PiDrone: An Autonomous Educational Drone using Raspberry Pi and Python

Isaiah Brand[1*], Josh Roy[2*], Aaron Ray[3], John Oberlin[4], Stefanie Tellex[5]

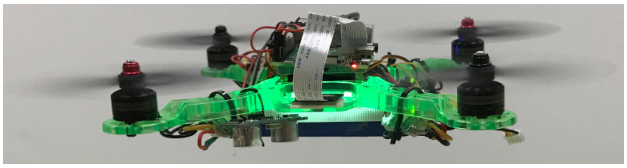
***Abstract*— A compelling robotics course begins with a compelling robot. We introduce a new low-cost aerial educational platform, the PiDrone, along with an associated college-level introductory robotics course. In a series of projects, students incrementally build, program, and test their own drones to create an autonomous aircraft capable of using a downward facing RGB camera and infrared distance sensor to visually localize and maintain position. The PiDrone runs Python and the Robotics Operating System (ROS) framework on an onboard Raspberry Pi, providing an accessible and inexpensive platform for introducing students to robotics. Students can use any web and SSH capable computer as a base station and programming platform. The projects and supplementary homeworks introduce PID control, state estimation, and high-level planning, giving students the opportunity to exercise their new skills in an exciting long-term project.**

I. Introduction

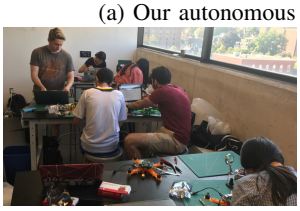
The increasing prevalence of robotic and autonomous systems in everyday life increases the demand for engineers to develop and produce these systems, and also generates a need for more people to understand these technologies and their impact on society.

While there are a variety of robotics courses currently offered in universities worldwide, many of these courses are targeted at advanced undergraduate or graduate students. The wide range of prerequisite knowledge required for these classes is necessary for some interesting applications in robotics, but the high barrier to entry prevents many students from taking these courses and being exposed to the field. Generally, courses that take a more introductory approach either use simulation or ground robots in restricted domains. While these simplified architectures and environments serve to introduce specific tools used in robotics, they often fail to prepare students for the challenges and complexity associated with robotics in the real world.

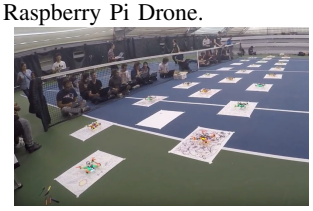
The choice of a flying platform provides an opportunity to teach lessons about robot safety and potential impacts on society, as well as concrete skills for a growing aerial robotics industry. The novelty and potential capabilities of a flying platform also serve to keep students engaged. Of course, flying robots have complex hardware, non-trivial control, and low fault tolerance. These factors can make UAVs more difficult to work with than ground platforms, but we took the challenges associated with a flying platform to be a pedagogical opportunity because they paint an accurate picture of obstacles faced by real, situated robots.



(a) Our autonomous Raspberry Pi Drone.



(b) Students building their drones.



(c) Class demo day

Fig. 1: The hardware platform and scenes from our class.

To create an introductory robotics curriculum that is accessible, compelling, and realistic, we developed our own educational robot, the $220 PiDrone, an autonomous drone based on the Raspberry Pi. All autonomous behavior is implemented onboard in Python using the ROS framework [1]. An onboard camera, infrared distance sensor, and IMU allow the drone to maintain target velocities and positions.

We also developed an introductory college-level robotics course with the PiDrone at its core. The course can be taken directly after a first-year computer science course and expects only basic knowledge of Python, enabling us to admit undergraduate students with no previous robotics experience. To best give students a comprehensive view of robotics and its role in society, whether or not they continue to work in the field, we developed the following core competencies:

Hardware and Assembly: Robots are programmable hardware and, accordingly, many problems are due to hardware issues. Our course introduces students to hardware assembly skills such as soldering, as well as debugging hardware/software stacks.

Closed Loop Control: Robots must take actions in the physical world. Our course introduces students to feedback control with a PID controller in several contexts.

Estimation and Sensor Fusion: Robots must estimate the state of the world. Students learn to fuse information from multiple sensors and smooth estimates.

