UAS Flight Testing in Support of Research for Academia: Lessons and Experiences from the Field

Matthew Anderson*, Kai Lehmkuehler†, Jeremy Randle‡, K.C. Wong§, and Soon-Jo Chung¶
California Institute of Technology, Pasadena, CA, 91125
The University of Sydney, NSW, 2006, Australia

Uninhabited Aerial Vehicles (UAVs) and miniaturized high-performance computing have enabled flight testing to become incredibly accessible, changing the status quo from an expensive, high-knowledge-barrier endeavour to a relatively low-cost exercise that is within the reach of small-scale research institutions and individuals. Due to this ease of entry, flight testing with UAVs is becoming increasingly commonplace as the technology allows cutting-edge research to leave the realm of simulation and enter real-world trials in very short time frames without the restrictions and costs of piloted, full-scale flight. Where traditional flight test engineers required many years of specialized training, budding UAV flight test engineers often start with little-to-no prior personal or in-house experience, and go through the same trail-and-error processes as those before them. This paper aims to document many years of experience flight testing at a university level, both to provide a basic understanding of the process for anyone getting started, and to share ideas with more experienced operators. It covers aspects such as instrumentation, flight controllers, airframes, bridging the sim-to-real gap and methodologies for conducting safe and efficient flight test campaigns.

I. Introduction

Uninhabited Aerial Vehicles (UAVs) have caused a paradigm shift in the development and testing of airborne technologies. Previously, testing these technologies often required complex and expensive test campaigns involving traditional aircraft and all the regulatory, training, safety, and logistical constraints that accompany human flight. While UAV flights are still bound by these classes of constraints, the smaller size, lower cost, and lower risk factors mean testing new payloads, algorithms and aircraft is now well within the reach of even low-budget, small-scale research institutions and individuals.

Fig. 1 A collection of the wide range of aircraft tested in a single day (August 2014). Among these tests were air-to-air docking [1], deployment of a UAV from a quadrotor [2], system identification of a Piper [3], on-board visual positioning for a multirotor, and tracking Swift Parrots.

*Staff Scientist in Aerospace, Caltech, 1200 E California Blvd, CA
†Lilium, Munich, Germany. Previously PhD Student and Post-Doc, School of AMME, The University of Sydney, NSW, 2006, Australia
‡Senior Technical Officer, Australian Centre for Field Robotics, The University of Sydney, NSW, 2006, Australia
§Associate Professor, School of AMME, The University of Sydney, NSW, 2006, Australia, AIAA Associate Fellow
¶Bren Professor of Aerospace and Control and Dynamical Systems (Caltech) and Jet Propulsion Laboratory Senior Research Scientist; 1200 E California Blvd, CA, AIAA Associate Fellow

University-level research and education has benefited greatly from the wide accessibility and ease of use of UAVs. Concepts such as aircraft design, control, handling, flight planning and operations all lend themselves well to being taught at a small UAV scale, enabling students to design, build, program and fly real aircraft with minimal risk and cost. The use of multirotors to test the 'latest and greatest' flight control algorithms is becoming almost ubiquitous, and the ever-increasing onboard compute power available means computationally expensive tasks like computer vision can now easily be run in real time. Outside of the aeronautics domain, the ability to easily strap a sensor payload to a commercial-off-the-shelf (COTS) aircraft has enabled scientists to extend their data collection campaigns to new heights and capture data from previously inaccessible locations and angles.

This level of wide accessibility and ease of use of UAVs has however introduced its own set of potential risks, challenges, and pitfalls for researchers. Whereas previously piloting aircraft and conducting flight test campaigns took years of training, anyone can now easily buy an UAV and go flying. While the operation of the UAV may seems simple to the user, it still remains an incredibly complex piece of equipment that has operational limitations, yet there are no real training requirements to ensure the operator understands these. In cases where a custom solution is required, the researcher not only has to develop the software they wish to test, but has to become an expert in the design, manufacture, assembly, maintenance, piloting, tuning, and operations of their aircraft. Any project involving flight depends on the many layers of the uninhabited aerial system (UAS), from the airframe, sensors, flight controller, compute package, ground station, pilot and operations, to all work together. At best, faults in these layers can prevent tests from being carried out or foul the data sufficiently that it is unusable. At worst, faults can potentially cause a catastrophic system failure resulting in not only a loss of data, but can set a project back many months as equipment is repaired or replaced.

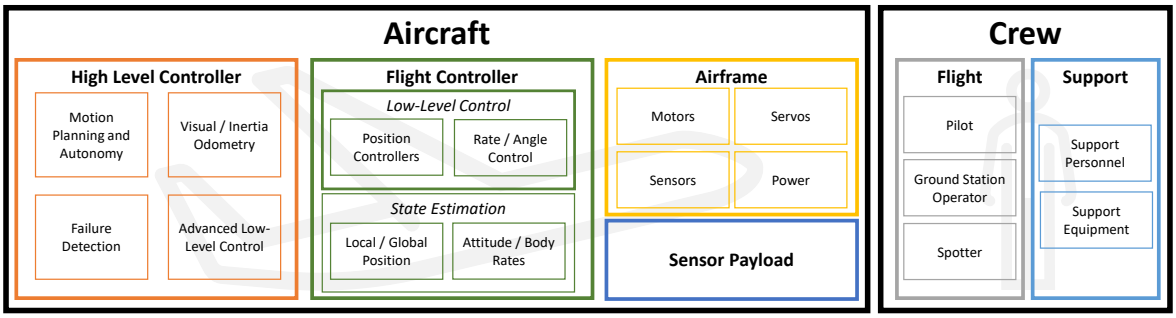

Fig. 2 UASs are complex systems with many intertwining elements. This diagram shows some of the major components of a typical UAS, all of which must work together for a successful mission that delivers meaningful data.

This paper aims to document many years of experience designing, supporting and piloting UAVs for research. While there are many previous papers and books that discuss flight testing UAVs (such as [4–6]), they typically focus more on the operations of established research institutions with existing capabilities, rather than testing cutting-edge technologies in a university environment. A casual observer may relate UAVs to hobbyist RC (radio controlled) models, but the key difference is that UAVs have integrated flight controllers with various levels of automation/autonomy and are used as airborne instruments or as payload platforms for a broad range of missions in research, commercial and/or defence industries.

Section II will discuss the different types of uses for UAVs in research and how this affects decisions such as what type of instrumentation, flight controller, payload, airframe and test campaigns to use. Section III will also present methods of how to move from simulation to lab to real-world testing efficiently, including important aspects such as planning, safety and many little 'gotchas' that may happen along the way. Finally, Section IV will consider ways to use UAVs as a tool for education and how to incorporate them into a university-level syllabus. While this paper does not claim to contain the only (or necessarily the best) way to do things, it does present many concepts, ideas and techniques that are typically unwritten and are instead passed down from one generation of researcher to the next. It is intended to be used as both a starting point for young researchers with little to no prior in-group UAV experience, and as a knowledge-sharing exercise for more experienced UAV researchers and operators who are looking to build upon the experience of others to improve their own operations.

II. UAS Flight Project Design

UASs are a complex set of technologies, with many intertwining elements that must all work together (Figure 2). No amount of cutting edge control can make a poorly designed aircraft fly well, no state estimator can 'fix' improperly installed and uncalibrated sensors, and no compute package can operate without a power system to keep everything going.

Different types of research require different capabilities, however quite often, many of these capabilities can be best met with COTS solutions, saving significant amounts of time and money which allows efforts to be focused on the actual research for the project. It is important to apply a ground up approach to the design of the UAS as all-too-often researchers have a pre-conceived idea of what test they want to perform, with little thought as to whether is the best test to perform. This section details the considerations to take into account when selecting each of the elements in the UAS, ordered approximately such that minimal iteration should be required.

A. Different Types of Flight Testing

Research projects involving UAVs typically fall under one of four general categories - performance analysis, system identification, controls testing, and payload flights. Each of these different types of projects requires different levels of sensing, flight controller capability, and onboard computational power, and hence the research question plays a significant role in the the whole system design. Furthermore, the level of autonomy designed into the system needs to be carefully considered [7], further affecting the system design choices.

1. Performance Analysis

Performance analysis requires a good knowledge of the overall aircraft state (such as angles of attack and sideslip, attitude, altitude, control-position, throttle commands and battery monitoring), quantities which are often readily available from (and logged by) COTS flight controllers. In general, the required maneuvers are simple (glide tests, straight and level powered flights), meaning little additional compute (if any) is required to perform them.

2. System Identification

System identification, essential for the optimisation and robustness of flight control systems, requires a good knowledge of the overall aircraft state, however, timing plays a key role and additional care must be taken to synchronise (or at least timestamp) the data as accurately as possible. Direct feedback such as propeller RPM and control surface deflections are also crucial as these provide a direct measurement of the control inputs into the system and help eliminate uncertainties which can be caused by variable loads due to different air speeds and atmospheric conditions. Furthermore, system identification usually requires specific control inputs to correctly excite the modes of the aircraft and specialised (albeit computationally inexpensive) software to be able to repeatably and accurately reproduce control inputs.

3. Control Algorithm Testing

Control algorithm testing (and more broadly state estimation, motion planning and computer vision tasks) may not require as high-fidelity state estimates as performance analysis and system identification, however often require significantly more computing power to run the algorithms and additional sensors (such as cameras). This can very quickly require a significant payload fraction of small aircraft as not just the computer, but the sensors, wires, voltage regulators and connectors all quickly add up in both weight and volume.

4. Payload Flights

Finally, payload flights are where a sensor package generates the primary data of interest, and the UAV is merely a means by which to get the sensor package to the desired location. Often, the payload is completely self-contained in terms of power, logging and control, and only needs to be mounted to the UAV with little-to-no interfacing between the two. Examples of this type of testing include carrying custom science payloads (such as air quality sensors), or where the UAV itself already contains the sensor capability (such as a COTS UAV + camera system).

B. System Instrumentation

Instrumentation is used to sense the state of the aircraft and the wider environment, thus correctly instrumenting the aircraft is key to successfully obtaining useful data from any form of testing. Flight controllers require a minimum level of sensing in order to be able to stabilise an aircraft, however this may need to be augmented with additional sensors in order to be able gain an appropriately rich knowledge of the aircraft's state for the task at hand. Consideration should also be taken into account for states that make development easier (such as battery voltage monitoring) as this will help reduce uncertainty during testing and increase the chances of success.

Where possible, states should be directly measured rather than inferred from other measurements. While this may seem intuitive, it is often disregarded. As an example, when measuring control deflections, one may simply statically map the servo command to an output angle, rather than directly measuring the angle of the control surface. This static mapping introduces significant errors from sources such as control delays, servo dynamics and surface loading, all of which would be captured by directly measuring the control surface deflection. Similarly, while RPM measurements may be inferred from a static mapping of the control input, motor dynamics, airspeed, battery voltage, and the atmosphere all affect the actual RPM.

Fig. 3 Various payloads flown with sUAS in support of research. These include radar systems for tomography, gas samplers, cameras and IMUs for VIO, and external compute packages for AI/ML control.

Synchronisation of the sensor streams is another aspect which is often not taken into account. Aircraft are dynamic vehicles that are constantly changing (Figure 4), meaning unless data is captured by all the sensors at exactly the same time, each data stream represents the aircraft at a slightly different state making accurate state reconstruction difficult. This can somewhat be corrected during post-processing assuming the data is timestamped well, however this requires significantly higher data rates than if the data was captured synchronously.

Correctly synchronising sensors is difficult, and hence the task at hand should also be considered. As an example, if performance testing is being conducted, long-term averages may be of interest, meaning the synchronisation requirements may be able to be relaxed as the data is likely to be averaged anyway. At the other end of the spectrum, for visual inertial odometry (VIO) which uses pixel projections into the world, even small discrepancies between when the image was taken and when the angle was measured can lead to large errors, necessitating the need for hardware synchronisation.

Finally, adding multiple sensors that measure the same quantity can help reduce uncertainty when processing data. While this must be weighed up against size, weight and power (SWaP) constraints, the extra sensing can help determine when a sensor is not working correctly, provide extra information into filters, and help uncover anomalies with the aircraft. Additionally, oftentimes with sensors such as 9-DoF IMUs (accelerometers + gyros + magnetometers), the optimal location for one of the measurements may not be the optimal location for another. In these cases, multiple sensors of the same kind can allow the best data stream to be chosen for flight path reconstruction.

