Autonomous Agents: An Advanced Course on AI Integration and Deployment

Stephanie Rosenthal and Reid Simmons

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
{srosenth,reids}@cs.cmu.edu

Abstract

A majority of the courses on autonomous systems focus on robotics, despite the growing use of autonomous agents in a wide spectrum of applications, from smart homes to intelligent traffic control. Our goal in designing a new senior-level undergraduate course is to teach the integration of a variety of AI techniques in uncertain environments, without the dependence on topics such as robotic control and localization. We chose the application of an autonomous greenhouse to frame our discussions and our student projects because of the greenhouse's self-contained nature and objective metrics for successfully growing plants. We detail our curriculum design, including lecture topics and assignments, and our iterative process for updating the course over the last four years. Finally, we present some student feedback about the course and opportunities for future improvement.

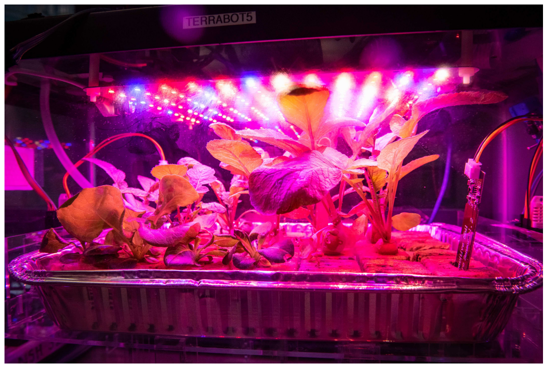


Figure 1: Students create AI agents to autonomously grow plants in a fish tank-sized greenhouse, called the TerraBot.

Introduction

Autonomous systems are becoming increasingly prevalent in today's society – from delivery and cleaning robots, to Mars rovers, to intelligent traffic control, to smart homes – and our students are increasingly interested in working on these autonomous systems when they graduate. We wanted to offer a senior-level undergraduate course that would present some of the advanced AI techniques used in integrated autonomous systems, focusing on the topics of perception, decision making, action, and learning. Our vision was to develop a project-oriented course that enabled students to understand how environmental uncertainty and sensor noise complicates the goal of creating a completely autonomous agent, capable of running for weeks without human intervention.

In early 2019, we began exploring other courses offered on autonomy and the topics that we were interested in. Most were graduate-level courses that focused on autonomous mobile robots or self-driving car applications (Institut Polytechnique de Paris 2020; Columbia University 2020; University of Edinburgh 2018; University of Texas 2015; University of Memphis 2010; University of Central Florida 2007). These courses tended to include subjects such as kinematics and dynamics, path planning, and localization. While those topics are all useful and interesting, we wanted a course geared more towards AI than Robotics, where the agent operates in the physical world but is not a mobile robot. We investigated several other options for projects that we could adapt, including a smart home, traffic control, and network security (University of Texas Austin 2022; Cal State Los Angeles 2020). While each of those applications is important, we felt that the infrastructure needed would be prohibitive for us, especially for multiple teams to work concurrently.

When we were introduced to the MIT Personal Food Computer project (Castelló Ferrer et al. 2019), we decided that an autonomous greenhouse had all the characteristics we were looking for – interaction with the real world, with plenty of uncertainty in the environment, sensors, and actuators; quantifiable metrics as to how well the agent performs (e.g., did the plants thrive and how big did they get); the greenhouse units were standalone and easily replicated; plants could reach near maturity in the 2-3 week grow periods that we had allotted. However, while the Personal Grow Computer itself has many of the aspects we desired, it was not exactly suited to our needs. Thus, in the summer of 2019, we hired three undergraduates to design and implement prototype greenhouse hardware, which we named the TerraBot (Figure 1). The TerraBot is completely standalone – it has on-board computers along with sensors and actuators that enable a software agent to monitor and control light level, soil moisture, temperature, and humidity. More details on the TerraBot are provided later in the paper.

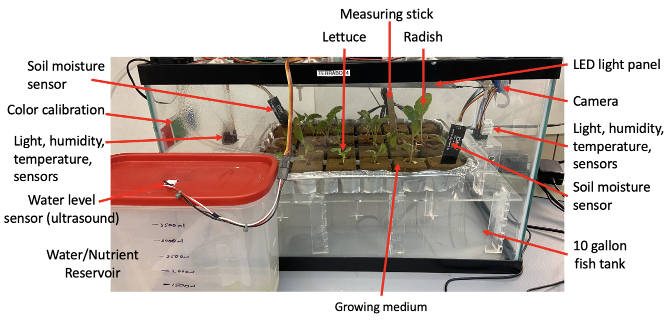


Figure 2: The TerraBot's sensors and actuators.

