HoRoSim, a Holistic Robot Simulator: Arduino Code, Electronic Circuits and Physics

Andres Faiña (✉)

Robotics, Evolution and Art Lab (REAL),
IT University of Copenhagen,
Copenhagen S 2300, Denmark
anfv@itu.dk

Abstract. Online teaching, which has been the only way to keep teaching during the pandemic, imposes severe restrictions on hand-on robotic courses. Robot simulators can help to reduce the impact, but most of them use high-level commands to control the robots. In this paper, a new holistic simulator, HoRoSim, is presented. The new simulator uses abstractions of electronic circuits, Arduino code and a robot simulator with a physics engine to simulate microcontroller-based robots. To the best of our knowledge, it is the only tool that simulates Arduino code and multi-body physics. In order to illustrate its possibilities, two use cases are analysed: a line-following robot, which was used by our students to finish their mandatory activities, and a PID controller testbed. Preliminary tests with master's students indicate that HoRoSim can help to increase student engagement during online courses.

Keywords: Robot simulator, Arduino, Electronics, Physics, Educational tools

1 Introduction

There are a lot of robots controlled by microcontrollers that interact with physical objects, such as line-following robots or walking robots. In these robots, sensors are used to transform physical properties into electrical signals, which are then processed by the microcontroller. Based on these inputs, the microcontroller produces electric signals as outputs, which are then processed by an electric circuit to power actuators. Finally, these actuators produce some effect in the environment (turning a motor for example) and the sensors generate new signals. Thus, teaching these robotic first principles involves a wide curricula that includes mechanics, electronics and programming.

The main aim of teaching these basic topics in a robotics course is to demystify these areas, show the big picture to the students and force them to understand that robotics is not only computer science. At IT University of Copenhagen, we use this holistic approach in the first course of the robotics specialization. The course is called "How to Make (Almost) Anything"[1] and is

[1]Inspired by the course of the same name taught at MIT by Neil Gershenfeld.

targeted to Computer Science students (21 to 30 years of age) without previous knowledge in robotics, mechanics or electronics. The approach is comparable to the famous "NAND to tetris" course [1] for teaching computer science, where computer science abstractions are made up concrete through detailed implementations. In our course, each lecture focuses on a specific technique or topic which allows students to build some components of a custom line following robot, which is completely functional after 7 weeks. The approach used was not to simulate anything and work directly in hardware. This strategy worked relatively well until March 2020, when the pandemic forced a lock down of the university. This caused that half of the students could not get access to their robots while still having pending assignments to hand in. This suddenly created the need for a simulator.

Simulators have several drawbacks. First, they are not accurate, which provokes a reality gap between the simulation and the real behaviour. Second, they introduce limitations as not all the aspects of the robots are taken into account and not all the components are available to be simulated. And finally, the manufacturing of physical prototypes cannot be recreated. To avoid the limitations, there are several robots designed to be used in education [2,3]. However, simulators have some advantages over working in hardware: (1) They provide a quick way to prototype and check that everything works as expected, (2) they do not require any hardware to learn (apart from a computer), which is especially relevant for online learning, (3) they reduce the running costs of the course and (4) students lose their fear to break the hardware which allow them to experiment more freely.

While there are a lot of simulators that allow us to simulate robots, there is a lack of a holistic simulator. Most of the times, teaching robotics involves the use of several simulators, as there is no one that covers all the topics (physics, electronics, and embedded programming). As an example, when using robot simulators, the electronic aspects are usually ignored and students do not need to think in the electronic hardware that controls the actuators, the type of actuators of the robot or the low-level controllers used. Other simulators are focused on electronics and microcontrollers but are unable to simulate multi-body physics.

This paper introduces a new holistic simulator that combines three fundamental aspects of robotics: physics, electronics, and embedded programming. The main advantage is that it forces students to understand the basic principles of robotics before moving to more advanced topics. The simulator, called HoRoSim, consists of a robot simulator that simulates multi-body physics (CoppeliaSim), custom libraries for the Arduino code and the logic to simulate basic electronic circuits.

