Infotech@Aerospace 2011
29 - 31 March 2011, St. Louis, Missouri

AIAA 2011-1581

A Project-based Undergraduate Aerospace Sequence, with Embedded Computational Intelligence

Ella M. Atkins*
University of Michigan, Ann Arbor, Michigan, 48105, U.S.A

The engineering curriculum is at a crossroads: students must continue to understand the traditional "fundamentals" attributed to the Aerospace field, but it also is recognized that 21st century students must be adept with systems and software. Given pressure to maintain four-year degree programs, many Universities have restricted their initiatives to technical electives and fostered the development of extra-curricular student teams; at best programs have introduced one new software or project-based course while maintaining legacy requirements nearly as they were decades ago. This paper examines a series of courses, currently electives, that if adopted into the required curriculum can offer Aerospace students exposure in each undergraduate year to progressively more advanced computing theory/practice in the context of motivational project-based activities. We focus on a series of project activities in use at the University of Michigan aimed at giving students a "fundamental" understanding of the Aerospace system and its software we hope will ultimately approach the level of understanding students have traditionally gained in mathematical and physics-based concepts.

I. Introduction

The primary goal of an undergraduate engineering education is for students to gain an ability to "think deeply and logically" about complex engineering problems. For Aerospace, this objective has primarily been accomplished through in-depth exposure to fundamental modeling and problem-solving methods derived from the principles of calculus. By understanding the origins of the orbital elements, beam bending, or lift and drag on an airfoil, students are able to grasp the connections between how and why platforms can fly given the conservation laws (mass, momentum, and energy) and the ever-present gravitational forces. As we find ourselves in the "computational age", with systems ranging from laptops and sophisticated cell phones to low-cost cloud computing sites, two questions emerge. First, what impact does and should the modern computer have on how and what we teach our students? Currently computers are used by our undergraduates as word processors and for basic math in support of homework, but they are largely treated as black boxes except for a "freshman computing" course many find painful. Presuming we find a single first-year exposure insufficient, a position others share,[1,2,3] a second question emerges: how can we integrate the same level of deep thinking about the "algorithm" and "information manipulation" into our Aerospace curriculum as we have achieved over the centuries for calculus-based mathematics? This paper focuses on these questions in a specific context we view as an opportunity in this regard: project-based education.

Recently, both academia and industry have begun to recognize the importance of project-based education in the Aerospace curriculum. Through design, build, test (DBT) or analogously conceive, design, innovate, operate (CDIO),[4] the student graduates with some understanding of the full system life cycle. In this manner students will look beyond the single-component analysis or testing with which they may be initially tasked to better meet integration or system-level requirements. At the University of Michigan a wide spectrum of project-based activities are available to supplement the regular curriculum. Below, we summarize the current Michigan Aerospace undergraduate program, highlighting available project-based curricular and extra-curricular activities that capture a number of students in our program. As will be apparent, while there are a number of optional opportunities for students to gain nontrivial experience with software and systems, this exposure remains mostly optional, and many students even who pursue projects believe it appropriate to identify a role requiring only skills of traditional Aerospace origin.

The remainder of the paper will describe efforts to expand the Aerospace curriculum in a manner that can ultimately offer students repeat exposure each undergraduate year to project-based education with nontrivial computing/software content. We first describe an "advanced" alternative to our freshman computing course in which students are offered a challenging project and more object-oriented programming background if they are

* Associate Professor, Aerospace Engineering Department, University of Michigan, Ann Arbor, MI, Associate Fellow.

sufficiently motivated and/or prepared for this challenge. In this context, we have developed two embedded systems projects, one "structured" project with clear instructions and one "design-oriented" project in which students are offered the opportunity to go through the entire design, build, test process. In the next section, we discuss a possible extension to a proposed sophomore design-build-test course that would allow all Aerospace students to gain exposure to embedded programming as part of the Aerospace system design/built/test cycle. We then describe an upper-level course called Flight Software Systems, first introduced by the author at the University of Maryland then again at the University of Michigan, to expose students to fundamental computing "theory" as well as to more advanced software engineering "practices" required to succeed in developing code for today's Aerospace platforms.

While the sophomore and upper-level software-oriented exposures remain optional, our flight dynamics and control (FDC) faculty have reached consensus on requiring all PhD students take a new "Aerospace Information Systems" course, now in the midst of its second offering. This course is pursuing a new collaborative, autonomous systems project that requires students to experience the full algorithm design, implementation, and test cycle for a challenging autonomous systems problem. While the goal with our undergraduate projects is to challenge but motivate students with projects that can be a complete success given sufficient effort, this graduate course is aimed at exposing students to a progressively-challenging set of project-based activities that require students to negotiate strategies for component-level and partial system testing prior to deploying the "full system" and "hoping it works". We conclude the paper with a discussion of how our approach can ultimately provide sufficient exposure to students that not only can they use "computational tools", they also can understand how the tools can be modified, interfaced, and ultimately manipulated efficiently, correctly, and as part of a multi-disciplinary project team.

