Implementing a Digital Twin for a Robotic Platform to Support Large-Scale Coding Classes

Michael Heeney
Department of Computer Science, Middlesex University, London, UK

Kelly Androutsopoulos
Department of Computer Science, Middlesex University, London, UK

Franco Raimondi
Gran Sasso Science Institute, L'Aquila, Italy

Abstract

Constructionist learning involves learners that are actively engaged in the construction of an entity that reflects the learning achievements. When learning to code, such a physical entity can take the shape of a robot, or of a robotic arm, or any other hardware device that is used to manifest the effect of the code that students are writing. Hardware devices have been used in primary and secondary schools, and also in Higher Education. Unfortunately, the use of hardware devices is limited as it does not scale to large cohorts and requires a physical space for face-to-face teaching.

In this paper we introduce a digital twin for a robotic platform to replicate a classroom setting used for teaching first year undergraduate Computer Science students. We describe the architecture of the system and its implementation.

2012 ACM Subject Classification Applied computing → Interactive learning environments

Keywords and phrases digital twin, introductory programming, constructionism, robotics, computer science education

Digital Object Identifier 10.4230/OASIcs.ICPEC.2024.15

1 Introduction

According to constructionism, learning *"happens especially felicitously in a context where the learner is consciously engaged in constructing a public entity, whether it's a sandcastle or a theory of the universe"* [24]. The case of *learning to code* is a special situation in which:
1) students need to learn a *static* syntax to encode an algorithm to achieve a desired goal, but also
2) students need to learn how to associate a dynamical behaviour to the code they have written, to make sure not only that the code is syntactically correct, but also that it achieves its desired goals. Typically, the dynamical behaviour of code is often hidden and students can only observe its final output, unless they use a debugger.

An approach based on constructionism can provide *physical manifestations* for code that is easier to understand than traces from a debugger, and thus help in providing a better strategy for students to understand the dynamical behaviour (and the consequences) of the code they write, in that students can observe not only the final state of a program, but also all the intermediate steps taken, if these correspond to states of a robot or any other hardware platform. Several approaches have investigated strategies to support constructionism in the setting of learning to code, see for instance [19] and references therein. Approaches based on actual robots have been used in primary schools to support the development of computational thinking and to also to support teachers' confidence [9]. In Higher Education, robots have been used on a range of topics, including to support teaching of functional patterns [6].

Virtual platforms such as Scratch[1] can provide a replacement for physical robots. Unfortunately, these platforms have typically targeted younger users, with very few options (if any) available for the Higher Education sector.

We have developed a *digital twin* for a robotic platform with the explicit goal of providing physical manifestations of code for students in the Higher Education sector. At a high level, a digital twin is the virtual counterpart of a physical system, implementing all its functional properties and reproducing its physical characteristics with high accuracy. Digital twins have been used in a range of domains, including manufacturing, avionics, architecture, etc. One of the key differences between a digital twin and a simulation is the fact that a simulation may remove non-relevant features, for instance a car may be depicted as a rectangle in a 2-dimensional space, while a digital twin would reproduce not only the dynamical behaviour of a car, but also its other aspects, such as the ability to open doors etc.

In this paper we describe the details of a Digital Twin platform that we have developed in Unity[2]. We present the architecture of the system, centred around the extension of an existing approach for service-based development of microcontrollers [4].

