An Open-Source Software Framework for Reinforcement Learning-based Control of Tracked Robots in Simulated Indoor Environments

A Mitriakov, Panagiotis Papadakis, Serge Garlatti

An Open-Source Software Framework for Reinforcement Learning-based Control of Tracked Robots in Simulated Indoor Environments

A. Mitriakov, P. Papadakis, S. Garlatti

IMT Atlantique, Lab-STICC, UMR 6285, team RAMBO, F-29238 Brest, France

ARTICLE HISTORY
Compiled May 18, 2022

ABSTRACT
A simulation framework based on the open-source robotic software Gazebo and the Robot Operating System (ROS) is presented for articulated tracked robots, designed for reinforcement learning-based (RL) control skill acquisition. In particular, it is destined to serve as a research tool in the development and evaluation of methods in the domain of mobility learning for articulated tracked robots, in 3D indoor environments. Its architecture allows to interchange between different RL libraries and algorithm implementations, while learning can be customized to endow specific properties within a control skill. To demonstrate its utility, we focus on the most demanding case of staircase ascent and descent using depth image data, while respecting safety via reward function shaping and incremental, domain randomization-based, end-to-end learning.

KEYWORDS
Reinforcement learning, robot control, incremental learning, domain randomizaion, staircase negotiation, tracked robots

1. Introduction

Autonomous navigation is a fundamental skill for contemporary mobile robots whose domains of application are continuously expanding. Characteristically, unmanned ground vehicles (UGV) tend to receive increased attention in various domains such as assisting and warehouse robotics, autonomous vehicles, etc [1]. This was particularly motivated by a shift of the community interest from conventional methods to machine learning-based (LB) ones [2] in view of their prominent results in the development of human-level skills [3].

As far as robot navigation is concerned, LB methods have been successfully employed to perform end-to-end control [2] which directly maps sensor observations to actions, often through the employment of reinforcement learning (RL). Thanks to RL, hard to engineer or sophisticated behaviours can be developed through trial-and-error interaction with an environment [4]. RL-based end-to-end methods mainly address outdoor, in-the-wild navigation [1], [5] which can differ from indoor navigation in that

CONTACT P. Papadakis. Email: panagiotis.papadakis@imt-atlantique.fr

The work is performed in the context of the project REACT, project VITAAL and is financed by Brest Metropole, the region of Brittany and the European Regional Development Fund.

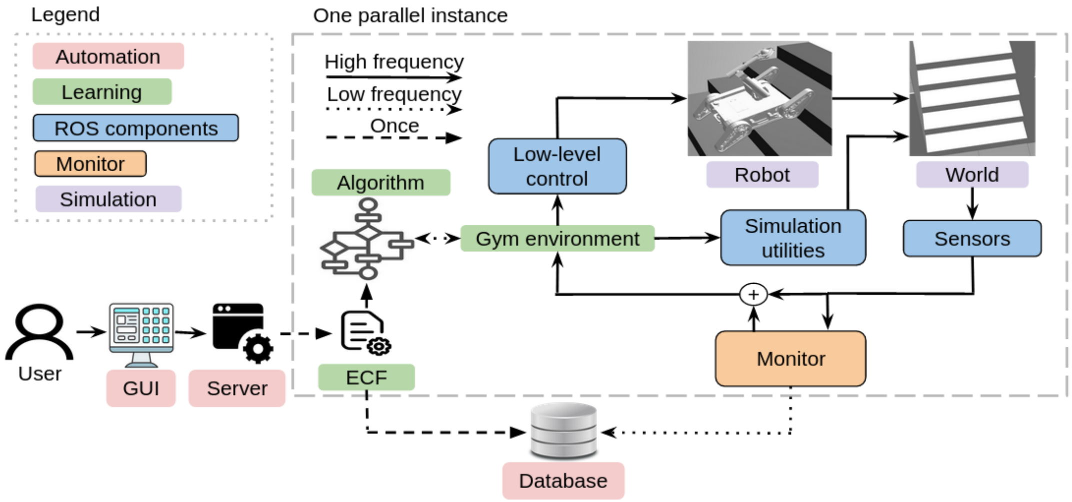


Figure 1. Global overview of the proposed software framework

a robot can never collide and damage its surroundings, as it must guarantee safety of the platform as well as that of the environment. The particular topic of indoor control and navigation is usually associated with navigation on flat surfaces [6,7] where researchers employ RL to a mobile robot for learning to reach a goal while avoiding obstacles [8,9]. However, in real-world the robot should navigate in multi-floor environments which supposes the capacity to negotiate staircases or steps. To the best of our knowledge, control development for the tasks of learning-based staircase ascent and descent is scarcely treated according to [10], especially concerning tracked robots that exhibit a good trade-off between 3D traversability and stability skills.

Learning through RL requires multiple interactions with the environment [11] in order to gain sufficient amount of data for policy parameters update towards maximization of the expected return by the behaviours. Often, learning relies on deployment in flat and fixed environments from a limited sets of preset environments for navigation learning [7,12]. Unfortunately, usage of one or a small number of environments does not encompass domain randomization (DR), which makes the policy suited to only a small number of possible situations [13]. The physics-based simulator [14] (`gazebosim.org`) is usually used alongside the Robot Operation System (ROS) [15] (`ros.org`) which allows to simulate various types of sensors, yet scene rendering is far from realistic. This aspect is addressed in simulations presented in [16,17] that provide highly realistic scenes. It is shown that a robot can learn to navigate in such texture-rich environments, still, articulated tracked robots are not integrated.

To the best of our knowledge, there is no publicly available software framework which addresses robot safety and introduces simulation environments for control learning in the case of ascent and descent of staircase for tracked robots, optionally equipped with an arm. Furthermore, there is a lack of common utilities for development of tracked robot controllers, which constrains researchers to develop custom robot models and environments from scratch. In view of these shortcomings, our goal is to provide a software framework (see Figure 1), residing on Gazebo and ROS, that goes beyond the current state-of-the-art in the following directions :

- Development of an open-source simulation framework and pipeline for 2D navigation and staircase negotiation using reinforcement-learning for articulated

tracked robots. The framework is available at `github.com/gwaxG/robot_ws` and its GUI at `github.com/gwaxG/robot-simu`.
- Successful development of end-to-end controllers for staircase ascent and descent, showing that the proposed incremental DR surpasses conventional uniform DR.

The latter point in particular allows us to go beyond our previous work [10] where we developed staircase negotiation controllers relying on an external, step and robot state estimation set-up and on uniformly varying simulated environments.

The remainder of the paper is organized as follows. In Section 2 we present the background of our work. Section 3 unfolds the intention of the developed framework and provides a formal description of the RL problem that is treated by our framework. Section 4 presents all framework components allowing potential users to experiment with alternative ways of learning control for tracked robots in indoor 3D environments. Finally, in Section 5 we show its application on control learning for staircase ascent and descent traversals.

2. Background

Through the use of RL, a robot can develop complex and difficult-to-engineer behaviours [11] through multiple trial-and-error interaction with an environment. Mnih et al. [3] motivated the scientific community to discover new frontiers of RL usage, for example in autonomous ground vehicle perception and control where RL is often employed in a deep learning setting [2]. Authors of [5] agree that such solutions can exhibit prominent results, but a number of issues still remain to be addressed more thoroughly in order to enhance robustness and generalization.

Simulators The key concept in RL is interaction with a training environment [4] that is usually built on top of a simulator. Authors of [18] present a wide comparison of robotic simulators, over-viewing 18 major simulators and physics engines used in robotics and highlight popular selections. Accordingly, Unity shows better applicability to situations were visual data is important. V-REP [19] is a simulator commonly used in robotics that provides a user-friendly world and robot model configuration tools, exhibiting comparable computational cost and simulation accuracy with Gazebo which is another popular simulator. Nogueira et al. of [20] compare V-Rep and Gazebo in detail showing that V-Rep is far behind Gazebo in terms of integration with ROS, which represents the state-of-the-art for modern robotics development. Gazebo enables total control of the development since it is open source in contrast to commercial V-Reps. For these reasons, Gazebo seems to remain the mostly widely used simulator in the robotics community.