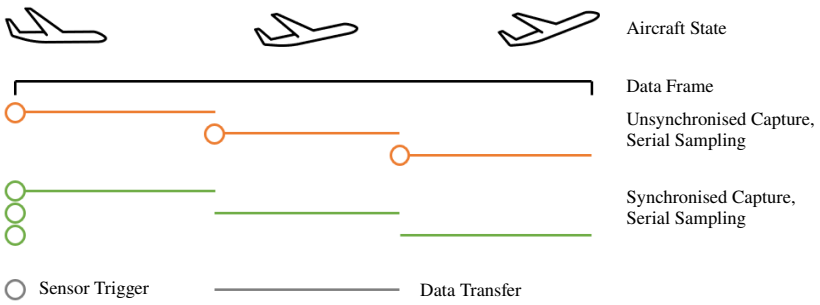

Fig. 4 Capturing synchronised data is important where dynamics are being modelled. During a single data frame, the aircraft will change state, meaning sensors will not be measuring the same aircraft state if not synchronised correctly.

C. Flight Controllers

The flight controller is responsible for the high-rate stabilisation of the vehicle. This subsection is focused primarily on the flight controller software choice (rather than the hardware which is dealt with in Section II.D), however it should be kept in mind that the choice of both are linked.

1. Considerations

Each flight controller solution has its own strengths and weaknesses, thus a thorough review of the different options available is worthwhile as research groups will typically pick a flight control software and continue to use that into the future. Aspects that should be considered are:

- The flight controller capabilities:
  - Does the flight controller support the functionality (aircraft configuration, logging, way-point navigation, etc.) that is required for the project?
  - Does the flight controller support integration with other systems such as ROS and ground station software?
- The available support:
  - Is there an active online community that can help support the debugging of issues and aircraft configuration (such as parameter set up)?
  - How comprehensive is the online documentation?

2. Open Source

Open source flight controllers have their source code freely open, available, and modifiable by the community, meaning the flight controller can be used as-is, or modified freely to suit specific needs. In the past 10-15 years, open source flight controller have matured significantly, with ArduPilot[*], PX4[†], Paparazzi[‡], and many more all providing advanced features backed up by thousands of hours on real hardware.

As the source code for the flight controller is open, hardware issues may be overcome by porting the flight code to bespoke flight controller hardware. In this case, there are often guides available online to help the researcher modify the necessary files. Similarly, if particular functionality isn't available, a researcher may be able to add this in easily, depending upon how the code is structured. A lack of available support/documentation on the other hand is very difficult to overcome, especially given the complexity of modern flight controllers.

3. Closed Source

Closed source flight controllers typically come with a COTS aircraft, or are purchased (along with the associated hardware) to install on a custom vehicle. Examples of such systems are the Cloud Cap Piccolo[§] and the DJI autopilots[¶].

[*]ArduPilot - `https://ardupilot.org/`
[†]PX4 Autopilot - `https://px4.io/`
[‡]Paparazzi Flight Controller - `https://wiki.paparazziuav.org/wiki/Main_Page`
[§]Cloud Cap Piccolo - `https://www.cloudcaptech.com/products/piccolo-autopilots`
[¶]DJI Naza-M V2 -`https://www.dji.com/naza-m-v2`

Closed source flight controllers are beneficial in that they may have some form of certification, dedicated developers and support, (potentially) higher standards for quality control, and a more 'integrated' ecosystem. This however comes at not just the financial cost of the software, but also in a lack of flexibility as changes to any part of the system is out of the hands of the researcher. For example, if six months into a project a new component needs to be integrated (such as a camera, computer, or sensor), the researcher is either required to select from a (likely) limited set of 'approved' systems, or is beholden to the developer to implement the new functionality.

4. Custom Solutions

Custom solutions give you complete control over the flight controller, however they require long development times and significant effort to get working. Developing a custom flight controller might be required if there is a particular core requirement that is not currently available on existing flight stacks (such as hardware sensor synchronisation [3, 8]), for bespoke hardware for which existing flight stacks can't be ported, or if the use case is not allowed by licence agreements (such as for UAVs with offensive capabilities). Custom flight controllers can leverage existing drivers for sensors and interfaces to lessen the development effort required, however ensuring the code is stable and bug-free is still a significant undertaking. Furthermore, it is very easy to de-scope features that maintain safety for the operators, presenting a very real threat of injury to operators due to unscheduled motor starts or actuator movements.

D. Compute Package

The compute package is responsible for doing all the computing on board the aircraft, and is one of the main payloads carried by research vehicles for acquiring data from sensors, executing computationally-complex algorithms, and for recording data. Generally, the term compute package would exclude the low-level flight controller (as this is required for the aircraft to fly), however for this paper, it will be addressed in this section as it more generally deals with the computing hardware.

This subsection presents the considerations when choosing compute packages, both for high- and low-level processing, and the key points in determining what type of compute hardware to install on a UAV.

1. Compute Package Architectures

The two common compute package architectures are running a micro-processor-based flight controller linked to a 'companion' computer, and an all-in-one system where the main computer combines the function of the two. All-in-one system (such as a Qualcomm Flight[‖] or Emlid Navio[**]) have significant SWaP advantages, however severely limit flexibility as projects mature due to their highly integrated nature. On the other hand, a separate flight controller and companion computer allow each of the two components to be upgraded as necessary and give far more flexibility in what components can be used together, allowing the hardware to be more closely matched to the requirements of the project.

Though the following subsections will break the components down into individual parts, the same considerations apply regardless of which architecture is chosen for use on a UAV.

2. Low-Level Flight Controller

The low-level flight controller is responsible for the high-rate stabilisation of the vehicle. Though the algorithms used are typically very computationally inexpensive, they are required to run very predictably and at high loop rates, meaning they are typically implemented on a dedicated micro-controller-based system.

There are many different options for which hardware to use, and often each hardware stack supports multiple flight controller software stacks, which leads to (for better or worse) a large array of potential hardware options. In general, the main aspects to consider when selecting the hardware for the low-level flight controller are:

- Does the hardware support the software stack that is planned to be used?
- Does the flight controller hardware have all the I/O (servo outputs, UART ports, I2C ports, etc.) that are required?
- Is the available flight controller hardware the right size/shape for the purpose?
- Are the connectors used to interface to the board readily available?
- Is the flight controller hardware readily available at an acceptable price?
- Does the hardware support the timing requirements for the task?

[‖] Qualcomm Flight - `https://developer.qualcomm.com/hardware/qualcomm-flight-rb5`

[**] Emlid Navio - `https://navio2.emlid.com/`

The main challenge is ensuring the flight controller hardware has the correct I/O for the task at hand. Even though systems such as quadrotors may seem simple as they require only 4 outputs, there are many inputs that are required such as for command and control, additional sensors (such as GPS or rangefinders), and for interfacing with the companion computers. Oftentimes multiple sensors can be placed on a single bus (such as for I2C or CAN), however UART ports, a one-to-one connection, often run out. Hence, it is important in the design stage to have an idea of what sensors will be required (and what bus they use), and to create a system diagram to ensure there is enough I/O ports and bus bandwidth. Depending on the projects timing requirements, sensors on a single bus may not be suitable (if readings take too long) and a more parallel architecture is preferable. The hardware must also support high enough data rates to log and save all the data generated, plus whatever debug data is also required, which can quickly add up.

Secondary considerations are the size of the flight controller, availability of connectors for interfacing, and the cost of the flight controller. Typically the most critical of these is the availability of connectors for creating custom cables - flight controllers often used very small connectors that are difficult to crimp, hence most will opt for pre-crimped cables which may not be easily sourced. During a typical project, cables will be cut, soldered and re-routed many times, meaning an ample supply will be required. Finally, when determining the allowable size of the flight controller (and more generally for any compute package), cable routing must be taken into account. Ports are often side-mounted and cables have a non-zero bending radius, meaning wiring into/out of the hardware contributes significantly to the effective footprint.

3. Companion Computer

Companion computers are computers that are designed to interface with the flight controller, providing additional compute power. They may be responsible for running motion planning algorithms, visual interial odometry (VIO), recording data from sensors, or any other function not performed by the main flight controller. Which companion computer to use depends on the compute required and the space available, and in most cases there are many different options to choose from.

For many tasks, small, single-board computers (SBCs) such as the Raspberry Pis provide ample computing power. Their small size and power requirements make them ideal for UAV work, and their low cost lessens the consequences of any crashes. For use cases such as neural networks where there are training and deployment phases, while they may not be able to perform any training, deployment of these algorithms is computationally cheaper and hence SBCs can be used. Deployment from development computers to SBC-based flight hardware is occasionally challenging due to the differing instruction sets (x86 vs. ARM), though this gap is quickly being closed as more personal computers switch to ARM-based processors.

When more compute power is required or sensors are used that provide high data rates, Mini PCs (such as NUCs) are a good solution. Mini PCs are designed as desktop replacements meaning software compatibility is high between development computers and the flight hardware, easing the transition to deployment on hardware. Though they are often heavy, significant weight reductions can be achieved by de-casing the Mini PCs, though this makes the computer significantly more vulnerable to damage during transport, handling and crashing. Mini PCs come in a wide range of sizes, performance specifications, and costs, meaning the trade space is quite open to whichever performance/cost point the researcher chooses.

In cases where the higher level computing can be highly parallelised and broken down into multiple, smaller simultaneous calculations, GPU-based computers (such as the nvidia Jetson and nvidia Orin) offer very good performance. As such, GPU-based computers do well at tasks such as computer vision and machine learning, though are not as well suited for tasks such as motion planning. GPU-based computers have a steep learning curve, requiring specialised coding skills to take full advantage of the hardware, though these skills are becoming more common as robotics increasingly adopts AI. Furthermore, as GPU-based computers become more mainstream in robotics, pre-built software packages are becoming more abundant, further reducing the learning curve.

4. Payload Sensor Computers

In cases where a COTS instrument package is used (such as a multi/hyper-spectral camera), all the necessary compute, processing and logging may come with the system. This greatly decreases the design burden on the researcher as all the 'sizing' required for the compute system has already been completed. These types of systems typically just need to be bolted on (and perhaps supplied with power), and the data is processed offline once the flight has been completed.

E. Airframe

The airframe effectively holds all the parts of the system together and provides a platform to transport the sensors and compute package to where they need to go. The choice of airframe can have a significant effect on the quality of the collected data, the ease of development and testing, the resilience to crashes, and the cost of the system. The key considerations when choosing an airframe are

- What is the required range and endurance of the aircraft?
- How much payload volume and mass is required?
- Where will the aircraft be tested, how will it be transported, and who will be the pilot?
- How does the airframe choice affect sensor performance?
- What level of redundancy is required?

This section presents many of the different considerations when choosing an airframe and how the choice can affect the system as a whole.

1. Configuration

There are currently three main aircraft configurations used for research - multirotors, fixed-wings, and powered-lift hybrids (Figure 5). Multirotors are by far the most common - they are comparatively easy to build, have a high payload fraction, can be flown in small spaces, are easy to pilot, and in general, are reasonably resilient to crashes. Fixed-wings require significantly more skill to build and fly, require more space to operate, and have lower payload fractions, however are unmatched in terms of range and endurance. Powered-lift hybrids combine a multirotor and fixed-wing configuration into a single vehicle, extending the range and endurance compared to a standard multirotor, but at the cost of complexity and the parasitic weight and drag of the lifting motors during cruise.

