Trial by Flyer: Building Quadcopters From Scratch in a Ten-Week Capstone Course

Steven Swanson
University of Califonia, San Diego
San Diego, California
swanson@cs.ucsd.edu

ABSTRACT

We describe our experience teaching an intensive capstone course in which pairs of students build the hardware and software for a remote-controller quad-rotor aircraft (i.e., a quadcopter or "drone") from scratch in one 10-week quarter. The course covers printed circuit board (PCB) design and assembly, basic control theory and sensor fusion, and embedded systems programming. To reduce the workload on course staff and provide higher-quality feedback on student designs, we have implemented an automated PCB design checking tool/autograder. We describe the course content in detail, identify the challenges it presents to students and course staff, and propose changes to further increase student success and improve the scalability of the course.

CCS CONCEPTS

• **Social and professional topics** → **Model curricula**; • **Hardware** → **PCB design and layout**; • **Computer systems organization** → Robotic control; Embedded software.

KEYWORDS

Quadcopters, Capstone, Robotics

ACM Reference Format:
Steven Swanson. 2019. Trial by Flyer: Building Quadcopters From Scratch in a Ten-Week Capstone Course. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287451

1 INTRODUCTION

Compute-capable devices and robots are permeating every aspect of our students' lives, and many of the "killer apps" of the future will lie at the intersection of the computing and the physical world – robotics, the internet of things, personal electronics, etc. Students prepared to create those applications must understand how to design, build, and debug systems that include both code and physical components. Likewise, programming has become an integral part of every engineering discipline, so many engineering students can benefit from hands-on experience building a programming complex systems. Educators have devised a wide range of capstone courses to provide students with experience at the boundary of hardware and software [20, 22–25], but providing this experience in the face of growing enrollments is a significant challenge.

To provide students with project-based, hands-on experience we have developed a 10-week (one academic quarter) capstone, interdisciplinary course in which students design and build all of the hardware and software components of a remote-controlled quadrotor aircraft (or "quadcopter"). The course content includes embedded system/microcontroller programming, debugging software-controlled mechanical systems, working on interdisciplinary teams, PCB design, basic sensing and control theory, and practical skills in PCB assembly, soldering, and debugging electro-mechanical systems. The class targets senior undergraduate (or graduate) computer science, computer engineering, and other engineering students. We have taught the class annually for the last four years in the computer science and engineering department at the University of California, San Diego. The class has been growing in size and most recently had 24 students working in twelve groups.

A particularly challenging aspect of teaching the course is providing detailed, thorough, and reliable design reviews for student designs. Even a small error in the PCB design can lead to failure, and most teams require multiple reviews before their design is ready to manufacture. The workload can be crushing for the instructors.

To remedy this problem, we developed an automatic design checker call *QuadLint* that can verify many aspects of student designs. The tool drastically reduces the number of human design reviews required and lets those reviews focus on less tedious aspects of the design review process. It is thorough enough to fully automate grading for some of the preliminary labs in the course. We believe QuadLint is the first autograder to handle PCB designs.

apluThe emphasis on building hardware is unusual for a computer science course, but understanding how to build computer systems is critical for students hoping to craft next-generation devices that rely on the careful blending of hardware and software[1].

The course materials are all freely available online[2]. The only exceptions are a complete "solution" to the project and the source code for our automatic design checker. We are happy to share these with educators.

Based on surveys, course evaluations, and anecdotal reports, students find the class to be very challenging, but also exciting, engaging, educational, and fun. The course staff reports that it is great fun to teach as well.

This paper describes the course content, the challenges that arise in teaching the class, our design checking tool, and student reflections on the course. Most of the descriptions reflect the most recent instance of the class (Spring 2018). We also detail our plans for the next version which will address many of these challenges and should allow the class to grow further in size.

2 PROJECT OVERVIEW

The class is an intensive, capstone-style course. It moves very quickly and students need to stay engaged and work hard to succeed. With just a very few exceptions over the last four years, students rise to the course's challenge with aplomb.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02.
https://doi.org/10.1145/3287324.3287451

[1] Also, as Turing Award winner Alan Kay famously said "People who are really serious about software should make their own hardware"
[2] https://sites.google.com/a/eng.ucsd.edu/quadcopterclass/
https://github.com/NVSL?q=QuadClass

