Preparing Software Engineers to Develop Robot Systems

Carl Hildebrandt
The University of Virginia, USA
hildebrandt.carl@virginia.edu

Meriel von Stein
The University of Virginia, USA
meriel@virginia.edu

Trey Woodlief
The University of Virginia, USA
adw8dm@virginia.edu

Sebastian Elbaum
The University of Virginia, USA
selbaum@virginia.edu

ABSTRACT

Robotics is a rapidly expanding field that needs software engineers. Most of our undergraduates, however, are not equipped to manage the unique challenges associated with the development of software for modern robots. In this work we introduce a course we have designed and delivered to better prepare students to develop software for robot systems. The course is unique in that: it emphasizes the distinctive challenges of software development for robots paired with the software engineering techniques that may help manage those challenges, it provides many opportunities for experiential learning across the robotics and software engineering interface, and it lowers the barriers for learning how to build such systems. In this work we describe the principles and innovations of the course, its content and delivery, and finish with the lessons we have learned.

KEYWORDS

robotics, software engineering, education

ACM Reference Format:
Carl Hildebrandt, Meriel von Stein, Trey Woodlief, and Sebastian Elbaum. 2022. Preparing Software Engineers to Develop Robot Systems. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The robotics field has grown steadily for the last two decades. The number of research initiatives in robotics around the world has surged and now includes staple programs like the US DARPA Challenges [1, 3, 20], US National Robotics Initiative [12], the Together Through Innovation robotics-related program in Germany, and Japan's New Robot Strategy [17]. Such research efforts combined with an emerging market have energized the robotics industry, which is projected to grow by 25% between 2020-2025 [2]. This growth is expected to result in new jobs requiring specialized knowledge in robotics and in the software that underlies such systems.

Calls to prepare our software engineers for this robotic revolution [23] have been met primarily through specialized graduate-level courses or through massive open online courses (MOOCs). The graduate-level courses are either taught with a specialized goal, e.g., designing a car to drive around a race track [15], or paired with other specialized concepts such as AI, control theory, or mechatronics [7, 8, 13]. However, these classes overlook the fact that robotics heavily relies on software. Thus, graduates of these classes lack the software engineering (SE) principles key to designing and developing real-world robotic applications. MOOCs, such as the "Robotics Software Engineer" Udacity course [4], aim to scale up the number of students introduced to these topics, though graduation rates seem to temper that potential [19]. Furthermore, MOOCs fall short in that they aim for a breadth of applicants, meaning graduates may lack in other fundamental SE aspects. For example, the aforementioned course does not call for any computer science (CS) prerequisites.

In academia, traditional undergraduate CS curricula typically offer few opportunities to familiarize oneself with basic concepts, algorithms, and practices required for the software development of robotic systems, despite its inherent value as a means to bring in many facets of engineering [5]. Thus, for example, required CS courses typically do not cover how to represent the state of a system that includes not just the cyber elements but also the physical ones, the algorithms to manage the noise and uncertainty associated with sensors and actuators, or the architectures that rely on large middleware layers to conquer system complexity [8]. On the other hand, elective courses specializing in robotics usually focus on particular aspects of the robot system pipeline, such as embedded devices, vision and signal processing, or planning and control, without addressing the unique and foundational aspects of designing and implementing the software for such systems.

With the current landscape of CS and SE education and opportunities in robotics, we set the **goal of developing a course that would enable upper-level undergraduate students in computational disciplines to gain expertise on foundational aspects of software development for robotics**. Underlying this goal is the (yet untested) expectation that the knowledge acquired will better prepare students to develop robot systems and that the material covered will make them consider this new career path.

Achieving that goal is not straightforward for a variety of reasons. First, robotics is a multidisciplinary and rapidly expanding field, so determining the essential elements most relevant to cost-effective robot software development is challenging. Second, there is a constant tension between how to distribute the emphasis between robotics and SE topics and practices. Further, identifying faculty that feel comfortable balancing that tension is particularly complicated as there is no guideline to support such an endeavor. Third, SE and robotics can be more engaging and effectively taught when applied in practice, yet there is no available integrated platform that enables experiential learning across both areas. Fourth, typical robotics courses can require significant upfront investment in equipment, and personnel to maintain those systems once in place, making the adoption of such courses limited to institutions or

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

students with the available resources. Even then, those are difficult to transition beyond their original associated courses.

For the last two years, we have designed, delivered, and refined a course meant to achieve the stated goal while addressing the enumerated challenges. The next sections describe the principles that guided the design and delivery of the course (Table 1), the course's key innovations, its structured content and delivery strategies, and the lessons we learned in the process.

2 PRINCIPLES

The course design and delivery are guided by a set of principles inspired by our software engineering experiences building and deploying robot systems, our collaborations with peers in software engineering and robotics from academia and industry, and the gaps we identified in the curriculum. The principles listed in Table 1, and described below, have stayed consistent over the last two deliveries of the course.

P1. Prioritize the challenges of robotics that are unique from other CS systems. The field of robotics is vast, and it brings together many challenges from different disciplines. However, from a software development perspective, not all the topics and practices have the same relevance. We specifically prioritized topics and practices that we identified as unique to robotics such as those associated with sensing, perception, planning, control, and actuation in the world, with special attention to noise and uncertainty management, raised when sensors and actuators operate in the real world. We include emerging issues as well, such as increasingly relevant ethical concerns about the popularization of robotics. We do not aim to cover all algorithms and techniques at each stage of the robotic pipeline, but rather canonical ones to highlight their challenges and the general approaches.

P2. Prioritize the unique software engineering techniques and practices required by robot system development. We assume students have had either a software engineering course or an equivalent software development experience. However, this course goes beyond that baseline, emphasizing the complex notions of system state imbued in robot systems, the specialized robot architectures and design principles to deal with issues like leaky hardware abstractions and state machines, the extensive use of sophisticated APIs and component reuse, the standardization of types, and the emphasis on simulation as a means of developing and testing. The course reinforces conventional software development practices students may have already seen, from modularization to configuration management, and stretches students' understanding of their application through the labs.

P3. Provide opportunities for experiential-learning to encourage students to practice and reflect on their experiences. At the end of the course, we want students to have an understanding and appreciation for a typical pipeline of a robotic system, together with the relevant software engineering techniques to support the development of that pipeline. By delivering practical experience combined with frequent reflection, we aim for students to feel comfortable and more confident that they have the skills to approach new robotics projects in the future.

	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Prioritize the unique software engineering techniques and practices required by robot system development
P3	Provide opportunities for experiential-learning to encourage students to practice and reflect on their experience
P4	Lower adoption barriers by making the material more accessible
P5	Reinforce foundational material across both SE and robotics

Table 1: Course Principles

P4. Lower adoption barriers by making the material more accessible. Financially, logistically, and in terms of prior knowledge, robotics can be a challenging field in which to get started. We target a broad audience of undergraduate CS students for whom this may be their first introduction to robotics (although students in computer engineering and systems engineering have successfully completed this course as well). This course aims to lower the barrier to entry for robotics so that it is approachable for any student with a general CS background and access to either a laptop or school computing resources such as department computers.

