A Teaching System for Hands-on Quadcopter Control

Paul N. Beuchat * Yvonne R. Stürz * John Lygeros *

* Automatic Control Laboratory, ETH Zürich, ZH 8006, Switzerland
(e-mail: {beuchatp,stuerzy,lygeros}@control.ee.ethz.ch).

Abstract: Quadcopters, popular in consumer and commercial applications, are a perfect testing ground for both basic and advanced control techniques. This makes them an ideal platform for offering experiment-based control courses. We present the software system developed to teach hands-on quadcopter control, that we have used to offer a short course to groups of 32 undergraduate electrical engineering students. Our system allows students to work simultaneously, each with a separate quadcopter. The software makes the hands-on class easy to manage, allowing the instructor to focus on the pedagogic aspects.

© 2019, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: Control education using laboratory equipment, general control laboratories, open educational resources, integrating research and teaching, interactive control education tools.

1. INTRODUCTION

Experiment-based classes are an integral part of control engineering education for both undergraduate and graduate degrees (Antsaklis et al., 1999). One style of experiment-based class is where the instructor conducts the experiments and together with a moderately-sized group of (10-30) students observes the results. This style is well suited for experiments that are costly to perform or require bulky equipment, for example, control of a car engine. Another style is hands-on classes where each student, or perhaps a pair of students, conducts the experiment themselves. This style offers a greater learning potential due to direct interaction of the student with the experiment, for example, control of an inverted pendulum. However, the hands-on classes are often condensed into a single 3-4 hour session to achieve sufficient throughput of students. Experiment-based classes with quadcopters fall in between these two styles because a bulky flying space is required but the quadcopters are ideal for gaining hands-on experience with basic control concepts over multiple sessions. This paper describes the multi-agent software system and hands-on course we created to enable teaching a moderate class size of undergraduate students all working simultaneously with quadcopters.

Quadcopters are an ideal pedagogical platform for exposing students to control methods. The steps to achieve stable hover of a quadcopter require several core competencies taught in the undergraduate control curriculum, including non-linear modelling, equilibrium analysis and linearization, continuous-time and discrete-time linear system control techniques, and discrete-time representations. An experiment-based quadcopter class can combine multiple pieces of knowledge in a way that guides students to reach the application and analysis levels of Bloom’s taxonomy (Bloom et al., 1956). Moreover, interaction with the system is an important part of experiment-based control education as it activates the students to be more involved in the learning process (Bencomo, 2004). The availability of small-scale, low-cost quadcopters makes this possible without endangering the students or the equipment (Giernacki et al., 2017). Finally, with multiple students working simultaneously to achieve stable hover with their quadcopters, the instructor is able to create a collaborative learning environment. This not only enhances the learning outcomes (Prince, 2004), but also contributes towards the development of soft skills (Feisel and Rosa, 2005).

A range of software and hardware systems exist for setting up a multi-agent quadcopter platform, but they are either not openly available or not tailored to enabling a dynamic and collaborative classroom environment. The quadcopter platform from Lupashin et al. (2014) has been used to display impressive acrobatic feats, but remains closed source and research focused. The Crazyswarm platform from Preiss et al. (2017), also research focused, is open source and demonstrated a swarm of 49 nano-quadcopters with the control algorithm implemented on-board the quadcopters and the off-board software centralized and streamlined for minimizing delays. This limits the ease of adapting the Crazyswarm framework to a classroom setting where the off-board software needs to be distributed for allowing multiple students to work simultaneously.

The development of Giernacki et al. (2017) uses the same nano-quadcopter and provides a simulation and real-world experiment environment, but lacks the management layer required to run a flexible experiment-based class.

The remainder of the paper is structured as follows. Section 2 describes the multi-agent software system we developed to enable a distributed, flexible, and easy-to-manage environment for teaching quadcopter control. Section 3 presents the course we conduct each semester at the Automatic Control Laboratory of ETH Zürich to achieve the pedagogic goals described. Section 4 closes by highlighting extensions and opportunities that our quadcopter testbed offers.
2. SYSTEM OVERVIEW

This section first describes the hardware components used, and then explains how the software we developed enables students to work both independently and collaboratively. Finally, the key aspects that make the classroom environment simple and robust are illustrated.

2.1 Hardware components

Crazyflie 2.0 quadcopter: This quadcopter is ideal for a classroom environment because it is small, robust, inexpensive, safe to interact with, easy to maintain, and open-source for both the software and hardware. The Crazyflie 2.0 is designed especially for research and education purposes, and features extensively in the literature, see for example: Höög et al. (2015); Campos-Macías et al. (2017); Bucki and Mueller (2018).