Robot modeling Tracked robots (see Figure 2) are endowed with superior traversability skills and have simpler locomotion in comparison to their legged counterparts. From this perspective they are particularly suited for deployment in applications where it is necessary to negotiate staircases. Sokolov et al. [21] highlight problems associated with the development of track models within Gazebo due to model instability. Thankfully, Pecka et al. [22] alleviate this problem performing a detailed analysis of robot tracks within Gazebo and developing the Contact Surface Motion (CSM) model of articulated robot tracks. Authors rework the dynamic simulation formulation enabling the model to find the force to move the robot at a certain velocity. It does not simulate grousers or deformable tracks. Still, it is plausible on flat and rough terrains and constitutes the best trade-off between track simulation adequacy and the

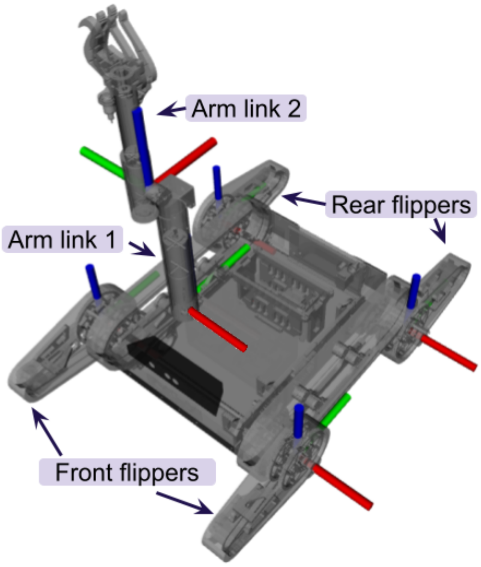


Figure 2. Tracked robot morphology; RGB axes denote XYZ coordinate system of joints; All joints rotate around the X-axis apart from the joint connecting the two arm links which rotates around the Y-axis

computational overhead of simulation.

Learning environments In relation to indoor navigation and control learning, the *AI2-THOR* framework [16] based on Unity provides near-realistic 3D indoor scenes for AI research where agents can navigate and interact with the environment. Authors consider that it can be helpful not only for reinforcement learning but also for learning by interaction, imitation learning, etc. Still, this framework supports only discrete actions and does not address robot dynamics where agents operate in continuous action spaces. The work [17] presents *Interactive Gibson Benchmark* which develops its own simulator based on the physics engine Bullet [23] including a renderer and over one hundred 3D reconstructed environments. It is probably among the best environments to date able to generate highly realistic images and simulate physical interaction. In its most recent version, DR is integrated for objects and materials [17]. However, tracked articulated robots are still not included into this framework.

Various research groups study navigation learning in indoor environments by using popular tools such as Gazebo and ROS. For example, authors of [8] train a robot to navigate in only two instances of a 2D environment. Tai et al. [7] also limit their environment variation by two pre-constructed settings. The robot can cross one real-world maze to reach a goal, but its scalability in a general setting is unknown.

Concerning learning environments, authors of [9] offer a set of 6 settings in total and three robots to use with RL algorithms. Yet, multi-floor domains are not considered neither is DR for ground obstacles, which could serve as a data augmentation technique [24]. The utility of this technique becomes more evident in indoor environments populated with complex but traversable stairs, where robot dynamics become harder to model and necessitate additional attention to safety [25]. At the same time, the environment should vary sufficiently to cover the multitude of possible stair parameters eventually met during deployment [26] both in reality and in simulation. This is achieved by randomization of environment parameters at every training episode. This makes the trained policy more robust and transferable to varying simulated and real-world environments [10,13,27,28]. Unfortunately, we can hardly refer to any available simulation environment that accounts for this demand for articulated tracked robots operating indoor.

Staircase negotiation learning To the best of the authors' knowledge, the problem of 3D traversibility learning for articulated tracked robots is weakly addressed in literature. One of the earliest approaches that employed reinforcement learning for adaptive traversability learning was presented in [29]. The authors developed an RL-based solution for rough terrain traversal. Despite the demonstration of prominent results, expert manual assistance is required in the learning process while actions are limited by pre-defined morphological configurations while contemporary RL methods [30,31] allow the robot to discover such configurations alone.

Another approach is proposed by Pecka et al. [25] whose contribution amounts to the development of RL algorithms for Search and Rescue scenarios that implement constraints in a RL algorithm. That extension showed that a small number of iterations is sufficient to learn flipper control for the traversal of an unknown obstacle. However, the authors did not integrate DR in their approach making the controller not transferable on different staircases.

Contribution The state-of-the-art in simulation environments for RL does not allow learning the task of staircase negotiation learning for articulated tracked robots, especially lacking domain randomization as an essential component of learning. To mitigate this we provide an API-like framework that consists of a DR simulation environment interfaced with the developed OpenAI Gym [32] environment which represents a bridge between the simulation environment and any RL algorithm which implements the OpenAI Gym interface. We base the simulation environment on ROS and Gazebo attracted by the rich inventory of plugins and libraries, the fact that they are open-source and that they facilitate the deployment and transfer of policies learnt in simulation towards a real platform. Finally, we go beyond the state-of-the art by applying our framework to end-to-end control learning of staircase negotiation and provide quantitative experiments that demonstrate its performance.

3. Objective of the framework and problem statement

We propose a framework which consists of a ready-to-use simulation environment that can be used for control learning of articulated tracked robots via RL. The framework is also customizable, thus, it can be used for performance evaluation by allowing the user to control the environment. It is built upon and extends earlier recent works of the authors [33] and [26] that allowed learning in simulation and deploying/transferring of policies learnt in simulation to a real commercial robot [10].

In RL, Q-tables were usually used for value and action-value function approximation [11]. However, applying RL algorithms to robotics it is unavoidable to use function approximators due to high-dimensional action and observation spaces. Another reason is that robots tend to operate within continuous states and actions. Lastly, a function approximator can boost learning, because its updates influence multiple states.

Among a variety of existing function approximators, used to represent what we refer as controllers, we could consider radial basis function neural networks (RBF-NN) and artificial neural networks (ANN). RBF-NNs precede ANN, namely, deep ANN. Their advantages are fast convergence to a global optimum and trial-and-error efficiency, still, RBF-NNs may suffer from difficulty of integration to deep architectures. In contrast, deep ANNs which are endowed with the capability to learn different representations at every intermediate level, allow to attain superior performance [3]. Moreover, ANNs have become significantly wide-spread and supported by many libraries, therefore our framework supports out-of-the-box ANNs, although other types of controllers can be used since they do not interfere with framework components.

To solve a control learning problem, we can use RL algorithms where the policy is directly optimized. The main idea consists in performing gradient ascent over policy parameters to maximize an expected gradient return.

In an indoor navigation setting we can distinguish three principal tasks: 2D navigation, ascent and descent traversals of a staircase. In each of them, the learning objective is to move the robot to a goal, which is a point in space, avoiding ground obstacles on the floor or negotiating a staircase. The user can impose additional constraints which can be optimized jointly with the achievement of the goal.