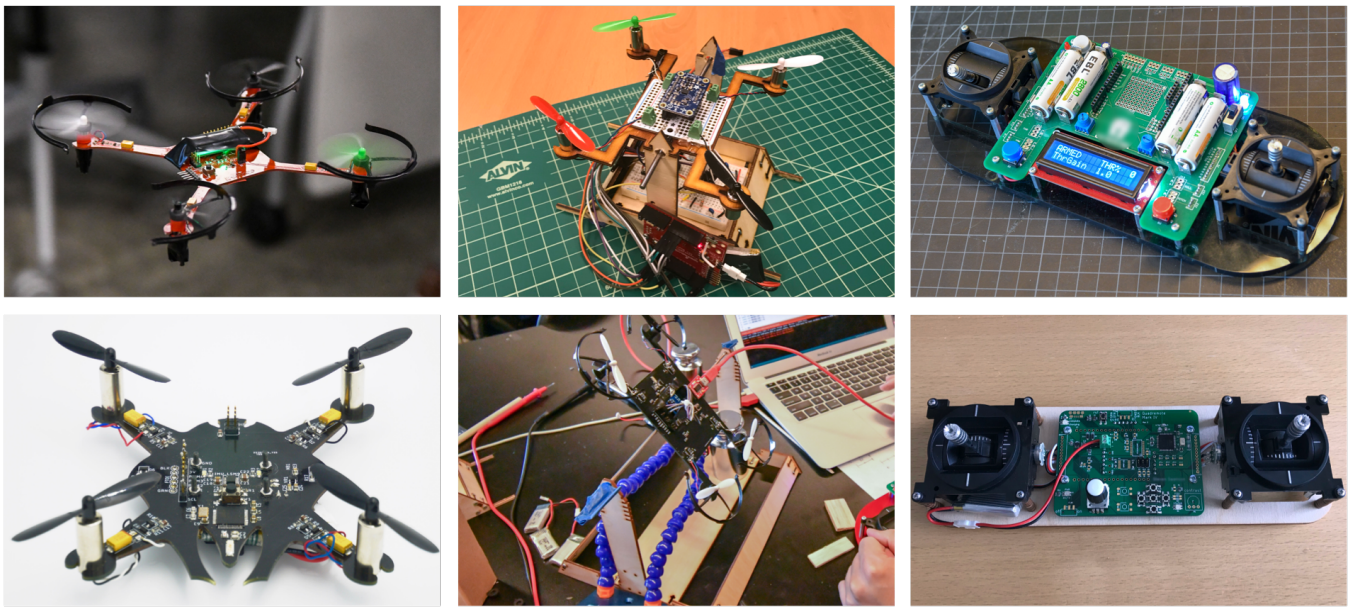

Figure 1: Photos from the Class – (From left to right) Two student quadcopters (top and bottom), the old test stand (top) and the new (bottom), and the current, expensive remote control (top) and the cost-optimized version (bottom).

H1: Eagle Intro	Complete an Eagle tutorial on building libraries, schematic assembly, layout, design checks, and CAM file generation.
H2: PCB Libraries	Build several Eagle library parts (of varying complexity) from scratch by reading and interpreting data sheets.
H3: Schematic creation	Create the schematic for their quadcopter based on a combination of reference designs, written specifications, and datasheets. Design an LED-based lighting element.
H4: PCB Layout	Design the shape of their PCB and layout the components to meet mechanical, electrical, and aesthetic constraints.
H5: PCB Assembly	Assemble their PCB using solder paste, tweezers, and a reflow oven.
S1: Software Intro	Setup the Arduino IDE and program their remote control and PID test stand to allow remote control of the motor speed. Assemble a PID test stand.
S2: Sensing	Implement sensor filtering and fusion using software and the IMU's internal filters to provide clean, low-noise measurements of the quadcopter's orientation.
S3: PID	Implement a one-channel PID controller to stabilize the PID test stand. Control the pitch of the test stand using their remote control.
S4: Flight Control	Implement flight operation functions such as "arming" the quadcopter and calibrating of the gimbals and IMU.
H6: Flight!	Combine the above elements to make their quadcopter fly.

Table 1: Lab Descriptions – The course comprises ten labs broken into three groups. Many of the hardware and software labs run concurrently to fit them all into ten weeks.

The quadcopters that students design use a PCB to host their electronics and to serve as its airframe (Figure 1-left). They measure less than 10 cm on a side and are suitable for flight indoors over short distances. The motors are moderately powerful, "brushed" electric motors powered by a small lithium-polymer (LiPo) battery, and we use small, plastic propellers. The quadcopters are easy to operate safely, and a blow from the propeller at full speed is painful but not particularly dangerous. Students wear eye protection around their flying quadcopters.

