MoveRL: To A Safer Robotic Reinforcement Learning Environment

Liu, Gaoyuan; De Winter, Joris; Vanderborght, Bram; Nowe, Ann; Steckelmacher, Denis

Published in:
The 33rd Benelux Conference on Artificial Intelligence and the 30th Belgian Dutch Conference on Machine Learning (BNAIC/BENELEARN 2021)

DOI:
10.1007/978-3-030-93842-0_14

Publication date:
2022

MoveRL: To A Safer Robotic Reinforcement Learning Environment

Gaoyuan Liu, Joris De Winter, Bram Vanderborght, Ann Nowé, and Denis Steckelmacher

Vrije Universiteit Brussel
gaoyuan.liu@vub.be

Abstract. The deployment of Reinforcement Learning (RL) on physical robots still stumbles on several challenges, such as sample-efficiency, safety, reproducibility, cost, and software platforms. In this paper, we introduce MoveRL, an environment that exposes a standard OpenAI Gym interface, and allows any off-the-shelf RL agent to control a robot built on ROS, the Robot OS. ROS is the standard abstraction layer used by roboticists, and allows to observe and control both simulated and physical robots. By providing a bridge between the Gym and ROS, our environment allows an easy evaluation of RL algorithms in highly-accurate simulators, or real-world robots, without any change of software. In addition to a Gym-ROS bridge, our environment also leverages MoveIt, a state-of-the-art collision-aware robot motion planner, to prevent the RL agent from executing actions that would lead to a collision. Our experimental results show that a standard PPO agent is able to control a simulated commercial robot arm in an environment with moving obstacles, while almost perfectly avoiding collisions even in the early stages of learning. We also show that the use of MoveIt slightly increases the sample-efficiency of the RL agent. Combined, these results show that RL on robots is possible in a safe way, and that it is possible to leverage state-of-the-art robotic techniques to improve how an RL agent learns. We hope that our environment will allow more (future) RL algorithms to be evaluated on commercial robotic tasks.
Github repository: https://github.com/Gaoyuan-Liu/MoveRL

1 Introduction

Reinforcement Learning is a Machine Learning approach that allows an agent to learn what action to execute in which situation, to maximize a scalar reward [14]. On robots, Reinforcement Learning has the potential of allowing to learn near-optimal controllers on challenging tasks, on which classical methods such as planning are not applicable, for instance due to the unavailability of a good model, or high stochasticity or unexpected events around the robot. However, in practice, Reinforcement Learning is not often used on robots.

Several challenges currently prevent the use of Reinforcement Learning on robots, such as safety, sample-efficiency, the ease of implementation of RL on

robots from a software perspective, and the trust designers must have in RL to use it. In this paper, we propose a new OpenAI Gym [3] environment that allows real-world robotic experiments to be performed, addressing these two challenges:

Software compatibility with robots. Existing Reinforcement Learning environments that have robots in mind, such as the Gym Mujoco environments [3], the DeepMind control suite [26], or PyBullet environments [29], implement environment-specific robotic arms or bodies (not industry-standard robots), using embedded simulators (not a connection to an industry-standard simulator). As such, these environments can be used to show that RL works on robots in theory, but do not help implementing RL on a real-world robot. Our main contribution, MoveRL, interfaces an RL agent with the Gym API to ROS, the Robot OS, used by industry-standard simulators (such as Gazebo) and robots. This allows direct learning on the robot, or easy transfer of an agent learned in simulation to a physical robot (without having to re-implement anything).

Safety The Robot OS comes with many packages that allow to build complete robotic systems, with planning, collision avoidance, simultaneous localization and mapping, ... In this paper, we use MoveIt [4] to transform an action selected by an RL agent into a motion plan for a robot, while avoiding collisions with obstacles. MoveIt gets its knowledge about obstacles from the ROS network, which means that it is inherently compatible with simulators (that know where obstacles are) and depth cameras, that produce the same information on real robots [11].

Our empirical results in the Gazebo simulator, using a simulated real-world robot (the Franka Emika Panda manipulator), show that combining an unmodified implementation of PPO [24] from the stable-baselines3 [22] with a ROS environment is possible, and that leveraging MoveIt for action execution allows to prevent almost every collision, even in the early stages of learning.