3.1. Problem statement and treatment by RL

To address the previously described problem, the framework contributes a RL episode life-cycle. RL learning is based on generating data through agent interaction with an environment where at every time step t, an agent being in a state $\boldsymbol{s}_{t-1} \in S$ selects an action $\boldsymbol{a}_{t-1} \in A$ with respect to its policy $\pi_{\boldsymbol{\theta}}$ where $\boldsymbol{\theta}$ is the vector of policy parameters, transits to a new state $\boldsymbol{s}_t$ receiving a scalar reward r. The multitude of transitions from the initial state $\boldsymbol{s}_1$ to the termination state $\boldsymbol{s}_{T+1}$ is called a trajectory $\tau = (\boldsymbol{s}_1, \boldsymbol{a}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_T, \boldsymbol{a}_T, \boldsymbol{s}_{T+1})$ where T is the number of time steps of an episode. A policy $\pi_{\boldsymbol{\theta}}$ is a function parameterized with $\boldsymbol{\theta}$ which sets a rule that helps to take an action. It can be either deterministic $\boldsymbol{a_t} = f(\boldsymbol{s_t})$ where an action is taken at a specific state or stochastic via $\pi_{\boldsymbol{\theta}}(\boldsymbol{a_t}|\boldsymbol{s_t})$ which specifies a conditional probability distribution and provides the action probabilities in a defined state from which we sample the most probable action. During learning, we search for the optimal parameters $\boldsymbol{\theta}^*$ which maximize the expected return :

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} E_{\tau \sim \pi_{\boldsymbol{\theta}}}[R(\tau)] \tag{1}$$

where $R(\tau) = \sum_{t=0}^{T} \gamma^t r_t$ is the cumulative reward over a trajectory τ and γ is a discount factor that can weigh the importance of rewards obtained at different times.

DR Indoor environments can vary significantly and unless this is accounted for during training, the gap between simulation and reality will reduce the performance of the policy during testing, whether it is deployed in simulation or in the real-world. Furthermore, staircase ascent and descent traversals impose strict requirements such as safety where the robot has to perform safe actions in varying environments.

To cover various possible environment configurations we enhance RL with DR in our framework. To do so, we parameterize the training domain $e_{\boldsymbol{\xi}}$ where every domain has a configuration $\boldsymbol{\xi} \in \Xi \subset \mathbb{R}^N$, where Ξ is the configuration space of dimension N. We apply randomization during learning and a policy is learnt on multitude of parameterized environments which favors generalization and allows to maximize the expected return over a distribution of configurations. Thus, the optimal policy parameters are obtained by:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} E_{\boldsymbol{\xi} \sim \Xi}[E_{\tau \sim \pi_{\boldsymbol{\theta}}, e_{\boldsymbol{\xi}}}[R(\tau)]] \tag{2}$$

We assume that to navigate within an indoor environment a robot has access to the following state:

$$\boldsymbol{s} = (v, w, \alpha^{left}_{front}, \alpha^{right}_{front}, \alpha^{left}_{rear}, \alpha^{right}_{rear}, \beta_1, \beta_2, \phi, \psi, f_1, ..., f_{N_f}) \tag{3}$$

where v, w represent linear and angular velocities, $\alpha^{left}_{front}, \alpha^{right}_{front}, \alpha^{left}_{rear}, \alpha^{right}_{rear}, \beta_1, \beta_2$ flipper and arm angles (see Figure 2), where rotation of robot links is considered around red x-axes with the exception of the arm link 2 that rotates around the green y-axis, ϕ and ψ are pitch and roll angles of the platform chassis and $f_1, ..., f_{N_f}$ are N_f in total, general environment features. Note that this general form of the state vector can be altered by the user through an experiment configuration file (ECF) (see section 4.4), for example, in the case where the robot does not require angular velocity or arm control. The control vector $\boldsymbol{a}$ is composed of commands to the robot as follows:

$$\boldsymbol{a} = (\psi^{left}_{front}, \psi^{right}_{front}, \psi^{left}_{rear}, \psi^{right}_{rear}, v_a, w_a, \beta^a_1, \beta^a_2) \tag{4}$$

where $\psi^{left}_{front}, \psi^{right}_{front}, \psi^{left}_{rear}$ and ψ^{right}_{rear} are flipper rotation angles, v_a and w_a are linear and angular velocity commands, β^a_1 and β^a_2 are arm control angles. As in the case of the state vector, the usage of the control vector components can be controlled through the ECF and connected to used robot parts, as will be presented in detail in the sequel.

4. Framework architecture

This section presents the framework and its components, unfolds its workflow from its most low-level parts such as the simulator to the high-level ones such as RL algorithm employment and automation tools. It serves as a guide for the manual modification of inner components to integrate new algorithms, sensors and experiment workflows. Hereafter, we present the simulation environment, the robot command dispatching and perception utilities and the learning environment which drives the learning process. Finally, we discuss algorithm and library integration utilities.

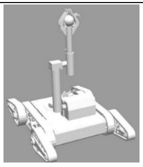
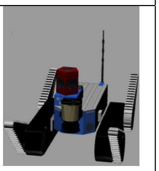
4.1. Simulation

World Our framework is based on the open source physics-based Gazebo simulator which resides at the core of our system. This simulator is widely used and popular within the robotics community and already contains multiple functionalities for control of robotic simulation models. Still, Gazebo can be considered as "low-level" because users have to assemble them according to a scenario. *World* accepts requests from *simulation utilities* which will be described next and spawns a corresponding environment. *Robot* operates in it and provides the output to *sensors*.

Robot Our framework operates with a simulated tracked robot model in the unified robotic description format (URDF, `wiki.ros.org/urdf`) which consists of a body, front and rear flippers and that can be equipped with an arm, whose mass and geometry can be set to match those of a real robot. Interaction between tracks and staircase surface is performed by adopting the CSM model [22]. Front and rear flippers can be jointly or separately controlled. This component receives commands from *low-level control.*

Table I lists examples of two developed articulated robot models that can be trained or operated in simulation. Each robot possesses at least 4 degrees of freedom (DOFs) which consist of linear velocity, angular velocity, front and rear flipper angles. This means that the pair of front flippers is controlled by only DOF, and similarly for the pair of rear flippers. In the case of separate flipper control, robots have 6 DOFs while if an arm is also present and controlled, the total number of DOF raises to 8. These robots possess similar negotiating capabilities.

TABLE I: Robot model characteristics

Property	**Robot 1**	**Robot 2**
Photo		
Mass, kg	20.5	29
Length, m	0.98	1.196
Width, m	0.7	0.61
Height, m	0.4	0.46

