Scaffolding Strategies for Teaching ROS 2: An Approach Using JupyterLab and iRobot™ Education's Create® 3 Robot

Miss Kathryn Lara Wujciak, Tufts University

Kathryn Wujciak recently graduated from Tufts University with a bachelor's degree in mechanical engineering and is pursuing a master's in the same field this year. She has been a teaching assistant for multiple robotics courses throughout her time at Tufts. Kathryn is passionate about educational robotics and hopes to lower the barrier of entry for new engineers.

Dr. Briana M Bouchard, Tufts University

Briana Bouchard is an Assistant Teaching Professor in the Department of Mechanical Engineering at Tufts University. She earned her Ph.D. in Mechanical Engineering, M.S. in Engineering Management, and B.S. in Mechanical engineering from Tufts University. Her research focuses on educational robotics and increasing the accessibility of ROS 2 using the Create 3 robot. She primarily teaches engineering design and introductory robotics courses at Tufts and has previously taught courses in electronics, electronic portfolios, and the Internet of Things.

Prof. Chris Buergin Rogers, Tufts University

Chris is a professor of Mechanical Engineering at Tufts University with research in engineering education, robotics, musical instrument design, IoT, and anything else that sounds cool..

Scaffolding Strategies for Teaching ROS 2: An Approach Using JupyterLab and iRobot™ Education's Create® 3 Robot

I. Introduction

ROS 2 is an open source software development kit for various robotics applications. ROS 2 stretches across industries to provide assistance in research, prototyping, and development. It is an advanced computer science concept often discussed at the graduate level and above. Because it requires some background in either Python or C++, Linux, and understanding of publisher/subscriber messaging structure, it is not commonly taught in undergraduate classrooms. However, it is often used in the robotics industry at the professional level. We aim to lower the barrier of entry of using ROS 2 using scaffolding tools and strategies with JupyterLab and iRobot™ Education's Create® 3 robot. The Create® 3 robot is ROS 2 compatible, a rarity among educational robots. We used this platform to teach ROS 2 to first-year students across multiple engineering disciplines. The JupyterLab and Create® 3 robot setup made it possible to implement scaffolding techniques to optimize learning for students at an individual level, with class exercises starting simple and growing in complexity. Outside of class, collaboration during project-based learning further expanded the students' understanding of the in-class topics. Using previous pedagogical theory and data from this class, a new and improved JupyterLab is under development to efficiently teach ROS 2 to undergraduates with little to no prior experience. This paper provides an overview of the theoretical underpinnings of our work and outlines our initial implementation for adaptation by others.

II. Literature Review

A. Individual

Scaffolding refers to a structured framework that supports and guides learners while gaining new skills or knowledge, and fades as students become more proficient [1]. Scaffolding is an educational strategy that instructors have used to bridge the gap between the student's current understanding and the more complex learning goal [2]. Domain-general scaffolding is support that stretches across all learning environments [1], [3]. This could include improving problem-solving skills, critical thinking, monitoring one's own progress, or goal-setting [1], [3]. For example, an instructor might provide a worksheet where the student writes weekly goals to help develop their goal-setting skills. The student may learn how to effectively set goals which could help in many areas of learning. Domain-specific scaffolding is instructional guidance that is specific to one domain [1], [3]. It aims to make understanding complex concepts in that domain more attainable. For example, in a computer science environment, it could look like "fill-in-the-blank" code or black-boxing code. There are many techniques within scaffolding, and below we will detail those relevant to our course.

On an individual level, scaffolding can improve the comprehension of computer science topics [1], [3], [4], [5], [6]. Scaffolding provides the structure and tools to help learners develop independence as learners. It is one out of many principles in self-regulated learning [1], [3]. Self-regulated learning (SRL) is a method of teaching that puts the student at the center of their own learning [1], [3], [6], [7]. Students monitor their own progress, set personal goals, and use specific strategies. It is accepted that the use of scaffolds can support SRL processes [1], [6], [7]. Zheng [1] did a meta-analysis on 29 articles that looked at the effects of SRL scaffolds on academic performance in computer-based learning environments (CBLEs), concluding that scaffolds within SRL improve academic performance. This study suggests that a CBLE is optimal for scaffolding due to the interactive nature, flexibility, and collaboration abilities. This meta-analysis found that SRL scaffolds had the most positive impact in a CBLE in comparison to other learning environments such as a traditional lecture setting [1]. This is because in CBLEs that practice open-ended learning, students are pushed to have stronger engagement with the material and higher intrinsic motivation in comparison to traditional learning environments [8]. Research indicates that successful learners often exhibit high engagement and intrinsic motivation during a self-regulated learning activity [9]. The interaction between students' intrinsic motivation and their performance within SRL in a CBLE significantly impacts their learning according to a study conducted in an introductory engineering class [9]. CBLEs are optimal environments for SRL scaffolds. Scaffolds can range from conceptual to procedural, and strategic to metacognitive. Domain-general scaffolds are more effective in terms of academic performance than domain-specific scaffolds [10], but both used together was the most effective as presented in [1, Tab. 7], [11]. SRL fits well within domain-general scaffolding, as it is a skill that can be applied to many learning environments. SRL puts the student in the driver’s seat and lets them take control of their learning, with the help of domain-specific scaffolding techniques [1].