Fig. 1 shows one of the Crazyflie 2.0 quadcopters used in our class. The key additions we developed are the blue and red parts, 3D printed using Polylactic acid (PLA) material. The blue part is used to attach the retro-reflective markers needed for the motion capture (Mocap) system described below. The two main advantages of our design are: (i) the markers can be placed at a large separation, allowing easy detection even when using inexpensive Mocap systems with low resolution and (ii) unique arrangements of three markers are easier to create, making the system more robust for distinguishing Crazyflies after an occlusion. The markers themselves are 20mm diameter polystyrene spheres covered with retro-reflective tape. The red part secures the battery on the underside of the Crazyflie, thus lowering the center of gravity, and also allows for the easy attachment of additional payload. With our additions the Crazyflie weighs 32g, and it can still carry an additional 5g payload. The Crazyradio PA4, made by Bitcraze (2019), is used for sending commands from the off-board computer to the Crazyflie, and for receiving the battery voltage and Inertial Measurement Unit (IMU) readings.

Motion capture (Mocap) system: In order to localize multiple quadcopters simultaneously, at high frequency, and with respect to the same inertial coordinate system, we use a commercial Mocap system. Multiple cameras, placed around the flying area, synchronously emit infra-red light and detect reflections from retro-reflective markers mounted on the Crazyflies to be tracked. To distinguish between Crazyflies, a unique arrangement of markers is used for each object. Our software is compatible with the Mocap system produced by Vicon (Vicon, 2019), and a beta version of our software currently under development supports the systems produced by Qualisys (Qualisys, 2019) and OptiTrack (Optitrack, 2019).

The key advantage offered by Mocap systems for a classroom environment is the high accuracy, Meriaux et al. (2017), as this allows the student to clearly experience how changes to the controller are reflected in changes of the quadcopters behavior. Alternative localization technologies viable for quadrotors include ultra-wideband (Hamer and D’Andrea, 2018) and on-board sensing. The former is significantly less accurate than a Mocap system, while the latter adds significant sensor payload and computation to the quadcopter.

2.2 Software architecture

Robot Operating System (ROS): This is an open-source meta-operating system, widely used for developing robotic platforms (Quigley et al., 2009), with a framework based on so-called nodes. A node is a stand-alone executable process that interacts at runtime with other nodes through a variety of message types. The key advantages of ROS that we utilize for our software are:

(1) It facilitates running nodes across multiple computers, killing and relaunching the nodes whenever necessary. This allowed development of an architecture where students can work independently.

(2) It provides message passing directly between these nodes and abstracts the details of the network, communication, and lower layers that actually transfer the data. This allowed development of a distributed architecture where students can perform collaborative control tasks with only a few additional lines of code.

(3) It enables updating parameter values during runtime. This allows students to change their controller gains mid-flight and immediately observe the influence on flight performance.

The architecture of nodes and messages we developed makes the classroom environment flexible and easy to manage for the teacher and enables students to begin directly with controller implementation. The following list explains the nodes of our software, and Fig. 3 shows the message connections between these nodes. In the typical classroom setting the first three nodes on the list are:...
launching the ROS nodes for the flying zone, Crazyflie, student,
and updates connected nodes when changes occur.

- Mocap Data Stream node: Retrieves the position and orientation data of the Crazyflies from the Mocap system and publishes this for use by the other nodes.
- Flying Agent Client node: Manages all aspects of a single Crazyflie, including: retrieving Mocap data relevant for the allocated Crazyflie, selecting which controller to use, implementing the safety mechanisms, and sending commands to the Crazyflie via the Crazyradio.
- Default and Custom Controller nodes: These nodes run the estimation and control computations.
- Radio nodes: Each node sends commands to and receives data from a single Crazyflie using the protocol specific to the Crazyradio PA USB dongle.
- Flying Agent GUI node: Allows the student to conduct experiments easily and quickly by accessing all the features of the Flying Agent Client.
- Battery Monitor node: Filters the raw battery voltage and publishes the current status for other nodes to use.

The ROS allows a range of programming languages to be used for nodes, the nodes of our software all use C++ because of its run-time speed and ubiquity.