2 Notations

The Reinforcement Learning literature considers an agent that executes actions in a Markov Decision Process, defined by a tuple $\langle S, A, R, T, \mu_0, \gamma \rangle$. S is the state space, that can be either discrete or continuous. In this paper, we consider continuous state-spaces, in which each state is a vector of several real values. A is the action space. In this paper, we consider a continuous action space, in which each action is a vector of real values. $R(s, a, s')$ is the reward function, that produces a single real value after a transition from state s to s', caused by the execution of action a. $T(s, a)$ is the transition function, that maps a state and an action to a new state. μ_0 is the initial state distribution, that defines in which state the agent may start an episode, and $\gamma < 1$ is a real value, the discount factor.

Most Reinforcement Learning literature follows the notation described above. However, roboticians use other notations, that appear in the literature related to

ROS and MoveIt, and that we sometimes use in this paper when interfacing with these components. We provide a brief summary of the differences of notation in the table below:

RL	Meaning	Motion Planning
s	State (observation)	q (if joint angles) p (if end-effector position)
a	Action	q_i (target joint angles)
r	Reward	r or c (cost $c = -r$)

3 Related Work

Our main contribution allows a Reinforcement Learning agent to interface with the Robot OS, for easy control of simulated or physical robots, with the use of a motion planner to ensure safety. We now provide a related work review of other approaches at robotic environments for Reinforcement Learning, techniques that allow to make a Reinforcement Learning agent safer, and motion planning libraries.

3.1 RL Robotic Environment

To tackle various challenges in robot RL [15], numerous robotic RL environments are developed with different platforms. A brief survey of robotic RL environment frameworks can be found in [13] and [9]. Each work emphasises specific merits with regards to particular issues. In this paper, we only review frameworks which are wildly accepted as benchmark, and particularly, we discuss how they consider safety when learning.

Mujoco To improve the reproducibility in RL robotics research, SURREAL [8] is built on the MuJoCo simulation environment and physics engine [27]. Mujoco is widely used for RL environments, and frameworks with the same physics engine can be found in [21, 31, 1]. Such projects usually focus on theoretical RL research, and lack compatibility with robotic software such as the Robot OS. Moreover, the use of Mujoco requires a license id (free for academic purposes, paid otherwise).

PyBullet PyBullet is an open-source physics engine, used by [30] to implement several Reinforcement Learning environments in simulated 3D spaces. These environments allow the agent to control every joint of the robots, but do not provide any safety mechanism or collision avoidance. An RL environment for a quadcopter is developed with PyBullet by [19], but collision avoidance is not considered, even though it appears crucial for a quadcopter.

Gazebo Gazebo is an open-source simulator with a graphical interface, and an interface to the Robot OS ROS [6]. A Gym environment for interacting with Gazebo is proposed in [16], but collisions are allowed to happen in the simulator, which makes replacing the simulator with a real-world robot impractical. However, because Gazebo and ROS have large communities that developed many industry-proven tools, we base our main contribution on these two pieces of software, and add MoveIt for collision avoidance.

3.2 Safe Reinforcement Learning

RL safety is normally defined as a mechanism which can ensure reasonable system performance and/or respect safety constraints during the training or validation processes.

Definition and Survey RL safety approaches can be categorized in two classes: tuning the optimization criterion of the algorithm to encourage safe behavior, and directly intervening on the exploration of the agent to prevent unsafe actions from being executed.

With the optimization approach, maximizing the long-term reward can generate statistically safer policy, but does not necessarily avoid the rare occurrences of damage, neither ensures safety during training. The exploration approach provides a *shielding* mechanism that modifies or prevents unsafe actions [10]. Several approaches to Safe RL, belonging to the two classes described above, are reviewed in [28].

Safe exploration In this paper, we focus our attention onto Safe RL approaches that consider physical issues, and in particular prevent physical damage. In a danger-sensitive learning environment, such as robotics, the importance of damage avoidance is higher than obtaining high rewards. Therefore, a shielding layer maintaining zero-constraint-violations throughout whole learning process is necessary.

[23] introduce *safe exploration*, more specifically Constrained Reinforcement Learning, and address two challenges: 1) the difficulty of designing reward functions that nicely balance punishing unsafe actions, and encouraging the agent to learn the desired skill; 2) the fact that eventually learning the optimal safe policy does not guarantee that no unsafe action has been performed while learning.