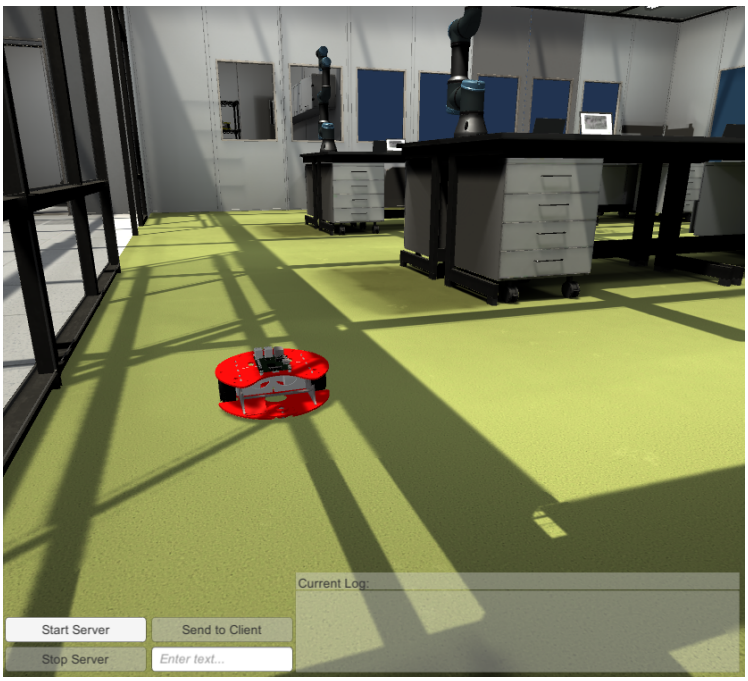

Figure 1 The current prototype with text field for logs, information and sent/received data on the TCP/IP network.

2 Related Work

Constructionism is rooted in Piaget's constructivist approaches [1]. Both constructivism and constructionism have played a key role in the past five decades in STEM education, and advent of increasingly better connectivity has then resulted in the development e-Learning platforms for a range of subjects [27] and across year groups, from primary school

[1] https://scratch.mit.edu/
[2] https://unity.com

to higher education. In parallel, robots have been used to support the teaching of robotics, electronics, manufacturing and STEM subjects in general, building on the success of the maker movement [23].

Robots are used as a platform to teach many concepts in Computer Science. They provide an educational tool for introducing students to embedded systems and computational thinking for interacting with the environment around them. When hardware is introduced, students can conceptualise a topology of the system; from the microcontroller and General Purpose Input/Output (GPIO) functionality, to the software, libraries and interfacing/networking involved to carry out a simple action such as driving forward. Robotics in education naturally puts together continuous and discrete computation and provides an opportunity to reason about error and uncertainty, an important topic of study at an undergraduate level [29].

In regards to computational thinking, other forms of technologies have been developed to help students start to grasp the concepts of programming including augmented reality [21].

Within the task of building a robot, the layers of the system allow researchers and educators to deploy teaching strategies and subject specificity for different target audiences. For example, within the makers movement [23], there have been studies to understand the benefits of robotics to teach programming, electronics, fabrication and general STEM as an overall subject.

When creating a robot for teaching, there is an emphasis whether to buy an existing product, or to create a bespoke product which the educators can iterate over time. Developing a prototype in a maker movement [13] style allows open sourcing of hardware and software, collaboration between staff and students and integration with higher education facilities. One of the main benefits robot platforms following this tradition is the cost. Working robots can be physically built for under £50 with a Raspberry Pi Zero board [34]. However, as described by Correll et al [11], scalability is a factor. For example, designing complex systems becomes a labour of maintenance, explanation of hardware, etc., which can create bottlenecks for delivery.

When developing a simulation part of a platform, linking the virtual environment to this platform is similar to gaming platforms, also used in educational teaching strategies. Games are widely accepted as an engaging and motivating tool in the CS curriculum [18]. There are many advantages including the ability to increase learning interest, enhancing confidence in learning and also lead to long term knowledge retention. It is also noted that gaming environments can produce effectiveness in learners due to their response time. They can generate options to users which may not be available for the learner at the particular time. They are flexible and allow choices without risk, exploration for the user, which stimulates curiosity, discovering learning and perseverance [20].

With the emergence of digital twins, users are able to run counterparts to a physical system. This could be a digital copy of a robotic system, which runs in parallel; this is a common technology in the automotive industry [5]. These models have a plethora of uses in a design life cycle and have many benefits including time saving of ideas, increased quality of work, reduced risks in design and increased efficiency. Links have also been made to how digital twins serve well as an educational tool, bringing together many aspects of different learning theories [12]. Additionally, the manifestation of the understanding of how individual parts of a system, including sensors and actuators work, is beneficial to students [8].

A tangible artefact, such as a robot gives the user embodiment, allowing them to express their understanding of code through the robot e.g. movement, sensing the environment etc. Embodiment in robotic hardware has been seen to have positive benefits over an array of year groups [17, 22, 33]. Popular beneficial themes include engagement of material, self-efficacy,

attitudes towards the subject and improvement of grades. These benefits are continued between different forms of platform, for example, from bespoke and custom hardware [10], to commercial robots or for virtual environments and simulations [3, 30].