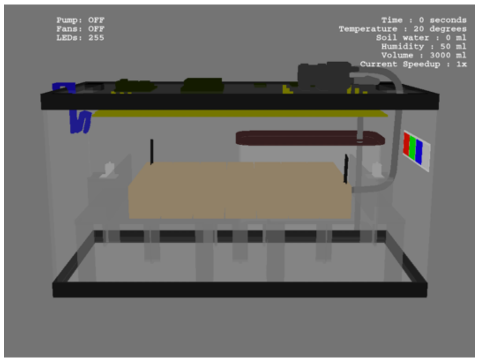


Figure 3: 3D graphical simulator of the TerraBot.

The course itself is structured around teaching AI and system integration concepts and applying those concepts to the autonomous greenhouse domain. Students must exercise critical thinking skills to select and implement algorithms that best apply to the greenhouse domain, and utilize software tools (some open source, some that we developed as part of the course) that enable them to develop robust and reliable agents that can operate the TerraBot autonomously for weeks at a time.

The course has four individual assignments, three group project assignments, two group testing assignments, and two group greenhouse deployments with in-class presentations. The assignments are scaffolded so that the students build up their autonomous agent code base throughout the semester. All individual assignments are designed to apply lecture concepts to a diverse set of realistic applications, and group assignments focus on merging their individual contributions and adding new algorithms to aid in growing plants in the TerraBot. The assignments culminate in two 2-week long grow periods (which we often extend for an additional 1-2 weeks, to give the plants more time to mature).

Overall, the course has evolved into a very well-regarded offering for undergraduates majoring in AI (36%), Computer Science (46%), Electrical Engineering (9%), and other (9%). Class composition has been 61% seniors, 36% juniors, and 3% sophomores. Students particularly comment on how they appreciate the opportunity to apply the theoretical concepts they are learning to a real-world problem, especially one in which there is no "correct" answer. We believe that such experience will be invaluable in their future careers, be it in industry or academia.

The TerraBot

The TerraBot (Figure 2) is housed in a 10-gallon fish tank and has three actuators: a water pump, controllable red/blue LED grow lights, and fans to control humidity and temperature. It has seven different sensor types: two each (to help deal with noise) of humidity, temperature, light, and moisture sensors; a color camera, an ultrasonic sensor to determine the level of the water in the reservoir, and during the COVID-19 pandemic we added a microphone, so that students testing the system remotely could hear when the pump and fans turn on and off. The plants are grown in a rock wool medium, which sits in an aluminum tray and is fed by a hose punctured with small holes, resembling drip irrigation. A 3D printed measuring stick, with markings every centimeter, is used to estimate the height of the plants from an uncalibrated color camera. The tray sits on a laser-cut acrylic base, and a laser-cut acrylic plate is used to cover the top of the tank and hold the lights and all the electronics. We grow both radishes and lettuce – the radishes generally flourish, but the lettuce is more difficult to grow, needing more precise growing conditions. In this way, most of the agents will be able to grow radishes, but only the better agents will be successful with the lettuce.[1] Each TerraBot costs about $400, which includes sheets of rock wool for the grow periods (the only consumable cost, aside from the seeds).

The TerraBot is controlled by an Arduino that connects the sensors and actuators, and a Raspberry Pi that communicates with the Arduino and runs the TerraBot infrastructure software and the students' agents (the camera and microphone are also connected to the Pi). The Robot Operating System (ROS) is used to communicate between all of the software entities, including the software running on the Arduino (Quigley et al. 2009). The TerraBot software and the students' agents are written in Python.[2]

The TerraBot has undergone several iterations in the three years since the first prototype. For one, connectors were added to the wires to make it easy to unhook the sensors and remove the top plate. The growing medium, which was originally a mat meant for growing microgreens, was replaced with rock wool that is about 1.5" deep, and the moisture sensors were repositioned to make it easier to determine when the plants needed to be watered. For the Fall 2022 iteration of the class, we experimented with load cells to weigh the pan and its contents, to better enable the agents to determine when to water – which has consistently been the most difficult task.

Perhaps the biggest change, though, was building a 3D graphical simulator of the TerraBot (Figure 3). Since plants grow so slowly, it is difficult to develop and debug software

[1]Some of the teams take their radishes and lettuce home to eat as a salad after we turn off their autonomous agents.

[2]The TerraBot software, plans for constructing the greenhouses, and details on using the hardware and software is available at https://github.com/reidgs/TerraBot. Lecture slides and assignments can be obtained by request.

on the actual hardware. The simulator, which has the exact same ROS interface as the physical greenhouse, enables students to safely and quickly test their agents. The simulator is as faithful to the real world as we could reasonably achieve. It simulates the growth of plants, evaporation and transpiration, and the fluctuation of ambient light, humidity, and temperature, depending on the time of day, whether the fan or lights are on, etc. Even the camera is simulated, enabling students to try out their computer vision algorithms before deploying on the hardware. Time can be sped up, so that days of testing can be accomplished in a matter of minutes (although the simulation automatically slows down when the pump or fans are on, since the agent needs to have real-time control over those time-sensitive operations).