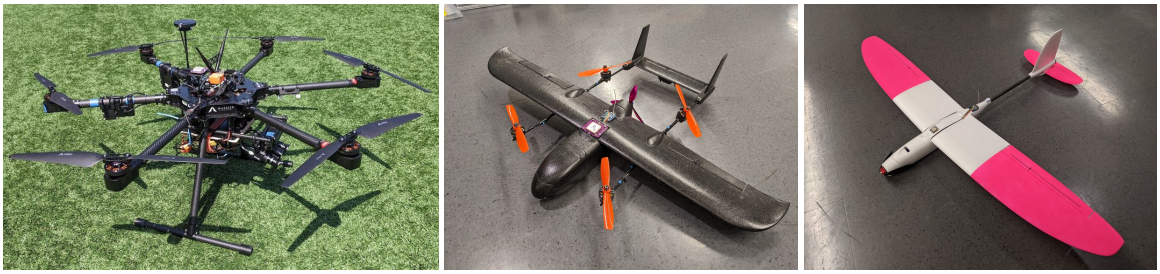

Fig. 5 There are many different configurations of aircraft. This figure shows the three most common configurations used in research - multirotors (left), powered-lift hybrids (centre), and fixed-wings (right).

Many types of research such controller development and multi-agent planning may only require short flights, on the order of minutes, to collect data or successfully demonstrate capabilities, meaning aircraft endurance isn't a critical decision factor. While changing batteries often does slow testing down, it means less of the total weight of the aircraft needs to be devoted to batteries, meaning either more sensors / compute can be carried, or that smaller, lighter systems can be used. For flights that require long, continuous periods on station or long transits (such as for science missions), multirotors may not have the performance required. In this case, powered-lift or fixed-wing vehicles are the only solution, though often, some creative planning may enable flights to be completed across multiple batteries. Unless a very long endurance is required, the aircraft should be electrically powered as internal combustion engines introduce severe vibrations that are difficult to effectively filter out mechanically and in software.

Methods for launching and recovering the aircraft are perhaps the second-most major driver for choice of a particular configuration. Vertical take-off and landing makes things very simple and requires little space for launch and recovery of the vehicle, but comes at the cost of being extremely power intensive. Fixed-wing vehicles require more space for take-off and landing, and there are many choices as to how this is achieved. A traditional undercarriage allows for a standard take-off and landing which is very gentle and safe, however a prepared runway is usually required, limiting the fields that can be operated from. Hand launching is another popular option as it doesn't require a runway, though it is riskier (poor throws are common) and there is a low upper limit to the maximum weight of the aircraft. Bungee launches are similar to hand launches but can launch much heavier loads in a more consistent fashion, however are non-trivial to build such that operate well and in a repeatable manner. They also apply high loads to the aircraft structure. A launch dolly provides the gentle launch of a runway take-off, but removes the drag of a landing gear during flight. At the extreme end of the spectrum, the test aircraft can be dropped from a multirotor [2] (Figure 6). Recovery without an

undercarriage is usually achieved with a belly-landing which will quickly degrade the bottom of the fuselage if it is not adequately protected.

Higher-risk projects or those with expensive payloads may also strongly weight redundancy when determining the aircraft configuration. A quadrotor offers almost no redundancy, with a failed propeller or motor resulting in a near certain crash, though there have been demonstrations of controlled flight with a lost rotor [9]. Multirotors with six or more motors are generally tolerant to a single motor failure (assuming sufficient thrust is available), and though they may suffer severely degraded performance, will likely be able to make a controlled landing. Multirotors are not tolerant to a main battery failure, hence care must be taken to always land with battery capacity to spare. Failure of the main batteries is generally not catastrophic for fixed-wing and powered-lift vehicles as often the servos will continue to drive control surfaces until well after the main propellers have stopped working due their lower operating voltages, allowing the vehicle to glide for a landing. Failure of control surfaces may also occur, and, depending upon the set up, a single failure may cause a total loss of control (such as for a single elevator servo). For added safety, some operators may choose to add a parachute to the system, though this adds weight and once deployed, the aircraft will be uncontrollable and will drift with the wind.

Fig. 6 Dropping aircraft from a tether can be used when a conventional launch is impractical [2]

Propeller location can vary between aircraft of the same configuration, and additional thought must be given to how that will impact the collected data. Pusher configurations make integration of all the components easier as the front is typically available for sensors and batteries. Additionally, the propeller wash over the tail is good for controllability, however it can introduce low frequency noise that is difficult to filter out because of the unsteady flow across the tail and the disturbed propeller inflow behind the fuselage. Additionally, the blown control surface means that throttle settings can affect control in a non-linear way. The thrust axis should go through the CG location to minimise this effect. Finally, hand launching is more dangerous with a pusher propeller, and component cooling might be more difficult at the rear of the fuselage. Twin propellers on the wings are another means of clearing the nose section at the expense of additional complexity and control issues due to asymmetry in yaw.

2. Aircraft Size

Payload mass and volume are the primary drivers of the overall size of the aircraft, and it is important to allow room for the sensor and compute systems to grow as oftentimes the final system differs significantly from the initial implementation. An important consideration when sizing is the aircraft is not just the components of the payload, but also the wiring and cables required to join them all together. Although computers continue to be miniaturised, the associated wires and connectors have generally stayed about the same, and now often take up more volume than the computers and sensors do.

Larger aircraft have more space to fit equipment and are in general more stable, however are more expensive, can potentially cause more damage and are more difficult to transport. Due to their size, airframes are typically dis-assembled for storage and transport. In these cases, there must be a system for ensuring the aircraft is reassembled with good alignment to prevent sensors installed in different parts of the aircraft from becoming mis-aligned. Experiment design must take this into account, and may require re-calibration each time the aircraft is re-assembled.

3. Build Material

Airframes are typically made from either foam, plywood/balsa, or carbon fiber. The material the airframe is made from affects many things including the robustness, the consistency, sensor noise and RF transmission.

Foam airframes are more robust to mishaps and are typically cheaper, but often have worse noise characteristics due to the flexibility as the natural frequencies of the structure are lower and more often interfere with the rigid body dynamics. They also tend to flex significantly under load (wings relative to the fuselage, tail relative to fuselage), which may change the flight characteristics, and will warp/deform over time due to heat and being stored incorrectly. This makes foam undesirable for any type of airframe characterisation as the vehicle dynamics are likely to change noticeably over even days of use. Foam aircraft are easy to fix in the field and are thus great for beginners and development, however any mature project will likely find foam to cause more problems than it solves.

Traditional plywood/balsa construction typically has a better controlled noise environment (see Section II.F.3) and a higher stiffness, meaning the airframe deforms less during flight and hence will give more consistent results. Plywood/balsa aircraft also have the advantage that the wings are typically hollow, providing ample room for routing wires and housing sensors, and the structure can often be made significantly lighter than a foam-equivalent. Crashes with plywood/balsa aircraft are much more likely to be complete write-offs than for foam aircraft - even seemingly benign crashes that load the aircraft in just the wrong way can quickly 'match-stick' the plane.

Carbon fiber aircraft provide the ultimate stiffness-to-weight ratio (and is arguably the most aesthetically pleasing), though this comes at a considerable monetary cost. Carbon fiber is also electrically conductive and therefore radio receivers, GPS, and anything else that relies on RF signals need to be placed with special care to ensure they continue to function well. Carbon fiber is the go-to for multirotor designs as multirotors are easily constructed with beams and plates, meaning that though repair might be difficult, broken parts can simply be replaced.

4. Plug and Play vs. Almost Ready to Fly vs. Scratch Build

Designing and building a well-behaved, stable fixed-wing aircraft requires a wide variety of skills, many of which are not typically taught at university. As such, it is generally preferable to purchase aircraft off the shelf as the challenging aerodynamic and stability work has already done. Plug and Play (PnP) aircraft typically come finished such that only a battery and receiver need to be added. While these are quick to get up and going, they can be difficult to instrument as the airframe designers typically leave little left over room for any additional components outside the original design specification.

Almost ready to fly (ARF) aircraft take some assembly, however are much easier to modify and typically make the best aircraft for research. They allow the user to install their own electronics (so you can add servos and motors that provide angle / RPM feedback), and as the fuselage often comes in two halves, space can easily be cut to house extra components.

Scratch building an aircraft is likely the only option for exotic designs, however in these cases, significant time should be set aside for building and refining the design such that the aircraft flies well. In all cases, spares should be bought as if a crash (or wear and tear) writes off an aircraft, a replacement may no longer be sold, meaning any custom integration and airframe characterisation that has been already completed needs to be redone.

Multirotors are much more forgiving when being designed and built. If the components are correctly sized for the task[††], it is relatively easy to assemble a system following online videos and tutorials that works well for most projects. Plug and play multirotors are becoming increasingly available, however these are typically highly-integrated units designed for high performance first-person view (FPV) racing, making them unsuitable for most research. Similarly, as most multirotor frames are designed for FPV, they are typically not designed to take additional compute units (such as a Raspberry Pi), so some modification will be required. As multirotors lend themselves well to boom-and-plate type construction, a scratch build frame can quickly be designed and manufactured using a mix of COTS and 3D printed components.

Ultimately, a stable, robust aircraft is key to the successful development of any research project. Even where a higher performance aircraft is ultimately required, conducting the majority of the development work on a more benign aircraft and porting it over later in the project is highly recommended.

[††]Sizing components for a custom multirotor can be daunting, however there are plenty of resources online to help with this. `e-calc.ch` is also a great resource for trying out motor, ESC and propeller combinations before buying parts.

5. Power Systems

The power system is responsible for powering the aircraft systems, sensors, and compute package, and is one of the most critical, yet most overlooked, aspects of any UAS setup. Power issues are difficult to diagnose as sensors will often brown out when the power system is under heavy load and recover, rather than completely fail. Loss of logging and/or momentary control failures are key indicators of a brownout, though the symptoms may vary wildly.

Though not always practical, it is good practice to use separate batteries for propulsion and the electronics as this reduces electronic noise fed into the sensors significantly. As brushless motors are slowed down, they cause voltage spikes which are fed back into the power system, causing fluctuations that adversely affect sensors, computers, and another other systems that require clean power. Furthermore it allows the electronics to be powered on while having no power to the motors, greatly increasing safety while developing on the vehicle.

A port for ground power (i.e. shore power) should also be added parallel to the electronics battery to enable the compute systems to stay turned on during battery changes. Typically there is a lot of setup, configuration, and checking each time the compute package is booted, and having the shore power means these checks only need to be completed once during the day. It also means the flight battery can be kept at full charge right up until roll out to the runway, ensuring the maximum amount of flight time can be achieved.

6. Colour Scheme

The colour scheme doesn't just make the aircraft look good, it is a crucial element in helping the pilot maintain the orientation so that they can effectively take control as required. This is particularly important in research where the pilot quickly has to retake control of the aircraft from unusual attitudes, often with little to no warning.

Fig. 7 Painting aircraft in bright colours with different patterns on the top and bottom helps the pilot maintain orientation of the aircraft at long distances

While it may be tempting to paint an aircraft in dark, 'stealthy' colours to make it look cool, this makes it extremely difficult to track the attitude, especially as the aircraft is silhouetted against the sky. Bright colours that stand out and are different on the top and bottom (Figure 7) have proven to work well, with the orientation visible at long ranges and in a wide variety of lighting conditions. For multirotors, bright navigation lights and coloured tape on the rear arms and back of the landing legs has worked well.

F. Integrating the Hardware

1. Creating a FlatSat

Generally, the first step for integrating new components together is to create a 'FlatSat' - a flat board with all the parts laid out such that they are easily accessible. The FlatSat helps identify compatibility issues between parts, enables tools like logic analysers to be easily connected, and helps contain all the wires and sensors during initial testing. FlatSats often continue to be used all the way throughout the project as they are much easier to develop on (as opposed to an assembled aircraft), and allow hypotheses (such as the radio is interfering with the GPS) to be quickly and easily checked. Switching out components is also much easier with a FlatSat, enabling many different combinations to be trialled quickly and efficiently.

2. Sensor Placement

The location of the sensors on the aircraft can greatly affect the their performance, and some trial-and-error is usually required to find the optimal positions. An example layout is given in Figure 8, though in general:

- Inertial measurement units (IMUs) should be positioned near the centre of mass in a low-vibration environment, and be directly mounted to a solid part of the aircraft structure to prevent secondary vibrations and phase lags
- Magnetometers should be place away from any metal or alternating currents
- GPS antennas should be placed away from any radio transmitters or other active GPS antennas
- Airspeed sensors should be placed away from the fuselage and out of the way of any propeller wash
- Absolute pressure should always be measured using a static port on the airspeed sensor or another suitable location. Measuring it inside the fuselage can lead to significant errors, especially if the fuselage is vented.
- Antennas should be placed external to any carbon fiber or metal structures and orientated to optimise reception along the flight path
- Cables between sensors should be kept short enough to ensure reliable communications (this depends on the communications protocol) and be twisted to help with noise rejection
- Wires should be tied down to the structure to minimise movement in flight and long-term fatigue

As aircraft are flexible structures, care must be taken when mounting sensors that measure angles (such as IMUs and flow angle sensors). Even small deflections, on the order of 1 or 2 degrees, significantly affect the data collected from the aircraft, and require significant characterisation efforts to correct.