4.2. ROS components

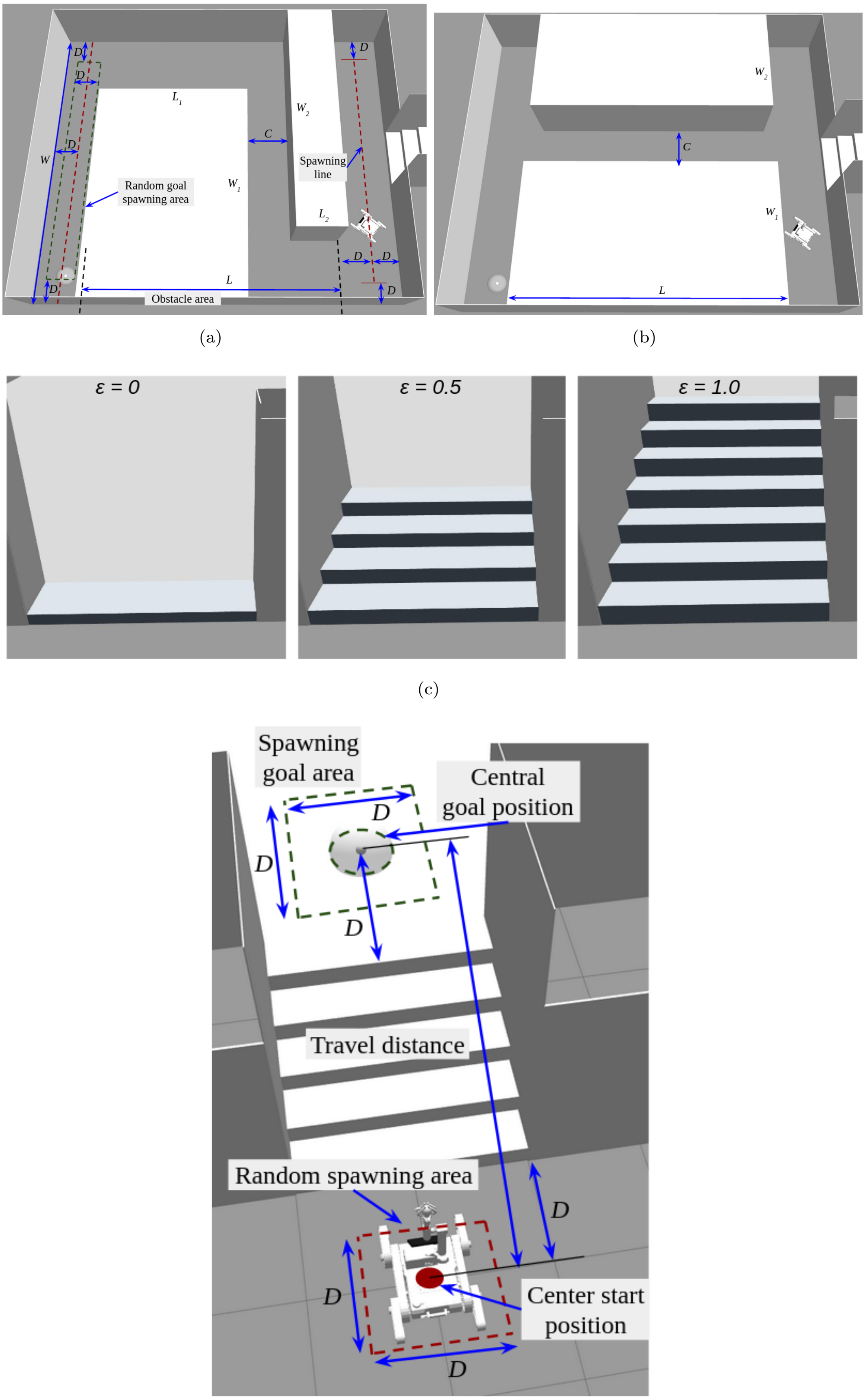
ROS (Robot Operating System) [15] is open-source software under a BSD license. It helps to develop software for robot applications through providing libraries and tools. They consist of device drivers, hardware abstraction, visualizers, libraries, package management, message-passing, and more. The following components of our framework are developed using ROS standards.

4.2.1. Simulation utilities

By careful parameterization of the source domain in simulation, i.e. the definition of Ξ and its samples $\boldsymbol{\xi}$, we seek to generate a sufficiently rich and representative set of situations for the three main tasks of indoor navigation corresponding to 2D navigation, staircase ascent and descent whose environments are presented below. *Simulation utilities* accept commands from *gym environment*, produce models that are usable within *world* and requests the latter to load them.

Domain randomization for 2D navigation In indoor environments which are organized in piece-wise orthogonal configurations (see *Manhattan world* assumption [34]), the robot has to ordinarily traverse hallways or perform more complex zigzag navigation on a 2D ground, as shown in Figure 3 (a) and (b). D, W, L are constant environment parameters which define the size/scale of the environment, the width and the length of the obstacle area. By varying these obstacle parameters, the framework forms an environment configuration vector $\boldsymbol{\xi} = (W_1, L_1, W_2, L_2, C)$, where $W_i \in [\frac{W}{2} - \frac{C}{2}, W - C]$ and $L_i \in [L - W, L]$ are the width and length of obstacles 1 and 2 respectively, $C \in [C^{min}, C^{max}]$ is the minimum distance between obstacles.

The framework generates with equal probability a hallway or a zigzag environment. In the first case, we sample $\boldsymbol{\xi}$ from defined uniform distributions. In the second case,



(a)

(b)

(c)

(d)

Figure 3. Illustration of zigzag (a) and hallway (b) environments, complexity increment of the staircase environment (c), ascent task environment (d)

L_1 and L_2 are equal L, C and W_1 are uniformly sampled within their values, W_2 is sampled from $[0, L - W_1 - C]$.

The user defines in the experiment configuration whether goal and robot spawning is random or fixed. The appearance at random can be seen as a part of DR enhancement where we vary unseen situations through experiment episode initialization. To spawn a robot and the goal, the framework provides 2 separated ROS services. In the case of random goal appearing, the goal can appear either in the rectangle of *random goal spawning area* (see Figure 3 (a)) or at its geometrical center in the fixed goal appearing situation. The framework puts the robot on *spawning line* either at a random point on the same line and random orientation to the goal or at the center of the spawning line with an orientation that is perpendicular to the first step.

Domain randomization for staircase negotiation Two other tasks involved in indoor navigation concern staircase ascent and descent, making the staircase generation an important feature of our framework. A straight staircase can be represented by the step length l, the step height h and the number of steps n, i.e. by an environment configuration $\boldsymbol{\xi} = (l, h, n)$. Minimum and maximum environment configurations are denoted by $\boldsymbol{\xi}^{min}$ and $\boldsymbol{\xi}^{max}$.

We enable two types of staircase generation. The first one assumes $\boldsymbol{\xi}$ is distributed uniformly to generate samples and we term as *uniform* environment. The second type assumes that $\boldsymbol{\xi}$ follows a normal distribution with a diagonal covariance matrix. The corresponding mean and covariance matrix depend on a parameter ϵ which regulates the complexity (and in turn the difficulty), of the generated staircase. We term environments produced with this technique as *incremental* Gaussian environments.

The vector of difference between maximum and minimum environment configurations is the following:

$$\boldsymbol{\Delta} = \boldsymbol{\xi}^{max} - \boldsymbol{\xi}^{min} = (\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, \boldsymbol{\Delta}_3) = (l^{max} - l^{min}, h^{max} - h^{min}, n^{max} - n^{min}) \quad (5)$$

Equations (6a) and (6b) present sampling of the environment configuration $\boldsymbol{\xi}$ and clipping of every sampled element:

$$\boldsymbol{\xi} \leftarrow \mathcal{N}(\boldsymbol{\xi}^{min} + \epsilon \cdot \boldsymbol{\Delta}, \epsilon \cdot \mathrm{diag}(\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, \boldsymbol{\Delta}_3)) \quad (6a)$$

$$\boldsymbol{\xi}_i = \begin{cases} \boldsymbol{\xi}_i^{max}, & \text{if } \boldsymbol{\xi}_i > \boldsymbol{\xi}_i^{max} \\ \boldsymbol{\xi}_i, & \text{if } \boldsymbol{\xi}_i^{min} \leq \boldsymbol{\xi}_i \leq \boldsymbol{\xi}_i^{max} \\ \boldsymbol{\xi}_i^{min}, & \text{otherwise} \end{cases} \quad (6b)$$

where $\boldsymbol{\xi}$ is a candidate configuration, $\leftarrow \mathcal{N}(\cdot, \cdot)$ indicates random sampling from a normal distribution, $\boldsymbol{\xi}^{min}$ and $\boldsymbol{\xi}^{max}$ represent minimum and maximum admissible environment configurations. Equation (6a) indicates that we sample a candidate experiment configuration from the normal distribution with the mean $\boldsymbol{\xi}^{min} + \epsilon \cdot \boldsymbol{\Delta}$ and the covariance matrix $\epsilon \cdot \mathrm{diag}(\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, \boldsymbol{\Delta}_3)$, where ϵ enables to increase the mean and covariance and is automatically reset according to learning progress that we define as mean positive episode reward over last N_ϵ episodes which is configured at the episode beginning. The value of N_ϵ was empirically determined allowing to obtain superior performance compared to non-incremental DR (see section 5). This leaves space for further performance gains if this point is addressed more thoroughly by the users of the framework. Equation (6b) clips $\boldsymbol{\xi}$, so that it fits in the prescribed limits.

