applied sciences

MDPI

Article

Facilitating Robot Learning in Virtual Environments: A Deep Reinforcement Learning Framework

Algirdas Laukaitis *, Andrej Šareiko and Dalius Mažeika

The Faculty of Fundamental Sciences, Vilnius Gediminas Technical University, Saulėtekio al. 11, LT-10223 Vilnius, Lithuania; andrej.sareiko@stud.vilniustech.lt (A.Š.); dalius.mazeika@vilniustech.lt (D.M.)
* Correspondence: algirdas.laukaitis@vilniustech.lt

Abstract: Deep reinforcement learning algorithms have demonstrated significant potential in showcasing robotic capabilities within virtual environments. However, applying DRL for practical robot development in realistic simulators like Webots remains challenging due to limitations in existing frameworks, such as complex dependencies and reliance on unrealistic control paradigms like a 'supervisor'. This study introduces an open-source framework and a novel pattern-based method designed to facilitate the exploration of robot learning capabilities through reinforcement learning algorithms in specialized virtual testing environments built on Webots. Our approach simplifies setup by avoiding burdensome external package installations and, crucially, removes the dependency on an unrealistic 'supervisor' entity, offering a more practical and real-world-aligned solution. Designed to leverage Webots' realistic simulation capabilities, the proposed method and system are validated through various examples, ranging from the classic inverted pendulum scenario to a production robot utilized in an actual assembly line. The developed code and examples are publicly accessible on GitHub for the deep reinforcement learning research community.

Keywords: reinforcement learning; virtual environments; robot testing; Webots; Stable-Baselines3; simulation tools; digital twin; robotic training

Academic Editor: Rui Araújo

Received: 30 March 2025
Revised: 25 April 2025
Accepted: 29 April 2025
Published: 30 April 2025

Citation: Laukaitis, A.; Šareiko, A.; Mažeika, D. Facilitating Robot Learning in Virtual Environments: A Deep Reinforcement Learning Framework. *Appl. Sci.* **2025**, *15*, 5016. https://doi.org/10.3390/app15095016

1. Introduction

In the evolving field of robotics and artificial intelligence, the integration of reinforcement learning (RL) has emerged as a transformative approach to training autonomous agents. Reinforcement learning, a subset of machine learning, empowers robots to interact with their environment, learn from their experiences, and optimize their behavior through a process of trial and error reinforced by reward mechanisms. Virtual environments, meanwhile, offer a safe, flexible, and cost-efficient platform for conducting these sophisticated training sessions, significantly accelerating the development of intelligent robotic systems [1].

The utilization of virtual environments in robotics provides a crucial advantage: the ability to simulate complex real-world scenarios without the risks and constraints associated with physical experimentation. From dynamic navigation in cluttered spaces to precise manipulation of objects, virtual environments can replicate a diverse array of challenges that robots may encounter in the real world. These settings are particularly beneficial for training tasks that involve high levels of uncertainty or potential hazards, such as disaster response robotics or autonomous driving systems. Moreover, advancements in simulation technologies now enable the creation of hyper-realistic environments, bridging the gap between synthetic training and practical deployment.

Reinforcement learning applications in virtual environments have already demonstrated significant potential across industries [2]. Robots trained via RL in simulation have been successfully deployed in manufacturing for precision assembly tasks, in healthcare for automated assistance, and even in space exploration for autonomous navigation [3–5]. However, the transition from virtual to real-world scenarios remains a formidable challenge, primarily due to discrepancies in dynamics, noise levels, and environmental variability. Addressing these challenges necessitates robust techniques for domain adaptation and transfer learning to ensure the seamless applicability of policies learned in simulation.

This study investigates the methodologies, development paradigms, and persistent challenges inherent in applying reinforcement learning (RL) within virtual environments. We present a comprehensive analysis of fundamental principles, recent technological advancements, and examples that show the practical benefits of this approach. Furthermore, we carefully look at the limitations and suggest future research to improve the accuracy and reliability of simulated training environments. By studying these topics, we aim to show how reinforcement learning can significantly change robotics, helping to build more adaptable, efficient, and practical machines.

As an example, we can examine Figure 1, which shows the development of a Universal Robots UR5e digital twin, created for this research project. On the left, the figure displays the physical UR5e robot used in a product assembly line. Currently, the robot's movements are programmed manually. Any changes to the environment require reprogramming, which can be expensive and lead to errors. To reduce the risk of damage to the robot or its surroundings, we created a digital representation of the robot and its environment. In the center of Figure 1, you can see the environment we designed for the UR5e using Fusion 360 software (2.0.2155). The important parts of this environment were then transferred to the Webots simulation software (R2025a). The UR5e model from Webots was added to this environment, as shown on the right side of Figure 1. Using the framework and tools described in this paper, we can now test different reinforcement learning algorithms without the risk of damaging the physical UR5e. Once we find the best movements for the UR5e to complete its tasks, we can transfer the deep learning models to the physical robot using the Robot Operating System (ROS) tools provided with Webots.

