Preparing Software Engineers to Develop Robot Systems

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

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

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

Sebastian Elbaum
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: 1) it emphasizes the distinctive challenges of software development for robots paired with the software engineering techniques that may help manage those challenges, 2) it provides many opportunities for experiential learning across the robotics and software engineering interface, and 3) it lowers the barriers for learning how to build such systems. We describe the principles and innovations of the course, its content and delivery, and finish with the lessons we have learned.

CCS CONCEPTS

• Computer systems organization → Robotics; **• Applied computing → Education**.

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 *44th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '22), May 21–29, 2022, Pittsburgh, PA, USA*. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3510456.3514161

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, 29], US National Robotics Initiative [16], the Together Through Innovation robotics-related program in Germany, and Japan's New Robot Strategy [24]. 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 [32] have been met primarily through specialized graduate-level courses or through massive open online courses (MOOCs). The graduate-level courses are either focused on particular aspects of robotics such as AI, control theory, or mechatronics [9, 12, 17], or have broader topical coverage but on specific domain platforms [22]. However, these classes overlook the fact that robotics heavily relies on software and the development process to generate that software. Thus, graduates of these classes lack the key software engineering (SE) principles 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 [26]. 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.

At the undergraduate level, traditional CS curricula typically offer few opportunities to familiarize oneself with basic concepts, algorithms, and practices required for the software development of robotic systems, despite the inherent value of these systems as a means to connect many facets of engineering [6]. 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 [12]. On the other hand, elective undergraduate courses specializing in robotics usually focus on particular aspects of the robot system pipeline. For example, courses on embedded systems [10, 18], unique issues arising from discrete and continuous behaviors [8, 19], vision and signal processing [21, 33], or planning and control [5, 11, 28] examine that specialty 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 of 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

Table 1: Course design and delivery guiding principles.

	Principle
P1	Prioritize the challenges of robotics that are unique from other CS systems
P2	Focus on 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

that enables experiential learning across both areas. Fourth, typical robotics courses can require significant upfront investment in equipment, as well as personnel to maintain those systems once in place. This limits the adoption of such courses only to institutions or students with the available resources. Even then, those systems 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, 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 of 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 as robots are integrated in our lives. 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 to develop software for them.

P2. Focus on 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 tool for development 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.

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, leaves students with a solid foundation for approaching future endeavors in robotics, and that ties into their existing knowledge base of software engineering, which is more endemic to computer science curricula.

We describe how the course fulfills these principles in Section 3 with further detail and examples in Sections 4 and 5.

3 DESIGN AND PEDAGOGIC INNOVATIONS

Throughout the course development and delivery, we were guided by the principles in Table 1. When incorporated into the course's design, these principles resulted in a set of pedagogic innovations that we now present 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 start by emphasizing in the lectures the differences between robots and more traditional systems, including development life-cycle differences. The lab sequence that parallels the lectures begins by exploring the Robot Operating System (ROS) [25] to give students a practical framework for understanding a robotic system's

Table 2: Design innovations to implement course principles.

	Innovation	P1	P2	P3	P4	P5
I1	Cover robotics fundamentals	✔	✔			
I2	Offer different levels of abstraction	✔			✔	
I3	Pair SE and Robotics topics throughout the course	✔	✔			
I4	Enable students to make design and implementation decisions in labs and project			✔		✔
I5	Make use of demonstration, conversation, and checkpoints			✔		✔
I6	Use a 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			✔	✔	

key architectural components. Then, we introduce students to the robotics pipeline including the concepts of perception, planning, control, and localization, all using ROS. We present key approaches and algorithms to concretize each phase. This means, for example, that although there are whole textbooks on mobile planning algorithms, we only cover a handful of representative reactive and model-based algorithms to illustrate the range of possibilities.

I2. Offer different levels of abstraction. (P1, P4) Introducing physical, hardware, and software abstractions enables us to overcome the complexity of robot systems and present the challenges incrementally. For each new covered concept, we abstract away all aspects that are not immediately relevant. Then, as the course progresses, the abstractions are refined, and more details are incorporated. This progression allows students to become comfortable with the concepts without being overwhelmed. For example, the simulator includes a waypoint controller in the introductory labs, allowing the students to control the system by sending it simple target waypoints in an empty world. As we expose the students to physical concepts like roll, pitch, yaw, and thrust, and control algorithms they are able to create their own sophisticated controller. In subsequent lectures as we introduce more complex worlds with walls, they need to develop planning components that provide high-level guidance to their controller. We carry this refinement process forward into lectures and labs.