[5] consider that some states can be identified as unsafe, and propose a method to avoid these states. In [20], a safety layer is applied in a real-robot system, the safety layer modifies possibly risky actions to the closest valid alternatives which satisfy safety constrains, but such constraints are difficult to define especially when the environment is noisy or uncertain. Similar structured safety guarantee is also utilized in [2].

When positioning this paper in relation to existing work, it is important to note that existing work focuses on preventing the execution of specific unsafe actions, and let the designer define what an unsafe action is. In this paper, we

use MoveIt to automatically detect what would be unsafe actions, freeing the designer from this task. Moreover, existing work considers that moving from a safe state to a safe state is safe. This is not the case in practice, as we explain in Section 4.7: the path between two safe states may go through a wall, and therefore be unsafe. Our contribution detects these unsafe actions.

3.3 Path Planning

Path planning is one of the most fundamental problems in autonomous robotics, particularly, in the scenarios where robots have to execute tasks in an environment with obstacles. Sampling-based methods offer a solution to overcome the complexity of deterministic robot planning algorithms for a robots with many degrees of freedom (many joints). A comprehensive survey can be found in [7]. An open-source library for sampling-based motion planning OMPL (Open Motion Planning Library) is proposed by [25], and is integrated in an open-source framework, MoveIt!, that offers an state of the art path planning based on several well-known libraries. MoveIt also integrates implementations of useful robotics functions, such 3D perception, kinematics calculation and control [1].

4 Contribution

Our main contribution is a Gym environment, that allows to interface unmodified Reinforcement Learning agents written in Python with a simulated or physical robot exposed on ROS-Noetic, the Robot OS, a collection of libraries and network protocols that allow components of robotic systems (hardware, software, planners, ...) to communicate in an industry-standard way. We also leverage MoveIt, a state-of-the-art motion planner, for efficient action execution and collision avoidance.

The general architecture of our main contribution, MoveRL, is depicted in Figure 1. Every time-step, the agent receives an observation and reward from the environment, and selects a *raw action*, a desired position of the robot, not yet guaranteed safe. The raw action passes through a collision monitor, based on MoveIt, that observes the current position of obstacles and verifies the action. Verifying the action relies on the possibility to simulate its outcome, which is possible on physical robots by using *co-simulation* (a simulated version of the robot runs in parallel with the physical robot, an approach very common in industrial robotics and transparently supported by ROS). In this paper, we only use pure simulation, and leave co-simulation with a physical robot to future work.

If the raw action will cause a collision, it is discarded, leading to no movement of the robot for this time-step, and a punishment given to the Reinforcement Learning agent. If the raw action is safe, the planner computes a collision-free path to guarantee the safety during execution. Our action shielding mechanism ensures that no collision happens when the agent moves.

[1] https://MoveIt.ros.org/

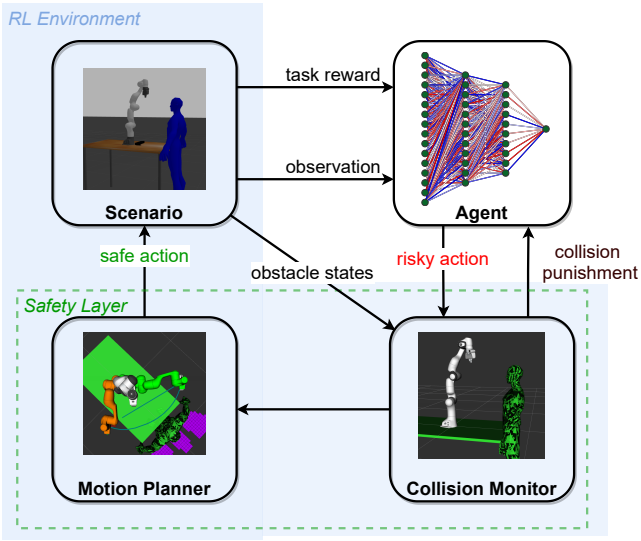


Fig. 1. Our MoveRL framework. Our safety layer leveraging the MoveIt motion planner contains 2 modules: 1) a collision monitor, that detects actions that lead to collisions, and 2) a motion planner, that plans a collision-free path to reach the target locations encoded in (previously-identified) safe actions.

We now detail all the software components of our proposed MoveRL. Our implementation is available at https://github.com/Gaoyuan-Liu/MoveRL.

