2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)
August 20-24, 2022. Mexico City, Mexico

2022 IEEE 18th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-9042-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASE49997.2022.9926586

FRobs_RL: A Flexible Robotics Reinforcement Learning Library

Jose Manuel Fajardo* [1], Felipe González Roldan[1], Sebastian Realpe[1], Juan D. Hernández[2], Ze Ji[3] and Pedro-F. Cardenas[1]

***Abstract*— Reinforcement learning (RL) has become an interesting topic in robotics applications as it can solve complex problems in specific scenarios. The small amount of RL-tools focused on robotics, plus the lack of features such as easy transfer of simulated environments to real hardware, are obstacles to the widespread use of RL in robotic applications. *FRobs_RL* is a Python library that aims to facilitate the implementation, testing, and deployment of RL algorithms in intelligent robotic applications using robot operating system (ROS), Gazebo, and OpenAI Gym. *FRobs_RL* provides an Application Programming Interface (API) to simplify the creation of RL environments, where users can import a wide variety of robot models as well as different simulated environments. With the *FRobs_RL* library, users do not need to be experts in ROS, Gym, or Gazebo to create a realistic RL application. Using the library, we created and tested two environments containing common robotic tasks; one is a reacher task using a robotic manipulator, and the other is a mapless navigation task using a mobile robot. The library is available in GitHub [1].**

***Index Terms*— Deep Learning Methods, Reinforcement Learning, Software Architecture for Robotic and Automation.**

I. INTRODUCTION

As robots become more popular and prevalent in many applications, including not only sophisticated industrial settings but also unstructured environments such as stores or homes, the problems that robots have to deal with are more complex [1]. Many planning or decision making algorithms that have been studied and tested in structured environments, i.e., environments in which we have complete information, cannot handle more complex real world scenarios which are unstructured and dynamic [2].

An approach that is being explored to enable new robotic applications and has shown great potential for solving intricate control problems is reinforcement learning (RL) [3], especially a variation that uses neural networks known as deep reinforcement learning (DRL). DRL has been used successfully in many fields like the video games industry [4], transportation [5], as well as in robotics. DRL has solved robotics problems related to the control of quadruped robots [6], humanoids [7] and manipulation tasks [8]. DRL can be used in an end-to-end manner [9], where all the data from sensors fed the network, and the network outputs control all actuators. In addition DRL can be combined with other techniques used in robotics such as the operational space control [10] or perception algorithms that use supervised learning [11]. The variety of usages of DRL makes it highly flexible and adaptable to a broad range of applications.

Although RL has proven to be effective in solving complex problems in robotics and could be functional in many applications, it still has issues entering the robotic research community due to many reasons. One of these reasons is that almost all users that employ RL only test their algorithms and implementations in simulation because the interface with a real robot may be time-consuming and can demand the development of additional software and tests [12]. Additionally, even if some researchers with substantial knowledge in robotics are interested in RL and want to implement a DRL-based algorithm in a novel application, they would have to develop a framework to establish the proper software and interfaces between the learning algorithm and the robotic hardware [13]. On the other hand, realistic simulation environment for training is considered crucial to minimize potential problems when deploying trained agents to hardware platforms.

To alleviate some of the problems mentioned above, in this paper, we introduce *FRobs_RL*. This library uses the robot operating system (ROS) [14] middleware to manage the interface between the learning algorithm and the robotic hardware, which is the most widely used middleware and with the largest community [15]. The proposed library uses Gazebo [16] as its simulation environment since many robotics engineers or users have experience using ROS and Gazebo. The Gazebo environment is widely adopted by the robotics community largely due to its embedded physics engine that ensures high accuracy of simulation and reliability. To manage the RL environment implementation, our proposed library uses OpenAI Gym [17]. In addition, the library incorporates stable-baselines3 (SB3) [18] to allow users with limited knowledge in RL to directly use developed state-of-the-art (SOTA) DRL algorithms in their robotics applications.