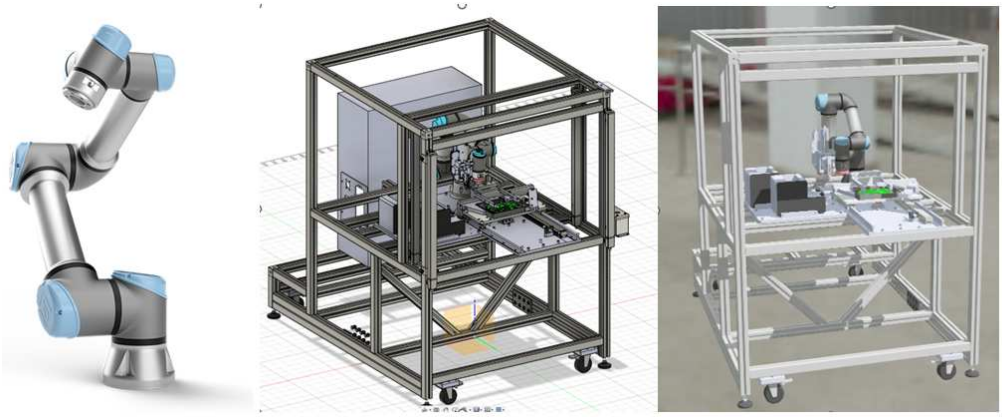

Figure 1. Development process that was used in this paper's research project of a Universal Robots UR5e digital twin: (**Left**) the physical UR5e cobot, (**Center**) its environment modeled in Fusion 360, and (**Right**) the simulated environment in Webots for robot testing.

The remainder of this paper is structured as follows. Section 2 reviews related work. Section 3 presents the general design patterns of the proposed solution and compares them to existing reinforcement learning approaches. Section 4 details the implementation of reinforcement learning algorithms within the Webots environment. Section 5 presents

the experimental results obtained using the proposed framework. Finally, the Section 6 summarizes the findings and outlines potential directions for future research.

2. Related Work

Reinforcement learning [6] is a branch of machine learning focused on enabling agents to learn optimal behaviors by interacting with their environment through trial and error. RL leverages a system of rewards and penalties to encourage learning through feedback, allowing agents to make decisions that maximize cumulative rewards [7]. A cornerstone of this methodology is the Markov decision process (MDP), which formalizes the interaction between an agent and its environment in terms of states, actions, transition probabilities, and rewards [8].

A key feature of RL is the balance between exploration and exploitation. Exploration refers to testing new actions to discover potentially better strategies, while exploitation focuses on leveraging known strategies to maximize immediate rewards. Modern RL employs advanced algorithms like proximal policy optimization (PPO) [9], deep Q-networks (DQN), and advantage actor-critic (A2C) [10], which allow agents to tackle complex, high-dimensional tasks effectively.

The OpenAI Gym (now Gymnasium) framework has emerged as a vital tool in the development and testing of RL algorithms [11]. It provides a standardized interface for a diverse range of simulated environments, from simple tasks like CartPole to intricate robotic control simulations [2]. Gym supports both discrete and continuous action spaces, enabling researchers and practitioners to benchmark their RL models under consistent conditions. By offering ready-to-use environments, it accelerates experimentation, promotes reproducibility, and fosters innovation in the RL community.

The Gym framework's modularity extends its usability, allowing developers to create custom environments tailored to specific applications. For instance, robotic RL, with tools like Gym's robotics suite, which integrate simulated tasks with physics engines like MuJoCo [12], enables training of manipulation or navigation policies in virtual settings before deploying them in real-world scenarios. This ability to simulate environments mitigates risks and reduces costs during the training phase, making Gym a pivotal resource for RL research.

Deep reinforcement learning (DRL) [13], which combines RL with deep neural networks, has further enhanced the scope of Gym applications. Frameworks like Stable-Baselines3 [14] and Ray's RLlib [15] build on Gym to provide robust RL algorithm implementations, simplifying the process of training agents in Gym-compatible environments. These integrations underscore Gym's flexibility and its role in advancing state-of-the-art RL techniques. In summary, RL principles, bolstered by frameworks like OpenAI Gym, continue to drive progress in artificial intelligence and robotics. By providing accessible, standardized environments, Gym empowers researchers and practitioners to push the boundaries of what autonomous agents can achieve in simulated and real-world scenarios.

As reinforcement learning (RL) problems increase in complexity, simulation environments become essential for development and testing. Simulation reduces the need for direct experimentation with physical systems, which is particularly critical for tasks with high degrees of freedom, such as autonomous driving. Consequently, simulation platforms like Webots [16] are increasingly important, especially those that are open-source.

Alternatives to Webots include Gazebo [17], RoboDK [18], CoppeliaSim, OpenRave, and Unity [19], all of which support the integration of deep reinforcement learning (DRL) algorithms. This paper focuses on Webots due to its capability to create robots from scratch, its realistic graphics, and its compatibility with the Robot Operating System (ROS). However, many aspects of the proposed framework can be adapted to other simulation environments.

These simulation platforms typically provide pre-built robot models, ranging from simple robots like the E-puck [20] and iCub [21] to industrial robots such as the UR5e [22] and IRB 4600 [23], which are utilized in this research.

A key distinction between environments like Gymnasium and simulation platforms like Webots is the fidelity of actuator and sensor modeling. In Webots, actuators and sensors, such as robotic arms and LiDAR, are designed to closely resemble their physical counterparts. Webots allows robot programming via controllers in multiple languages, including C, Python, and MATLAB. For DRL implementations, Python is preferred due to its ease of use and the straightforward translation of C examples.

Several projects aim to facilitate RL in robotic simulators. Gym-Ignition [24] provides an OpenAI Gym interface for Gazebo, supporting reproducible robot environments, external software integration, multiple physics and rendering engines, and ROS compatibility. Zamora [25] extends the Gym interface with ROS compatibility for Gazebo. Lopez [26] offers ROS 2 compatibility and is applied in real-world scenarios. NVIDIA Isaac ROS [27] provides a comprehensive framework for DRL and robotics, featuring photorealistic rendering and parallelization. While Deepbots [28] represented an earlier effort to facilitate DRL in Webots, its implementation presents several limitations for current research. Firstly, it relies on the now-deprecated OpenAI Gym standard, leading to compatibility issues with modern RL libraries and environments. Furthermore, integrating Deepbots requires the installation of a dedicated Python package, which itself carries dependencies often requiring older versions of core libraries, potentially causing conflicts within contemporary development setups. To our knowledge, other generic wrappers interfacing Webots with the current OpenAI Gymnasium are not implemented.

