensure the weight of the battery was supported. It was determined that the part needed to be reconstructed out of a lighter material (poster board), however, as the fiberglass was heavier than expected and could not be easily modified once assembled. The cross sectional strength was not as impactful as initially anticipated as the battery did not require any extra support aside from the corner supports that were later put into place.

After our first test flight, we discovered an issue with the mounting of the ailerons on the wings: the hinges would pop off the wings of the OpenUAS during transportation after a flight. This was fixed by using epoxy to reconnect the hinges and the ailerons to the crafts wing. For future designs of the OpenUAS, we will include a method to incorporate a stronger connection of the hinges and remove the need for epoxy.

With the current design of the OpenUAS, we have focused on the use of the launch rail for take-off instead of a hand-launching because it gives a more reliable take-off speed and orientation. A minor issue that was discovered with the design of the launch rail was how unsteady the rail was during launching. To resolve this issue we will distance the rails further apart from each other in the future designs.

IV. ELECTRONICS AND SOFTWARE

In parallel with the external design and manufacturing process, we focus on designing and testing the iron bird: a completely connected collection of all electronics required to fly the OpenUAS, all housed outside of the aircraft for easy access. Some of the core features of the iron bird is its customizability, ease of assembly, and seamless implementation into the airframe. As such, the components listed below are intended to be used as a reference, where the end user could easily add, remove, or swap components should they see fit.

A. Electronic Hardware

We list the currently supported electrical components in the OpenUAS in Table IV. Some components come in bundles and are denoted by a * or ** to indicate that they can be purchased for less money in a bundle together. These components are the bare minimum to implement an autopilot into the OpenUAS and exclude options such as airspeed sensors and LIDARs, which we still support.

Component	**Model**	**Price**
Flight Computer	Holybro Pixhawk 4 Autopilot	$170 (SKU20068)*
GPS	Holybro Pixhawk 4 Neo-M8N GPS	$170 (SKU20068)*
Power Module	Holybro APM Power Module 12S - PM02 V3	$170 (SKU20068)*
RC Transceiver	FrSky X8R 8/16Ch S.Bus ACCST Telemetry Receiver W/Smart Port	$244.99 (SKU2153)**
RC controller	FrSky Taranis X9D plus	$244.99 (SKU2153)**
Telemetry Radios	Holybro 433Mhz 915Mhz Transceiver Radio Telemetry Set	$39.00
ESC	BadAss Renegade 85A ESC	$79.99
Motor	BadAss 2826-690Kv brushless motor	$69.99
Propeller	APC 14x12 E	$9.05
Battery	Ovonic 5000mAh 11.1V 3S 50C 3 Cell LiPo	$32.99
	TOTAL:	**$613.02**

TABLE IV
IRON BIRD OPENUAS COMPONENTS LIST

We selected the Pixhawk 4 for the flight computer. The flight controller is designed to run the PX4 flight stack [17], an open source flight software that is highly configurable, customizable, and has built-in autonomous flight capabilities. This was chosen over other offerings, such as ArduPilot, due to its documentation and sustained support for the chosen Pixhawk flight computer. An end user could decide to use another flight computer, or none at all, if it better fits their specifications, however.

Components including the GPS, airspeed sensor, power module, and Telemetry Radios where chosen from a list of recommended components off of the PX4 setup guide [18] and interface directly with the Pixhawk 4. These components all have plug-and-play capabilities, which allows for easy and simple electronics setup using the ground station.

The RC transceiver connects with the supported Taranis X9D Plus controller and also directly with the Pixhawk 4, which minimizes setup of the RC controller, further reducing setup complexity.