Infrastructure: Robots require multiple processes to coordinate different sensors and actuators. Our course acquaints

*Denotes equal contribution
[1]Brown University brand@brown.edu
[2]Brown University josh_roy@brown.edu
[3]Brown University aaron_ray@brown.edu
[4]Brown University oberlin@cs.brown.edu
[5]Brown University stefie10@cs.brown.edu

students with message passing architectures using ROS [1], because it is an industry standard. We also emphasize networking, development workflow in Linux, and other fundamentals.

Safety and Regulatory Concerns: Robots are potentially dangerous tools. Our course teaches students to use the PiDrone safely and understand the complex regulatory framework that surrounds unmanned aerial systems.

In this paper we present the PiDrone and associated introductory robotics course, as well as an evaluation and characterization of the PiDrone's flight and autonomous capabilities. For a video overview of the course, please see the video submission and a longer version on YouTube [1].

II. Related Work

The PiDrone and class have some similarities to existing courses and robotic platforms but fill a gap in available introductory-level robotics courses and extensible drone platforms for education.

MIT's Feedback Control Systems [2], offered by Sertac Karaman, gives each student a Parrot Rolling Spider which they program and use to develop control systems. Karaman's students develop advanced control algorithms using Matlab and Simulink which run onboard the Rolling Spider using a custom firmware. Our use of the Raspberry Pi increases the computational power onboard, enabling us to present a solution in Python and ROS, increasing accessibility. Additionally, our course includes an introduction to the hardware components of the drone, giving students a broad introduction to robotics.

Another drone-centric course, the Udacity Flying Cars Nanodegree [3], aims to teach students about advanced concepts in drone technology. It is an advanced course that prepares students for work with unmanned aerial systems, while our course uses drones as an introduction to the whole field of robotics. Additionally, Udacity's course focuses on simulation, with an optional physical extension, whereas a physical robot is integral to our approach.

MIT's Duckietown class [4], [5] uses multiple autonomous Raspberry Pi ground robots to teach concepts in autonomous driving. This course is more advanced than ours, providing an opportunity to study multi-robot interactions at a small scale, but requires significantly more background in robotics. In addition, it uses a ground robot instead of an aerial robot.

We considered several off-the-shelf drone platforms to use in the course. Table I demonstrates the differences between the PiDrone and the other platforms we considered.

The Parrot AR Drone [6] shares many capabilities with the PiDrone, but it does not officially allow users to program the onboard computer. Instead, the user sends commands to the existing software with a phone or computer. For extensible programming, students need a ROS-enabled base station, which is a significant burden when teaching a large class. By using ROS on-board the Raspberry Pi, we allow students to use any SSH and Javascript capable computer as a base station, without reinstalling their OS. The Parrot AR Drone is also a discontinued product, so it is not feasible for our course.

[1] https://youtu.be/SoBIIoTgz5M

TABLE I: Low Cost Educational Robotic Platforms

Product	Cost	Onboard Python	ROS	Onboard GNU/Linux Compute	Open/Extensible Hardware	Flies
Pidrone	$220	✔	✔	✔	✔	✔
Crazyflie	$180	✘	✔	✔	✔	✔
Rolling Spider	$40	✘	✘	✘	✘	✔
Parrot AR drone[1]	$300	✘	✔	✔	✘	✔
Duckietown	$150	✔	✔	✔	✔	✘

[1] Discontinued

The Crazyflie [7] is a tiny WiFi-enabled drone developed specifically to support experimentation and modification. The platform is open-source and well documented and supports add-on boards that extend its capabilities. The Crazyflie has been successfully used in a class at UC Berkeley [8]. Though this aircraft was a contender our course because of it's small size and open-source community, the limited compute power onboard, lack of support for ROS, and prebuilt nature of aircraft didn't satisfy all of our goals for the class. Critically, at the time we were unable to find instances of the Crazyflie being used with an onboard camera (though an optical flow add-on sensor has since become available for the platform [9]).

Picopter [10] has developed a similar platform to ours, but it is targeted at Makers and drone enthusiasts rather than an educational robotics setting. It focuses on racing control with GPS localization rather than autonomy. Additionally, the Picopter is not currently available for sale, so it does not provide a feasible platform for our course.

After a survey of available systems, we concluded that our goals could best be met by developing our own educational robot. We wanted our drone to be approachable, extensible, ground up, and powerful — metrics we felt were not entirely satisfied by existing platforms. We think the resulting PiDrone fills a gap in the existing educational drones.