Most significantly, existing approaches like Deepbots often necessitate reliance on the Webots 'Supervisor' concept for managing the simulation state and facilitating the RL interaction loop (observation, action, reward). Our proposed framework introduces a key architectural improvement by largely removing this dependency on the Supervisor for the core RL agent–environment interaction. This presents the first generic interface for Webots built directly upon the updated OpenAI Gymnasium standard. By circumventing the older dependencies and the mandatory use of the Supervisor for basic interaction, our framework standardizes and significantly simplifies the process of applying modern RL algorithms within the Webots simulation environment.

3. Design Patterns of Reinforcement Learning in Virtual Environments

Reinforcement learning within virtual environments is facilitated through a dynamic interplay of simulation tools, RL frameworks, and machine learning algorithms. At the core of the presented architecture lies the need for a simulated environment capable of accurately modeling real-world dynamics, providing a controlled and risk-free setting for robotic training. Tools like Webots, Gazebo, and Unity play a pivotal role by offering physics engines that replicate the physical world, including gravity, friction, and collisions, along with providing sensory feedback such as visual, tactile, or distance-based data.

The WebotsRL system presented in this paper incorporates three design patterns used for experimentation. The first two patterns follow the reinforcement learning loop commonly found in frameworks like Gymnasium or Deepbots. In these frameworks, the environment receives an action from the robot at each step and returns a pair of observations and rewards. This process is repeated until the robot either achieves its goal or an exception terminates the loop. While this approach is effective for testing reinforcement learning in virtual environments, we identified a need for a more complex strategy when transferring RL-trained models and processes to real-world robots. To address this, we developed a

third design pattern that builds upon the first two, eliminating the Supervisor concept to better facilitate the transition to physical robots.

Figure 2 illustrates two design patterns (UML class diagram) comparable to those introduced for Webots in [28]. The pattern shown on the left side of Figure 2 represents the simplest approach for testing reinforcement learning (RL) algorithms in the Webots virtual robot simulation environment. This pattern is composed of two key classes: RLRobot and RLModel. The RLRobot class defines the robots to be trained and is responsible for gathering observations from the environment, executing actions, and resetting the environment to its initial state once the robot either achieves its goal or encounters a failure during the learning process. Complementing this, the RLModel class serves as the core component for the reinforcement learning algorithm within the framework. It acts as an interface, allowing for the implementation and integration of various reinforcement learning algorithms. Furthermore, this class is responsible for managing the learning lifecycle through three primary methods: learn(), which initiates the simulation process within the virtual environment to train the model's parameters; predict(), which is used to evaluate the trained model by collecting behavioral metrics and rendering the resulting behavior in the Webots environment; and save(), which provides functionality to store the trained model for subsequent reuse.

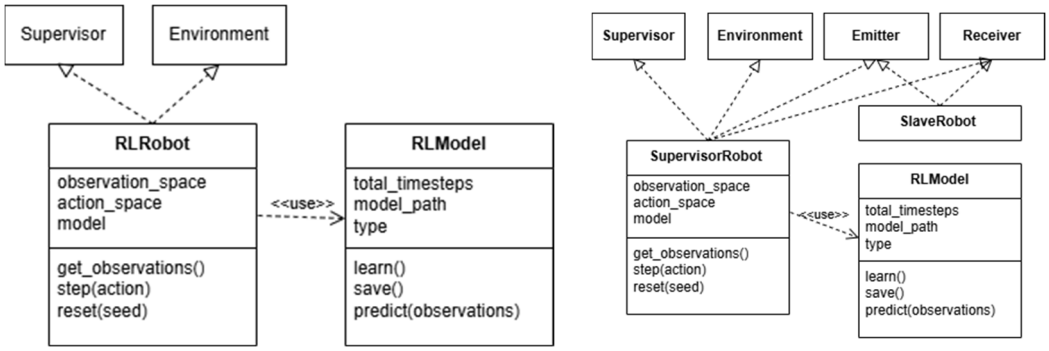

Figure 2. (**Left**) Simple design pattern with integrated Supervisor for robot RL implementation. (**Right**) Design pattern with separated Supervisor using Emitter–Receiver interface.

A critical limitation of this design pattern, particularly when transferring RL-trained models to physical environments and robots, is its reliance on inheritance from the Supervisor class. In many RL implementations, the Supervisor class is included to grant the robot comprehensive knowledge of the simulated environment. This allows access to data beyond the scope of the robot's sensors and enables manipulations such as repositioning objects in the environment, measuring distances to targets, or retrieving speed values of various objects using Webots' built-in libraries. However, such capabilities are unrealistic in real-world scenarios, where physical robots must rely solely on their sensors or external information provided by other robots or computational devices, such as dedicated servers.

To address these limitations, the design pattern illustrated on the right side of Figure 2 is proposed. This pattern adheres more closely to real-world constraints and aligns with the recommendations of Webots systems and previous work by [28].

The second design pattern introduces two robot classes: SupervisorRobot and SlaveRobot. Additionally, it incorporates the Emitter and Receiver classes provided by the Webots system to simulate communication between robots using string messages over virtual radio waves. In the proposed WebotsRL system, the Emitter and Receiver are used for communication between the SupervisorRobot and SlaveRobot, with the Emitter broadcasting messages and the Receiver receiving them.

Instances of SupervisorRobot are typically not typical robots and usually lack mass or physical properties in the simulation. For example, they can represent a computational device, such as a PC, that transmits actions to robots without interacting with the scene. Similarly, the Emitter and Receiver components could be simulated wireless devices, like Wi-Fi or Bluetooth modules.