Effective scaffolding comes in four phases: planning, monitoring, control, and reaction and reflection [3], [8]. The planning phase involves planning for the problem such as guiding questions, making a concept map, or planning ahead as seen in [1, Tab. 1], [3]. The monitoring phase could have diagrams, prompts for self-explanation or reasoning, or cognitive feedback done by the student [3], [12]. In the control phase, there could be worked out examples, processing and reflective prompts, or guiding questions [3], [10]. Lastly, in the reflection phase, students reflect on the learning they’ve done [3], [13]. As previously mentioned, effective scaffolds can be both domain-general and domain-specific in each phase. In the context of computer-based learning environments, or CBLEs, prompts (both domain-general and domain-specific) are often used to help learners navigate complex tasks or problems, monitor their progress, and regulate their learning strategies [1], [3], [5], [10]. Research on CBLEs found that prompts were the most effective scaffolds for supporting SRL processes in CBLEs, especially for processes during the control phase [1], [3].

One specific scaffolding technique is well-timed support. The concept of scaffolding is based on the idea that learners can achieve learning objectives that they could not reach on their own with the help of well-timed support. Well-timed support proved to be a very effective domain-specific scaffolding tool. An example of well-timed support is the Test My Code (TMC) software developed by the Department of Computer Science at University of Helsinki to benefit both the instructor and student [4]. TMC (1) enables building of scaffolding into programming exercises; (2) retrieves and updates tasks into the students' programming environment as students work on them, and (3) causes no additional overhead to students' programming process [4]. Small exercises build into bigger programs. The incremental tasks replicate the problem-solving process, helping the student not only acquire coding knowledge but also problem-solving skills. TMC was used with students as young as 11 years old. It has been used in several programming courses at the University of Helsinki, and studies indicate that students found TMC helped them learn programming more effectively and efficiently. Hellas et al. emphasize the importance of well-defined and achievable tasks for students, especially in the early phases of a course, to build self-confidence and encourage them to tackle new challenges [4].

When discussing well-timed support, it is natural to consider feedback. There are two types of feedback: external and internal [5]. External feedback could be feedback from an instructor or generated from a computer. Internal feedback could be self-reflection or self-assessment. The relationship between internal and external feedback in SRL is crucial [5]. Internal feedback (learner's self-assessment) can be enhanced by external feedback (from environment and peers or instructors) [5]. The combination of both internal and external feedback improves student performance in SRL [5]. It allows the student to have a more concrete idea of their understanding of the material. Feedback is important when designing a curriculum that involves scaffolding in self-regulated learning environments [1], [3], [5], [10].

Having explored multiple scaffolding strategies, it's important to discuss the practical application of them within various learning activities. In other words, now that we have the techniques, how will we implement them? Scaffolding can be presented in many different types of class activities. Doukakis et al. [2] suggest pedagogical strategies that are most effective for teaching that content, and the technological tools and resources that can be used to support learning. The main learning categories include Think (reading, discussing, listening), Practice (algorithm development, algorithmic puzzles), Interpret (case studies, analyzing algorithms), Apply (open-ended problems, project-based learning), Evaluate (solution testing, peer evaluation), and Create (presentation, documenting, product development) [2]. For example, well-timed support could be incorporated in a "practice" activity such as algorithm development. Additionally, feedback could be applied to an "evaluate" activity such as solution testing. The researchers in [2] suggest multiple technology-integrated learning activities that could include a number of different scaffolding techniques within them. Although it is not necessary to apply activities in each

category for every class, it is important to diversify the activities in and out of class so students are exposed to multiple ways of learning and problem-solving.

B. Collaboration

Scaffolding in a CBLE is very effective especially for individual learners as discussed above. However, another important component of CBLE is collaboration [3], [14]. Collaborative learning encourages students to engage in discussions and evaluate different perspectives, which helps develop their critical thinking skills [14]. Collaborative learning is necessary to prepare engineering students for the professional world, where it is a daily occurrence [14], [15], [16].