III. Robot Architecture

The PiDrone platform is designed to be inexpensive, robust, extensible, and autonomous. Components for the platform are readily available online and are sourced almost entirely from HobbyKing.com. A complete parts list can be found on our course website [11].

The aircraft is built around a durable, single-piece plastic frame. It uses 5 inch, 3-bladed props that were found to be particularly resistant to shattering in crashes. We chose brushless DC motors with reinforced shafts and bearings that proved much more resilient than many other motors we tested. This component adds significantly to the price, and in the future we plan to test alternatives in order to reduce the platform cost.

TABLE II: Parts List

Subsystem	Item	Cost
Computation	Raspberry Pi	$35
Avionics	Skyline32	$15
Avionics	4 Motors	$60
Avionics	4 ESCs	$40
Sensors	IR distance sensor	$20
Sensors	Pi Camera	$15
Power	Battery and Charger	$30
Hardware	Frame and Hardware	$5
Total Cost		$220

The motors are driven by Afro 12 Amp optoisolated electronic speed controllers (ESCs) because they proved reliable in testing, and their power and signal indicator lights are useful for students when debugging hardware problems with their drones. The ESCs were reflashed with BLHeli firmware instead of the default SimonK firmware, as the new firmware improved the response of the motors to rapidly changing throttle inputs produced by the PID controller.

Power for the system comes from a three-cell 1500 milliamp-hour lithium polymer battery. Every student was given three batteries and two chargers. With three batteries, students can fly almost continuously by charging their batteries in rotation, with a flight time of 7 minutes on one battery.

The low-level attitude control and the higher-level autonomy control run on two separate boards. The Skyline32 Acro, an off-the-shelf flight controller for racing drones, runs low-level control; it has an onboard 6-axis IMU and generates PWM commands which are sent to the ESCs to keep the drone at the desired attitude and throttle.

Students implement higher-level controllers in Python on the Raspberry Pi 3, which is connected to the Skyline32 via USB. Although alternatives to the Raspberry Pi like BeagleBone and Intel Edison offer similar or greater compute power, the Raspberry Pi is documented extensively online and has a vibrant open source community. This is important for ensuring that PiDrone is accessible. We use the Raspberry Pi 3 because it features onboard WiFi and offers the most compute power of the Pis — we want students to be able to implement powerful autonomy and continue to expand the capabilities of their PiDrone beyond the course.

IV. Software Architecture

The software stack for the PiDrone, as shown in Figure 2, is split into sensing nodes, control nodes, and nodes for communicating with the base station. The Cleanflight [12] firmware that runs on the Skyline32 uses a PID to keep the drone at a desired attitude. A set of Python ROS nodes on the Raspberry Pi estimate altitude, position, and velocity, and generate attitude commands for the Skyline32. The drone is controlled from the base station via a Javascript interface, shown in Figure 3, and SSH over the network allows for editing and running scripts on the Raspberry Pi.

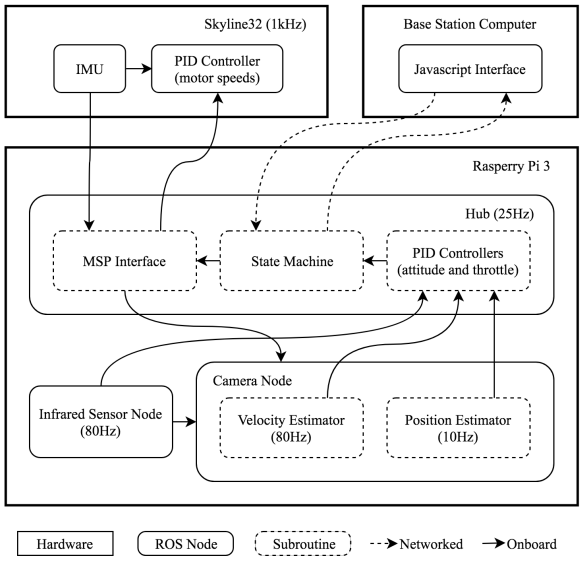


Fig. 2: PiDrone software architecture.