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 introduced by this domain. This helps to connect challenges and solutions, 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 robotics components. Throughout this effort, emphasis is put on the ROS framework, and its array of tools to support software development. We use ROS because it is commonly used in research and industry, contains a high-level and robust API for robotics including standardized message types, adheres to a specialized architecture, and comes with sophisticated tool support for building and executing these systems.

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 structure the labs with a set of objectives and starter code, and students make individual design decisions in the implementation of their solutions. We provide skeleton code in each lab and in the final project that gives a description of the input and output of different functions. 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, whether is using color matching, neural networks, edge detection, or numerous other object recognition approaches and libraries. This setup allows students to progress through the lab at their own pace, while also having the freedom to further explore different pieces of the labs to greater depth.

I5. Make use of demonstration, conversation, and checkpoints (P3, P5) We design checkpoints across lectures, labs, and the project. For example, each lecture includes two checkpoints that allow students to work in small groups to address a prompt defining a problem or an exercise that tests their understanding of a concept when applied to a different context. These prompts facilitate the reflection and reinforcement of fundamental concepts, foster cohorts of collaborating students, and assist the instructor in identifying aspects requiring further elaboration. Similarly, we split each of the labs into multiple checkpoints. We structure labs around building up a drone system to extend its functionality and handle various tasks throughout the semester. 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. 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, moving in a straight line, and moving in a zigzag."* Upon completion, each student's lab is 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 a drone simulator for hands-on experience. (P3, P4, P5) Integrating physical hardware such as specific robotics platforms incurs upfront costs of money and time. This creates logistical difficulties even 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 is extremely lightweight yet highly functional, with realistic drone kinematics and a variety of sensors including a downfacing camera, LIDAR, GPS, and pressure sensors. We distribute the simulator by packaging it inside a virtual machine (VM) so that students can run the simulator on their own machines regardless of OS. We find that using such a simulator in an educational setting has several advantages: 1) it allows for complete control over hardware behavior, including the amount of noise and unpredictability it exhibits, 2) 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, 3) it cuts down on time spent maintaining and sourcing physical hardware, instead allowing for focus on the fundamental challenges that they present, such as noise and imprecision, 4) simulators are 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, and 5) simulators require less time, space, and specialized knowledge to maintain, and they can be delivered entirely remotely, allowing for a successful deployment during the COVID-19 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 structure the robotics concepts to build off of each other so that students need only focus on the current concept while building on previously covered ones. This helps students to piece together significant parts of the system pipeline as the course progresses. For example, understanding the simulation environment and ROS communication is a prerequisite to implement the different robot states, knowing the robot state aids in developing accurate sensor filtering and fusion models, and having filtered data is key to to interpret the environment in which the robot operates.

Note that the different levels of abstractions introduced in **I2** are necessary but not sufficient to support this innovation. The course structure and code reuse are key enablers to limit the portions of new material and code that is unfamiliar to the students in each lab. This idea guided the development of the skeleton code we provide to the students; for each lab, we maximize code reuse, allowing students to reapply their understanding of previous code modules to future labs. This allows students to focus on the new material and experiment with the new concepts.

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 while those coming to the course with prior knowledge can sharpen their skills. Additionally, we provide students access to recorded lectures asynchronously, allowing them to review the material on their own time and at their own pace. 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 the feedback they received and resubmit the lab. We note that this built-in flexibility was an asset when teaching during the COVID-19 pandemic.

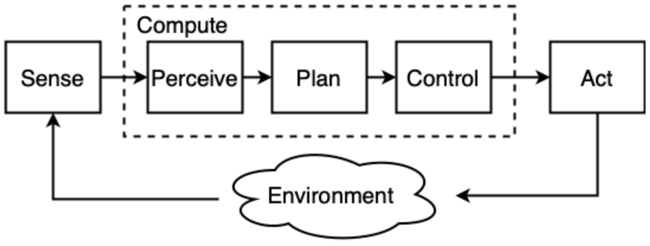

Figure 1: A robot's conceptual architecture.

4 COURSE STRUCTURE

Developing software for robot systems is challenging as robots must sense, actuate in, 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. This section describes how we structured the course while keeping in mind both the design principles and innovations presented in previous sections.