E. Dringenberg et al. [15] identified and described the categories of collaborative, ill-structured problem-solving experiences from the interview data of 27 first-year engineering students. The study's findings suggest that students' experiences of collaborative, ill-structured problem-solving (open-ended, loosely structured) can improve and educators should design learning environments that encourage students to experience collaboration in various ways. However, the common thread in all collaborative interactions is gathering more than one perspective. Seeking multiple perspectives benefits each student in the group and is particularly helpful in a CBLE [14], [15], [16]. It is important to give space such that all perspectives are welcome and encouraged.

Collaboration invites multiple perspectives to solve a problem. If all perspectives are welcome, students are given space to explore originality. Haungs et al. [16] discuss how to create a collaborative environment where students are given open-ended assignments to promote creativity. This allows students to work together in teams, communicate with each other, and solve problems collectively. A perfect environment for collaboration is project-based learning [14], [16]. Project-based learning (PBL) is a teaching method that follows constructivist learning [16]. A traditional method of learning would involve learning facts, while constructivism is a combination of existing knowledge, social context, and the problem at hand [16]. Working individually as a professional engineer is uncommon, so it is necessary to expose students to project-based learning to gain experience [14], [16].

Although collaboration is key to project work, it is important to provide opportunities for students to actively participate in individual learning which can enhance collaboration in the future. They learn essential skills in independent work that they can later apply in collaboration with their peers. The students are more likely to collaborate after they have a chance to learn the content at their own pace [16]. Thus, the combination of individual learning (with scaffolding techniques) and collaborative learning is optimal when teaching advanced engineering topics.

III. Teaching ROS 2 with JupyterLab: Class Approach

Our class, *Introduction to Engineering: Remote Exploration with Roomba*, was composed of 31 first year engineering students. The class had one professor and 2 undergraduate teaching assistants (TAs). Both TAs studied ROS 2 in the summer prior to the class, so they had a basic working knowledge of the material. The main goal of the class was to give first-year students a sense of different engineering disciplines and engineering problem solving. It was only open to first year students who generally had limited coding experience. In-class activities were mostly individual, while weekly projects were done with a partner. In-class activities included significant domain-specific scaffolding, while the weekly projects were ill-structured and collaborative.

Class was composed of short lectures (10-15 minutes) and class activities (60-65 minutes). The short lectures, although included limited interaction, were examples of computational thinking activities as described by [2]. The class used JupyterLab for all class activities and homework. JupyterLab is a web-based interactive computing platform. Although it is compatible with many languages, this class used it with Python only. The JupyterLab was set up such that students could navigate to a server that housed all of the class material. Students would log in to their personal workspace, which contained their individual activities and assignments. The server was set up by the professor on a campus computer such that all students could access it. This software allowed for multiple mediums of interaction such as full code scripts, code blocks, and plain text (for explanations). There were also files that provided support such as troubleshooting pages and extra practice on various concepts. The robot used in this class was iRobot™ Education's Create® 3 robot. It will be referred to as "the robot" in this paper. It is one of the only ROS 2 compatible robots available for student use. In JupyterLab, code can be run one section at a time or line by line. For example, some files included portions of a full file that could be run one code block at a time initially, then it could be run all together. Additionally, JupyterLab allowed for text explanations to be written anywhere in the file.

The creative flexibility of JupyterLab + Create® 3 robot allowed us to implement different projects that spanned both physical fabrication and coding. For example, students were able to pull data from cloud services such as Airtable and send that data to the Create® 3 robot, while fabricating it to look like a racecar. Microprocessors could be connected to the robot and send data via serial communication. We used a Maker Pi RP2040, which is a microcontroller designed by Raspberry Pi®[1]. It is an inexpensive microcontroller with many options for external sensors and actuators. A single script in JupyterLab could control the robot and Maker Pi RP2040 together. For example, if the light sensor was triggered, the robot might drive forward. Each project in the class took advantage of the various capabilities of this setup.

All projects except for the midterm were done with partners. Projects aimed to replicate real life applications, encourage critical thinking, and draw multiple perspectives from the class. In one

[1] Raspberry Pi® is a trademark of Raspberry Pi Trading.

project, the robot was a baseball player. The "baseball player" was built on top and the robot had to run the bases, blinking and beeping at each base. Additionally, the students turned the robot into a proportional controller that acted as an autonomous car. All the robots were in a line and had to stay a specific distance apart while moving forward. The class project schedule is noted below in Table 1. In addition to learning engineering concepts, students learned to document and present in this class. After each project, students were required to "market" their project on a website called "Notion." This process is an example of the "Create" activity type [2]. Students were able to synthesize their work and transform it into something able to be presented to others.

Table 1. "Introduction to Engineering" class project schedule.

