Hardware Integration in Robotics Courses: How to Do It and Why It Matters

Evan Kusa, Duke University

Evan Kusa is a mechanical engineer specializing in robotics, advanced prototyping, and engineering education. As Senior Research and Development Engineer at Duke University, he manages the Garage Lab; the premier design and prototyping facility for the Mechanical Engineering Department. His work focuses on modernizing lab spaces, developing robotics curricula, and fostering cross-disciplinary collaboration. Evan has authored comprehensive lab materials on robotics fundamentals, including ROS2, Linux, and robotic manipulation, and has led major equipment expansions to enhance research and teaching capabilities. His background includes technical roles in advanced manufacturing and aerospace maintenance, and he is passionate about mentoring students in hands-on engineering and design.

Prof. Siobhan Oca, Duke University

Siobhan Rigby Oca is an assistant professor of the practice in the Thomas Lord Department of Mechanical Engineering and Materials Science at Duke University, NC, USA. She received her B.Sc. from Massachusetts Institute of Technology and Master in Translational Medicine from the Universities of California Berkeley and San Francisco. She completed her Ph.D. in Mechanical Engineering in 2022 from Duke University. Her research interests include applied medical robotics, human robot interaction, and robotics education.

Hardware Integration in Introductory Robotics Courses: Building Self-Efficacy Through Structured Labs with Physical Robots

Abstract

This paper reports a three-year curriculum study examining whether hardware integration in an introductory ROS 2 robotics course improves student self-efficacy beyond what structured simulation-based scaffolding alone achieves. The course evolved from unstructured tutorials (2023, $n \approx 40$) to nine structured lab sessions (2024, $n = 16$ paired) to a ten-lab hardware-integrated sequence (2025, $n = 40$). The 2025 curriculum introduced three physical platforms (Kinova Gen3 Lite for manipulation, TurtleBot 4 for navigation, and Crazyflie 2.1 for control systems), deployed through prebuilt Docker container images to eliminate dependency conflicts and reduce environment-setup overhead. The study uses a quasi-experimental historical cohort design; students were not randomly assigned across years, and all between-cohort comparisons are descriptive rather than causal.

Midterm self-efficacy was assessed using task-specific Likert instruments following Carberry et al. (2010). The 2025 cohort reported higher complex project comfort ($M = 3.30$) than the 2024 structured-scaffolding cohort ($M = 3.00$); ROS comfort remained stable across all three cohorts. Degree level (graduate vs. undergraduate) was not associated with self-efficacy on any measure (Mann-Whitney U, all $p > 0.2$). Disciplinary background was the more informative stratification: CS/ECE students reported significantly higher programming comfort than ME students (100% vs. 55.6% positive, $p = 0.015$, Fisher's exact test). Behavioral evidence corroborated survey findings: near-universal completion of an optional hardware midterm extension, final projects that integrated hardware beyond curriculum requirements, and custom Docker container creation that demonstrated infrastructure independence.

Qualitative themes identified a persistent scaffolding calibration problem, with students in the same cohort finding guidance simultaneously too prescriptive and too sparse. This tension appears structural to heterogeneous cohorts rather than addressable through hardware alone. The paper shares reusable assets (versioned container images, ten structured lab packets, survey instruments) and discusses sim-to-real friction, containerization trade-offs, and the practical constraints on hardware access within fixed lab sessions.

I. Introduction

Simulation-only robotics curricula produce confident simulators, not necessarily confident operators. Students who succeed in simulation often encounter a "confidence cliff" at first hardware deployment: algorithms that worked flawlessly fail on physical systems, and students lack the diagnostic vocabulary to understand why. The gap is infrastructural. Network discovery failures, timing mismatches, sensor noise, and calibration drift are problems that simulation cannot teach because it does not produce them.

ROS 2 has become the de facto middleware for robotics development, and educational institutions have introduced ROS 2 curricula at both undergraduate and graduate levels [1, 2, 3]. Yet evidence for how to integrate physical hardware effectively, at what point, with what scaffolding, and for which learning objectives, remains thin. Most curricula treat hardware as a terminal capstone rather than an integrated learning component. For those that treat ROS as a core educational concept, it is typically the entire focus of the course [4].

Prior work [5] demonstrated that structured scaffolding improves self-efficacy trajectories: students in a

lab-based curriculum showed ROS comfort gains from midterm to final (3.38 → 3.63) while students in an unstructured curriculum showed decline (3.39 → 3.29). The current work examines whether hardware integration produces additional benefits beyond structured scaffolding alone.

This paper addresses two research questions:

RQ1: Do targeted hardware labs improve self-efficacy relative to simulation-only instruction?

We hypothesize that physical deployment provides mastery experiences [6] and integration-layer knowledge that simulation cannot.

RQ2: How do we scaffold physical systems work without reducing labs to templates?

Effective hardware integration requires enough structure for success but not so much that students complete tasks without engaging underlying principles.

The study context is an introductory robotics course enrolling 50–60 students per term, primarily mechanical engineering (67.5%) and electrical/computer engineering (25%). The curriculum evolved over three years: unstructured tutorials (2023), structured lab packets (2024), and hardware-integrated instruction (2025). The current ten-lab sequence spans three platforms (Kinova Gen3 Lite for manipulation, TurtleBot 4 for navigation, and Crazyflie for control) with Docker containerization providing reproducible environments.