The remainder of the paper is organized as follows. Section II describes the background of the proposed library, including the basic concepts of RL and its components. Section III describes the main goals of the proposed library, its input and output files, its functionalities and includes a brief example script. In Section IV, two example cases using the proposed library are described. Finally, Section V presents a brief discussion of the advantages and uses of the library, and in Section VI conclusions and future work are

* Corresponding author.
[1] Mechanical and Mechatronics Deparment National University of Colombia, Bogota, Colombia {jmfajardod, fegonzalezro, srealper, pfcardenash}@unal.edu.co
[2] School of Computer Science and Informatics, Cardiff University, UK, HernandezVegaJ@cardiff.ac.uk
[3] School of Engineering, Cardiff University, UK, jiz1@cardiff.ac.uk
[1] https://github.com/jmfajardod/frobs_rl

listed.

II. BACKGROUND

FRobs_RL is primarily built around OpenAI Gym, which is one of the most popular RL libraries. Gym was designed to be a general framework in which any type of RL task can be defined. All tasks in RL can be described as a Markov Decision Process (MDP) [19]. Specifically, in RL, the MDP is modeled with two main components: the agent, which is the system that can be directly controlled, and the environment, which is everything outside the agent, i.e., all that cannot be controlled and is relevant for the current task. The main goal of every RL algorithm is to "discover" which actions best achieve the task based on the state of the environment. The process of evaluating an action is done using a reward signal as an indicator to determine if the action taken gets the agent closer or farther away from the goal of the task [3].

OpenAI Gym has a basic structure in which the user needs to create a Python class, called *env*, in which at least three functions must be defined: an *initialization_function*, a *step_function*, and a *reset_function* [17]. In the initialization function, the user must load all the required variables and define the observation and action spaces. The observation is the information of the environment that the agent can obtain. Instead of the concept of state, the concept of observation is used, as usually the agent can not have complete knowledge of the environment, and the states are generally only partially observable [20]. In the step function, the user inputs the action that the agent must take. This function must return an observation, the reward of the action, and extra information if the user requires it. Finally, the reset function handles if the task meets the termination condition. In RL, the task can be classified into two types based on its length: continuing or episodic. A continuing task is a never-ending process, while an episode task has clear ending states defined.

Although RL has been used in many robotic applications [1], [21], [7], there is not a standard library to define the tasks in a way that the communication between the RL algorithm and a simulator or a real robotic platform can be handled without much software developing. The ease of communication to the simulator has great relevance, as RL approaches generally require vast exploration of the action space, so the training, or at least its first stages, are done by simulation. When the network is trained, it is transferred to the real robot. This simulation to real transfer (Sim2Real) generally means that new software must be written to communicate with the robot, as communication with simulators is often done differently from the actual platform [12].

Previous tools have used ROS, Gazebo, and OpenAI Gym [22], [13], [23]. Nevertheless, they have significant issues. These tools are built as forks of the main OpenAI Gym, which means that all new features of the library or bugs must be continuously merged, and the library can not be installed from the official source. This latter aspect leads to several maintainability issues. Another major problem is the creation or definition of new RL tasks or OpenAI Gym envs. Finally, with the previous tools creating a simple *env* takes a considerable amount of time, and there are not enough functionalities to quickly create an interface between Gym and ROS/Gazebo, which results in the need of developing large pieces of code, which defeats the purpose of a general tool.

To reduce the complexity of integrating an RL algorithm to a robotic task, *FRobs_RL* has a Gym *env* in which all the communication between the algorithm, the robot, and the simulator is handled. The *env* is created using a standard installation of OpenAI Gym, which minimizes the maintainability issues. In the *env*, all the communication between the RL algorithm with Gazebo or a real robotic platform is handled using ROS. Finally, this *env* can be inherited by any task that the user wants to develop by filling similar functions to the ones proposed in the OpenAI Gym library, such as the action, observation, and reward are specific for every task.

In general, after defining the task in a Gym env the user must then use or develop an RL algorithm that uses the actions, observations, and rewards of the env. Previously RL algorithms could only handle problems with a small number of actions or states, mainly because RL heavily relies on the concepts of the environment state and the action taken in that state. When problems had more actions and states, the exploration became exponentially larger and more time-consuming, and the curse of dimensionality became visible [2]. This issue meant that RL was not suitable for extensive problems like the ones seen in robotics. However, in recent years many approximation techniques have been developed. Neural networks are especially good at approximating the observation and actions spaces without a massive memory capacity. Applying a neural network to an RL algorithm resulted in DRL algorithms which can solve complex problems with good performance [24].

