

A Reconfigurable Flight Management System for Small-Scale Unmanned Air Systems

Ella M. Atkins,^{*} Ryan D. Eubank,[†] and Andrew T. Klesh[‡]
University of Michigan, Ann Arbor, MI 48109

Unmanned air systems are becoming increasingly pervasive in academia as well as industry. Aerospace research and education have to-date focused on the fundamental aerodynamics, structures, and control technologies required for these systems, typically relying on commercial avionics packages to provide the hardware and baseline software needed for unmanned vehicle flight testing. This paper describes an ongoing effort at the University of Michigan to design, implement, and adapt to multiple platforms an open-source, reconfigurable flight management system. Emphasis has been placed on modularity and extensibility, resulting in a system that equally addresses Aerospace education and research challenges. This paper overviews the avionics and software design, implementation, and testing processes completed to-date. Use of the University of Michigan Flight Management System (UM-FMS) to support education in software engineering, embedded instrumentation, and multi-layer control is discussed, along with adaptation to specific flight vehicles.

I. Introduction

The proliferation of unmanned air system (UAS) platforms has spawned a wide range of autopilots ranging from low-power microprocessor units to sophisticated avionics systems that support complex, long-duration missions. Typical UAS autopilots^{viii,2} include the inertial navigation sensors, servo interfaces, and software required to capture real-time flight data and autonomously fly through a prescribed waypoint sequence. Popular microprocessor-based systems include products from Cloud Cap (<http://www.cloudcaptech.com/>), Micropilot (<http://www.micropilot.com/>), and Procerus (<http://www.procerusuav.com/>). Such commercial off-the-shelf (COTS) units enable researchers and application specialists to bypass costly development efforts, moving rapidly from airframe integration to autonomous waypoint flight. Although manufacturers accommodate a wide array of applications, missions may require capabilities that exceed the supported sensing and microprocessor capabilities. In such cases, three options are available: (1) Rely on ground-based processors and/or sensors to supplement onboard capabilities, (2) Augment the COTS autopilot with distinct onboard processing and/or sensor systems, or (3) Develop a custom onboard avionics architecture that does not use a COTS autopilot.

All three options have been elected for existing UAS. UAS often downlink video and rely on capable ground-based processors to identify and uplink targets of interest or may use ground-based navigation systems (e.g., Vicon cameras³) for GPS-denied environments (option 1). Optic flow algorithms have recently migrated onboard (option 2) to enable computationally-tractable relative motion characterization. Option 3 has been adopted by researchers studying fundamentally different strategies of control and/or communication (e.g., MAV swarms⁴ that require high inter-vehicle communication bandwidths). In short, the choice of autopilot “option” depends on mission goals, vehicle specifications, and environmental constraints.

University laboratories undertake dual objectives: to achieve fundamental and system-level research goals while also educating a broad range of students in theoretical and practical concepts. Certainly, the research goals themselves can exceed COTS autopilot capabilities. With focus on navigation or control, option 1 or 2 is likely the best choice: augment the system with better sensors or control laws but capitalize on the hardware and software infrastructure offered by the COTS products. Even with focus on mission planning or guidance a “COTS augmentation” option may be most efficient: rely on an existing autopilot for waypoint following with a more capable processor added to perform computationally-intensive planning activities. From a system-level perspective, such solutions are faster and less labor-intensive than ‘custom’ system development. As the number and complexity of augmentations increases, however, overall system complexity may ultimately be at least as complex with a specialized COTS

^{*} Associate Professor, Aerospace Engineering Dept., University of Michigan, Associate Fellow

[†] Graduate Research Assistant, Aerospace Engineering Dept., University of Michigan, Student Member

[‡] Graduate Fellow, Aerospace Engineering Dept., University of Michigan, Student Member

autopilot as without it. Additionally, COTS products provide a layer of abstraction above embedded sensors, servos, and real-time software, eliminating motivation for students to learn how embedded systems are designed, developed, debugged, and validated. Such an architecture trains students to “black box” avionics sensors and software – a paradigm they will carry into their future careers. This is a suboptimal education strategy since Aerospace systems will increasingly rely on embedded and networked systems for efficient, safe operation.

This paper describes the University of Michigan’s open-source Flight Management System (UM-FMS) developed to directly address the education and extensibility challenges difficult to achieve with COTS products. Our “custom” software and hardware is used in the Aerospace classroom, an elective tutorial series, and multiple research laboratories. Undergraduate and graduate students have participated in the development of UM-FMS, from ground-station graphical user interface (GUI) written in QT to the embedded C code in Linux-based and microprocessor environments. Emphasis is placed on intuitive displays at the highest levels to protocols for communication, sensor data acquisition, and actuation at the lowest [microprocessor] level. Synchronization and real-time execution constraints are emphasized throughout to ensure consistent data sufficiently recent to characterize current state. Within a plug-and-play framework, students also write and test guidance and control algorithms with the goal of flight testing their code as well as testing in simulation.