Week 1	**Baseball** - The robot will drive to each of the bases, then beep and blink at each one. *Concepts: Python basics* *Instructor preparation: Write a wrapper library (as noted in Fig. 2 and Fig. 3) that allows students to use minimal lines of Python code that are simple to comprehend.*
Week 2	**Jousting** - Students will control the robot with their keyboard and joust another robot. Students will build "knights" on top of the robots using laser-cut materials. *Concepts: Teleoperation, laser-cutting* *Instructor preparation: Use teleop code from iRobot™ Education to allow students to control the robot with their keyboard.*
Week 3	**TeleRobot Races** - Students will remotely race the robots to a finish line. They will do this by attaching their phone to the robot by building a phone holder on top. One partner will control the robot from another room via Zoom and teleop. *Concepts: Teleoperation, laser-cutting, 3D printing* *Instructor preparation: Use previous teleop code.*
Week 4	**Soccer Shootout #1** - Students will build a "leg" to allow the robot to score a goal by using a microprocessor and servo motor. *Concepts: Maker Pi RP2040, servo motor, laser-cutting* *Instructor preparation: Use previously written wrapper library and provide starter code to use the servo with the microprocessor during class activities.*
Week 5	**Relay Race** - The class robots will be placed in a line and must pass a ping pong ball by turning 180º, driving to the next robot, and delivering the ball. *Concepts: Maker Pi RP2040, light sensor, ROS 2 (publisher), laser-cutting* *Instructor preparation: Use wrapper library to introduce and use publishers in ROS 2 during class. Provide light starter code to use a light sensor with a microprocessor.*
Week 6	**Soccer Shootout #2** - The robot will score a goal by using a microprocessor and controlling it through the serial port. It will be autonomous such that once the light sensor is triggered, it will send that information to the microprocessor, then

	subsequently to the robot in order to perform an action (kicking a ball in the goal past a goalie robot). *Concepts: Maker Pi RP2040, light sensor, serial communication, ROS 2 (publisher)* *Instructor preparation: Introduce the serial port on the robot and serial starter code. Use previous light sensor code to send information to the robot over serial such that it moves.*
Week 7	**Obstacle Avoidance** - Students will control the robot through an obstacle course using a smartphone connected to Thunkables and Airtable. Thunkables is treated as the app interface, with Airtable handling the values being published to the robot in order to control its movement. *Concepts: Thunkables, Airtable, ROS 2 (publisher)* *Instructor preparation: Introduce Thunkables and Airtable then show how to publish values to a topic on the robot using both tools.*
Week 8	**Autonomous Cars** - Students will use ROS 2 to write a proportional controller that makes the robot drive "autonomously" behind another robot such that it doesn't crash into the "car" ahead, yet stays closely behind. *Concepts: Proportional controller, ROS 2 (publisher)* *Instructor preparation: Introduce proportional control concepts. Provide publisher code with less scaffolding for students to work through.*
Week 9	*Prepare for midterm*
Week 10	**Midterm** (individual) - Students will make a robotic animal of their choosing using sensors and actuators with ROS 2. *Concepts: sensors, actuators, ROS 2 (publishers, subscribers), laser-cutting, 3D printing* *Instructor preparation: Introduce subscribers and external sensors/actuators. Dive deeper into ROS 2 using class activities to remove wrapper library scaffolding.*
Week 11	*Prepare for final*
Week 12	**Final Project** (group) - The robot will joust another robot and try to throw it off its "horse" using primarily ROS 2 actions. The winner is the robot whose "horse" stays intact. *Concepts: ROS 2 (actions, publishers, subscribers), laser-cutting, 3D printing* *Instructor preparation: Introduce actions and continue to explain overall ROS 2 concepts without wrapper library.*
Week 13	*Work on final portfolio*
Week 14	*Work on final portfolio*

Week 15	**Final Portfolio** (individual) - Students submit a “Notion” portfolio which includes pages for each project done throughout the semester.

As noted above, each project was done in Python using JupyterLab. JupyterLab allowed students to learn in a SRL environment. JupyterLab made it possible to apply scaffolding and ease the integration of ROS 2 into projects. Gradual learning experiences through black-boxing code were applied to the provided scripts. Similar to TMC, JupyterLab made it possible to break down complex programming tasks into smaller, more manageable steps. This provided students with immediate external feedback on their code, and gradually increased the complexity of programming tasks as students progressed through the course. The interface took advantage of well-timed support. Students were able to see immediate feedback which either helped lead them to the correct solution, or gave them confidence to continue with the activity.

In order to incrementally increase complexity, it was important to start simple. Syntax was made easier by utilizing wrapper libraries that had more complex functionalities and were referenced in simpler scripts. For example, instead of writing a script to make the robot drive forward, students could use a prewritten wrapper library that would condense the code into one line instead of a full script. See Fig. 1 for the simplified script, and see the wrapper library in Fig. 2 and Fig. 3.