To allow the user directly using DRL algorithms in their robotic tasks, *FRobs_RL* includes an API integration of the SB3 library [18], in which many SOTA algorithms are included. Among them are value-based algorithms like DQN [24], policy-based algorithms like PPO [25] or TD3 [26], and actor-critic algorithms like SAC [27].

III. THE *FRobs_RL* LIBRARY

FRobs_RL is intended for research use in both RL algorithms and robotics implementations. Our library aims to reduce the difficulty of implementing an RL algorithm for any ROS-based robotic architecture both in simulation and on real hardware, for users developing new RL algorithms as well as implementing those algorithms in robotics. With these considerations in mind, the library has the following design goals:

1) *ROS Integration*: By using ROS along with popular RL libraries, such as OpenAI Gym and SB3, the implementation or testing of RL algorithms in robotics should require less coding and development effort.
2) *Easy tasks definition*: By providing templates of the files to define a task, which are defined as Gym envs,

the time of task implementation with any robot should decrease significantly.

3) *Integration of SOTA RL algorithms*: By directly integrating algorithms, such as PPO, TD3, SAC, among others, the necessity to write algorithms should be minimized, and the creation of all the required files to train and test a model should decrease.
4) *Direct Sim2Real transfer*: By using the Gazebo simulator and a framework that uses ROS as middleware, the migration from a simulation setting to real robot hardware should be direct and should not need to create more files.

A. Input and Output

As *FRobs_RL* integrates many concepts, such as robotics middleware, RL task definitions, and algorithms, it has several types of files for configuration, logging, and scene or robot hardware definition. The main types of files that *FRobs_RL* uses are the following:

1) *URDF and SDF files*: Due to the use of ROS and Gazebo, robots are described using the URDF format, and simulation scenes are defined using SDF files. *FRobs_RL* contains functions to effortlessly parse and load URDF files into the ROS parameter server with only a few commands within a Python script. The library also contains functions to spawn URDF or SDF in the simulator or initialize Gazebo with a set of scenes directly using Python commands.
2) *YAML files*: *FRobs_RL* has functions to directly load the parameters into the ROS parameter server, whether from a ROS package or a given system path and in different namespaces. Loading the parameters in different namespaces facilitates the training of multiple parallel RL agents, as the library or the user can easily keep track of the states/parameters of each agent.
3) *ROS nodes and launch files*: RL episodes may need to change some system configurations or use different programs. *FRobs_RL* contains functions to directly run or stop ROS nodes or ROS launch files straight from Python, where it is also possible to set different parameters, e.g., the name or namespace of the ROS node or the arguments of the ROS launch. *FRobs_RL* can execute ROS nodes and launch files in a new terminal or inside the same terminal running the Gym env. This feature helps to monitor the ROS logs or close the envs as previous libraries did not properly close all ROS or Gazebo processes.
4) *Logging and RL models*: If the integrated models of SB3 are used, *FRobs_RL* is set to save or display multiple types of logs. By default, it is configured to display the logs in the terminal using stdout and save them using both TensorBoard and CSV formats. As SB3 uses PyTorch , a common deep learning library, the trained models are saved in ZIP files, and the replay buffers are stored in PKL files with the same name as the corresponding model.

B. ROS and Gazebo Integration With Gym

FRobs_RL manages the integration between Gym and ROS/Gazebo using the communication mechanisms provided by ROS, i.e., topics, services, and actions. The communication process is managed using a basic Gym env in the library called **RobotBasicEnv**. Within the **RobotBasicEnv** program, all RL scheme steps are managed, like launching the ROSCore, resetting the simulator or the robot controllers at the end of an episode, the step function of the RL scheme in which the observations are obtained from the environment, and the actions are sent to the robot. Optionally, the user can launch the simulator or spawn the robot directly from this env if the associated parameters are set.

Although the functions for obtaining the observation, the reward, and the action are called in the **RobotBasicEnv** env, the user needs to fill these functions when creating the task environment, as the specific action, observation and reward must be defined for each task.

C. Definition of Gym envs

To define a task in *FRobs_RL* it is recommended that the user creates two files: the **RobotEnv** and the **TaskEnv**, which is a similar approach as previous libraries [23].