4.1 The Gym environment

To be compatible with standard RL algorithm implementations, such as in Stable Baseline 3 [22], our contribution needs to be implemented as an OpenAI Gym [3] environment. A Gym environment is a Python class that contains attributes that describe its state and action spaces (both `Box` in our case), and methods that allow actions to be executed in the environment. The *reset* method resets the environment to an initial state, and returns the first observation of the episode. The *step* method takes an *action* as input, passes to ROS and MoveIt for safe execution, and produces a new state (*observation*) and *reward*. We describe all these steps in more detail later. A *done* signal is also returned by *step*, and allows the environment to choose when an episode should terminate.

4.2 Observation Space

Since we consider kinematics observation, we make an assumption that the position of obstacles can be detected by sensors, or is available in simulation. The observation space contains two parts: the state of the robot, and the state of the obstacles. For the robot state, we developed two environments with two different

kinds of observation: joint angles $\mathbf{q} = [q_1, q_2, ..., q_n]$ for an n degrees-of-freedom robot, and end-effector position and orientation $\mathbf{p} = [p_x^{ee}, p_y^{ee}, p_z^{ee}, o_x^{ee}, o_y^{ee}, o_z^{ee}, o_w^{ee}]$, for tasks in which the robot has an end-effector such as a gripper. For obstacle state, the agent can observe the position and orientation of the obstacles: $[p_x^{obs}, p_y^{obs}, p_z^{obs}, o_x^{obs}, o_y^{obs}, o_z^{obs}, o_w^{obs}]$. Our environment class can adjust the size of state space according to the number of obstacles in the simulation.

Note that the agent observes the position and orientation of the obstacles (cylinders, spheres, rods, cubes), but not their shape or dimensions. This is not a problem, as an RL agent is perfectly able to learn what positions in relation to the center and orientation of an obstacle will translate to negative rewards. So, the agent sorts of learns the shape of the obstacles by feel, and does not need to be provided that information.

4.3 Action Space

Our environment exposes a continuous action space, for which actions are vectors of real values. More precisely, we consider that the action produced by the agent is a target configuration of the robot, so a list of real values that define the angle at which every joint of the robot must be set. Robotics libraries call this set of angles q_i, and the Reinforcement Learning literature calls this a. The action space is constrained by the physical abilities of the robot, with joint position limits $q_{i,\text{limit}}$ and joint velocity limits $\dot{q}_{i,\text{limit}}$.

The physical constraint on the speed of a joint requires careful engineering of how the agent produces an action. Given a time-step duration Δt, we must ensure that the action $q_{i,\text{cmd}}$ produced by the agent, and sent to the environment, differs (in absolute value) from the previous action by at most $\Delta t \cdot \dot{q}_{i,\text{limit}}$, for every element of $q_{i,\text{cmd}}$. We must also ensure that the action $q_{i,\text{cmd}}$ is part of the allowed range of joint angles $[q_{i,\text{min}}, q_{i,\text{max}}]$.

We implement these constraints as follows: the policy of the agent produces the change in joint positions $\Delta q_{i,\text{cmd}}$, instead of the absolute value of the joint positions $q_{i,\text{cmd}}$. Then, we clip $\Delta q_{i,\text{cmd}}$ to $[-\Delta t \dot{q}_{i,\text{limit}}, \Delta t \dot{q}_{i,\text{limit}}]$, produce $q_{i,\text{cmd}} = q_{i,\text{prev timestep}} + \Delta q_{i,\text{cmd}}$, and clip $q_{i,\text{cmd}}$ to the range $[q_{i,\text{min}}, q_{i,\text{max}}]$. This clipped value is sent to the environment, that uses MoveIt to detect and avoid collisions.

4.4 Why do we need sequences of actions?

Most tasks on which we evaluate our framework consist of moving the end effector of a robot to a specific target location. Given the action set described above, it may seem logical that only one action is necessary for that: putting the robot in the pose that puts the end effector at the target location. However, in practice, a sequence of actions is needed for the following reasons:

- The time-step has a fixed duration and the robot cannot move infinitely quickly, so the actions have to progressively bring the robot close to the target location;

– Even if state of the art, MoveIt has difficulties planning motions on long distances, especially when there are concave obstacles in the scene. Reinforcement Learning is particularly useful in this case, as its optimization of the discounted sum of rewards allows the agent to take actions that move away from the goal in the short term, but allow to reach it in the long term.

