CLAWAR 2020: 23rd International Conference on
Climbing and Walking Robots and the Support
Technologies for Mobile Machines,
Moscow, Russian Federation, 24-26 August 2020.
https://doi.org/10.13180/clawar.2020.24-26.08.58

EVALUATING DEEP REINFORCEMENT LEARNING ALGORITHMS FOR QUADRUPEDAL SLOPE HANDLING

ATHANASIOS S. MASTROGEORGIOU*[1], YEHIA S. ELBAHRAWY*[2], KONSTANTINOS MACHAIRAS[1], ANDRÉS KECSKEMETHY[2] and EVANGELOS G. PAPADOPOULOS[1]

[1]*School of Mechanical Engineering, National Technical University of Athens, 9 Heroon Polytechniou Str. 15780, Athens, Greece. E-mail: {*amast, kmach, egpapado*}*@central.ntua.gr, *http://www.mech.ntua.gr*

[2]*Faculty of Engineering, University of Duisburg-Essen, Duisburg, 47057, Germany, E-mail: {yehia.el-bahrawy, andres.kecskemethy}@stud.uni-due.de*

In recent years, a number of deep reinforcement learning (DRL) algorithms have emerged that promise to automate the development of locomotion controllers and map sensory observations to low-level actions. However, legged locomotion still is a challenging task for DRL algorithms, especially when slope handling is required. As a result, a framework using commonly used tools (ROS, Gazebo, etc.) and specific slope handling scenarios would enable the evaluation of recent DRL algorithms in order to choose the appropriate algorithm for a given task. In this work, an evaluation framework is proposed that combines DRL with trajectory planning at toe level aiming at reducing training time and facilitate decision-making in slope-handling cases. The proposed evaluation scheme is extensively tested in a Gazebo environment and valuable results are produced using three state-of-the-art DRL algorithms.

1. Introduction

Recently, there has been an increased interest in legged robots. Legged systems interact with their surroundings through multiple contact points changing continuously. Such systems can traverse various terrain types, or handle terrain discontinuities employing accurate foot placement, making them more versatile than wheeled robots. Yet, quadrupeds have complex dynamics and many degrees of freedom that must be well orchestrated for achieving a robust and stable locomotion pattern. Controlling such high-dimensional, non-linear, and underactuated systems is a long-standing research challenge.

1.1. *DRL in Quadrupeds, from Simulation to Reality*

Model-based control approaches require an accurate dynamics model of the robot and include state estimation to achieve contact scheduling, trajectory optimization, and foot placement planning [1, 2, 3, 4]. In contrast, data-driven methods, such as model-free DRL, already have produced promising results showing that they can overcome several challenges and limitations of model-based approaches by training effective controllers directly from experience. Very recent promising research results in legged robotics demonstrated that learned locomotion policies can be transferred from simulation to reality by using high-fidelity simulations [5, 6, 7]. Specifically, this was achieved by learning parts of the simulated model from real data [5], or by model parameter estimation [6]. Model-free methods have been applied to bipeds like Cassie from Agility Robotics [8], without resorting on model-based simplifications commonly used to realize control policies. All approaches usually consist of a simulation environment and a DRL algorithm that performs part of the footstep planning and control.

1.2. *Benchmarking DRL Algorithms*

The massive emergence of DRL algorithms makes it difficult to evaluate the progress in this domain of quadrupedal control due to the lack of a commonly adopted framework. The focus

* The first two authors contributed to this work equally.

of this work is to conclude which DRL algorithm should be chosen for slope handling – a common problem that quadrupeds need to tackle – by utilizing development tools commonly used worldwide by researchers in legged robotics and DRL. Similar attempts in the area of continuous control have been presented in [10] and [11]; however, these try to be broad by focusing on algorithm benchmarking and not on algorithm application on legged robotics. As a result, they either use simple robots [11] or simplified 2D models [12].

In this paper, benchmarking results for state-of-the-art DRL algorithms are presented, employing a realistic 3D model of the Laelaps II quadruped (Figure 1), and a trotting controller as in [17]. The benchmarking scheme mainly consists of two parts: the applied DRL algorithm part, and the toe level trajectory planning part. The controller performance is tested extensively using an accurate model in Gazebo, simulating the full dynamics of the robot.

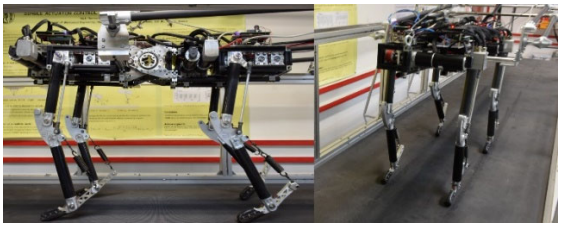


Figure 1. The quadruped robot Laelaps II, built by the Legged Robotics Team at the Control Systems Lab of NTUA, on the Lab's treadmill.

This paper consists of four sections. Section 2 presents an overview of the simulation environment, the detailed Laelaps II 3D model in Gazebo and the trajectory planning part. Section 3 presents the benchmarking architecture and the chosen DRL algorithms. In Section 4, experimental results are presented and discussed aiming to reach useful conclusions.