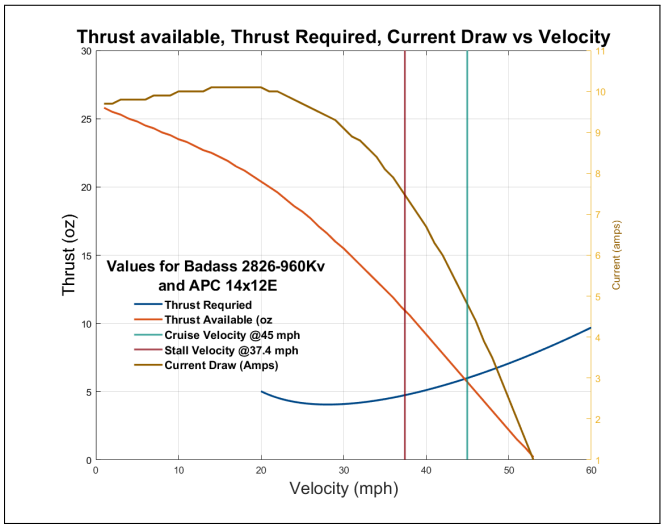

Fig. 6. Theoretical Thrust Required Curve Plotted with Theoretical Thrust Available (left axis) and Current draw (right axis). The throttle setting on Motocalc was lowered to 54% such that the thrust available curve, seen in black, intersects with the thrust required curve, in blue, at the selected cruise velocity of 45 mph. The current draw corresponding to flying at 45 mph can be seen where the cruise velocity intersects the current draw curve. They intersect at approximately 4.4 amps

1) Motor and Propeller selection: To determine a motor and propeller combination, and subsequently what ESC should be used for the OpenUAS, analysis was done using data from both CFD and MotoCalc 8 [19]. The parasitic drag coefficient $C_{D,0}$ was found using data from CFD analysis run on the OpenUAS model and found the thrust required curve as shown in [20, p. 212] as equation 1

$$Thrust = D = \frac{1}{2}\rho_\infty V_\infty^2 S_{ref} C_{D_0} + \frac{2KW^2}{\rho_\infty V_\infty^2 S_{ref}} \quad (1)$$

The maximum lift coefficient $C_{L_{max}}$ was found using CFD, and was used to compute the stall velocity using the equation 2 below [20, p. 316].

$$V_{stall} = \sqrt{\frac{2W}{\rho_\infty S_{ref} C_{L_{max}}}} \quad (2)$$

This came out to be 37 mph, however, subsequent stall testing written about in V found this number to be around 26 mph. Using the thrust required curve, we calculated the max range cruise velocity as the velocity at the minimum of the thrust required curve [20], as 27 mph, which was lower than the theoretical stall velocity. Our pilot selected a cruise velocity of 45 mph to be sufficiently above the stall velocity to compare current draw of motor and prop combinations.

We generated theoretical values for current draw, efficiency, and thrust available at varying velocities for different combinations of motors and propellers using MotoCalc 8. Thrust available, Current Draw, and Efficiency where plotted with the thrust required curve to determine the most efficient combination of motor and prop at our selected cruise speed.

Initially, the motor and prop combination with the most excess thrust was selected over other valid combinations with higher efficiencies due to the higher theoretical stall velocity. This was a BadAss 3520-970Kv and the APC 15x8E propeller. At 0 mph MotoCalc determined the maximum current draw to be 70 amps, and the current draw in steady state flight at 7 amps so the BadAss Renegade 85A ESC was therefore chosen due to its 85 amp continuous current draw and 100 amp burst current. After an initial test flight, it was determined that the the stall velocity was much lower than the CFD indicated, 26 mph vs 37 mph, so the BadAss 2826-690Kv motor and the ACP 14x12E prop where chosen as this combination only drew up to 36 amps, still provided sufficient thrust, and was overall more efficient. In the future we plan to re-run this analysis at values closer to the theoretical cruise speed of 27 mph to have more accurate current draw data available. We also plan to make this data available in an easily accessible format such that a motor and prop can be selected depending on the payload an OpenUAS user has.