II. Existing University of Michigan Aerospace Curriculum and Project Opportunities

The University of Michigan undergraduate Aerospace curriculum is structured around three traditional "pillars": structures, gas dynamics, and dynamics and control. The set of required courses currently required in the undergraduate Aerospace program are summarized by discipline in Figure 1.[5] A total of 129 credit hours are required for a student to be eligible for a Bachelor's Degree in Aerospace Engineering, of which 94 are from required courses (Figure 1), 10 are technical electives, 9 are general electives, and 16 are humanities and social sciences. The courses highlighted in Figure 1, ENGR 100, AEROSP 481 or 483, and AEROSP 405 are the only required courses with substantial project-based activity, but apart from one "microprocessors" section of ENGR 100 (taught by a computer science professor), these courses focus on project-based activities structured around the three traditional "pillars". Quite simply, students gain no appreciable algorithm or software engineering exposure in any of the currently-required project-based Aerospace courses. Perhaps the most obvious deficiency found in most Aerospace programs is in the "Aircraft Design" course, where the cockpit might as well be a blank panel with direct mechanical links to the controls. Students apply their knowledge of structures, aerodynamics, and propulsion to design and analyze the physics-based properties of the aircraft and its engines, but they do not consider most aspects of avionics in their designs. Software is rarely mentioned although it is generally recognized as one of the most costly technologies to develop and certify in modern Aerospace platforms. When mentioning this to instructors, the most common response is that there is "too much to cover in a single course". Another significant factor, less often mentioned, is that students have been far better prepared to develop and analyze physics-based models than to develop and analyze logic and finite-state models.

Several of the Michigan Aerospace faculty have recognized the importance of addressing deficiencies in both computational and project-based engineering skill sets in our graduates. Our advisory board has also recognized these deficiencies, and has aptly suggested expansion in a direction aimed at improving student conceptual understanding rather than expanding teaching of "software applications". Figure 2 shows the set of project-based courses, both required and elective, currently offered to Aerospace students. Note that a new Multidisciplinary (MD) Minor program[6] has been developed that can offer students an even more comprehensive design experience should they choose this degree option. In Figure 2, graduate courses available to but not often taken by undergraduates are italicized; project-based courses with appreciable logic/algorithm and software content are highlighted in blue. The first, ENGR 151, is an accelerated version of ENGR 101, the default freshman computing courses. Although less than 10% of Aerospace students opt for ENGR 151, it is still a useful tool for appropriately challenging and preparing Aerospace students who elect that path. Described in Section III, it also provides a glimpse of what would be possible in a freshman computing course should K-12 better prepare our first-year students with background in logic and programming (in any language). The next highlighted course in Figure 2, AEROSP 495 (ultimately planned to become AEROSP 205, potentially a required sophomore course), is under development by Professor Pete Washabaugh, who also developed a highly-respected section of the first-year ENGR 100 course in which students design, build, and flight test blimps such as that shown in Figure 3.[7]

Core Math/Science (31 credits):

MATH 115, 116, 215, and 216 (Calc I – IV) (4 credits each; 16 credits)
CHEM 125/126 and 130, or 210 and 211 (5 credits, lab)
Physics 140 with Lab 141 (5 credits, lab)
Physics 240 with Lab 241 (5 credits, lab)

First-Year Engineering (8 credits):

ENGR 100, Intro to Engineering (4 credits, Tech Comm (Technical Communication))
ENGR 101 (or **151**), Intro (or Accelerated Intro) to Computers & Programming (4 credits, lab)

Structures & Materials (12 credits):

MATSCIE 220, Intro to Materials (4 credits)
AEROSP 215, Intro to Solid Mechanics and Aerospace Structures (4 credits)
AEROSP 315, Aircraft and Spacecraft Structures (4 credits)

Gas Dynamics (12 credits):

AEROSP 225, Intro to Gas Dynamics (4 credits)
AEROSP 325, Aerodynamics (4 credits)
AEROSP 335, Aircraft and Spacecraft Propulsion (4 credits)

Dynamics & Control (11 credits):

MECHENG 240, Intro to Dynamics and Vibrations (4 credits)
AEROSP 347, Space Flight Mechanics (3 credits)
AEROSP 348, Aircraft Dynamics and Control (3 credits)

Interdisciplinary/Systems (12 credits):

AEROSP 201, Intro to Aerospace Engineering (3 credits)
AEROSP 285, Aero Engineering Seminar (1 credit, Tech Comm)
EECS 314, Circuit Analysis and Electronics (4 credits)
AEROSP 481/483, Aircraft (481) or Space System (483) Design (4 credits)

Laboratory (8 credits):

AEROSP 305, Aerospace Engr Lab I (4 credits, Tech Comm, lab)
AEROSP 405, Aerospace Engr Lab II (4 credits, Tech Comm, lab/projects)

Figure 1: University of Michigan Aerospace Engineering Required Undergraduate Courses for 2010-2011.