P5. Reinforce foundational material across both SE and robotics. Though the subjects of SE and robotics are not often presented together, each can reinforce the other. We aim to deliver the course in a manner that promotes the reinforcement of common themes and leaves students with a solid foundation for approaching future endeavors in robotics that ties into their existing knowledge base of software engineering, which is more endemic to computer science curricula.

How these principles are fulfilled is described in Section 3 and further detail and examples are given in Sections 4 and 5.

3 DESIGN INNOVATIONS

Throughout the course development and delivery, we were guided by the principles in Table 1. When implemented into the course's design, these principles resulted in a set of innovations which we present in this section. We present these innovations along with the corresponding principle(s) in parentheses. Table 2 summarizes the relationship between principles and innovations.

I1. Cover robotics fundamentals. (P1, P2) We want students to complete this course with a broad understanding of the basic concepts of robotics and the challenges faced in their application. To do so, we started by emphasizing the differences between robots and more traditional systems, followed by development life-cycle differences. The lab sequence began by teaching students about the Robot Operating System (ROS) [18] to give them a practical framework for understanding a robotic system's architecture. Then, we introduce students to the robotics pipeline and the concepts of perception, control, planning, and localization. We present a few approaches/algorithms to concretize the idea in each area instead of giving depth in any specific area. This means, for example, that although there are whole textbooks on mobile planning algorithms, we only cover two representative reactive algorithms and three model-based algorithms.

	Innovation	P1	P2	P3	P4	P5
I1	Cover robotics fundamentals	✔	✔			
I2	Offering different levels of hardware abstraction	✔			✔	
I3	Pair SE and Robotics topics throughout the course	✔	✔			
I4	Enable students to make design and implementation decisions in labs and project			✔		✔
I5	Making use of demonstration, conversation, and checkpoints			✔		✔
I6	Use drone simulator for hands-on experience			✔	✔	✔
I7	Incrementally build concepts to minimize required background knowledge in robotics				✔	✔
I8	Incorporate flexibility into the course schedule and allow for self-paced labs			✔	✔	

Table 2: Design Innovations to implement Course Principles

I2. Offering different levels of hardware abstraction. (P1, P4) Introducing physical and hardware abstractions at various levels enables the high-level goals of the course while helping us conquer the complexity of robot systems. We began by working with physical and hardware entities that were black boxes with few or no parameters, no noise, and no uncertainty, but by the end of the course, we had revealed each of those boxes, fully enabling their challenges. We carry this process forward into lectures and labs. For each concept, we abstract away all aspects that are not immediately relevant to the current concept. As the course progresses, the hardware abstraction is refined, and more details are incorporated. This progression allows students to become comfortable with the concepts without being overwhelmed.

I3. Pair SE and Robotics topics throughout the course. (P1, P2) When introducing a robotics concept, we also introduce closely related and required SE concepts. Once the connection is made, we highlight the importance of the SE concepts as they pertain to robotics and how they must be adjusted to the unique challenges presented by robotics. This helps familiarize the topics while also highlighting key areas where students can apply their SE expertise in this domain. For example, the familiar concept of code reuse is discussed alongside modular hardware, concepts from automata are used when explaining robot state machines for control, and design specifications and unit tests are discussed when validating complex robotics systems. Throughout this effort, emphasis is put on the ROS middleware, and its array of tools to support software development. We used ROS as it has a high-level and robust API for robotics used by many in research and industry, equipped with a specialized architecture and types system.

I4. Enable students to make design and implementation decisions in labs and project. (P3, P5) We use weekly labs in which students apply the concepts learned in lectures to reinforce their understanding of course material. To encourage students' mastery of the concepts, we structured the labs with a set of objectives and starter code, and students make individual design decisions in the implementation of their solutions. The skeleton code provided in each lab and project gives a description of the input and output of different methods. Beyond that, students can use data structures and coding conventions they are most familiar with, alongside what they have learned in the course, to implement their solutions. For example, in Lab 5, students are required to implement a perception node for object identification. The students are free to decide how this is done, for example with color matching, neural networks, Haar features, edge detection, or numerous other object recognition approaches. This setup also allows students to progress through the lab at their own pace.

I5. Making use of demonstration, conversation, and checkpoints (P3, P5) We design checkpoints across lectures, labs, and the project. For example, each lecture included two checkpoints that allowed students to work in small groups to address a prompt defining a problem or an exercise that tested their understanding of a concept when applied to a different context. These prompts facilitated the reflection and reinforcement of fundamental concepts, fostered cohorts of collaborating students, and assisted the instructor in identifying aspects requiring further elaboration. Similarly, we split each of the labs into multiple checkpoints. The first checkpoint applies the learned concept in a synthetic and minimal application. The latter checkpoints then focus on integrating the concept within the larger system. We encourage students to show us their checkpoints as they progress to avoid getting stuck or sidetracked early in the assignment. We also ask follow-up questions at each stage to ensure that the student has adequately grasped the concept and is well equipped to move forward. Labs were structured around building up a drone system to be able to extend its functionality and handle various tasks throughout the semester. For example, in the lab concerning robot control, the drone is tasked with following a ship. One of the checkpoints is: *"Showcase your drone working when the ship is stationary, the ship moves in a straight line, and the ship is moving in a zigzag."* When completed, each student's lab was graded through an individual system demonstration to a teaching staff member. The course culminates with a final integrative project, and students present their solutions to the class on the last day.

I6. Use drone simulator for hands-on experience. (P3, P4, P5) Integrating physical hardware such as specific robotics platforms can require upfront costs of money and time. This can create logistical difficulty before the course has begun and can take up time throughout the semester as hardware is damaged or wears out and requires special-order items to fix or replace. We thus created a simulator described in Section 5.2, that was both extremely lightweight yet highly functional, with realistic drone kinematics and a variety of sensors including a downfacing camera, LIDAR, GPS, and pressure sensors. The simulator was then packaged inside a virtual machine (VM) so that students could run the simulator on their own machines regardless of OS. This gave students first-hand experience applying the robotic concepts taught in this class. Using a simulator in an educational setting has several advantages. First, it allows for complete control over hardware behavior, including the amount of noise and unpredictability it exhibits. Second, it allows for the "hardware" to be reliable in that it does not break down or wear out over time and behaves the same under comparable circumstances. Third, it cuts down on time spent maintaining and

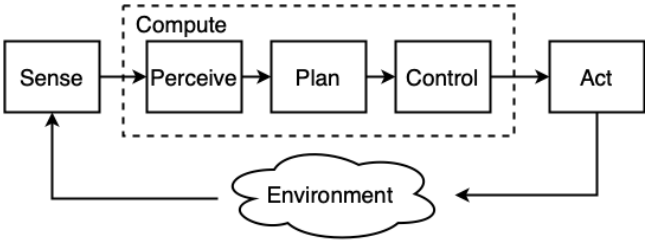

Figure 1: A robot's conceptual architecture

sourcing physical hardware to focus on the fundamental challenges that they present, such as noise and imprecision. Fourth, they are also more conducive to teaching an introduction to robotics course, as the abstractions discussed in **I2** allow for adjustment of noise, sensor accuracy, and other parameters. Finally, simulators require less time, space, and specialized knowledge to maintain, and they can be delivered entirely remotely, allowing for a successful deployment during the pandemic. Although students do not gain hands-on experience from a hardware perspective, i.e., how to physically set up, maintain, and handle a robot, simulation aims to prepare them to pick up these skills quickly in the future.