Figure 3 (c) shows staircases sampled for 3 different ϵ values. As in the case of 2D environment, the user selects whether goal and/or robot are spawned randomly or at fixed preset positions. In ascent, when random spawning is chosen, the robot can appear at random position and orientation at *Random spawning area* (see Figure 3 (d)) or at *center start position* being aligned with the staircase in fixed spawning. The goal can be spawned in the same way either at *spawning goal area* or at *central goal position.* The goal and robot spawning areas are reversed in descent.

4.2.2. Low-level control

The framework provides control software which accepts a sole ROS-based message issued from *gym environment* for controlling the entire action space of the robot, handles and dispatches it to corresponding ROS and Gazebo low-level controllers and further provides feedback through a message providing information about current linear and angular robot velocity and joint configuration state. Then, flipper and arm joint rotation angles are limited to reflect real robot operation. A spawned robot can be operated with a keyboard and a ROS message.

4.2.3. Sensors

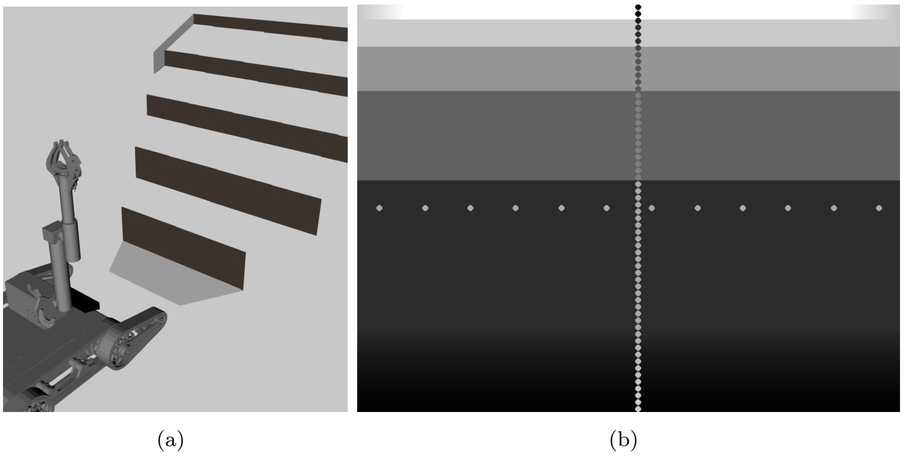
Another central infrastructure component of our framework is related to perception. Using an RGB-D sensor to perceive the environment within *world*, the framework provides a baseline depth-based feature extractor and dispatches its calculations down to *gym environment* and *monitor.* We have decided to rely on the depth image due to the poor realism of simulated RGB images. The robot facing a staircase and its depth perception is presented in Figure 4 (a) while Figure 4 (b) shows the associated features whose coloring is reversed to avoid melding with the background.

Borrowing the idea of feature extraction from [7], we convert every depth image into vertical and horizontal beam groups. The user can choose the cardinality of horizontal and vertical beams after which the framework calculates beams positions and averages non NaN (Not-a-Number) depth image pixel values around them within an area predefined by a user. As it was shown in [7], 10 horizontal beams are sufficient to learn map-less navigation, but we opt for using more vertical beams to better retrieve the structure of the staircase in ascent and descent tasks.

The robot further contains a simulated inertial measurement unit, the output of which is published within the ROS ecosystem and used in forming the observation vector and calculating the reward. Alongside, the framework further provides the ground truth pose of the robot.

4.3. Monitor

Rewards and episode termination signals are issued from *Monitor* (see Figure 1) whose importance was particularly highlighted in our earlier work [26]. These calculations are produced with help of *sensors* output and, being concatenated with image features, goes down to the *gym environment.* An appropriate reward function drives learning towards the acquisition of complex behaviours and a well implemented termination signal can boost learning. As part of the framework and baseline methodology, we provide the exact same set of three reward functions (see section 3.B [26]) which can be used to learn ascent and descent staircase traversals, that could serve as a starting



(a)

(b)

Figure 4. (a) Robot in front of a staircase, (b) horizontal and vertical extracted features

point for future research and bench-marking.

Recalling [26], the basis positive reward is the travelled distance from the robot starting position to the goal. This reward is used in all tasks. In staircase traversal tasks, we additionally add negative penalties weighted by their scaling coefficients used to control arbitrary bias of negative penalties. The COG deviation-based penalty drives learning towards acquisition of a safe behaviour which enhances robot stability in ascent. The angular velocity-based negative penalty mitigates drop impacts during descent transitions.

Finally, the component *monitor* receives data from the most of ROS utilities to provide an adequate guidance of the learning process. It monitors the robot-goal distance, provides safety estimation, which monitors COG deviation and pitch angular velocity of the platform, forms termination signals and sends data to a *database*. We consider three cases when *monitor* triggers a termination signal, namely: (i) tipping over, (ii) exceeding episode time steps and (iii) reaching the goal. Safety estimation is performed for staircase ascent when the component continuously monitors mean time step center of gravity (COG) deviation and mean time step pitch angular velocity for descent.

4.4. Learning

This subsection unfolds the details of the *learning* components of the framework, as shown in the beginning in Figure 1. Our framework allows to integrate different implementations of reinforcement learning algorithms as long as they support the OpenAI Gym [32] which is a standard toolkit and operates as an interface between a RL algorithm and an agent environment. The key component *learning* of our framework encompasses RL algorithms instantiation, an OpenAI Gym implementation and their ECF.

ECF Before every experiment, the user can instantiate an *ECF* which is received from *server* and contains user-defined RL algorithm and simulation environment parameters of the experiment loaded by *algorithm*, passed to *Gym environment* and saved by *database*. File templates can vary for integration of custom experiments and RL libraries. Finally, the ECF is used for policy testing saving robot performance such as travelled distance and safety measures.

Algorithm As soon as an experience starts the *ECF* is consumed by *algorithm*. The latter selects a library and a corresponding RL algorithm with defined parameters. This also instantiates *Gym environment* after which policy training is triggered.

Gym environment Our framework integrates an OpenAI Gym environment which receives signals and calculations of *monitor*, provides experiment data back to *algorithm*, requests *simulation utilities* for a new environment and, finally, robot and goal respawning controlling *robot* through *low-level control*. *Gym environment* dispatches rewards on every time step to *algorithm* as well as episode termination signals received from *monitor*. It forms the observation vector (cf. eq. (3)), which is passed to the policy π_θ, obtains an action vector and sends it to *low-level control*. Finally, *Gym environment* receives the output from *monitor* related to time step reward and termination signals, based on which the RL algorithm continues to perform policy optimization if required and the cycle repeats.

4.5. Automation

Launching of all aforementioned components, creating the *ECF* and data saving can be manually performed by the user. However, to simplify interaction with the environment we provide certain *automation* tools, listed below.

Graphical user interface (GUI) This component enables managing of experiments and visualization of results through sending commands to *server*. Figure 5 (a) and (c) shows screenshots of its windows, the first one presenting the creation of a *configuration* file and the second a visualization of ongoing learning results.