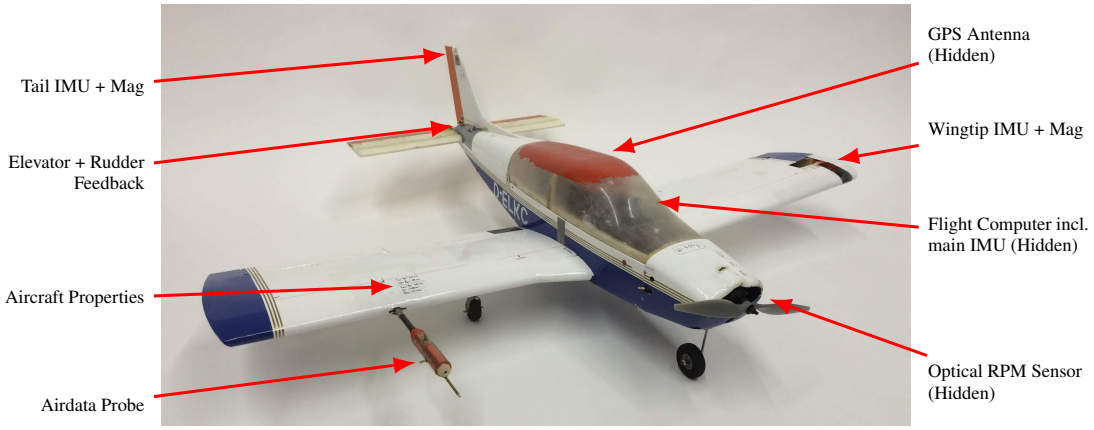

Fig. 8 A model Piper used for system identification [3]

A ground plane for the GPS antenna made from thin sheet metal or aluminium foil (Figure 9) can drastically improve GPS reception as most commercial antennas are designed to operate on the metal bodies of a car. This can help ameliorate issues where GPS reception is weak in general, or where GPS is lost in banked turns. Any ground plane installed for the GPS system acts as RF shield and must be compatible with all other radio systems in the aircraft (remember to perform radio range checks).

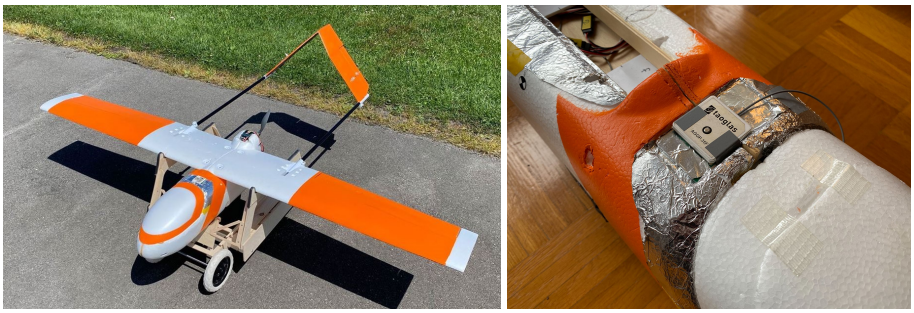

Fig. 9 Example of a GPS antenna ground plane. Note the continuation of the ground plane towards the wing mount.

Sensors like air data probes are typically mounted on the wings, and thus are very prone to damage. Mountings that can 'break away,' such as those held on by magnets and aligned with V-shaped mounts, can significantly improve the survivability of these sensors during a crash.

3. Vibrations

Vibrations are much more critical on small airframes than on full-scale aircraft. The low mass and inertias of small airframes lead to high levels of vibration not only from the propulsion system, but also from atmospheric turbulence. Vibration damping techniques typically use the inertia of the device to isolate the sensor from the noisy airframe, which does not work for the light components used on small airframes. Hence, for successful small airframe flight testing it is of paramount importance to minimise all sources of vibration as it is almost impossible to remove them later, especially in softer foam airframes.

Motors and propellers are one of the primary producers of vibrations. Even small imbalances at high RPMs cause significant accelerations into the system, fouling IMU data and making flight path reconstruction difficult. Luckily, vibrations due to propellers occur at very predictable frequencies, meaning the can be efficiently removed in real-time and during post-processing using frequency-based filters. This however do not preclude the need to mechanically balance the system before flight, and an effective method for reducing vibrations from motors through mechanically balancing can be found in Appendix B.C.

An aerodynamically clean airframe is far less noisy than a 'dirty' airframe, with landing gear, large cavities and other obstacles such as cameras and antennas all being sources of aerodynamic vibrations. If this vibration level generated by the aerodynamics of the aircraft are too high for the planned project, no amount of engine balancing will help. Hence, it is important to determine the base noise level of a platform during glide tests in calm weather early in the project so that any issues can be fixed early. As an example, one of the author's projects ran into considerable difficulties with aerodynamic-induced vibration. The open cavities for retractable landing gear caused so much vibration that the attitude control system became overloaded by the sensor noise, which amplified those measured low frequency vibrations on the rotation rates.

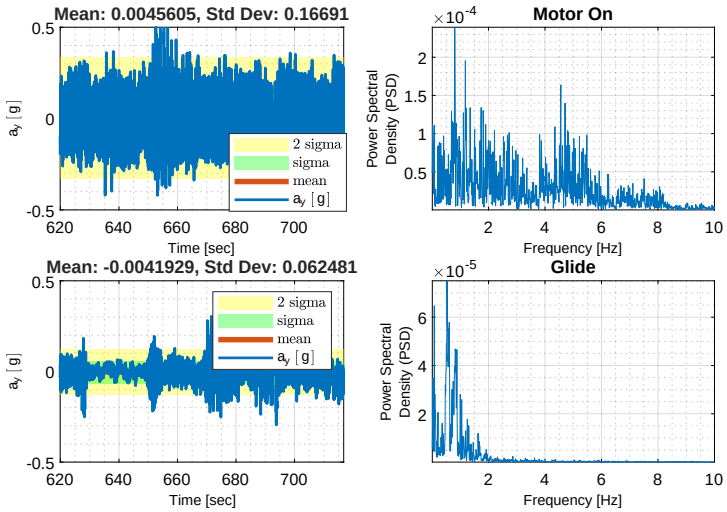

Fig. 10 Airframe vibration noise profiles change depending upon if the motor is running (top) or when gliding (bottom)

Figure 10 shows the base vibration levels during a glide test, where it can be compared to motor on segments earlier and later in the flight. The magnitude of the noise floor is much lower than with motor on and the noise has much lower frequencies.

G. Integrating the Software

1. Communicating with Sensors

Software stacks quickly become very large and complicated as different sensors and new functionality is added. Keeping each element separate and having them communicate on a central network is ideal as it helps compartmentalise each component, meaning a change in one sensor, controller or other element is unlikely to affect any of the others.

Many different research groups have their own environments for doing this, however, ROS has become the defacto standard for use in robotics. ROS [10] (and the newer ROS2 [11]) act as a middleware that facilitates communication between each of the different elements in the system via a publisher / subscriber system. As ROS should be familiar to those in robotics, the details won't be discussed here, but instead this section will focus more the architecture of the software.

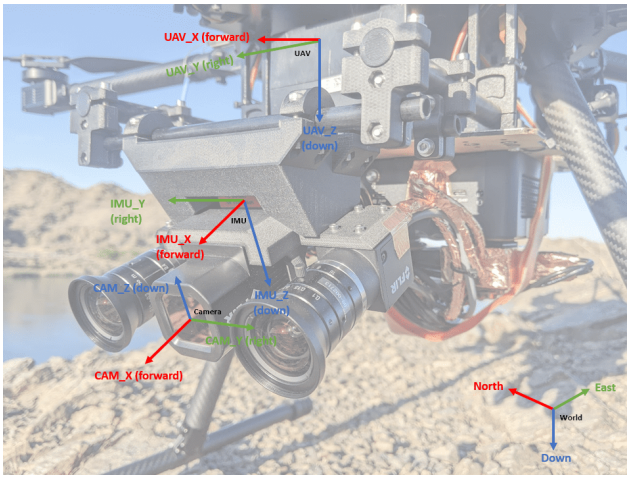

Fig. 11 Even a simple UAV with a camera may have several co-ordinate frames that all need to be tracked. Note the cameras are mounted upside down on this UAV and that the camera frames used here are oriented differently that typically used in computer vision tools like OpenCV.

Any sensor data should have the frame of reference and units added to the variable name, ideally throughout the code but at a minimum when the values are logged. It is very easy to loose track of where data channels have come from (Figure 11) and in what units they are, especially several months after the data was taken. Furthermore, some form of self-defining log structure (including any parameters), or a header that is placed in each log file helps decrypt what was logged, what settings were used, and any other important information into the future.

2. Software Architecture

The main concept when designing the software architecture for a flight project is to always have a fallback which enables either 1) the pilot to regain full control of the aircraft, or 2) the aircraft to automatically land itself. How this is implemented depends upon the task at hand, however the general idea is to fall back to software that was not developed for the project, and assume the code developed for the project is 'not trustworthy.' As an example, if ArduPilot ('known good') is the low-level flight controller, switching back to a manual mode (or an automatic landing mode) via an RC transmitter connected directly to the flight controller is likely to succeed in regaining control of the aircraft. This does however require that the companion computer code is controlled sufficiently that the aircraft cannot automatically switch itself into a computer-controlled mode[‡‡] (such as GUIDED or OFFBOARD). In extreme cases, completely severing the connection between the flight controller and the companion computer via a relay controlled by the RC transmitter may be warranted to guarantee no commands are received from the companion computer during manually piloted flights. Completely severing the connection is also useful when the companion computer is providing part of the state estimate to the aircraft - algorithms such as VIO can go unstable very quickly, and completely cutting off the bad data stream will give the onboard filters the most time possible to recover a good state estimate. In cases where the flight controller is part

[‡‡]Ideally, especially for outdoor testing, the companion computer should never be able to switch the flight mode as this can lead to unexpected behaviour resulting in events such as the vehicle flying away (because it has commanded straight and level) or unexpected start ups of the motor.

of the development process, an existing, 'known good' flight controller can be run in parallel, with relays (controlled via the 'known good' flight controller) used to switch between which flight controller has control of the actuators.

3. Reducing Pilot Workload

Flight testing demands a high workload of the pilot, and automatic fallbacks that help automatically recover the aircraft if testing isn't going to plan are worthwhile adding. The software developer should have an idea of the expected envelope that the aircraft will fly in during testing (and if not, ask the pilot), and hence they can develop routines that will abort the test if this envelope is exceeded. As an example, during system identification inputs, if the aircraft exceeds ±30 deg pitch angle or the airspeed drops too low, the system will abort and command straight and level. Once again, the pilot should always be able to regain full control of the aircraft, and these automatic fallbacks should never be able to switch into other flight modes.

Where possible, maneuvers and missions should be pre-programmed (and simulated) ahead of time so that the inputs into the aircraft are repeatable across different flights. This makes comparing data across flights significantly easier, and also increases the chance collected will be what the researchers require. Pre-programmed flights also don't exclude the pilot from also manually piloting the aircraft, giving the opportunity to modify plans on the fly as required.

4. Interfacing with the Wider UAS Ecosystem

The UAV itself is just one element in a wider UAS ecosystem, and there are many mature, open (and closed) source tools available for performing functions such as ground station control. MAVlink[§§], a lightweight communication protocol designed specifically for resource-constrained flight controllers, is the defacto standard for communicating with low-level flight controllers. As MAVlink has a standard message set and is open source, it can easily be integrated into custom flight controllers, allowing developers to leverage the existing UAS ecosystems. MAVlink is supported in ROS (via mavros[¶¶]), significantly reducing the development times required to get the flight controller communicating with other computers on the UAS network.