I7. Incrementally build concepts to minimize required background knowledge in robotics. (P4, P5) To ensure that the course is widely accessible to general CS students, we assume only a foundation of basic SE and knowledge of operating systems as a prerequisite. We structured the robotics concepts to build off of each other so that students could quickly get up to a working speed. We designed labs so that students could focus only on the current concept. For example, the machine state lab did not require sensor filtering, the perception lab did not require the future topics of control or navigation, and the path planning lab did not require transformations. Course structure and code reuse ensure that only small portions of new code are unfamiliar to students in each lab.

I8. Incorporate flexibility into the course schedule and allow for self-paced labs. (P3, P4) We anticipate that students will arrive in this course with a variety of prior experiences. Flexibility built into the course calendar lets students progress at their own pace. Those who need to catch up have additional time for assistance, those coming to the course with prior knowledge can sharpen their skills, and students can access the recorded lectures asynchronously. All TAs have office hours spaced throughout the week, allowing for students to come with questions on assignments or get checked off early and move forward. Since checkpoints allow for multiple attempts, students can learn from their mistakes and resubmit.

4 COURSE STRUCTURE

Developing software for robot systems is challenging as robots must sense, actuate, and represent the physical world. Sensing the physical world is usually noisy, actuating in and on the world is often inaccurate, and the knowledge and representation of the world is incomplete and uncertain. This course aims to enable students to explore and become familiar with robotic and SE approaches to cope with those challenges.

	Lecture Topic	Lab
1	Introduction	Setup and basic ROS
2	Distinguishing development features	ROS processes, communication, and simulation environments
3	Software machinery	Types and machines
4	Robot and world semantics	Sensor filtering and fusion
5	Perception	Perception through image analysis
6	From the trenches: industry speaker	Robotics and ethics
7	Controlling your robot	Controlling and testing robots
8	Making plans	(Slack Day)
9	Localization and navigation	Mapping and motion planning
11	Transformations	Transformations
12	Advanced robotics	(Holiday)
13	Project Definition	Project Development
14	Project Development	Project Presentations

Table 3: Course schedule showcasing the lab lecture pairings

4.1 Overview

The topics covered in our course are shown in Table 3. The topics' content was defined by the course scope, driven by our aim to give the students **coverage of robotics fundamentals (I1)** to allow them to make **design and implementation decisions (I4)** for a complete robot solution.

We structured the course in such a way as to minimize the amount of prerequisite knowledge required by the students (I7). This meant that we needed to start from the ground and build the concepts up. We started by introducing the students to the development features and software machinery, ending with advanced robotics.

Another unique feature of this class was the idea of reinforcing what the students learned in class through weekly labs. The topics for the labs were decided based on what was taught that week. However, they aimed to provide a more **practical approach through a simulator (I6)** while also helping students reinforce and retain the concepts learned through **demonstrations, conversations, and checkpoints (I5)**.

Table 3 shows the topics we covered and how the topics slowly expand, providing students with the knowledge to ultimately implement a complete robot architecture as shown in Figure 1. We highlight that the topics maintain breadth by introducing topics not commonly found in a computer science curriculum, such as ethics in robotics, discussions with industry leaders, and advanced concepts in robotics.

4.2 Lectures

Lectures usually began with a 2-min student presentation of a robot of their choice, generally playing a video of the robot while the student did a voice-over describing the robot, its purpose, the technical challenges it overcomes, and its societal implications. These

presentations contributed to **demonstrations and conversations (I5)** with the students. These demonstrations of current robots also served as an excellent target to refer back to during the lectures, where for example, the exploration displayed by the robot in the video could be discussed during the navigation and mapping lecture. This also served as a pleasant way to keep students engaged and make students feel more comfortable talking in front of the class.

Following the presentation, the instructor gave a lecture on the day's topic. We developed a set of 10 core lectures that **pair SE and robotics (I3)**, while also maintaining the **incremental building concepts to minimize required background knowledge (I7)** in lectures. The first 3 lectures provided fundamental building blocks describing the development challenges in robotic systems and how they differ from more traditional software applications. We start by emphasizing the richer notions of state and how robots include the cyber state, electrical-mechanical state, and a representation of the physical environment in which the robot operates (as opposed to traditional systems only considering the cyber state). In later lectures, we revisit this concept by building on how to encode such notions of state.

Next, we delve into how the development life cycle in robotics differs from pure software, in particular the points of contention and interfacing between software and hardware development. We cover a variety of topics from specifications which include richer notions of state and timeliness, to testing, which consists of the intensive use of simulation. The last foundational lecture covers architectures and design patterns for robotics, including synchronous and asynchronous mechanisms, the difference between open and closed-loop, and the extensive use of abstractions that are often leaky. We also cover the use of state machines, which are extensively used in robotics, and let us reiterate different notions of robot state. Follow-up lectures and labs revisit these topics as we use those foundational elements throughout the robotic pipeline and build more complex systems incrementally.

The rest of the lectures were aligned with fundamental elements of the robot development pipeline (I1) shown in Figure 1, from sensing and perception to motion planning and control. These topics were integrated with the SE practices that were most relevant to those components. These lectures are meant to explain the challenges in each pipeline component cleanly, describe the insights to address the challenges, and at least a couple of canonical algorithmic approaches to solve them. So, for example, in the area of sensing, we briefly cover sensor families (spending 20 minutes instead of the 1 or 2 classes as a robotics course would do) and instead spend more time on the principles to manage noise through calibration and various forms of filtering, and on the built-in data types and APIs available to support those tasks. Contrary to other robotic courses, throughout these lectures, we also attempted to de-emphasize equations and instead focus more on snippets of code to better fit the CS student body.

The lectures usually contain two exercises, each taking around 2-5 minutes to complete. These exercises were generally solved in small groups to encourage students to interact and discuss different ideas. For example, in the control lecture, students were asked to reflect on the limitations of a basic bang-bang controller to steer a bicycle and define the elements of a cruise control controller. These exercises not only facilitate student engagement but also give us insight into what elements require further explanation.