Physical feedback obtained by using mobile robots has the ability to strengthen the concepts of programming and motivate the students. On one hand, students are able to become motivated by the technology and at the same time, have a better relationship between the notion of theory and how that relates to practical results [31]. By creating a realistic counterpart, the idea is to keep the irreplaceable position in the educational process [28], but also keep the realism of the physical feedback obtained with real hardware and show the similarities in class.

It has been reported [14] that experiences with physical manifestations and existing projects can bring students to be motivated into learning and be interested in the subject. Furthermore, students developed ideas further than originally set, setting additional goals once understanding the system and technology further.

When developing a robotic platform, or its digital counterpart, the engagement of students will allow them to spend more time developing programs and working on solutions. With repetition of exercises, seeing changes and working on a solution in iterations, it is has been suggested [15] that students build better solutions overall.

MIRTO [2] is an example of a cheap, open-source, robotic platform used in face-to-face teaching. In this paper, the learning objectives were achieved much earlier than in traditional and theoretical models of teaching. This tool is influenced by the constructionism approach, and it allows the students to explore the capabilities of the robot independently. For example, before working towards the marking objectives of the module, introductory activities would include to understand wheel rotation and navigation. This includes defining the speeds of the motors, alignment and bump sensors. These tasks allow students to understand by trial and error the notion of real-time system and control, reinforcing knowledge and skills from previous work. Gamification in the classroom by means of line-following races led to further engagement [6].

In 2020, with academic institutions going into lockdown due to the COVID-19 virus, the ability to create virtual labs for simulating practical skills [25] was essential to continue teaching practical skills in classes. Digital twins allow this to happen by creating a reliable and high fidelity prototype. However, with the development of this kind of systems, the scalability for experimentation should increase and provide feedback as close as possible to the original physical systems [32].

Existing literature shows the benefits of mobile robotics in Computer Science education. The robots can be virtual or tangible. However, most of the case studies are up to keystage 5 in the United Kingdom (16-18 year olds). To the best of our knowledge, there does not seem to be an undergraduate Computer Science course using a digital twin to support teaching programming to these students.

Testing the principles of constructionist approaches and how to measure the effect of the learning methodology has been subject to various studies. Kafai [16] created a comparative study, in which the classroom was split between a constructionist approach using a creation of video games which incorporated mathematical problems requiring an understanding of fractions to solve; with another group received traditional teaching for fraction instruction. This study concluded a better understanding of fractions, increased engagement/motivation of the students, better critical thinking and a development of soft skills through collaborative learning. Benefits of Scratch as a constructionist tool have been widely shown, particularly by Resnick and Brennan [7] in which results concluded positive improvements in the ability to develop an iterative design cycle with experimenting and iteration of code and objects, logical reasoning through testing and debugging, creativity and problem solving.

3 The MIRTO Robotic Platform

The MIRTO robotic platform was developed to support a specific pedagogy. We take a constructionist approach to teaching the fundamentals to first year undergraduate Computer Science students with MIRTO. Hardware is demonstrated using digital electronic circuits to discuss system architecture, microcontrollers and microprocessors such as Raspberry Pi. Programming concepts are taught with Racket (a dialect of Lisp). In the last third of the academic year, the fundamentals of these topics are brought together using a robot with the acronym MIRTO.

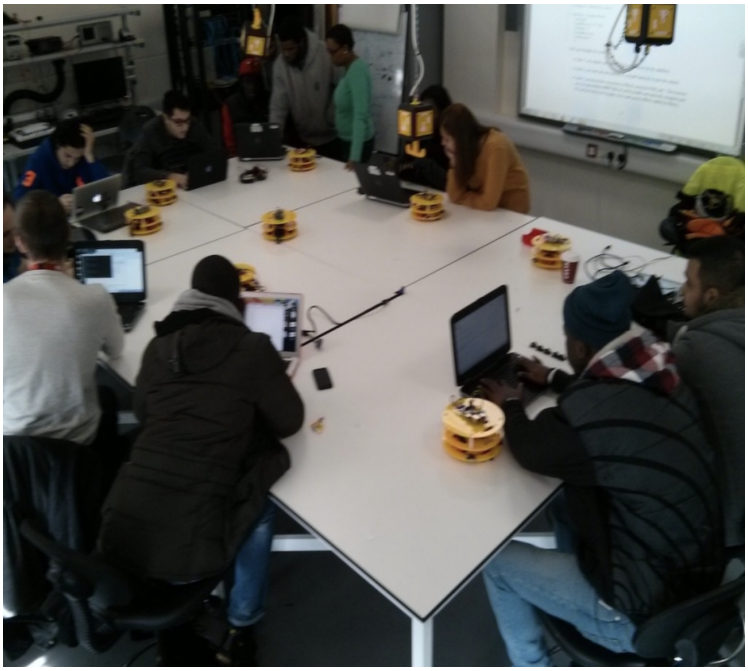

Figure 2 Students programming robots in a classroom setting.