2) Batteries: During preliminary flight test analysis, our main challenge was balancing the weight and max thrust of the aircraft, notably during takeoff. We specifically selected the battery to maintain the max thrust while decreasing weight. First, the battery has to be able to output enough current to sustain the system at full throttle. Batteries for remote-controlled cars or UAS are generally able to achieve this. Next, the battery has to have a voltage that is compatible with the electrical system; we chose 11.1V. With these requirements, we can select the amp-hours of the battery to decrease the weight of the battery with the drawback being that lower amp-hours correlate to shorter flight time. A battery with a different value can then be selected depending on the planned mission to allow the OpenUAS to be more general purpose.

B. Software

We are running PX4 Autopilot [17] on the flight computer for the OpenUAS. We choose PX4 over other software stacks such as ArduPilot due to the well maintained documentation, high customizability, and ongoing support for the Pixhawk hardware. PX4 allows the OpenUAS to follow autonomous flight paths, assist the pilot in maintaining stable flight, and can collect and record data during tests. If the end user does not require these capabilities, it can be optionally excluded.

1) Simulation: We are currently experimenting with computer simulated flight environments. The latest version of PX4 allows for fully simulated flights using Gazebo [21] to simulate the world that the software interacts with. QGroundControl [22], the ground station software that we utilize to monitor the vehicle in flight, connects to a version of the PX4 software running locally on a lab computer. From there, PX4 communicates with Gazebo, also running locally on a lab computer, to interact with the simulated world.

Our analysis includes basic flight simulations of manual and autonomous flight modes. These simulations give us a general idea of how the real UAS will behave without fear of crashing and allow us to introduce variables during flight to observe what actions the UAS will take to correct them. As can be seen in figure 7 we are currently using a default fixed-wing airframe, provided by the existing PX4 software, for

Fig. 7. Default PX4 plane simulated inside Gazebo [21].

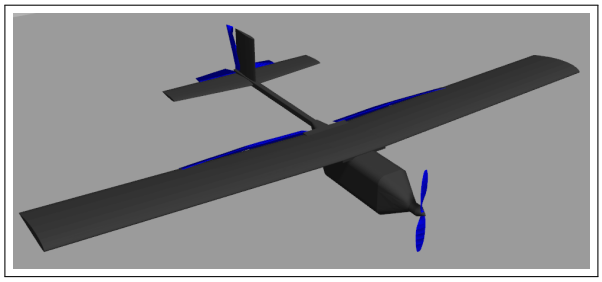

Fig. 8. Work in progress vehicle model of the OpenUAS frame

the simulations. However, we plan to add our own OpenUAS design, seen in figure 8, to the simulator to more accurately simulate the flight of our aircraft. This system will allow us to train new pilots and test new software features in a safe, cost-free environment.

We also explored hardware in the loop (HITL) simulation. QGroundControl connects to the Pixhawk hardware over radio and the Pixhawk sends commands and receives results from a simulated world running on a lab computer. While this integrates the flight hardware into the testing structure, HITL simulation is no longer supported by the PX4 software for fixed-wing aircraft. We ultimately determined that HITL simulation does not provide a significant advantage over a standard full simulation composition for the amount of additional work required due to the dropped support.

2) Custom Firmware and GitHub Repository: The OpenUAS team also maintains a version of the PX4 firmware. This version includes a new software airframe for the OpenUAS vehicle, allowing us to directly program the vehicle configurations into the source code. The team utilizes GitHub to keep track of the changes made and to share the custom version with others. The GitHub [3]repository is a fork from the stable version of the PX4 software and is set to automatically create a pull request when a new version of the stable PX4 software is released.

Additionally, the team forked the [4]submodule that is responsible for the flight simulation. The vehicle model and its configurations are stored within this submodule. To facilitate autonomous integration, this submodule automatically merges changes from the upstream source when they are available. This workflow allows the team to keep track of its modifications, while staying up to date with the latest version of the official PX4 firmware.

Within the custom airframe, the team has configured an output mixer specific to the OpenUAS's structural design, enabled an additional LIDAR sensor for altitude measurements, and tuned the parameters for autonomous maneuvers including autonomous takeoff and landing.