4.5 Reward Function

The reward function is customized for each specific task, but always consists of the sum of two terms: a task-specific reward and a task-agnostic safety term, $r = r_{\text{task}} + r_{\text{safety}}$.

We describe r_{task} in the next sections. r_{safety} is 0 when an action would cause no collision, and some negative constant when an action is detected as being unsafe (and cancelled). The choice of the constant is described in our experiments, and needs to be large enough that the agent learns to avoid states that can potentially lead to collisions (especially when there are moving obstacles), but not too much, so that the agent does not become too conservative (and learns a policy that remains immobile, for instance).

4.6 Initial States and Termination

Every episode, we initialize the simulated robot to a random pose, to ensure good exploration. The episode terminates when the end effector of the robot reaches a pre-defined goal position, or after 100 time-steps.

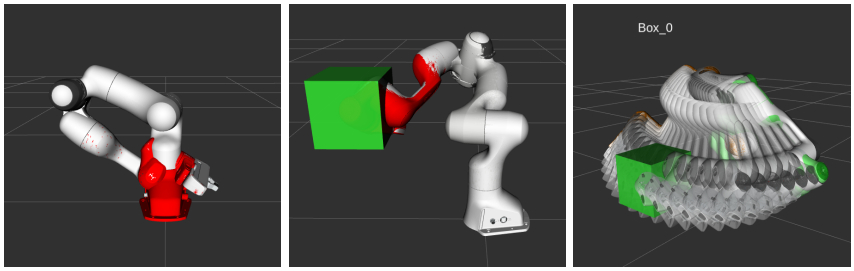
4.7 Safety Guarantee

After having described the different components of our environment (state space, action space, ...), we now discuss the different types of collisions that can be detected by MoveIt, and provide details on how we query MoveIt from a Gym environment. A full description with code would go beyond the page limit of this paper, but the complete source code that we use in our experiments is available on Github (link in the abstract).

Collision Types To comprehensively consider the potential risk during training, we categorize collisions into three types:

(a) self collision: two parts of robot itself collide with each other; (b) pose collision: the commanded configuration $q_{i,\text{cmd}}$ collides with objects in the environment; (c) path collision: the direct path between two configurations contains collisions with objects in the environment. Figure 2 shows examples of these three types of collisions.

Avoiding self-collision and pose collision during learning can be achieved by constrained inverse kinematics and state-validation checking at each time-step. However, the avoiding path collisions is more challenging, and tends to be neglected in the Safe RL literature since it is difficult to do state validation



(a) (b) (c)

Fig. 2. (a) Self collision (b) Pose collision (c) Path collision

checking continuously. Therefore, even when the state for two adjacent steps are safe, the direct path (without planning) can still collide with obstacles. We give an unified solution to avoid the aforementioned three types of collision, which is integrating MoveIt as an safety layer in the RL environment.

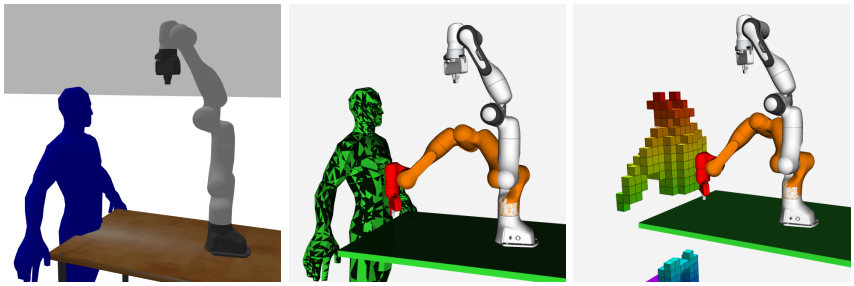
Collision Detection The MoveIt provide package `Planning Scene` allows to manage an abstract representation of the environment surrounding a simulated or physical robot. This environment can contain obstacles of two possible types: scene object and octomap [12].

Scene objects have an explicit shape, such as a 3D mesh or a primitive shape (cylinder, box, sphere). It is used when a coarse obstacle is enough, for instance a big cylinder around a human, to encoder a general area that has to be avoided. An Octomap is built from a depth camera (or produced by a simulator), and allows to precisely measure the presence of an obstacle around the robot, without having to model it. The trade-off between precision and efficiency can be adjusted with the resolution parameter of the octomap.