4.1 Schedule Overview

The topics we cover in the course are shown in Table 3. The topics' content are 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 structure the course in such a way as to **minimize the amount of prerequisite knowledge required by the students (I7)**. This requires that we start from the ground and **build up the concepts through different levels of abstractions (I2)**. We start by introducing the students to the development features and software machinery and conclude with advanced robotics topics. Another unique feature of this class is the idea of reinforcing what the students learned in class through weekly labs. The topics for the labs reflect the lecture material for that week. However, they aim 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 also shows how the topics incrementally expand in scope and reach, providing students with the knowledge to ultimately implement a complete robot pipeline as shown in Figure 1. Note that we introduce lectures and labs rarely seen in CS programs, from the integration of robotics and software engineering principles in the same lectures and labs, to ethics in robotics labs and discussions with industry leaders.

Table 3: Course schedule showcasing the lab lecture pairings.

	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	(Free 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

4.2 Lectures

Lectures usually begin with a 2-minute student presentation of a robot of their choice, generally playing a video of the robot while the student does a voice-over describing the robot, its purpose, the technical challenges it overcomes, and its societal implications. These presentations contribute to **demonstrations and conversations (I5)** with the students. These demonstrations of current robots also serve as an excellent target to refer back to during the lectures, where for example, the exploration displayed by the robot in the video can be discussed during the navigation and mapping lecture. This also serves 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 gives 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 of concepts to minimize required background knowledge (I7)** in lectures. The first 3 lectures provide 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 are 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 are integrated with the SE practices that are 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 are generally solved in small groups to encourage students to interact and discuss different ideas. For example, in the control lecture, students are asked to sketch an algorithm for a basic bang-bang controller to steer a bicycle and reflect on the limitations of such controller, and then to 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 closed 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 (FSMs). 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, how they are represented in code, and how to scale them up to support the development of complex robotic systems.

This lecture fulfills several of our design innovations. First, as one of the first lectures focusing on technical details of the robotics pipeline, it **introduces the students to the fundamentals of robotics architectures (I1)** which **lays the foundations for future lectures delving into different aspects of robotics pipeline (I7)**. Second, by relating these concepts to simple robotics systems the students are familiar with, i.e. the autonomous vacuum cleaner, the lecture helps **foster a conversation with the students to internalize the concepts (I5)**. Finally, we **pair the concepts of system models and FSMs (I3)**, allowing students to transition their prior knowledge toward application in robotics.

4.3 Labs

We design the labs as a series of challenge-based learning opportunities [34] 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 the lab (I5)**, allowing TAs to check students' work in stages such that they do not veer off track. This 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 during office hours before the 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 are 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 are introduced early on and revisited in following labs. Code for the successive labs are released incrementally to increase complexity throughout the course, incrementally activating drone functionality as needed **(I2)**. 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 are 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).*

This lab applies the concepts of state representation, environment abstraction, and modeling covered during the lectures while expanding into some of the more complex topics in ROS. Although we try to instill good SE practices throughout the course, this lab's emphasis is on **SE principles that are most useful in robotics (I3)**, from implementing finite state machines to keep track of the robot's state to enabling capture and replay at the ROS level to support debugging of complex states. More specifically, the learning outcomes of this lab are:

- 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 FSM
- How to use ROS type messages, parameters, and services

To incorporate these goals into the lab, we ask students to develop a safety node for the drone. The correctly-implemented safety node monitors both the commands sent to the drone and the drone's position. The safety node keeps track of the drone state and only accepts waypoints within the geofence to be sent to the drone when the drone is in a hovering state and thus able to accept commands (as shown in Figure 2). This lab also highlights **the importance of developing code that is easily parameterizable, allowing for easy reuse (I3)**. Accordingly, the predefined area is left as a parameter that can be easily changed depending on the student's needs. This lab also requires 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 are given the **freedom to implement their own solution (I4)** within the confines of the skeleton code. This aids the teaching process, as poor design decisions such as logging only to a terminal using print statements can 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 have 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 need to develop their own control layer to position the drone in relation to a moving ship using various sensors. The learning objectives of the lab ar 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