MIRTO is an open source robotic platform that contains the following components:

1. Raspberry Pi: A custom image of Raspbian extended with Racket to compile code on the Pi. This also enables networking opportunities such as custom wifi networks, editing of a Linux image, ssh, ethernet etc. This can be networked with an Arduino microcontroller to control GPIO or these two boards can be swapped with a Raspberry Pi Pico.
2. Custom PCB: The top-layer of the robot has all of the GPIO options of the robot available in one place including motorshield for controlling 2 x DC motors, an RGB LED, an infrared array for line following, 2 tilt switches for hit detection, a speaker and a LCD for displaying messages and connection information.
3. ASIP: Arduino Service Interface Protocol[3] enables a computer to discover, configure, read, write a microcontrollers general purpose IO pins. As standard, ASIP uses a serial connection to the Raspberry Pi.
4. Along with ASIP, there are various libraries to work with programming languages including Java and Python. For first year students the focus is on the Racket[4] library.

[3] https://github.com/michaelmargolis/asip/
[4] https://github.com/fraimondi/racket-asip

To support iteration of teaching material, anonymous questionnaires are delivered to students, in which many reported enjoyment using MIRTO and use of hardware across the year. However, in order to scale the use of robots for large classes and for online teaching, we developed a digital twin to serve as a real-time counterpart of the physical robots in class. Other reasons for designing a digital twin include:
1. Sustainability of manufacturing: The robots take time to design, build and repair, which could be labour-intensive for large cohorts.
2. Time management: Students only have a 2-hour class per week to work on the robots. The demand was very high outside the classroom for students to test, but due to the last point, we were unable to loan these with fairness.
3. Training: The robots need a range of expertise across various disciplines to operate and maintain and in scale, this was proved difficult for staff availability.

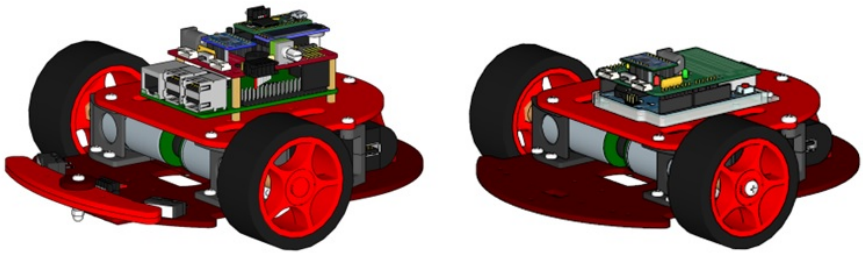

Figure 3 The latest iteration of MIRTO 2024.

3.1 Software for MIRTO

Students can complete exercises independently or in groups on software problems that involve MIRTO. They are given a selection of exercises to try various GPIO of the robot and then given examples of code to run and understand what it is doing. The exercises consist of moving the robot in a room at different speeds and for a certain time (understanding the timing issues) avoiding obstacles and bumping into walls. More advanced exercises involve following a line and improving on precision. Examples of instructions are given in Table 1.

Table 1 Instructions for the MIRTO using DrRacket.

Instruction	Description
(setMotor 0 0)	*Set the speed for the motors. The first integer assigns the left (0) or right motor (1). The second integer assigns the speed of the motor. Value range is from -255 to 255.*
(stopMotor 0)	*This will stop a single motor. The integer assigns the motor as above.*
(stopMotors)	*Stops both the motors.*
(getIR 0)	*Prints the reflected value from a infrared sensor array. The integer is the sensor number in the array. Currently there are 3 sensors. Value range is from 0 - 255.*
(setLCDMessages “message” 0)	*Write a message to a five line liquid crystal display. The text in “” is what will be printed. The integer is the line on the display.*
(leftBump?)	This will check the state of a bump sensor, printed as a 0 or 1 for true or false. There are two sensors on the robot, leftBump and rightBump.
(analog-read 0)	Reads the on board potentiometer on the robots PCB. This could be any analog sensor which is assigned by the integer.
(playTone 0 0)	Plays a tone on a piezo buzzer based on a particular frequency. The first integer assigns the frequency. The second assigns the duration in seconds.