The TerraBot simulator keeps track of the state of the (simulated) greenhouse, computing temperature and humidity based on whether the lights and fans are on, the amount of moisture in the "soil", and the state of the plants (e.g., more mature plants transpire more). The health of the simulated plants is calculated based on the amount of light and water they receive, along with the temperature and humidity levels (there are optimal levels for both). As the plant health deteriorates, the graphics change from bright green upright plants to yellow and brown, drooping plants. Students can use this to see how their plants are doing, and can create computer vision programs to detect unhealthy plants.

The Curriculum

The main objective of the course is to teach students how to integrate AI techniques into a complete system that can operate in a dynamic and uncertain environment for weeks without human intervention. In addition to teaching the use of AI techniques to support autonomous operation, the course emphasizes the need for a well-structured software architecture and the importance of thorough testing and validation prior to deployment. Finally, the course includes discussion of ethical issues involved in the deployment of autonomous agents. The course prerequisites are an introductory class in either Artificial Intelligence or Machine Learning (which, in turn, have prerequisites in programming and linear algebra).

Unit 1: Infrastructure. The course begins with two lectures that provide background information that is necessary for understanding and carrying out the assignments. The first lecture introduces the concept of middleware and the Robot Operating System (ROS) (Quigley et al. 2009), which is the way the students' code interfaces with both the hardware and the simulator. We discuss publish/subscribe and query/response forms of message passing, how to send and receive messages in ROS, and the various messages that the TerraBot supports. The second lecture introduces the basics of growing plants – specifically, how moisture, light, temperature, and humidity must all be kept in good balance to ensure optimal growth. Various approaches to autonomous greenhouses are presented. Then, the TerraBot hardware is introduced, including how the various sensors and actuators are critical in determining the growth of the plants.

Unit 2: Software Architecture. The curriculum presents different architectural approaches to autonomous systems. Students are exposed to behavior-based (reactive) architectures (Brooks 1986) and layered architectures (Schreckenghost et al. 1998), and the pros and cons of each are presented (Coste-Maniere and Simmons 2000). We argue that behavior-based architectures are well-suited for applications that need quick responses but little need for planning. Conversely, layered architectures that combine behaviors, planning and scheduling, and execution monitoring are well-suited for applications that are relatively slow moving but need to plan out how to effectively deal with conflicting demands for physical resources. The first homework assignment introduces students to the two architectural types and has them evaluate how well the architectures work in controlling the TerraBots (described below).

The course then proceeds to teach Finite State Machines (FSM) (Wang and Tepfenhart 2019), which form the basis for many implementations of the behavior layer of architectures for autonomy (Coste-Maniere and Simmons 2000). Simple FSM examples are presented (ATM, bread-making machine) for students to see how transitions between nodes are triggered by sensor input and how actions associated with transitions can affect the state of the world.

Unit 3: Computer Vision. In preparation for the first grow period, the next two lectures are on computer vision. We discuss the various sensors often used in autonomous systems (monocular, stereo, and depth cameras, lidar, radar) and present some relatively simple methods for segmenting out vegetative material from monocular color images. The lecture introduces color spaces, including advantages and disadvantages of the various color spaces, gamma correction, and color calibration using a calibration chart. We introduce the use of color histograms to do image classification, CNNs for object detection, and several vegetative indices (Wójtowicz et al. 2016) that have been formulated specifically for detecting foliage. The second lecture on computer vision introduces the OpenCV library (Bradski 2000) and has the students experiment with using OpenCV, in preparation for the group computer vision assignment.

Unit 4: Testing and Deployment. During the first two years of the course, we noticed that students did not spend sufficient time testing their agents prior to the grow period, leading to some very unreliable behavior such as failing to turn the pump off after watering when unexpected circumstances like sensors breaking or noisy sensor data occurred. This led us to include two lectures on testing and deployment, where we introduce methods for testing the concepts of safety and liveness. We discuss unit testing, system integration, and regression testing. We discuss various techniques, such as the use of simulation, testing for edge cases, and formal verification (Koopman and Wagner 2016). The TerraBot infrastructure contains a facility for describing tests that can be run either periodically or in response to some sensor condition, and the students are taught how to write test cases in that language. For instance, they can test whether the lights are turned off (and stay off) every night, or that once the pump is turned on it stays on for no more than 10 seconds. These tests are run during simulation, and we encourage students to run them for "weeks" and from many different starting conditions. After a testing assignment to practice with these new tools, the groups have a week to prepare for the first two-week grow period.