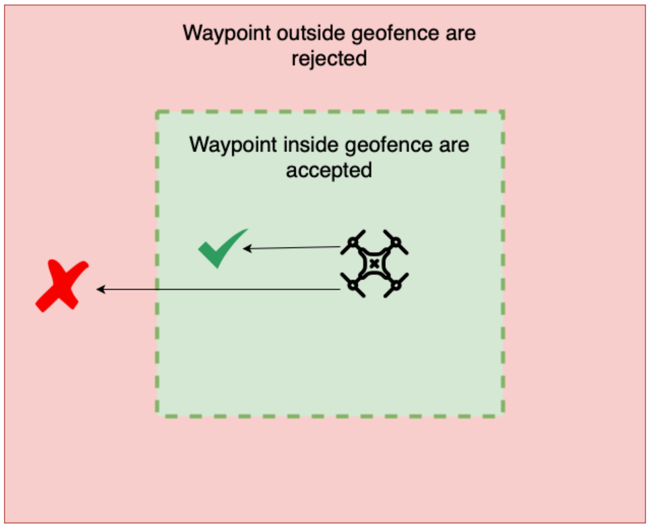
Waypoint outside geofence are rejected
Waypoint inside geofence are accepted

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

camera, the drone adjusts its position using its PID controller to try to center itself on top of the ship. To support this objective, we provide a popular PID controller class [7] and activate multiple drone sensors for more precise positioning. The student then needs to instantiate the PID class, fuse data from multiple sensors for positioning, tune the controllers, and test the implementation to make sure the system behaves correctly. Students can choose their own PID tuning strategies as long as they perform within the bounds of provided rostests [27]. Students are also required to design additional rostests to showcase the features of the ship-follower drone and justify why these tests are necessary and suitable for the problem definition. This is done to give the students more design autonomy and to encourage them to think more deeply about demonstrating successful functionality in a robotics context.

This lab fulfills the design innovations of this course in several ways through its design and delivery. First, it involves a **fundamental aspect of robotics (I1)**, the notoriously tricky tuning of a PID controller complicated by sensors operating at different rates. Second, its implementation as a reusable module combined with both unit and system tests demonstrates **sound SE principles (I3)**. Third, 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)**. To show that they have completed the lab challenges, 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)**. 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 design a couple of specific activities to more actively tackle some of these issues.

4.4.1 Industry Perspective: Guest Speaker. In each of the two prior iterations of the course, we invited two industry speakers to visit us and discuss their career paths, the trajectories of their companies, 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 speakers to give a background about their company and its products, explain what they work on day to day, and generally provide the perspective of what it is like to work in robotics. The early parts of the conversations were helpful to reinforce **the need to make learning robotics more accessible (P4)** as the speakers took us through their career path into developing robot systems. They also emphasized the **need for SE in robotics (P5)** as they 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 conversations 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 [13], these materials are generally spread around a few other courses and often not as relevant to robotics. Due to its physical nature, interactions with humans, and potential safety concerns, we feel that it is imperative to provide specialized coverage of ethics in the context of robots. Therefore, we dedicate a full lab to ethics, 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 are:

- 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 are split into teams and given the side of for or against on each issue. In addition, we provide 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 turn in a written outline of their argument. During the lab, each pair of teams complete an 11-minute debate with time split between each side. The class then asks questions to each side, and the debate concludes 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 help students to think critically about how ethics should inform research and engineering in robotics.

4.5 Project

The final project brings together all of the concepts introduced throughout the course and lets the students apply these concepts to a more complex problem. The project 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, areas of the map may randomly experience a fire the drone must avoid, and the dog moves about randomly when it is not under the drone.

To complete this challenge, **students must incorporate concepts introduced throughout the semester (I7)**. They must develop a perception component to identify the dog when it is 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 purposefully give the students freedom by leaving the implementation for the project open-ended. **Thus, students can design the required capabilities using any techniques or strategies they have learned throughout the course (I4)**. We provide 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 are up to the student.

Alongside the requirements put forth in the project description, **the checkpoints integrate SE concepts such as code reuse, state modeling, and testing (I3)**. A significant portion of the project involves code reuse, as much of the code in previous labs can be reused or adapted for this challenge. Additionally, students are required to implement a system state model to track the drone's progress and upcoming tasks. Finally, all subsystems need to pass the provided test cases successfully, and students must write new tests of their own.

Students are 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 focuses on building up and testing new functionality such as path planning through the disaster site, avoiding the fires, and locating the dog through perception. The second stage focuses on integrating that new functionality into the system, similar to traditional engineering pipelines of complex projects.