Below, we describe the application of our flight management system to two diverse UAS applications, demonstrating its flexibility and extensibility. The Flying Fish seaplane UAS aimed at persistent ocean surveillance was the first platform to be fully-automated with support from our flight management software.^{5,6,7} This platform requires a system capable of autonomously initiating and executing multiple flight “hops” across a predefined watch circle, and is currently being extended to support long-duration deployment with solar energy harvesting. The SolarBubbles student team at the University of Michigan has also adopted our flight management system, with the goal of approaching the longest-duration continuous flights with a lightweight (under 15 pound) airframe and solar recharge capability.⁸ Other platforms, ranging from a foam MiG-27 (modified FQM-117B target drone) to an aerobatic Funtana, have adopted our flight management package for education and research purposes. Below we describe the architecture, software, and implementation of our flight management system. We then summarize its deployment to-date in for research and education purposes. We conclude with ongoing efforts to synergistically extend the FMS and use the FMS to enhance the computational literacy of current and future generations of University of Michigan undergraduate and graduate students.

II. Avionics System

Figure 1 shows a high-level diagram of the flight management system avionics. The Gumstix processor, alone or networked to a second Gumstix “stack”, is mounted atop the Atmel-based Robostix microprocessor. Whether Connex, Verdex (the current Gumstix), or alternative processor, the objective of carrying a processor of appreciable computational power is to maximize the ability to support a suite of flexible, reconfigurable onboard computations rather than rely on a persistent datalink. The Gumstix processor runs an embedded Linux kernel, facilitating development and debugging of complex multi-threaded applications with standard operating system tools. Communication is achieved through TTL (convertible to RS/232) serial ports, with an 802.11b wireless link available at a cost of nontrivial power use to speed data transfer. Robostix, an Atmel ATMega128, is programmed in C with an AVR cross-compiler, relying on interrupts as well as a single primary execution thread to synchronize read/write operations. As shown in Figure 1, the processing architecture is flexible and extensible to multiple Gumstix/Robostix stacks should I/O or processing requirements exceed the capacity of a single stack. A spread spectrum modem connects the avionics with a ground station laptop. Data is displayed on the ground station but stored onboard to facilitate high-rate acquisition and synchronized time stamps.

As shown in Figure 1, our system has the option of using COTS or custom subsystems integrated with Gumstix/Robostix. Initial development efforts relied upon the Microbotics MIDG-II inertial navigation system (INS) that communicates GPS, raw attitude sensor, and filtered state estimates to Gumstix via serial link. The Microbotics Servo Switch Controller (SSC) also provides the special-purpose function of switching between radio-control (R/C) pilot and flight computer servo commands. This design is a compromise that enables customization of all guidance and control software, as well as custom integration of all additional sensors (e.g., air-data), but that eliminates potentially time-intensive INS sensor calibration and filtering steps. Because COTS INS are costly and student education is also a primary goal of UM-FMS, we also have developed a micro-sensor IMU package that is

manually programmed, enabling a student to develop and test calibration and filtering algorithms and compare estimates with the MIDG-II “benchmark” data available on the same system. The custom micro-sensor package can also be used alone in flight tests where risk or payload capacity would preclude use of the costly COTS package.