The quadcopters use an Arduino-compatible, 16 MHz Atmega128RFA microcontroller [19] that includes an IEEE 802.15.4 [16] radio. To measure the quadcopter's orientation, we use an inertial measurement unit (IMU) [28] that contains a 3-axis accelerometer and a 3-axis gyroscope. The purpose-built remote control (Figure 1-right) uses the same microcontroller, providing a uniform programming environment for the course.

We teach the class in a "Makerspace" run by our engineering school to support hands-on engineering courses. It provides an array of useful equipment, including high-quality soldering equipment, a professional reflow soldering oven, a laser cutter, and an array of hand tools.

3 COURSE CONTENT

The course breaks building a quadcopter into four high-level tasks: designing the PCB, implementing the flight control software, assembling the PCB, and getting the quadcopter flying. Below we describe each of these tasks, key challenges they present for the students and the course staff, and how we approach them in the class. Table 1 briefly describes the course's ten labs.

3.1 Designing and Manufacturing PCBs

The first task introduces students to the key concepts of PCBs design, the tools and processes used to design and manufacture them, and best practices for PCB design.

They complete labs Lab H1 and Lab H2 on their own. For the remaining labs, they work in pairs. We manufacture the student PCBs through JLPCB [12]. The boards usually arrive with 5-7 days of ordering. We build four-layer PCBs.

Key Topics This task addresses the following concepts and skills.

(1) **PCB principles:** What PCBs are, how they work, and how they are manufactured.

(2) **PCB Design Tools:** The abstractions and concepts that PCB design tools provide to describe PCB designs – schematics, part libraries, layers, design rules, etc.
(3) **Design From Example:** How to adapt reference designs to a particular application.
(4) **Design From Datasheet:** How to interpret device datasheets to integrate a device into a design.
(5) **Part Selection:** How to select appropriate components for a design from among many, many options.
(6) **Design Requirements:** How to describe, interpret and satisfy electrical, mechanical, and application requirements for a PCB.
(7) **PCB Design Best Practices:** How to design schematics and PCBs that functional, manufacturable, and comprehensible to other engineers.
(8) **PCB Manufacturing Process:** How to prepare a design for manufacturing and interpret the design requirements of a particular manufacturer.
(9) **The Cost of PCB Errors:** The importance of detailed design review and attention to detail in PCB design.

The last point represents a significant departure from most other areas of computer science, where the compile-test-debug cycle is extremely short and fixing many errors is nearly free. In the class, an error in a student design can delay their project by at least two weeks – an eternity in a ten-week class. In the real world, PCB design errors can take months to address.

Student Challenges Students face two types of challenge in this part of the class.

The first is using the tools. We use the "standard" version of Autodesk's Eagle PCB design package. It is free for students and relatively simple to use, but its interface has some rough edges.

Second, some students struggle with the "correct by inspection" requirement that comes with designing PCBs. For some, this requirement is stressful. Others struggle to pay enough attention to detail while reviewing their design.

Staff Challenges The main challenge for the course staff is performing design checks to help ensure the student's PCBs will work. The typical quadcopter schematic has ~190 electrical connections and an error in any of them can cause the board to fail. In addition, there are numerous constraints that the PCB layout must meet in order to work properly. Checking these is tedious, time-consuming, and the staff can easily overlook subtle problems. The workload compounds because the students must keep revising their designs until they are correct.

We address this challenge in three ways. First, we require the students to "pay" for design reviews (see below). Second, students perform a peer design review for the schematic and another for the PCB layout. Third, the staff only performs careful design checks on the final schematic and layout. The earlier labs rely mostly on automated design checks (See Section 5).

Teaching The PCB portion of the course moves very quickly and starts the first day of class. In quick succession, students learn the basics of Eagle (Lab H1 – 2 days), how to build high-quality PCB libraries (Lab H2 – 1 week), and then design (Lab H3 – 1.5 weeks) and layout (Lab H4 – 1.5 weeks) their PCB.

We introduce each topic with a lecture and then describe the lab and demonstrate the key features of Eagle they will use. We also describe the electrical components they will use and discuss how to interpret and use datasheets and reference designs.

We use an unusual method for grading the hardware labs that reflects the fact that, in the real world, PCB design errors are expensive. Students "pay" for design reviews (by course staff or QuadLint) with points deducted from their lab grade. This incentivizes them to find and fix problems themselves by inspection rather than relying on QuadLint or the staff.