2 Related Work

There are several simulators that can be used to simulate some aspects of robots. Physics engines such as ODE [4] or Bullet [5] can simulate multi-body dynamics and use collision detection algorithms to calculate the distance from one sensor to a body. However, the input of these physics engines requires to specify the torques or forces for each joint and the output is just the position of the bodies in the system for the next time step. Robot simulators such as Gazebo [6], CoppeliaSim (before known as V-Rep) [7] or Webots [8] can facilitate the use of the physics engines as they provide a graphical user interface where the robot and physical bodies are displayed, a set of controllers for the joints (move joint with constant speed, etc.), sensors (cameras, distance sensors, etc.), and predefined models of robots and common objects. Nevertheless, they ignore the fact that these actuators and sensors are controlled by some electronic circuit.

The most popular electronic simulator is SPICE [9] and its different forks such as LTspice [10] and Ngspice [11]. These simulators take the definition of an electronic circuit through a netlist and can perform different types of analyses: direct current operating point, transient, noise, etc. However, the complexity of these analyses is especially high when simulating devices that interact with the real world (motors, switches, photo-diodes, etc.) as they require to specify the properties of their components depending on the physical environment.

There are different simulators for microcontrollers, but most of them are very specialized tools. These simulators change the registers of the microcontroller based on the firmware introduced without simulating other hardware than LEDs [12]. Some of them have been extended to interact with electronic circuits. For example, Proteus [13] or SimulIDE [14] are able to combine the simulation of a microcontroller with a SPICE simulation. They are very useful to learn electronics, however they are not able to perform physics simulations and their use for teaching robotics is limited.

3 HoRoSim

HoRoSim [2] has four main components. The first one is the Arduino code that students program, which has been extended with a function to specify the hardware devices employed (electronic circuits, motors and sensors). The second one is the HoRoSim libraries that replace the standard libraries used by Arduino. The third component is a robot simulator, CoppeliaSim [7], that is employed for visualization and calculation of the physics (multi-body dynamics, object collision, distance sensors, etc.). Finally, there is a user interface to handle those devices that are not part of the physical simulation and that the user just uses to interface with the microcontroller (buttons, potentiometers and LEDs). A representation of these four components is shown in Fig. 1.

HoRoSim is used from the Arduino IDE and uses the same sketch building process as Arduino. Basically, HoRoSim installs a new Arduino "core" (software API) that compiles and links the user's program to the HoRoSim libraries. Thus, the user can choose to use HoRoSim (and the Arduino board to be simulated[3]) from the Board menu in the Arduino IDE. For the user, the only difference is

[2]HoRoSim is open source (MIT license) and available for download at https://bitbucket.org/afaina/horosim

[3]Currently, only Arduino Unos are supported.

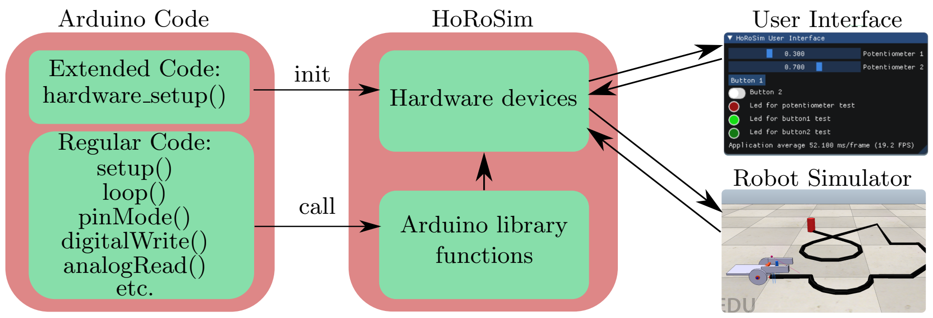


Fig. 1: Main components of the HoRoSim simulator: The Arduino code is extended to specify the hardware devices employed, the code is compiled using the HoRoSim libraries and the libraries are in charge of communicating with a robot simulator (CoppeliaSim) and the user interface.

that HoRoSim needs an extra function in the program that specifies the hardware devices (motors, sensors, LEDs and its circuits) that are connected to the Arduino. This is done in a function called *hardware_setup()* that is mandatory. In this function, the user creates instances of the hardware devices employed and specifies their properties and the pins of the Arduino that are used to interact with them. Two implementations of the hardware setup function can be seen in section 4.