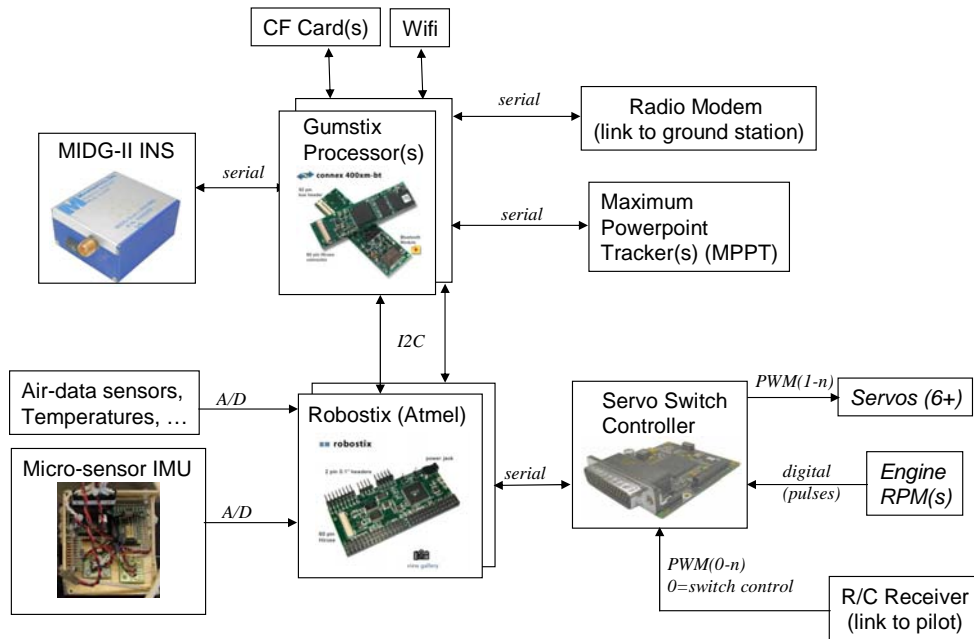


Figure 1: Flight Management System Hardware.

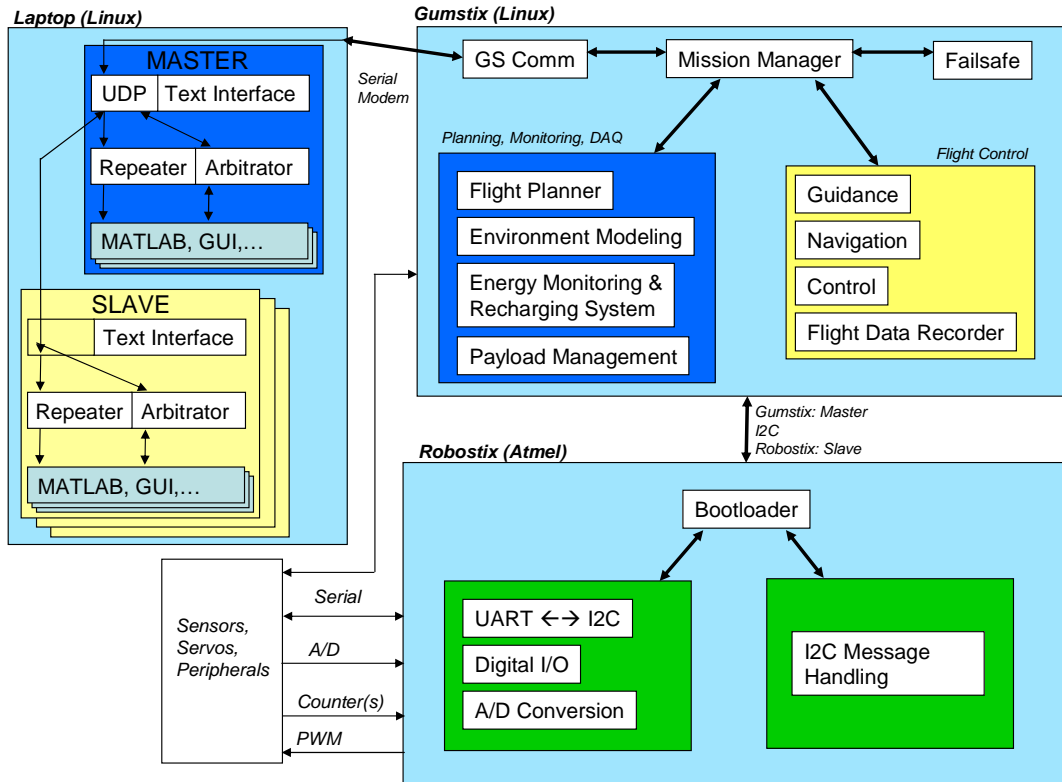


Figure 2: Single-Processor Flight Management Software Architecture.

III. Software Design and Implementation

Figure 2 shows the flight management software processes distributed across Gumstix, Robostix, and ground station processors. The Gumstix software is organized with separate threads for communication, data acquisition, navigation and control, and flight planning. A simple state-based mode manager switches between operational modes before, during, and after flight, including transitions to and from failsafe modes.

The onboard Gumstix software is organized with separate threads for data acquisition, navigation and control, flight planning, and communication. The software is built around a core set of global data structures, data manipulation functions, and communicated messages common to every element of the software system from ground station to airframe. The data structures are populated by separate data acquisition threads handling sensors connected to either the Gumstix board or the Robostix daughter card, via an I²C interface. In this framework, additional sensors and computer boards integrate seamlessly with the data acquisition system. Data structures are updated by the navigation, control, and planning utilities required to handle flight operations.