In this design pattern, the reinforcement learning framework operates as follows: the virtual environment is first reset to its initial state. Then, one or more SlaveRobot instances collect initial observations from the environment and use their Emitter to send the data to the SupervisorRobot. The SupervisorRobot, through its Receiver, gathers these observations and passes them to the RLModel instance. Additionally, the SupervisorRobot can augment the observations with extra information from the Webots Supervisor class, such as distances, absolute positions, and object speeds, which are inaccessible to the SlaveRobot through its sensors.

The RLModel generates an action based on these observations, which is transmitted back to the SlaveRobot using the Emitter. The SlaveRobot then executes the action through its actuators. This process is repeated iteratively until the SlaveRobot achieves its objective or a termination condition, such as reaching the maximum episode count or violating predefined constraints, is met.

Following a significant drawback of the second RL design pattern is its reliance on the SupervisorRobot class, which inherits from the Webots Supervisor class. While this inheritance simplifies the implementation of methods such as reset()—intended to reset the environment to its initial state—it renders the design impractical for real-world applications. The functionality provided by the Supervisor class violates physical constraints inherent in real-world scenarios, making it infeasible to transfer these capabilities from simulation to reality. This limitation underscores the intermediate nature of virtual robot simulations, which ultimately serve as a stepping stone toward physical implementations of RL systems.

To overcome this constraint, we propose a third RL design pattern that eliminates all references to the Webots Supervisor class within the simulation code. Instead of utilizing a SupervisorRobot class instance, this pattern introduces the concept of an external server, referred to as the RobotServer, which is accessed through internet-based communication, such as URL links. As shown in Figure 3 (UML class diagram), the RobotServer handles RL tasks for robots lacking the computational capacity to process complex tasks, such as object detection during visual processing.

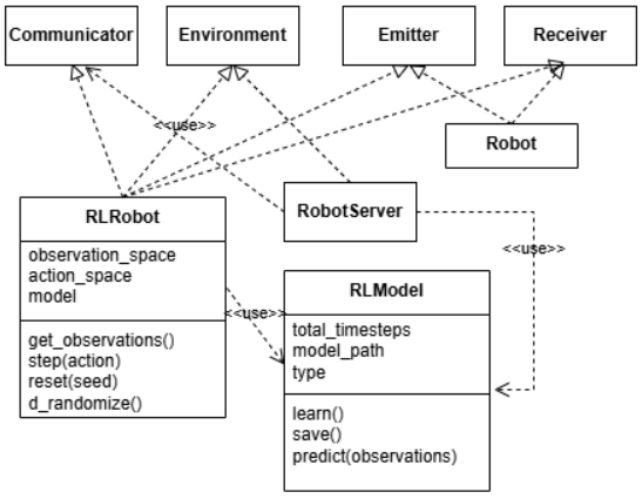

Figure 3. Proposed design pattern to facilitate the transfer of reinforcement learning models from virtual simulations to real-world applications.

Additionally, this design pattern introduces two types of robots: (1) RLRobot instances, which possess the computational power to perform RL tasks locally within the simulation, and (2) Robot instances, which rely solely on their sensors and actuators. These Robot instances communicate with the RobotServer via the Communicator class over the internet or interact with other robots using the Emitter and Receiver classes.

Crucially, to enhance the robustness of policies learned in the virtual environment and facilitate transfer to real-world robots (addressing the sim-to-real gap), domain randomization is explicitly incorporated during the training of RLRobot instances. Our implementation, detailed in the accompanying GitHub project, employs two primary randomization techniques. First, during the initial phase of an episode, random actions are executed for a randomized time window, ensuring the robot starts learning from a diverse set of initial states. Second, random actions are probabilistically injected throughout the training process. This is managed within the RLRobot class via a dedicated method, d_randomize(), as indicated in Figure 3. This method is called at each training step and uses a probabilistic check to decide whether to introduce a random action, thereby simulating unexpected perturbations or variations. The specific parameters controlling the nature and frequency of these random actions (e.g., action selection, duration, probability) are configurable by the DRL engineer. This allows tuning of the randomization level to ensure it promotes robustness without introducing excessive noise that might destabilize learning or prevent convergence due to physical constraints. This overall approach ensures a more realistic alignment with physical constraints while maintaining the flexibility required for effective RL experimentation and robust policy development.

4. Process of Digital Twin Creation

The concept of a digital twin has become increasingly prominent, particularly in the field of industrial automation. Figure 1 illustrates the digital twin of the UR5e cobot, which was developed as part of this research project. In this section, we outline a generalized process and method developed during the study. This process is intended to guide both companies and researchers in creating digital twins for robots that require testing using reinforcement learning techniques.

The creation of a digital twin begins with designing an RL-compatible environment, following standard APIs like OpenAI Gym to ensure compatibility with RL libraries such as Stable-Baselines3 and TensorFlow Agents [29]. The RL agent perceives the environment through simulated sensor data, selects actions from a predefined action space, and receives rewards based on task performance. The reward structure is designed to align with training objectives, guiding the agent toward optimal behavior.

To enhance real-world applicability, domain randomization introduces variations in textures, lighting, and physical parameters, improving generalization. After training, learned policies are validated in simulation and refined for real-world deployment. This approach accelerates robotic development while reducing costs and risks.

In this study, Webots is integrated with Stable-Baselines3 to provide a structured RL testing environment. Stable-Baselines3 offers scalable RL algorithms, facilitating efficient communication between the agent and simulation. This integration supports iterative optimization, task design, and real-world readiness. Figure 4 (UML sequence diagram) presents the workflow for developing and deploying a digital twin in a production environment, involving key roles such as the Production Company, Simulation Engineer, and RL Engineer. The following section details the steps of this process.