With the majority of the exercises, students are given the freedom to explore a snippet of code and guess what it will do. Then when they compile the program, they can see how the robot will react. With the exploration of the wheels, sensors and other input/output of the robot, students embed themselves in a creative learning process. This supports reflection in class and discussion between peers. During observations, an array of interpretations of solutions for the same task are created using sensors to determine movement, outputs of LCD messages or colour of LEDs to show a state or explanation of location etc. With this notion of creative learning intertwined with the technology, students are able to follow a creative learning spiral [26]. Examples of the concepts taught using MIRTO include higher-order functions, string processing and open vs closed loop systems [6].

4 Implementation of Digital Twin

We have implemented a digital twin in Unity, to replicate a classroom setting for students. Bespoke classroom objects such as the mobile robot, robotic arms and some furniture were all designed in Blender.

4.1 High-Level System Architecture

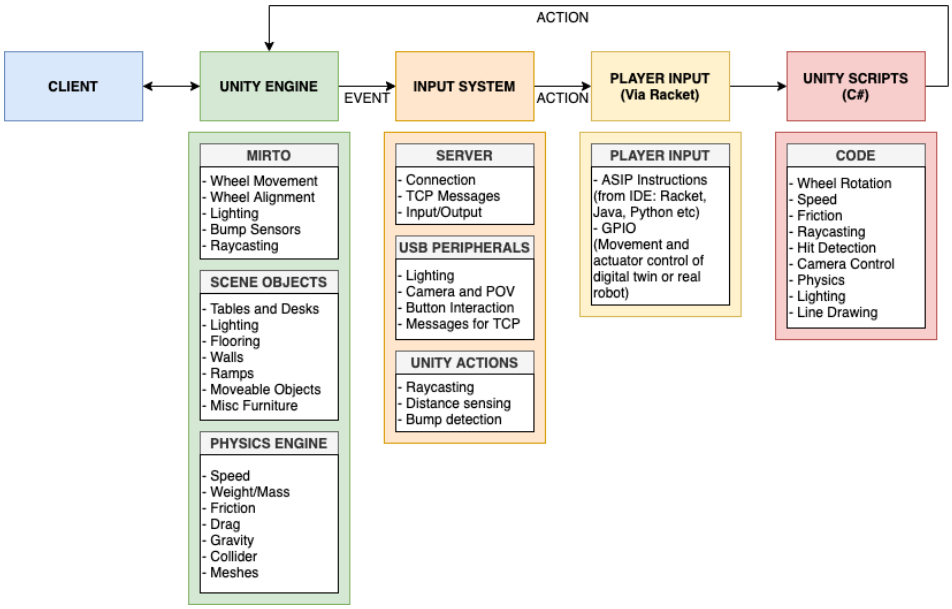

Figure 4 Digital Twin Architecture for the Online Provision.

The digital twin created in Unity replaces the robot described above and the overall architecture of the system is illustrated in Figure 4. At the high level, a student will control the client of the system. This is typically a program written in Racket. When run, the Racket program will establish a TCP connection with Unity and begin to translate Racket messages into instructions. These instructions are then converted to C# scripts which will control many aspects of the scene of the digital twin including the orientation and direction of the robot, the speed and movement of the robot, control and reading of the sensors of the robot (bump sensors, infrared sensors, LEDs etc). In turn, in some of these instances, Unity will send back string messages to Racket, to let the program know which sensors have

been interacted with, looking for a follow up instruction if needed. For example, students are taught different ways to navigate with the robot. This could include driving, using bump sensors to turn if there are objects in the way, line following etc.

In Unity, the digital twin is created on the same part list used for the physical manufacturing of the robot. Therefore, the parts are the same scale and the scene replicates the classroom setting. The scene is created using a range of furniture, walls and lighting. All these components contain physics properties and meshes so the robot (and user) is able to have a realistic interaction in the environment.