The state-based mode manager switches between operational modes before and during flight and handles detected exception conditions. Rule-based logic provides appropriate behaviors, stabilizing controllers, and reference trajectories to handle a given set of operational conditions and mission requirements. In the event of a detected failure or out-of-envelope flight condition the mode manager sets failsafe parameters and adopts control laws that maximize safety. To-date, mode switching logic has been based on performance specifications garnered from human-piloted flight data or analytical models (e.g., stability derivatives), adopting conservative flight envelope estimates and mission-specific flight profiles. The communications thread handles serial communication via a wireless modem, though the modularity of the threaded structure allows flexibility in communication protocol and link type (e.g., serial, wifi). Onboard storage to nonvolatile memory is managed as a function of real-time execution requirements. If required control loop rates are higher than that supportable with real-time nonvolatile memory read/write delays, the read/write delay is avoided by storing data to memory until the system controller is sufficiently stable or dormant to operate at lower response rates.

The Robostix daughterboard runs an Atmel ATMega128 processor to handle communication and data collection. This smaller daughterboard acts as a switch and communication translator for inter-processor communications (I²C) and several UART lines. It also serves as an analog-to-digital (A/D) converter, provides multiple digital input/output (DIO) lines and allows for programming of any additional Atmel microprocessors on the I²C network. The software running onboard Robostix initially runs a boot-program to allow for in-circuit programming, followed by an interrupt based main program. Analog data capture, UART communication receipt and circular-buffer filling all trigger interrupts to signal communication or processing. All analog and UART data is recorded in circular buffers and segmented into 33 byte sequences for transmission over the i2c network. Current research is investigating how to improve the speed of the I²C network by decreasing the overall number of messages sent.

Figure 3 shows the ground station system, including laptop, extended-range serial modem, wireless router, power system, and R/C transmitter. The QT-based ground station graphical user interface (GUI), shown in Figures 4 and 5, is used to display uplinked flight data in an easily-comprehensible format and to facilitate user specification of the mission (e.g., waypoints) as required. The ground station (GS) software is centered around a compact high-reliability text-based interface and data handling utility. The text-based interface features formatted displays for at-a-glance status monitoring and a command-line interface to the UAS. The data distribution system utilizes a trickle-down architecture wherein the master computer executes separate threads to arbitrate communication with the UAS and to broadcast information downstream to slave nodes. Similarly, each slave node executes a master thread arbitrating upstream communication to the master node and a repeater thread to forward data to local client processes. Leveraging this structural symmetry allows every machine in the GS network to share common interface

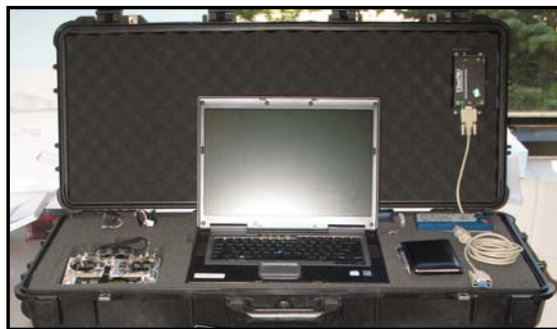


Figure 3: Ground Station Hardware.

code. Atop the core software several larger software packages provide additional GS utility including real-time state-variable plotting via an interface to MATLAB and a user-customizable graphical user interface (GUI). The GUI was coded using the QT open-source cross-platform graphical development software from Trolltech. The GUI has several different display modes which include a traditional heads up display (HUD), an overhead GPS map, as well as an engineering display showing key system parameter in a compact format. The HUD provides limited piloting capabilities for beyond-line-of-sight operation and the overhead map features utilities for building waypoint-based flight plans. The extensibility and open-source nature of the code facilitate use in an educational environment as students can explore, update, or upgrade the system as needed. For example, students are currently working to embed airborne video streams into the HUD and to interface the overhead maps with online satellite imagery and mapping utilities.