Ethics. We then briefly pivot to studying ethics of deployed autonomous systems. The ethics lecture is presented as a case study on a particular ethical topic such as autonomous warfare or AI in healthcare. Students are asked to read opposing viewpoints to prepare for classroom discussion. We present ethical responsibilities of AI systems including transparency, accountability, privacy, human autonomy, and the mitigation/reduction of bias. We discuss properties of trustworthy systems and how they may affect human trust in AI systems. Then, we discuss how an AI application may be engineered towards these principles and what challenges they may face when doing so.

Unit 5: Modeling Uncertainty. Because students have already taken at least one Machine Learning or Artificial Intelligence course (and in many cases more than one), our first machine learning lecture focuses on reviewing common models (SVMs, decision trees, KNN, linear regression, neural networks) and comparing and contrasting conditions when they might be used. The second lecture then focuses on time-series models and active learning methods. Learning about how to collect additional data given different kinds of uncertainty, especially data from humans, is important for deploying autonomous agents that can learn from feedback.

While the grow period is happening, the students often observe things going wrong, and they have no way to fix them without debugging and restarting their agent. Thus, we next present two lectures on handling uncertainty and execution monitoring. The first lecture covers state estimation – Kalman filters and variants (extended and unscented Kalman filters) and particle filters. The second lecture covers two approaches to monitoring: fault models and expectation-based approaches. We introduce students to Failure Mode and Effect Analysis (Stamatis 2003) and both symbolic (Kleer and Williams 1987) and probabilistic (Verma et al. 2002) expectation-based monitoring, that use model-based methods to detect divergence from nominal behavior. Students are encouraged to incorporate both state estimation and monitoring techniques into their agents, in preparation for the second grow period.

Unit 6: Explainability. Students also typically notice that their agents perform in unexpected ways. To address this, we have two lectures on generating explanations of system behavior. We cover visualization techniques for neural network classifiers (Konam et al. 2018), model-based explanations (Hayes and Shah 2017), reward-based explanations (Sukkerd, Simmons, and Garlan 2020), and legibility (Dragan, Lee, and Srinivasa 2013). We then spend a lecture on a specific model-based technique for generating "what", "why" and "why-not" explanations using execution traces generated by an FSM.

Unit 7: CSPs and Planning. We spend a lecture on constraint satisfaction problems (CSPs), talking about why they are useful and how they work, conceptually. Students have an in-class exercise to code a solver for Kakuro (Simonis 2008) using the OR-Tools package[3]. The next lecture presents how CSPs can be used for activity scheduling, using either an approach where each activity (behavior) runs for a fixed length of time or an approach where different behaviors can run for different lengths of time. As an in-class exercise, students use OR-Tools to do simple classroom scheduling using both approaches, and compare and contrast them in terms of ease of coding and efficiency of the resulting schedules. We then have a lecture on resource utilization – how to create schedules that prevent overuse of scare resources (in our case, the resources are the lights, fans, and pump, which cannot be used by different behaviors simultaneously) and minimizing resource use (e.g., minimizing use of the fans while still maintaining a good level of humidity).

We also have two lectures on planning techniques. While we have presented different techniques over the years, in the latest edition of the course we focused on POMDPs (Kaelbling, Littman, and Cassandra 1998) and Monte Carlo Tree Search (Browne et al. 2012). For subsequent semesters, we are considering having a lecture on deep reinforcement learning (Mnih et al. 2013), as this is a trending topic in AI agents.