The digital twin incorporates wheel movement by operating scripts to control wheel colliders, which are a slip-based tyre friction model generated for wheels. Along with the collider, there is a rotation animation, to give realism and understanding of speed for the user.

Figure 5 The digital twin has 3 rooms for students to explore and run their programs in.

For the sensors of the robot, Unity uses raycasting for the bump sensors. With raycasting, the bump sensors on the 3D model send out a "ray" from a camera point until it finds a surface it collides with. The ray sent is very short, to mimic being close enough to the object to "hitting" it. Once hit, a boolean value is triggered, sending a message back to the Racket client. Similarly, with the infrared sensor, 3 raycasts are sent (as separate threads) to the floor to detect colour in the flooring, this is to mimic lines on the surface for students to complete line following exercises.

The environment uses an input system with various methods. For example, as described above, Racket will send messages via a TCP server to control the robot; Unity can also output messages back to Racket. USB peripherals can also control the movement, alignment and camera position of the user in the scene. Additionally, Unity actions such as the raycasting can provide input to control the environment. Racket is also the main input for the environment, controlling the digital twin and its movement. All of these methods of inputs will generate the scripts written in C to control the speed, direction and friction of the wheels, line drawing (to help with navigation) and sensor and input/output control.

4.2 Implementation

Let's consider an example to see how the digital twin works in practice. Assume that a student needs to write code for the following task: "drive the robot forward for two seconds then stop"; the Racket code would be the following:

```
1 #lang racket
2 (require "AsipMain.rkt")
3   (open-asip)
4   (setMotors 150 150)
5   (sleep 2)
6   (stopMotors)
7   (close-asip)
```

This is identical for a physical robot or for the digital twin. In the case of the digital twin, instead of these commands being sent to the serial port of the robot, ASIP[5] will send a string to the TCP server running in Unity. The above code does the following:
1: Include the AsipMain Racket library
3: We setup the connection to the robot. This will connect to the localhost at port 54010 for a digital twin.
5: The robot is asked to move both wheels forward. The power selected is 150 (the range is between -255 to 255).
6: There is a sleep for 2 seconds, which allows the wheels to run during this time.
7: The wheels are turned off, the speed is set to 0.
8: Close the TCP connection

For the example above, the actual strings sent over TCP would be in two parts, to turn the motors on and then off again. Concretely, the following messages will be sent:

```
"M,m,0,150";"M,m,1,150";"M,m,0,0";"M,m,1,0"
```

When a motor command is sent, it is translated to two commands back to control each motor separately. This allows us to also set individual motor movement for turning and concise movements.

In Unity, once the string is received, it is split into four different variables to understand its role in ASIP and what it should be controlling. Each string starts with a letter, which is the header of the request; this could include M (motors), E (encoders), T (tone – for a piezo), P (RGB LED control), L (LCD on the robot), I (port to pin mapping) etc. Each of these strings has separators (,) and other tags and numeric values that go with it. Once the command above is identified as a motor command, wheel colliders in Unity that control the wheels will move using the following command:

[5] https://github.com/michaelmargolis/asip/tree/master/documents

```
leftWheelInput.motorTorque = asipLeftWheelInput * motorForce;
rightWheelInput.motorTorque = asipRightWheelInput * motorForce;
```

Within the movement calculations, there are parameters set to increase the wheel speed in the virtual space, similar to the classroom setting. For example, 0 being the no movement, 255 being the fastest speed and -255 reversing at the maximum speed. At a later date, there will be planned tests to ensure accurate comparisons between the virtual and physical spaces.

For reference of the user, in the Unity GUI, there is a display box to show all of the messages received from ASIP, as well as messages sent and error messages etc. This feature can be shown/hidden by the user as well as the ability to draw a line showing the movement path of the digital twin to help students understand path planning and movement instructions further, similar to designs of LOGO and turtle robots in the 1980s [24].

5 Conclusion and future work

We have presented the development of a multi-programming language digital twin platform to replicate a classroom setting used for teaching Computer Science to undergraduate students. The digital twin is planned to be released as an open-source platform. The digital twin can run side-by-side with its physical counterpart. Students are able to use the physical model in classes and once resources are not available, for example out of class, students can login to a virtual world to continue their programming assignments and continue developing their computational thinking. The online platform and physical platform both use Racket (a dialect of Lisp) as its main programming language. However, our virtual environment can be used by students across year groups and can be programmed in multiple languages including Java, Python, as well as Racket.