Currently, the hardware devices that HoRoSim can simulate are shown in Fig. 2 and include:

– Direct current (DC) motors controlled with a transistor or with motor controllers (H-bridges)
– Stepper motors controlled by drivers with a STEP/DIR interface
– Radio control (RC) servomotors
– Proximity sensors like ultrasound or infrared (IR) sensors (analogue or digital)
– Infrared sensors used as vision sensors to detect colours (analogue or digital)
– Potentiometers connected to a joint in the robot simulator
– LEDs (User Interface)
– Potentiometers (User Interface)
– Buttons (momentary or latching) (User Interface)

The program is compiled using a C++ compiler using the Arduino IDE interface (verify button), which creates an executable program. This program can be run from the Arduino IDE (upload button) or through a terminal in the computer.

Once the compiled program is launched, it connects to the robot simulator (CoppeliaSim) through TCP/IP communication and starts a user interface. The robot simulator is used to perform the rigid-body dynamic calculations.

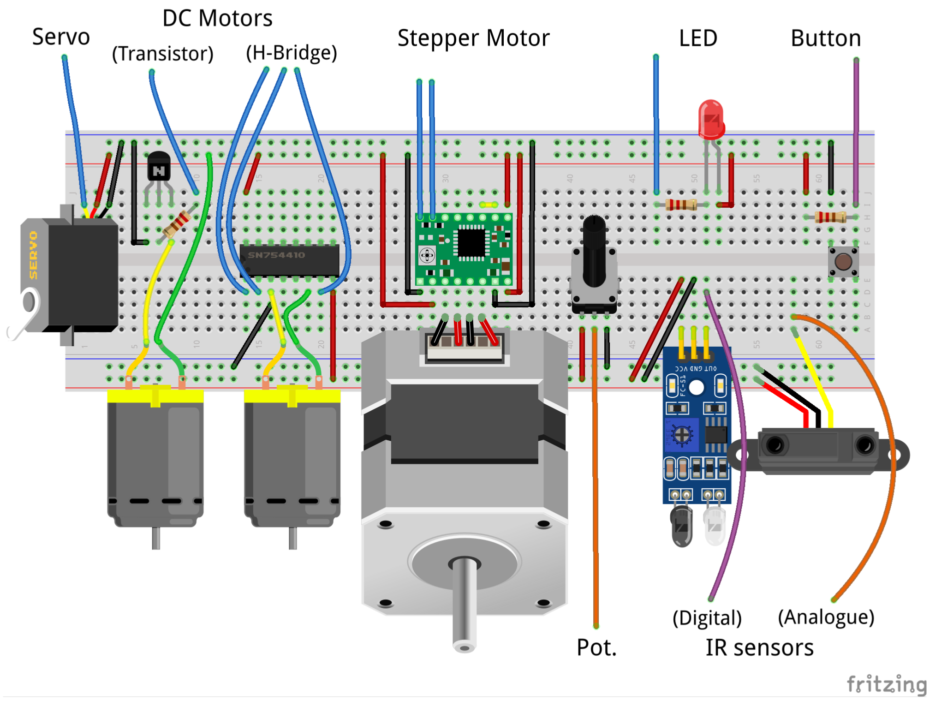


Fig. 2: The different hardware devices wired in a breadboard; note that this view is for illustration purposes only and it is not available in the simulator. The wires that come out of the breadboard are the connections that need to be specified in the *hardware_setup* function (blue for Arduino outputs, purple for digital inputs and orange for analogue inputs). For the sake of clarity, the power applied to the rails of the breadboard is not shown.