ENGR 100, Intro to Engineering (term projects) (4 credits, project-based)
ENGR 151, Accelerated Intro to Computers and Programming (4 credits, default/advanced project options)
AEROSP 495 (205), Aerospace Engineering Systems (Design/Build/Test) (3 credits, term project)
AEROSP 450, Flight Software Systems (3 credits, embedded system/software projects)
AEROSP 405, Aerospace Engr Lab II (4 credits, experimental projects)
AEROSP 481/483, Aircraft (481) or Space System (483) Design (4 credits, paper study design projects)
AEROSP 290, Directed Study (sophomore elective) (1-3 credits)
AEROSP 390, Directed Study (junior elective) (1-3 credits)
AEROSP 490, Directed Study (senior elective) (1-3 credits)
AEROSP 590, Directed Study (graduate elective) (1-6 credits)
AEROSP 552 (740), Aerospace Information Systems (3 credits, autonomous flight team project)

Figure 2: University of Michigan Required & Elective Courses with Design Project Content.

In the current elective offering of AEROSP 495/205, students design, build, and operate hovercraft equipped with an Atmel microprocessor enabling closed-loop control. Software-oriented modules for this course are still under development, although a preliminary implementation of an "autonomous hovercraft" capability was developed as a student design project in Fall 2010 as will be described below in Section III. The third course, Flight Software Systems (see Section IV also), has been offered annually as a technical elective since Fall 2006; its goal is to appropriately mix computer science theory with practice to better equip students not only to program but also to converse with computer science colleagues and to understand the fundamental principles used by the CS community to model and validate their logic and code. The final highlighted course, AEROSP 552, is a graduate-level course in Aerospace Information Systems. Described further in Section V, it provides primarily flight dynamics and control graduate students well-prepared in control theory but potentially with no more computing background than ENGR 101 an introduction to computing theory and more in-depth coverage of information theory (communication) as well as the deterministic and uncertain inference algorithms beginning to be deployed in air and space vehicles. Students this term are completing a challenging "information systems" design project (in progress) that will also be described below.

Figure 3: Student-Designed Blimp from an 'Introduction to Engineering' (ENGR100) Course.

Students at the University of Michigan can find exposure to project-based education outside the classroom as well. Motivated by competitions, record-setting attempts, or novel missions (e.g., CubeSats) becoming possible for undergraduate students to accomplish, a number of Aerospace students join project teams as early as their first year and often remain in these teams through graduation. Michigan recently developed the Wilson Student Project Center (http://www.engin.umich.edu/teamprojects/) that hosts the majority of the engineering student teams, including design/build/fly, aerial robotics, and long-endurance flight teams from Aerospace. The Student Space Systems Fabrication Lab (S3FL) (http://s3fl.org/) provides students interested in space systems a broad set of opportunities to participate in projects ranging from KC-135 experiments to a NASA-sponsored CubeSat project planned for launch.

III. A First-Year "Accelerated" Computing Option

University of Michigan College of Engineering students declare a major at the end of their first year, requiring early courses to be common across all engineering majors. Student preparation for the first-year computing course is even more varied than for math, with some students never exposed to any programming while others know several languages and have developed applications in high school, some for profit. Quite simply, if all first-year students are mixed in a common course, the top students learn nothing, decide "computer science" is boring (if they are not majors), and in many cases intimidate less-prepared students. The University of Michigan now offers two first-year "programming" courses, one that assumes no background (ENGR 101) and another (ENGR 151), self-selected by students based on their motivation and an optional placement exam, that contains substantial material beyond ENGR 101. Table 1 summarizes the topics covered in ENGR 101 and ENGR 151. In its initial two offerings (Fall 2009 and Fall 2010), ENGR 151 has attracted approximately 100 out of 1300 first-year engineering students, indicating that fewer than 1 in 10 entering engineering students either feel sufficiently prepared or motivated to elect an "accelerated" programming option. Not all ENGR 151 students have programming background; ability to think logically and strong work ethic have thusfar been determined sufficient for success in ENGR 151, although a strong background certainly reduces workload.

Table 1: Comparison of Topics Covered in First-Year Computing Courses Engr101 vs. Engr151.