We have selected one lecture from the course to use as an example case. The following section describes how this lecture implements the innovations described in Section 3.

4.2.1 Example Lecture: Robotic Architecture and Software Machinery (Week 3). This lecture aims to introduce the fundamental concepts related to robotics architecture and modeling machinery in robotics. Robotics software is primarily asynchronous and event-driven and made up of many subsystems. These subsystems operate with different timelines and levels of certainty in a close-loop, refer to and update a complex system state, and must be managed in relation to the system's overall goals.

This lecture begins with the basic conceptual architecture of robotics (sensing-computing-acting, similar to Figure 1), incrementally incorporating richer notions of state to encode more of the system and the environment. Next, it covers critical domain-specific architectures from deliberative and monolithic to hybrid and probabilistic ones, bringing out the design tradeoffs over different scenarios applied first to an autonomous vacuum cleaner but then also discussed in the context of a modern autonomous vehicle. Architectures are followed by the coverage of system models using finite state machines (FSM). We assume that most students have seen an FSM before, so this material focuses primarily on what types of states they can encode, how they can assist in understanding the real world and decoupling behaviors, and how to scale them up to support the development of complex robotic systems.

4.3 Labs

We design the labs as a series of challenge-based learning opportunities [24] in connection with the lecture topics. We coordinate labs with lecture topics to reinforce concepts introduced in the most recent set of lectures. This sets up the **features and challenges of robotics through an SE lens (I3)** and reinforces foundational concepts. The labs, similar to the lectures, are structured with **incremental and increasing complexity (I7)**. All labs are built around the same core system: a micro-drone with realistic kinematics and dynamics, operating in a simulator further described in Section 5. First, the lab introduces a robotics feature or challenge with a short example exercise. Then, it is integrated into the drone system and paired with an SE concept. We space **checkpoints throughout a lab (I5)**, allowing students to progress at their own pace and allowing TAs to check students' work in stages such that they do not veer off track. It also gives students confidence before moving onto the next checkpoint as their work is reviewed as they **work at their own pace (I8)**. The TAs checked students for understanding of the newly introduced robotics concept and how they have used one or more SE concepts to address it throughout the lab. This includes identifying the strengths and weaknesses of their solution and what they would need to improve their solution. Points are awarded for each set of checkpoints, and students are allowed multiple attempts to complete them, either during the lab or before next lab.

As the course progresses, labs increase in complexity based on what the students have already learned. Each new lab builds on previously taught SE concepts and functionality implemented in previous labs. This helped inform the ordering of the lecture/lab

topics so that the labs could **build on each other (I7)**. For example, students were first taught the publish-subscribe architecture and node structures in ROS using a simple example before applying them to the drone simulation. As a result, SE fundamentals such as software architecture and state modeling were introduced early on and revisited in following labs. Code for the successive labs was released incrementally to increase complexity throughout the course, activating drone functionality as needed. For instance, topics such as perception and transforms in the world frame required enabling a simulated "down-facing camera" on the drone and a transformed tower positioning system respectively; these features were not included initially to allow students to grasp simpler sensors and concepts without being overwhelmed.

We have selected two labs from the course to use as example cases and highlight the progression between labs. The following sections describe how these labs implement the above innovations.

4.3.1 Example Lab: Software Machinery (Week 3). In this lab, we wanted to apply the lecture concepts of state, environment abstraction, and modeling by expanding into some of the more complex topics in ROS. Although we tried to instill good SE practices throughout, our emphasis was on **SE principles that are most useful in robotics (I3)**, from implementing finite state machines to keep track of the robot's state to enable capture and replay at the ROS level to support debugging of complex states. More specifically, the learning outcomes of this lab were:

- How to add ROS nodes to an existing system
- How to keep rich logs of messages between nodes
- How to track and implement states with an FSA
- How to use ROS type messages, parameters, and services

To incorporate these goals into the lab, we asked for students to develop a safety node for the drone. The safety node would monitor both the commands sent to the drone and the drone's position. The safety node would keep track of the drone state and only accept waypoints within the geofence to be sent to the drone when the drone was in a hovering state and thus able to accept commands (as shown in Figure 2). We also wanted to highlight **the importance of developing code that is easily parameterizable, allowing for easy reuse (I3)**. Thus the predefined area was left as a parameter that could be easily changed depending on the student's needs. This lab also required students to **keep logging information about the drone's state to debug any issues that arise (I1)**, for example, a drone accepting commands when not in the correct state. For both the modeling of the state and environment, as well as logging, the students had the **freedom to implement their own solution (I4)** within the confines of the skeleton code. This aided the teaching process, as poor design decisions such as logging only to a terminal using print statements could be pointed out and rectified **during the checkpoint discussion (I5)**.

4.3.2 Example Lab: Controlling and Testing Robots (Week 7). At this point in the class, students had been exposed to the foundational material and exercised control of the robot through waypoint commands provided through our API. That is, given a waypoint, the drone's internal controllers would move the drone accordingly. For this lab, students needed to develop their own control layer

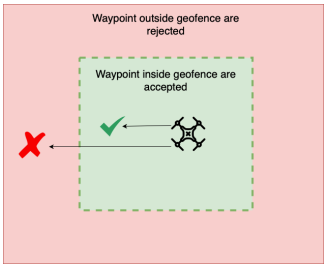

Figure 2: We showcased both safety and states using a cage system that blocks the drone from leaving a set area.

to position the drone in relation to a moving ship using various sensors. The learning objectives of the lab were as follows:

- How to implement a PID controller
- How to tune each PID term
- How to use the ROS testing framework

The lab's objective is to build a subsystem so that the drone can follow a moving ship at sea. The drone will first head towards a ship using a coarse approximation of the ship's location provided by the ship's beacon and a waypoint follower from previous labs. Then, when the ship is within the sights of the drone's down-facing camera, the drone adjusts its position using its PID controller to try to center itself on top of the ship. To support that objective, we provided a popular PID controller class [6] and activated multiple drone sensors for more precise positioning. Student needed to instantiate the PID class, fuse data from multiple sensors for positioning, tune their controllers, and test their implementation to make sure the system behaved correctly. Students could choose their PID tuning strategies as long as they perform within the bounds of provided rostests. Students were also required to design additional rostests to showcase the features of the ship-follower drone and justify why these tests were necessary. This was done to give the students more design autonomy and to think more deeply about demonstrating successful functionality in a robotics context.

This lab fulfills the design innovations of this course in several ways. First, it involves a **fundamental aspect of robotics (I1)**, the notoriously tricky tuning of a PID controller complicated by sensors operating at different rates. Its implementation as a reusable module combined with both unit and system tests demonstrates **sound SE principles (I3)**. Second, the introduction of this lab comes at a time in the course when students have already been introduced to sensors, state, and perception, giving them the fundamental tools required to **implement their own solution (I4)**. Third, the students then design and write tests to confirm that their solution works as expected. Students are also given a set of tests to self-check as they reach incremental checkpoints and have access to circulating staff during lab time and during office hours throughout the week for support allowing for **flexibility and working at their own pace (I8)**. Finally, the entire lab can be completed within the VM, including running unit tests and **using the simulator (I6)**.