Fig. 1. Simplified script utilizing prewritten wrapper library (written by Chris Rogers).

```python
import rclpy, os, sys
from Subs.ROS2Lib import Drive, Audio
import time

class Create():
    def __init__ (self, namespace = ''):
        rclpy.init(args = None)
        self.namespace = namespace
        self.drive_client = Drive(namespace)
        self.audio_publisher = Audio(namespace)

        print('ros domain: ' + str(os.environ['ROS_DOMAIN_ID']))
        print('middleware: ' + str(os.environ['RMW_IMPLEMENTATION']))
        time.sleep(1)

    def beep(self, frequency = 440):
        '''
        Beeps
        '''
        print('publish beep ', end = '')
        self.audio_publisher.beep(frequency)
        time.sleep(1)
        print('done')

    def forward(self,dist = 0.5):
        '''
        Goes the distance and then stops the ROS2 connection
        '''
        speed = 0.25
        print('forward %0.2f: goal' % dist, end = '')
        self.drive_client.set_goal(float(dist),speed)
        print(' set ', end = '')
        self.wait(self.drive_client)
        print('done')
```

Fig. 2. “CreateLib” wrapper library (written by Dr. Chris Rogers) referenced in a simplified script.

```python
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
import time

from irobot_create_msgs.action import DriveDistance
from irobot_create_msgs.msg import AudioNoteVector
from irobot_create_msgs.msg import AudioNote
from builtin_interfaces.msg import Duration
import geometry_msgs.msg

class Audio(Node):

    def __init__(self, namespace = '/Happy'):
        super().__init__('audio_publisher')
        self.audio_publisher = self.create_publisher(AudioNoteVector, namespace + '/cmd_audio', 10)
        self.audio = AudioNoteVector()
        self.append = False

    def beep(self, frequency = 440):
        self.audio.notes = [AudioNote(frequency = frequency, max_runtime = Duration(sec = 1, nanosec = 0))]
        self.audio_publisher.publish(self.audio)

class Drive(Node):

    def __init__(self, namespace = '/Happy'):
        self.done = False
        super().__init__('drive_distance_action_client')
        self._action = ActionClient(self, DriveDistance, namespace + '/drive_distance')

    def set_goal(self, distance = 0.5, speed = 0.15):
        self.done = False
        goal_msg = DriveDistance.Goal()
        goal_msg.distance = distance
        goal_msg.max_translation_speed = speed
        self._action.wait_for_server()  # Wait for the server to be available and then send the goal.
        self._send_goal_future = self._action.send_goal_async(goal_msg)
        self._send_goal_future.add_done_callback(self.goal_request_callback)

    def goal_request_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected :(')
            self.done = True
            return
        self.get_logger().info('Goal accepted :)')
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Result: {0}'.format(result))
        self.done = True
```

Fig. 3. “ROSLib” library (written by Dr. Chris Rogers) that is referenced in “CreateLib.”

The ability to run one line at a time (or blocks of code at a time) helps the student understand the significance of each line in the script. Additionally, if the code blocks were sections of a larger script, this would aid in debugging. The student could run that block to determine if it returns any errors. At the end of the file, the student could run the script in its entirety. Additionally, students may have an opportunity to “fill in the blank” for certain code cells or have to fix broken code in order to run the script without errors. This allows students to understand each file line by line. This type of “algorithm development” [2] takes advantage of immediate feedback in SRL which is an effective method of scaffolding [5].

The JupyterLab software allows for many files to be at the student’s disposal. Some files provide a bit more help with more scaffolding (see Fig. 1.), while others allow the student to have more autonomy (see Fig. 4. below).

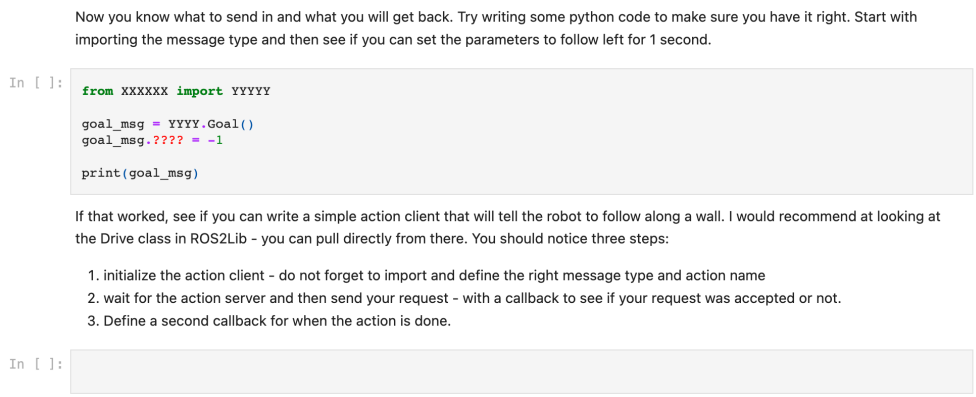

Fig. 4. Snippet of file with minimal scaffolding.