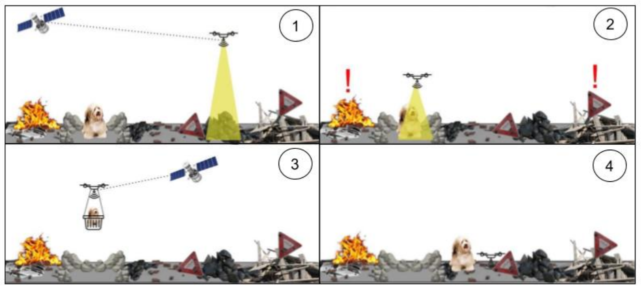

Figure 3: High-level overview of the project challenge.

On the final day of class, each student delivers a 3-minute presentation showcasing their implementation. Students must explain their design decisions and include a video of their solution in action. Students describe which parts of the project were easy and difficult to complete and demonstrate that their solution can successfully rescue the dog 3 times in a row to show that their solution is stable. We include a contest for the fastest completion of the mission to add a bit more excitement.

4.6 Grading Philosophy and Criteria

We use four 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 do not fully understand the assignment are given additional assistance by the teaching staff. We give partial credit 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 discuss a type of controller applied to steer a bicycle, while the quiz requests its application to an elevator while discussing when it may not work as well on such a system. Quizzes are 8% of the grade.

Third, as mentioned earlier, for every class, at least one student presents a 2-minute 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 is only 2% of the student's grade, but it helps the students to connect the systems they admire with the techniques we are covering to build them.

Last, grading for the final project operates like the labs. Again, the project relies on automated tests, demonstration, presentation, and discussion to determine the students' success at various checkpoints. The project is 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 [14]. 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 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 equipment. We believe that to lower the entry barrier into robotics, systems operating in simulation offers 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 their software solutions (I2, I6)**.

Accurately simulating the physics and kinematics of drones and environments is complex and expensive. To address the complexity, we built on an existing drone simulator, FlightGoggles[15], 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 [20], 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 fully meet the third

[1]The lab website can be found at https://less-lab-uva.github.io/CS4501-Website/. Please also feel free to contact the authors of this paper for the lab solutions.

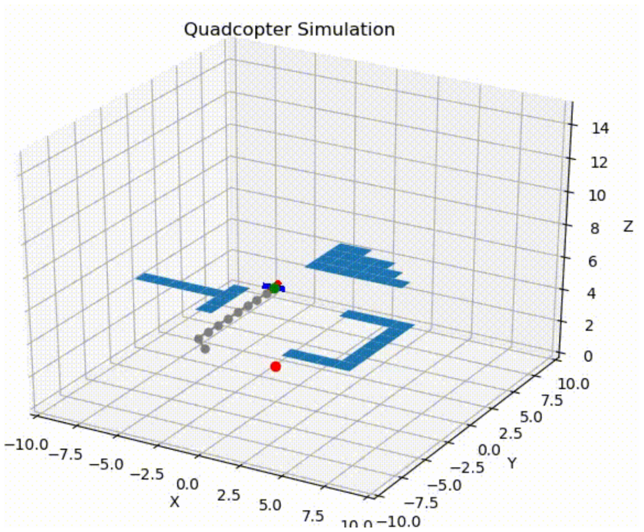

Figure 4: The final simulation showing drone navigation through a series of obstacles.

requirement, we increased the simulator's functionality by incorporating a pressure sensor, a down-facing camera, a range finder, and a LIDAR. 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 design 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 [23] 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 teaching assistants that have 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 impact the effectiveness of the learning experience, but this is something we still have to assess.

5.4 Going Virtual due to COVID-19

We first designed and began teaching the course immediately prior to the COVID-19 pandemic, with the first iteration of the course transitioning to virtual learning mid way through the semester. Along with many others working and learning from home, 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 [35] extensively (caveat: our school already had a license).

We applied what we learned from this experience to improve and reteach the course fully virtually the following year during our second iteration. We employed Zoom to deliver the course lectures. We extensively used its chat capabilities for questions and its 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 the teaching staff to visit the rooms to check progress. For the labs, we also relied extensively on Slack [30] 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.

Both **the use of different levels of abstraction (I2)** and **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.