The labs are worth 10 points. We give them 12 points to start and subtract 0.5 points for each design review they request. Completing a lab requires passing both a QuadLint review and a human review, so they can score up to 11 out of 10. We do not deduct points for anything else on the PCB design labs – the lab must be completed correctly for them to move on. This may seem harsh, but the alternative is manufacturing a PCB with known flaws, which is pointless.

3.2 Implementing Flight Control Software

The flight control software task provides students with first-hand experience implementing a software system that controls a real-world device, comprises two communicating components (the remote and the quadcopter), and presents challenging debugging problems. The two central challenges are 1) combining the inputs from the gyroscope and the accelerometer to provide accurate, responsive measurement of the quadcopter's orientation and 2) implementing and tuning a proportional, integral, derivative (PID) [14] controller to control that orientation.

We provide some pre-built equipment for this portion of the course: a PID test stand (Figure 1 top-middle). The test stand supports a mockup that closely matches components of the quadcopter they will build. It has a microprocessor break-out board (red) [26, 27][3], an IMU breakout board (blue) [17, 18], and a laser-cut plywood frame. It has a pivot that allows the mockup to tip back and forth, simulating a quadcopter that moves on one axis.

We leverage the open-source library support for the IMU [5, 7, 8] and microcontroller [9, 26] provided by SparkFun [1] and Adafruit [2].

Key Topics This task addresses the following concepts and skills:

(1) **Attitude sensing:** How to measure orientation by combining gyroscope and accelerometer measurements.
(2) **Sensor Filtering:** How to use the IMU's built-in filtering facilities to reduce the burden on software.
(3) **Complementary Filtering:** How to use a complementary filter to combine sensor inputs to provide more stable, lower-noise measurements.
(4) **Basic PID control:** How PID controllers work and how to implement and tune them in software.
(5) **Debugging physical systems:** How to debug and tune software that controls a physical system.

Items two through four are the subjects of entire courses (which most student have not taken). We cover the basics and provide intuition for the underlying theory.

Student Challenges Students struggle with several aspects of this task. The first is that the notion of "correctness" for both the filtering and PID code is qualitative rather than quantitative. We provide guidance about the how algorithms should behave, but there is not a crisply-defined "correct" answer.

Second, poor performance can stem from many sources: Conceptual misunderstanding, algorithm implementation errors, arithmetic problems (e.g., overflow), misconfiguration of the IMU, and poor parameter settings. Finding the root cause of a problem can be hard.

Staff Challenges The main staff challenge is to help the students in debugging their code efficiently. The algorithms are not very

[3]Unfortunately, Sparkfun has discontinued this breakout board. We have a good number of them, but we are migrating away from it (see Section 7).

complex, but there are many ways to implement them and getting oriented in each team's code base is impractical.

Instead, when they face a problem, we have students verify that individual components of the system are working properly starting with the simplest. For instance, for trouble with a PID controller, we ask them to verify the correctness of their raw sensor readings, then their filtering code, etc. to identify where, exactly, the problem lies.

Teaching We divide this task into three labs, each with an associated lecture. The first (Lab S1 – 1 week) covers the basic hardware components on the remote and the quadcopter. They assemble code from example programs to demonstrate that they can read sensor data from the IMU and control the speed of motors on their test stand using the remote control.

Lab S2 (1.5 weeks) and the associated lecture covers sensor filtering. The students use a complementary filter [15] combined with the IMU's built-in high- and low-pass filters to generate accurate, low-noise orientation measurements.

In Lab S3 (1.5 weeks) they implement a PID controller to stabilize and control the pitch of the test stand. They must implement PID correctly, tune its parameters, and translate the controller's output to motor power levels.

3.3 PCB Assembly

This task teaches how to assemble a moderately complex PCB. It is a delicate and potentially frustrating process.

Key Topics We teach the following skills:

(1) **Reflow soldering:** How reflow soldering works and how to achieve good results, including how to applying solder paste to the board.
(2) **Placing parts:** How to place parts precisely with tweezers and how to ensure the correct orientation of polarized parts.
(3) **PCB rework techniques:** How to fix common problems in hand-assembled PCBs.
(4) **Hand soldering:** How to solder through-hole components by hand.
(5) **Flashing the bootloader:** How to install the low-level firmware on their microcontroller.
(6) **Testing and bring up:** How to systematically check that each component of their quadcopter is functioning properly.

Student Challenges Assembling boards is hard for several reasons. First, many students have no experience soldering. Common problems include a lack of patience, applying too much solder paste, and applying it imprecisely.