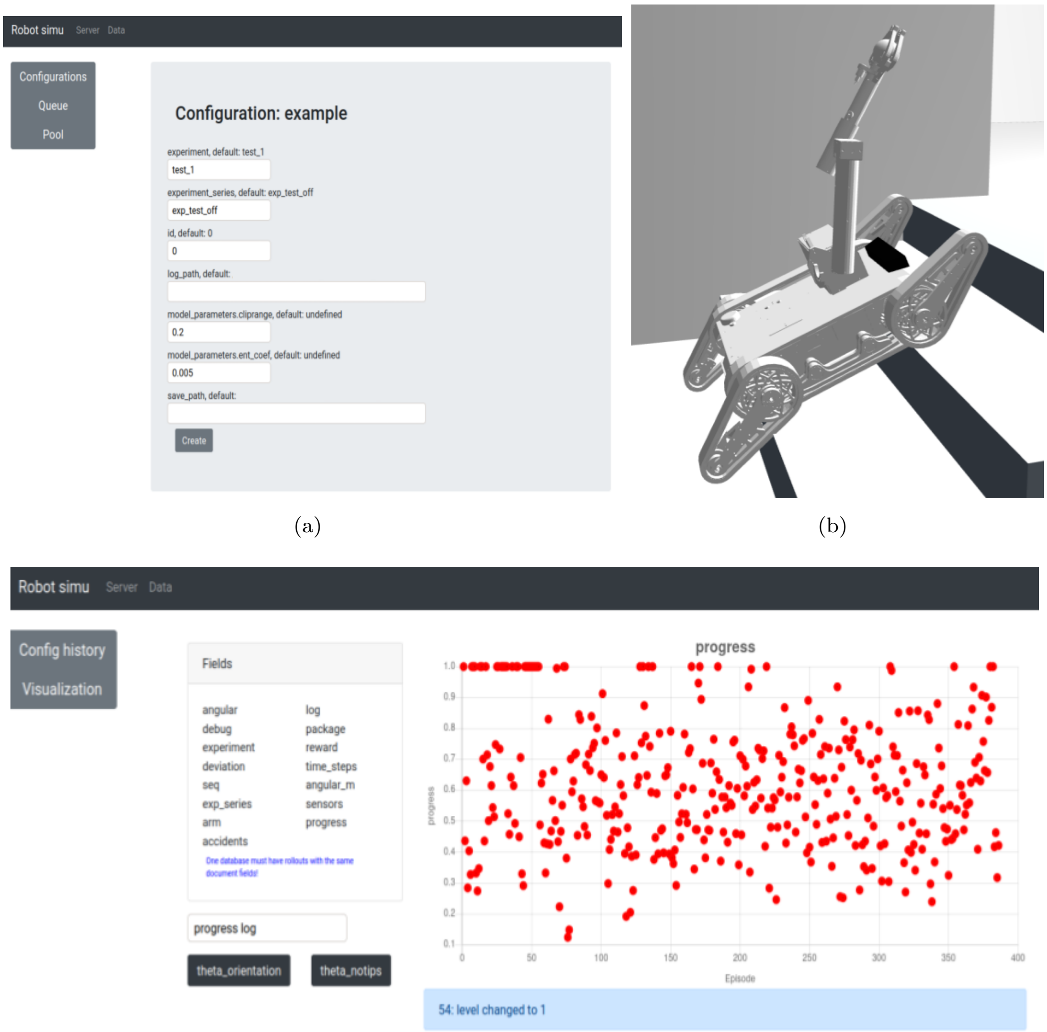
Server This is a light-weight process which functions in parallel to the rest of the framework, helps to visualize learning results and run experiments, otherwise every experiment would need to be launched manually. The user can start an experiment through *GUI* which requests the *server*, the later creates the *ECF* and launches *algorithm*. To visualize the data of the experiment, *GUI* interacts with *server*, which requests *database*, and returns data.

Database To promote a disciplined approach to data management we employ a database within the component *database*. The later is a program which works in parallel to other components and stores data from *ECF* and every learning episode statistics performing ”create”, ”read”, ”update” and ”delete” operations.

5. Experiments

This section presents the developed simulated robots that are provided with the framework and the conducted experiments which allowed to obtain ascent and descent policies with optimization of desired properties. We endorse perception principles from [7] where authors study the case of 2D navigation learning and investigate its extension for control learning in ascent and descent tasks. These experiments are provided as proof-of-concept for the utility of the framework, with the hope of stimulating further research and allowing to benchmark different RL approaches for learning control of articulated tracked robots in indoor environments. The paper is accompanied by a repository where the framework is stored `github.com/gwaxG/robot_ws`, additionally, GUI is located on `github.com/gwaxG/robot-simu`.

In the scope of this paper we employ Soft Actor-Critic (SAC) [31]. This is a recent RL algorithm which has shown better performance compared to its counterparts. It optimizes a stochastic policy in an off-policy way where its key feature is maximization



Configuration: example

experiment, default: test_1
test_1

experiment_series, default: exp_test_off
exp_test_off

id, default: 0
0

log_path, default:

model_parameters.cliprange, default: undefined
0.2

model_parameters.ent_coef, default: undefined
0.005

save_path, default:

(a)

(b)

One database must have rollouts with the same document fields!

progress log

54: level changed to 1

(c)

Figure 5. Framework GUI and simulation views: (a) experiment configuration, (b) robot learning ascent task, (c) episode reward scatter

of both the policy entropy and the expected return. We employ SAC mostly with its original hyperparameters from the *Stable baselines3* RL library [35] which contains many other state-of-the-art algorithms.

TABLE II: Learning tasks

Task id	**Direction**	**Environment**	**Criterion**
Asc-inc-cog	ascent	incremental	COG
Asc-uni-cog	ascent	uniform	COG
Des-inc-ang	descent	incremental	ang. vel.
Des-uni-ang	descent	uniform	ang. vel.

Environment configuration We deploy our framework and obtain policies *asc-inc-cog*, *asc-uni-cog*, *des-inc-ang* and *des-uni-ang* for tasks presented in Table II where

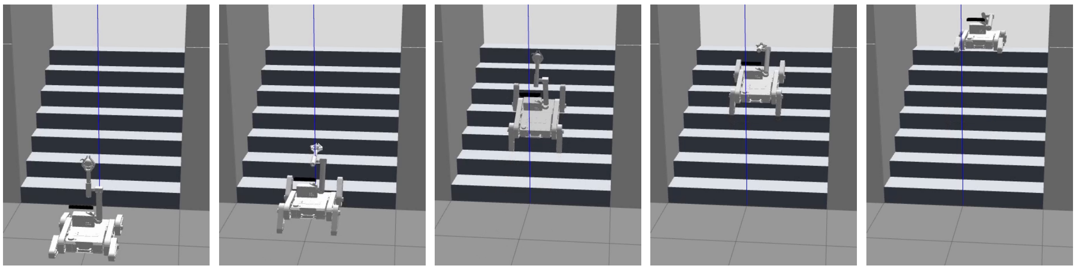


Figure 6. Snapshots of robot ascent of a staircase using a policy learnt in an end-to-end fashion.

the staircase parameters are sampled from incremental (**-inc-**) and uniform (**-uni-**) distributions and robot 1 was employed using 5 DOFs which are linear velocity, angles of paired front and rear flippers and 2 arm joint angles. Commonly, in every experiment the robot starts off being by facing the staircase. Angular velocity control is redundant in such setting because the robot can move only forward, therefore it is omitted in the action vector as well as horizontal features and roll base angle in the observation vector, and it makes sense to couple left and right flipper angles at the front and at the rear respectively. The observation vector is populated with $(v, \alpha_{front}, \alpha_{rear}, \beta_1, \beta_2, \phi, f_1, ..., f_{60})$, i.e. linear velocity, front and rear flipper angles where front and rear flippers are coupled, arm joint angles, pitch angle of the platform and 60 vertical features extracted from the depth image. We employ a 2-layer perceptron in which each inner layer possesses 64 neurons whose weights are updated by SAC where we use hyper-parameters proposed by the library implementation besides two parameters. After tuning, we have set the initial value of the entropy regularization to 0.5 and the entropy regularization coefficient to 0.05, γ was set to the library default value 0.99.