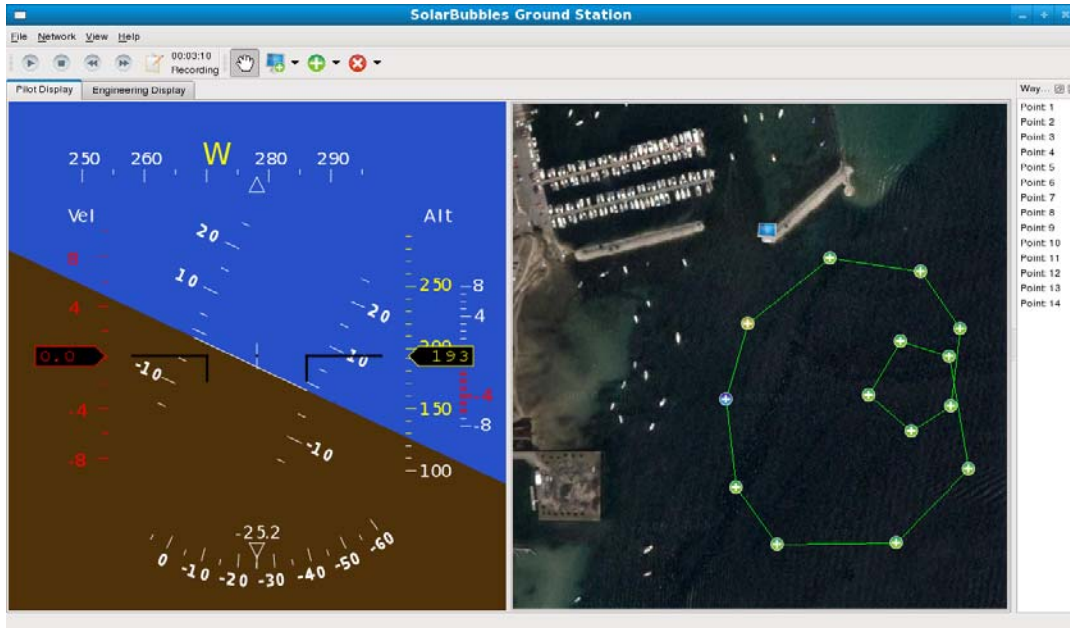


Figure 4: Main Ground Station GUI: Heads-up Display (HUD) and Overhead Area Map.

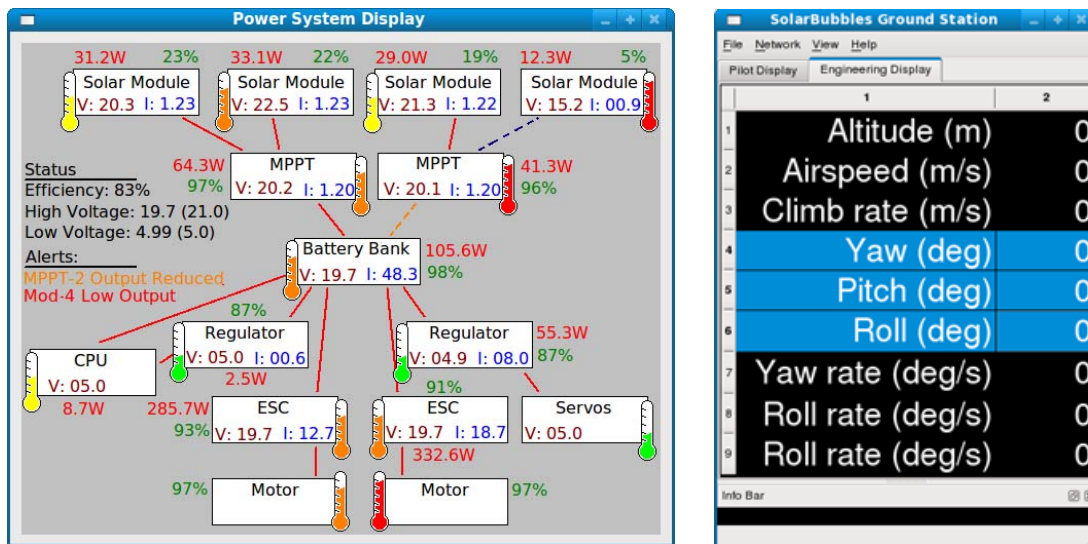


Figure 5: Ground Station Auxiliary Displays

IV. UAS Deployment and Performance Summary

Figure 6 shows three UAS platforms in which our flight management software has been integrated to-date. The Flying Fish aircraft^{5,6,7} (Figure 6b) has relied on our flight management system software and is the platform most extensively tested to-date with UM-FMS. For Flying Fish, avionics components are mounted on both sides of a composite board and affixed in a cylindrical waterproof housing. Flying Fish achieved autonomous, self-initiated flight across a prescribed watch circle in October 2007. Efforts are now focused on extending to longer-duration deployments and also integrating solar energy harvesting capability in a second-generation platform currently under development. Specifics of the Flying Fish guidance and control algorithms are provided in Eubank et. al.⁷, and use of flight data for estimating vehicle power usage and environmental conditions is described in Meadows et. al.⁶

The HUI aircraft⁸ (Figure 6a) developed by the University of Michigan's SolarBubbles student group is based on a high-efficiency powered glider design and has the goal of long-endurance flight. The SolarBubbles team is currently constructing a new airframe (not pictured) with embedded solar cells, and will use UM-FMS to manage onboard energy as well as autonomously guide and fly the aircraft through a repeated waypoint pattern designed from energy-optimal guidance given operating area constraints. In HUI, the UM-FMS was hung on a custom pod below the airframe given the limited space constraints in the narrow fuselage and thin wing sections. The Yellowtail aircraft is being designed to support UM-FMS avionics within the wing and fuselage to decrease drag.