The components that do not need the robot simulator (LEDs, buttons and potentiometers) are rendered in the user interface using ImGui, a graphical user interface library for C++.

Calls to the Arduino functions are executed using the HoRoSim libraries. Currently, the list of available Arduino functions is shown in Table 1. The functions that interact with hardware devices (I/O and Servo libraries) call the respective functions for each hardware device instance that has been defined in the *hardware_setup* function. The code in each hardware device instance will check that the pin of the Arduino that is passed as an argument matches the pin where the hardware device is connected and that the pin has been initialized properly. If these conditions are met, the HoRoSim libraries call the functions to replicate the expected behaviour in the robot simulator or user interface. Therefore, a digital output pin in the Arduino can control several devices. An example of this software architecture for the digitalWrite function can be seen in Fig. 3. In this case, the user has defined a DC motor controlled by a transistor, which

Table 1: Arduino functions implemented in HoRoSim

I/O	Servo	Serial	Time
pinMode	attach	begin	delay
digitalRead	write	write	delayMicroseconds
digitalWrite	writeMicroseconds	print	millis
analogWrite	read	println	micros
analogRead	detach		
	attached		

base is connected to a specific pin in the Arduino. If the conditions are met, the library sends a message to CoppeliaSim to set a new speed (0 or maximum speed) depending on the argument passed as value. In case of functions that return a value (analogRead and digitalRead), the mechanism is similar. The logic implemented takes into account that the analogue values can only be read in analogue pins and analogue writes can only be used in the pins that have pulse width modulated (PWM) hardware.

Thus, there is no simulation of the electronic circuits. The libraries just replicate the behaviour of the electronics. Of course, this means that there are some electronic aspects that are not addressed such as the appropriate values of the components used or the wiring of the circuits.

Before running HoRoSim, the CoppeliaSim simulator needs to be launched and the user needs to load the scene that contains the robot and environment to simulate. The user can also decide which physics engine to use (ODE, Bullet, Newton, etc.). The name of the joints and sensors used in the scene should be introduced as arguments to the appropriate hardware devices in the *hardware_setup* function. The simulation is automatically started when the student presses the upload button in the Arduino IDE and stops automatically when the user interface is closed.

In order to create the scenes, the students need to use the functionality provided by CoppeliaSim. This process consists in generating the body of the robot using primitive shapes and connecting them together with joints (kinematic pairs which can be passive or actuated). This process can be time consuming, but it only needs to be performed once.

The simulation is asynchronous. CoppeliaSim updates the physics engine in its own thread and HoRoSim sends the commands and reads the sensors independently. Therefore, the handling of the time in HoRoSim is not accurate. The delays and time related functions are handled by the chrono library. In addition to the inaccuracies caused by the operating system, the communications with the simulator and the update of the graphical user interface introduce more delays. As a workaround, students can slow down the physics simulator through its graphical user interface (GUI). However, we have found that most robots can be simulated without noticeable side effects.

```
void digitalWrite(char pin, char value) {
  for each HardwareDevice instance
    hardwareDevice->digitalWrite(pin, value)
}
```

(a) The Arduino functions call the functions for each hardware device instance.

```
void DCMotor_Transistor::digitalWrite(int pin, int value) {
  if pin is not the pin that controls the transistor:
    return
  if pin has not been initialized as OUTPUT:
    return

  if value is LOW:
    set speed to 0
  else:
    //rpm_max is an argument passed during the
    //hardware device initialization
    set speed to rpm_max

  //Call the function that sends a command to CoppeliaSim
  setTargetSpeed(speed)
}
```

(b) Example of a hardware device (a transistor that controls a DC motor)