A. Design Decisions

The primary goal for the PiDrone is to be a powerful tool for teaching the aforementioned core competencies and to provide a very realistic introduction to robotics for students. The course was limited to a single college semester, so we designed projects and obstacles to teach the appropriate concepts in a timely fashion. Due to time and safety constraints, we restricted the domain over which the drone would be flown to flat, level surfaces. At the same time, we chose not to provide maps or structured surfaces. This simplifies the vision and localization problems while still providing a realistic and challenging environment in which students developed autonomous capabilities on the PiDrone.

Methods such as structure from motion are beyond the scope of this course and would require significant prerequisite knowledge to fully understand. By electing to use the simplifications detailed above, we ensure that any student with a basic understanding of computer science will be able to understand and implement the algorithms used in velocity and position estimation.

The software on the PiDrone is written in entirely in Python, as the accessibility and teachability of Python outweighed performance benefits of a faster language. See the Robot Performance section for a comparison between Python and C++ on the drone. By using Python, students are able to focus more on the concepts and algorithms used in the software of the drone rather than the programming language used to implement it.

The software stack of the drone is built around ROS. This allows for extensibility of the drone, since much pre-written software is compatible with ROS. Furthermore, it exposes students to ROS, an industry standard, giving them skills to develop software for many other robots.

B. Hub

The Hub contains PID controllers and the state machine. In velocity mode, the Hub accepts velocity estimates and uses a PID to drive the drone's estimated velocity to its target. In position mode, the Hub accepts position estimates modulates the velocity controller set-point to reach a target position. A third PID uses the height estimates from the infrared node to modulate the throttle and drive the drone to a target altitude. The state machine can be in armed, disarmed and flying states, and regulates communication with the Skyline32 depending on the mode. Attitude commands are sent to the Skyline32 via the MultiWii Serial Protocol (MSP) interface [13].

C. Low Level Control

Low-level attitude control is handled by the Skyline32 Acro flight controller. The Pi continuously sends desired angle and thrust commands to the Skyline32; if the Skyline32 stops receiving commands, it enters a failsafe mode and disarms the drone. This timeout ensures that the drone acts safely even if the controlling program on the Raspberry Pi crashes. The Cleanflight firmware running onboard the Skyline32 can be configured with a free Chrome App — this was advantageous for the class because students could use their own laptops to flash and configure the Skyline32.

D. Infrared Node

The infrared sensor node reads data from the IR depth sensor and publishes the distance to the floor on a ROS topic. This range data is used by the PID controller for altitude control, and by the position and velocity nodes for normalizing by the distance to the scene. The Raspberry Pi does not have an onboard analog to digital converter (ADC), so we use an Adafruit ADC to send values to the Pi via I^2C. Voltages are converted to meters and published.

E. Camera Node

The camera node fetches images from the camera and contains the velocity estimator and the position estimator. The position estimator only runs when the drone is in position mode.

1) Velocity Estimation: We implemented a fast, efficient velocity estimator by exploiting the Raspberry Pi's GPU H.264 video encoder [14]. The H.264 encoding pipeline uses a motion estimation block, which calculates the most likely motion of 16x16 pixel macro-blocks from frame to frame. The built-in `RaspiVid` tools allow the user to access these motion vectors, which, like optical flow, can be used for simple visual odometry after scaling by the camera's distance from the floor. The H.264 video encoding takes place almost entirely on the GPU, so these motion vectors can be used with almost no CPU overhead. As a result, we can estimate velocity at approximately 80 Hz.

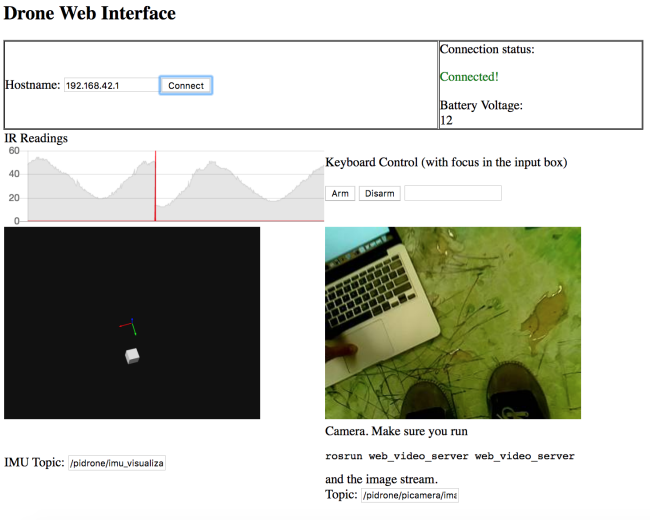


Fig. 3: Screenshot of the Javascript interface showing IMU, range data, and camera information.