Each task is performed three times in total and lasts up to 20000 time steps but can be terminated earlier if the average return over the latest 30 episodes reaches an empirical threshold value of 0.6 for descent and 0.5 for ascent when learning converges.

Ascent task performance analysis Figure 7 presents smoothed cumulative reward curves during *asc-inc-cog* and *asc-uni-cog* tasks learning with $\pm\sigma$ bands. Since the duration of each experiment can vary, the x-axis presents a universal *learning time* which reflects scaling of all experiments time steps to the $0-1$ scale. For the *asc-inc-cog* task, we can see that cumulative reward raises up to 0.4 by the end of learning time. The task *asc-uni-cog* shows the same dynamics, however its reward curve is mainly located below the curve of *asc-inc-cog* task and converges to 0.0 which shows boost of learning with Gaussian sampling of environment.

Figure 8 (a) shows evolution of COG deviation during learning, we can see that its values significantly drops from $0.25m$ down to $0.14m$. The agent shows its capability to learn staircase traversal relying on visual perception and to increase its safety through COG deviation minimization. Speaking about performance of the *asc-uni-cog*, we can see that COG deviation is not optimized to the same extent and converges to $0.17m$. Thus, we can conclude that the incremental DR leads to a more optimized behaviour. Finally, Figure 6 shows snapshots of the robot successfully ascending a staircase by employing a trained policy (full video is provided as supplementary material).

Descent task performance analysis Figure 7 presents cumulative reward in *des-inc-ang* and *des-uni-ang* tasks. Reward curves and mean episode pitch angular velocity show the same pace of convergence and attain similar values. Reward curves begin at -0.2 and -0.42, then they drastically increase up to 0.5 by the end of learning,

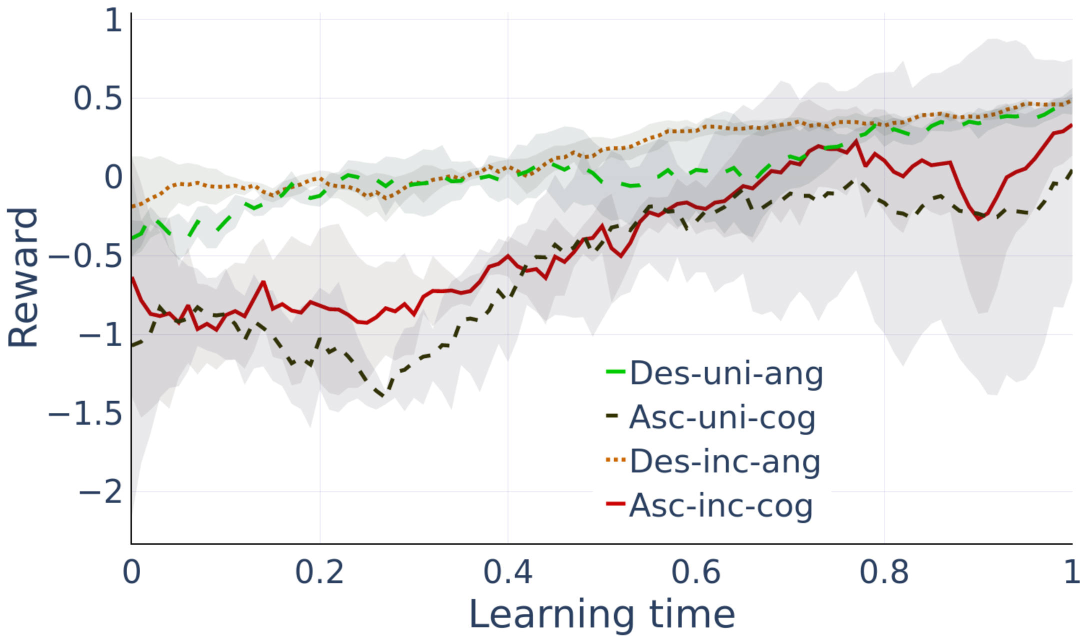
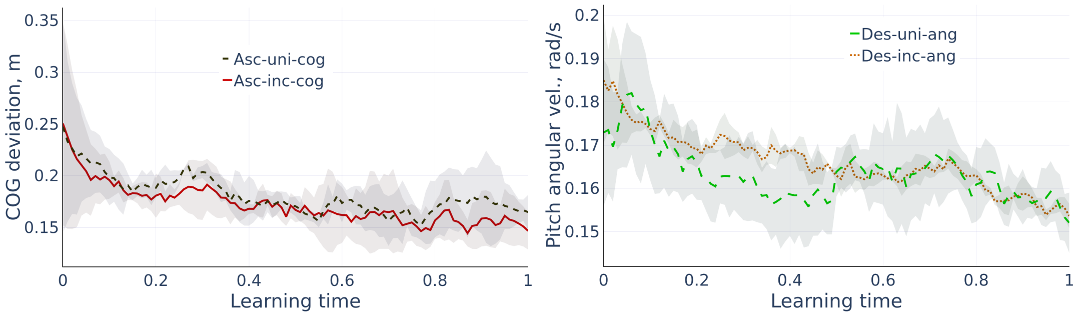


Figure 7. Reward convergence



(a)

(b)

Figure 8. Evolution of optimized values during task execution

and they slowly continue to improve after 0.8 of learning time. They both trigger termination by the early stopping criterion when the value 0.5 is reached. Overall, however, the task learnt incrementally *des-inc-ang* has fewer performance oscillations and hence seems more stable than the non-incremental.

Mean time step angular velocity (see Figure 8 (b)) drops from 0.185 and 0.173 for *des-inc-ang* and *des-uni-ang* respectively, to 0.155 by the end of learning for both tasks. As previously, we can notice here too that overall the task learnt incrementally oscillates less. In descent, the robot is always able to advance downstairs through even a small velocity application, the principal goal is to mitigate drop impacts which could be even further by inappropriate behaviours. Nonetheless the curves of pitch angular velocity evolution show that the robot achieves the prescribed goal.

Test performance Figure 9 presents how much the performance of a policy trained in the uniform environment differs from a policy trained in the incremental environment for ascent and descent tasks. To obtain these statistics, a policy trained for a

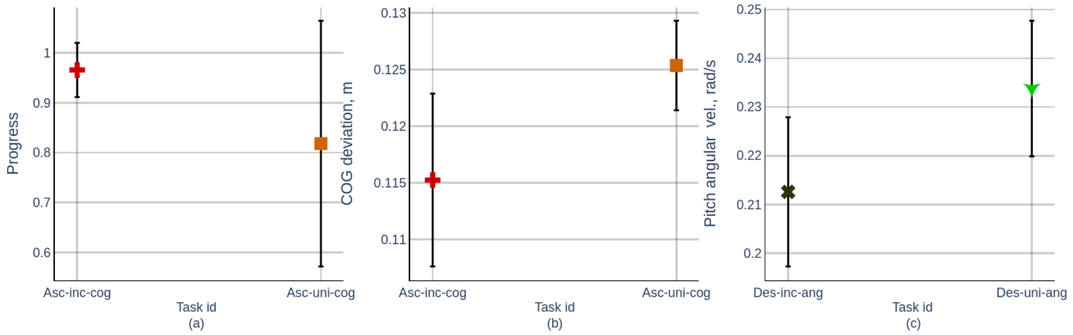


Figure 9. Comparison of average performances obtained when employing conventional, uniform environment sampling while learning (-uni-) or incremental domain randomization (-inc-)

given task was applied 10 trials on a fixed staircase with an environment configuration of the highest complexity $\boldsymbol{\xi} = (h^{max}, l^{min}, n^{max})$. Then, relative travelled distance (progress), COG deviation and angular velocity were recorded during trials.