Topic	*Engr101*	*Engr151*
Intro. to computers, logic, and algorithms	*	*
C++ Algorithms & Syntax		
Data types and input/output	*	*
Math and logical operators/functions	*	*
Functions & procedures	*	*
Selection	*	*
Iteration	*	*
Recursion	*	*
File reading/writing	*	*
Built-in Classes (e.g., vector, string)	*	*
Pointers & dynamic memory allocation	-	*
Data structures and linked lists		*
C++ Classes (Object-oriented programming)		*
Parallel computing introduction		-
Networked computing introduction		-
Embedded systems project		*
Matlab Algorithms & Syntax		
Operators, lists, vectors, matrices	*	*
Matrix/vector indexing, vectorization	*	*
Math operators and functions	*	*
Scripts & functions	*	*
Selection & Iteration	*	*
Plotting	*	*
Linear algebra (intro)		*
Simulink, auto-coding		-
Numerical analysis project/application	-	*

Always covered (*), Sometimes covered (-), Never covered (blank)

Both ENGR 101 and ENGR 151 are structured as three lecture hours and two lab hours aimed at offering students individual attention as needed. Numerous office and tutorial hours are also available. All instructors offer "projects" not assignments, although in ENGR 101 projects tend to be simple programs aimed at teaching concepts without intimidating struggling students. While many in industry assume young Aerospace engineers' programming deficiencies are due to "poor instruction" in the first-year course, it is this author's opinion that ENGR 101, at least at Michigan, is well-taught, but it is unfortunately an "island" (single exposure) for students who have no programming background and who do not elect any additional courses in algorithms or software. Contrast this with physics-based force and moment balance, concepts that recur in most every Aerospace course students take.

As a motivator for all the "extra work" students are required to complete in ENGR 151, we offer a hardware-based "robot control" project. Engineering students almost all have indicated exposure to "real hardware" is both motivational and fun, even if it requires numerous hours to complete well. A pair of single degree-of-freedom platforms called "TableSat" (Tabletop Satellite), originally developed in collaboration with NASA Goddard Space Flight Center and Professor Robert Sanner at the University of Maryland, have proven capable of offering students ranging from "motivated freshmen" to graduate students challenges with noisy sensors and embedded real-time control. Shown in Figure 4, TableSat rotates about a single low-friction central pivot point, simulating a drag-free single degree-of-freedom spacecraft. TableSat is driven by two low-cost computer fans and contains three sensor types: a single rate gyro, a three-axis magnetometer, and a set of sun/light sensors. This sensor suite was motivated by the desire to measure quantities comparable to those available on spacecraft. As shown in Figure 5, a student user is able to remotely view TableSat through an overhead webcam. TableSat is equipped with a Diamond Systems Athena II PC/104 computer networked through a router to the campus network. Students are asked to develop and compile code on a stand-alone QNX server, move their code to TableSat, then execute it. Through the network interface, students have persistent access to the hardware, only actually seeing TableSat during a brief tour of the lab when the assignment is first distributed.

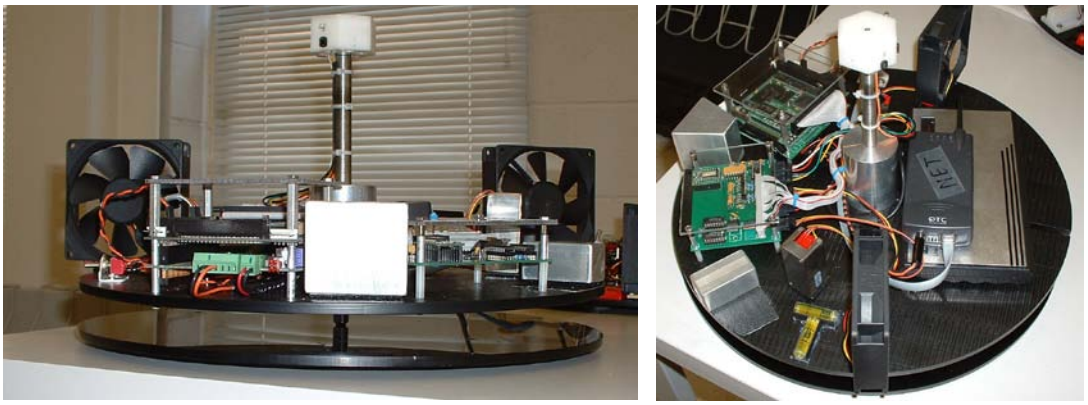

Figure 4: TableSat for Embedded Computing and Control Projects.

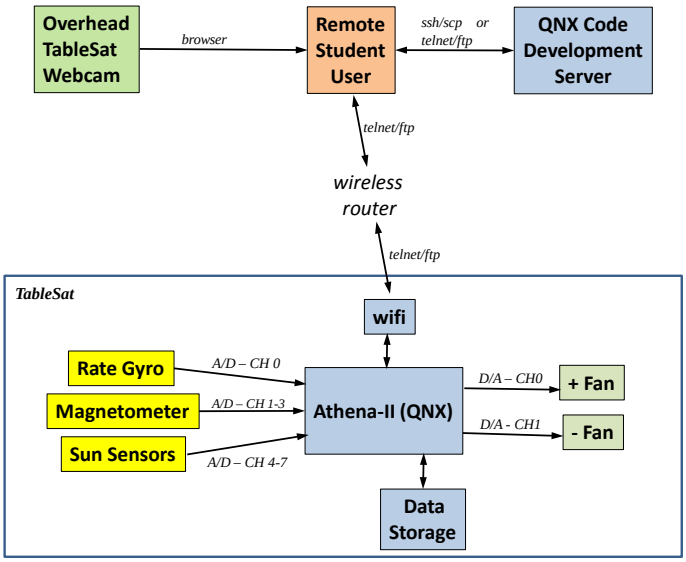

Figure 5: TableSat Components and User Interfaces.