Soldering the IMU is especially challenging. In the latest iteration of the class, the failure rate was 75%, and several teams resorted to "hot wiring" the IMU breakout board onto their quadcopter. This is a clear area where we need to provide a better method[4]

Staff Challenges This has emerged as the most labor-intensive section of the class for the staff, especially since QuadLint has taken on many of the design reviews. The boards come back from the manufacturer in batches, so many groups need intensive, one-on-one guidance about part placement, etc. at the same time. This leads to extremely busy staff, extended lab hours, and a lot of student waiting.

Teaching We provide a brief lecture about reflow soldering and a small group tutorial about applying solder paste and placing parts.

[4]Interestingly, in previous years the success rate has much higher. Several things have changed since then, but the most likely culprit seems to be low-quality solder paste.

3.4 Flight

In the final stage (Lab H6) of the course, students combine their PID controller software with their PCB to create working, flying quadcopter. This includes extending the flight control software from one-axis (pitch) to three (pitch, roll, and yaw), loading it onto their PCB, tuning it to achieve stable flight. Stable hovering counts as success.

Key Topics The key skills covered include the following:

(1) **Integration and testing:** How to generalize their PID controller to work in a more challenging environment.
(2) **Flying a quadcopter:** How to maneuver a quadcopter safely.

Student Challenges Student success during for stage varies widely. Some students get it working quickly, but others struggle, and many do not succeed. In some cases, failure has a clear cause (e.g., their PID from Lab S2 did not work well). In other cases, the cause is less clear. It would be useful to perform interviews with teams to understand what went wrong.

Staff Challenges The challenges here are similar to those for the flight controller portion of the class. The staff mostly provide debugging and moral support to frustrated students.

Teaching There is no additional material to present for this portion of the class. Students start this lab when they have finished Lab H5 and Lab S4 and work on it until the end of the quarter. In the best case, they have three weeks. Some have just a few days.

4 LOGISTICS

Running the course successfully has required us to carefully manage class size and the course schedule. We have also worked to craft a grading scheme to reward student effort and account for the challenging nature of the course.

Please see the appendix of [30] for a more detailed discussion of course logistics.

4.1 Admissions

Students must apply to take the course. We advertise to graduate and undergraduates across the school of engineering, drawing students mostly from computer science, electrical engineering, mechanical and aerospace engineering, and math.

We accept students based on some demonstration of success in projects-based coursework and GPA. We also prioritize students graduating before the next iteration of the class. The last time we taught the class (Spring 2018), we had 119 applicants, accepted 41, and 24 ended up attending the class: Twelve computer science, computer engineering, or robotics students, nine mechanical or aerospace engineering, and three math.

4.2 Schedule

The course schedule is very tight, since our university runs on 10-week quarters. We meet twice a week with a 1.5-hour lecture section followed immediately by 1.5 hours of lab time. The boundary between the lecture and lab is blurry.

We run the hardware and software labs in parallel. We lecture about and then start a new lab roughly every day for the first three weeks of the quarter. From that point on, groups progress at different rates.

Figure 2 illustrates the nominal schedule (down the center of the figure) and the slippage that invariably occurs (the outer edges). The largest delays are in finalizing the PCB layout, assembly, and completing the PID controller and flight software.

In earlier versions of the course, we noticed that students (especially computer science majors) would devote too much time to

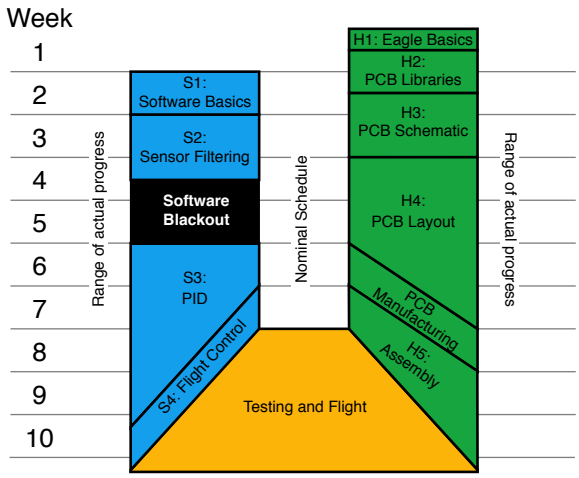

Figure 2: Course Schedule – The schedule for the class moves quickly but accommodates delays. The nominal schedule (down the center) leaves over three weeks for students to get their quadcopters flying. In practice, the timeline varies widely between groups (the edges).