II. Related Work

A. ROS Education and Scaffolding

ROS-based instruction has proliferated but approaches vary considerably. Wujciak et al. [3] describe scaffolding-intensive instruction using JupyterLab and the iRobot Create 3, distinguishing domain-general scaffolds (goal-setting, self-monitoring) from domain-specific scaffolds (fill-in-the-blank code, black-boxing). Their finding that combining scaffold types produces strongest outcomes informs our lab design. Krūmiņš et al. [2] present a remote web lab using Docker and VNC, addressing setup burden through browser-based access to simulated and physical robots. We adopt similar containerization but with local execution to preserve rapid iteration cycles. Cabrera and Raiti [1] demonstrate that students can learn SLAM and manipulation concepts through Zoom-mediated teleoperation of TurtleBot 3 and Kinova Gen3 Lite platforms; Amsters and Slaets [7] report on the TurtleBot 3 as an educational platform for graduate mobile-robotics instruction more broadly. Our work examines whether similar hardware integration benefits in-person instruction.

B. Self-Efficacy in Engineering Education

Self-efficacy, defined as an individual's judgment of capability to execute courses of action [6], predicts persistence in engineering education. Bandura identifies four sources: mastery experiences, vicarious experiences, verbal persuasion, and physiological states. Hardware deployment provides direct mastery experiences and immediate physiological feedback when robots execute intended behavior. Carberry et al. [8] validated task-specific self-efficacy instruments for engineering design tasks. Following their approach, our survey targets specific competencies: ROS 2 tool comfort, complex project confidence, and perceived lab utility.

C. Scaffolding Calibration

The tension between guidance and discovery in engineering education is well-documented [9, 10]. Scaffolding theory [11] holds that learners require structured support early, with support faded as competence develops. The "scaffolding calibration problem" emerges with heterogeneous cohorts: scaffolding sufficient for novices may reduce experts to template-following; scaffolding for experts may strand novices. Our prior work [5] documented this in robotics instruction: some students reported "copy-paste" completion while others found guidance insufficient. The current work examines whether hardware integration forces deeper engagement by introducing failure modes that templates cannot anticipate.

D. Containerization Trade-offs

Docker-based instruction reduces setup burden by encapsulating dependencies in portable images [2]. Error messages become reproducible; documentation remains accurate; debugging focuses on student code rather than configuration. However, students may complete courses without understanding abstracted dependencies, a concern for ROS, where professional practice demands system-level fluency. Our curriculum attempts balance by making container structure visible: students examine Dockerfiles and modify images for final projects.

III. Course Design and Lab Sequence

A. Design Principles

Four interlocking principles governed the lab curriculum's design.

Scaffolded Skill Progression. The sequence builds competence incrementally, moving from command-line interactions through shell scripting to Python-based ROS 2 node development and finally to integrated autonomous systems. This scaffolded approach, which provides structured support that fades as students gain proficiency, reflects established pedagogical theory for computer-based learning environments [3, 12]. Each phase assumes mastery of prior tools: students writing motion planning code in Lab 5 already understand workspace builds and topic inspection from Labs 1–4; students implementing A* exploration in Lab 10 have internalized the Nav2 goal interface from Lab 9.

Layered Autonomy Architecture. A recurring architectural motif separates high-level strategic decision-making from low-level execution. We refer to this as the "strategist-pilot" pattern. Student-authored nodes function as mission commanders issuing directives (target poses, grasp sequences, exploration goals), while established frameworks (Nav2, MoveIt 2, PID controllers) handle real-time motion planning and control. This decoupling matches industry practice and allows failure analysis at the appropriate level.

Reproducibility Enforcement. Every lab runs inside a Docker container pulled from a centrally maintained registry, eliminating environment inconsistencies that historically consumed substantial debugging time. This containerization approach mirrors best practices for ROS education infrastructure [2]. Version-pinned dependencies ensure that troubleshooting documentation remains accurate across semesters.

Guidance-Autonomy Calibration. Lab packets occupy deliberate middle ground between step-by-step templates and open-ended assignments. Each lab structures work as milestones with explicit checkpoints, but instructions emphasize *what* to achieve rather than dictating implementation. Troubleshooting sections describe symptoms and diagnoses, prompting students to reason about system behavior rather than pattern-match error messages. This addresses the "copy-paste" concern identified in prior iterations [5].

B. Platform Selection Rationale

Three hardware platforms anchor the curriculum, each selected for a specific pedagogical target and logistical profile. This multi-platform approach reflects the recognition that both mobile and manipulator robots serve as valuable pedagogical vehicles [1].

Table 1: Hardware Platform Selection

Platform	Pedagogical Target	Rationale
Kinova Gen3 Lite	Motion planning, gripper control	Collaborative; lower safety overhead
TurtleBot 4	SLAM, Nav2, exploration	Native ROS 2 support; indoor-scale
Crazyflie	PID tuning, control analysis	Low cost (~$200); safe failures

The Kinova Gen3 Lite serves the manipulation sequence. As a collaborative arm operating at lower speeds and forces, it reduces safety infrastructure requirements. The TurtleBot 4 anchors navigation, with native ROS 2 support eliminating driver compatibility issues. The Crazyflie micro-quadrotor addresses control systems with minimal risk: at 29 grams, crashes are pedagogically useful rather than catastrophic.

C. Lab Progression Summary

The ten-lab sequence divides into four thematic phases.

1. Foundational (Labs 1–4). Students configure their development environment (Ubuntu VM, Docker), establish Git workflows, and learn ROS 2 through command-line tools. Lab 3 introduces programmatic control via shell scripts. Lab 4 marks the transition to Python node development with a three-node publisher-relay-subscriber pipeline.

2. Manipulation (Labs 5–6). Using the Kinova Gen3 Lite in simulation, students command MoveIt 2 for motion planning. Lab 5 distinguishes joint-space from Cartesian goals and introduces collision objects. Lab 6 integrates these into a pick-and-place pipeline, with the critical learning target being planning scene synchronization: calling `attach_collision_object` after grasping.

3. Control (Labs 7–8). The Crazyflie platform introduces feedback control. Lab 7 focuses on PID tuning in simulation. Lab 8 bridges to hardware: students receive logged flight data from physical drone runs and write analysis scripts computing RMS error, overshoot, and settling time.