4.4 Crosscutting Issues

Building robotic systems is not just a technical challenge as it has distinct social and ethical ramifications as well. We aim to provide exposure of such issues in the discussions during lectures and labs,

but we also designed a couple of specific activities to more actively tackle some of the issues.

4.4.1 Industry Perspective: Guest Speaker. We invited two industry speakers to visit us and discuss their career paths, their companies trajectories, and the challenges they face. Due to scheduling challenges, only one of them was able to come each semester. The latest visitor is the founder and CEO of a company that builds specialized self-driving vehicles. Prior to the class period, we provided students with the speaker bio and a link to the company website. We asked each student to prepare two questions for the speaker.

We asked the engineer and CEO to give a background about their company and its products, explain what they work pm day to day, and generally provide the perspective of what it is like to work in robotics. The early part of the conversation was helpful to reinforce **the need to make learning robotics more accessible (P4)** as the speaker took us through his career path into developing robot systems. Later it also emphasized the **need for SE in robotics (P5)** as he kept referring to the difficulties identifying software engineers ready to tackle the robot development challenges, and hence the opportunities for future graduates with the skills we were covering. Ultimately, the conversation helped to bring other elements that were not purely technical, such as business pressures and client demands that may not align with the technical capabilities or concerns at all.

4.4.2 Ethics Lab. While most CS curricula require specific coverage of ethics [9], these materials are generally spread around a few other courses and often not as relevant to robotics. Due to its physical nature, direct interactions with humans, and potential safety concerns, we felt that it was imperative to provide specialized coverage of ethics in the context of robots. Therefore, we dedicated a full lab to it, aiming to highlight key issues in robotics and provide students with a framework to explore ethical considerations in robotics. The learning objectives for the lab were:

- How to synthesize meaningful questions from situations that present ethical problems.
- How to find and use related work to inform these questions.
- How to connect the current state of the art to possible future scenarios and their implications.
- How to participate in debating ethical questions respectfully and productively.

We set up mock debates during the lab based on 5 prompts that highlight ethical considerations of the present day, from autonomous robots in warfare to the safety responsibilities of self-driving cars. Ahead of the lab, students were split into teams and given the side of for or against on each issue. In addition, we provided students with a few resources for ethical frameworks and prior published research in the field and encouraged them to do additional research. On the day of the lab, students turned in a written outline of their argument. During the lab, each pair of teams completed an 11-minute debate with time split between each side. The class then asked questions to each side, and the debate concluded with the rest of the class voting on a winner for the debate, with the winner being awarded extra credit. By thoroughly researching their assigned topic and interacting with each of the other debates through questions and voting, we helped students to think critically about how ethics should inform research and engineering in robotics.

4.5 Project

We designed the final project to bring together all of the concepts introduced throughout the course and have the students apply these concepts to a more complex problem. The challenge is for the drone to perform a search-and-rescue mission of a dog trapped in a disaster area. Figure 3 shows the high-level stages of a successful search-and-rescue: 1) searching the area for the dog using a map and coordinate transforms from a satellite, 2) using perception to avoid unmapped obstacles on the way and to pinpoint the dog's exact location, 3) picking the dog up by lowering the altitude of the drone directly over the dog, and 4) finding a safe place to land. However, multiple features of the environment are adversarial and work against the drone completing its mission. For example, fires pop up in the map at random times, and the dog moves about randomly when it is not under the drone.

To complete this challenge, **students had to incorporate concepts introduced throughout the semester (I7)**. They had to develop a perception component to identify the dog when it was within range of the down-facing onboard camera, handle noise when using the down-facing LIDAR to find a place to land, plan a path through the disaster area accounting for known and unknown obstacles, keep track of the mission state in order to decide what to do next, and calculate the position of the drone relative to the dog using a global transformation from a satellite connection.

We purposely left the implementation for this project open-ended. **Thus, students could design the required capabilities using any techniques or strategies they had learned throughout the course (I4)**. We provided a code skeleton to form the basis of the environment and sensors onboard the drone to unify the assessment, but the processing and action related to the environment and sensors were up to them.

Alongside the requirements put forth in the project description, **the checkpoints integrated SE concepts such as testing, system integration, state modeling, and software architecture (I3)**. A significant portion of the project involved code reuse, as much of the code in previous labs could be reused or adapted for this challenge. Students had to implement a system state model to track the drone's progress and upcoming tasks. All subsystems needed to pass the provided test cases successfully, and students had to write new tests of their own.

Students were given 3.5 weeks to complete the final project. We split the checkpoints into two stages to better accommodate **demonstrations and conversation (I5)** as well as allow students to more easily **work at their own pace in between sections (I8)**. The first stage focused on building up and testing new functionality such as path planning through the disaster site, avoiding the appearing fires, and locating the dog through perception. The second stage focused on integrating that new functionality into the system, similar to traditional engineering pipelines of complex projects.

On the final day of class, students delivered 3-minute presentations showcasing their implementation. Students had to explain their design decisions and include a video of their solution in action.

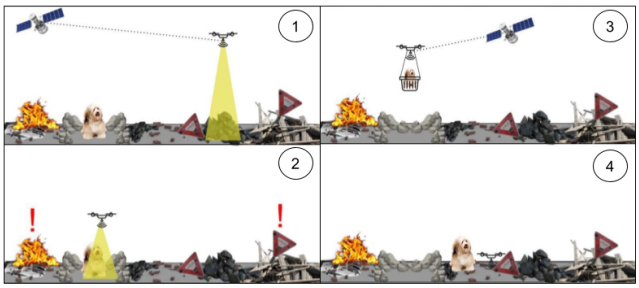

Figure 3: High-level overview of the project challenge.

Students described which parts of the project were easy and difficult to complete and had to demonstrate that their solution could successfully rescue the dog 3 times in a row to show that their solution was stable. We added a contest for the fastest completion of the mission to add a bit more excitement.

4.6 Grading Philosophy and Criteria

We use three mechanisms to simultaneously assess student progress and provide extra opportunities for feedback. First, each lab is worth between 5% and 10% of the course grade. Students that did not fully understand the assignment were given additional assistance by the teaching staff. Partial credit was given to late checkpoints to encourage students to continue working with the material.

Second, we use biweekly open-book quizzes that aim to apply or reflect on the application of a studied technique in a different context. For example, the lectures on controllers may have discussed a type of controller applied to steer a bicycle, while the quiz would request its application to an elevator while discussing when it may not work as well on such a system. Quizzes were 8% of the grade.