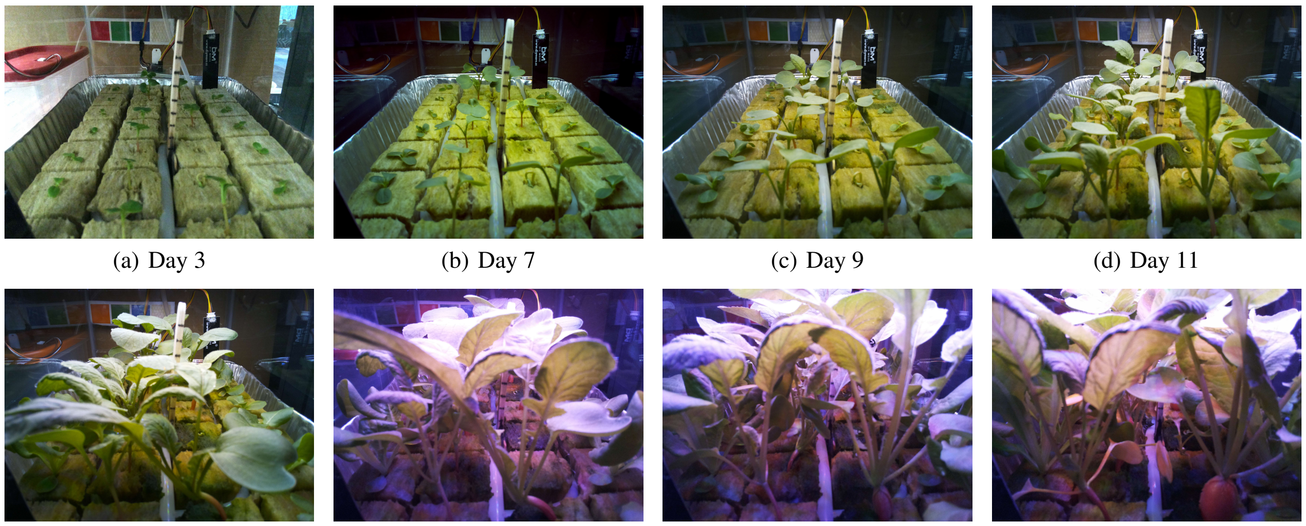
Case Study. At this point, the students are in the midst of the second grow period, so subsequent lectures do not impact their agents. We have an interactive class that presents a case study for how one would design an autonomous home care assistant. We introduce Maslow's hierarchy of needs (Maslow 1943) and have the students discuss what tasks are both feasible and desirable to be done by an autonomous agent, given current technology. In small groups, students come up with requirements that such an assistant agent would need, including the behaviors, scheduling, monitoring, and learning capabilities. Finally, we discuss how to test and validate the agent and how it can be designed such that an elderly user would trust it.

The Assignments

At the start of the semester, students are given code defining "tasks" that control the light level, humidity, temperature, and soil moisture of the greenhouse, implemented using if/else statements. They are also given starter code for an autonomous agent that takes in sensor data, runs one or more of the tasks, and outputs actuator commands based on those tasks. All of the code is written in Python within the ROS middleware framework, which allows for seamless incorporation of multiple hardware platforms and sensors. The starter agent can control the greenhouse, though not well. The assignments slowly add more complexity to the agent in order for it to grow plants more effectively. Additionally, most assignments include a written portion that asks students to analyze, describe, compare and contrast, or reflect on their experiences with the assignment and/or the greenhouse.

Assignment 1: Architectures and ROS. The first (individual) assignment is designed to help students understand the basic structure of the greenhouse agent. They are tasked with first exploring how ROS moves data between different "nodes" that send and receive data. Then, they use that framework to connect the greenhouse agent tasks to the sensor and actuation data streams. Given that the agent can now

[3]https://developers.google.com/optimization



(a) Day 3 (b) Day 7 (c) Day 9 (d) Day 11

(e) Day 13 (f) Day 15 (g) Day 17 (h) Day 19

Figure 4: A typical progression of plant growth in the Terrabot. Radish (middle two rows) and lettuce (outer rows) germinate in about 3 days. After 19 days, full-grown radishes are visible.

receive data and send actuator commands, they then implement the two different agent architectures presented in class (behavior-based and layered). Finally, the students evaluate the benefits and costs of using each architectural approach and then apply what they have learned as they assess which approach would be best for a new application.

Assignment 2: Finite State Machines (FSMs). The autonomous greenhouse agent (and many other applications) do not require complex search algorithms to determine what actions to take because the dynamics of the environment are not well modeled. Instead, we use FSMs, which can be very reactive to the changing environment. This individual assignment requires students to first practice building finite state machines on an autonomous coffee maker application. Then, they move on to re-implement the greenhouse tasks, originally implemented with if/else statements, using an open-source FSM library[4]. Students must compare and contrast if-statements and the FSM framework in terms of testability. Finally, we ask students to begin to develop tests that they could use to evaluate whether their FSM implementation matches the starter code.

Assignment 3: Computer Vision. Students strengthen their agent's ability to monitor plants by adding a new sensor stream – camera images. In this group assignment, three-member teams of students work to add vision processing to improve their agent behavior. They implement a new behavior to autonomously take pictures using the camera, and they use the basic computer vision techniques that we cover in lecture to develop algorithms to segment plants from the background and assess plant growth and health.

Finally, in anticipation of the first grow period, we have the students move their simulation code to the real greenhouses. They must characterize the behavior of the water pump and the noise in the moisture sensors so that they can perfect their agent's behavior on the real hardware.

Grow Period A: Testing and Deployment. We found that students struggle to find ways to test their code. They have a hard time finding edge cases that should be tested and are reluctant to run tests (in simulation) for hours or days in order to find bugs that occur later in the growing period. They are concerned about breaking the greenhouses, and so do are hesitant to test on the real hardware. In order to encourage the students to rigorously test their agents, the students have a one-week group assignment to create and validate tests of the agent prior to deployment. We require them to 1) develop a logging facility that can be used to monitor the agent over time, 2) compare and contrast sensor data from the simulator and real hardware over a 24-hour period and adjust any necessary thresholds in their tasks (e.g., pump on time), and 3) develop an email task to send status messages each day so that the students (and professors) can detect failures that occur during deployment.