The baseline TableSat project requires students to implement and tune single degree-of-freedom (DOF) proportional-integral-derivative (PID) controllers on an embedded computing environment, a Diamond Systems Athena II PC/104 computer running the QNX operating system. Students are able to use the Diamond Systems Corporation Universal Driver (DSCUD) software (www.diamondsystems.com) and are given two simple example C++ programs as a start. They are then asked to complete the following project goals:

- *Task 1 - Rate Control:* Read the (low-noise) rate gyro, implement and tune a PI controller to achieve a commanded angular velocity with fast response time, little overshoot, and near-zero steady-state error
- *Task 2 - Magnetometer Calibration*: Read (x,y) magnetometer readings and calibrate to provide a measure of compass heading. This task is not straightforward because the magnetometer is noisy and significantly impacted by building structure.
- *Optional Task 1 - Attitude Control*: Use the rate gyro and magnetometer in a PID controller to achieve a commanded heading
- *Optional Task 2 – Sun Sensor Attitude Control:* Calibrate an array of [noisy] light sensors and perform attitude control based on pointing TableSat in a commanded direction relative to a single light source.

Educational objectives for the TableSat project include:

- Learn basic principles of a real-time operating system (e.g., QNX, www.qnx.com)
- Gain experience with remote and embedded code and data management (e.g., `ssh, scp, telnet, ftp`)
- Learn about controlling execution rate and interfacing with hardware (e.g., A/D, D/A)

- Gain experience with noisy sensors and non-ideal actuators for a single DOF feedback control application

Feedback from the ENGR 101 instructors is that ENGR 151 has largely removed the "intimidation factor" from ENGR 101. However, in the group of 100 students in ENG 151, backgrounds are still highly varied. In Fall 2010, students with substantial programming backgrounds were offered the option of completing a ½ term "design project" rather than sitting through lectures and completing assignments they found to be a review. Nine students self-selected this "design project" option and worked far more hours than any other students in the class, giving positive feedback nonetheless. For this project, students were offered the opportunity to "automate" one of the hovercraft previously built for AEROSP 495 (205), shown in Figure 6. Because of their level of ambition and desire for challenge, these students were offered the opportunity to select their own hardware and develop their software from scratch. They were divided into four subteams based on the set of software algorithms and interfaces required for autonomous operation: [machine] vision, communication, navigation, and autopilot (control). These subteams are shown in Figure 7, along with the primary responsibilities assigned to each subteam. The students were guided at a high level but were allowed to "fail" and "try again" as long as safety was maintained. Figure 8 shows the hardware students chose, where CAEN = "Computer Aided Engineering Network," aerocad1 is the Linux box used for ground station and software development, ADC = analog/digital converter, and PWM = pulse width modulation. Choosing Gumstix Overo as the primary onboard computer, the students also chose low-cost inertial sensors but used a high-quality network camera already available in the lab. They began by installing Linux Ubuntu and the Gumstix Overo environment on a Linux box and proceeded to fully and independently develop all software required to automate the hovercraft. Although the team was ultimately successful in driving the underactuated hovercraft to a target point within the overhead camera's field of view, the process was not without learning experiences. The navigation team learned that A/D channels fail if high voltages are applied and that sensors fail if polarity is reversed. The vision team learned that calibration algorithms require substantial tuning and that network camera delay as well as frame rate can be a major issue for real-time control. The autopilot team learned quite a bit about how to go from ideal physics-based dynamics models to "practical" PD control, while the communications team learned not only about network communication protocols but also about coordinating the data of three groups with rather independent student team members. We plan to offer a similar option in future years, as ENGR 151 was therefore able to provide appropriate challenge for every student in the class without "leaving behind" those that hadn't programmed most of their lives.