Third, as mentioned earlier, for every class, at least one student presented a 2-min video of their choice of a robot, explaining the most interesting techniques used, the challenges overcome by the system and which ones are pending, and what are the societal implications if successful. This presentation was only 2% of the student grade, but it helped the students to connect the systems they admire with the techniques we are covering to build them.

Last, grading for the final project operated like the labs. Again, the project relied on automated tests, demonstration, presentation, and discussion to determine the students' success at various checkpoints. The project was worth 20% of the course grade.

5 COURSE DELIVERY

The leading principle guiding the delivery of this course was **to lower the adoption barriers (P4)**. That applied to all the course materials, from the lectures which were delivered synchronously but also recorded for asynchronous delivery to students that were unable to attend lectures, for example, due to residing in different times zones, to the labs which are meant to be performed by the students on their own and with limited resources. We describe three key enablers for the course to align with this principle.

5.1 Delivery Infrastructure

The course content was delivered through three mechanisms:

(1) Lab website, which contained the syllabus, lectures, lab instructions, and general announcements.
(2) Lab code, which contained the starting versions of all code required by the students to begin implementing a lab.
(3) Lab solutions, future lab instructions, and future lectures that were only available to the teaching staff.

To improve the ease of access and management, all three of these collections of data were stored in three separate repositories hosted on Github [10]. Using GitHub had various benefits beyond the traditional configuration and control management for the team. First, it allows easy permission setting, allowing public data to be easily shared with the students. Second, GitHub provides a mechanism to automatically generate websites using GitHub pages. This meant that the hosting and publishing of the lab website was simple and straightforward. The website contained all information, including the VM setup, the lab instructions, and the lecture slides [1].

5.2 Lab Packaging

The website contained a set of instructions for each lab, and a link to the VM we built. We preconfigured the VM to include ROS, our simulator, and all libraries and packages required for the labs. Additional elements for each were accessible in the class GitHub repository. We describe the simulator and the VM in more detail later in this section.

As discussed earlier, using a physical drone for labs comes with many considerations such as space, storage, charging, maintainability, safety, student and teaching staff expertise, remote support, and the upfront cost of purchasing robots. We believe that to lower the entry barrier into robotics, systems operating in simulation offer a compelling alternative.

Still, a simulator needed to meet four requirements: 1) be computationally lightweight to operate on the students' machines, 2) contain a visual component allowing students to see how their system would behave under different software solutions 3) be functional and accurate enough to provide a realistic experience of a drone's functionality and behavior, including the sensors, physical forces, and noise inherent in these systems, and 4) abstract enough of the hardware to allow students to focus on implementing software solutions (I2, I6).

Accurately simulating the physics and kinematics of drones and environments is complex and expensive. To address the complexity, we build on an existing drone simulator, FlightGoggles[11], which has a highly realistic but expensive visualization engine. To meet the first requirement, we decoupled the expensive visualization engine, retaining only the highly accurate kinematics and physics engine. Furthermore, we pruned unnecessary components and reduced the rates of many of the sensing and computation loops.

To meet the second requirement, we created a visualization component building on a bare-bones quadrotor simulator [14], and adding capabilities such as the ability to display trajectories, obstacles, and dynamic objects such as a ground robot. By this point, we had partially met the third requirement. To meet the third requirement, we increased the simulator's functionality by incorporating a pressure sensor, a down-facing camera, a range finder, and a LIDAR.

[1]The lab website can be found at *anonymized*. Please also feel free to contact the authors of this paper for the lab solutions.

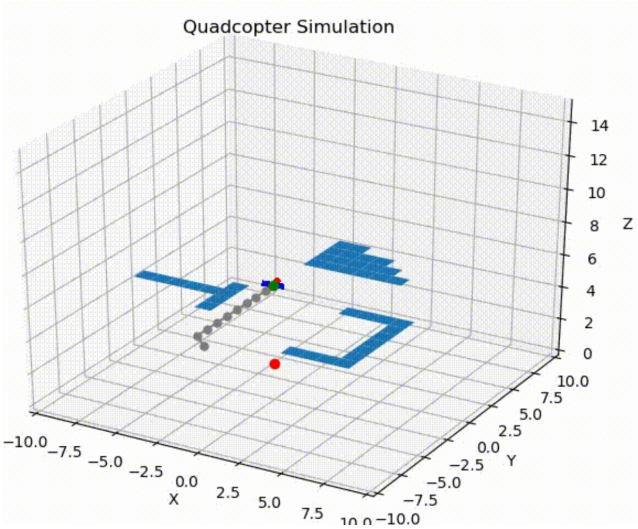

Figure 4: The final simulation showing drone navigation through a series of obstacles.

We also added features such as obstacle collision detection and external entities like a "satellite" that returned the drone's position in a different frame of reference and sensor noise controls. The last requirement was met through our **abstraction of the hardware inside the simulator (I2)**. We designed the simulator only to expose the sensor readings and actuation commands. Thus, all of the kinematics, physics, visualization, and hardware were hidden from a student's perspective, and the simulator could be treated as a black box, allowing students to focus on the software implementation of algorithms and techniques taught in the class while also allowing us to control the levels of noise and precision in the sensors.

The simulator's visualization component is shown in Figure 4 with the drone (green dot with red and blue arms), and a planned path (gray dots) through a series of obstacles (blue). The figure also shows a ground robot (red dot) used in one of the labs. Note that many of the interesting components of the simulator lie in the background, for example, where we could adjust the noise levels based on the lab, add or remove sensors on the drone when needed by the students, or adapt the simulation rate based on students' hardware. This ability to provide students with an inexpensive yet highly functional simulator was critical to this course's success.

We designed the simulator to run one of the most stable ROS versions, ROS Kinetic, on a machine running Ubuntu 18.04. Unfortunately, many students do not have access to an Ubuntu machine and run either Windows or Mac OS. This meant that we needed to come up with a solution that was OS-independent. A natural solution to this was to use a VM. We selected a freely available and OS-independent downloadable virtualization environment, VirtualBox [16] which further lowered the barrier for entry to the students. We tested the VM on systems that provided only 2GB of RAM, 2 CPU cores, and 10GB of hard drive space to the virtual machine. These requirements meant that almost all laptops or desktops sold today, even with minimum specifications, are able to install and run the virtual machine and simulation. The use of virtual machines also shifted the expertise required from setting up robotics simulation software to installing a VM. While students may not have had prior experience with VMs, VM software is well documented, and teaching staff can preconfigure most aspects of the experience by delivering the VM with the required libraries and tools.

5.3 Required Teaching Personnel

The first iteration of the course included the instructor and two part-time teaching assistants. The second iteration included three part-time teaching assistants. For both of these iterations, as the course was being developed and refined, a significant amount of time was invested in the lecture and lab design. We expect that cost to decrease as the course stabilizes.

