Robotics learning through a game-inspired simulator integrated with robotized RC cars

Anderson Molter, Sanjay Sarma O.V., Bryce Davis, and Ramviyas Parasuraman

Heterogeneous Robotics Lab, University of Georgia, Athens, GA 30602, USA
{amolter,sanjaysarmaov,ramviyas}@uga.edu

Abstract. This short paper introduces a novel learning technology for teaching and learning foundation courses in robotics and AI at universities. We present the integrative simulation-hardware framework, which consists of a lightweight game-inspired car simulation tool and a low-cost hardware design of RC cars converted into robot platforms. The main objective of this contribution is the creation and utilization of inexpensive and open-sourced educational materials. We show the current developments in the Unity-based simulation tool and its hardware integration design and implementation of a PID controller-based algorithm for lane following as an illustrative educational module.

Keywords: Autonomous Cars, Simulation, Learning Technology

1 Introduction

Robotics is an excellent example of STEM education, and the field is believed to dominate technological development and commercial industries in the near future. Specifically, Artificial Intelligence (AI) and the autonomous vehicles industry see rapid growth and research and development progress in the last decade. To equip students to contribute actively and become leaders in changing people's lives through this technology, we need inspirational, motivational, and transformative methods to teach robotics and AI courses. Specifically, we propose a new learning technology to achieve hands-on educational objectives aligned with experiential and active learning methodology, which are widely becoming a focus in interdisciplinary curriculum initiatives at universities across the globe.

In this research, we focus on innovation in low-cost, open, intuitive learning and education methodologies for teaching Robotics, AI, and related subjects. Here, we integrate the use of games (e.g., Unity [1] simulator tool for testing robotic algorithms) and toys (RC vehicles converted to innovative robot platforms) in the proposed project. We first transform (robotize) a commercial RC car into a programmable car capable of sensing its environment through cameras and IR sensors and moving on a race track on its own using a microcomputer unit. Then, we develop a game-inspired Unity-based simulator[1] for the car. Both

[1] See more information and a live demo at https://github.com/herolab-uga/herocars.

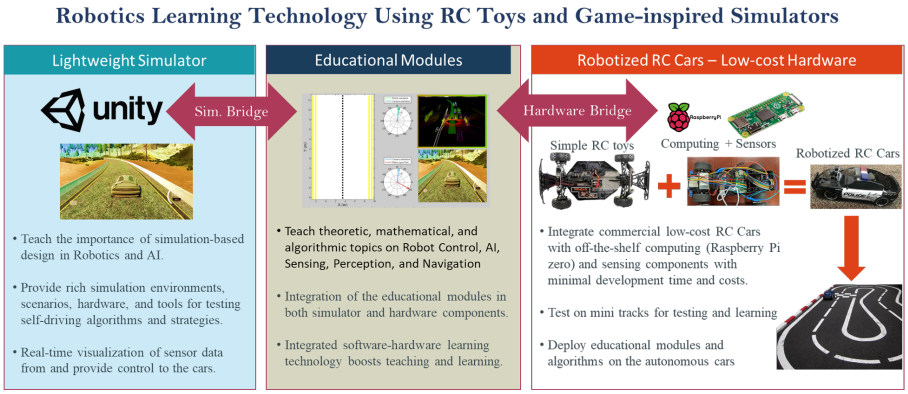

Fig. 1: *An overview of the proposed learning technology for Robotics and AI courses in university-level higher educational curriculum.*

the real car and the simulator are integrated in terms of the same educational modules that help teach and learn Robotics and AI concepts such as lane following and motor control. The proposed approach is depicted in Fig. 1.

This paper outlines our contributions and progress made so far on the game-inspired simulator tool for evaluating autonomous controllers and discussing its integration with the real robotized RC car. We use a PID controller as an illustrative educational module. We aim to open-source our contributions once we achieve a stable development phase.

2 Background and Related Work

Conventionally, robotics and related courses involving a practical curriculum use educational toys such as LEGO Mindstorms, Robotis PLAY 700 kit, etc. Although they serve to intuitive and gain hands-on experience with robots, real-world robotics research and commercial products are very different from these educational toy kits. So, teachers and researchers have always come up with innovative ways of designing new educations-yet-research-oriented robot platforms (e.g., e-puck robot [2] and Cambridge minicar [3]).

On the other hand, real research robot platforms are highly expensive, which limits the use in undergraduate and graduate course practicum and projects. Therefore, simulator-based educational tools have been developed, but most of them focus on autonomous driving as an independent module [4].

To bridge these gaps, we develop a new game-based simulator that can be well-integrated with the real car in terms of functionalities and interfaces for the educational modules. This proposed learning technology-based education method is expected to complement active learning [5], problem-based solving [6,7], and experiential learning methodologies [8] techniques. Therefore, it provides an intuitive understanding and inspiring solutions for implementing the theoretical and algorithmic concepts of robotics, AI, and autonomous vehicles.