Lab S2 rather than finalizing their PCB layouts. To prevent this, we added a "software blackout" when they must focus only on their PCB layouts and the course staff refuses to answer questions about the software labs.

We do not enforce deadlines on any of the labs after Lab H3.

4.3 Grading

Aside from the "pay-for-review" grading mechanism described above, grading in the course is a combination of participation (showing and doing work), and "checking off" the completion of the labs. Our experience is that students, almost without exception, have worked hard and learned a great deal, even if their quadcopter does not fly in the end. Generally speaking, working hard in the class will earn a "B", and any semblance of flight earns an "A". Stable flight rates an "A+".

4.4 Staffing

The course staffs usually consists of the instructor and one TA. When we have taught the class, the instructor has been deeply involved in designing and running the labs. The staff needs to be familiar with all of the course content and have practical experience with designing, assembling, and debugging PCBs. Experience with PID control is also very useful, although that (and the other software components) are easier to pick up "on the fly," if the staff has general experience with Arduino programming, since errors are less costly and debugging is easier.

The best preparation for teaching the class is doing the project start-to-finish.

4.5 Equipment

The course requires hand-soldering, reflow soldering, and the facilities to build a test stand.

A good soldering iron is necessary, but not terribly expensive. We use soldering irons similar to the Hakko FX-888D ($110).

For reflow, we use a high-end reflow oven, but this is not necessary. The first three iterations of the course used a "heat gun" (i.e., paint stripper) instead of the reflow oven and the results were very good. There is also a rich DIY reflow soldering community online.

We use the laser cutter to build the test stands, but in earlier iterations we 3D-printed the test stands or let the students devise their own testing rig. They showed remarkable ingenuity with materials including rubber bands, cardboard, and popsicle sticks.

5 AUTOMATED DESIGN CHECKS

Checking PCB schematics and layouts is a time consuming, error-prone, and critical to maximizing student success in the class. In the earlier versions of the course, the course staff would quickly become overwhelmed with design reviews. This led to uncaught errors and exhaustion. The effort required effectively limited the number of students we could accommodate in the class.

To reduce this burden, we developed a custom, web-based course management tool called QuadLint that performs automated checks on student PCB schematics and layouts and some other useful classroom functions (e.g., tracking student progress and submitting labs).

QuadLint is, we believe, the first autograder that checks specific design requirements for PCB designs. All PCB design tools provide a suite of design rule checks (DRC) and some PCB manufacturing houses provide automated tests for manufacturability [13], but QuadLint provides an extensible, programmable mechanism for checking the higher-level correctness of designs.

5.1 Checking Designs

QuadLint has two modes. The *quick check* mode generates warnings that flag common violations of good PCB/schematic design style, similar to the checks `lint` performs on source code. These style checks could be applied to any PCB design. Failure to fix a warning will not, in itself, negatively impact the function of the PCB. Students can run quick checks as frequently as they want.

QuadLint's *full check* mode identifies errors that are likely to cause their PCB to not work correctly. Most of these checks only apply to quadcopters designed in this class: They are extremely specific to the reference designs, datasheets, and specifications we provided. The students "pay" for each full check with 0.5 points off their score for the current lab. This prevents them from relying on QuadLint – if they ever design a PCB on their own, they will not have the benefit of QuadLint's detailed checks to catch their errors.

QuadLint lets students explain why they feel a particular error or warning is unjustified. Once they have fixed or explained all their errors and warnings, they submit the design for human review (which costs another 0.5 points). The course staff looks at their explanations and can approve or reject them and provide written feedback. If errors remain, the students fix them and resubmit. If there are no errors, they move on to the next lab.

QuadLint can perform a nearly complete check of student schematics (i.e., QuadLint passes, it will work). Checking PCB layouts is more difficult because many of the requirements are geometric or spatial.

5.2 Implementation

QuadLint runs in the cloud on Google's AppEngine [21]. It is written in Python and relies heavily on the Swoop library [29] for reading and manipulating Eagle PCB design files. We are happy to share the source code with educators.

6 STUDENT FEEDBACK

Based on anonymous course evaluations administered at the end of the class, students enjoy the class and find it valuable. Among 12 respondents from the most recent class of 24 students, 100% of students rated their enjoyment of the class a 4 ("some") or 5 ("a lot") out of 5. 93% of the students answered "yes" or "probably" (4

or 5 out of 5) to whether they could build a PCB on their own, all of them felt more confident soldering, and 91% felt more confident in debugging electronics. 70% of students thought the pace of the course was "just right" or "could have been a bit faster", and the remainder felt it "could have been a bit slower."