However, another significant investment focused on supporting active learning. We wanted students to practice and reflect on what they had learned. The practical components of this lab can happen in the students' own time, with some support when they get stuck or have questions. For the reflection, however, we found that the checkpoints were critical to making sure **the students were up to date and understood the concepts through discussion (I5, I8)**. These discussions on the topics covered also meant that students had a higher chance of retaining the knowledge as they were encouraged to engage and think about what they had learned and understood while explaining their work verbally using course concepts. These interactions and frequent checks necessitated a high teacher-to-student ratio (1 to 5) with assistants with a deep understanding of the content to identify and correct any misunderstandings by the students. Investing fewer teaching resources in this class is likely to affect the effectiveness of the learning experience, but this is something we still have to assess.

5.4 Considerations due to COVID

We first designed and taught the course prior to the pandemic and then had a chance to improve it and reteach it virtually during the COVID pandemic. Along with many others working and learning from quarantine, we had to move quickly to rely heavily on virtual platforms for course delivery during the transition. We played with several platforms, but reliability, availability, and cost being the drivers, we ended up using Zoom [25] extensively (caveat: our school already had a license).

We employed Zoom to deliver the course lectures, extensively used its chat capabilities for questions and polling capabilities to keep the students engaged, and have some quick, anonymous assessments. We also took advantage of Zoom for recording the lectures and making them available to everyone asynchronously. For the labs, we also used Zoom, especially the breakout rooms capabilities, to partition the course into smaller groups where students could work on the labs in tandem. This facilitated student interactions and helped us to visit the rooms to check progress. For the labs, we also relied extensively on Slack [21] to organize requests for assistance when they came in rapid succession.

6 LESSONS LEARNED

In this section, we highlight some key takeaways from our experiences teaching this course across two iterations, reflecting on

what elements worked well and what areas need improvement or additional consideration for the next iteration.

6.1 What worked well

The **pairing of SE and robotics topics throughout the course (I3)** played a vital role in how students designed, developed, and presented their code. From a general application of SE principles, we found, for example, that students at the beginning of the course were much more likely to design their application into different classes or processes improperly. Our consistent reiteration of sound SE principles like architecture, modularization, and component reuse reduced how often we found these cases later in the course. In terms of specific pairings of SE to robotics, we were surprised by how quickly students embraced practices like testing through simulation, debugging with capture and replay, or the programming of deployment files. Connecting such practices to domain challenges was key as other SE phases like specifications that were not paired as well were not equally embraced.

Building flexibility into the course (I8) was another key to success. This was especially important due to the pandemic, with course delivery occurring remotely and a myriad of challenges external to the course. As a result, we found that flexibility in all aspects of the course was required for success. Flexibility was built into the lectures by giving students access to recordings and slides. The labs were designed with checkpoints breaking the lab up into bite-sized chunks, allowing students to work at their own pace and seek guidance and support from the teaching assistants.

The investment made for the **use of a simulator (I6)** was extremely rewarding. Although the system and simulator will likely continue to evolve, we found that even a lightweight simulator can sufficiently capture system dynamics and kinematics, as well as accurately emulate sensors, including noise and uncertainty, to provide the students with a rich experience. This was especially apparent during conversations with students during checkpoints who could quickly identify common problems faced in robotics after only experiencing a simulated drone.

The **incremental scaffolding of course materials (I7)** provided many benefits. First, we rarely perceived students being overwhelmed by the introduced material, and when they were, we could guide them to a previous lecture or lab. Second, students were often able to build on or connect to previous labs and lectures when completing a new lab, and the same applied to checkpoints. Third, while incrementally building a solid foundation, we were still able to have a final project that included the development of all software components of a robot pipeline. Fourth, from a delivery perspective, it let us manage the progress and pace.

Our **team structure and process** was also effective for designing and evolving the course. We had a team of two or three graduate students and a faculty member refining the upcoming content, taking different roles to either lead the design of the lab, its implementation, its validation, or its write-up. The rotation of roles enabled all team members to acquire substantial knowledge in teaching and quality assessment while also providing the students with labs of similar structure and quality.

Students **demonstrating and reflecting during checkpoints (I5)** to the teaching staff was also particularly effective for providing additional learning opportunities. It helped foster students' knowledge in areas they were interested in, as the teaching staff could expand on the topic and give students more depth, and it provided an early opportunity to reinforce lecture concepts and nudge the student in the right direction to prevent their misunderstanding from propagating further in the assignment.

6.2 What needs revision

There are several aspects of the course that require adjustments to address recurring challenges.

Students' machinery diversity remains a challenge to provide the same opportunities, grading criteria, and experiences to all students. Despite our efforts to provide a lightweight virtual machine, system, and simulator, we had a few students with older laptops that struggled with the more resource-intensive labs, such as the perception lab that requires multiple matrix operations per cycle to process images. One of the features we built into the lab for precisely this problem was the ability to run the simulator at a slower rate, slowing down the simulated time. However, an unforeseen yet obvious consequence of this was the additional time needed to iterate over different implementations and designs. This made it more difficult to make progress at the same rate as students with faster machines. Students with weaker machines or connectivity also struggled to run video conferencing software concurrently with the simulation, making it difficult for teaching staff to provide remote support in a timely manner. In these cases, we asked the students to either connect to our online sessions through another device or provide screenshots of the lab so they did not have to run both the simulator and video conferencing software simultaneously. Moving forward, we continue to examine other possibilities for delivering the labs and simulation to students.

The identification of **fundamental robotic topics and matching SE practices** (corresponding to **P1** and **P2**) is consolidating, but it is likely to remain in constant refinement as both fields evolve. For example, with the emergence of ROS2.0 [22] aspects like the support for real-time systems or the operation of multiple robots opens new opportunities for more exciting labs in robotics, but also requires coverage of new SE material like modeling time constraints and managing multiple deployments. We anticipate this to be an ongoing process.

The checkpointing practice, while integral to the success of the course, does come with drawbacks. We aimed to **give students design freedoms in labs and project design (I4)**; however, the more freedom a student is given to design their solution, the more time it takes to justify their choices if they do not fully understand them during checkpoints. This requires both time and expertise from teaching staff to follow what a student is trying to implement. We will continue refining lab deliverables to find the right balance in this area and may need to make some concessions in the use of checkpoints to scale the course up with the existing resources.

Another challenge is defining a set of **prerequisites** for the course that is inclusive enough to allow for a wide variety of students with different backgrounds while also ensuring that a student has the skills required to learn and succeed in this class. In the first iteration of the course we were relaxed in the enforcement of prerequisites and found that several students, particularly students

with limited software development experience, found this class too challenging. In the second iteration, we enforced the prerequisites to require at least a software engineering and an operating system course and found far fewer students struggling but also a smaller course. We are still trying to find the right balance.