4. Navigation (Labs 9–10). The TurtleBot 4 introduces autonomous navigation. Lab 9 covers SLAM and Nav2 in simulation, progressing to frontier-based exploration. Lab 10 deploys on physical hardware with explicit safety protocols and replaces naive nearest-frontier with A* path-cost analysis. This sim-to-real transition represents a core challenge in robotics education [1].

D. Lab Sequence Evolution

Table 2 summarizes the lab-by-lab progression across the three cohort years. The 2023 cohort completed no structured lab sessions; students worked through ROS tutorials asynchronously without milestone checkpoints or hardware exposure. The 2024 sequence introduced nine structured labs, all simulation-based; graduate students had optional access to a UR5e manipulator for the final project but not during lab sessions. The 2025 revision retained the four foundational labs (with Docker containerization and procedural refinements), replaced the UR5e simulation with the Kinova Gen3 Lite, added two Crazyflie control labs including a required hardware data-analysis session, and expanded navigation to a tenth lab requiring physical TurtleBot 4 deployment.

E. Infrastructure

Container images are maintained in a GitLab registry with hierarchical structure: base images provide core ROS 2 tooling; platform-specific images layer simulation models and framework configurations. Version pinning extends to all transitive dependencies, ensuring error messages match documentation across semesters.

Each lab documents standardized launch sequences. Hardware labs add pre-flight verification (pinging robots, checking TF transforms, inspecting sensor data) before autonomous operation. Common failure modes are catalogued with diagnostic procedures, prompting systematic reasoning rather than blind recovery execution.

Table 2: Lab Sequence by Cohort Year. 2023 had no structured labs; all entries show "—". The HW? column applies to 2025: "Yes" marks the two labs requiring hardware engagement (Lab 8 Crazyflie data from physical flights; Lab 10 TurtleBot 4 deployment). Lab 6 offered optional physical deployment on the Kinova arm within the same session.

Lab	2024 Topic	2025 Topic	Change	HW?
1	Environment setup	Environment setup (Docker)	Containerized	No
2	Shell, Git	Shell, Git (Docker)	Containerized	No
3	Shell scripting	Shell scripting (Docker)	Containerized	No
4	Python ROS 2 nodes	Python ROS 2 nodes (Docker)	Containerized	No
5	UR5e URDF, MoveIt 2 (sim)	Kinova MoveIt 2 (sim)	New platform	No
6	UR5e pick-and-place (sim)	Kinova pick-and-place	Physical optional	Optional
7	TurtleBot 4 SLAM (sim)	Crazyflie PID tuning (sim)	New platform	No
8	TurtleBot 4 nav (sim)	Crazyflie data analysis	Required hardware	Yes
9	A* path planning (sim)	TurtleBot 4 SLAM/Nav2 (sim)	Reordered	No
10	—	TurtleBot 4 navigation	New lab	Yes

IV. Methods

A. Study Design

This study extends prior work examining the evolution of an introductory robotics course at a research university [5]. Teaching ROS-based robotics presents well-documented challenges: the framework requires familiarity with Linux, Python/C++, and publisher-subscriber architectures [1, 3].

Three successive course offerings represent distinct curricular stages:

- **Fall 2023 ($n \approx 40$):** Tutorial-based instruction without structured lab sessions. Students completed ROS tutorials asynchronously with minimal scaffolding and limited hardware exposure.
- **Fall 2024 ($n = 16$ paired):** Introduction of nine structured lab sessions with progressive complexity and milestone checkpoints [5]. Hardware exposure remained limited. Analysis required matched midterm-final responses, yielding 16 paired observations.
- **Fall 2025 ($n = 40$):** Structured lab sequence expanded to ten labs with hardware integration across three platforms and Docker-based containerized environments.

Design and Confounds. This study uses a quasi-experimental historical cohort design: three successive offerings of the same course, each representing a distinct curricular condition. Students were not randomly assigned across years, and cohort-level differences in prior programming exposure, academic preparation, and background are unmeasured confounds. The TA team changed across iterations, introducing TA effectiveness as an uncontrolled variable. Instructor experience increased monotonically from 2023 to 2025, confounding the effect of each curricular change with accumulated pedagogical refinement. The 2024 to 2025 comparison is particularly constrained: hardware integration was implemented simultaneously with Docker containerization and revised lab content, making it impossible to isolate any single factor. These confounds are acknowledged throughout; all between-cohort comparisons are descriptive rather than causal.

The 2023 to 2024 transition demonstrated that structured scaffolding improved ROS comfort trajectories (midterm-to-final: $3.39 \rightarrow 3.29$ decline in 2023 vs. $3.38 \rightarrow 3.63$ gain in 2024) and programming confidence ($4.23 \rightarrow 4.11$ vs. $4.13 \rightarrow 4.69$). The current study examines whether hardware integration is associated with additional gains.

All cohorts completed the same core survey instrument at midterm. The 2025 end-of-semester survey was not administered, limiting analysis to cross-sectional comparison. Data collection was IRB-approved.

B. Instruments

The 2025 cohort ($n = 40$, 91% response rate) included 22 graduate students (55%) and 18 undergraduates (45%). Mechanical engineering students comprised the majority ($n = 27$, 67.5%), followed by electrical/computer engineering and computer science students ($n = 10$, 25%); the remaining 3 respondents (7.5%) were graduate students in other departments (biomedical engineering, $n = 2$; civil and environmental engineering, $n = 1$). Demographics were self-reported and collected only in 2025.

Self-Efficacy Scales. Self-efficacy is defined as an individual's judgment of their capability to execute courses of action for a given task [6], and is a well-established predictor of persistence in engineering education [8]. Adapting the task-specific self-concept approach of Carberry et al. [8], our items used a five-point Likert scale (the original instrument used a 0–100 scale with ten-unit intervals). Three dimensions were assessed: ROS 2 tool confidence, complex project readiness, and perceived lab value.