In the Flying Fish and SolarBubbles applications, the Gumstix / Robostix embedded avionics architecture was adopted. Whether executing in a single or multi-threaded mode, the guidance, navigation, and control loop is currently limited to an execution frequency of just under 10 Hz due to active-wait I²C communication delay (Gumstix master awaiting Atmel slave response). This rate has been found sufficient for our stable airframes, although our COTS INS is capable of providing full state estimates at a rate of up to 50 Hz.



a) SolarBubbles HUI.



b) Flying Fish (first-generation)



c) Funtana (forward/slow/hovering flight)

Figure 6: Testbeds for the University of Michigan's Flight Management System (UM-FMS).

A COTS Funtana aerobatic aircraft (Figure 6c) relies on a version of UM-FMS adapted to an embedded PC/104 processing architecture. With the goal of accurate control of slow and hovering flight based on an aerodynamic data system augmentation to inertial feedback,¹⁰ the embedded Diamond Systems Athena computer was selected to enable higher-resolution A/D conversion and faster sampling rates in the kHz range. Note that a PC/104-based avionics system provides more computational power and higher-accuracy data collection capability but requires substantially more cargo volume and weight than the Gumstix/Robostix design thus is not appropriate for a small-scale, long-endurance platform.

V. UM-FMS for Aerospace Education

A primary goal of the UM-FMS is to offer Aerospace students a software implementation that exposes them to the spectrum of concepts typically found in embedded vehicle control systems, and that offers students a package in which they can easily augment one or more software elements without breaking the entire product. Below, we present background in the state of Aerospace Engineering computing education at the University of Michigan and then discuss how UM-FMS contributes to this effort.

A. The State of Computing in the UM Aerospace Curriculum

Both undergraduate and graduate Aerospace Engineering curricula at the University of Michigan focus, by tradition, almost exclusively on the fundamental engineering topics of structures, gas dynamics, and dynamics and control. However, it is clear that the modern engineer requires increasingly higher levels of computational competency to develop and apply their education. This trend has resulted in the introduction of increasingly-advanced computational tools to address problems such as computer-aided design (CAD) or simulation (e.g., with Matlab), but the fact remains that typical introductory coverage of these tools provides little insight into the fundamental algorithms and underlying computations used to provide the functionality these tools offer. Despite the migration of computational tools into the engineering classrooms, the basic curriculum of most institutions has been slow to change. For example, the current undergraduate Aerospace curriculum at the University of Michigan requires only a single 4-credit computing course for freshmen (Engineering 101). The course presumes no programming background and focuses on procedural algorithm development in the context of C++ and Matlab. Since University of Michigan undergraduates do not declare their major until the second year, Engineering 101 is currently taught in six 225-seat sections mixing students of all engineering majors and all preparation levels. Although the College of Engineering places significant importance on this course, translating to comprehensive high-quality lecture, lab, and out-of-class support, Engineering 101 is insufficient alone to prepare students for careers involving challenging computational analysis or embedded control applications. This is especially true given the absence of standardization in K-12 computer science curricula, resulting in an extremely broad distribution of skill sets in freshmen of all engineering majors. To meet the needs of the top 10-15% of the engineering freshman computational literacy distribution, the University of Michigan has recently created an “accelerated” freshman programming option available by self-selection guided by a “gateway” exam. The students in the accelerated course progress through object-oriented programming, engineering analysis, and embedded systems topics at a pace commensurate with some programming background and natural intuition for algorithm design and implementation. The remaining 85-90% of our engineering students, however, learn the fundamentals of procedural programming in C++ and Matlab but have only the one semester of introductory exposure.

There is indisputably a critical need for engineering students to reinforce and strengthen their computational skills through long-term use. Given established curricula that resists change both within and across course offerings, how, then, can such skills be repeatedly introduced and reinforced? The University of Michigan currently offers a senior elective course, Flight Software Systems, that serves as a continuation of Engineering 101 with emphasis on object-oriented programming and fundamental computing concepts beyond the code and algorithm. This course also requires students to complete substantive programming projects aimed at building or significantly extending Aerospace-relevant embedded or analysis applications. We also are offering a graduate course in Aerospace Information Systems aimed more at fundamental theory with some application-oriented content. Long-term we hope to better integrate in-depth computational content to existing courses, but at the algorithmic and computational thinking level, *not* at the “cookbook” level (e.g., using an ODE solver in Matlab or running a LabView program to capture experimental data).