2) Position Estimation: Velocity estimates alone are not sufficient to allow the drone to stay exactly at a target position or move a target distance. To achieve this, we use a separate position estimator node based on OpenCV's `estimateRigidTransform` function, from which an estimate of the drone's relative x, y, yaw pose is published.

`estimateRigidTransform` finds corresponding features in two images and returns an affine transform from the first image to the second. When position-hold mode is enabled, a video frame is saved to memory. That first frame and subsequent frames are then passed to `estimateRigidTransform` which yields an estimate of the drone's position relative to this first frame. The drone is not always in a position where it can see its first frame, so each frame is also correlated to the previous to get a position delta between frames. The sum of these deltas gives a position estimate for when the drone is unable to see its first frame. When the drone sees its first frame again after having not seen it for a time, disagreements between the two position estimates are smoothed together with an exponential moving average to avoid jerking.

The affine transformation matrix between frames does not account for the roll and pitch of the camera out of the image plane. These degrees of freedom are encapsulated in the homography matrix between frames, yet we found through testing that `estimateRigidTransform` can be run at a much higher frame-rate on the Pi than the corresponding homography estimation pipeline, and the benefits of more frequent updates outweighed those of a more accurate position estimate. Additionally, the drone does not roll and pitch significantly when flying at the slow speeds enforced in the class, making `estimateRigidTransform` a good approximation. This decision was explained in class and used to explain the tradeoffs when using approximations in modeling physical systems.

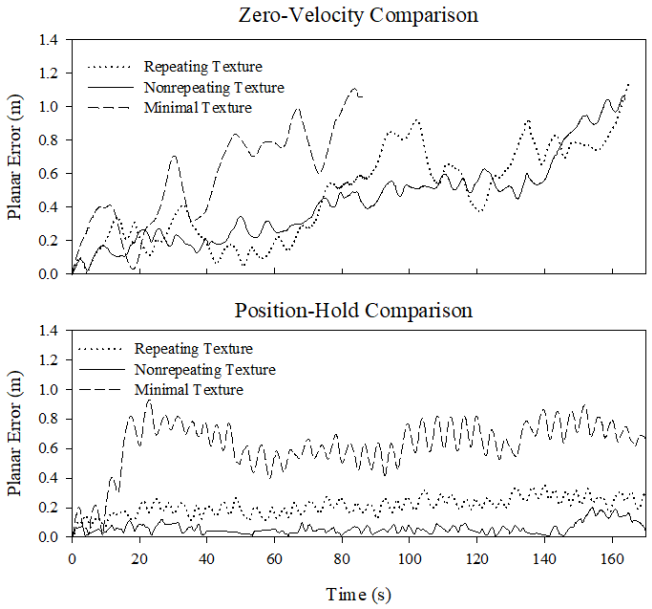


Fig. 4: Planar error for our position and velocity controllers over three different surfaces captured by an external motion capture system.

F. Offboard Software

Students controlled their drone through a webpage built with Robot Web Tools [15]. This interface allowed them to read data from their sensors, arm or disarm the drone, and issue velocity or position commands (Figure 3). Our browser-based approach allows students to fly their drones without needing to download any programs to their base station. Users interested in extending the drone's capabilities beyond the scope of the course can easily interface the drone with an off-board computer using ROS instead of this browser interface. The Raspberry Pi can either host its own WiFi access point or connect to an external network, allowing the base station to connect to the drone locally.

Students used our lab's reactive-variables framework to send high-level sequences of commands to the PiDrone. This framework allows students to sequence behaviors such as "take off, fly forward, hover, perform position hold, wait 1 minute, land." This framework runs off-board on a ROS capable machine, but in the future we plan to explore running it onboard the Pi.

V. Robot Performance

The drone communicates with the Skyline32 at 25 Hz, allowing our software on the Pi to get IMU orientation updates and send new attitude commands to the Skyline32 at 25 Hz.