1) *RobotEnv*: In this env, the functions related directly to the robot architecture are set, e.g., the parsing and loading of the robot URDF model in the ROS parameter server, the spawn of the robot in Gazebo, and the functions to send actions or receive information from the robot. This class inherits the **RobotBasicEnv** env. Thus, it manages all the RL related communication between ROS, Gazebo, and Gym. Even if the user obtains some information from the robot, e.g., LIDAR scans, odometry, or joint positions, this may not be the entire observation in the RL task. As some additional information from the external environment can be obtained, e.g., the distance from a robot frame to the goal or if the robot has collided with objects.
2) *TaskEnv*: Within this env, all the functions related to the action, observation, reward, and episode end criteria must be implemented. This env inherits all the methods already implemented in the **RobotEnv** class, which facilitates the creation of new tasks using the same robot hardware since the **RobotEnv** env can be reused.

Although the use of the RobotEnv/TaskEnv is suggested, the user can create the gym env in many different ways, but always inheriting the **RobotBasicEnv** class.

D. Use of Integrated RL Algorithms

As *FRobs_RL* integrates the SB3 library, it includes algorithms such as A2C, PPO, SAC, DQN, DDPG, and TD3. To use any of these techniques, the user only needs to import them from *FRobs_RL* and associate the env to the algorithm. Although a default YAML file is located in our library, the user can create a copy of this parameter file and set the parameters as desired.

E. Example environment

A reacher task [28] that uses an ABB IRB 120 was implemented to test the library, this task is explained in a detailed way in Section IV. The entire process needed to create the **RobotEnv**, **TaskEnv**, the RL parameter YAML file and the script needed to train the policy is available in the library documentation [2]. The example shows how to import the robot's URDF model, and how to override the methods of the basic classes to set the desired reward, observation, and action, among others. It is also shown how to modify the YAML file of the TD3 algorithm and how to create a small script, in less than 40 lines of code, to train the policy. The example is available at the resources library [3].

IV. EXAMPLE USE-CASES

Example environments demonstrate the advantages of *FRobs_RL* in the definition of the task, the direct use of SOTA RL algorithms, models saving and loading processes, and the deployment of the trained models. Although only a manipulator and a mobile robot were used in the example environments, any type of robot can be used with *FRobs_RL*.

A. Manipulator Reacher

The reacher task [28] is a typical application of RL algorithms with robotic manipulators, where the manipulator must achieve a spatial position or pose. The reacher task can be viewed as a learning problem where the RL is going to learn the robot Jacobian matrix and achieve a spatial position through multiple iterations. In each iteration, there will be a small joint movement that will take the end-effector closer to the desired goal.

The reacher task can be viewed as a learning problem where the RL is going to learn the robot Jacobian matrix and achieve a spatial position through multiple iterations. In each iteration, small joint movements will take the end-effector closer to the desired goal. The reward function consists of a dense and a sparse components. The dense reward is based on the distance from the end-effector to the goal. The sparse rewards are used when the robot reaches the goal position or when any joint reaches its position limits.

The MoveIt [29] package is used to move the robot, calculate a valid starting joint configuration and a feasible goal position, the latter is important to speed up the training as sending the robot to an infeasible goal would mean that the episode would never end. Figure 1 shows the reacher environment.

The PPO, SAC, and TD3 algorithms were trained in the reacher environment with a real-time factor of 20.0 in Gazebo. The training of all algorithms was made in five rounds of 100k steps each, using the algorithms default parameters contained in SB3. *FRobs_RL* automatically saves the training logs in both Tensorboard and CSV format. The logs show that TD3 outperforms the other algorithms, as it requires fewer steps to learn a behavior with an adequate mean reward and achieve the best success rate. Figure 2 presents the plot of the episode mean reward obtained from the saved logs.

[2] https://frobs-rl.readthedocs.io/en/latest/guide/example_enviroment.html
[3] https://github.com/jmfajardod/frobs_rl_resources

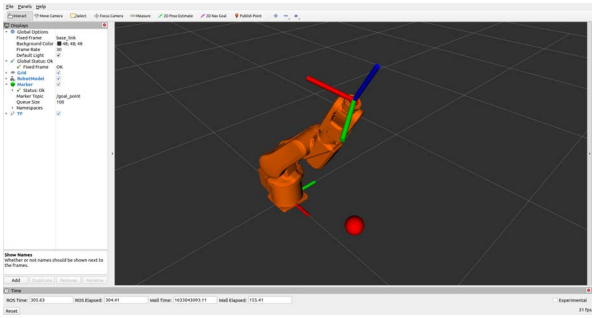

Fig. 1: ABB Reacher environment implemented using *FRobs_RL* visualized in RViz. The goal position is visualized through a red sphere marker.