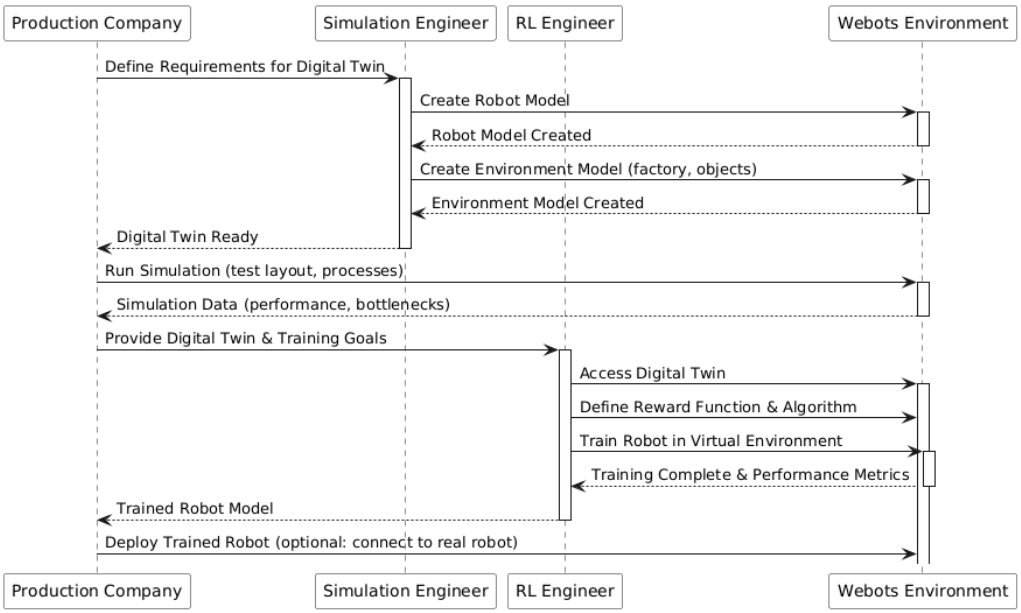

Figure 4. Sequence diagram of the generalized digital twin creation process.

1. Defining Requirements for the Digital Twin: The Production Company first defines the digital twin's requirements, which are communicated to the Simulation Engineer. This includes specifying the robot's tasks (e.g., pick-and-place, welding), operational constraints (speed, payload, workspace limits), and necessary precision. Key data requirements for reinforcement learning include CAD models, sensor data, and historical operational data. Data fusion integrates information from sources like robot controllers and industrial protocols. Effective data management ensures accuracy and reliability, enabling the digital twin to replicate real-world conditions for training and validation. The following Table 1 summarizes these data requirements.
2. Creating the Detailed Robot and Environmental Model in Webots: The Simulation Engineer develops a detailed robot and environment model within Webots. The robot model includes kinematics, dynamics, actuators, sensors, and an end-effector, ensuring accurate simulation of motion and interactions. The environment model replicates the production layout, including objects, obstacles, and sensor placements. Balancing model fidelity and computational efficiency is essential. While high-detail models improve realism, they increase computational cost. The appropriate level of detail depends on task complexity, ensuring effective reinforcement learning training without unnecessary resource overhead.
3. Simulation and Performance Bottleneck Identification: Once the digital twin is created in Webots, the Production Company runs simulations to test different layouts, optimize workflows, and identify bottlenecks. By analyzing robot performance under varying conditions, such as speed, load, and potential failure scenarios, engineers can detect inefficiencies, reachability issues, and collision risks before physical deployment. Simulation data, including cycle times, trajectory analysis, and energy consumption, provide insights for refining system design and improving operational efficiency. This virtual testing approach reduces costs and minimizes disruptions to the production process.
4. Formulating Training Goals for the Reinforcement Learning Engineer: The Production Company defines training objectives based on simulation analysis and provides the digital twin to the RL Engineer. These objectives specify tasks, such as pick-and-place or path planning, along with performance metrics like success rate and cycle time.

Constraints, including safety limits and operational boundaries, are established to ensure feasibility in real-world deployment.

5. Reinforcement Learning Training with the Digital Twin: The RL Engineer utilizes the digital twin in Webots to define the reward function, select training algorithms, and train the robot. The reward function guides learning by assigning positive rewards for task completion and penalties for undesired actions. The choice of algorithm, such as soft actor-critic, depends on task complexity and learning stability. Through iterative trial and error, the RL agent refines its policy to maximize cumulative rewards. The digital twin enables safe, cost-effective training, allowing multiple simulations to accelerate learning without risks to physical equipment or production.
6. Delivery of the Trained Robot Model and Performance Metrics: The RL Engineer delivers the trained robot model and performance metrics to the production company. The model, typically a control policy or neural network weights, represents the learned behavior. Performance metrics, such as cumulative reward progression, policy stability, task completion rate, and cycle time, assess training effectiveness. These metrics help determine if the learned policy meets operational requirements before real-world deployment.
7. Deployment of the Trained Robot and Real-World Connection: The production company deploys the trained model on the physical robot, ensuring compatibility between simulation and hardware. The robot then executes tasks autonomously based on the learned policy. Optionally, real-time integration with the digital twin allows continuous monitoring, performance evaluation, and further refinement of the control policy using real-world data.

Table 1. Summary of the key data requirements for a digital twin of a robot designed for reinforcement learning.