H. Data Management

1. Aircraft Reference Quantities and Mass Properties

For all projects that involve the characterisation of the airframe itself, the mass properties must be known and recorded. It is recommended to record the mass properties in the flight test data-set, especially if the properties change between tests. This way, errors in correlating notes and data are avoided as all the aircraft properties can be automatically imported for each data set.

While the aircraft mass can easily be measured before the test using a scale, the inertial properties are much more involved. Methods include swing tests of the airframe [12, 13] or extracting the properties from a detailed CAD model. For small fixed wing airframes, the measured inertias can diverge significantly from the values extracted from CAD due to the phenomenon of added mass [13]. This must be taken into account when choosing the method, otherwise large errors in the identified aircraft model can occur [3].

To measure the reference quantities and the CG location, it is recommended to choose a fixed reference point that is easily accessed and is unlikely to change. The authors often use the rear center of the firewall/motor mount. On some airframes, it may be more convenient to determine dimension relative to a point located on a stand or another location not on the airframe itself, however this point must be fixed and repeatable.

The longitudinal and lateral CG points are easy to find using either three scales or a balancing technique, however the vertical CG location on fixed-wing aircraft is often difficult to determine. One technique that is relatively easy to perform is to find the longitudinal CG, and then suspend the aircraft from a mounting point on the top of the vertical fin or another convenient off-center location (Figure 12). A string with a weight suspended from the mounting point towards the CG location will intersect with a vertical line through the longitudinal CG location. The intersection point is the vertical CG location, and can be measured using a 1:1 photocopy of a ruler taped to the fuselage.

[§§] MAVlink - `https://mavlink.io/`

[¶¶] mavros - `https://github.com/mavlink/mavros`

Fig. 12 Procedure to determine the vertical CG of a fixed-wing airframe

2. Recording Events

A lot happens in a single flight, yet typically, very little of the recorded dataset is of interest. As maneuvers for controls testing, performance analysis and system identification typically last for seconds, many different tests can be performed in a single flight, meaning knowing exactly what was performed when can be challenging. Hence, an efficient system for recording what happened is crucial such that the important segments can be extracted from amidst the rest of the flight.

Thus far, the most effective system the authors have found for recording and extracting maneuver datasets has been to have a dedicated data channel mapped to a maneuver enum. This, coupled with handwritten flight logs containing takeoff / landing times, quick descriptions of events (planned and unexpected), and any notes from the flight crew, allows for data to be located and extracted easily in the future. In particular, dedicating a data channel to identifying maneuvers allows for data analysis to be batch analysed, meaning entire flight campaigns can be processed with a single script. This enables a very fast turn around time between the aircraft being flown and the first pass at analysing the data, allowing multiple test, analyse, improve cycles to be completed in a single day.

3. Post-Processing and Data Processing Tools

Effective and quick data analysis is key to the success of a project, but can be challenging due to the large amounts of data produced. Where possible, the data should be inspected while in the field so that flight plans and configurations can be modified as required, dramatically improving the iteration rate of the testing. Most available ground station packages for the common autopilot systems feature some kind of data plotting capability, but this usually limited to very simple plots, meaning more in-depth analysis is usually required. Pre-written MATLAB or python scripts that can import, process, and display plots of the flight data of interest enable quick checks to be performed between flights so that potential issues can be caught and fixed.

For more in-depth analysis, the authors developed kVIS, an open source MATLAB app for plotting and analysis of time sampled data [14]. kVIS is based on a board support package (BSP) concept and can easily adapted to any data source. It has powerful plotting features, data representation in a clear tree structure, event management, report generation and the ability to save and apply complex plot views to the data allow for fast and comprehensive data analysis. A plug-in architecture allows to integrate with any custom algorithms, greatly speeding up data processing through automation. kVIS has basic BSPs for ArduPilot and PX4, and it is easy to create custom BSPs based on the provided examples. kVIS is available on GitHub [14], and is used as the primary flight data analysis tool on the high profile Lilium Jet, as well as for several PhD projects in academia.

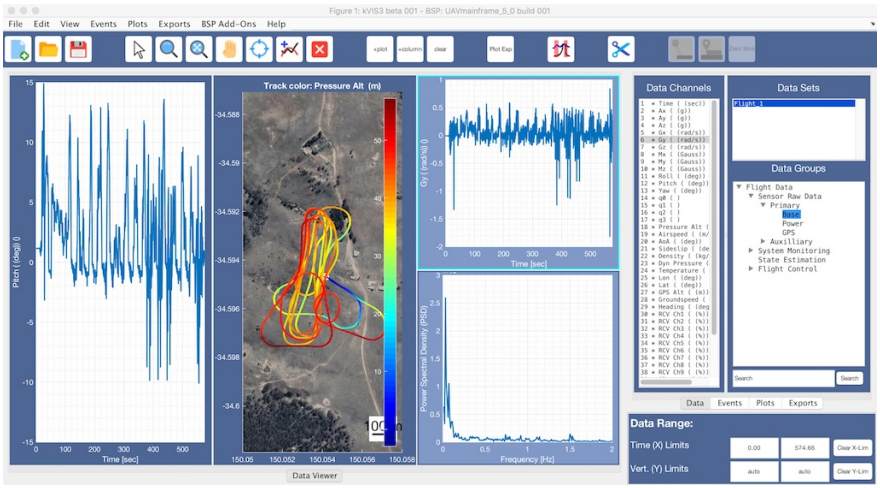

Fig. 13 kVIS data management user interface. kVIS supports advanced signal processing features including filtering, spectrum analysis and automatic maneuver extraction.

4. Flight Path Reconstruction

One of the key tools for most types of analysis involving the aircraft state is accurate flight path reconstruction based on a rigid body motion model in 6 DoF [3]. While the aircraft usually knows its state in real-time for onboard control (usually via an onboard Kalman filter), this real-time filter is tuned to favour consistent, stable estimates at some cost to the accuracy. By recording the raw sensor data from the aircraft, Kalman filters can be run offline and tuned to provide more accurate solutions without risking the aircraft. More advanced techniques like iterated runs and backward smoothing are only possible in post processing. Flight path reconstruction estimates sensor errors (depending on the model used), wind conditions, and is able to fix sensor dropouts and produce high frequency estimates from slow signals like GPS.

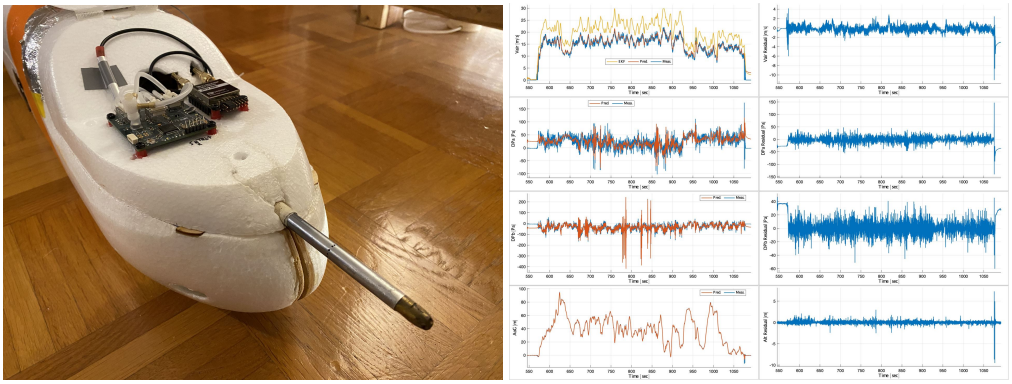

Fig. 14 Kalman Filter-based calibration of a 5-Hole airdata probe mounted close to the fuselage. Note the large errors in the airspeed measurements.

Using the sensor error calibration capability considerably reduces the need to calibrate and align the sensors in the airframe, which is difficult at this scale. Figure 14 shows an example of a 5-hole airdata probe installed on the nose of the fuselage, fairly close to the structure. As can be seen in the foam structure around the probe, it has seen its share of damage due to its exposed position. The Kalman filter can estimate the calibration factors for the flow angles, the airspeed measurement distortion from the body behind the probe and the installation errors from the damaged mount to produce a reliable result despite the flawed sensor. With a bit of practice, this can be a powerful capability and should always be preferred over using raw data from the sensors. However, the more information on the sensor system is available from calibration or previous experience, the better the Kalman filter will work. It is not a solution that allows to do away with all good practice in data acquisition.

I. Cybersecurity

One often overlooked aspect of robotics in general is cybersecurity. The internet is almost indispensable to any project, however robots and systems that are connected to it are always vulnerable to attack***. The authors are by no means experts in the field of cybersecurity, and researchers should always consult up-to-date resources on current best practices, however there are some simple steps that should be taken to reduce the risk.

- At a minimum, usernames and passwords should be changed from the defaults. More recent versions of Ubuntu enforce this on first boot, however many older distros do not, and this is a very simple step to make it significantly harder for unauthorised access to occur.
- Disabling passwords over ssh and instead using ssh keys not only makes it more difficult to guess the password, it makes accessing the system via ssh for authenticated users much easier as a password doesn't need to be entered each time.
- Software should be kept up-to-date to ensure any known exploits are patched.
- Care should be taken to correctly type names when installing python packages. There are many malicious packages available that have similar names to standard packages, opportunistically waiting for typos during installation.

III. Flight Testing

Flight test campaigns are an involved and iterative process that takes years of experience to become proficient at. Rarely are objectives achieved on the first attempt, however, with careful planning and preparation, safe and efficient flight tests that yield useful data are possible. This section details the flight test campaigning process, focusing on data collection techniques and the transition from simulation to lab to field tests.

A. Testing in Simulation

Simulation is a powerful tool for development. Not only is the hardware not at risk, the turn around time between coding, testing, and seeing the results is increased orders of magnitude over hardware testing. Furthermore, full, perfect state knowledge allows the root cause of issues to be investigated with far less uncertainty, reducing the guesswork required to solve problems. Simulation also allows datasets to be generated, ideas to be tested, and analysis techniques to be verified without needing to physically access the aircraft. ArduPilot, PX4 and ROS all have their own simulation back-ends that can help facilitate the use of simulation environments. Examples of simulation environments using ArduPilot and ROS are available as part of the resources listed in Appendix A.

1. Choosing a Simulation Environment

Simulation environments are not all built the same, and correctly choosing one that is fit for purpose is important. Simple flight models, like those provided by the ArduPilot SITL (software-in-the-loop) and PX4 HITL (hardware-in-the-loop) simulators, allow 'low hanging fruit' bugs (like reversed signs in equations), integration issues, mission planning, etc., to all be tested using the actual flight code in the loop with limited computational resources. These simulators can usually be coupled with higher-fidelity modelling, tools such as Realflight, X-Plane, Microsoft AirSim, ROS Gazebo and nVidia Issac which can be used to model sensors, actuators, aerodynamics, and environments in more detail, however with significantly higher computational resource requirements, a steeper learning curve, and a longer integration time.

The key to choosing a simulation environment is not necessarily to match the aircraft that being used as closely as possible, but instead to ensure the behaviour of the developed algorithms can be tested correctly. For example, the process for collecting and processing data for system identification is virtually the same regardless of the aircraft, so it is only important to match the approximate type of aircraft, rather than the exact dynamics. Conversely, algorithms that response to actuator failures will likely need more detailed simulations as it may be critical that gain tuning and controls responses match the real world behaviour.

2. Minimising the Sim-to-Real Gap

A simulator will never be able to fully represent the real world, and hence a balance needs to found between the diminishing returns of increasingly complex simulators and the effort/risk of actual testing. This means that for the

***As a side note, the authors once had a student whose system was hacked and tuned into a bitcoin miner. The vulnerability was that the default username and password were left unchanged. It was noticed by the university's network administrator and quickly shutdown, however this shows how exposed robotic systems can be.

optimal development of a project, the sim-to-real gap will never be zero. Optimally minimising this gap is a bit of an art, however there are a few general guidelines to follow.

Where possible, the hardware running the algorithms should slowly be migrated to running on the actual flight hardware so that issues with performance, timings, etc. can more easily be isolated. Knowing the limitations of the simulator also allows for a more informed flight test plan, as more time can be dedicated to investigating aspects that we not well simulated and hence have a higher likelihood of not working first time.