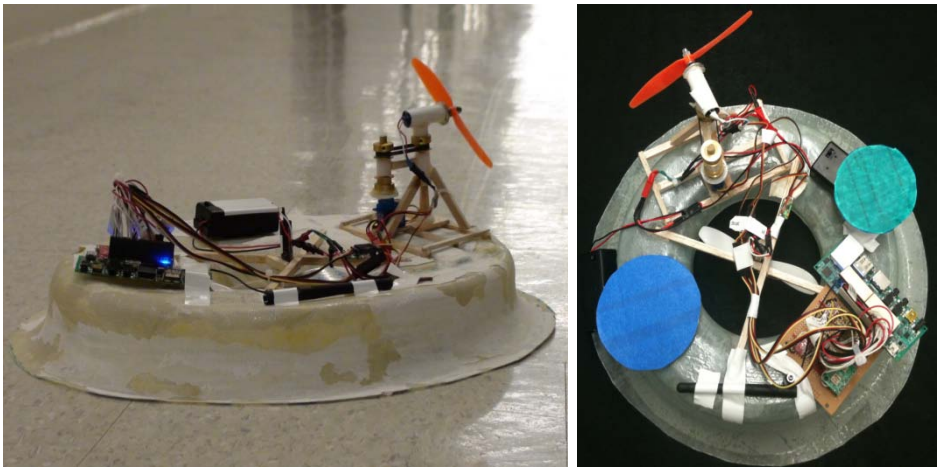

Figure 6: Hovercraft for Guidance, Navigation, and Control Projects.

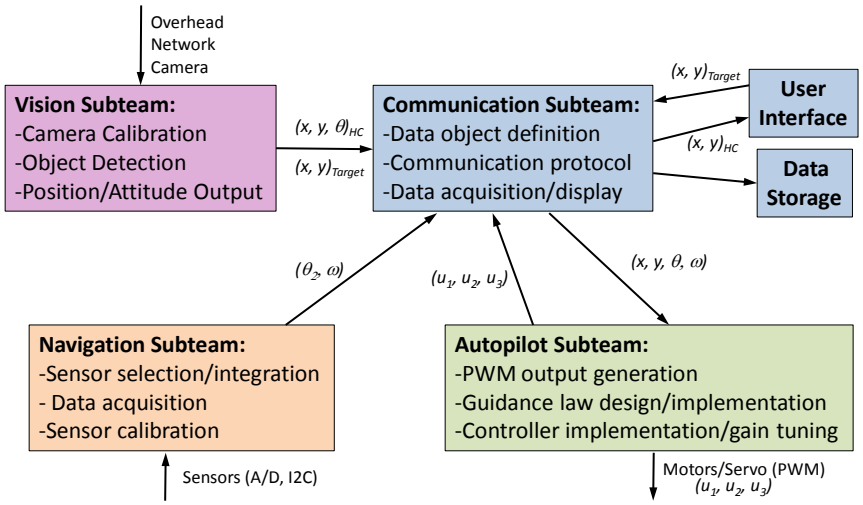

Figure 7: Autonomous Hovercraft Project Software-Centric Subteams.

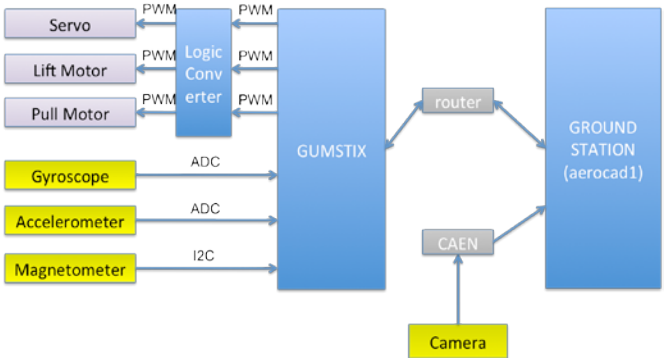

Figure 8: Autonomous Hovercraft Project Hardware.

Although this paper is not focused on K-12, no discussion of ENGR 101 and ENGR 151 would be complete without mentioning that *if* K-12 education included even one year-long logic and programming course for all students, all entering engineering students could be successful with a course the pace ENGR 151 offers. Without this background, however, students give up if their first course is offered at the ENGR 151 pace. This is important for the Aerospace community to recognize as it further explains why one freshman course is not sufficient to produce graduates skilled at software development.

IV. An Upper-level Course in Flight Software Systems

Recognizing that students are motivated by "software that flies", the author has introduced an upper-level Aerospace elective entitled "Flight Software Systems" at both the University of Maryland and the University of Michigan. Covering topics listed in Figure 9, Flight Software Systems combines lectures on "theory" and "practice", exposing students to the fundamental discrete modeling computing concepts foundational to the computer science field while also continuing to better educate students on contemporary practices in software development and testing. This "computing" course has similar motivations to those developed at other Universities[8,9]: to better prepare students to understand and develop algorithms and software. The distinction is that Flight Software Systems focuses more on the computing theory and less on the software engineering software processes, which by this author's view is best understood after concepts in logic and discrete systems, finite state machines, inference, and model checking for validation & verification have been mastered. The Flight Software Systems course is designed to allow students to define their own course projects or to select a course from a list of pre-defined options. Pre-defined options are instructor-dependent and have included advanced software development for TableSat or a six-wheel rover when this author was instructor; students have also been offered the

opportunity to define their own project, potentially connected to their extra-curricular project team so long as this project fits within the context of "flight software" development.