The diversity of students' machinery presents an continual 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 in retrospect was the additional time needed by those students with a slower simulator 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 (P1, P2)** is consolidating, but it is likely to remain in constant refinement as both fields evolve. For example, the emergence of ROS2.0 [31] and 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 an impressive 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

Through this paper we have introduced a course aiming at equipping students with a unique understanding of the challenges in developing the software underlying robotic systems and a set of tools to address those challenges. We have shared the guiding principles of the course, highlighted the most important pedagogical innovations, described its structure and delivery, and analyzed what aspects worked and which ones needs revision.

We are planning several changes to the course content and delivery in the future. First, we will refine the integration of software engineering and robotic materials, such as through the addition of formal specifications as part of the planning lectures. We will also iterate on key labs, such as the control lab, where the abstractions did not offer enough flexibility to accommodate different controller implementation and manipulate noise accurately. Second, we aim to further assess the course in terms of achieving its learning objectives. Third, we are working with other institutions to facilitate their adoption of the course materials so that they may create different paths through the course to fit diverse audiences.

ACKNOWLEDGMENTS

We would like to thank the A.N. Evans for her assistance on the first iteration of the course. We would also like to thank the students in both instances of the course for their enthusiasm, patience, and support. Last, the authors are thankful to the NSF for funding in part this work through grants #1924777 and #1909414, and to the AFOSR through grant FA9550-21-1-0164. Trey Woodlief was supported by a University of Virginia SEAS Fellowship.

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] Abhijeet Agnihotri, Matthew O'Kelly, Rahul Mangharam, and Houssam Abbas. 2020. Teaching autonomous systems at 1/10th-scale: Design of the f1/10 racecar, simulators and curriculum. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 657–663.

[6] 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

[7] 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.

[8] Kerstin Bauer and Klaus Schneider. 2012. Teaching cyber-physical systems: A programming approach. In *Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education*. 1–8.

[9] 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

[10] Tanya Lee Ann Crenshaw. 2012. Using robots and contract learning to teach cyber-physical systems to undergraduates. *IEEE Transactions on Education* 56, 1 (2012), 116–120.

[11] Zoe Doulgeri and Tilemachos Matiakis. 2006. A web telerobotic system to teach industrial robot path planning and control. *IEEE Transactions on education* 49, 2 (2006), 263–270.

[12] 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

[13] CC2020 Task Force. 2020. *Computing Curricula 2020: Paradigms for Global Computing Education*. Association for Computing Machinery, New York, NY, USA.

[14] github. 2020. GitHub. https://github.com/

[15] 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

[16] Daniel J Hicks and Reid Simmons. 2019. The national robotics initiative: a five-year retrospective. *IEEE Robotics & Automation Magazine* 26, 3 (2019), 70–77.

[17] 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

[18] Edward Ashford Lee and Sanjit Arunkumar Seshia. 2016. *Introduction to embedded systems: A cyber-physical systems approach*. Mit Press.

[19] Jan Lunze and Françoise Lamnabhi-Lagarrigue. 2009. *Handbook of hybrid systems control: theory, tools, applications*. Cambridge University Press.

[20] Abhijit Majumdar. 2013. Quadcopter simulator. https://github.com/abhijitmajumdar/Quadcopter_simulator.

[21] Julien Marot and Salah Bourennane. 2017. Raspberry Pi for image processing education. In *2017 25th European Signal Processing Conference (EUSIPCO)*. IEEE, 2364–2366.

[22] 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).

[23] Oracle. 2020. VirtualBox Virtual Machine v6.1.16. https://www.virtualbox.org/.

[24] 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).

[25] 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.

[26] Justin Reich and José A Ruipérez-Valiente. 2019. The MOOC pivot. *Science* 363, 6423 (2019), 130–131.

[27] ROS.org. 2019. Slack. http://wiki.ros.org/rostest

[28] Pedro Sanz. 2009. Robotics: Modeling, planning, and control (siciliano, b. et al; 2009)[on the shelf]. *IEEE Robotics & Automation Magazine* 16, 4 (2009), 101–101.

[29] Guna Seetharaman, Arun Lakhotia, and Erik Philip Blasch. 2006. Unmanned vehicles come of age: The DARPA grand challenge. *Computer* 39, 12 (2006), 26–29.

[30] Slack. 2020. Slack. https://slack.com/

[31] 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

[32] David S Touretzky. 2010. Preparing computer science students for the robotics revolution. *Commun. ACM* 53, 8 (2010), 27–29.

[33] Aurelio Uncini. 2015. *Fundamentals of adaptive signal processing*. Springer.

[34] 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

[35] Zoom. 2020. Zoom. https://zoom.com/