In many cases, simple simulators have sufficient fidelity to minimise the sim-to-real gap to levels where many of the bugs have been fixed and it is more time efficient to fly and collect real-world data. Getting enough confidence to fly earlier on in the project using a simple simulator allows for the experience of the flight test to feed back into the simulator, informing researchers as to where the fidelity of the simulation needs to be improved.

When corrupting the simulator data with noise to test the system robustness, it is strongly advised to use real, measured noise sequences in various conditions over the standard Gaussian white noise. Most algorithms assume Gaussian noise in their derivation and do handle such noise very well. Real noise from vibrations and turbulence does have quite different statistics, and system performance can degrade considerably in response to real noise spectra. If not tested in advance, this might lead to significant problems during the flight test.

Finally, dedicating time to practice real-world flights in the simulator also helps to minimising the sim-to-real gap. Flight testing likely involves many additional personnel, and practicing where the aircraft needs to be positioned, what calls will be used, what to watch out for, etc., can dramatically improve the efficiency of testing when moving to hardware.

B. Testing in the Lab

Testing in a laboratory environment is typically the first step in any flight testing. While the flights are likely to be space constrained, the close access, controlled environment makes it a very useful space to do as much hardware debugging as possible. This subsection covers some of key aspects the authors have found useful when testing in a lab that are often overlooked.

1. Battery Management

Flights in a lab environment are often short, with many cycles of take off, landing, code update, and repeat, being performed in a single battery. As such, very little of the time the computers are on is actually spent flying. To reduce the number of battery changes, a ground power system that can power the electronics (and not the motors) is very useful as it allows the main battery to be unplugged (increasing safety) while keeping computers on. 'Battery beepers' that sound a loud alarm when the batteries fall below a certain voltage are also very handy as it is very easy to loose track of the flight times, and ultimately helps prevent battery over-discharges.

2. Communications and Networking

Once the compute payload is integrated with the aircraft, connecting to it directly with a keyboard and monitor becomes difficult, thus a reliable network connection needs to be established. As is easy to loose computers on large networks, a dedicated router (and network) for connecting to the aircraft should be used, and the network details written on the router. As routers are often 12 V powered, they can be made portable using a LiPo battery and a regulator, meaning the networking setup can be identical between the lab and the field. Though wireless connections are usually preferred, a wired Ethernet connection alternative should also be available if possible as the authors have had many an occasion of wireless refusing to work during field trips.

3. Tuning

After a checkout flight, the first set of flights should be dedicated to tuning the flight controller. Flight controllers are typically set up as a series of individual control loops cascaded together to form complex control behaviour. At the very end of this controller cascade is typically the rate controller that converts desired angular rates to control outputs (actuator commands). The rate controller is typically the most difficult to tune, and yet, arguably is the most critical as every other output of every other controller ultimately passes through this controller. A poorly tuned rate controller means the aircraft will never be able to deliver a 'snappy' response, resulting in any upstream loops being adversely affected. Though an aircraft may appear to fly 'well enough' with the default tuning parameters, time spent to

correctly tune the lower levels of control will yield improved performance in the upper levels of control. Tuning in small spaces can be difficult, however some attempt should be made and can be done by applying impulses (to minimise displacements) into each axis and reviewing flight logs until the desired commanded-to-actual performance is achieved. Both ArduPilot and PX4 have automated tuning routines which are highly recommended, just remember to read the documentation as the maneuvers they command can be quite aggressive if incorrectly used, and typically are suitable only for outdoor use.

Where a motion capture system is being used (such as Vicon or Optitrack), the on-board state estimator will also likely have to be tuned for maximum performance. Motion capture systems have non-negligible delays, and especially for aggressive flights, this delay needs to be taken into account. Motion capture systems are also not 100 % reliable, and the appropriate failsafes should be set to land the aircraft if the motion capture data stream fails.

Tuning the attitude loops of a new design using custom flight controllers can be challenging until one gets a feel for the gain magnitudes. Doing this in flight will likely cause several crashes and significant damage. Tuning in simulation should always be the first option, however the authors have had good experience in either rigging up the airframe such that all translational degrees of freedoms are locked, or for smaller vehicles have one person holding it up while the operator modifies the gains. This technique also allows for fast feedback on the results as the holder can immediately feel the forces and moments applied. Appropriate precautions, like wearing gloves and face protection and the airframe having a solid frame to grab onto will be necessary to avoid any injury.

4. Flying on a Tether

A tether hung from the ceiling is an effective way of preventing an aircraft from hitting the ground during testing. When rigged correctly, a tether has a minimal affects the dynamics of the aircraft, making it quite representative of tether-free flight. The sideways motion of a vehicle is limited by the tether, making large translations difficult, however more 'risky' maneuvers such as tuning and checking controller stability (especially during early development) which require little translation work well (Figure 15). While typically used for multirotor work, given a large enough space, tethers can even been used for fixed-wing flights [15].

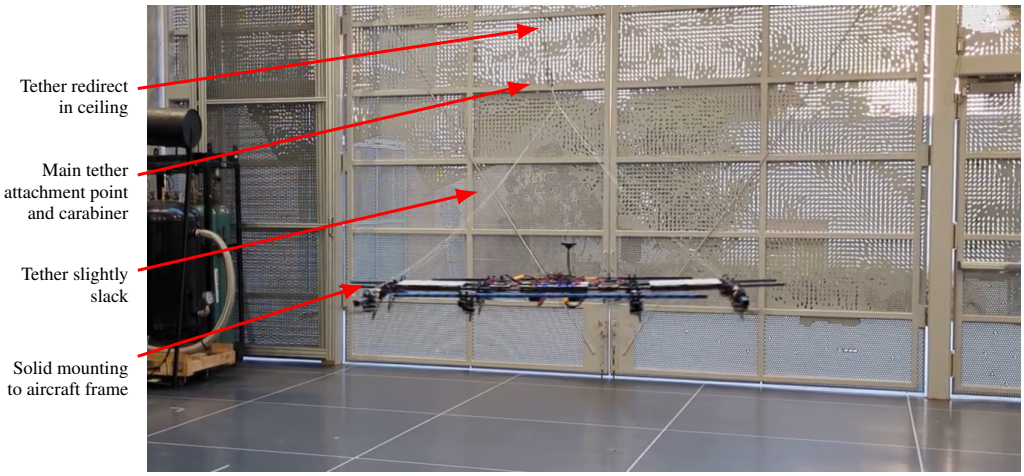

Fig. 15 Flying a control demonstrator developed for the Autonomous Flying Ambulance project [16] on a multi-point tether system to tune control gains.

Setting up a tether requires a sturdy, high pivot point (such as an eye bolt in the ceiling) which a rope needs to be passed through and attached above the centre of mass of the aircraft. Rope with some stretch should be used as this provides a 'softer' landing onto the rope if the motors are abruptly stopped, reducing the forces applied to the airframe. On the side opposite of the eye bolt to the rope, weights should be added to the rope to ensure it does not become too slack and risk being cut by the propellers, though too many weights will cause the tether to pull on the aircraft. The end of the rope should be tied down to provide a 'hard stop' which will catch the aircraft, with the length set such that the stretch of the tether is taken into account.

C. Field Testing

Field testing is an exciting, yet challenging, component of flight testing - all the preparation and development work is put to the test in a high-risk, high-reward environment (Figure 16). Planning is the single biggest challenge when moving out into the field, hence this section will focus mostly on different aspects of planning a field test, rather than the testing itself.

Fig. 16 Field testing represents the true test of system capabilities. It however comes with significantly higher risk and more severe consequences when compared to laboratory testing.

1. Planning a Test

Field trips should be planned around optimising an outcome (testing an algorithm, collecting a specific set of data, etc.). Plans should evolve from what is required to achieve that outcome at the lowest risk, with a focus on envelope expansion - starting small with stability checks and then moving to increasing more difficult tasks and maneuvers.

Setting a series of minimum goals (such that the trip can be deemed a success) and a set of stretch goals (for if time permits) helps keep the testing on task, and can help give a sense of achievement even if not every goal is achieved. When planning the tests to carry out, they should be ordered such that lower risk flights are flown earlier (thus increasing the likely number of tests conducted before an accident), however the relative importance of each test should also be considered. At least some of the flights will likely need to be repeated (parameters weren't set correctly, logging wasn't going correctly, etc.) so this should also be taken into account. Finally, if the outcomes of the tests are unpredictable, developing alternate test plans for cases when following the primary test plan isn't feasible can help the team make informed decision during the test day.

2. Finding a Location

Finding a suitable location for flight testing can be challenging. Flight testing (and flying in general) requires a large amount of unpopulated space with good visibility that is nominally close to home. The airspace above the field must also allow for remotely piloted aircraft operations, so this should also be checked during location scouting.

For development work, university sports grounds work well - they're flat, unobstructed, close to the lab, and in general, large enough to support multirotor or small fixed-wing aircraft work. Local model aircraft fields are another good option - memberships are often inexpensive and other users of the field are usually very interested in the flight tests and are often willing to help out. The club may have special rules around what can and can't be done at the field, so reaching out to the club beforehand to let them know what you're planning to do is a must.

If the project requires flying over a specific type of terrain (such as a forest or mountains), finding a suitable location becomes more difficult. Aerial imagery and local knowledge can help significantly here - look for areas that are sparsely populated, have reasonably easy access, and have clear spaces for take off and recovery.

3. Paperwork

Paperwork is (unfortunately) a necessary part of flight testing. The time required to get the appropriate training and certification, land owner and airspace permissions, insurance coverage, and other pieces of paper should not be underestimated and should be started as soon as possible (lead times of 6 months are not uncommon). Oftentimes it is not clear initially what is required, and in particular for field tests in public places, multiple different entities may need to be worked with to get all the necessary documentation, so actively asking questions when unsure is a must to ensure compliance.

While paperwork like this may be seen as a hindrance, it should instead be viewed as a way to help make an operation safer. Even though the people approving the paperwork may not know much about UAV operations, the paperwork is a means for establishing a documented safety plan that can help expose where safety can be improved, and, that if something goes wrong, it can be shown that everything was done to prevent that accident from happening.

4. Checklisting and Packing

Checklists should be developed to ensure critical elements are not forgotten. Examples where checklists should be used are for checking if an algorithm is ready to be tested on hardware, configuring the aircraft, setting up missions and packing for the field trip. Checklists should also be developed with clear go/no-go criteria, and include items such as checking the weather forecast and for any active NOTAMs. As part of the supplementary materials available with this paper (Appendix A), several checklists are available that are currently used by the authors when supporting their own field trips.

Knowing what to bring on a field trip differs with each project and will typically evolve over time, however plenty of batteries, propellers, cable ties, tools for minor repairs, as well as any approvals and documentation is usually a good start. While it may be tempting to bring the entire lab, there is a limit to what can be fixed in the field, and some fixes are best left to being done in a lab to ensure they're done properly. Having a dedicated set of tools and equipment for field work is recommended as this reduces the risk of items being left behind.

5. Executing the Field Test

At this point, nothing should be new. Though the environment is different, the flights should have already been rehearsed in simulation or in the lab, meaning (ideally) very little has changed. While field test will inevitably unearth new bugs to find and fix, all the software tools have been developed and tested, meaning the test can proceed as efficiently as possible.

Ultimately and above all else, the team should operate safely. While flying UAVs always entails some form of risk and there is always the possibility of a crash or other unexpected event, the team should never accept any unnecessary risk when making decisions. It is better to abandon a test early, fix any problems, and fly again another day than to push an aircraft to the breaking point. Bringing the person in-charge of the project (such as the PI) is often a bad idea as they are more likely that other members of the team to push for results, increasing the risk above levels that otherwise would have been considered unacceptable.

Tests should be repeated several times to account for random variables like turbulence and other errors. A single execution will inevitably lead to disappointment due to data recording problems, insufficient maneuver execution, a wind gust just hitting during the maneuver, or any other imaginable problem. High error bounds in the analysis of a single maneuver can render the results inconclusive.