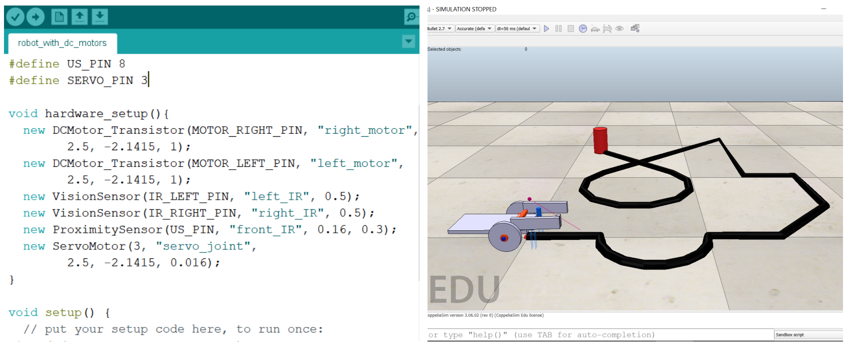
Fig. 3: Pseudo-code that illustrates how HoRoSim works. The Arduino functions call the functions of the hardware devices instances. Each hardware device instance checks that that pin passed as argument is used in that device and that it has been initialized properly. If so, the correct behaviour is applied or the correct value returned.

4 Use Cases

In this section, we will illustrate the potential of the simulator with two examples: A line following robot and a testbed to study PID controllers.

4.1 Line Following Robot

This example was used by our master´s students (21 to 30 years of age) to finish a mandatory assignment of the course as the real robots were locked in the university during the lockdown. The assignment consisted in programming a robot able to follow a line, find a can at the end of the line, grasp it with a gripper and place it near a crossing. This example replicates all the hardware that the students had in their line-following robots: DC motors controlled by transistors, an Arduino, 2 IR sensors for detecting the line on the floor, an ultrasound sensor



```
#define US_PIN 8
#define SERVO_PIN 3

void hardware_setup(){
  new DCMotor_Transistor(MOTOR_RIGHT_PIN, "right_motor",
        2.5, -2.1415, 1);
  new DCMotor_Transistor(MOTOR_LEFT_PIN, "left_motor",
        2.5, -2.1415, 1);
  new VisionSensor(IR_LEFT_PIN, "left_IR", 0.5);
  new VisionSensor(IR_RIGHT_PIN, "right_IR", 0.5);
  new ProximitySensor(US_PIN, "front_IR", 0.16, 0.3);
  new ServoMotor(3, "servo_joint",
        2.5, -2.1415, 0.016);
}

void setup() {
  // put your setup code here, to run once:
```

(a) Arduino code (b) CoppeliaSim

Fig. 4: A line following robot that was used to work on a mandatory activity during the lockdown.

to detect the can and a gripper controlled by a RC servomotor. Figure 4a shows the Arduino code that initializes the *hardware_setup* function and Figure 4b shows the robot following the line.

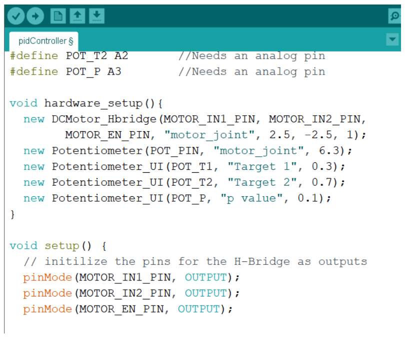
The scene with the robot, the line and the can was given to the students. To avoid problems, we also gave them a small Arduino sketch with the *hardware_setup* function already implemented to check that the connection between HoRoSim and CoppeliaSim worked correctly.[4] As the installation of HoRoSim is a bit cumbersome in Windows, an Ubuntu virtual machine was available to download with HoRoSim already installed (students still had to install CoppeliaSim in their computers).