1. Course Introduction
2. *Extended Evening Tutorial:* Introduction to Linux; Review of C++ from ENGR101
3. *Theory*: Discrete Mathematics I
4. *Practice*: C/C++ data structures
5. *Theory*: Discrete Mathematics II
6. *Practice*: C++ classes; intro to object-oriented programming
7. *Theory*: Cardinality & Binary relations
8. *Practice*: Multi-threaded programming; shared data (pthread, mutex)
9. *Theory*: Strings & languages
10. *Practice*: Network communication with (TCP,UDP)
11. *Theory*: Languages II
12. *Practice*: Serial port programming for embedded applications
13. *Theory*: Deterministic finite automata
14. *Practice*: The Gumstix/Robostix embedded computing platform
15. *Practice*: Version Control (subversion)
16. *Theory*: Regular expressions; nondeterministic finite automata (Text Ch. 9)
17. *Practice*: TableSat (Tabletop Satellite): QNX O/S, A/D, D/A conversion
18. *Theory*: Operations on Regular Machinery (Text Ch. 10)
19. *Practice*: Programming the Atmel microcontroller
20. *Theory*: Binary Decision Diagrams (BDDs) (Text Ch. 11)
21. *Practice*: Real-time timers and their application
22. *Theory*: Computational Complexity (Text Ch. 19)
23. *Practice*: Real-time scheduling algorithms
24. *Theory*: Model Checking: Basics (Text Ch. 21)
25. *Practice*: Introduction to Software Engineering
26. *Practice*: Intro to the Unified Modeling Language (UML)
27. *Final Design Project Presentations (embedded systems: Atmel, Gumstix, Athena II)*

Figure 9: Flight Software Systems (AEROSP 450) Syllabus.

V. Progress in the FDC Graduate Program

Michigan Aerospace has to some extent been more successful in requiring the study of computing theory at the graduate level than at the undergraduate level (except for the first-year course). In 2009, the Aerospace Flight Dynamics & Control (FDC) group, with heritage in information theory and systems extending back to the 1970s, elected to require a new course, "Aerospace Information Systems," become a core course for all FDC PhD students. For reference, the set of Aerospace core courses for the PhD program is shown in Figure 10; students much select any three of these courses for their preliminary exam, although most students elect all the courses in their subdiscipline. The outline for Aerospace Information Systems, currently AEROSP 740 (planned to become AEROSP 552), is shown in Figure 11. Four primary modules are covered: discrete systems and automata theory (overlapping with Flight Software, not required for graduate students), deterministic search and planning, information theory, and inference with uncertainty. The FDC team has agreed these areas not only expose our students to computing fundamentals but also are synergistic with emerging needs of highly-autonomous Aerospace platforms, manned and unmanned. This term (Winter 2011), we are pursuing a challenging final project illustrated in Figure 12 in which two competing hovercraft are racing to a waypoint with navigation data provided by a pair of quadrotors. This project is ambitious but the students have a progressively challenging set of steps leading to this ultimate goal, which they may or may not achieve. Note that students are aware of the level of challenge; they "fail" only if they do not attempt to solve the problem; they succeed if they accomplish a reasonable set of subgoals leading to the fully-coordinated four-vehicle mission to be flown in Michigan's FXB atrium to avoid the need for FAA approval.

VI. Conclusions

This paper has presented, in the context of the existing Michigan Aerospace curriculum, a series of computing courses that emphasize project-based education as a means to motivate students and to provide a learning experience

difficult to achieve in traditional problem sets. To ensure all our students are appropriately endowed with computational and project experience as undergraduates, we still must overcome credit limit constraints to fit new content into the legacy curriculum. We have managed to fit such required content into our FDC PhD program, offering hope that we can ultimately require exposure to both project and algorithm/software content in each and every undergraduate year.

Category 1: Core Graduate Courses:

Structures:
AE 513 Foundations of Solid and Structural Mechanics
AE 518 Theory of Elastic Stability
AE 543 Structural Dynamics
Gas Dynamics:
AE 520 Compressible Flow
AE 522 Viscous Flow
AE 532 Molecular Gas Dynamics
Dynamics & Control:
AE 540 Intermediate Dynamics
AE 550 Linear Systems
AE 552 Aerospace Information Systems

Category 2: Other Graduate Courses:

Structures:
AE 510 Finite Elements
AE 514 Foundations of Solid and Structural Mechanics
AE 516 Mechanics of Fibrous Composites
AE 544 Aeroelasticity
AE 545 Helicopter Aeromechanics
Gas Dynamics:
AE 523 Computational Fluid Dynamics
AE 525 Turbulent Flows
AE 533 Combustion Processes
NERS 571 Intermediate Plasma Physics I
Flight Dynamics & Control (FDC):
AE 548 Astrodynamics
AE 572 Aircraft Dynamics and Control
AE 573 Spacecraft Dynamics and Control
AE 584 Avionics, Navigation, and Guidance of Aerospace Vehicles
AE 5XX Any other FDC 500 level course approved by advisor

Figure 10: University of Michigan Aerospace Engineering Doctorate Preliminary Exam Courses for 2010-2011.