The JupyterLab supports SRL at an individual level. It is important to start with individual work to acquire technical skills which can then be brought to group work. The basis of knowledge allows students to be more confident when interacting with peers. Collaboration was also an important aspect in this class. As previously mentioned, all projects (except for the midterm) were collaborative. Project-based learning (PBL) was an important component of the class. Projects in class were intentionally ill-structured and open-ended [15]. This allowed for critical thinking. If there's never one right answer, students have the flexibility to think outside-the-box and explore the material they've learned. PBL is another activity suggested by Doukakis et al. [2] defined as an "Apply" activity type. PBL allows space to create collaborative environments and encourage creativity. The JupyterLab and robot setup made it possible to implement both domain-general and domain-specific scaffolding techniques to optimize learning for students. Although this curriculum hasn't been externally replicated, it may be of significant interest to other instructors who want to teach ROS 2.

IV. Methods

We collected data through three methods. As previously noted, the students would document their weekly projects on the web-based note taking platform Notion, which was used as an electronic portfolio platform in this course. These pages showcased each student's process and thinking throughout the project. We would also record field notes of classmate to classmate, classmate to TA, and classmate to instructor interactions during class. Lastly, we observed and interacted with students during office hours to experience their thinking and problem-solving. Pseudonyms have been used for all students quoted in the findings section.

V. Findings and student outcomes from Fall 2022 class

A. Individual

Multiple conclusions were drawn from the acquired data. There was an impact of scaffolding on student engagement and performance. Smaller, more manageable steps with immediate feedback

on code was beneficial. The immediate feedback served as external feedback. This allowed for class to be run such that not every student needed instructor attention. They could work in a SRL environment that was appropriate for them. Prompts in the JupyterLab provided direction when students were met with confusion, ultimately leading them to ask the right questions. They were able to ask specific questions on specific parts of the script since it was broken up in a way that was digestible. As they were able to understand and learn to ask questions, comprehension increased. Below is an example of this pattern in a conversation between a student and the TA near the end of the semester.

> **Rachel**: "How do I make the robot use the drive action and wall follow action simultaneously?"
> **TA**: "Wall follow includes the drive forward action. You don't need both."
> **Rachel**: "Ok I see. So, if I use the wall follow action, it will follow the wall while also driving forward?"
> **TA**: "Exactly!"

Since the student had a working knowledge of the project, they were able to eloquently articulate their confusion. This was positively proportional to comprehension. As students understood more, they were able to articulate their questions more efficiently.

Another route they were able to take was exploring the troubleshooting pages. There were troubleshooting pages available allowing students to find the problem if one was presented without instructor assistance. The page was written by the professor. This allowed students to debug even if they didn't have debugging experience. Connection issues were prevalent in the class, and this troubleshooting page allowed students to fix the problem even with no experience with wireless connection topics. They could simply run the script, and identify the issue. As students learned that these pages were available and helpful, they increasingly relied on them throughout the semester. Thus, less questions were directed towards the instructors as students had the resource where they could find the answer. This further indicates that they were able to improve their problem-solving skills through SRL. JupyterLab allowed for individuals to work at their own pace, whether it was working through an activity or solving a problem. Although there was adequate external feedback, it seemed that internal feedback triggers were lacking. In the future, it would be important to include more internal feedback such as self-reflection questions. Overall, JupyterLab provided an environment where students were able to learn individually.

B. Collaboration

Another key finding was that more collaboration yielded more apparent comprehension within multiple students. Since each student had access to all the same files, they were able to go to each other for help. The nature of the class activities and projects increased collaboration in the classroom. As collaboration increased, students seemed to have a stronger understanding of

material in the class and topics in the projects. Additionally, the instructors observed increased critical thinking and creativity due to open-endedness of questions. Students could bring multiple perspectives to the projects and generate an inventive idea [14], [15], [16]. As a result, projects were very different between groups. The solutions produced were very diverse even though each team was given the same resources to solve the problem [17]. The software was consistent between teams, and everyone was given access to the same hardware components. Teams changed each week, so students were able to get to know one another. When they got more comfortable with each other, they collaborated more both within groups and as a class. It is important to design projects that foster collaboration and invite all perspectives. That said, there is value in individual work during class time so students can learn at their own pace at the beginning. Once the students have a basis of understanding, they are much more confident working in a group. It was important to have the JupyterLab activities so students could learn the concepts at an individual pace, while having the weekly projects to encourage collaboration and practice their newly acquired skills. Below is dialogue between two students displaying an interaction which yielded learning for both participants.