Data Category	Specific Data Points	Importance for RL
Robot Model Data	CAD models (geometry, mass, inertia), kinematic and dynamic parameters (joint limits, motor characteristics)	Accurate representation of the robot's physical properties and movement capabilities is crucial for realistic simulation and effective RL training
Environment Data	CAD models of the production line layout, machinery, fixtures, and objects the robot interacts with	Enables the creation of a realistic virtual environment where the robot can learn to interact with its surroundings
Operational Data	Robot controller data (joint positions, velocities, accelerations, torques), sensor data (proximity, force, vision), PLC data, system logs	Provides real-time data on the robot's behavior and the state of the production line, which can be used to validate the digital twin and inform the reward function for RL
Task Definition Data	Specific goals and constraints of the task the robot needs to learn (e.g., target positions, assembly sequences, cycle times)	Defines the objective for the reinforcement learning agent and helps in designing an appropriate reward function
Performance Data	Metrics for evaluating the robot's performance (e.g., success rate, cycle time, energy consumption)	Used to assess the effectiveness of the RL training in the simulation and to evaluate the performance of the deployed robot in the real world
Communication Data	Details of communication protocols and data links between the digital twin, the physical robot, sensors, and other systems (e.g., UR-RTDE, ROS/ROS2)	Enables the transfer of trained models to the physical robot and the potential for real-time synchronization and monitoring

This sequence showcases the importance of collaboration and the utility of tools like Webots and Stable-Baselines3 in developing efficient, well-trained robotic systems. The diagram also highlights how virtual environments, such as Webots, are indispensable for bridging simulation and real-world applications in a controlled and iterative manner.

5. Example Robots and Experiment Results

To evaluate the RL patterns from Section 2 and the digital twin method from Section 3, we selected two Webots environments, as shown in Figure 5. Both environments involve the inverted pendulum problem [30]. The first (Figure 5, left) replicates the Deepbots reinforcement learning framework for Webots [28]. It consists of a cart with a one-meter pole attached via a free hinge, equipped with a sensor to measure the pole's angle. The task requires maintaining the pole in a vertical position by moving the cart forward or backward, using a discrete proximal policy optimization algorithm [31]. The observation space includes the cart's position and velocity, the pole's angle, and its angular velocity. The agent selects between two actions—moving forward or backward—and receives a reward of +1 per step, including the termination step. Episodes terminate after 1950 steps or when the pole falls or the cart moves beyond ± 0.4 m. A task is considered solved if the agent achieves an average score above 1950 over 100 consecutive episodes.

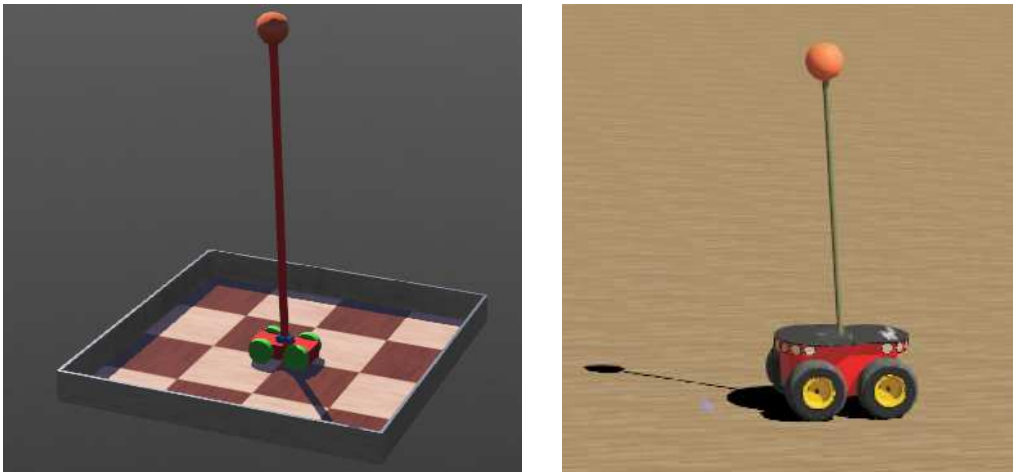

Figure 5. Two Webots environments used to test the methods presented in this paper.

We used this environment to validate the consistency of our framework against results from [28] and to assess the impact of Webots' "speed-up simulation" mode on learning efficiency. The PPO agent, implemented with a two-layer neural network (10 ReLU neurons per layer), successfully solved the problem within approximately 3.5 h of simulated time, consistent with prior work. However, running the simulation in "speed-up" mode reduced execution time to less than 10 min without affecting performance. The learning curve (Figure 6, left) confirms alignment with previous results.

The second environment utilizes a Webots model of the Pioneer 3-AT, a four-wheel, skid-steer robot. This platform is selected for its suitability in reinforcement learning experiments due to its well-defined action space (skid-steer drive with motor control), sensor availability (wheel encoders), and robust physical characteristics (12 kg weight, 0.7 m/s max speed, 35% max traversable grade). The robot's microcontroller and I/O capabilities, including digital and analog inputs, allow for diverse sensor integration and control. These features facilitate the development and testing of RL algorithms for tasks involving navigation and manipulation in varied terrains.