I: Discrete Systems & Automata Theory
Data structures introduction; extended evening tutorial reviewing Linux and C++
Boolean algebra review, sets and relations
Trees, graphs, array search and sort algorithms
Languages & deterministic finite state automata (DFSA)
Embedded programming (DIO, A/D, D/A, PWM, serial)
Nondeterministic finite state automata
Decision trees and binary decision diagrams (BDDs)
Timed automata & hybrid systems

II. Deterministic Search and Planning
Intro to AI & decision-making
Uninformed & informed search; heuristics
Hill climbing, simulated annealing, genetic algorithms (GAs)
Intro to AI Planning

Project introduction: formal requirements & design specification

III: Information Theory (Statistical Communication Theory)
Uncertainty, entropy, and inference
Data compression: the source coding theorem, symbol & stream codes
Communication over a noisy channel, error-correcting codes
Project progress review: Goal adjustment, subteam coordination, early results

IV. Inference with Uncertainty
Bayes Nets
Markov Decision Processes (MDP), Partially-observable MDPs
Overview of Reinforcement Learning (RL) & Neural Networks
Project presentation & demonstration

Figure 11: Aerospace Information Systems (Aero 552 / 740) Syllabus.

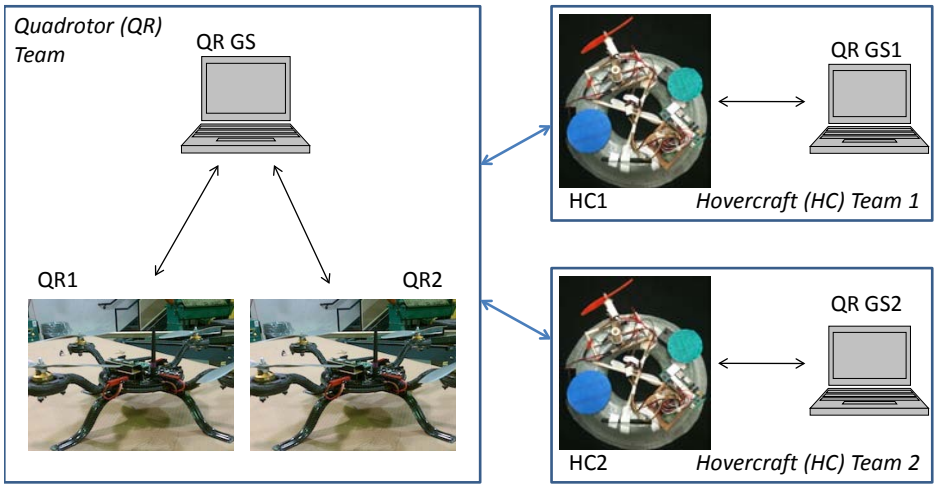

Figure 12: A Multi-platform Graduate-level Information Systems Project (*in progress*).

VII. References

[1] Long, L. N., "CyberScience and 21st Century Education," *Aerospace America*, April 2011.

[2] Long, L. N., "The Critical Need for Software Engineering Education," *CrossTalk: The Journal of Defense Software Engineering*, Vol. 21, No. 1, Jan., 2008.

[3] Arrington, E. A., "Structured Programming Education Requirements for Aerospace Engineering," *49th Aerospace Sciences Meeting*, AIAA, Orlando, FL, Jan. 4-7, 2011 (AIAA-2011-466).

[4] Crawley, E., Niewoehner, R., Gray, P., and Koster, J., "North American Aerospace Project: Adaptable Design/Build Projects for Aerospace Education," *49th Aerospace Sciences Meeting*, AIAA, Orlando, FL, Jan. 4-7, 2011 (AIAA-2011-554).

[5] http://aerospace.engin.umich.edu/AcademicPrograms/undergrad/sampleplan.html.

[6] http://www.engin.umich.edu/minors/multidisciplinarydesign/members/mdminordetails.html.

[7] Washabaugh, P., "An Experiential Introduction to Aerospace Engineering," *45th Aerospace Sciences Meeting*, AIAA, Reno, NV, Jan. 8-11, 2007 (AIAA-2007-296).

[8] Long, L.N. and Janrathitikarn, O. "A New Software Engineering Course for Undergraduate and Graduate Students," *AIAA Infotech@Aerospace Conference*, Atlanta, GA, April 20-22, 2010 (AIAA-2010-3505).

[9] Brannon, Mary L., Janrathitikarn, O., and Long, Lyle N. "Evaluation of a Student Team Project in an Introduction to Software Engineering Course for Aerospace Engineers," *ASEE Annual Conference*, Louisville, KY, June 20-23, 2010.