> **Sally**: "Do you see anything in my code that doesn't look right? I'm not able to pull any data from Airtable.
> **Julia**: "Everything seems good at first glance. Your code looks like it has the same structure as mine."
> **Sally**: "Hm. Maybe my syntax is slightly off in the json parsing string. I heard some other students having issues with that."
> **Julia**: "Oh yeah, it looks like you're missing a slash in the URL string."

This interaction shows how each student brought a unique perspective which ultimately led them both to the solution. Julia initially looked at the code structure as a whole (from a conceptual standpoint) to see if Sally was on the right track. Julia confirmed the code looked right which led Sally to take a closer look and deduce that the lack of connection could be due to invalid syntax. After Sally offered that idea, Julia was able to compare the syntax in her json string with that of Sally's. The student with the issue was able to resolve the problem, and the student helping was able to reflect upon her work more closely. Both mutually benefited from the collaborative interaction.

C. Overall

Students were able to successfully write ROS 2 programs by utilizing both individual and collaborative learning strategies. As previously noted in Table 1, by the end of the semester, students were able to write multiple ROS 2 scripts that ranged from a proportional controller to a robotic joust. Each team successfully fulfilled the project requirements, thereby demonstrating the ability to program using ROS 2. Some teams even went beyond the requirements, showcasing their passion and enthusiasm for the content. However, there were a few factors that

could provide more support in the JupyterLab. First, more structure could improve ease of increasing difficulty. For example, it may be helpful to have folders based on difficulty so students can easily increase or decrease in complexity. Additionally, the structure suggested by Devolder et al. [3] (planning, monitoring, control, and reaction and reflection) could make scaffolding more efficient. Second, more feedback, both internal and external, could improve student performance in a SRL environment [5]. Adding more internal feedback questions would urge students to self-reflect on what they're learning. Overall, students responded positively to the JupyterLab. Even advanced students enjoyed using JupyterLab because they were able to be more independent if desired. Students with little prior knowledge were able to gradually understand how Python and ROS 2 worked and how it interacted with the robot.

There are many practical implications and class recommendations. In CBLEs, it is beneficial to apply scaffolding methods. Well-timed support, external and internal feedback, and black-boxing code are examples of efficient domain-specific scaffolding strategies. There are many lessons learned from applying scaffolding in teaching ROS 2. It eases students into understanding, avoids teaching complex concepts up front, and allows for students to learn by doing instead of by reading. Finally, it gives students autonomy in their learning.

There are effective ways to implement scaffolding. It is important to not overwhelm the student, give options for students to go further, and provide opportunities for students to learn more if they're confused. Using both a robot and learning software that has the ability to build from simple to complex gives flexibility for both the student and instructor.

VI. Future Work

A new proposed JupyterLab is in development. It has an independent section and a project section. The independent section includes scaffolding techniques that help the individual learner gain a basis of technical skills. Some domain-specific techniques include black-boxing code, well-timed support, and a combination of external and internal feedback [1], [3], [5]. It also includes domain-general scaffolding such as self-reflection and concept-mapping. There are four pages in the individual section: planning, monitoring, control, and reflection [3], [8]. Each page implements specific scaffolding techniques related to the title. The project section aims to foster collaboration during projects. Each project has descriptions, tasks, and hints for completion. Additionally, the projects have multiple aspects so it can easily be split up between group members. It is not intended to replicate a full semester's curriculum, but give insight into how an instructor could implement the scaffolding techniques previously discussed.

VII. Conclusion

A combination of domain-general and domain-specific scaffolding in a SRL environment proved to be a successful way to teach advanced computer science topics such as ROS 2. JupyterLab allowed students to learn basic skills at an individual pace which could then be applied to collaborative tasks. Black-boxing of code, immediate feedback, breaking down complex programming tasks into smaller, more manageable steps, and gradually increasing the complexity of programming tasks all helped in improving students' understanding of coding. Collaboration during project-based learning further developed the students' understanding of complex topics. Open-ended problems left room for students to exhibit creativity and passion for the material. Further study may provide insight into how increased comprehension influences effective collaboration on more advanced open-ended problems in the classroom.

VII. References:

[1] L. Zheng, "The effectiveness of self-regulated learning scaffolds on academic performance in computer-based learning environments: a meta-analysis," *Asia Pacific Education Review*, vol. 17, no. 2, pp. 187–202, Apr. 2016, doi: https://doi.org/10.1007/s12564-016-9426-9.