Finally, and as previously mentioned, the key pending activity is the design and implementation of an instrument to empirically assess the success of this course in terms of its goal and objectives. Thus far, we have received positive feedback but lack solid assessment instruments to judge the effectiveness of the course to meet its goal and learning objectives. For example, the course evaluations were extremely positive, the instructor received multiple complimentary emails after the class was completed about the course content and delivery (which is not usually the case), and to our knowledge 10% of the class has joined the robotics industry at this time. Such feedback is encouraging and supports the need for a formal course evaluation in the next iteration.

7 CONCLUSION

This paper presents a course that emphasizes the software engineering challenges of robotic systems. We reason that courses that focus on the intersection of software engineering and robotics give students a unique understanding and set of tools to solve robotic problems in the real world. This course focuses on 5 design principles and highlights how each helps create a novel and innovative introduction to robotics through SE.

This paper describes the course structure and learning objectives to give insight into how we prepared the course and why we believe it provided benefit to the students. The course structure was designed to pair lectures and labs to help reinforce the concepts students were learning. Additionally, the paper highlights the unique social and ethical components introduced by real-world robotic systems. The paper then describes the final project, which aims to tie together all concepts learned while giving the students the freedom to design, implement, and test an end-to-end solution for the proposed problem.

Finally, the paper looks at both what worked well and what did not. We found that pairing the lectures and labs and having inherent flexibility in the course helped us to be resilient during the uncertainty of the COVID-19 pandemic and allowed us to pace the course based on where students struggled or excelled. We found that investing instructor time in configuring a custom-built simulator allowed us to capture the essence of robotic systems while removing the complex dependencies and high costs of existing simulators that made remote learning infeasible. Finally, we found that by having multiple graduate student teaching assistants to help provide hands-on checkpoints, we could maximize the learning achieved by the undergraduate students while exposing graduate students to teaching. Moreover, exposing the TAs to the realities of designing and delivering a new course helped them better understand fundamental aspects of pedagogy, as graduate students choosing whether to pursue teaching careers.

We also identified several challenges which need more consideration. First, we find that the diversity of students' machinery remains an issue, even with a simulator as lightweight as the one used. Second, we acknowledge that the identification of fundamental robotic and software engineering topics needs constant refinement, as both fields are rapidly evolving. Third, giving the students freedom of implementation and thus providing them more ability to learn and explore on their own requires more time and expertise from teaching staff which may be difficult to achieve. Finally, as this course is aimed at a wide variety of students and draws on a basic understanding of robotics and software engineering, we continue to work at striking the right balance of prerequisite courses to allow for students to succeed while reaching a wide audience.

Acknowledgments

We would like to thank the A.N. Evans for her assistance on the first iteration of the course. Additionally we would like to thank the students who took the course.

REFERENCES

[1] [n.d.]. DARPA Subterranean (SubT) Challenge. https://www.darpa.mil/program/darpa-subterranean-challenge.
[2] [n.d.]. Global Robotics Market Growth, Trends, and Forecasts Report 2020-2025: Advent Of Industry 4.0 Driving Automation & Increasing Emphasis On Safety. https://www.businesswire.com/news/home/20201216005516/en/.
[3] [n.d.]. The Grand Challenge. https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles.
[4] [n.d.]. Udacity Robotics Software Engineer. https://www.udacity.com/course/robotics-software-engineer--nd209.
[5] D.J. Ahlgren. 2002. Meeting educational objectives and outcomes through robotics education. In *Proceedings of the 5th Biannual World Automation Congress*, Vol. 14. 395–404. https://doi.org/10.1109/WAC.2002.1049471
[6] Kiam Heong Ang, Gregory Chong, and Yun Li. 2005. PID control system analysis, design, and technology. *IEEE transactions on control systems technology* 13, 4 (2005), 559–576.
[7] Grzegorz Cielniak, Nicola Bellotto, and Tom Duckett. 2013. Integrating Mobile Robotics and Vision With Undergraduate Computer Science. *IEEE Transactions on Education* 56, 1 (2013), 48–53. https://doi.org/10.1109/TE.2012.2213822
[8] Joel M. Esposito. 2017. The State of Robotics Education: Proposed Goals for Positively Transforming Robotics Education at Postsecondary Institutions. *IEEE Robotics Automation Magazine* 24, 3 (2017), 157–164. https://doi.org/10.1109/MRA.2016.2636375
[9] CC2020 Task Force. 2020. *Computing Curricula 2020: Paradigms for Global Computing Education.* Association for Computing Machinery, New York, NY, USA.
[10] github. 2020. GitHub. https://github.com/
[11] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. 2019. FlightGoggles: Photorealistic Sensor Simulation for Perception-driven Robotics using Photogrammetry and Virtual Reality. arXiv:arXiv:1905.11377
[12] Daniel J Hicks and Reid Simmons. 2019. The national robotics initiative: a five-year retrospective. *IEEE Robotics & Automation Magazine* 26, 3 (2019), 70–77.
[13] Seul Jung. 2013. Experiences in Developing an Experimental Robotics Course Program for Undergraduate Education. *IEEE Transactions on Education* 56, 1 (2013), 129–136. https://doi.org/10.1109/TE.2012.2213601
[14] Abhijit Majumdar. 2013. Quadcopter simulator. https://github.com/abhijitmajumdar/Quadcopter_simulator.
[15] Matthew O'Kelly, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Dipshil Agarwal, Madhur Behl, Paolo Burgio, et al. 2019. F1/10: An open-source autonomous cyber-physical platform. *arXiv preprint arXiv:1901.08567* (2019).
[16] Oracle. 2020. VirtualBox Virtual Machine v6.1.16. https://www.virtualbox.org/.
[17] IFR Pressroom. 2021. Investment in Robotics Research – Global Report 2021. https://ifr.org/ifr-press-releases/news/investment-in-robotics-research-global-report-2021. (27 May 2021).
[18] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, Vol. 3. Kobe, Japan, 5.
[19] Justin Reich and José A Ruipérez-Valiente. 2019. The MOOC pivot. *Science* 363, 6423 (2019), 130–131.
[20] Guna Seetharaman, Arun Lakhotia, and Erik Philip Blasch. 2006. Unmanned vehicles come of age: The DARPA grand challenge. *Computer* 39, 12 (2006), 26–29.
[21] Slack. 2020. Slack. https://slack.com/
[22] Dirk Thomas, William Woodall, and Esteve Fernandez. 2014. Next-generation ROS: Building on DDS. In *ROSCon Chicago 2014.* Open Robotics, Mountain View, CA. https://doi.org/10.36288/ROSCon2014-900183

[23] David S Touretzky. 2010. Preparing computer science students for the robotics revolution. *Commun. ACM* 53, 8 (2010), 27–29.
[24] Scooter Willis, Greg Byrd, and Brian David Johnson. 2017. Challenge-Based Learning. *Computer* 50, 7 (2017), 13–16. https://doi.org/10.1109/MC.2017.216
[25] Zoom. 2020. Zoom. https://zoom.com/