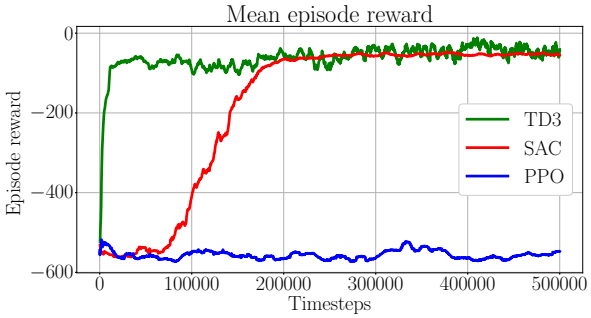

Fig. 2: Example of the episode mean reward plot obtained with saved logs.

B. Mobile robot mapless navigation

The mobile robot mapless navigation problem [30], [31] is where a robot requires to move from an initial position to a desired goal without having prior knowledge of its environment and without the need to construct a map constantly. The previous statements are beneficial compared to map-based navigation as it eliminates the time-consuming need to construct a map, and it reduces the computing resources needed in the robot to run a Simultaneous Localization and Mapping (SLAM) algorithm.

Three mapless navigation environments using a Kobuki mobile robot were implemented. In all the implemented environments, the agent actions are linear and angular velocities. In all environments, the reward function consists of a dense and a sparse component. The dense reward is proportional to the distance from the robot to the goal. The sparse rewards are given when the robot collides with the environment or reaches the goal.

The first environment has a free space with no obstacles in which the observations are only the distance and angle from the robot to the goal. There are eight dynamic obstacles in the second environment with random velocities and trajectories that the robot must avoid using a LIDAR sensor. The second

environment observations consist of the distance and angle from the robot to the goal and LIDAR measurements.

The third environment is a maze with two dynamic obstacles. In this environment, the mapless navigation resembles a local planner used in mobile robotics, as some pre-planned trajectories are provided to the robot. The RL mapless navigation algorithm is fed with points from the pre-planned trajectory as the desired goals; when the robot reaches a point, the next in the trajectory is sent until the last point of the trajectory is detected. The observation vector is the same as the second environment. Figure 3 displays the third environment.

C. Sim2Real Transfer

To test the ease of deployment of a trained RL network using the library, we trained a reacher application using the ABB IRB140 manipulator. As the ABB robot available in the LabSIR does not have a capability of motion streaming, i.e., to stream the joint commands in real-time, we used a motion download strategy, i.e., we pre-planned 100 trajectories with different random valid goal poses and saved the joint commands in a rosbag. After obtaining the rosbag, we formed the trajectories comprising every movement until the goal was reached. The created trajectories were sent to the robot to evaluate the performance of the movements and the RL network. We obtained 100% successful trajectories, i.e., all trajectories reached the desired goal pose in less than 100 movements and did not have self-collisions in their movements. Figure 4 shows the robot setup used to test the trained model [4].

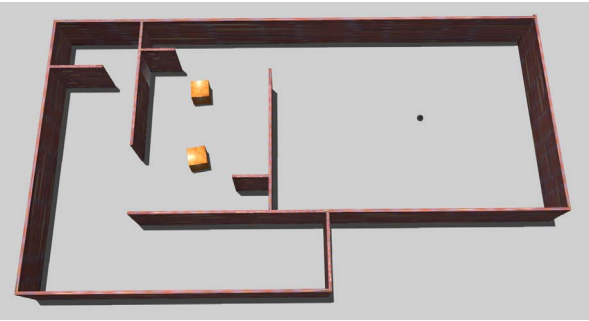

Fig. 3: Kobuki maze environment implemented using *FRobs_RL*.