4.2 PID Testbed

This example illustrates how to use HoRoSim for teaching complex topics. In this case, a testbed for PID controllers has been implemented.[5] The scene consists of a big disc that is static (not affected by the physics engine) with a graduated scale where a DC motor (red joint) controls a dial (dynamic body). The DC motor is controlled by an H-bridge and, therefore, it can rotate the dial in both directions. The angle of the dial is measured by a potentiometer, which is connected to the dial. Hence, a feedback loop can be created. The Arduino sketch of this example setups these hardware devices and three extra potentiometers to interact with the

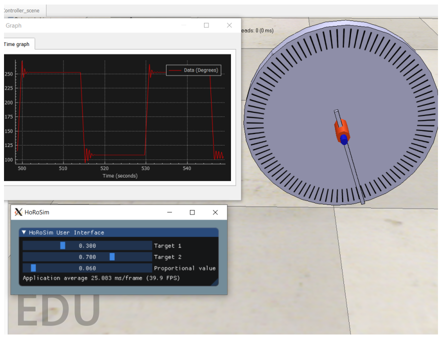
[4]The scene and the Arduino code with the *hardware_setup* function implemented are available for download at https://bitbucket.org/afaina/horosim/src/master/examples/lineFollowingRobot/. The code to follow the line is not provided as it is the students´ task to program the robot. A video of the robot following the line is available at https://bitbucket.org/afaina/horosim

[5]The Arduino code and the scene are available for download at https://bitbucket.org/afaina/horosim/src/master/examples/pidController/. A video is available at https://bitbucket.org/afaina/horosim/src/master/videos/pid_controller.mp4



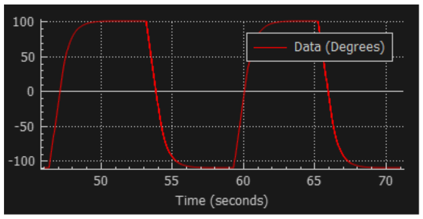
```
#define POT_T2 A2        //Needs an analog pin
#define POT_P A3         //Needs an analog pin

void hardware_setup(){
  new DCMotor_Hbridge(MOTOR_IN1_PIN, MOTOR_IN2_PIN,
        MOTOR_EN_PIN, "motor_joint", 2.5, -2.5, 1);
  new Potentiometer(POT_PIN, "motor_joint", 6.3);
  new Potentiometer_UI(POT_T1, "Target 1", 0.3);
  new Potentiometer_UI(POT_T2, "Target 2", 0.7);
  new Potentiometer_UI(POT_P, "p value", 0.1);
}

void setup() {
  // initilize the pins for the H-Bridge as outputs
  pinMode(MOTOR_IN1_PIN, OUTPUT);
  pinMode(MOTOR_IN2_PIN, OUTPUT);
  pinMode(MOTOR_EN_PIN, OUTPUT);
```

(a) Arduino code

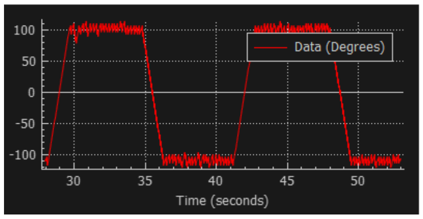


```
HoRoSim User Interface
0.300   Target 1
0.700   Target 2
0.060   Proportional value
Application average 25.083 ms/frame (39.9 FPS)
```

(b) CoppeliaSim and UI



(c) p=1



(d) p=100

Fig. 5: Simulation of a PID testbed. The dial is moved between two different target positions using a proportional controller. The user can change the target positions and the proportional constant by moving the potentiometers in the user interface (sliders). The angle of the dial is plotted for proportional values of 1, 10 and 100 in subfigures c, b and d respectively. Thus, different behaviours are observed: overdamped, overshooting and oscillation around the target position.

testbed, see Fig. 5a. The code of the sketch implements a closed loop proportional (P) controller, but of course this could be extended to PID controllers. The dial is moved continuously between two target positions by the DC motor every few seconds. The target positions and the proportional constant can be adjusted by tuning the three potentiometers in the user interface (Fig. 5b). In addition, a graph element has been inserted in the scene. This graph plots the angle of the dial (in degrees) versus time and it is updated automatically during the simulation.