While the use of the digital twin in classrooms has shown to be promising, we plan to conduct further experiments with students to empirically evaluate its impact. Current surveys suggest there is a strong link to engagement and computational thinking when using the digital twin. However, more work is to be carried out to fortify these claims.

For future work, more work is needed for creating an open source and robust digital twin for all to host, deploy and use for their teaching and training. Emphasis will be made on the GUI, to deploy visual indicators on all sensors and actuators, so students and staff can get visual feedback of the tools they are using, rather then printed syntax and text. Within the GUI, students will also be able to select a lab space, to best replicate the room they are in for size. Additional to this, menus will be designed to enable students to select the connection type; this could include web socket connections, TCP/IP as stated previously or how they want to connect, via simulation only or as a digital twin.

References

1 Edith Ackermann. Piaget's constructivism, papert's constructionism: What's the difference? In *Constructivism: Uses and perspectives in education.*, pages 85–94, 2001.

2 Kelly Androutsopoulos, Leonidas Aristodemou, Jaap Boender, Michele Bottone, Edward Currie, Inas El-Aroussi, Bob Fields, Lorenzo Gheri, Nikos Gorogiannis, Michael Heeney, et al. Mirto: an open-source robotic platform for education. In *Proceedings of the 3rd European Conference of Software Engineering Education*, pages 55–62, 2018.

3 Mikko Apiola, Matti Lattu, and Tomi A Pasanen. Creativity and intrinsic motivation in computer science education: experimenting with robots. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 199–203, 2010. doi:10.1145/1822090.1822147.

4 Gianluca Barbon, Michael Margolis, Filippo Palumbo, Franco Raimondi, and Nick Weldin. Taking arduino to the internet of things: The asip programming model. *Computer Communications*, 89:128–140, 2016. doi:10.1016/j.comcom.2016.03.016.

5 Florian Biesinger and Michael Weyrich. The facets of digital twins in production and the automotive industry. In *2019 23rd international conference on mechatronics technology (ICMT)*, pages 1–6. IEEE, 2019.

6 Jaap Boender, Ed Currie, Martin Loomes, Franco Raimondi, and Giuseppe Primiero. Teaching functional patterns through robotic applications. In *Trends in Functional Programming in Education 2015*, June 2015.

7 Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, page 25, 2012.

8 Rolando Antonio Chacón Flores, Martí Sánchez Juny, Esther Real Saladrigas, Xavier Gironella Cobos, Jaume Puigagut Juárez, and Alberto Ledesma Villalba. Digital twins in civil and environmental engineering classrooms. In *EUCEET 2018: 4th International Conference on Civil Engineering Education: Challenges for the Third Millennium*, pages 1–10. International Centre for Numerical Methods in Engineering (CIMNE), 2018.

9 Christina Chalmers. Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction*, 17, July 2018. doi:10.1016/j.ijcci.2018.06.005.

10 CTE STEM Coach, David Archer, Lois Delcambre, Scott Britell, and Ananth Mohan-Contributors. K-12 stem robotics: Stealth computer science for the masses. *The Journal of Computing Sciences in Colleges*, page 152, 2011.

11 Nikolaus Correll, Rowan Wing, and David Coleman. A one-year introductory robotics curriculum for computer science upperclassmen. *IEEE Transactions on Education*, 56(1):54–60, 2012. doi:10.1109/TE.2012.2220774.

12 Joe David, Andrei Lobov, and Minna Lanz. Learning experiences involving digital twins. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 3681–3686. IEEE, 2018. doi:10.1109/IECON.2018.8591460.

13 Erica Rosenfeld Halverson and Kimberly Sheridan. The maker movement in education. *Harvard educational review*, 84(4):495–504, 2014.

14 Cindy K Harnett, Thomas R Tretter, and Stephanie B Philipp. Hackerspaces and engineering education. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–8. IEEE, 2014.

15 Michael Jonas. Do it again: Learning complex coding through repetition. In *Proceedings of the 18th Annual Conference on Information Technology Education*, pages 121–125, 2017. doi:10.1145/3125659.3125690.

16 Yasmin B Kafai. *Minds in play: Computer game design as a context for children's learning*. Routledge, 2012.

17 Christopher Kitts and Neil Quinn. An interdisciplinary field robotics program for undergraduate computer science and engineering education. *Journal on Educational Resources in Computing (JERIC)*, 4(2):3–es, 2004.