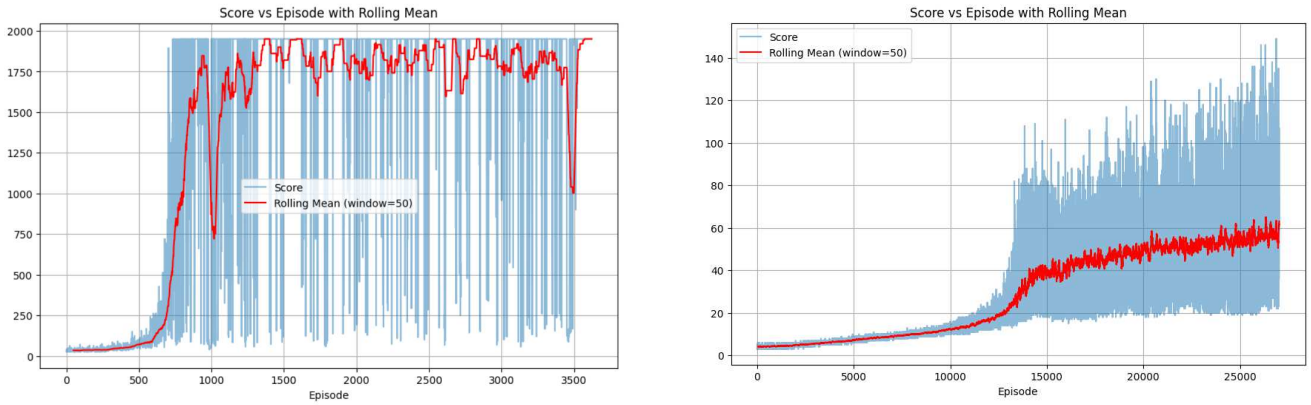

Figure 6. The learning curves for both inverted pendulum environments.

The Pioneer 3-AT Webots model closely replicates its physical counterpart. This study evaluates whether an RL algorithm can achieve stable inverted pendulum control, similar to the toy cart example. Figure 6 (right) shows the learning curve over the first 1 million steps, where the model required approximately 30,000 training episodes but achieved only around 60 cumulative rewards per episode, indicating slower learning compared to the simpler system. To test long-term learning potential, we extended training to 32 million steps, randomizing the initial pendulum angle for each episode to enhance realism. We observed faster convergence when initial conditions matched the toy cart experiment, but by randomizing the initial pendulum angle in each episode, we noticed a significant drop in learning curve convergence. Figure 7 presents results using the trained model, where the average cumulative reward reached 500. However, occasional low rewards persisted, highlighting the challenge of RL in complex environments and the potential for errors in learned policies.

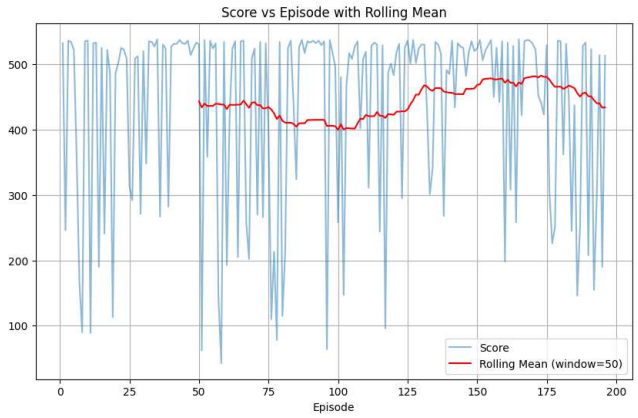

Figure 7. The learning curves of Pioneer 3-AT after 32 million steps.

6. Conclusions

This study introduced an open-source system and novel reinforcement learning design patterns for training and evaluating robotic models in Webots. By integrating reinforcement learning frameworks with a structured digital twin methodology, we demonstrated an efficient approach for simulating and optimizing robot behavior before deployment in real-world environments.

We analyzed three RL design patterns, highlighting the limitations of traditional Supervisor-based implementations and proposing an alternative that removes reliance on Webots' Supervisor class. This new approach improves transferability by utilizing an external RobotServer for task processing, ensuring a more realistic framework that aligns

with physical constraints. Our results indicate that this methodology facilitates seamless adaptation from virtual training to real-world execution.

Furthermore, we demonstrated the digital twin process, detailing the steps required for creating an RL-compatible simulation environment. Through domain randomization and iterative training, the digital twin enhances the robustness of RL models, allowing them to generalize across varying conditions. The integration of Webots with Stable-Baselines3 proved effective for structured RL experimentation, supporting scalable and efficient learning.

Our experiments with the inverted pendulum and Pioneer 3-AT robot models provided insights into the impact of environment complexity on RL training efficiency. While the toy cart example achieved rapid convergence, the more realistic Pioneer 3-AT scenario required significantly longer training durations. The introduction of randomized initial conditions further slowed convergence, emphasizing the challenges of RL in complex systems. Despite prolonged training, occasional low rewards persisted, indicating potential areas for further refinement in RL methodologies for real-world applications. Future work will focus on refining RL algorithms to improve learning efficiency in complex environments and expanding the digital twin framework to support additional robotic applications.