Open-Ended Prompts. The 2025 survey included six free-response items soliciting feedback on pace, lab structure, and course components. These were analyzed using thematic coding informed by scaffolding theory [3, 12].

C. Artifact Measures

Beyond survey data, the study collected learning artifacts to triangulate self-reported perceptions. Lab completion rates exceeded 95% for all labs. Final project outcomes were assessed qualitatively for hardware utilization, self-directed integration, and infrastructure independence.

D. Replication Details

The 2025 course ran on ROS 2 Jazzy Jalisco on Ubuntu 24.04. Simulation used Gazebo Sim 8 (Harmonic). The `gz_ros2_control` package was built from source at commit `ae0f473b` of the upstream repository; the apt-packaged version contained a segfault in `GazeboSimSystem::initSim` that prevented controller loading. All other ROS 2 packages were installed via apt at their Jazzy-tier versions.

Container Infrastructure. All lab work ran inside Docker containers pulled from the course registry at `gitlab-registry.oit.duke.edu/introtorobotics/mems-robotics-toolkit`. Images follow a hierarchical structure: `osrf/ros:jazzy-desktop-full` serves as the upstream base; a `base-jazzy` layer adds Gazebo Sim, the source-built `gz_ros2_control`, and common development tools; platform-specific layers (`kinova-jazzy-latest`, `turtlebot4-jazzy-latest`, `crazyflie-jazzy-latest`) add hardware drivers and simulation models. Students pull pre-built images without compiling locally. The workflow is: `docker pull` on the host VM, `docker run` with the course workspace mounted as a volume, Terminator launched inside the container, lab package built with `colcon build --symlink-install`, and nodes executed inside the container. Host display forwarding (`xhost +local:docker`, `-e DISPLAY`, `-v /tmp/.X11-unix`) enables Gazebo and RViz. Student compute ran on Linux VMs provisioned through Duke's Virtual Computing Manager (VCM), accessed via FastX.

Platform Connectivity. The Kinova Gen3 Lite connects via wired Ethernet at a static IP (`192.168.1.10`); no onboard compute is present and all ROS 2 nodes run on the host. The `ros2_kortex` driver and `pymoveit2` are installed in the Kinova image layer; the container uses `--net=host` for DDS discovery. The TurtleBot 4 communicates over the lab Wi-Fi network using SUBNET discovery mode with `ROS_DOMAIN_ID=0` and `rmw_fastrtps_cpp`; the Create 3 base and onboard Raspberry Pi 4 serve as robot-side compute. The Crazyflie 2.1 communicates via a Crazyradio PA USB dongle; the container requires `--privileged` for USB passthrough, and `cflib` provides the Python interface. Full lab packets, Docker image recipes, and survey instruments are available through Kusa [13].

E. Limitations

Several limitations constrain interpretation. The 2024 to 2025 comparison confounds hardware integration with Docker containerization and revised content. The missing 2025 post-intervention data precludes within-cohort trajectory analysis; the midterm snapshot captures perceptions after Labs 1–6 but before the hardware-intensive control and navigation phases (Labs 7–10). Students were not randomly assigned across years. Self-efficacy measures capture perceived confidence rather than demonstrated competence; behavioral evidence (midterm extra credit, final project outcomes) partially compensates.

V. Results

A. Quantitative Findings

Table 3 presents mean self-efficacy scores across the three cohorts at midterm. The 2025 snapshot was collected after Labs 1–6 and the Kinova arm midterm deployment but before the required hardware sessions in Labs 8 (Crazyflie data analysis) and 10 (TurtleBot 4 navigation). It therefore captures early-sequence gains and the midterm hardware experience but does not reflect the Crazyflie and TurtleBot 4 hardware phases.

Table 3: Midterm Self-Efficacy Means by Cohort (5-point scale). In 2025, $n = 40$ for ROS comfort, programming comfort, documentation comfort, and complex project comfort; $n = 39$ for the two lab-value items (one respondent did not complete the final block).

Measure	**2023** ($n \approx 40$)	**2024** ($n = 16$)	**2025**
ROS comfort	3.39	3.38	3.48
Programming comfort	4.23	4.13	3.88
Documentation comfort	3.95	3.88	3.58
Complex project comfort	3.08	3.00	3.30
Labs help final project	3.95	4.25	4.18
Labs as reference	4.27	4.38	4.38

Complex project comfort shows the clearest improvement (Figure 1): 2025 students reported $M = 3.30$, compared to $M = 3.00$ (2024) and $M = 3.08$ (2023). This +0.30 gain over the structured-scaffolding-only cohort suggests hardware integration may specifically address the gap between conceptual knowledge and applied confidence, a distinction central to self-efficacy theory [6, 8].

ROS comfort showed consistent improvement across cohorts ($3.39 \rightarrow 3.38 \rightarrow 3.48$). Kusa and Oca [5] reported ROS comfort gains from midterm to final in 2024 ($3.38 \rightarrow 3.63$); whether the 2025 cohort would have shown similar gains cannot be determined without end-of-semester data.

Documentation comfort declined across cohorts ($3.95 \rightarrow 3.88 \rightarrow 3.58$), consistent with the observation in Kusa and Oca [5] that structured labs may not inherently develop autonomous documentation skills.

Perceived lab value remained consistently high. Labs as future reference scored identically in 2024 and 2025 ($M = 4.38$, $n = 39$ in 2025; one respondent skipped this block).

The gap between systems knowledge confidence (82.5% positive, $n = 40$) and complex project confidence (45% positive, $n = 40$) is 37.5 percentage points. Students appear to distinguish between understanding robotics conceptually and feeling prepared to build systems independently. This aligns with self-efficacy theory's emphasis on task-specific confidence [6].