Our current best-practice is to first capture as many students as possible in our flight software and information systems courses, generally preferred by Aerospace students to out-of-department courses due to the prerequisites associated with upper-level non-Aero courses. Next, for students who have not yet taken the flight software or information systems course, University of Michigan faculty and graduate students, including all authors of this paper, have developed a series of tutorials aimed at exposing students to the fundamentals of more advanced programming logic and specific applications. To-date, tutorials have been biased toward embedded control system topics, but discussions are underway to offer an analogous tutorial series aimed at teaching efficient algorithm design and implementation for engineering analysis.

B. The Role of UM-FMS in Aerospace Computing Education

UM-FMS provides the opportunity for students to gain experience with three distinct programming environments: a ground station with appreciable computational power and graphics interface, an embedded processor running an operating system (Linux), and a microprocessor (Atmel). Students expressing an interest in our tutorial sessions and subsequently one of the student groups or research labs involved with UM-FMS are directed toward one of these programming environments based on experience and interest. By default, new students with only Engineering 101 experience will be directed toward UM-FMS GUI development, typically augmentation of an existing QT graphic. Any student more interested in embedded instrumentation will be directed toward straightforward microprocessor programming projects, using WinAVR or a Linux AVR cross-compiler for Atmel code development in C. Experienced faculty and students work with new students initially, with software development sessions maximizing overall education and development productivity. As students mature, their responsibilities migrate across all processing platforms, including inter-system communication protocols. Faculty and 1-2 senior graduate students maintain a current release with subversion to minimize situations in which functional code is broken or lost.

The Aerospace computation tutorial, to-date offered as a series of optional night sessions for interested students, is structured as shown below in Figure 7. Lecture Series Part I, divided over 4-5 sessions, introduces students to concepts not covered during the freshman computing course. Simple example C and C++ code, derived in part from UM-FMS modules, accompanies a lecture-style presentation of design, logic, and implementation specifics. Although only offered to Aerospace research groups to-date, we anticipate offering the Part I lecture series annually (fall term) to interested Aerospace students at all levels. Part II of this series is offered to the subset of students that plan to engage in UM-FMS development activities as part of their research or coursework.[§] To-date, Part II has been conducted on an as-needed basis, with students focusing on one exclusive component as described above. As students gain experience they are encouraged to broaden their expertise based on their interests and specific project goals.

<p>Part I: Software development “beyond Engineering 101”:</p> <ul style="list-style-type: none"> ○ Data abstraction and object-oriented programming ○ Socket-based network communication ○ Parallel/asynchronous execution with multiple threads ○ Serial communication protocols (RS/232, I2C, ...) and data interfaces (A/D, D/A, PWM, DIO) ○ Sensor data calibration, processing, and filtering ○ Software development with subversion and doxygen <p>Part II: UM-FMS:</p> <ul style="list-style-type: none"> ○ Shared data structures ○ Ground Station: QT-based GUI, communication links ○ Embedded Processor I (Gumstix): Thread functionality and real-time requirements, communication protocols ○ Microprocessor (Atmel): Registers, interrupts, A/D, D/A, PWM ○ Embedded Processor II (PC/104): Combines real-time threads, communication, and hardware interfaces on an integrated CPU/DAQ/network board; Linux and QNX operating system options
--

Figure 7: Aerospace Engineering “Computational Literacy” Tutorial Series with UM-FMS

[§] Students with sufficient background are not required to attend these lectures prior to working with UM-FMS. It is our experience that such Aerospace students are rare. It is also our experience that students are typically not over-confident about their programming skills thus will not claim to have experience in the Figure 7 topic areas unless the really do.

C. Complementary Aerospace Electronics Education

There is also evidence of a need for engineering students to become more familiar with the basic electronics, instrumentation and data acquisition often used in laboratory and project settings. To support this need, especially amongst multi-disciplinary student project teams, a sequence of labs has been established by students for students. This sequence of five labs is offered in both fall and winter semesters as an extracurricular course taught by experienced members of these student teams. Each team identifies several areas or needs that they wish to learn more about. Experienced graduate and senior undergraduate students create labs, or build upon previous labs, to provide a hands-on opportunity to practice the needed skills. Members of all student teams are then encouraged to pick and choose labs that will benefit both themselves and their team's project needs.