Due to the higher noise levels of small aircraft, maneuvers typically need to be larger and/or longer, while observing the applicable theory of the phenomenon under test, like non-linearities. Finding the appropriate balance between the two requirements can require significant iteration on the test plans. Sometimes the theory under test can be modified to enable a higher signal-to-noise ratio in the flight test. An effective tool for system identification tasks for example is the use of non-dimensional parameters with measured, variable dynamic pressure used for the calculations, or the use of model structures that are applicable over larger flight conditions instead of the standard linear small disturbance model approximations [3].

D. Determining and Sharing Causes of Failures

Projects that use UAS, especially at a university level, often experience at least one crash (usually more), and usually due to a preventable failure (Figure 17). Even where crashes are expected to happen (low-level controller development, high-speed manoeuvring, and multi-agent work in close formation) and are prepared for, many incidents are still due to

something unrelated like a propeller cutting a wire or a bolt coming loose. Universities are a place of learning, and UAS work is usually low enough consequence that little thought is given to pass valuable lessons on.

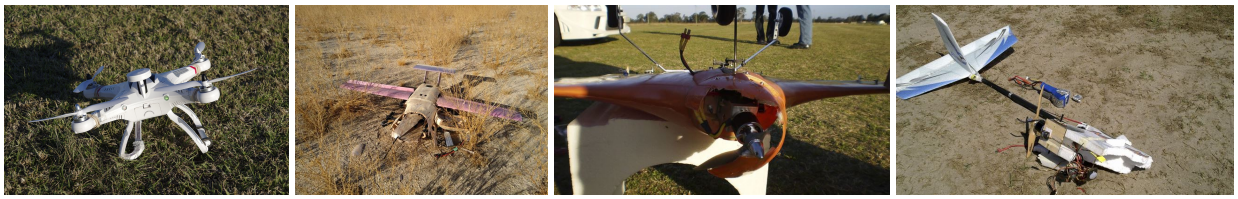

Fig. 17 Crashes. Causes from left to right - not confirming correct throttle levels set on transmitter for UAV handover, not accounting for the loss of thrust with altitude, not capturing the forward shift of the aerodynamic centre during high-thrust, low airspeed conditions for a blended wing body, aero-elastic flutter of the elevator during a dive leading to loss of control.

In person, researchers often share their stories of obscure failures that were not foreseen yet they are rarely written down for a more general audience. It is important that these experiences are shared amongst the community (a great example of a log analysis indicating an aircraft was hit by rocks exploding from a volcano is in [17]), enabling everyone to benefit from the collective knowledge and hopefully reducing occurrences of avoidable failures in the future. As part of this goal, a collection of experiences and notes from the authors are available in Apendix B.

IV. UAS for Use in Teaching

UAVs can be a great enabler for students to learn experientially about aircraft design, flight mechanics and control autonomy. The smaller scale of UAVs offer students a wholistic system experience of flight platforms systems while being manageable in a university environment and operationally in compliance with aviation regulations relating to RPAS (Remotely Piloted Aircraft Systems). The challenge, however, in condensing a vast array of content into a semester, while minimising the support time required of teaching assistants towards students gaining valuable hands-on learning. Furthermore, as academia has a high turn-over rate as people graduate, knowledge transfer can be difficult, meaning a robust, repeatable syllabus must be designed that takes this turnover into account. This section discusses some ideas for integrating UAS into coursework and research training at university level.

A. Course-Work Level

While it is easier to offload as much of the syllabus into simulation as possible, students are typically more engaged when some form of experiential learning can be introduced. For early-year university students who have little knowledge of aerodynamics, building a simple RC kit plane (or glider) can be a good start (Figure 18). This allows students to experiment with parameters such as CG placement and wing design through experimentation, introducing them to the concepts without requiring the complicated knowledge behind it. This also introduces students to practical skills - how to wire servos, how to reverse motor directions, what hardware does what, etc. - giving a solid base and allowing them to tackle more challenging projects in future years.

Fig. 18 Student projects using UAS. The fleet of RC models built by students when constrained from building the kitplane due to COVID-19 restrictions in 2021 (left) and RC model being flown by autopilot (Mission Planner) in simulation (RealFlight 9.5) (right).

As an example, at The University of Sydney (USyd), COVID-19 restrictions in 2021 prevented the Year 1 students from being on campus to work on building a kitplane [18], so each student was sent an RC kitplane which they completed at home. The model type was subsequently flown as a UAV in a SITL simulation setup utilising Mission Planner and RealFlight 9.5[†††]. The feedback on learning outcomes from students was very affirmative, and highlighted the potential to facilitate this base-level learning in UAS into the coursework curriculum.

Design-build-fly (DBF) competitions work well for senior students grouped into small groups, especially when a little twist can be brought into the project (such as requiring an air launching, making the airframe edible, or adding some packaging constraints). These projects can scope creep very quickly, so artificially limiting the design choices students have a little can ensure at least a couple of successful aircraft. One of the easiest things is to limit the electronics (such as given servos, motors and flight controllers), and this also helps with sourcing parts which can sometimes have long lead times. Mission design and planning also a good skill to teach and easy as much of this can be done in simulation, then conducted in the real world with real aircraft.

USyd offers a Coursework-Masters level unit in aircraft design (also available to senior UG students), where the emphasis in recent years has been on producing a flightworthy concept-demonstrator of their designs within one semester. The requirements have been to design a UAV to achieve some contemporary mission, and, working in teams, the students each take on sub-system lead roles towards achieving a design which they can build and demonstrate in flight. In 2021, the RFP (request for proposal) was for a tube-launched UAV for a short-range wildlife monitoring mission (Figure 19, left), and although completed, it was not possible to test fly the technology-demonstrator prototype due to COVID-19 restrictions.

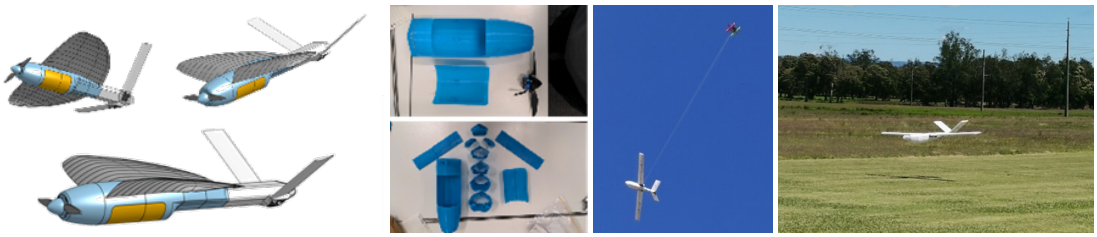

Fig. 19 Examples of UAV designs achieved by students in semester-long class: Tube-Launch, Short-Range Wildlife Monitoring UAV (left) and an Airborne-Deployed Blood Delivery UAV (right), both designed by coursework students

In 2022, the RFP was presented by an industry partner engineer to develop a flight-deployed UAV from a typical transport aircraft to deliver a unit of blood in an emergency situation. The feedback from students was that the final weeks they spent in the UAV-Lab building (and often rebuilding) the airframes were the most beneficial in their aircraft-design learning experience because they could finally relate their prototype to the analyses and modelling they had done prior. Figure 19 (right) shows the AirCrane [2] facilitating air-deployment, and the successful landing of one of the designs.

Learning opportunities are not just limited to aircraft design, and can be extended to practical demonstrations of guidance, navigation and control (GNC) of UAVs. With the ready-availability of the low-cost RoboMaster Tellos[‡‡‡] and Bitcraze CrazyFlie[§§§], it has become possible for coursework students to learn complex UAS operations and controls in an flight-operational environment to provide a more enhanced learning experience and outcomes over the pure-simulation setting. Integrated the UAVs with a motion capture camera system for position estimates, students can experience working within the relatively-complex field of GNC using relatively light, inexpensive hardware that is crash tolerant. In 2022, the small class of USyd students developed a simple Drone-Corridor environment with geo-fences and control algorithms to maintain safe flight operations in a potentially cluttered environment. They also developed elements of collision avoidance, as well as elements of formation-flight swarming. Such low-cost UAS offer the feasible potential to integrate more experiential learning modules in coursework curriculum to develop operational skills to enhance research using UAS.

Teaching piloting skills as part of coursework is challenging as it is time, resource, and instructor intensive, however even a small amount of exposure however can help students understand the practical aspects of operating a UAS. Keeping

[†††]Mission Planner - `https://ardupilot.org/planner/`, RealFlight 9.5 - `https://www.realflight.com/products/rf95/index.php`, Using Mission Planner with Realflight - `https://ardupilot.org/dev/docs/sitl-with-realflight.html`

[‡‡‡]RoboMaster TT - DJI - `https://www.dji.com/robomaster-tt`

[§§§]Bitcraze Crazyflie 2.1 - `https://www.bitcraze.io/products/crazyflie-2-1/`

the piloting simple and emphasising accurate control of the UAV is the key. Using a COTS system that is relatively easy to pilot (such as a DJI quadrotor), students can be taught to fly box patterns, vertical circles, figures of eight, etc., within a relatively small amount of time.

B. Research Training

For undergraduate-level research where there is more time to develop a system, integrating more advanced controls algorithms and planning can be achieved. Integration of autonomous control via a companion computer (and the associated networking, comms, etc. that is required) comes with a steep learning curve, however these skills are often transferable to robotics in general, setting students up well for their careers. As with coursework projects, some form of scope limitation is important, and it is suggested that a project only has one aspect that requires significant development (such as airframe design, controls algorithms, motion planning algorithms, etc.).

Graduate-level researchers have more skills and more time to complete a project, and hence the projects usually grow larger and more complex. One key recommendation is to develop a robust set of tools early on in the project that can be used to support the in-depth research and analysis that is usually required. UAS are complicated systems, and while cutting corners early on can lead to rapid progress, long term it is very detrimental as the system will inevitably starts to fall apart and the initial work will need to be re-done.

V. Conclusion

This paper has presented many aspects of designing and operating UAVs for research including instrumentation, flight controllers, airframes and methodologies for conducting safe and efficient flight test campaigns. It has presented many of the considerations that need to be taken into account at all phases of the design, from how to choose instruments and software packages to flight platforms and testing locations. The key take away is that a methodical, well planned system based on the requirements set out by the project is key. These recommendations are coupled with insights from the authors' personal experiences with working with UAS, addressing many of the little 'gotchas' every flight project experiences. Finally, this paper has presented some example in which UAVs have been successfully integrated into course-work at a university level, delivering an experiential curriculum that has shown high levels of student engagement.

References

[1] Wilson, D., Goktogan, A., and Sukkarieh, S., "Guidance and Navigation for UAV Airborne Docking," *Robotics: Science and Systems XI*, Robotics: Science and Systems Foundation, 2015. https://doi.org/10.15607/rss.2015.xi.017.

[2] Anderson, M., Lehmkuehler, K., and Wong, K., "Flight Experimentation Towards Enhanced UAV Capabilities – The Multi-Rotor Air-Crane," *AIAC17*, Melbourne, Australia, 2017.

[3] Lehmkuehler, K., "A Direct Comparison of Small Aircraft Dynamics Between Wind Tunnel and Flight Tests," Phd thesis, The University of Sydney, 2016. URL http://hdl.handle.net/2123/16511.

[4] Gregory, J., and Liu, T., *Introduction to Flight Testing*, Wiley, 2021. https://doi.org/10.1002/9781118949818.

[5] Pontzer, A. E., Lower, M. D., and Miller, J. R., "Unique Aspects of Flight Testing Unmanned Aircraft Systems," Tech. Rep. RTO AGARDograph 300, AG-300-V2, NATO RTO, 2010.

[6] Dauer, J. C., Adolf, F.-M., and Lorenz, S., "Flight Testing of an Unmanned Aircraft System – A Research Perspective," *STO Meeting Proceedings*, 2015. STO-MP-SCI-269.

[7] Williams, W., and Harris, M., "The Challenge of Flight-Testing Unmanned Air Vehicles," *Systems Engineering, Test and Evaluation Conference*, Sydney, Australia, 2002.

[8] Anderson, M., "A Methodology for Aerodynamic Parameter Estimation of Tail-Sitting Multi-Rotors," Phd thesis, The University of Sydney and Université Libre de Bruxelles, 2018. URL http://hdl.handle.net/2123/18858.