**Qt-based Graphical User Interfaces (GUI):** The two GUIs we developed are a key part of how our software is tailored to the classroom setting. We chose the Qt (2019) framework because it supports the development of highly-customized complex GUIs, allowing many interesting future possibilities to be pursued. Moreover, the Qt Creator Interactive Development Environment (IDE) provides an intuitive tool for creating GUIs, making it possible for new developers to contribute to our software.

**Cascaded control architecture:** Common practice for quadcopter control is to use a cascaded architecture,
The teacher can just use the Teacher workflow:

Teacher workflow: The teacher can just use the System Config GUI (shaded red in Fig. 3) to allocate the {flying zone, Crazyflie, student} tuples appropriate for the class. Our software allows configurations to be saved and loaded, making this task even quicker. If necessary, the Mocap system can be recalibrated without closing and relaunching our software, causing minimal interruption to the class.

Student workflow: During class the students implement and test their controllers by interacting with the Custom Controller and Flying Agent GUI nodes (shaded blue in Fig. 3). The typical workflow for the students is:

1. Shut down all ROS nodes running on their computers.
2. Edit the C++ code of the Custom Controller node to make updates to their control algorithms.
3. Launch the respective ROS nodes on their computers.
4. Use the Flying Agent GUI to test the flight performance of their controllers, and change controller gains in real-time, see Fig. 5.

Once the students are familiar with the normal workflow, they may start changing the parameters or code of the Flying Agent Client node to complete more advanced tasks, for example, performing a flip.

Safety mechanisms: If a Crazyflie leaves its allocated flying zone or performs a high angle maneuver, then the Flying Agent Client reverts to the Default Controller. Although a crash may still occur in some cases, the low inertia of the Crazyflie provides additional protection against damage. If a low battery voltage is reached, this either triggers take-off, if it occurs during flight, or it prevents take-off, if it occurs while the Crazyflie is on the ground. This protects the battery from reaching voltage levels low enough to cause permanent damage.

Our software system is open source, with the latest stable version found at (Beuchat and Romero, 2019).

3. HANDS-ON CONTROL COURSE

The Electrical Engineering Bachelor’s programme at ETH Zürich includes courses designed to impart practical knowledge and skills, and also to encourage independent experimentation. In this context, we created the course presented in this section, which uses the quadcopter teaching platform described in Sec. 2 to provide a hands-on classroom experience for 32 undergraduate students each semester. The class aims to help the students:

1. learn and understand the specific model, dynamics, and control of quadcopters,
2. apply system and control theory from their undergraduate control courses to a real-world system, and
3. experience the complete chain of modelling, simulating, control design, implementing, and testing.

3.1 Pedagogic Aspects

The hands-on experience of applying control theory to a real system complements the experiences from conventional lectures, where the students mainly hear, read, and write. For example, students typically respond with a strong emotional jolt when a “divide by zero” error causes their quadcopter to shut-down and fall out of the sky. This provides a different appreciation for the concept compared with forgetting the “divide by zero” case in an exercise and having marks deducted accordingly.

Experiencing the complete development cycle in one project, i.e., the stages of modeling, simulation, controller development, implementation, and testing, shows the students how decisions in the early stage effect the performance later on in the experiments. Moreover, the importance of simulation and testing in control engineering becomes obvious. Similar to the example above, students experience a different emotion when seeing their quadcopter
crash compared with seeing traces diverge to infinity in a simulation environment. In addition, the fast dynamics of a quadcopter is challenging and motivating, in the sense that mistakes in the controller design are observed through unstable flight, and at the same time, achieving stable flight is rewarding.

Productive failure can also be used as a pedagogic element (Kapur, 2008). The structure of the course allows opportunities for students to make mistakes, and the safety mechanisms, described in Sec. 2.3, enable the students to observe the outcome of such mistakes multiple times without damaging the system. For example, positive feedback or high integrator gains in the altitude controller cause the quadcopter to accelerate towards the ceiling, and the Default Controller is activated just-in-time to avoid a collision. This creates an appreciation for the mistakes, and allows students to experiment with the limits and sensitivity of their controller design. The students thus learn from poor design choices and unforeseen errors, until they are rewarded not only with a working flight controller, but also with a deeper understanding of the system and the underlying control theory.

Working in teams of two, and towards the end of the course collaborating with other teams, teaches the students that complex tasks are better solved as a team. In addition, the team work makes the class more enjoyable and efficient. As this hands-on course is offered in parallel with the first control theory courses, it demonstrates the importance of control in real-world systems and can motivate students to pursue advanced control courses in their further studies.