Labs received the strongest learning contribution endorsement: 80.5% rated labs as contributing "A lot" or "A great deal," compared to 78.0% for homework and 51.2% for the midterm project ($n = 41$ on these items;

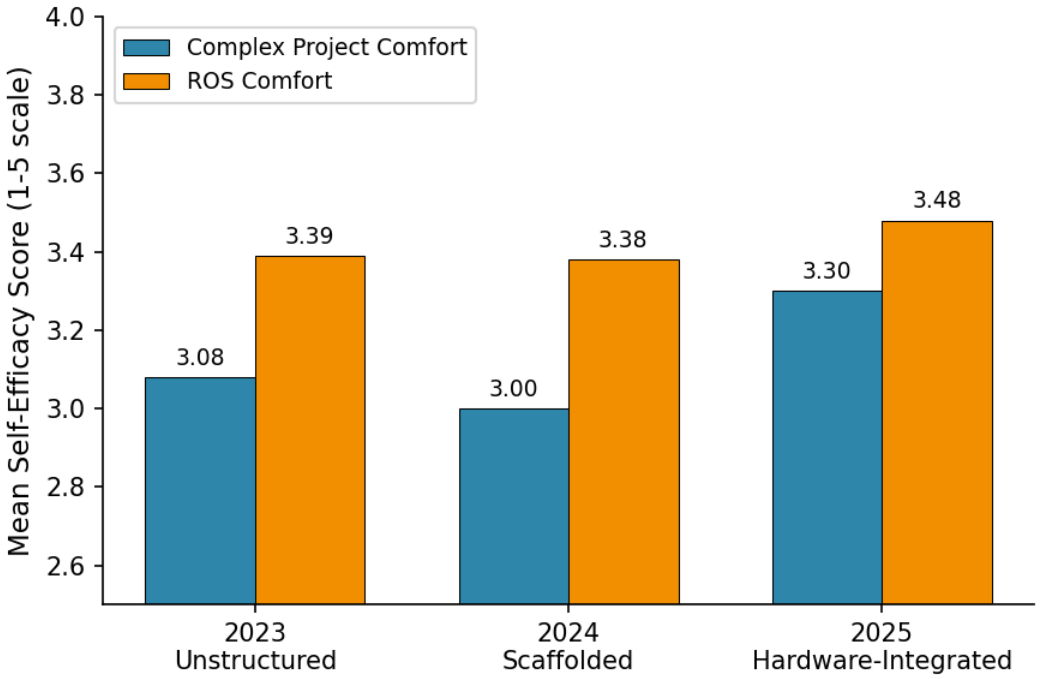

Figure 1: Cross-cohort comparison of self-efficacy measures at midterm. Complex project comfort dipped in 2024 ($M = 3.00$) relative to 2023 ($M = 3.08$), then recovered in 2025 ($M = 3.30$); the 2025 gain over the structured-only cohort is the primary quantitative signal for hardware integration. ROS comfort remained stable across all three cohorts.

one additional respondent completed the contribution ratings but skipped the later self-efficacy items). On prospective utility, 89.7% rated labs as a likely future reference ($n = 39$).

Degree level was not associated with self-efficacy on any measure. Graduate students ($n = 22$) and undergraduates ($n = 18$) reported closely matched midterm means across all six items; no comparison approached significance (Mann-Whitney U, all $p > 0.2$). Undergraduates scored numerically higher on five of six measures, with the largest gaps on labs as reference ($\Delta = 0.32$) and labs help final project ($\Delta = 0.28$); graduate students scored marginally higher only on ROS comfort ($\Delta = 0.06$). This pattern likely reflects the disciplinary composition of each group: the undergraduate cohort contains a higher proportion of CS/ECE students relative to ME students than the graduate cohort does.

CS/ECE majors reported significantly higher programming comfort than ME majors (100% vs. 55.6% positive, $p = 0.015$, Fisher's exact test), suggesting ME students may require additional scaffolding for programming-intensive aspects, consistent with scaffolding theory's emphasis on calibrating support to learner backgrounds [12]. This gap persisted across ROS comfort and complex project confidence (Figure 2). Disciplinary background is a more informative stratification than degree level for understanding self-efficacy variation in this cohort.

B. Qualitative Findings

Thematic analysis of 21 substantive responses identified five primary themes: PACE, SCAFFOLD, SIM2REAL, TRANSFER, and ENGAGE.

ENGAGE: Labs as Learning Vehicle. Students recognized labs as valuable despite frustration: "To be honest, although I hate the lab, it's the most helpful part for me." This pattern aligns with the 82.5% contribution rating and suggests productive struggle rather than wasted effort.

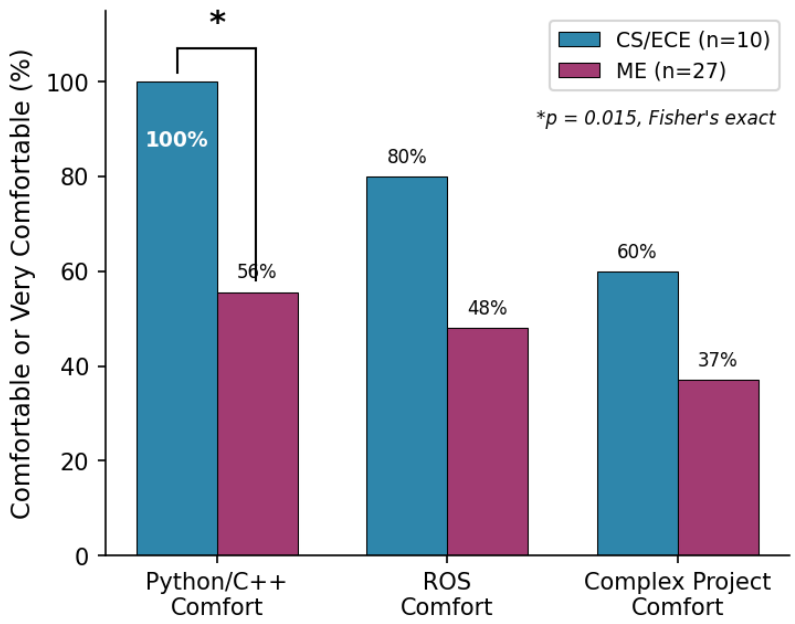

Figure 2: Self-efficacy by major field. CS/ECE students reported higher comfort across all measures, with statistically significant differences in programming comfort.