The end goal is to integrate the custom airframe into the official version of PX4 or to provide a pre-compiled version of the custom firmware for others to easily load onto the Pixhawk flight computer when building the OpenUAS.

V. FLIGHT TEST AND EVALUATION

A. Flight Procedures

We created test procedure documents for both ground and flight tests to specify the objectives, success criteria, supplies, and procedures for a successful test in addition to serving as a place to record data and lessons learned. These documents allow the team to write out a detailed plan for each test before it occurs to make sure no details are overlooked when the test is being performed. An electronics preflight checks and arming procedures document was also created to specify the specific procedures to follow to correctly calibrate and arm the iron bird and ensure the OpenUAS is ready for flight.

B. Test Flights

Fig. 9. OpenUAS Group members (left to right) Glick, Edgren, Burken, Levandowski, and Johannsen at Flight Test Range

The first flight of the OpenUAS occurred on November 22nd, 2020 at the Central Iowa Aeromodelers flying field southeast of Ames, Iowa; see Figure 9. Weather conditions at the time were unlimited visibility and clear skies, and the wind was out of the west at a gusty 7-10 mph.

A hand launch technique was employed for the takeoff since landing gear had not yet been incorporated into the OpenUAS design and the launch rail was found to be too unstable to properly launch the OpenUAS. A team member held the OpenUAS above their head while the motor was ran up to full power. The team member ran forward with the OpenUAS and a slight forward push as he let go of it was sufficient for the OpenUAS to quickly gain flying airspeed with little to no dip in flight path after release.

[3]https://github.com/LTL-AERO/PX4-Autopilot/tree/stable
[4]https://github.com/LTL-AERO/PX4-SITL_gazebo

Fig. 10. OpenUAS climbing to cruise altitude shortly after hand-launch 10 ft off the Ground

With the motor and prop combination optimized for thrust the OpenUAS climbed very well, see Figure 10, and was quickly at a comfortable cruise flight altitude.

Once in the air, the OpenUAS was not as stable as desired and demonstrated nearly neutral static stability about its lateral and longitudinal axis. However, this is no less stable than many aerobatic radio-controlled aircraft so the OpenUAS was still well within a reasonable margin of stability and safety. The OpenUAS also tended to hunt left and right in yaw and constantly required small aileron and elevator inputs to keep it on a straight flight path. The amount that these characteristics were affected or caused by the wind conditions during the test flight is currently undetermined and will be explored in future test flights with more ideal winds. The half-span ailerons on the OpenUAS were extremely effective, as was the elevator. The rudder was not as effective as desired but could still create small yawing moments.

The OpenUAS cruised comfortably at seventy-five percent power. Engaging Stabilize mode on the Pixhawk 4 effectively eliminated the stability problems and made the OpenUAS almost as easy to fly as an E-flite Apprentice radio-controlled student trainer aircraft [6].

Two subsequent flights of the OpenUAS have tested the addition of landing gear, a larger rudder, and a different wing design. The landing gear allowed the OpenUAS to takeoff and land from both hard surface and well-cut grass runways and caused no adverse flight characteristics. The larger rudder allowed more yaw control at slower speeds and improved spin recoverability. The wing design had identical dimensions to the original wing, but had a NACA 23012 airfoil and was made out of XPS foam coated in a thin layer of epoxy. Flight with this wing caused no noticeable differences in flight characteristics, however the wing flexed considerably in flight, leading to the conclusion that its manufacturing method was not ideal.

Test flights of the OpenUAS have also focused on determining a stall speed and average cruise speed based on ground speed and airspeed data measured from a pitot tube. The PX4 also computes a windspeed estimate as well as factors compressibility effects to calculate a true airspeed, which is recorded into the flight logs. Stall tests were performed in Position Mode, which maintains the heading and altitude of the craft, while the throttle of the craft was decreased until a stall was noticed. In the data, we looked for a drop in altitude in while the craft was in position mode and read the corresponding true airspeed reading. These tests determined the stall speed of the OpenUAS to be approximately 12 m/s and the average cruise speed 23 m/s.