Immediately following this assignment, the students start their agents on the real hardware and we let them run autonomously for two weeks. We encourage students to monitor their agent's behavior and intervene if there is a major issue. Following the deployment, the students present to the class about their camera algorithms, their tests, how their agent performed and why, and what takeaways they learned to improve their agent for the second deployment.

Assignment 4: Sensor Modeling. Students work individually to develop machine learning models to predict temperature and humidity sensor values over time. Students practice feature engineering, feature selection, and model selection. This assignment serves to connect their more-theoretical machine learning education with realistic applications. We find that students often struggle with this assignment if they try to rush and do not spend the time to analyze

[4]https://github.com/pytransitions/transitions

the sensor data, determine and engineer useful features, and then run feature and model selection with those features.

Assignment 5: Execution Monitoring. In this group assignment, students consider multiple ways for the agent to autonomously change its logic based on sensed information. First, students write code to measure insolation (the amount of light the plants receive), estimate how much opportunity there is for lighting the rest of the day, and autonomously modify the light levels to reach, but not exceed, a pre-defined target insolation for the day. Second, students implement a Kalman filter for the humidity sensors. Finally, we suppose that our agent can detect when sensors and wire connections are not working, and have the students implement a logic-based diagnostic tool to determine the likely culprit of a set of failures.

Assignment 6: Explainability. Students work individually to generate explanations about finite state machine transitions. First, they write code to automatically analyze FSMs and log files of sensor data to generate "why" explanations as an ordered list of transitions that occur. Students then implement breadth first search through the FSM transitions to generate a "how" explanation of certain actuator outcomes. In order to do this, they have to test all possible transitions that could occur for different sensor values. Valid sequences of sensor values and FSM states that produce the desired output form the explanations. Finally, they use counterfactual approaches to implement "why not" explanations. Explaining why something did not happen is a lot more complex, since they have to reason about what might have happened to lead to an actuation, but did not.

Grow Period B: Testing and Deployment. Before Grow Period B, student groups test their code again on the real hardware, update their code based on their takeaways from Grow Period A, and also add to their email behavior to be more explainable. The emails must incorporate a wide variety of data that they pull from several different agent tasks. For example, students utilize their computer vision assignment to email day-to-day plant growth and the status of their plants (e.g., as healthy, ok, or unhealthy).

For this grow period, students are penalized if they restart their agents and the reservoirs are filled with only 1500ml of water. After two weeks of autonomous operation, the groups present their algorithms and email designs. Since this is the last grow period, we allow the agents to continue running for another two weeks until final exam time. At that point, many of the plants are quite large and the radishes fully matured (Figure 5).

Assignment 7: Scheduling. This last group assignment, which occurs during grow period B, is more open-ended than the others. Students use constraint-satisfaction techniques to optimize the greenhouse agent schedule based on a set of requirements. Students can choose from two ideas we provide or pick their own. Students need to implement execution monitors to determine when behaviors are actually needed (e.g., they turn actuators on) and when they should be needed (e.g., some sensor limit is reached when the associated behavior is not running) and update the schedule based on the sensed data. Student groups present their algorithms and resulting schedules to the class and we discuss different scheduling strategies and lessons learned.

Figure 5: Students with harvested radishes.

Assignment Grading and Evaluation

The course grade is broken down in the following way:

- 20% Individual Assignments (5% each)
- 30% Group Assignments (10% each)
- 10% Grow Period Group Testing Assignments (5% each)
- 10% Grow Period Deployments/Presentations (5% each)
- 10% Midterm
- 15% Final Exam
- 5% Peer Evaluation

We chose to grade individual and group assignments in simulation for completeness (passing all required tests) and robustness (designing and submitting additional tests). For each assignment, we provide the simulator and some of the test cases to students so that they can debug their assignments before submission. We reserve some test cases for grading to encourage students to test more thoroughly before submitting. For the testing and deployment assignments, we grade in three ways: First, students must provide their documented test cases, which we grade for the variety of tests that are performed and successful ability to run them. Second, we grade the ability of the autonomous agents to grow plants over the two-week deployments. Third, we have found that their reflection and takeaways are important for student learning and improvement through the semester. The midterm and a final exam ask students to apply what they have learned to a new autonomous agent application through essay-based questions. We utilize rubrics to grade the essays based on their ability to consider a new application and apply course concepts to it. Finally, we ask students to write peer evaluations for members of their group as a motivation for them to stay involved in the projects throughout the semester.