7 FUTURE IMPROVEMENTS

Planning is already underway for the next iteration of the course, and we are making some significant changes and smaller adjustments to improve the student success rates and reduce (or least more evenly distribute) work for the staff. Below we detail the planned changes for each of course's four main task.

7.1 Designing and Manufacturing PCBs

We are continuing to refine QuadLint to catch more common (and uncommon) errors in student designs and we are also adding features that will make it easier for less expert course staff to perform design reviews. We are working on ways to make the peer-design reviews more useful and effective.

7.2 Implementing Flight Control Software

The breakout board-based PID testing setup leads to several problems: Building it takes time and the electrical connections can be unreliable which leads to problems in the software labs.

We are addressing all of these problems by replacing the breakout-board based design with a newly-designed PCB that integrates all the components of their final design. This will attach to laser-cut arms to hold motors. The result will essentially be a quadcopter that fits on a newly-designed, larger test stand (Figure 1-bottom-middle) that will also accommodate their final quadcopters. The new board will use the same components and circuits as their final designs, so the work they do in the software labs should transfer better to their quadcopters.

7.3 PCB Assembly

The biggest problems we face in PCB assembly are the low success rate for soldering IMUs, the huge spike in staff workload that occurs during the assembly process, and the amount of time it takes students to assemble their quadcopters. We plan several changes that will address these challenges.

The first is that we have been experimenting with better soldering techniques for the IMU and considering different IMUs that come in easier-to-solder packages [6].

The second is that we will have students assemble the PCB for their remote control. The remote control assembly lab will run early in the course, and we will divide the class into 2-3 cohorts that will do the lab at staggered times, to make teaching and supporting them more manageable. We will provide "loaner" remotes to ensure that groups can make progress on other labs if assembly goes poorly.

Finally, we will use stencils to apply solder paste to the PCBs instead of plastic syringes. It is faster, more precise, yields better-looking PCBs, and is how "real" PCBs are assembled.

7.4 Flight

No significant changes are planned, but the changes outlined above should leave students with more time at the end of the class to get their quadcopters flying.

7.5 Cost

A frequent complaint about the course is that the students cannot continue working on their quadcopters after the course ends since they cannot keep the remote controls. The reason for this is that

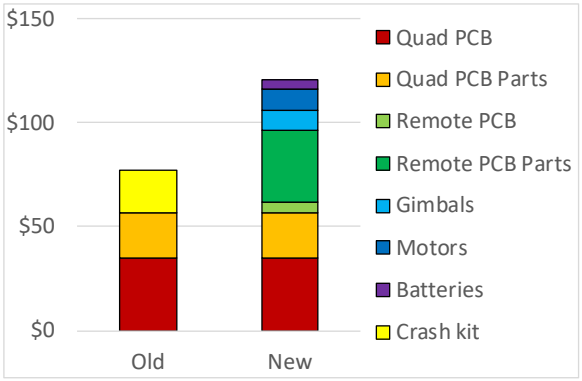

Figure 3: Per-student Costs – The per-student cost for the most recent iteration (left) did not include the remote. Our planned changes will raise costs, but students will have everything they need to program their quadcopter after the class.

the remote controls (Figure 1-top-right) we have been using are expensive – ~$150. The quadcopter costs about ~$35 per board plus $40 in for the components (including a "crash kit" [10] that provides batteries, motors, etc. as spare parts for a small, commercial quadcopter), so charging a lab fee large enough to provide a full set of hardware for each student to keep is not feasible.

We have set a goal of reducing costs enough to let each student keep a remote, a quadcopter, and all the other necessary equipment for ~$120. Figure 3 shows the breakdown in per-student cost for the last iteration of the class and the projections for the new version. Note that the per-student cost goes up, but in the new version, they can keep the remote.

The new remote (Figure 1-bottom-right) dispenses with useful-but-expensive on-board LCD for displaying status information ($25). We also added an integrated LiPo battery charger, so they can charge their own batteries. The remote uses the same kind of battery as the quadcopter. Further savings come from buying electrical components [11], motors [3], and batteries [4] in larger quantities (i.e., enough for 2-3 years of the course). Ordering in small quantities costs $10-$20 per student.

8 CONCLUSION

Many of the "killer apps" of the near future will lie at the intersection hardware, software, sensing, robotics, and/or wireless communications. Building a quadcopter gives students hands-on training in all of these aspects of system design simultaneously.