18 Stan Kurkovsky. Making computing attractive for non-majors: a course design. *Journal of Computing Sciences in Colleges*, 22(3):90–97, 2007.

19 Michael Lodi, Dario Malchiodi, Mattia Monga, Anna Morpurgo, and Bernadette Spieler. Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey. *Olympiads in Informatics: An International Journal*, 13:99–121, July 2019. doi:10.15388/ioi.2019.07.

20 Ju Long. Just for fun: using programming games in software programming training and education. *Journal of Information Technology Education: Research*, 6(1):279–290, 2007.

21 Luis Carlos Martins, Lázaro Vinicius Lima, and Pedro Rangel Henriques. Lcsmar, an ar based tool to inspect imperative programs. In *4th International Computer Programming Education Conference (ICPEC 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

22 Monica M McGill. Learning to program with personal robots: Influences on student motivation. *ACM Transactions on Computing Education (TOCE)*, 12(1):1–32, 2012. doi:10.1145/2133797.2133801.

23 Sofia Papavlasopoulou, Michail N. Giannakos, and Letizia Jaccheri. Empirical studies on the maker movement, a promising approach to learning: A literature review. *Entertainment Computing*, 18(C):57–78, 2017. doi:10.1016/j.entcom.2016.09.002.

24 Seymour Papert and Idit Harel. Situating constructionism. In Seymour Papert and Idit Harel, editors, *Constructionism*, chapter 1. Ablex Publishing Corporation, Norwood, NJ, 1991. URL: http://www.papert.org/articles/SituatingConstructionism.html.

25 Sandipan Ray and Sanjeeva Srivastava. Virtualization of science education: a lesson from the covid-19 pandemic. *Journal of proteins and proteomics*, 11:77–80, 2020.

26 Mitchel Resnick. Lifelong kindergarten: Cultivating creativity through projects, passion, peers and play. In *"Lifelong Kindergarten: Cultivating Creativity through Projects, Passion, Peers and Play"*, chapter 1. The MIT Press, Cambridge, Massachusetts, 2017.

27 T. Richter, S. Rudlof, B. Adjibadji, H. Bernlöhr, C. Grüninger, C.-D. Munz, A. Stock, C. Rohde, and R. Helmig. Viplab: a virtual programming laboratory for mathematics and engineering. *Interactive Technology and Smart Education*, 9:246–262, 2012. doi:10.1109/ISM.2011.95.

28 Carlos Rodriguez, Jose L Guzman, Manuel Berenguel, and Sebastian Dormido. Teaching real-time programming using mobile robots. *IFAC-PapersOnLine*, 49(6):10–15, 2016.

29 Daniela Rus. Teaching robotics everywhere. *IEEE Robotics & Automation Magazine*, 13(1):15–94, 2006. doi:10.1109/MRA.2006.1598048.

30 Ashraf Saad, Travis Shuff, Gabriel Loewen, and Kyle Burton. Supporting undergraduate computer science education using educational robots. In *Proceedings of the 50th Annual Southeast Regional Conference*, pages 343–344, 2012. doi:10.1145/2184512.2184596.

31 Payman Shakouri, Olga Duran, Andrzej Ordys, and Gordana Collier. Teaching fuzzy logic control based on a robotic implementation. *IFAC Proceedings Volumes*, 46(17):192–197, 2013. doi:10.3182/20130828-3-UK-2039.00047.

32 Jaroslav Sobota, Roman PiŜl, Pavel Balda, and MiloŜ Schlegel. Raspberry pi and arduino boards in control education. *IFAC Proceedings Volumes*, 46(17):7–12, 2013.

33 Eben B Witherspoon, Ross M Higashi, Christian D Schunn, Emily C Baehr, and Robin Shoop. Developing computational thinking through a virtual robotics programming curriculum. *ACM Transactions on Computing Education (TOCE)*, 18(1):1–20, 2017. doi:10.1145/3104982.

34 Narasimha Saii Yamanoor and Srihari Yamanoor. High quality, low cost education with the raspberry pi. In *2017 IEEE Global Humanitarian Technology Conference (GHTC)*, pages 1–5. IEEE, 2017. doi:10.1109/GHTC.2017.8239274.