The labs created to date include AC/DC Instrumentation, Embedded Microprocessor (Atmel) Programming, Communication and Telemetry Systems, Power Management, and Motor Control. The Atmel programming and communication systems tutorials have a different focus but contain overlapping material with their computing tutorial series counterparts. It is our experience that students benefit from multiple exposures to these concepts, particularly given the complementary emphases on software algorithms vs. electronic signals and systems. A wide variety of University of Michigan student teams including Baja Racer, Solar Car, Student Space Systems Fabrication Lab (S3FL), SolarBubbles, MASA, and Formula SAE have participated in these efforts. Typically students begin with a base course in the first two labs. The skills in these labs are then used in the latter three labs in specific project examples. The latter three labs rely on mini-Sumo robots as a practical example of how each of the systems works. As an alternative, student teams often bring their own problems (such as MASA's rocket test-stand valve controller) to the course and work with other students taking the course and the volunteer instructors to finish the project.

The formation of these labs was motivated by the need of the student project teams to understand and utilize basic electronics and programming. It was then from these teams that instructors emerged, some having taken the course in previous years. It is hoped in the future that these labs will be institutionalized in some manner to maintain a sustainable training program as new members join teams and others leave. Several of the students who have participated in this electronics course series as participants and instructors have also participated in UM-FMS development as well as the computational literacy tutorial program described previously.

VI. Conclusions and Future Work

This paper has presented the University of Michigan open-source flight management system (UM-FMS). Specifics of the avionics hardware and software design were presented, along with a summary of adaptations to three University of Michigan UAS platforms.

Like most software, the UM-FMS is continually evolving. Undergraduate and graduate students primarily from the SolarBubbles team are augmenting the GUI with improved displays such as real-time strip chart data displays and interactive waypoint specification. A PC/104 implementation is planned for a highly-instrumented flexible-wing aircraft, to be first flown in Summer 2009. Our educational objectives are only beginning to be achieved. We believe efforts such as UM-FMS at Michigan and other institutions are beginning to provide a tangible, motivational means to promote a higher level of computational literacy resulting in Aerospace Engineers who embrace and fundamentally understand the computational tools they will use throughout their careers.

VII. Acknowledgments

The authors would like to thank University of Michigan students Eric Gustafson and the SolarBubbles student team for their contributions to the ground station software. Many thanks also to the Flying Fish team members who supported flight management system integration and flight testing over a wide array of operating conditions.

VIII. References

- ¹ Johnson, E. and Schrage, D., "System Integration and Operation of a Research Unmanned Aerial Vehicle," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 1, no. 1, Jan. 2004.

- ² R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, , and M. Goodrich, "Autonomous Vehicle Technologies for Small Fixed-Wing UAVs," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, pp.92–108, Jan 2005.
- ³ A. Frank, J.S. McGrew, M. Valenti, D. Levine and J.P. How, "Hover, Transition, and Level Flight Control Design for a Single-Propeller Indoor Airplane," *AIAA Guidance, Navigation and Control Conference*, August 2007.
- ⁴ R. De Nardi, O. Holland, J. Woods, and A. Clark, "SwarMAV: A Swarm of Miniature Aerial Vehicles," *21st International UAV Systems Conference*, Bristol, UK, April 2006.
- ⁵ D. Macy, R. Eubank, E. Atkins, L. Bernal, P. Washabaugh, G. Meadows, N. Wild, D. Smith, H. Van Sumeren, "Flying Fish: A Persistent Ocean Surveillance Buoy with Autonomous Aerial Repositioning," *AUVSI Conference*, San Diego, CA, June 2008.
- ⁶ G. Meadows, E. Atkins, L. Bernal, P. Washabaugh, B. Gilchrist, L. Meadows, D. Smith, H. VanSumeren, D. Macy, R. Eubank, B. Smith, and J. Brown, "The Flying Fish Persistent Ocean Surveillance Platform," *AIAA Unmanned Unlimited Conference*, AIAA, Seattle, WA, April 2009.
- ⁷ R. Eubank, E. Atkins, and D. Macy, "Autonomous Guidance and Control of the Flying Fish Ocean Surveillance Platform", *AIAA Infotech@Aerospace Conference*, AIAA, Seattle, WA, April 2009.
- ⁸ A. Klesh, D. Macy, A. Smith, P. Senatore, N. Rooney, and J. Wiebenga, "SolarBubbles: An Autonomous Solar-Powered UAV," *AUVSI Conference*, San Diego, CA, June 2008.
- ⁹ I. Tobasco and E. Atkins, "A Compact Open-Source Inertial Measurement Unit," *AIAA Midwest Student Conference*, University of Illinois, Urbana-Champaign, IL, April 2009.
- ¹⁰ D. Yeo, J. Henderson, and E. Atkins, "An Aerodynamic Data System for Small Hovering Fixed-Wing UAS," *AIAA Guidance, Navigation, and Control Conference*, Chicago, IL, August 2009 (to appear).