Figure 9 (a) presents mean progress in ascent tasks over all trials, with error bars corresponding to one standard deviation (we refrain from using progress as performance measure in descent tasks, since the goal is reached more easily). As we can see, a policy *asc-inc-cog* exhibits nearly perfect performance 0.965 ± 0.05 in contrast to *asc-uni-cog* policy 0.82 ± 0.24 which tends to reach the goal less often and has higher variance. At the same time, the COG deviation of the policy (see Figure 9 (b)) *asc-inc-cog* is $0.115 \pm 0.008m$ which is lower than the COG deviation of the policy *asc-uni-cog* $0.125 \pm 0.005m$ by $0.01m$. The incremental environment also improves performance in descent tasks (see Figure 9 (c)) where the policy *des-inc-ang* exhibits better performance $0.215 \pm 0.15m$ against the one of the policy *des-uni-ang* that attains $0.235 \pm 0.14m$.

6. Conclusion

A reinforcement learning-based software framework for control learning of articulated tracked robots is presented. The framework is unique in its kind in terms of the type of task for which it is destined to be used, integrating domain randomization and the possibility of incremental learning, accompanied with two articulated tracked robot models.

The framework applied on control learning for ascent and descent staircase traversals with safety constraints has shown ability to learn reasonable skills with joint arm control relying on depth image features of the environment. Furthermore, enhancing domain randomization with sampling of environment configurations from a Gaussian distribution, that is controlled by the estimation of learning progress, has shown superior results in comparison to the uniform environment.

We believe that the framework could stimulate research and experimentation in various directions. For example, possible future improvements of the framework could further account for generation of spiral staircase generation or more complex variations of floor obstacles to increase complexity of the 2D control learning and better address the structural complexity of real-world environments. Another extension could concern varying the number of DOFs of the robots in an incremental learning setting, for example, by starting learning using 2 DOFs corresponding to linear and angular velocity control and then progressively adding additional DOFs of flipper and arm control while environment complexity increases. Finally, there are various points to be accounted for in the deployment in reality of policies learnt in simulation, such as eventual domain differences, noise levels or hardware constraints such as the placement of the RGB-D camera on the robot.

References

[1] Xiao X, Liu B, Warnell G, et al. Motion control for mobile robot navigation using machine learning: a survey. abs/201113112. 2020;.

[2] Guastella DC, Muscato G. Learning-based methods of perception and navigation for ground vehicles in unstructured environments: A review. Sensors. 2021;21(1).

[3] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. CoRR. 2013;abs/1312.5602.

[4] Kober J, Bagnell J, Peters J. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research. 2013 09;32:1238–1274.

[5] Tai L, Liu M. Deep-learning in mobile robotics - from perception to control systems: A survey on why and why not. ArXiv. 2016;abs/1612.07139.

[6] Ejaz MM, Tang TB, Lu CK. Vision-based autonomous navigation approach for a tracked robot using deep reinforcement learning. IEEE Sensors Journal. 2021;21(2).

[7] Tai L, Paolo G, Liu M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. CoRR. 2017;abs/1703.00420.

[8] Wenzel P, Schön T, Leal-Taixé L, et al. Vision-based mobile robotics obstacle avoidance with deep reinforcement learning. IEEE International Conference on Robotics and Automation. 2021;.

[9] Zamora I, Lopez NG, Vilches VM, et al. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. ArXiv. 2016;abs/1608.05742.

[10] Mitriakov A, Papadakis P, Kerdreux J, et al. Reinforcement learning based, staircase negotiation learning: Simulation and transfer to reality for articulated tracked robots. IEEE Robotics and Automation Magazine. 2021;28(4):10–20.

[11] Sutton RS, Barto AG. Reinforcement learning: an introduction. MIT Press; 2018.

[12] Chaffre T, Moras J, CHAN-HON-TONG A, et al. Sim-to-Real Transfer with Incremental Environment Complexity for Reinforcement Learning of Depth-Based Robot Navigation. In: International Conference on Informatics, Automation and Robotics; ????

[13] Tobin J, Fong R, Ray A, et al. Domain randomization for transferring deep neural networks from simulation to the real world. IEEE/RSJ International Conference on Intelligent Robots and Systems. 2017;.

[14] Koenig N, Howard A. Gazebo-3d multiple robot simulator with dynamics. Technical report; 2006.

[15] Quigley M, Conley K, Gerkey B, et al. Ros: an open-source robot operating system. ICRA Workshop on Open Source Software. 2009;.

[16] Zhu Y, Mottaghi R, Kolve E, et al. Target-driven visual navigation in indoor scenes using deep reinforcement learning. IEEE International Conference on Robotics and Automation. 2017;.

[17] Shen B, Xia F, Li C, et al. igibson, a simulation environment for interactive tasks in large realistic scenes. CoRR. 2020;abs/2012.02924.

[18] Santos Pessoa de Melo M, Gomes da Silva Neto J, Jorge Lima da Silva P, et al. Analysis and comparison of robotics 3d simulators. In: Symposium on Virtual and Augmented Reality; 2019.

[19] Rohmer E, Singh SPN, Freese M. V-rep: A versatile and scalable robot simulation framework. In: IEEE/RSJ International Conference on Intelligent Robots and Systems; 2013.

[20] Nogueira L. Comparative analysis between gazebo and v-rep robotic simulators. Seminario Interno de Cognicao Artificial-SICA. 2014;.

[21] Sokolov M, Afanasyev I, Lavrenov R, et al. Modelling a crawler-type ugv for urban search and rescue in gazebo environment. In: International Conference on Artificial Life and Robotics; 2017.

[22] Pecka M, Zimmermann K, Svoboda T. Fast simulation of vehicles with non-deformable tracks. CoRR. 2017;abs/1703.04316.

[23] Mahjourian R, Jaitly N, Lazic N, et al. Hierarchical policy design for sample-efficient learning of robot table tennis through self-play. CoRR. 2018;abs/1811.12927.

[24] Dulac-Arnold G, Mankowitz DJ, Hester T. Challenges of real-world reinforcement learning. CoRR. 2019;abs/1904.12901.

[25] Pecka M, alansk V, Zimmermann K, et al. Autonomous flipper control with safety constraints. IEEE/RSJ International Conference on Intelligent Robots and Systems. 2016; .

[26] Mitriakov A, Papadakis P, Mai Nguyen S, et al. Staircase negotiation learning for articulated tracked robots with varying degrees of freedom. IEEE International Symposium on Safety, Security, and Rescue Robotics. 2020;.

[27] Abdolmaleki A, Simes D, Lau N, et al. Contextual relative entropy policy search with covariance matrix adaptation. In: International Conference on Autonomous Robot Systems and Competitions; 2016. p. 94–99.

[28] Zhao W, Queralta JP, Westerlund T. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In: IEEE Symposium Series on Computational Intelligence; 2020.

[29] Zimmermann K, Zuzanek P, Reinstein M, et al. Adaptive traversability of unknown complex terrain with obstacles for mobile robots. In: IEEE International Conference on Robotics and Automation; 2014.

[30] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. CoRR. 2017;abs/1707.06347.

[31] Haarnoja T, Ha S, Zhou A, et al. Learning to walk via deep reinforcement learning. International Conference on Machine Learning. 2018;.

[32] Brockman G, Cheung V, Pettersson L, et al. Openai gym. abs/160601540. 2016;.

[33] Mitriakov A, Papadakis P, Mai Nguyen S, et al. Staircase traversal via reinforcement learning for active reconfiguration of assistive robots. IEEE International Conference on Fuzzy Systems. 2020;.

[34] Coughlan J, Yuille AL. The manhattan world assumption: Regularities in scene statistics which enable bayesian inference. Advances in Neural Information Processing Systems. 2001;13.

[35] Raffin A, Hill A, Ernestus M, et al. Stable baselines3 ; 2019.