Course Evaluation and Iteration

We have run this course four times – in the Falls of 2019–2022, with 13, 11, 9, and 28 students respectively. In the middle and at the end of the semester, we asked for student input and feedback about the course. The course structure and assignments have been updated in response to the student feedback. We report here feedback from Fall 2021, which has the most recent updates to the course that are reflected in this paper. Quantitatively, students reported spending an average of 6.5 hours per week (s.d. 0.94) on the assignments for the course plus 3 hours for lecture. They rated the difficulty of the course as 4.78 on a scale of 1-9 (s.d. 1.13). Qualitative feedback is given below.

Lectures. Students reported that they enjoyed the lecture content and the discussions.

> "I definitely enjoyed the content that was covered in lecture. All of the topics seemed pretty relevant to the overall course goals. I also enjoyed the discussions we had during class, especially because a lot of my actual understanding of the concepts occurred during those discussions."

They also reported enjoying the active learning activities that we incorporate in lectures. These activities, such as having student groups solve Kakuro problems using the Google OR-Tools package, serve two purposes: 1) practicing using software packages in advance of assignments; and 2) understanding how to translate high level concepts into mathematical and computational constraints.

One comment we received was that some of the lecture material is taught too high level. They want both more of the theory behind the algorithms and also a more direct application of the algorithms from class to the homework. We are iterating on the lecture material to provide more of the theoretical basis for the algorithms we teach.

Assignments. For the individual assignments, students reported that the assignments were reflective of the lecture material, though sometimes it was challenging to understand how to apply the concepts when starting the assignments:

> "For me, the most difficult part of the assignments is figuring out what direction to take at a high level, but once I figure out a general plan of action the assignment is usually pretty straightforward and not too difficult."

With respect to group assignments, one student said:

> "I would say the group assignments were definitely rewarding as they were of a larger scale compared to individual assignments."

Grow Period Deployments. Finally, we asked students to reflect on the grow periods and what they did to monitor their autonomous agents. All the students liked watching the plants grow as a result of their autonomous agent. They internalized their success based on the success of their plants:

> "The grow periods went well! I was happy with the success of the radishes; the lettuce results were unfortunate, I would have loved to have had an extra grow period to take another shot at fixing that."

Autonomous agents should not be left alone indefinitely, yet many students in the first two years of the course would not check their agent during deployment. One change that we had made for Fall 2021 was to have the students create email tasks for the grow periods, so that they would be more likely to view and respond to issues with their plants, rather than having to remember on their own to check. Several groups were able to fix bugs in their agents or in their reporting by analyzing their emails.

> "To monitor the agent, we would all look at the daily emails and make note of anything that seemed off. I noticed repeat/duplicate pictures during grow period A, which helped us make changes for the next grow period (since we were a little too late in the game when I noticed). I also noticed the incorrect plant heights, which led us to recalibrating the coordinates of the ruler, as well as the masks."

Overall Comments. The students were overall very happy with the course. They rated the course a 4.8 on a scale from 1 to 5. They were particularly happy with the uniqueness of the course topics:

> "I'm really enjoying this class! It strikes a great balance of learning about theory of the systems development work involved in creating an autonomous agent and practical applications..."

Their top request for changes for next year were 1) a bigger focus at the beginning of the semester on hardware, 2) more trips to the lab to see the growing and harvesting of the plants and 3) a third grow period. We found these suggestions interesting because of their focus on the physical agent and deployment rather than on the assignments and code. Because this course was initially given in the midst of the pandemic, we did not want students to be physically close to each other to view the hardware or the plants up close, though this constraint is no longer necessary. The desire for a third grow period is encouraging because it shows the students enjoyed and got feedback from the deployment of their agents throughout the semester. However, it is challenging to implement because we need enough content and time between the grow periods for students to reflect on their work and to make meaningful changes to their agent.

Conclusion

We set out to create a course to teach integration of advanced AI concepts and deployment strategies, but which did not focus on robotic applications. We elected to create autonomous greenhouses, which we call TerraBots, that grow radishes and lettuce. Students learn about agent architectures, finite state machines, computer vision, machine learning, uncertainty and execution monitoring, explainability, ethics, and scheduling. They complete individual and group assignments that apply the course concepts to the TerraBots as they build up the complexity of their autonomous agents. Twice during the semester, they deploy their agent, monitor its well-being, and present their experiences to the class. Overall, the students have found significant value and challenge in the experience of building and deploying AI systems. Due to the success of the course in previous semesters, our Fall 2022 enrollment has nearly tripled compared to previous semesters, and we look forward to continuing to iterate on the course materials.