The drone is able to process motion vectors from the H.264 pipeline and estimate velocity at 80 Hz and position at 10 Hz with 20% and 50% of the Pi's total CPU power respectively. In comparison, an implementation of the same pipeline in C++ without ROS (courtesy of an ambitious student, Frederick Rice) is able to estimate velocity at 85 Hz, position at 13 Hz using 3% and 70% of the CPU respectively. We suspect the increased power consumption in position hold in Rice's C++ pipeline is due to inefficient handling of camera frames. The frequency of velocity estimation appears to be limited by the frame rate of the camera, leading to minimal performance gain in C++. In position hold, both implementations make a call to the same OpenCV function, `estimateRigidTransform`, and most of the overhead is in that function resulting in similar loop frequencies. Thus, we conclude that the performance gained by forgoing ROS and using C++ instead of Python does not outweigh the pedagogical benefits of ROS and Python.

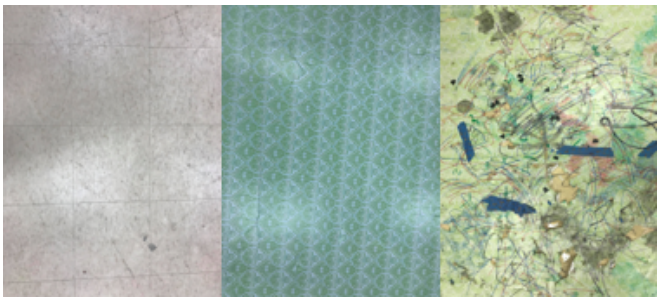


Fig. 5: From left to right: Minimal frequency surface, repeating-texture surface, non-repeating texture surface.

The PiDrone shows robust position-hold performance. Using a motion capture setup for ground truth, we found that the drone was able to remain within 0.2 meters of the target position. In practice this level of performance means the drone can perform position hold over a textured surface for the entire battery life of the drone.

Figure 4 shows the distance of the the PiDrone from a starting point while flying over the various surfaces pictured in Figure 5. The drone is best able to hold its position when flying over a high-frequency, non-repeating pattern. When in velocity mode, the drone performed similarly on repeating and non-repeating textured surfaces, but uniform-color floors or other surfaces without enough high-frequency features left the drone without an accurate velocity estimate. The drone would "slide" along such surfaces, behaving much like a car on a patch of ice.

In the lab-space where students flew most frequently, we provided a patterned cardboard pad embellished with high-frequency hand-drawn designs shown in 5. Multi-colored carpets also worked relatively well for velocity hold. We also found that by moving a textured poster under the aircraft while flying in velocity hold, the PiDrone is able to athletically "chase" what it sees with its downward facing camera as it tries to drive its visually-observed velocity to zero.

During the course, there were rarely more than a few students attempting to fly their drones in our lab space. With this limited number of required WiFi connections, we did not notice any network degradation in most cases. However, when multiple students published a video feed,

the network would only support a few drones at a time. At the end of the semester, we attempted to fly all students' drones at the same time in the same location, as depicted in Figure 1. In this case, we noticed substantial drops in network performance that led to unsatisfactory drone latency. We believe that the network performance loss was a result of the numerous independent and overlapping WiFi networks the drones launched. A possible solution to this problem is having the drones and base stations connect wirelessly to one router.

VI. Class Architecture

After developing the PiDrone platform, we developed a class [16] centered around our core educational goals and the capabilities of the new system. We wanted students to gain an in-depth understanding of all systems on their robot through a series on hands-on projects. We designed these projects to target specific subsystems of the drone, starting from the ground up.

Given that the projects were based mostly around building the drone and its autonomous capabilities, we wanted to give the students some familiarity with prerequisite systems, software, and concepts before applying them to the drone. We created homework assignments to introduce relevant skills before using them on the projects. For example, the first three homeworks were on safety, debugging, and ROS, all important skills that the students required before programming their drones.

Students assembled their drones as the first project. The build process involved workshops to introduce technical skills such as soldering and cable management, as well as a lecture to explain the hardware systems and basic vehicle dynamics. Throughout the course we held students responsible for debugging and maintaining their PiDrones, so developing hardware literacy and understanding of the system upfront was crucial for enabling students to work through hardware issues as they arose. Thorough knowledge of their drone's hardware allowed the students to better implement and debug their software, as they had a better understanding of the underlying physical systems.