Once the Planning Scene has been defined (and kept updated by the simulator or sensors, using ROS network messages that the Gym environment does not even need to bother with), MoveIt is able to detect collisions using libraries such as the FCL (Flexible Collision Library) [18].

We stress that the use of ROS allows to transparently interface our Gym environment with many well-regarded robotic packages, Planning Scene being only one. Other packages allow to stream updates to the position of the obstacles from a variety of sensors (and are usually shipped with the sensors), or to visualize various aspects of the scene (for instance, visualizing how a physical robot senses its surrounding). Figure 3 shows that it is possible to visualize a textured 3D render of a scene, along with information about the obstacles in it, and what motion planning has to be performed.

Path Planner It's worth noting that the direct path between two valid poses can still contain collisions, which we term as path collision. To avoid path collisions, a local planner is necessary, and will run every time-step, to produce a



(a) (b) (c)

Fig. 3. The collision detection methods in MoveIt. (a) Gazebo simulation (b) Collision detection with scene object (c) Collision detection with octomap. The white arm indicates current pose, the orange arm indicate the commanded pose, and the red parts indicate the collision links.

full motion from the pose of the robot on one time-step, to its pose at the next time-step.

For each time-step, given a valid action command, a planner plans a collision-free local path based on the present knowledge of the position of obstacles. In order to reduce the time consumption of training and minimize the planning delay, our primary consideration for choosing the most appropriate planner is efficiency and completeness. Therefore, we evaluated planners available in MoveIt by their solving time and path length, which can reflect the planners' efficiency in either planning and execution phases [17].

The planners' time consumption and planned path length on a benchmark task (putting a robotic arm into a desired pose) are shown in Figure 4. We note that our objective is not to identify the absolute best planner, but to provide an informed choice of planner for our Reinforcement Learning experiments. RRT and its derivative show merits in both planning time and path length. Therefore, we choose the RRT planner in our experiments.

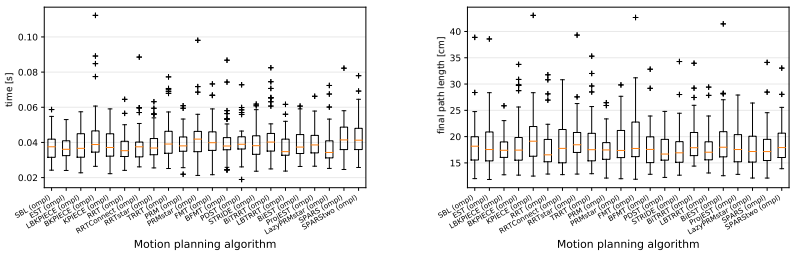


Fig. 4. *Left:* Planner planning-time comparison, *right:* Planner final path length comparison.

Goal-Reaching Task We now define a goal-reaching task, and detail how it is implemented with MoveIt. To achieve the goal-reaching task, we define a dense reward (non-zero whenever there is movement during a time-step), that is proportional to the change in distance between the end-effector's current position and the goal position. The task reward can be formalized as:

$$r_{\text{task}}(s) = \begin{cases} \kappa \Delta d(s) + r_{\text{goal}} & \text{if } s \text{ is close to } s_{\text{goal}} \\ \kappa \Delta d(s) & \text{otherwise} \end{cases}$$

where $r_{\text{task}}(s)$ is the task reward given to the agent when reaching state s, $\Delta d(s)$ is the distance between the end-effector in state s and the target end-effector location, r_{goal} is a fixed positive reward given when the target location is reached, and κ is a weighting constant, allowing to balance r_{task} and r_{safety}. The actual values of r_{goal} and κ are given in the next section.

5 Experiment

While our main contribution is a Gym environment that allows to learn tasks in ROS-based robotic environments with standard Reinforcement Learning algorithms, we also provide experimental results, that show that:

- Our framework, that we call MoveRL, works and actually allows an unmodified PPO agent to learn a task;
- Collisions can indeed be avoided, thanks to MoveIt, which allows simulated robots to be replaced with physical robots if need be.