3.2 Course content

The number of participating students depends on the size of the quadcopter platform. In our case, the room size, equipment available, and volume of the flying space enables a productive classroom environment with up to eight flying zones, Crazyflies, and student computers operating simultaneously, see Fig. 6. Thus the capacity of a hands-on session is 16 students who work in pairs, and we offer two parallel classes per semester.

The course curriculum is split over six sessions, the first three sessions focus on modeling and control through theory and simulations, and the remaining are hands-on sessions where the students implement the controllers they developed. A brief description of each session follows:

(Session 1) Is an interactive “chalk-and-talk” lecture where we derive and discuss the design and equations of motion for multirotor vehicles with an arbitrary number of propellers. By considering a multirotor vehicle we motivate discussion and questions that guide the student to understand why a symmetric quadcopter is well suited for applying basic control techniques. The modelling details are found in our course script (Beuchat, 2019a).

(Session 2) Students build the equations of motion in the Matlab/Simulink simulation environment, starting from a template that includes a stabilizing controller. In this simulation session we guide the students to observe and understand design choices for a quadcopter and related dynamic effects. For the Crazyflie 2.0 model parameters, we use those identified by Förster (2015).

(Session 3) Students continue building their simulation environment, first implementing the cascaded architecture shown in Fig. 4, and then using either a PID or LQR controller for the outer loop. To achieve this, we guide the students to linearize the equations of motion and observe the decoupling of the states and actuators, thus motivating a decoupled controller structure. The students learn about time scale separation of the positions and angles, motivating the cascaded control architecture. Using the linearized model for controller design allows students to apply the control theory studied in their undergraduate courses. Testing and tuning the controller in simulation highlights the link between the controller parameters and their effect on the closed-loop quadcopter dynamics.

(Session 4) This first hands-on session focuses on getting familiar with the experimental setup, working with the operating system, the coding framework, and conducting flight tests with the GUI. Once the students understand where the C++ file for implementing their controller sits within the system, they begin implementing and testing the real-world counterpart of Session 3.

(Session 5) The students continue implementing their controller and we guide them to observe and explain differences between their quadcopter’s behavior in simulation and experiments. For example, model mismatch of the mass, center of gravity, or inertia is influenced by the retro-reflective markers shown in Fig. 1. Changing the model parameters during experiments enhances understanding of the dynamic effects of model mismatch.

(Session 6) The students are given freedom to propose their own ideas of a control task to perform with their quadcopter, and subsequently they implement, test, and demonstrate it to their cohort. The student pairs are also free to collaborate with other pairs. Examples of control tasks performed include: trajectory tracking, flips, mimicking the trajectory of another object or quadcopter, and formation control with multiple quadcopters. Through the last two examples students discover aspects of distributed control and the required data communication. Highlights from Session 6 can be seen in this video (Beuchat, 2019b).

3.3 Student feedback

We have run the course in three consecutive semesters, receiving positive and constructive feedback each time as collected via an anonymous survey. From the instructor’s perspective, we observed the students to be engaged with all parts of the curriculum, motivated by the hands-on interaction with the quadcopter, and problem solving in a
collaborative fashion. These observations were confirmed by the survey results. To highlight how the hands-on course complements the material taught in other classes, in response to the survey statement “The course was useful for understanding the theory of the control systems 1 course”, 90% of students agreed or strongly agreed. The following two responses were received in the general comments section of the survey: “I really liked the course, and I am still very surprised on how easily we could design the control for a fairly complex system with only basic control theory tools that we had learned in 4–5 weeks of lecture!”; “It was a great course, one of the more useful and interesting ones that there are. Keep up the good work!”.

4. FURTHER USES OF THE PLATFORM

The hardware and software platform can be used in all levels of control education. Courses can also be designed for teaching advanced control concepts such as Robust Control, Model Predictive Control, or Distributed Control. Furthermore, Bachelor’s and Master’s projects can be run on the platform to combine theoretical, practical, and research considerations. The platform also serves as a testbed where advanced control concepts can be readily tested and validated. Evaluating the performance of novel control concepts and comparing it to that of existing ones in experiments, gives important insights and is very valuable for research. The streamlined software architecture described in Sec. 2 and 3 facilitates such research efforts. The platform has been used to implement distributed control and approximate dynamic programming algorithms, Romero et al. (2019), and is currently being used for the validation of learning-based control.