Initial test flights of the OpenUAS have proved the design to be airworthy and structurally sound, and initial modifications have already improved the design. Further test flights and data analysis will supply additional information on the OpenUAS, which will enable the team to make stability improvements and other refinements for OpenUAS 2.0.

C. Application of Runtime Verification on the OpenUAS

The OpenUAS is both a real-time and safety-critical system i.e., it does not have a high tolerance to a certain class of failures. Even with extensive design-time verification, we need to add runtime verification (verification performed in real time, during the flight) to meet our safety goals. We experimented with the Realizable, Responsive, Unobtrusive Unit (R2U2) [23], [24], a temporal logic-based runtime verification engine designed to embed on-board safety-critical systems to assess the health of the system according to our user-developed requirements.

To start, we set up R2U2 to reason over a small set of log files generated by PX4. This reasoning setup constitutes R2U2's "offline mode," where we prototype the setup over a trace file of logged data versus in "online mode" where data is fed into R2U2 running on-board the OpenUAS in real time. In the future, the team seeks to install R2U2 onto a Raspberry Pi or other companion computer and interface with the Pixhawk directly so that the data can be analyzed in real time and R2U2 can potentially send commands to the Pixhawk in response to any off-nominal events that R2U2 detects.

1) R2U2: R2U2 takes two inputs: a discrete stream/trace of data (e.g., sensor data) and a set of specifications formulated in Mission-time Linear Temporal Logic (MLTL) V-C.2. The tool uses novel algorithms [25] to check that the specifications are satisfied with respect to the input data and reports whether each specification has been satisfied in each time step in real time.

2) Developed Requirements: We developed a small set of formal requirements to reason over the available sensor data by informally writing requirements in English and then translating them into MLTL. To illustrate the work done so far, we focus on a single requirement inspired by a specification from a previous study [26], [27] and analyze the generated results based on the data captured during flight of the OpenUAS. In English, this requirement is:

"Once the UAS takes off, it should obtain an altitude of 300 meters within 15 seconds."

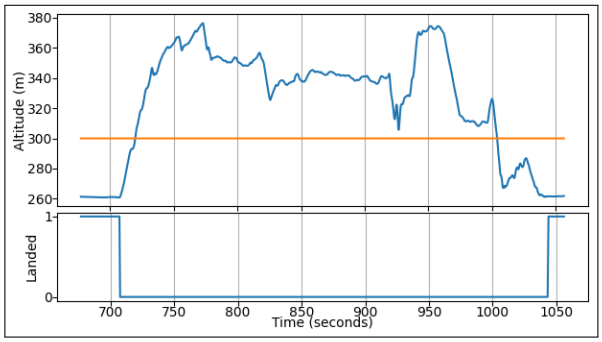

Fig. 11. Logged altitude and "landed" Boolean variable data

Mission-time Linear Temporal Logic (MLTL) [26], [28], [29], [30]:

⋆ finite set of atomic propositions {p q}
⋆ Boolean connectives: ¬, ∧, ∨, and →
⋆ temporal connectives *with interval time bounds*:

Symbol	Operator	Timeline
$\square_{[2,6]}p$	ALWAYS$_{[2,6]}$	
$\Diamond_{[0,7]}p$	EVENTUALLY$_{[0,7]}$	
$p\mathcal{U}_{[1,5]}q$	UNTIL$_{[1,5]}$	
$p\mathcal{R}_{[3,8]}q$	RELEASE$_{[3,8]}$	

There are three pieces of data we must have in order to check this requirement: takeoff status, altitude, and time. To keep track of the status of takeoff, we use logged "landed" Boolean calculated in the PX4 source code and approximate takeoff as when the value has changed from "1" to "0". For altitude we used GPS altitude since it is the most accurate method of all the on-board altitude sensors. Finally for time we used the associated timestamps with each data point. This logged data can be seen in 11.