5.1 Learning Scenarios

In our experiments, the robot learns to fetch the goal point with its end-effector by adjusting 7 joint angles. We consider 3 scenarios around this task, that differ in what kinds of obstacles are around the robot:

1. **Table**: The table holding the robot is the only exterior obstacle, thus self-collisions are considered as the major risks in this scenario.
2. **Human**: The robot and a human worker share a single workspace. The goal point is located between the human and the robot. Self-collisions and pose collisions would be the major risks. The human is modelled with basic shapes such as cylinders.
3. **Case**: The robot has to reach a goal location inside a box/case (walls with an opening on top). Finding how to enter the case is challenging in this task and benefits from the use of Reinforcement Learning. The thin walls of the case lead to possible path collisions (in addition to self-collisions and pose collisions).

The 3 training scenarios are illustrated in figure 5. Our Github repository contains Gazebo world files for all 3 scenarios.

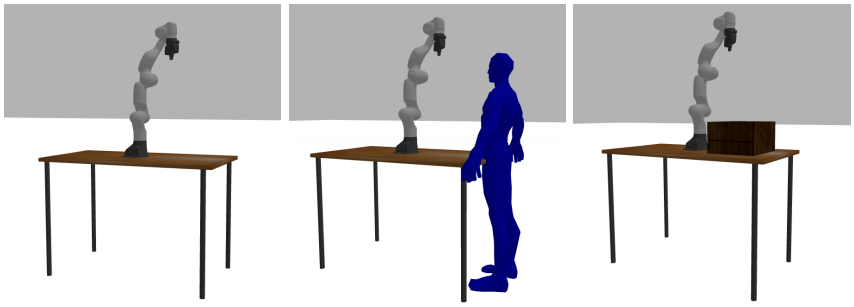


Fig. 5. From left to right: table world, human world and case world, and the first row shows the simulation environment in gazebo while the second row is the state presentation (robot and obstacles) in rviz.

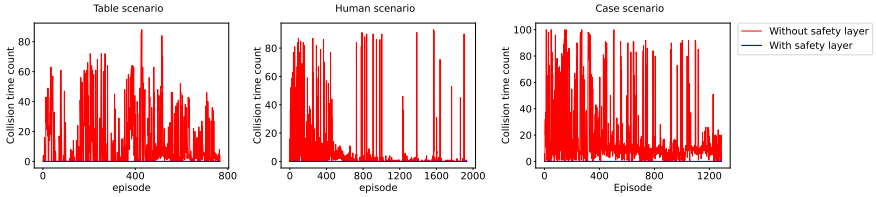


Fig. 6. Number of collisions occurring per episode, with and without collision avoidance with MoveIt. The blue line indicates that our safety layer, based on MoveIt, successfully prevents collisions throughout the learning process.

5.2 Learning Algorithm

We use a Reinforcement Learning agent from the Stable-Baselines 3 [22], that contains a set of reliable RL algorithms including A2C, DDPG, PPO, SAC and TD3. We choose PPO, as it is highly popular, compatible with continuous actions, and has many implementations. In this paper, we focus on showing that RL with ROS is possible, we do not aim at evaluating which RL algorithm performs the best. The hyper-parameters that we use for PPO in our experiments are the default values used Stable-Baselines 3, as of August 23rd, 2021, with the following changes: the policy network is MlpPolicy, the learning rate is 0.0005, the batch size is 200, and the number of steps between policy updates is 100.

5.3 Results

To evaluate our safety layer, we compare how a PPO agent learns with and without our collision avoidance method. We observe that our safety layer successfully prevents collisions, and has no negative impact on sample-efficiency:

Figure 6 shows that enabling our safety layer successfully prevents collisions, and that disabling our safety layer leads to a large amount of collisions.

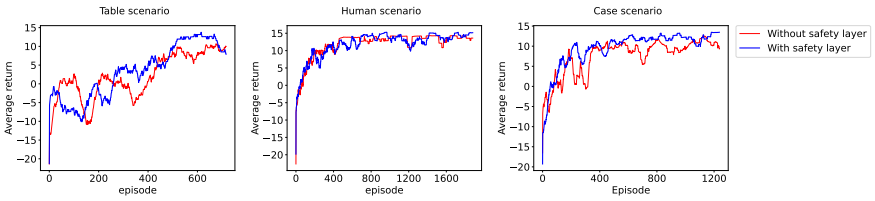


Fig. 7. Learning curves in each learning scenario. We confirm that our safety layer, that successfully prevents collisions, has no negative impact on sample-efficiency (the red and blue curves have the same shape). This shows that safety does not come at the expense of sample-efficiency with our MoveRL framework.