This example allows students to experiment with different controllers (and values of their constants) and observe the different behaviours that they produce. In Fig. 5b, a proportional value around 10 has been set with the potentiometer and the graph shows a small overshooting when reaching the targets. When changing the proportional value, different behaviours are observed as shown in Fig. 5c and 5d. Note that the simulator allows students to quickly change other properties and not only the software of the controller. For example, they could study the response of the system when increasing the mass of the dial or increasing the speed or torque of the motor.

5 Discussion

5.1 Students´ Reception

The students used the simulator to fulfil the last mandatory assignment of the course (programming a line-following robot) and the reception of the simulator was very good. As an example, one student wrote in the evaluation of the course: "I liked that Andres made the simulator which helped us finish the mandatories! It was very cool to use it actually.". And a lot of students liked the attempt to keep a hands-on course even during the lockdown, to which the simulator contributed.

Furthermore, all the students were able to program their robots to successfully pass the mandatory assignment. None of them complained about the simulator and most of the questions were about how to achieve that the robot could make the sharp turns and go straight over the crossing. In that sense, it was a pleasure to observe that the mandatory assignment could be realised online without any problems and triggering the same kind of questions as if they had used a physical robot.

The simulator, which was originally built to only cover the line following robot assignment, was extended to allow simulating other kinds of robots and machines, and more hardware devices were implemented. This gave students the possibility of using the simulator for their final projects (build a robot or a machine with mechanisms, electronics and embedded programming). When they started to work on their final projects, a small survey was carried out and 10 students replied. Half of them were interested in using the simulator for their final projects, while the other half discarded it. The reasons to discard the simulator were: 3 of them were very busy to use it, 1 machine could not be simulated, and 1 machine could be built at home and there was no need for the simulator. However, only one group out of twenty ended up using the simulator.

The low usage of the simulator (1 group out of 20) was motivated by several reasons. First, the students could use a small budget to order electronic and mechanical parts from online stores and prototyping services. Thus, most of the students managed to finish with a minimum prototype and there was no need for the simulator. However, the university had to use a considerable amount of money to pay for this when in a normal course the students can prototype their own parts at our workshop and use our stock materials. Additionally, some students reported a lack of engagement and difficulty to collaborate during the lockdown. Finally, the current limitations of the robot simulator, which are described in the next section, also contributed to the low usage of the simulator.

5.2 Limitations

The current implementation of the code has several limitations. First, there are still several functions and libraries available for Arduino that have not been implemented. They include interrupts, receiving serial messages and I2C (Inter-Integrated Circuit, Wire in Arduino) and SPI (Serial Peripheral Interface) libraries. They could be implemented in the future, but specific I2C and SPI libraries need to be implemented for each device (sensor, motor controller, etc.) which can be tedious.

In addition, the electronic circuits provided are abstractions of the physical world. Thus, the students are not forced to think how to wire the electric components between them. And the lack of a SPICE simulator makes impossible to detect problems that could happen when working with hardware (wiring, value of the components, power supply voltages, etc.).

Finally, a big limitation is that the students need to create a scene with a physical model of their prototypes. In our course, they use Computer-Aided Design (CAD) software to design their robots. However, the translation from these 3D models to CoppeliaSim is not automatic. One could import the parts in CoppeliaSim as meshes, but this makes the dynamic engine slower and can cause stability issues. Additionally, too many dynamically enabled parts can again slow down the simulator significantly. A better approach is to design the prototype in CoppeliaSim using primitive shapes that are faster to simulate and simplify the mechanisms. For example, most of our students implemented a gripper for the line following robot using a servo that rotates some gears, which open or close the end effector. In the simulator, this is simplified as a linear actuator that opens or closes a cuboid body. Thus, all the gears and their joints are not simulated. To compensate for this, a reduction ratio between the motor and the axis of movement was introduced as a parameter in the constructor of the hardware devices. This value represents the transmission (gears, belts or leadscrew) and it will reduce the speed and increase its force/torque accordingly.