[9] Lanzon, A., Freddi, A., and Longhi, S., "Flight Control of a Quadrotor Vehicle Subsequent to a Rotor Failure," *Journal of Guidance, Control, and Dynamics*, Vol. 37, No. 2, 2014, pp. 580–591. https://doi.org/10.2514/1.59869.

[10] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A., "ROS: An Open-Source Robot Operating System," *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, 2009.

[11] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W., "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, Vol. 7, No. 66, 2022, p. eabm6074. https://doi.org/10.1126/scirobotics.abm6074, URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[12] Bottasso, C. L., Leonello, D., Maffezzoli, A., and Riccardi, F., "A Procedure For The Identification Of The Inertial Properties Of Small-Size Uavs," *XX AIDAA Congress Milano*, 2009.

[13] Lehmkuehler, K., Wong, K. C., and Verstraete, D., "Methods for Accurate Measurements of Small Fixed Wing UAV Inertial Properties," *Aeronautical Journal*, Vol. online, 2016. https://doi.org/10.1017/aer.2016.105.

[14] Lehmkuehler, K., and Anderson, M., "kVIS3 – Kai's Data Visualisation And Manipulation Tool (3rd Edition)," *Software Program*, 2022. URL https://github.com/flyingk/kVIS3.

[15] Airbus Newsroom, "Freely Flapping Wing-Tips on Future Aircraft Just Took a Leap Forward," online, 2020. URL https://www.airbus.com/en/newsroom/stories/2020-10-freely-flapping-wing-tips-on-future-aircraft-just-took-a-leap-forward.

[16] Tang, E., Spieler, P., Anderson, M., and Chung, S.-J., "Design of the Next-Generation Autonomous Flying Ambulance," *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics, 2021. https://doi.org/10.2514/6.2021-1514.

[17] Wood, K., Liu, E. J., Richardson, T., Clarke, R., Freer, J., Aiuppa, A., Giudice, G., Bitetto, M., Mulina, K., and Itikarai, I., "BVLOS UAS Operations in Highly-Turbulent Volcanic Plumes," *Frontiers in Robotics and AI*, Vol. 7, 2020. https://doi.org/10.3389/frobt.2020.549716.

[18] Wong, K., "Experiential Learning for Year 1 Aeronautical Engineering," *12^{th} Annual Conference of the Australian Association for Engineering Education (AaeE 2001)*, Brisbane, Australia, 2001, pp. 13–18.

A. Resources

Online documentation is generally very good for the open source tools presented here, however getting started can be a daunting task, especially as there is a lot of reading and many different elements that need to work together before a first flight. As part of working with students studying UAS over many years, the authors have developed a series of tools and example codes to help beginners get started. The authors have also developed checklists, templates, and other documents that can help with planning flight tests, from questions pilots should ask before flying new code to flight logs used to track each flight. The main landing page for this work is available the UAS Tools Index page available at `https://github.com/AndersonRayner/uas_tools_index`. Some of the key resources are linked as part of this repo are:

- *kvis* - `https://github.com/flyingk/kVIS3`
  - An open source, MATLAB-based platform for UAV data analysis that features advanced signal processing, filtering, spectrum analysis and automatic maneuver extraction. kvis currently supports ArduPilot and PX4 data files, with support for additional data files easily added using the examples provided.
- *ardupilot_sitl_example* - `https://github.com/AndersonRayner/ardupilot_sitl_example`
  - An example setup for running the ArduPilot SITL simulator such that it can be used with ROS in a repeatable manner across different machines.
- *ros_uav_interfacing* - `https://github.com/AndersonRayner/ros_uav_interfacing`
  - Scripts that demonstrate interfacing ROS with a UAV using MAVlink and mavros, including setting low-latency connections and data stream rates.
- *ros_uav_command* - `https://github.com/AndersonRayner/ros_uav_command`
  - Example scripts ROS scripts for commanding MAVlink-enabled UAVs via MAVlink.
- *arduino_mavlink* - `https://github.com/AndersonRayner/arduino_mavlink`
  - A library for providing examples of using a MAVlink connection with Arduino. This can be used to communicate between an Arduino and MAVlink-based flight controller, and can also be used to share the telemetry connection between the two.
- *uas_templates* - `https://github.com/AndersonRayner/uas_templates`
  - A collection of templates for checklists, flight logging, etc. that have been found to be useful when planning field trips.

B. Debugging, Hints, and Notes

A. Hardware Integration

Debugging integration issues of a UAS can be challenging, and typically requires a lot of trial-and-error (or experience) to find the root cause/s. Below are listed some of the more common issues that the authors have encountered:

- Radio compatibility/interference, especially affecting the GPS antennas. GPS is a very weak signal and is hence easily affected by other systems on the UAV, especially as high-powered transmitters are often used on the aircraft to increase telemetry range.
  - It is important to do the debugging of any GPS issues outside with a good view of the sky so that other factors that affect GPS signals (such as multi-pathing) are minimised.
- PWM commands (for controlling motors and servos) are vulnerable to noisy RF environments. This typically manifests in random servo movements, especially when transmitting radios are bought close to the servo.
  - Shielding can help, however it is often easier to replace the servos for a different type which can often make the issue go away.
  - Many modern flight controllers output 3V3 command signals, so boosting the PWM voltage to 5V can also help the servo with noise rejection (just make sure your peripherals can handle the higher voltage signals).
  - Modern ESCs will often support Dshot for commands. Dshot is more robust that PWM, does not require calibration, and is checksummed so bad frames can be immediately rejected. Where possible, it is recommended to use Dshot for motor commands as the authors have experience strong enough interference on the PWM lines that motors have temporarily shut off mid-flight.
  - CANbus servos and ESCs are another option for good noise rejection characteristics, however the significantly higher cost of these components puts them out of reach of many projects.

- USB3 causes very broad spectrum noise signals, typically affecting (at a minimum) the GPS. This is easily diagnosed by starting the system without USB3 running, turning the USB3 sensor on, and observing the sensor degrading.
  - Fixing USB3 noise is difficult, and thus far the authors still very much apply a trial-and-error debugging technique. Shielding cables (especially around the connectors) with copper tape connected to ground works somewhat, however introducing distance between any USB3 lines and other sensors thus far has been the most effective.
  - USB2 does not appear to have the noise issues that USB3 has, so if it is acceptable to run the sensor at the lower data rate, placing a USB2 hub on the system to force the sensor to use USB2 is a quick fix.
- Use battery beepers to protect the batteries, especially during development. It is easy to get side tracked and forget about the battery, leaving it to over discharge and cause permanent damage. Computers can drain the battery at variable rates, depending upon the load on the processor, so simply setting a time to change the battery may not be enough.
- Tie down cables. UAVs are high-vibration environments and may cause wires to move around. Over the short-term, wires may move significantly enough to be cut by propellers, and over the long-term, any movement causes fatigue and premature failure.
  - USBs in particular don't like the vibrations of UAVs and become rather unreliable if allowed to move around. Screw-in USB terminals work well, though are often not available, so 3D printing a custom bracket with a cable tie to hold the USB cable can help align the USB port and ensure a reliable connection.
- Learn to read log files from your flight controller. Flight controllers record a lot of data about the health of the state estimators, how well the system is tracking commands, the sensor states, and a multitude of other data. Many issues can have multiple potential causes, so it is important to be able to read these logs to effectively diagnose issues.
  - Make a habit of storing the onboard logs from the flight controller after every day of testing. This way, if anything happens, all the logs from the aircraft are available and problems can be traced back through the history of the aircraft.
  - The online community around ArduPilot and PX4 is generally pretty good about helping with diagnosing issues from logs, so reading these forums is often a good learning experience as to what to look for.
- Make sure to do a power budget for any external sensors, especially those connected directly to the flight controller or the USB ports of a computer. Sensors pull varying current loads, especially during acquisition, and this can lead to very intermittent brown-outs and resets of sensors. The intermittent nature of the behaviour means the root cause is very difficult to track, however it is often a power problem.
  - Where possible, use external regulators to power sensors. Use an extra port on the GPS + I2C breakout boards which are supplied with many flight controllers to provide power from a separate regulator. Just make sure to remove the 5V power line supplied from the flight controller to prevent the two regulators from fighting each other
  - USB power allocation is often overlooked, and for systems that have multiple sensors connected via USB, the computer's USB rail is unlikely to be able to adequately power everything (especially if the computer is designed to be low power). Powered USB hubs can solve this problem by providing the USB power from an external source - just cut the cable that comes with the hub, solder on a regulator, and use that to power the hub.

B. Field Testing

Operations has its own set of interesting quirks, often not caught during laboratory-based testing. Below are some of the common and notable experience the authors have had in the field.

- Robotics and flight controllers typically have a different definition of 0 degrees heading. While robotics uses an ENU frame (meaning 0 deg heading is east), flight controllers typically use a NED frame (meaning 0 deg heading is north). This is a potential point of miscommunication between the pilot (who is used to NED) and the coders (who are used to ENU), so communications should make clear which reference is being used. If the aircraft is lined up incorrectly for launch, it may aggressively try to attain the correct heading centimetres above the ground, potentially leading to the landing gear being caught and a subsequent crash.
- Geo-fences are typically defined from the take-off location of the aircraft. In the case of a moving base (such as when on a boat), care must be taken that the geofence is large enough to account for the drifting location. With

most flight stacks, the geofence will continue to be enforced even during a return-to-home, meaning the aircraft will never be able to make it back to the operator.
- Always double check that the pilot can take control of the aircraft, and that there is an e-stop (or flight termination) implemented. Incorrect signs, an easy mistake to make, can cause an aircraft to quickly fly away leaving little to no time for the operator to stop their code.
- Keep batteries warm before flight when operating in cold environments. Battery performance (both power and total energy available) degrades significantly with decreasing temperature, directly affecting the flight time of a vehicle.
- Birds can be particularly territorial of airspace, and the authors have experienced attacks on aircraft by birds during operations. Not only can the birds damage the aircraft, but more importantly spinning propellers can cause significant injury to the animal. If problems are encountered, try flying at different times of the year (in particular, when the birds aren't nesting), modify flight plans to avoid areas with bird life, or simply find an alternate site.
- If doing night time operations, make sure to contact the local police department and let them know what you're doing. The bright lights and odd flight patterns mean people may mistake your aircraft for a UFO. The last thing you need is for red and blue flashing lights to approach your area of operations looking for little green men...

C. Dynamic Propeller/Motor Balancing

Propellers should be dynamically balanced in combination with the motor using an accelerometer attached to the primary airframe structure where possible to minimise vibrations. Using cable ties around the bell of the motor (Figure 20) which can be rotated to find the optimal position works well for this task, and can make the difference between success and failure of a flight test.

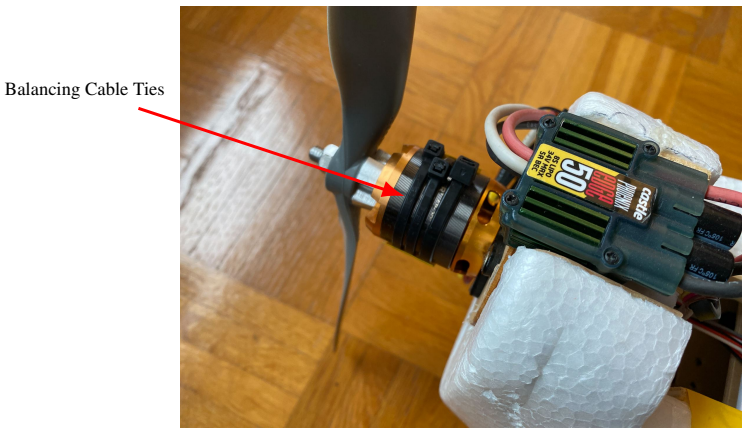

Fig. 20 Dynamic balancing of a motor-propeller combination with cable ties

This dynamic balancing technique is very easy and quickly done, but is specific to each propeller/motor combination and the propeller installation position on the motor, so make sure to mark the propeller position. As propellers are easily damaged, bring a fully balanced propeller/motor combination as spare parts, or keep detailed records/photos of the cable tie number, type and position for each replacement propeller.