The PiDrone's spinning propellers and high speed posed potential safety risks, so it was particularly important to emphasize safety before any students flew their drone. As students worked on assembling their drones for their first project, we spent two weeks discussing safety. We introduced the FAA rules for unmanned aerial vehicles and studied an NTSA crash report in depth as a homework. This process triggered discussions of safety in robotics generally, and the responsibility of the operator and the engineer to ensure safe operation.

Subsequent lectures centered on topics that the students would implement, such as PID controllers, optical flow, and sensor smoothing, using these concepts to introduce an overview of the field of robotics. We consistently compared features of the PiDrone's hardware and software stacks, to the technologies developed elsewhere in robotics in order to show the generalization of the concepts the students learned. There were also several guest lectures from industry and academia to give students a broader taste of the field.

For the second project, students implemented a simple PID in Python to control the altitude of their drone based on range values from the downward-facing IR sensor. They validated and tuned their controllers in a simulated environment before transitioning to testing on the PiDrone in a constrained environment — the drones were mounted to a slider which allowed them only to move up and down. Once they achieved stable control, students removed their drones and their altitude controllers on the free-flying drone. This enabled the students to gain confidence by testing their code in a safe environment before testing on the robot.

The third project required students to estimate the motion of the drone using optical flow. This included sensor smoothing and sensor integration, as students had to use IMU data to correct perceived optical flow for the rotation of the drone. Students then implemented a two axis PID controller which enabled the drone to maintain a target velocity in the plane.

To close the loop and enable position control, students estimated the drones position and wrote controllers to modulate the set-point of the velocity controller to achieve a target position in the fourth project. This introduced a common robotics tool, OpenCV, and demonstrated the power of layered control systems.

With a closed-loop, position-controlled drone, the fifth project explored higher level behavior using our lab's reactive-variable framework. Students were able to program their drones to perform semi-autonomous behaviors, such as hopping from one location to another.

VII. Discussion

The course outcomes validated our choice of drone design, as 24 of the 25 original students successfully built a drone and completed all five projects in the class. While there were several instances of components breaking on students' drones, none needed to be rebuilt from scratch and identifying the broken hardware gave students the opportunity to debug the problem and repair their own aircraft.

We also realized that the class must strike a delicate balance of breadth and depth. On one hand, as we intend the course to be an introduction to robotics, it is important to introduce students to a wide view of the skills required to become a successful roboticist. On the other hand, students must come away with a deep enough understanding of individual topics that they will be able to apply them to real problems. Some students gave feedback that they would have preferred deeper coverage on some topics. For example, students used the optical flow implementation provided by the Raspberry Pi, and we did not cover the algorithm in depth. More technical depth on specific algorithms like this did not fit with our vision for the course, but may fit in similar courses.

Another important piece of student feedback was that PID tuning assignments became too tedious. Students spent substantial time tuning their altitude and planar PID controls. After the experience of tuning the altitude control, the planar

PID tuning often required several more hours of tuning, yielding significantly less educational value.

VIII. Future Work

In future iterations of the course, students will still tune the altitude PID controller as they did in project 2 as tuning is an important skill when working with robotic systems. However, for subsequent PID projects we will give them reasonable parameters that are known to work after they have implemented their own controllers. This may still require some fine-tuning, but will save significant time compared to starting from scratch.

We plan to investigate the addition of an Extended Kalman Filter (EKF) project to the course. State estimation, particularly Kalman Filters, are critical to most mobile robots. Unfortunately, they are significantly more complicated than many of the other topics in the class, so a balance must be struck between the accessibility of the course, and extent to which we ask students to understand an EKF. It is possible that we could present the students with a completed EKF and ask them to make use of the state estimate, but we feel that part of the power of the PiDrone curriculum is that students build the whole robot from the ground up.

We will implement Simultaneous Localization and Mapping (SLAM) on the PiDrone and explore its use in the course. As with EKF, SLAM is more advanced than the material we covered in this first iteration of the course, and the value of adding SLAM as a project will have to be carefully assessed.

If EKF and SLAM are too complicated to explore in the context of the class, implementing them on the PiDrone will increase the capabilities of the platform and enable students or other users to explore higher level autonomy.