The main inconvenience of the necessity of generating a new and simplified model of the machine to simulate is that it requires to learn how to use CoppeliaSim. In a course where the students already learn several programs and topics, introducing one more increases the overload of the students. And the survey showed that most of them are already very busy and not able to handle more workload. Thus, this limitation needs to be addressed to increase the utility of the simulator.

6 Conclusions

The paper has shown a new robot simulator, HoRoSim, that has a holistic approach. Students can simulate Arduino code, electronic hardware and multi-body dynamics, which helps to teach the basic principles of robotics. To the best of our knowledge, it is the first simulator that allows users to simulate Arduino code combined with a physics engine. HoRoSim provides teachers with a flexible tool that can be used for assignments in robotic courses. Preliminary results have shown that it can contribute to keep hands-on courses and student engagement in online courses, which is especially relevant in the current pandemic.

In order to address the limitations of the current version, future work includes (1) generating CoppeliaSim models automatically from CAD designs, (2) allowing students to make their own electronic circuits by wiring electronic components, and (3) adding a SPICE simulator. We are currently looking at how we could integrate Fritzing [15] and Ngspice[11]. Fritzing could provide a platform where students create their circuits (as schematics or wiring the components in a breadboard), while Ngspice could simulate these circuits.

References

1. S. Schocken, Nand to tetris: Building a modern computer system from first principles, in: Proceedings of the 49th ACM Technical Symposium on Computer Science Education, 2018, pp. 1052–1052.
2. F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, A. Martinoli, The e-puck, a robot designed for education in engineering, in: Proceedings of the 9th conference on autonomous robot systems and competitions, Vol. 1, IPCB: Instituto Politécnico de Castelo Branco, 2009, pp. 59–65.
3. F. Bellas, M. Naya, G. Varela, L. Llamas, A. Prieto, J. C. Becerra, M. Bautista, A. Faiña, R. Duro, The robobo project: Bringing educational robotics closer to real-world applications, in: International Conference on Robotics and Education RiE 2017, Springer, 2017, pp. 226–237.
4. R. Smith, et al., Open dynamics engine (2005).
5. E. Coumans, et al., Bullet physics library, bulletphysics.org, accessed: 04-02-2021.
6. N. Koenig, A. Howard, Design and use paradigms for gazebo, an open-source multi-robot simulator, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vol. 3, IEEE, 2004, pp. 2149–2154.
7. E. Rohmer, S. P. Singh, M. Freese, V-rep: A versatile and scalable robot simulation framework, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2013, pp. 1321–1326.
8. O. Michel, Cyberbotics ltd. webots™: professional mobile robot simulation, International Journal of Advanced Robotic Systems 1 (1) (2004) 5.
9. L. Nagel, D. Pederson, Simulation program with integrated circuit emphasis, in: Midwest Symposium on Circuit Theory, 1973.
10. Analog Devices, Ltspice, https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html#, accessed: 08-01-2021.
11. H. Vogt, M. Hendrix, P. Nenzi, D. Warning, Ngspice users manual version 33 (2020).
12. P. F. Gonçalves, J. Sá, A. Coelho, J. Durães, An arduino simulator in classroom-a case study, in: First International Computer Programming Education Conference (ICPEC), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
13. Bo Su, Li Wang, Application of proteus virtual system modelling (vsm) in teaching of microcontroller, in: 2010 International Conference on E-Health Networking Digital Ecosystems and Technologies (EDT), Vol. 2, 2010, pp. 375–378. doi:10.1109/EDT.2010.5496343.
14. SimulIDE, https://www.simulide.com, accessed: 08-01-2021.
15. A. Knörig, R. Wettach, J. Cohen, Fritzing: a tool for advancing electronic prototyping for designers, in: Proceedings of the 3rd International Conference on Tangible and Embedded Interaction, 2009, pp. 351–358.