SIM2REAL: Infrastructure Friction. A consistent concern was time spent on environment issues versus robotics content: "I feel like most of it is figuring out non-robotics-related errors than it is robotics. By the time you get to the actual robotics, lab is halfway done." Containerization eliminated dependency conflicts but introduced its own failure modes [2].

SCAFFOLD: Scaffolding Calibration. Feedback revealed divergent views on guidance level, a challenge documented in scaffolding research [3, 12]. Some found structure supportive: "The lab instructions have gotten much better over time." Others identified over-scaffolding: "Labs utilize a lot of copy-paste structure... that often leaves some of the concepts very shallowly understood." One student suggested progressive fading: "It may be helpful on later labs to start taking away guidance to ensure that terminology and skills have reason to be learned."

TRANSFER: Hardware as Motivator. Students expressed enthusiasm for physical deployment: "Creative freedom and making it work physically." When offered 10% extra credit to implement the midterm task on the physical Kinova arm, almost every student completed the optional challenge. This is behavioral evidence of hardware confidence and intrinsic motivation [1].

PACE: Collaboration and Help-Seeking Patterns. TAs observed organic collaboration: students helped tablemates before seeking TA assistance. However, help-seeking shifted over the semester from conceptual questions to requests for full solutions, a pattern suggesting scaffolding may inadvertently train solution-dependence, with implications for pacing and instructional design.

C. Artifact Analysis

Midterm Hardware Challenge. Near-universal participation in the optional hardware extension provided striking behavioral evidence. Despite being non-compulsory, 38 of 40 students attempted and completed implementation on the physical Kinova arm, suggesting the structured sequence built sufficient confidence for voluntary sim-to-real transfer. TAs corroborated that the hardware midterm "went better than expected."

Lab 6 offered an optional physical deployment track within the same session; roughly 10 of 40 students proceeded to the physical arm. Uptake was limited by the 2.5-hour lab window: students who completed the simulation milestones quickly could proceed to the physical arm, but the coding workload left many without sufficient time. This time constraint is a persistent practical tension: students expressed enthusiasm for hardware ("Creative freedom and making it work physically") while the session structure limited access to it.

Final Project Outcomes. Final projects demonstrated elevated ambition compared to prior years, building on the increased sophistication reported by Kusa and Oca [5] following the 2024 scaffolding introduction.

Hardware utilization exceeded requirements. Undergraduates, not required to use physical hardware, voluntarily established hardware stretch goals. One team coordinated two Kinova arms; another integrated Crazyflie drones.

Self-directed integration. Multiple teams incorporated RGBD cameras not covered in curriculum, transferring beyond taught material.

Infrastructure independence. Most teams created custom Docker containers, modifying instructor-provided Dockerfiles for project-specific dependencies. This represents progression from "container as black box" to authentic developer workflow [2].

VI. Discussion

A. Findings Against Research Questions

RQ1: Do targeted hardware labs improve self-efficacy relative to simulation-only instruction?

Cross-cohort comparison provides partial evidence. The 2025 hardware-integrated cohort reported higher complex project comfort at midterm ($M = 3.30$) than the 2024 structured-scaffolding cohort ($M = 3.00$) or 2023 unstructured cohort ($M = 3.08$). This +0.30 gain suggests hardware integration addresses the conceptual-to-applied confidence gap.

Behavioral evidence corroborates survey findings: near-universal optional hardware challenge completion, final projects integrating hardware beyond requirements (dual-arm coordination, Crazyflie drones, RGBD cameras), and custom container creation that demonstrated infrastructure independence. The persistent gap between systems knowledge (82.5% positive) and project confidence (45% positive) indicates conceptual understanding does not automatically translate to applied confidence, but the 2025 curriculum coincided with a narrower gap than prior approaches.

RQ2: How do we scaffold physical systems work without reducing labs to templates?

The scaffolding calibration problem persists. Student feedback bifurcated between those finding guidance appropriate and those identifying copy-paste completion enabling shallow learning, identical to concerns in 2024 [5]. This suggests the tension is structural to heterogeneous cohorts rather than solvable through hardware alone.

However, behavioral evidence suggests scaffolding built transferable confidence despite the copy-paste concern. Students who may have copied commands during labs nonetheless completed optional hardware challenges, created custom containers, and integrated uncovered hardware. The scaffolding may function as intended even when perceived as template-following. Progressive fading, in which more structured early labs transition to open-ended later labs, is a clear design target for future iterations. TAs reinforced this, observing that students skipped manual text to copy commands directly, and suggested pre-lab quizzes to encourage engagement with conceptual content.

B. Sim-to-Real Challenges Observed

The curriculum documented specific failure modes during hardware deployment: network discovery failures consuming lab time; controller timing differences between simulation and physical actuators; planning scene synchronization errors (forgetting `attach_collision_object`); sensor noise requiring filtering absent in simulation; and environmental variability (dynamic obstacles, lighting). These are integration-layer challenges distinct from algorithmic difficulties, the "confidence cliff" the curriculum aims to address.

C. Containerization Trade-offs

Containerization delivered environment reproducibility: no dependency conflicts, exact failure replication. However, when container or VM issues arose, students lacked system-level knowledge to diagnose problems below the ROS abstraction layer. TAs confirmed this trade-off: containers eliminated setup struggles but coincided with dependency confusion when students began final projects. Final project outcomes suggest this limitation is surmountable, since most teams created custom containers by semester's end, but earlier explicit instruction on container mechanics might ease the transition.