The next step comes in formalizing this requirement in MLTL. We define takeoff as the state when the UAS is *landed* and in the next state the UAS is *not landed*. Formalizing this to MLTL we get:

$$landed \wedge \square_{[1,1]}(\neg landed) \tag{3}$$

As for the altitude section of our requirement, we say that the UAS must be at an altitude of 300 meters within 15 seconds. In MLTL:

$$\Diamond_{0,15} alt > 300 \tag{4}$$

The relationship between these two formulas can be expressed as a logical implication, or an "if then" statement. So our full specification can be described as:

$$(landed \wedge \square_{[1,1]}(\neg landed)) \rightarrow (\Diamond_{0,15} alt > 300) \tag{5}$$

Once we feed the data and outlined specifications into R2U2 we will get a single trace describing whether the specification was met at each timestep. We can plot traces for each

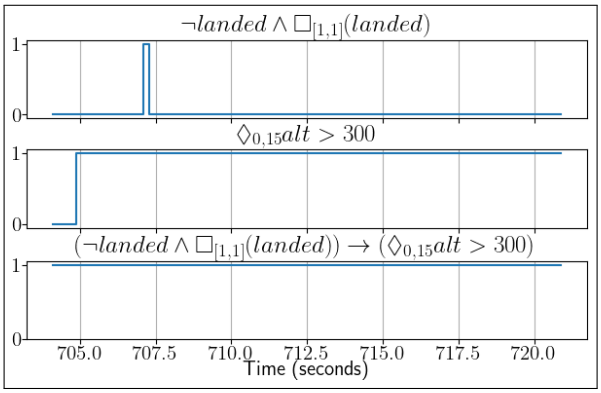

Fig. 12. Output generated by R2U2 based on data in Figure 11 for respective subformulas: 3, 4, 5. The outputs are boolean values with '1' and '0' corresponding to the specification being satisfied and unsatisfied respectively.

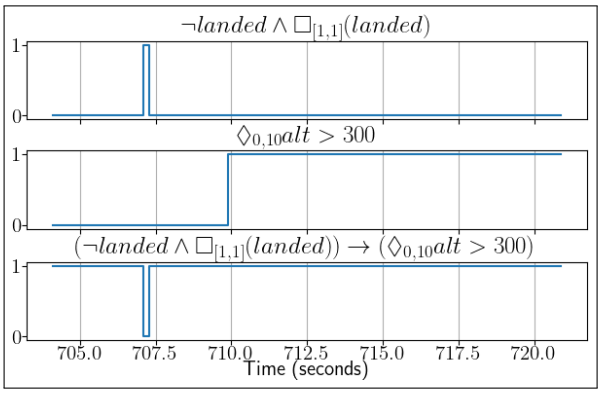

Fig. 13. Output generated by R2U2 based on data in Figure 11 for respective subformulas: 3, altered 4, 6. The outputs are boolean values with '1' and '0' corresponding to the specification being satisfied and unsatisfied respectively.

subformula as well as the overall formula for clarity in 12 and we will notice that for the time of interest (when we take off), the value of our specification is always '1,' meaning the OpenUAS behaved exactly as we expected.

As an exercise, we can change our specification such that we require an altitude of 300 meters within 10 seconds of takeoff. So our new overall specification will be:

$$(landed \wedge \square_{[1,1]}(\neg landed)) \rightarrow (\Diamond_{0,10} alt > 300) \tag{6}$$

Showing these graphs in figure 13, we see that the specification fails since we do not reach our desired altitude within 10 seconds of takeoff, and so the output of our new specification is '0' for the state when we takeoff, designating a failure to meet our stated requirement.