We will continue to experiment with different components for the PiDrone to bring down the cost. Moving more functionality into the Raspberry Pi could allow us to remove the Skyline32 and the IR sensor, further reducing the cost. We hope that extensive documentation on the course website and the reduced cost of the drone will make PiDrone useful to students, researchers and hobbyists outside of our course.

The PiDrone course will run again in Fall 2018 at Brown University.

IX. Conclusion

Although the number of students studying computer science is skyrocketing [17], Robotics poses a greater barrier to entry than many other specializations within computer science. While accessible introductory robotics courses do exist, they do not always accurately capture the challenges faced by real-world robotic systems.

As the role of robots in our society continues to expand, we wish to increase and improve robotics education. To this end, we developed a set of core competencies to teach in an introductory robotics course, and we created the PiDrone, a compelling and realistic flying robot that would form the core of this curriculum. After evaluating the course at Brown University in Fall 2017, we conclude that the PiDrone is a powerful platform to prepare the next generation to understand, interact with, and develop the robots that surround them.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grant numbers IIS-1637614, IIS-1426452, and IIS-1652561, the National Aeronautics and Space Administration under grant number NNX16AR61G, and DARPA under grant numbers W911NF-15-1-0503, W911NF-10-2-0016, and D15AP00102.

We would like to thank Amazon Robotics for their donation which funded development and equipment for this course.

We would also like to thank Lena Renshaw, who was a TA with us, and the students who took our class. In particular, Frederick Rice provided the C++ code that we compared with the PiDrone software.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[2] S. Karaman, "Feedback control systems," accessed: February 14, 2018. http://dronecontrol.mit.edu/.

[3] Udacity, "Flying car nanodegree program," accessed: February 14, 2018. www.udacity.com/Flying-Car.

[4] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, L. Carlone, M. Cap, Y. F. Chen, C. Choi, J. Dusek, Y. Fang, D. Hoehener, S. Y. Liu, M. Novitzky, I. F. Okuyama, J. Pazis, G. Rosman, V. Varricchio, H. C. Wang, D. Yershov, H. Zhao, M. Benjamin, C. Carr, M. Zuber, S. Karaman, E. Frazzoli, D. D. Vecchio, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1497–1504.

[5] J. Tani, L. Paull, M. T. Zuber, D. Rus, J. How, J. Leonard, and A. Censi, "Duckietown: an innovative way to teach autonomy," in *International Conference EduRobotics 2016*. Springer, 2016, pp. 104–121.

[6] Parrot, "Parrot ar drone 2.0," accessed: February 25, 2018. https://www.parrot.com/global/drones/parrot-ardrone-20-elite-edition.

[7] Bitcraze, "Crazyflie 2.0," accessed: February 14, 2018. https://store.bitcraze.io/products/crazyflie-2-0.

[8] M. Mueller, "Me 136: Introduction to control of unmanned aerial vehicles," accessed: February 20, 2018. http://muellerlab.berkeley.edu/teaching/.

[9] Bitcraze, "Crazyflie 2.0 flow deck," accessed: March 1, 2018. https://www.bitcraze.io/2017/07/crazyflie-2-0-flow-deck/.

[10] Pictopter, "Picopter," accessed: February 16, 2018. https://www.picopter.org/.

[11] I. Brand, A. Ray, L. Renshaw, J. Roy, and S. Tellex, "Cs1951r: Parts list," accessed: March 1, 2018. http://cs.brown.edu/courses/cs1951r/projects/build/parts.html.

[12] "Cleanflight," accessed: February 25, 2018. http://cleanflight.com/.

[13] "Multiwii," accessed: February 20, 2018. http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol.

[14] L. Upton, "Vectors from course motion estimation," accessed: February 14, 2018. https://www.raspberrypi.org/blog/vectors-from-coarse-motion-estimation/.

[15] R. Toris, J. Kammerl, D. V. Lu, J. Lee, O. C. Jenkins, S. Osentoski, M. Wills, and S. Chernova, "Robot web tools: Efficient messaging for cloud robotics," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 4530–4537.

[16] I. Brand, A. Ray, L. Renshaw, J. Roy, and S. Tellex, "Cs1951r," accessed: February 20, 2018. http://cs.brown.edu/courses/cs1951r/.

[17] C. R. A. (2017), "Generation cs: Computer science undergraduate enrollments surge since 2006," accessed: February 28, 2018. https://cra.org/data/Generation-CS/.