D. Limitations

The missing 2025 end-of-semester data precludes trajectory analysis; self-efficacy changes from PID tuning on physical drones and SLAM on physical TurtleBots are not measured. The 2024 to 2025 comparison confounds hardware integration with infrastructure changes. Students were not randomly assigned, and self-efficacy measures capture perceived rather than demonstrated competence.

VII. Conclusions and Future Work

This paper extends prior work [5] examining curriculum evolution from unstructured tutorials (2023) through structured scaffolding (2024) to hardware-integrated instruction (2025). The ten-lab sequence scaffolds students across three hardware platforms: Kinova Gen3 Lite for manipulation, TurtleBot 4 for navigation, and Crazyflie for control systems.

The primary contribution is evidence that hardware integration at instructional scale is associated with measurable benefits beyond structured scaffolding alone. The 2025 hardware-integrated cohort reported higher complex project comfort at midterm ($M = 3.30$) than the 2024 structured-scaffolding cohort ($M = 3.00$). Behavioral evidence corroborates: near-universal completion of optional hardware challenges, final projects that integrated hardware beyond curriculum requirements, and custom Docker container creation that demonstrated infrastructure independence.

Kusa and Oca [5] demonstrated that structured scaffolding improved self-efficacy trajectories; the current findings suggest hardware integration builds on those gains, coinciding with higher midterm baselines and more ambitious project outcomes. The scaffolding calibration problem persists, with some students finding guidance appropriate and others identifying copy-paste completion. This suggests the problem is structural to heterogeneous cohorts rather than solvable through hardware alone.

The curriculum produced reusable assets: versioned container images, ten structured lab packets, survey instruments, and bringup documentation. Table 4 summarizes the hardware profile; upfront cost is the principal adoption barrier, though the Crazyflie tier is accessible at minimal expense. Full lab packets, Docker images, and rubrics are available through Kusa [13].

Future work should implement consistent pre/post survey administration, develop competence measures complementing self-efficacy, refine scaffolding with progressive fading and embedded conceptual checks, and investigate long-term transfer through alumni follow-up. A persistent practical tension motivates a shift toward Jupyter notebook-based labs: students want hardware access, but coding-intensive labs leave insufficient session time to reach physical deployment. Notebooks reduce implementation overhead and allow

Table 4: Hardware Platform Summary

Platform	Vendor/Model	Onboard Compute	Sensors	Interface	Approx. Cost
Manipulation	Kinova Gen3 Lite	None (host only)	Encoder, F/T	Wired Ethernet	~$13,000
Navigation	TurtleBot 4	RPi 4 + Create 3	LIDAR, IMU, camera	Wi-Fi	~$1,200
Control	Crazyflie 2.1	nRF52840 (onboard)	IMU, barometer	Crazyradio USB	~$200

more students to reach the hardware within the allotted window. Container infrastructure already supports Jupyter (JupyterLab is embedded in the course Docker images); adapting the lab sequence to this format is the immediate next design iteration. The three-year progression demonstrates that hardware-integrated robotics instruction is deliverable at scale, imperfectly and iteratively but feasibly.

References

[1] M. E. Cabrera and J. Raiti, "Physical robots for teaching mobility and manipulation using ROS in remote learning," in *Proceedings of the ASEE Annual Conference & Exposition*, 2024. DOI: 10.18260/1-2--47844. [Online]. Available: https://peer.asee.org/47844.

[2] D. Krūmiņš et al., "Open remote web lab for learning robotics and ROS with physical and simulated robots in an authentic developer environment," *IEEE Transactions on Learning Technologies*, vol. 17, pp. 1313–1326, 2024. DOI: 10.1109/TLT.2024.3381858.

[3] K. L. Wujciak, B. M. Bouchard, and C. B. Rogers, "Scaffolding strategies for teaching ROS 2: An approach using JupyterLab and iRobot education's create 3 robot," in *Proceedings of the ASEE Annual Conference & Exposition*, 2024. DOI: 10.18260/1-2--47954. [Online]. Available: https://peer.asee.org/47954.

[4] S. Farzan, "Project-based learning for robot control theory: A robot operating system (ROS) based approach," *arXiv preprint*, 2023. arXiv: 2305.11279. [Online]. Available: https://arxiv.org/abs/2305.11279.

[5] E. Kusa and S. Oca, "Lessons learned in developing ROS asynchronous tutorials for robotics course: Guided versus inquiry based learning," in *Proceedings of the ASEE Annual Conference & Exposition*, (Montreal, Quebec, Canada, Jun. 2025), 2025. DOI: 10.18260/1-2--56917.

[6] A. Bandura, *Self-Efficacy: The Exercise of Control*. New York, NY: W. H. Freeman, 1997.

[7] R. Amsters and P. Slaets, "TurtleBot 3 as a robotics education platform," in *Robotics in Education: Current Research and Innovations*, ser. Advances in Intelligent Systems and Computing, International Conference on Robotics in Education (RiE), Springer, 2019, pp. 170–181. DOI: 10.1007/978-3-030-26945-6_16.

[8] A. R. Carberry, H.-S. Lee, and M. W. Ohland, "Measuring engineering design self-efficacy," *Journal of Engineering Education*, vol. 99, no. 1, pp. 71–79, 2010. DOI: 10.1002/j.2168-9830.2010.tb01043.x.

[9] P. A. Kirschner, J. Sweller, and R. E. Clark, "Why minimal guidance during instruction does not work," *Educational Psychologist*, vol. 41, no. 2, pp. 75–86, 2006. DOI: 10.1207/s15326985ep4102_1.