In future OpenUAS flights, we intend to connect the output verdicts of R2U2 to mitigation triggers, e.g., to help automatically cushion an unplanned landing instead of the pilot needing to observe an unrecoverable spin, and manually apply the opposite aileron to the direction of spin. Runtime verification specification patterns collected from previous case studies [31] will guide our requirements elicitation effort.

VI. CONCLUSION AND FUTURE WORK

We have created, documented, and completed initial flight demonstrations for the first totally open-source (in both soft-

ware and hardware), easily manufacturable, small, inexpensive, reconfigurable, fixed-wing UAS. Our initial designs are currently available for public use, now enabling users from researchers to students to conduct repeated flight experiments with varying payloads on an easy-to-repair testbed. As we add more polished documentation, user scenario guides, and trade-off analysis, fixed-wing UAS experiments will become increasingly accessible.

Our immediate next steps include improving take-off and landing to make both smoother sequences with less variance and lower probability of any damage to the aircraft, as well as improving aspects of aircraft control. In the future, in addition to aiming for general improvements to the design, we intend to create multiple versions of the OpenUAS focusing on different user needs. For example, one version will be the cheapest, easiest-to-manufacture version optimized for maximum accessibility to a wide range of student clubs, high schools, and hobbyists. Another version will optimize for maximum reconfigurability for different sized batteries, onboard sensors, and flight computers, to enable experiments with a broad range of autonomous capabilities. We plan to experiment with both smaller, catapult-launched and larger, runway-launched versions to enable different payloads for a broad range of experiments. After we create a core set of standard OpenUAS variant designs, we will focus on specializing the platform for supporting specific experiments in the Laboratory for Temporal Logic, such as gathering data from runtime verification of hybrid-UAS swarms and advanced algorithms for autonomous operations.

Our ultimate goal is to have a well-organized, well-documented github repository with various versions of the OpenUAS, each optimized for a different audience. As each design choice will be accompanied by documentation describing how it was made and what effect slight changes would have, users will be able to start with one OpenUAS version and further specialize the open design to their needs. This also means that if any specific COTS part is not available, it will be easy to figure out which other part(s) can be suitably substituted and if any other changes need to be made to other parts of the aircraft to accommodate the substitution. The OpenUAS project aims to make fixed-wing UAS flight testing more accessible (in terms of understanding, cost, and physical access) to broad audiences going forward.

References

[1] E. Denney and G. Pai, "Data Artifacts for Airworthiness of the Swift UAS," https://ti.arc.nasa.gov/publications/5660/download/, 2012, NASA Ames Research Center.

[2] J. Geist, K. Y. Rozier, and J. Schumann, "Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems," in *Runtime Verification (RV14)*, vol. 8734. Springer-Verlag, 2014, pp. 215–230.

[3] K. Y. Rozier, J. Schumann, and C. Ippolito, "Intelligent Hardware-Enabled Sensor and Software Safety and Health Management for Autonomous UAS," NASA, NASA Ames Research Center, Moffett Field, CA 94035, USA, Technical Memorandum NASA/TM-2015-218817, May 2015.

[4] S. N. Air and S. Museum, "AeroVironment RQ-14A Dragon Eye," Online: https://airandspace.si.edu/collection-objects/aerovironment-rq-14a-dragon-eye/nasm_A20070211000, 2003.

[5] N. C. Administrator, "NASA Flies Dragon Eye Unmanned Aircraft Into Volcanic Plume," Online: https://www.nasa.gov/topics/earth/earthmonth/volcanic-plume-uavs.html, 2013, NASA Ames Research Center.

[6] E-flite, "Apprentice STS 1.5m RTF Smart Trainer with SAFE," Online: https://www.horizonhobby.com/product/apprentice-sts-1.5m-rtf-smart-trainer-with-safe/EFL3700.html, 2020, EFL3700.

[7] Applied Aeronautics, "Albatross," Online: https://www.appliedaeronautics.com/albatross-uav, 2014.

[8] E-flite, "E-flite RC Product/Services," Online: https://www.facebook.com/EfliteRC/, 2021.