Demonstrations are an important part of communicating not only with other researchers, but also a general audience. During lab visits, high school students, for example, can experience the concept of feedback control in a playful way by changing parameters over the intuitive GUI shown in Fig. 5. If the Mocap system and computers are chosen accordingly the hardware platform is easy to move which, together with the robust software, makes it well suited for demonstration purposes outside the laboratory.

ACKNOWLEDGEMENTS

The authors would like to thank Angel Romero and Yvan Bossard for their contributions to developing the software (Sec. 2.2) and course (Sec. 3.2).

REFERENCES

Antsaklis, P., Basar, T., DeCarlo, R., McClamroch, N.H., Spong, M., and Yurkovich, S. (1999). Report on the NSF/CSS workshop on new directions in control engineering education. IEEE Control Systems Magazine, 19(5), 53–58.

Bencomo, S. (2004). Control learning: present and future. Annual Reviews in Control, 28(1), 115 – 136.

Beuchat, P.N. (2019a). N-rotor vehicles: modelling, control, and estimation. ETH Zürich Research Collection. doi:10.3929/ethz-b-000325583.

Beuchat, P.N. (2019b). Quad-rotor P&S: Highlights from 2018 autumn semester. URL youtube.com/watch?v=PEg-XHSXd58.

Beuchat, P.N. and Romero, A. (2019). D-FaLL-System. URL http://doi.org/10.5905/ethz-1007-180.

Bitcraze (2019). Bitcraze AB. URL bitcraze.io.

Bloom, B., Englehart, M., Furst, E., Hill, W., and Krathwohl, D. (1956). Taxonomy of educational objectives: The classification of educational goals. Longmans, NY.

Bucki, N. and Mueller, M.W. (2018). Improved quadcopter disturbance rejection using added angular momentum. In International Conference on Intelligent Robots and Systems (IROS), 4164–4170.

Campos-Macias, L., Gómez-Gutiérrez, D., Aldana-López, R., de la Guardia, R., and Parra-Vilchis, J.L. (2017). A hybrid method for online trajectory planning of mobile robots in cluttered environments. IEEE Robotics and Automation Letters, 2(2), 935–942.

Feisel, L.D. and Rosa, A.J. (2005). The role of the laboratory in undergraduate engineering education. Journal of Engineering Education, 94(1), 121–130.

 Förster, J. (2015). System identification of the Crazyflie 2.0 nano quadcopter. BSc. thesis, ETH Zürich.

Giernecki, W., Skwierczyński, M., Witwicki, W., Wroński, P., and Koziierski, P. (2017). Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. In Methods and Models in Automation and Robotics (MMAR), 37–42.

Hamer, M. and D’Andrea, R. (2018). Self-calibrating ultra-wideband network supporting multi-robot localization. IEEE Access, 6, 22292–22304.

Hönig, W., Milanes, C., Scaria, L., Phan, T., Bolas, M., and Ayanian, N. (2015). Mixed reality for robotics. In International Conference on Intelligent Robots and Systems (IROS), 5382–5387.

Kapur, M. (2008). Productive failure. Cognition and Instruction, 26(3), 379–424.

Lupashin, S., Hehn, M., Mueller, M.W., Schoellig, A.P., Sherback, M., and D’Andrea, R. (2014). A platform for aerial robotics research and demonstration: The flying machine arena. Mechatronics, 24(1), 41–54.

Merriaux, P., Dupuis, Y., Boutteau, R., Vasseur, P., and Savatier, X. (2017). A study of Vicon system positioning performance. Sensors, 17(7).

Optitrack (2019). NaturalPoint, Inc. DBA OptiTrack. URL optitrack.com.

Preiss, J.A., Hönig, W., Sukhatme, G.S., and Ayanian, N. (2017). Crazywarm: A large nano-quadcopter swarm. In International Conference on Robotics and Automation (ICRA), 3299–3304.

Prince, M. (2004). Does active learning work? a review of the research. Journal of Engineering Education, 93(3), 223–231.

Qt (2019). The Qt Company. URL qt.io.

Qualysis (2019). Qualysis AB. URL qualysys.com.

Quigley, M., Gerkey, B.P., Conley, K., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A.Y. (2009). ROS: an open-source Robot Operating System. In ICRA Workshop on Open Source Software.

Romero, A., Beuchat, P.N., Stürz, Y.R., Smith, R.S., and Lygeros, J. (2019). Nonlinear control of quadcopters via approximate dynamic programming. In European Control Conference (ECC).

Vicon (2019). Vicon Motion Systems Ltd. URL vicon.com.