[2] S. Doukakis and M.A. Papalaskari, "Scaffolding Technological Pedagogical Content Knowledge (TPACK) in Computer Science Education through Learning Activity Creation," in *2019 4th SouthEast Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDACECNSM)*, pp. 1–5. doi: https://doi.org/10.1109/SEEDACECNSM.2019.8908467.

[3] A. Devolder, J. van Braak, and J. Tondeur, "Supporting self-regulated learning in computer-based learning environments: systematic review of effects of scaffolding in the domain of science education," *Journal of Computer Assisted Learning*, vol. 28, no. 6, pp. 557–573, Feb. 2012, doi: https://doi.org/10.1111/j.1365-2729.2011.00476.x.

[4] A. Hellas, T. Vikberg, M. Luukkainen, and M. Pärtel, Scaffolding students' learning using test my code. New York, NY: Association for Computing Machinery, 2013, pp. 117–122. doi: https://doi.org/10.1145/2462476.2462501.

[5] C.-Y. Chou and N.-B. Zou, "An analysis of internal and external feedback in self-regulated learning activities mediated by self-regulated learning tools and open learner models," *International Journal of Educational Technology in Higher Education*, vol. 17, no. 1, Dec. 2020, doi: https://doi.org/10.1186/s41239-020-00233-y.

[6] N. Dabbagh and A. Kitsantas, "Using Web-based Pedagogical Tools as Scaffolds for Self-regulated Learning," Instructional Science, vol. 33, no. 5–6, pp. 513–540, Nov. 2005, doi: https://doi.org/10.1007/s11251-005-1278-3.

[7] A. M. Shapiro, "Hypermedia design as learner scaffolding," Educational Technology Research and Development, vol. 56, no. 1, pp. 29–44, Nov. 2007, doi: https://doi.org/10.1007/s11423-007-9063-4.

[8] F. I. Winters, J. A. Greene, and C. M. Costich, "Self-Regulation of Learning within Computer-based Learning Environments: A Critical Analysis," Educational Psychology Review, vol. 20, no. 4, pp. 429–444, Jul. 2008, doi: https://doi.org/10.1007/s10648-008-9080-9.

[9] S. Youn, Y. Chyung, A. Moll, and S. Berg, “The Journal of Effective Teaching an online journal devoted to teaching excellence,” *The Journal of Effective Teaching*, vol. 10, no. 1, pp. 22–37, 2010.

[10] H. W. Lee, K. Y. Lim, and B. L. Grabowski, “Improving self-regulation, learning strategy use, and achievement with metacognitive feedback,” Educational Technology Research and Development, vol. 58, no. 6, pp. 629–648, Feb. 2010, doi: https://doi.org/10.1007/s11423-010-9153-6.

[11] K. J. Crippen and B. L. Earl, “The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy,” Computers & Education, vol. 49, no. 3, pp. 809–821, Nov. 2007, doi: https://doi.org/10.1016/j.compedu.2005.11.018.

[12] R. Isaacson and F. Fujita, “Metacognitive Knowledge Monitoring and Self-Regulated Learning,” Journal of the Scholarship of Teaching and Learning, vol. 6, no. 1, pp. 39–55, 2006, Accessed: Aug. 19, 2023. [Online]. Available: https://scholarworks.iu.edu/journals/index.php/josotl/article/view/1624

[13] T. Lehmann, I. Hähnlein, and D. Ifenthaler, “Cognitive, metacognitive and motivational perspectives on preflection in self-regulated online learning,” Computers in Human Behavior, vol. 32, no. 32, pp. 313–323, Mar. 2014, doi: https://doi.org/10.1016/j.chb.2013.07.051.

[14] Özdemir Göl and A. Nafalski, “Collaborative Learning in Engineering Education *,” in Global J. of Engineering Education, Australia, 2007. Available: https://api.semanticscholar.org/CorpusID:201913032

[15] E. Dringenberg and Ş. Purzer, “Experiences of First-Year Engineering Students Working on Ill-Structured Problems in Teams,” *Journal of Engineering Education*, vol. 107, no. 3, pp. 442–467, Jul. 2018, doi: https://doi.org/10.1002/jee.20220.

[16] Haungs, M., & Clements, J., & Janzen, D. (2008, June), Improving Engineering Education Through Creativity, Collaboration, And Context In A First Year Course Paper presented at 2008 Annual Conference & Exposition, Pittsburgh, Pennsylvania. 10.18260/1-2--3316

[17] S. Willner-Giwerc, K. B. Wendell, C. B. Rogers, E. E. Danahy, and I. Stuopis, “Solution Diversity in Engineering Computing Final Projects,” peer.asee.org, Jun. 22, 2020. https://peer.asee.org/solution-diversity-in-engineering-computing-final-projects (accessed Aug. 19, 2023).