V. DISCUSSION

We presented the *FRobs_RL* library, which facilitates the implementation and deployment of RL tasks and environments in robotics applications using ROS, Gazebo, OpenAI Gym, and SB3. The core advantage of *FRobs_RL* is the considerable time reduction to implement intelligent robotics applications with RL algorithms in a great variety of tasks and robotic architectures. By directly using ROS and a realistic physics simulator as Gazebo for the training steps, the deployment or Sim2Real transfer of the trained model

[4]Video with the results: https://youtu.be/x6QIPuHeOSo

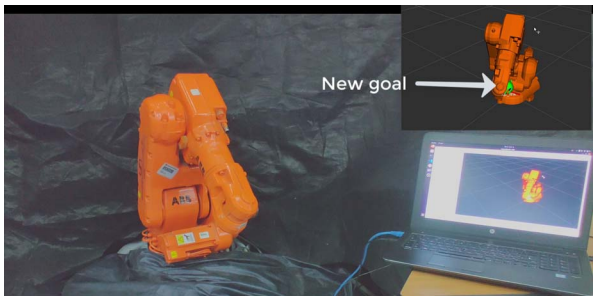

Fig. 4: Setup used to test the Sim2Real capabilities of the library in a reacher task.

is straightforward, which also reduces the effort of the user to implement an RL algorithm in a real-world application. We hope that *FRobs_RL* will enable a broader adoption and testing of RL algorithms in robotics. Not only in research but also on industrial projects so that users obtain another tool to solve complex robotics problems using machine learning techniques such as RL.

A. Comparison

We compared our library to OpenAI Gym, which supports communication with MUJOCO simulator, and PyREP [32], a tool for the CoppeliaSim simulator. Although Gym and MUJOCO are some of the most widespread tools for RL, they are not focused on robotic applications, this results in some difficulties when setting up environments for tasks such as manipulation grasping or mobile robot navigation. PyREP and *FRobs_RL* use specialized robotics simulators, which enable the creation of complex environments and reduce their creation time. Among the compared libraries, *FRobs_RL* is the only one that uses Gazebo, which is the simulator recommended when using RL not only in simulation but also in real life systems [33].

PyREP is a library focused on the simulator control and not on the robot control nor the creation of RL environment; this results in a more time-consuming task to create any RL task even when the robots are supported. Both OpenAI Gym and *FRobs_RL* have methods to directly create an RL environment defining its initialization, the action, the observation, and the reward of the agent.

Finally, between the compared libraries, only *FRobs_RL* has integration with ROS, which enables the use of many packages created in this middleware for robotic applications such as packages for control, odometry, planning, perception, among others. Consequently, the required time to create complex robotic learning environments with different algorithms is reduced. The integration of ROS with *FRobs_RL* also provides a direct way for Sim2Real transfer to compatible robots with ROS, this means that using a trained model through simulation, it can be transferred directly to an actual robot with minimal effort.

VI. CONCLUSIONS AND FUTURE WORK

In this work, we presented the *FRobs_RL* library, a specialized tool in the creation of environments to train

TABLE I: Comparison of robotic oriented RL libraries.

Library	Integration with ROS	Use of robotics simulator	Direct Sim2Real with ROS Compatible robots	Integration of RL algorithms	Support of URDF robot descriptions	Complexity of creation of new enviroments
OpenAI Gym (Mujoco)	No	No	No	No	Limited	Low
PyRep	No	Yes - CoppeliaSim	No	No	Yes	Medium
FRobs_RL	Yes	Yes - Gazebo	Yes	Yes	Yes	Low

reinforcement learning models applied in robotics and their later deployment in real hardware. *FRobs_RL* has integration with ROS, which allows an easy and straightforward transfer process from Gazebo to actual robots. The developed library also features an Application Programming Interface (API) to stable-baselines3 that facilitates the usage of SOTA RL algorithms.

FRobs_RL was tested in two typical robotics applications, a reacher task with an industrial manipulator arm and a mapless navigation task with a mobile robot. The reacher task model was deployed to an actual robot showing the direct and easy transfer from a simulation to an actual setup using the library. Although the library currently supports different manipulator arms and a mobile robot, its resources library needs to be populated with more models.

Another pending feature is the implementation of other tasks, e.g., navigation, perception using depth maps, among others, that would include different robotic algorithms already deployed in ROS, which would further demonstrate the value of the integration of the library with the middleware ROS. Even though a feature comparison was carried out between *FRobs_RL* and other RL frameworks, in the future, a quantitative comparison could offer accurate data about the time performance *FRobs_RL* when training RL models.