[10] A. W. Lazonder and R. Harmsen, "Meta-analysis of inquiry-based learning," *Review of Educational Research*, vol. 86, no. 3, pp. 681–718, 2016. DOI: 10.3102/0034654315627366.

[11] D. Wood, J. S. Bruner, and G. Ross, "The role of tutoring in problem solving," *Journal of Child Psychology and Psychiatry*, vol. 17, no. 2, pp. 89–100, 1976. DOI: 10.1111/j.1469-7610.1976.tb00381.x.

[12] 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, 2016. DOI: 10.1007/s12564-016-9426-9.

[13] E. Kusa. "MEMS intro to robotics: ROS 2 setup guides, lab manuals, and technical references," Duke University Department of Mechanical Engineering and Materials Science, Accessed: Apr. 17, 2026. [Online]. Available: https://mems-intro-to-robotics.github.io/.

A. Lab 6: Pick-and-Place Manipulation (Condensed Walkthrough)

Lab 6 is the culmination of the manipulation sequence and the first lab to offer optional physical hardware deployment. It is representative of the hardware-integration design: students work entirely inside the Kinova Docker container, with an optional section that transitions to the physical arm.

Learning Objectives

Students completing Lab 6 will: (1) implement a multi-step manipulation routine sequencing arm and gripper commands; (2) use approach/retreat poses for straight-line grasps and releases; (3) manage the MoveIt 2 planning scene dynamically by attaching objects on grasp and detaching on release; and (4) modify a working ROS 2 Python node to perform a more complex task (block stacking).

Prelab Reading

Students are expected to have completed Lab 5 (Kinova MoveIt 2 introduction) and to review the canonical pick-and-place flow before the session:

1. **Pick:** approach above object → descend → grasp → `attach_collision_object` → lift.
2. **Place:** approach above target → descend → release → `detach_collision_object` → retreat.

The `attach`/`detach` calls are the key new concept: they update the MoveIt planner's world model so it does not plan around the grasped object.

Procedure Summary

Environment setup. Students pull the Kinova image and start a single container with host networking and the course workspace mounted:

```
REGISTRY=gitlab-registry.oit.duke.edu/introtorobotics/mems-robotics-toolkit
docker pull ${REGISTRY}:kinova-jazzy-latest
xhost +local:docker
docker run --rm -it --net=host -e DISPLAY=$DISPLAY \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  -v ~/workspaces:/workspaces --name ros2_lab06 \
  ${REGISTRY}:kinova-jazzy-latest bash
```

Simulation launch (Pane A). Gazebo with the Gen3 Lite:

```
ros2 launch kortex_bringup kortex_sim_control.launch.py \
  sim_gazebo:=true robot_type:=gen3_lite \
  gripper:=gen3_lite_2f dof:=6 \
  use_sim_time:=true launch_rviz:=false \
  robot_controller:=joint_trajectory_controller
```

MoveIt + RViz (Pane B):

```
ros2 launch kinova_gen3_lite_moveit_config \
  sim.launch.py use_sim_time:=true
```

Object spawning (Pane D). A provided `spawn_blocks.sh` spawns a static table and three 40 mm cubes (red/blue/yellow) into Gazebo, then students mirror them into the planning scene:

```
ros2 run lab06_moveit pick_and_place \
  --ros-args -p task:=add_scene
```

Core task. Students receive a functional starter script that picks and replaces a single block. Milestone 1 requires modifying the place pose to stack block 1 on block 2. Milestone 2 requires implementing a loop-based `task_stack_all()` function that builds a three-block tower.

Canonical Failure Modes

1. **Cartesian fraction too low.** MoveIt cannot find a valid straight-line path for descent or ascent. *Diagnosis:* log shows fraction < 1.0. *Fix:* ensure pre-grasp and grasp poses share the same (x, y); adjust z by 1–2 mm; check that the table collision object is present in the planning scene.
2. **Gripper hangs / action server timeout.** Gripper closes but the node blocks indefinitely. *Diagnosis:* terminal shows "Waiting for action server…" with no timeout. *Fix:* use non-blocking gripper calls; confirm `ros2 action list | grep gripper` returns the expected action name; verify `gen3_lite_2f` gripper type in the launch.
3. **Block not in planning scene after `attach`.** Planner tries to avoid the grasped block and fails. *Diagnosis:* RViz "Attached Objects" panel is empty after grasp. *Fix:* confirm `attach_collision_object` is called immediately after `close_gripper` and before the lift move; verify the object ID string matches the name in the scene.
4. **Block knocked over before pickup.** Robot arm clips the block during approach. *Diagnosis:* visible collision in Gazebo before grasp. *Fix:* increase `approach_height`; verify the approach (x, y) exactly matches the block center; use the `gz service` reset commands to restore block positions without restarting simulation.
5. **Physics jitter / block slip after place.** Block slides or falls after release. *Diagnosis:* block moves after `detach_collision_object`. *Fix:* add a short `time.sleep()` after gripper open to let physics settle; verify place z does not intersect the table surface or prior blocks.

Hardware Deployment (Optional)

Students who complete Milestone 2 within the 2.5-hour session may proceed to the physical arm. The hardware launch replaces the Gazebo bringup with the real driver:

```
ros2 launch kortex_bringup gen3_lite.launch.py \
  robot_ip:=192.168.1.10 gripper:=gen3_lite_2f \
  launch_rviz:=false
```

Students measure physical block positions and update the node's `block_centers` dictionary accordingly. Safety protocol: clear workspace, designate one operator, locate the e-stop, maintain conservative clearance values.

Checkoff Criteria

1. Gazebo screenshot showing block 1 stacked on block 2 (Milestone 1).
2. Gazebo and RViz screenshots showing completed three-block tower (Milestone 2).
3. Written response identifying one failure encountered and the fix applied.
4. (Optional) Screenshot or short video of hardware pick-and-place cycle.