Author Contributions: Conceptualization, A.L., A.Š. and D.M.; methodology, A.L., A.Š. and D.M.; software, A.L. and A.Š. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The New WebotsRL project code and data are available online at https://github.com/aalgirdas/WebotsRL (accessed on 19 March 2025).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ayala, A.; Cruz, F.; Campos, D.; Rubio, R.; Fernandes, B.; Dazeley, R. A comparison of humanoid robot simulators: A quantitative approach. In Proceedings of the Joint IEEE 10th International Conference on Development and and Learning and Epigenetic Robotics (ICDL-EpiRob), Valparaiso, Chile, 26–30 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–6.
2. Yadav, P.; Mishra, A.; Kim, S. A comprehensive survey on multi-agent reinforcement learning for connected and automated vehicles. *Sensors* **2023**, *23*, 4710. [CrossRef] [PubMed]
3. Sivamayil, K.; Rajasekar, E.; Aljafari, B.; Nikolovski, S.; Vairavasundaram, S.; Vairavasundaram, I. A systematic study on reinforcement learning based applications. *Energies* **2023**, *16*, 1512. [CrossRef]
4. Qian, C.; Ren, H. Deep reinforcement learning in surgical robotics: Enhancing the automation level. In *Handbook of Robotic Surgery*; Academic Press: Cambridge, MA, USA, 2025; pp. 89–102.
5. Liu, W.; Wu, M.; Wan, G.; Xu, M. Digital twin of space environment: Development, challenges, applications, and future outlook. *Remote Sens.* **2024**, *16*, 3023. [CrossRef]
6. Tang, C.; Abbatematteo, B.; Hu, J.; Chandra, R.; Martín-Martín, R.; Stone, P. Deep reinforcement learning for robotics: A survey of real-world successes. *Annu. Rev. Control Robot. Auton. Syst.* **2024**, *8*, 457–469.
7. Kilinc, O.; Montana, G. Reinforcement learning for robotic manipulation using simulated locomotion demonstrations. *Mach. Learn.* **2022**, *111*, 465–486. [CrossRef]
8. Jonban, M.S.; Romeral, L.; Marzband, M.; Abusorrah, A. A reinforcement learning approach using Markov decision processes for battery energy storage control within a smart contract framework. *J. Energy Storage* **2024**, *86*, 111342.
9. Zhang, L.; Shen, L.; Yang, L.; Chen, S.; Yuan, B.; Wang, X.; Tao, D. Penalized proximal policy optimization for safe reinforcement learning. *arXiv* **2022**, arXiv:2205.11814.
10. Talaat, F.M. Effective deep Q-networks (EDQN) strategy for resource allocation based on optimized reinforcement learning algorithm. *Multimed. Tools Appl.* **2022**, *81*, 39945–39961.
11. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. Openai gym. *arXiv* **2016**, arXiv:1606.01540.

12. Todorov, E.; Erez, T.; Tassa, Y. Mujoco: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 5026–5033.
13. Ladosz, P.; Weng, L.; Kim, M.; Oh, H. Exploration in deep reinforcement learning: A survey. *Inf. Fusion* **2022**, *85*, 1–22.
14. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.* **2021**, *22*, 1–8.
15. Liang, E.; Liaw, R.; Nishihara, R.; Moritz, P.; Fox, R.; Gonzalez, J.; Jordan, M.; Stoica, I. Ray rllib: A composable and scalable reinforcement learning library. *arXiv* **2017**, arXiv:1712.09381.
16. Michel, O. Cyberbotics ltd. webots™: Professional mobile robot simulation. *Int. J. Adv. Robot. Syst.* **2004**, *1*, 5.
17. Uslu, E.; Cakmak, F.; Altuntaş, N.; Marangoz, S.; Amasyalı, M.F.; Yavuz, S. An architecture for multi-robot localization and mapping in the Gazebo/Robot Operating System simulation environment. *Simulation* **2017**, *93*, 771–780.
18. Garbev, A.; Atanassov, A. Comparative analysis of RoboDK and robot operating system for solving diagnostics tasks in off-line programming. In Proceedings of the 2020 International Conference Automatics and Informatics (ICAI), Varna, Bulgaria, 1–3 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1–5.
19. Tseeva, F.M.; Shogenova, M.M.; Senov, K.M.; Liana, K.V.; Bozieva, A.M. Comparative Analysis of Two Simulation Environments for Robots, Gazebo, and CoppeliaSim in the Context of Their Use for Teaching Students a Course in Robotic Systems Modeling. In Proceedings of the 2024 International Conference "Quality Management, Transport and Information Security, Information Technologies" (QM&TIS&IT), Nalchik, Russia, 23–27 September 2024; IEEE: Piscataway, NJ, USA, 2024; pp. 186–189.
20. Gonçalves, P.J.S.; Torres, P.; Alves, C.M.O. Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions. In Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, Castelo Branco, Portugal, 7 May 2009.
21. Metta, G.; Sandini, G.; Vernon, D.; Natale, L.; Nori, F. The iCub humanoid robot: An open platform for research in embodied cognition. In Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, Gaithersburg, MD, USA, 19–21 August 2008; pp. 50–56.
22. Universal Robots. UR5e. 2025. Available online: https://www.universal-robots.com/products/ur5e/ (accessed on 19 March 2025).
23. ABB. IRB 4600 Robot. 2025. Available online: https://new.abb.com/products/robotics/robots/articulated-robots/irb-4600 (accessed on 19 March 2025).
24. Ferigo, D.; Traversaro, S.; Metta, G.; Pucci, D. Gym-ignition: Reproducible robotic simulations for reinforcement learning. In Proceedings of the 2020 IEEE/SICE International Symposium on System Integration (SII), Honolulu, HI, USA, 12–15 January 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 885–890.
25. Zamora, I.; Lopez, N.G.; Vilches, V.M.; Cordero, A.H. Extending the openai gym for robotics: A toolkit for reinforcement learning using ros and gazebo. *arXiv* **2016**, arXiv:1608.05742.
26. Lopez, N.G.; Nuin, Y.L.E.; Moral, E.B.; Juan, L.U.S.; Rueda, A.S.; Vilches, V.M.; Kojcev, R. Gym-gazebo2, a toolkit for reinforcement learning using ROS 2 and Gazebo. *arXiv* **2019**, arXiv:1903.06278.
27. Nvidia: NVIDIA Isaac ROS. 2025. Available online: https://developer.nvidia.com/isaac/ros (accessed on 19 March 2025).
28. Kirtas, M.; Tsampazis, K.; Passalis, N.; Tefas, A. Deepbots: A webots-based deep reinforcement learning framework for robotics. In Proceedings of the Artificial Intelligence Applications and Innovations: 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, 5–7 June 2020; Proceedings, Part II 16; Springer International Publishing: Princeton, NJ, USA, 2020; pp. 64–75.
29. Hafner, D.; Davidson, J.; Vanhoucke, V. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv* **2017**, arXiv:1709.02878.
30. Barto, A.; Sutton, R.; Anderson, C. Neuron like elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.* **1970**, *13*, 834–846.
31. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.