3 Game-inspired Simulator for Robotics Education

3.1 Robotized RC Car Hardware

We tested different hardware components such as sensors, embedded electronics, computing boards, and other commercially available robotic-toy solutions. Currently, our car consists of these components, which include a power regulator, a Raspberry Pi Zero W - a tiny computer, a 5-channel infrared tracking sensor, a 9 DoF integral measurement unit comprising of gyroscope and magnetic compass, a motor driver to connect to the motors in the car chassis, and a camera that will stream live video feed for teaching advanced computer vision algorithms (reserved for future use). Fig. 2 shows an example of a toy RC car converted into a robotic platform that we use in testing software solutions.

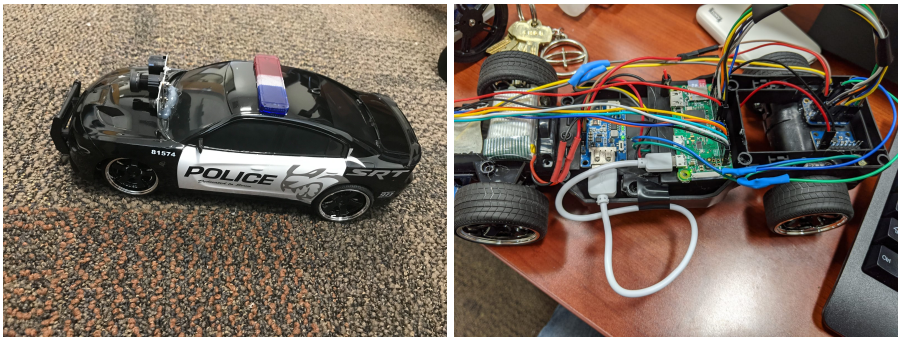

Fig. 2: *A sample assembled car outlook and its internal view.*

3.2 Unity-based Simulator

We chose to develop the simulator using the Unity tool because of its game-like environment and portability to any computing platform, including mobile devices. To accomplish the task of controlling our RC car around the simulated track, Unity's Rigid bodies have been applied to all road pieces and the 3D car model, and these will be used for both gravitational and friction physics within the simulation. An invisible line will be in the center of the road, and this will collide with colliders that have been placed on the car (to simulate line tracking). The AI will make its decisions based on which collider underneath the car the road collider is encountering, which is similar to how the IR sensors on the bottom of our real-life RC car work. So far, the physics of the 3D simulation has been defined, and colliders are set in place, so the next task is to get the car moving and design the AI that will control this car.

The AI, like the real-life RC car, will be driven by a PID (Proportional-Integral-Derivative) controller to make the decisions on whether the car should accelerate, turn left or right, or slow down. Fig. 3 shows the developments achieved so far. See https://github.com/herolab-uga/herocars for more details on the PID control implementation as an educational module.

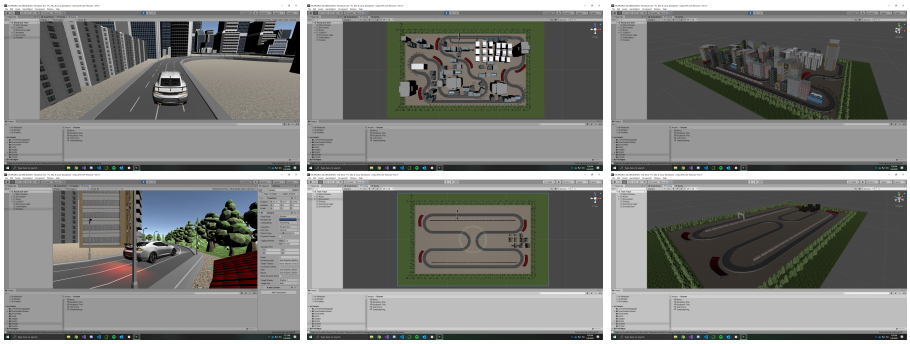

Fig. 3: *Screenshots of the Unity-based robot car simulator showing the car in the simulated environment along with two samples of the tracks.*

3.3 Simulator-Hardware Integration

Interaction with the cars in a meaningful way to allow students to control and chose what type of decisions the car would make was a goal from the beginning. The Raspberry Pi Zero W we used comes with a WiFi card that allows us to send commands to this Raspberry Pi after we connected to our network. The Raspberry Pi on the car and the Unity interface are both using TCP sockets to allow certain connections between the two. We designate connections based on the local IP4 of the Raspberry Pi. The Unity interface acts as a server that hosts a TCP server, a data server, and an interface that allows users to change the constants of the AI and control the car manually.

Once the scene begins in Unity, a car object is created which stores the same values as the real RC. Once every 60th of a second, the Unity simulation updates the car on any user input and value changes to the PID constants; the car will send the Unity interface the error the car is currently experiencing on the track. The exact values being received by the car are as follows: P constant, I constant, D constant, pause car, manual mode, left motion, right motion, forward motion, backward motion. The motions refer to the users manual commands telling the car to steer left or right and/or drive forward or backward. The car will stop all motion automatically if the car is either no longer on the designated track or if the car has become disconnected to the Unity interface. This is done so we do not have the car running off without the control of the user.