Building a quadcopter from scratch is 10 weeks is challenging, and fitting all of the necessary material into a single academic quarter has required careful planning and refinement over several iterations of the course. Students almost universally enjoy the class and report they learn many useful skills over the course of the quarter.

There is still much room for improvement. We are planning changes that should increase student success rate and allow us to scale the class to meet the growing demand for project-based courses from students.

ACKNOWLEDGMENTS

We would like to thank the reviewers for their constructive comments and Leo Porter for his comments and guidance. We would also like to thank Jorge Garza for TAing the course several times and debugging many, many problems.

REFERENCES

[1] http://sparkfun.com.
[2] http://adafruit.com.
[3] 4pcs 8520 motor 16000kv 1.0mm shaft jst1.25 connector. https://www.alibaba.com/product-detail/4pcs-8520-motor-16000kv-1-0mm_60753394316.html.
[4] 5pcs tenergy 3.7v 380mah lipo battery. https://www.amazon.com/gp/product/-B00HS5Y6G4.
[5] Adafruit lsm9ds1 library. https://github.com/adafruit/Adafruit_LSM9DS1.
[6] Adafruit precision nxp 9-dof breakout board - fxos8700 + fxas21002. https://www.adafruit.com/product/3463.
[7] Adafruit unified sensor driver. https://github.com/adafruit/Adafruit_Sensor.
[8] Ahrs (attitude and heading reference system) for adafruit's 9dof and 10dof breakouts. https://github.com/adafruit/Adafruit_AHRS.
[9] Atmega128rfa1 development board. https://github.com/sparkfun/-ATmega128RFA1_Dev.
[10] Avawo for hubsan x4 h107c parts crash pack 8-in-1 quadcopter red/white spare. https://www.amazon.com/gp/product/B00RROB6Q4.
[11] Digikey. http://digikey.com.
[12] Jlcpcb. https://jlcpcb.com/.
[13] Pcb file checker. https://www.4pcb.com/pcb-preorder.html.
[14] Pid controller. https://en.wikipedia.org/wiki/PID_controller.
[15] Reading a imu without kalman: The complementary filter. http://www.pieter-jan.com/node/11.
[16] Eee 802.15 wpan™ task group 4 (tg4). http://www.ieee802.org/15/pub/TG4.html.
[17] Adafruit 9-dof accel/mag/gyro+temp breakout board - lsm9ds1. https://www.adafruit.com/product/3387.
[18] Adafruit lsm9ds1 accelerometer + gyro + magnetometer 9-dof breakout. https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout.
[19] 8-bit avr microcontroller with low power 2.4ghz transceiver for zigbee and ieee 802.15.4. https://www.microchip.com/wwwproducts/en/ATmega128rfa1.
[20] R. W. Beard. Robot soccer: an ideal senior design experience. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 6, pages 3975–3979 vol.6, June 2000.
[21] Google app engine. https://cloud.google.com/appengine/.
[22] D. J. Jackson and K. G. Ricks. Fpga-based autonomous vehicle competitions in a capstone design course. In *2005 IEEE International Conference on Microelectronic Systems Education (MSE'05)*, pages 9–10, June 2005.
[23] J. C. Jensen, E. A. Lee, and S. A. Seshia. An introductory capstone design course on embedded systems. In *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, pages 1199–1202, May 2011.
[24] A. Saad. Mobile robotics as the platform for undergraduate capstone electrical and computer engineering design projects. In *34th Annual Frontiers in Education, 2004. FIE 2004.*, pages S2G–7, Oct 2004.
[25] A. Saad. Senior capstone design experiences for abet accredited undergraduate electrical and computer engineering education. In *Proceedings 2007 IEEE SoutheastCon*, pages 294–299, March 2007.
[26] Atmega128rfa1 dev board hookup guide. https://learn.sparkfun.com/tutorials/-atmega128rfa1-dev-board-hookup-guide.
[27] Atmega128rfa1 development board. https://www.sparkfun.com/products/retired/-11197.
[28] inemo inertial module: 3d accelerometer, 3d gyroscope, 3d magnetometer. https://www.st.com/en/mems-and-sensors/lsm9ds1.html.
[29] Swoop. https://pypi.org/project/Swoop/.
[30] Steven Swanson. Trial by flyer: Building quadcopters from scratch in a ten-week capstone course, 2018. https://arxiv.org/abs/1810.07646.