[9] J. Spitzer, "3DLabPrint," Online: https://3dlabprint.com/, 2021.

[10] M. Selig, "Uiuc airfoil coordinates database," Online: https://m-selig.ae.illinois.edu/ads/coord_database.html.

[11] A. Deperrois, "XFLR5." [Online]. Available: http://www.xflr5.tech/xflr5.htm

[12] N. K. Hieu and H. T. Loc, "Airfoil selection for fixed wing of small unmanned aerial vehicles," in *AETA 2015: Recent Advances in Electrical Engineering and Related Sciences*. Cham: Springer International Publishing, 2016, pp. 881–890.

[13] A. Gries, "Airfoil selection - iteration 2," Online: http://temporallogic.org/research/ICUAS21/docs/airfoilselection_iteration2.pdf, January 2020.

[14] ——, "Tail sizing," Online: http://temporallogic.org/research/ICUAS21/docs/tailsizing.pdf, September 2018.

[15] M. Drela and H. Youngren, "AVL." [Online]. Available: http://web.mit.edu/drela/Public/web/avl/

[16] S. D. I. Software, "Star CCM+." [Online]. Available: https://www.plm.automation.siemens.com/global/en/products/simcenter/STAR-CCM.html

[17] I. Dronecode Project, "PX4 autopilot." [Online]. Available: https://px4.io/

[18] ——, "PX4 user guide." [Online]. Available: https://docs.px4.io/master/en/

[19] I. Capable Computing, "Motocalc." [Online]. Available: http://www.motocalc.com/

[20] J. Anderson, *Aircraft Performance & Design*, ser. McGraw-Hill international editions. McGraw-Hill Education, 1999. [Online]. Available: https://books.google.com/books?id=PwtO7aiwbBwC

[21] O. S. R. Foundation, "Gazebo." [Online]. Available: http://gazebosim.org/

[22] I. Dronecode Project, "QGroundControl." [Online]. Available: http://qgroundcontrol.com/

[23] K. Y. Rozier and J. Schumann, "R2U2: Tool Overview," in *Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools (RV-CUBES)*, vol. 3. Seattle, WA, USA: Kalpa Publications, September 2017, pp. 138–156.

[24] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "Runtime Analysis with R2U2: A Tool Exhibition Report," in *Runtime Verification*. Madrid, Spain: Springer-Verlag, September 2016.

[25] B. Kempa, P. Zhang, P. H. Jones, J. Zambreno, and K. Y. Rozier, "Embedding Online Runtime Verification for Fault Disambiguation on Robonaut2," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*. Springer, September 2020, pp. 196–214.

[26] T. Reinbacher, K. Y. Rozier, and J. Schumann, "Temporal-logic based runtime observer pairs for system health management of real-time systems," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 8413. Springer-Verlag, April 2014, pp. 357–372.

[27] J. Schumann, P. Moosbrugger, and K. Y. Rozier, "R2U2: Monitoring and Diagnosis of Security Threats for Unmanned Aerial Systems," in *Conference on Runtime Verification (RV15)*. Vienna, Austria: Springer-Verlag, September 2015.

[28] J. Li and K. Y. Rozier, "MLTL Benchmark Generation via Formula Progression," in *Runtime Verification*. Limassol, Cyprus: Springer-Verlag, November 2018.

[29] "2019 Runtime Verification Benchmark Competition," https://www.rv-competition.org/.

[30] J. Li, M. Y. Vardi, and K. Y. Rozier, "Satisfiability checking for Mission-time LTL," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 11562. New York, NY, USA: Springer, July 2019, pp. 3–22.

[31] K. Y. Rozier, "Specification: The biggest bottleneck in formal methods and autonomy," in *Verified Software: Theories, Tools, and Experiments (VSTTE)*, ser. LNCS, vol. 9971. Toronto, ON, Canada: Springer-Verlag, July 2016, pp. 1–19.