REFERENCES

[1] J. Kindle, F. Furrer, T. Novkovic, J. J. Chung, R. Siegwart, and J. I. Nieto, "Whole-body control of a mobile manipulator using end-to-end reinforcement learning," *CoRR*, vol. abs/2003.02637, 2020.

[2] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," *CoRR*, vol. abs/1709.10087, 2017.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, sep 2013.

[4] A. G. Barto, P. S. Thomas, and R. S. Sutton, "Some recent applications of reinforcement learning," in *Workshop on Adaptive and Learning Systems*, 2017, p. 6.

[5] Y. Li, "Reinforcement learning applications," *CoRR*, vol. abs/1908.06973, 2019.

[6] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *CoRR*, vol. abs/1812.11103, 2018.

[7] D. Kim, J. Lee, and L. Sentis, "Robust dynamic locomotion via reinforcement learning and novel whole body controller," *CoRR*, vol. abs/1708.02205, 2017.

[8] P. Chen and W. Lu, "Deep reinforcement learning based moving object grasping," *Information Sciences*, vol. 565, pp. 62–76, jul 2021.

[9] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, "End-to-end robotic reinforcement learning without reward engineering," *CoRR*, vol. abs/1904.07854, 2019.

[10] M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer, "Learning to compose hierarchical object-centric controllers for robotic manipulation," *CoRR*, vol. abs/2011.04627, 2020.

[11] W. Guo, G. Dong, C. Chen, and M. Li, "Learning Pushing Skills Using Object Detection and Deep Reinforcement Learning," *Proceedings of 2019 IEEE International Conference on Mechatronics and Automation, ICMA 2019*, pp. 469–474, aug 2019.

[12] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," *CoRR*, vol. abs/2009.13303, 2020.

[13] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the openai gym for robotics: a toolkit for reinforcement learning using ROS and gazebo," *CoRR*, vol. abs/1608.05742, 2016.

[14] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," in *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, may 2009.

[15] E. Tsardoulias and P. Mitkas, "Robotic frameworks, architectures and middleware comparison," 2017.

[16] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154, 2004.

[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016.

[18] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," https://github.com/DLR-RM/stable-baselines3, 2019.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[20] M. T. J. Spaan, "Partially observable markov decision processes," *Adaptation, Learning, and Optimization*, vol. 12, pp. 387–414, 2012.

[21] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning," *CoRR*, vol. abs/1909.08399, 2019.

[22] A. Martínez-Tenor, J. A. Fernández-Madrigal, A. Cruz-Martín, and J. González-Jiménez, "Towards a common implementation of reinforcement learning for multiple robotic tasks," *Expert Systems with Applications*, vol. 100, pp. 246–259, jun 2018.

[23] A. Ezquerro, M. A. Rodriguez, and R. Tellez, "Openai_ros package," 2018. [Online]. Available: https://bitbucket.org/theconstructcore/openai_ros/src/kinetic-devel/

[24] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[26] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018.

[27] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *CoRR*, vol. abs/1812.05905, 2018.

[28] P. Aumjaud, D. McAuliffe, F. J. Rodríguez Lera, and P. Cardiff, "rl_reach: Reproducible reinforcement learning experiments for robotic reaching tasks," *Software Impacts*, vol. 8, p. 100061, may 2021.

[29] I. A. Sucan and S. Chitta, "Moveit," 2011. [Online]. Available: https://moveit.ros.org/

[30] J. C. de Jesus et al., "Soft actor-critic for navigation of mobile robots," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 102, pp. 1–11, 5 2021.

[31] N. Duo, Q. Wang, Q. Lv, H. Wei, and P. Zhang, "A deep reinforcement learning based mapless navigation algorithm using continuous actions," *Proceedings - 2019 International Conference on Robots and Intelligent System, ICRIS 2019*, pp. 63–68, 6 2019.

[32] S. James, M. Freese, and A. J. Davison, "Pyrep: Bringing V-REP to deep robot learning," *CoRR*, vol. abs/1906.11176, 2019.

[33] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing popular simulation environments in the scope of robotics and reinforcement learning," 2021.