Ethics Statement

This course focuses not only on the technical aspects of the deployment of AI systems but also on the ethical aspects of autonomous agents through a module on ethics and inclusion of ethical issues in our case studies.

Acknowledgments

Thanks to Jack Cameron, Mike Xu, Cara Li, and Greg Armstrong for developing the TerraBot hardware, David Buffkin for developing the simulator, and George Kantor and Gene Giacomelli for invaluable automated plant-growing advice.

References

Bradski, G. 2000. The openCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11): 120–123.

Brooks, R. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal on Robotics and Automation*, 2(1): 14–23.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1): 1–43.

Cal State Los Angeles. 2020. Autonomous Vehicles and Smart Infrastructure. https://ecatalog.calstatela.edu/preview_course_nopop.php?catoid=66&coid=492981. Accessed: 2022-12-06.

Castelló Ferrer, E.; Rye, J.; Brander, G.; Savas, T.; Chambers, D.; England, H.; and Harper, C. 2019. Personal Food Computer: A New Device for Controlled-Environment Agriculture. In Arai, K.; Bhatia, R.; and Kapoor, S., eds., *Proceedings of the Future Technologies Conference (FTC) 2018*, 1077–1096.

Columbia University. 2020. Autonomous Multi-Agent Systems. http://www.columbia.edu/~ja3451/courses/e6899.html. Accessed: 2022-12-06.

Coste-Maniere, E.; and Simmons, R. 2000. Architecture, the backbone of robotic systems. In *Proceedings IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, 67–72.

Dragan, A. D.; Lee, K. C.; and Srinivasa, S. S. 2013. Legibility and predictability of robot motion. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 301–308.

Hayes, B.; and Shah, J. A. 2017. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*, 303–312.

Institut Polytechnique de Paris. 2020. INF581 - Advanced Machine Learning and Autonomous Agents. https://moodle.polytechnique.fr/course/info.php?name=INF581-2020. Accessed: 2022-12-06.

Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2): 99–134.

Kleer, J. D.; and Williams, B. 1987. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1): 97–130.

Konam, S.; Quah, I.; Rosenthal, S.; and Veloso, M. 2018. Understanding convolutional networks with APPLE: Automatic patch pattern labeling for explanation. *Artificial Intelligence, Ethics, and Society*.

Koopman, P.; and Wagner, M. 2016. Challenges in Autonomous Vehicle Testing and Valdidation. *SAE International Journal of Transportation Safety*, 4(1): 15–24.

Maslow, A. 1943. A Theory of Human Motivation. *Psychological Review*, 50(4): 370–396.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *Neural Information Processing Systems (NIPS)*.

Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. Y.; et al. 2009. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, 5.

Schreckenghost, D.; Bonasso, P.; Kortenkamp, D.; and Ryan, D. 1998. Three tier architecture for controlling space life support systems. In *Proceedings. IEEE International Joint Symposia on Intelligence and Systems*, 195–201.

Simonis, H. 2008. Kakuro as a constraint problem. *Proc. seventh Int. Works. on Constraint Modelling and Reformulation*.

Stamatis, D. H. 2003. *Failure mode and effect analysis: FMEA from theory to execution*. Quality Press.

Sukkerd, R.; Simmons, R.; and Garlan, D. 2020. Tradeoff-focused contrastive explanation for mdp planning. In *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 1041–1048.

University of Central Florida. 2007. Engineering Applications of Autonomous Agents. http://www.cs.ucf.edu/~lboloni/Teaching/EEL6938_2007/index.html. Accessed: 2022-12-06.

University of Edinburgh. 2018. Decision Making in Robots and Autonomous Agents. http://www.drps.ed.ac.uk/17-18/dpt/cxinfr11090.htm. Accessed: 2022-12-06.

University of Memphis. 2010. Control of Autonomous Agents. https://www.memphis.edu/cs/courses/syllabi/7760.pdf. Accessed: 2022-12-06.

University of Texas. 2015. Autonomous Multiagent Systems. https://www.cs.utexas.edu/~patmac/cs344m/. Accessed: 2022-12-06.

University of Texas Austin. 2022. Smart Cities. https://ugs.utexas.edu/bdp/programs/sc. Accessed: 2022-12-06.

Verma, V.; Fernandez, J.; Simmons, R.; and Chatila, R. 2002. Probabilistic models for monitoring and fault diagnosis. In *The Workshop on Technical Challenges for Dependable Robots in Human Environments. Ed. Raja Chatila*.

Wang, J.; and Tepfenhart, W. 2019. *Formal Methods in Computer Science*. Chapman and Hall/CRC.

Wójtowicz, M.; Wójtowicz, A.; Piekarczyk, J.; et al. 2016. Application of remote sensing methods in agriculture. *Communications in Biometry and Crop Science*, 11(1): 31–50.