In Fig. 4, we show the UML diagram of the software architecture within the RC car. We start with our main driver, which is starting two threads that run the RC Controller and the TCP Client simultaneously. The TCP Client uses the server's IP address to connect to the TCP server being hosted on the Unity interface. This is where data will be sent from and received by the Unity interface; This class will also deal with update the RC Controller that was passed to the TCP Client with the user's current desires. At the same time, the RC Controller will be getting data from the IR sensors and calculating the error and desired degree to which to turn the motors using our PID controller. The RC Controller will then send a command to the motor driver, which will execute the commands that physically move the car in the desired direction.

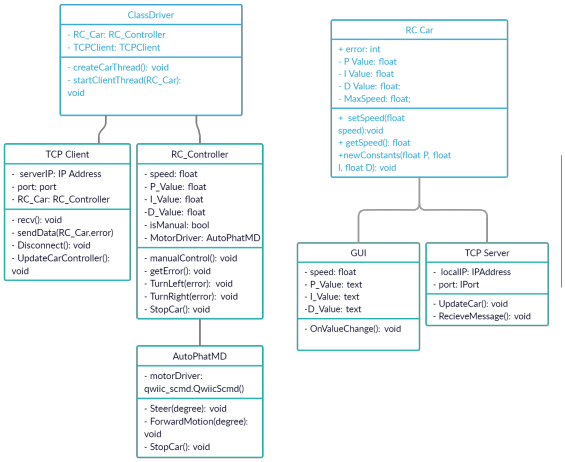

Fig. 4: *UML diagram of the RC Car driver software (left) and the UML diagram of the Unity Simulator Interface (right).*

Fig. 5: *Unity interface for user interaction with a real Car and the simulator.*

In Fig. 4, the UML diagram for our Unity simulator is also shown. The main class here is the RC Car class, which is an object that represents the RC car within Unity. This class stores our values, which, if changed, will be sent to the RC car for those changes to be implemented. The GUI class is waiting for changes to be made within the UI text fields and, if changed, will update the RC car object's values. Those values will then be sent to the RC car via the TCP Server class, which hosts a TCP Server on the network and can both receive and send data to the RC car.

Fig. 5 shows the GUI interface of our simulation. As stated previously, this car has been modeled to be very similar to the RC car in its turning radius and speed capabilities. It uses the same algorithm and has as many similarities to the RC car as possible. The Unity Interface includes buttons for manual control, text boxes that allow the user to change constants, current speed, and the current error reported from the car. As well as those things, in the middle is a space for camera data from the car. The Raspberry Pi is connected to a camera on the hood of the car and streams that camera data to the Unity Interface, allowing the user to see what the car sees.

The Unity Engine allows us to deploy our Unity interface using a WebGL client easily. The native web client allows us to skip the large download size a Unity desktop application may have and will grant users easy access to the interface needed to control the RC car. Because of the integration of the Unity framework with our RC car, we have many possibilities to what we can expand to when it comes to the educational modules using the RC car. For instance, we are currently working towards using the computer that hosts the Unity interface to train a Reinforced machine learning model based on both the IR sensor data and the camera data being streamed to the Unity interface. We use the native ml-agents package within Unity to seamlessly connect our Unity environment to a PyTorch learning brain.

4 Conclusion

We introduced a new affordable learning technology for hands-on robotics and AI education at higher education institutions. Game-inspired simulation design and a low-cost hardware prototype of robotized RC cars can achieve smooth integration to teach various educational modules. With a PID controller-based algorithm for lane following, we demonstrated the simulator design's current progress and its integration with the real hardware.

References

1. W. A. Mattingly, D.-j. Chang, R. Paris, N. Smith, J. Blevins, and M. Ouyang, "Robot design using unity for computer games and robotic simulations," in *2012 17th International Conference on Computer Games*. IEEE, 2012, pp. 56–59.
2. F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and 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, no. 1. Castelo Branco, Portugal, 2009, pp. 59–65.
3. N. Hyldmar, Y. He, and A. Prorok, "A fleet of miniature cars for experiments in cooperative driving," *arXiv preprint arXiv:1902.06133*, 2019.
4. D. Fernandes, F. Pinheiro, A. Dias, A. Martins, J. Almeida, and E. Silva, "Teaching robotics with a simulator environment developed for the autonomous driving competition," in *International Conference on Robotics in Education (RiE)*. Springer, 2019, pp. 387–399.
5. S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, "Active learning increases student performance in science, engineering, and mathematics," *Proceedings of the National Academy of Sciences*, vol. 111, no. 23, pp. 8410–8415, 2014.
6. A. W. Oliveira and A. O. Brown, "Exemplification in science instruction: Teaching and learning through examples," *Journal of Research in Science Teaching*, vol. 53, no. 5, pp. 737–767, 2016.
7. D. A. Muller, *Designing effective multimedia for physics education*. University of Sydney, 2008.
8. C. C. Chung, C. Cartwright, and M. Cole, "Assessing the impact of an autonomous robotics competition for stem education," *Journal of STEM Education: Innovations and Research*, vol. 15, no. 2, p. 24, 2014.