Figure 7 presents the learning curves in our three scenarios, with and without our collision avoidance method. Avoiding collisions does not appear to have any negative impact on the agent, as the learning curves are comparable. If it has any effect, it would be a slight increase in sample-efficiency, as seen in the Case scenario. We are happy with this result, as it shows that safety in Reinforcement Learning does not come (in our case) at the cost of sample-efficiency and final policy quality.

6 Conclusion

In this paper, we presented MoveRL, a Reinforcement Learning Gym environment for robotic manipulators, that builds the widely-used ROS platform for simulated and physical robots. Thanks to the dynamism of the ROS community, advanced algorithms for planning, obstacle detection and collision avoidance are available. We leverage them in our environment to produce a method for safe Reinforcement Learning on robots. Our experiments show that our safety mechanism indeed prevents collisions while an un-modified PPO agent learns a simulated robotic task, and that our method has no negative impact on sample-efficiency.

While the deployment of our method on a physical robot remains as future work, we hope that our new software and method will allow Reinforcement Learning researchers to more easily evaluate their methods on simulated real-world robots (as opposed to unrealistic robots as available in the Gym Mujoco tasks, for instance), and will allow robotic engineers to evaluate Reinforcement Learning for the tasks in which classical planning methods show limitations.

Acknowledgments

The first author is supported by the China Scholarship Council (CSC). The second author is supported by the Flemish Government under the Flemish AI Program (*Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen*).

Bibliography

[1] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on Robot Learning*, pages 1300–1313. PMLR, 2020.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 19(1):18–19, 2012.

[5] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[6] Brian Delhaisse, Leonel Rozo, and Darwin G Caldwell. Pyrobolearn: A python framework for robot learning practitioners. In *Conference on Robot Learning*, pages 1348–1358. PMLR, 2020.

[7] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77, 2014.

[8] Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, pages 767–782. PMLR, 2018.

[9] Diego Ferigo, Silvio Traversaro, Giorgio Metta, and Daniele Pucci. Gym-ignition: Reproducible robotic simulations for reinforcement learning. In *2020 IEEE/SICE International Symposium on System Integration (SII)*, pages 885–890. IEEE, 2020.

[10] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[11] Stefan Grushko, Aleš Vysockỳ, Vyomkesh Kumar Jha, Robert Pastor, Erik Prada, L'ubica Miková, Zdenko Bobovskỳ, Jiří Suder, Zdeněk Zeman, Jakub Milotek, et al. Tuning perception and motion planning parameters for moveit! framework. 2020.

[12] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.

[13] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[14] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[15] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[16] Nestor Gonzalez Lopez, Yue Leire Erro Nuin, Elias Barba Moral, Lander Usategui San Juan, Alejandro Solano Rueda, Víctor Mayoral Vilches, and Risto Kojcev. gym-gazebo2, a toolkit for reinforcement learning using ros 2 and gazebo. *arXiv preprint arXiv:1903.06278*, 2019.

[17] Mark Moll, Ioan A Sucan, and Lydia E Kavraki. Benchmarking motion planning algorithms: An extensible infrastructure for analysis and visualization. *IEEE Robotics & Automation Magazine*, 22(3):96–102, 2015.

[18] Jia Pan, Sachin Chitta, and Dinesh Manocha. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[19] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P Schoellig. Learning to fly–a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. *arXiv preprint arXiv:2103.02142*, 2021.

[20] Martin Pecka and Tomas Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *International Workshop on Modelling and Simulation for Autonomous Systems*, pages 357–375. Springer, 2014.

[21] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[22] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[23] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708*, 7, 2019.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[26] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[28] Akifumi Wachi and Yanan Sui. Safe reinforcement learning in constrained markov decision processes. In *International Conference on Machine Learning*, pages 9797–9806. PMLR, 2020.

[29] Xintong Yang, Ze Ji, Jing Wu, and Yu-Kun Lai. An open-source multi-goal reinforcement learning environment for robotic manipulation with pybullet. *arXiv preprint arXiv:2105.05985*, 2021.

[30] Xintong Yang, Ze Ji, Jing Wu, and Yu-Kun Lai. An open-source multi-goal reinforcement learning environment for robotic manipulation with pybullet. *arXiv preprint arXiv:2105.05985*, 2021.

[31] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.