2. Simulation Environment

For the simulations, tools such as the Gazebo[†] simulator, and the Robot Operating System (ROS[‡]) were employed. In this framework, different robot models described in Simulation Description Format (SDF) can be loaded and with the appropriate adjustments, trained using state-of-the-art DRL algorithms. In this work, three algorithms are applied to the Laelaps II model in Gazebo.

2.1. *Laelaps II Quadruped*

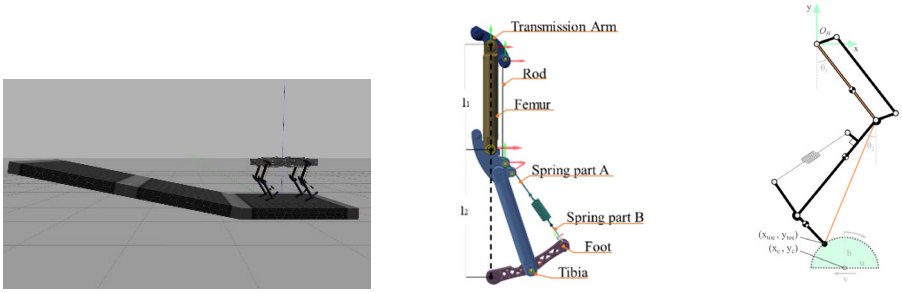
Laelaps II is a quadruped robot built by the Legged Robotics Team at the Control Systems Lab of NTUA. The robot parameters are presented in [17]. The actuation system of each leg comprises a RE50 Maxon motor for the hip and an EC45 Maxon motor for the knee. Both are equipped with gearboxes and belt-pulley transmissions. Since the knee motor is body-mounted, a parallel mechanism is used to drive the distal leg segment (tibia). The maximum torque/angular rate capabilities of the Laelaps II leg are 50 Nm/75 rpm for the hip, and 50 Nm/55 rpm for the knee; exceeding these limits will cause damage to the gearboxes, thus the gearboxes are responsible for the torque/angular rate limitations.

2.2. *Laelaps II Model in Gazebo*

For the needs of this work, a simulation environment was set up in Gazebo using parameters for the quadruped presented in [17]. This environment consists of the Laelaps robot on level terrain and ramps of various inclinations, see Figure 2a. For the ground contacts, the coefficient of restitution used by the Open Dynamics Engines was utilized. Concerning the robot model, the

[†] http://gazebosim.org/
[‡] https://www.ros.org/

original CAD files of the robot's legs and body were used and added in the xacro[§] robot model. The mass properties (mass, inertia matrix, CoM, etc.) for each leg part (see Figure 2b) were obtained from Solidworks CAD files. The tendon-like part of the leg was modeled as a prismatic joint with a spring constant equal to the that of the springs used in the Laelaps II robot legs, i.e.: 26480 N/m. The maximum torque and angular rate constraints were respected. The evaluation framework can be found at CSL's bitbucket repository[**].



(a) (b) (c)

Figure 2. (a) Laelaps II in Gazebo, (b) Detailed model of the Laelaps II leg in Gazebo, (c) The toes follow semi-elliptical trajectories. DRL selects the trajectory center for each leg and the planner produces the entire trajectories.

2.3. *Trajectory Planning*

A smooth yet simple trajectory planning algorithm is used in this work. The main goal of the scheme is to make the robot move forward at a straight line and constant velocity to perform a smooth car-like motion. To this end, points are generated along a semi-elliptical primitive in the workspace of each leg toe to be used as reference for the leg motion controller, see Figure 2c.

A main input to the algorithm is the current simulation time t_{sim}. Depending on t_{sim}, the algorithm determines the phase in which the leg should be at each moment (stance or swing). To achieve this, the modulo of t_{sim} with the trajectory period $T_{traj} = T_{stance} + T_{swing}$ is calculated first, where T_{stance} and T_{swing} are the durations of the stance and the swing phase respectively. Then, the result is compared with T_{swing} to determine the current leg phase. To also add a tunable phase shift in the motion of each leg, a term $\Delta t_{leg} \in [0,\ T_{traj})$ is added to t_{sim}, where $leg \in \{1,2,3,4\}$, see Figure 3b. In this way, the time input for the planner of each leg becomes $t_{leg} = t_{sim} + \Delta t_{leg}$. That said, a stance trajectory is performed if $\mathrm{mod}(t_{leg},\ T_{traj}) \in [0,\ T_{\mathrm{swing}})$ or a swing trajectory is performed if $\mathrm{mod}(t_{leg},\ T_{traj}) \in [T_{\mathrm{swing}},\ T_{traj})$.

Stance phase. The stance phase trajectory is mainly responsible for the desired robot motion; to propel the robot forward at a desired constant velocity v and a constant height h, the toes must follow a trajectory parallel to the ground at the desired velocity but in the opposite direction, i.e. backwards. The planning algorithm outputs the coordinates of the toe $x_{toe,leg}$, $y_{toe,leg}$ w.r.t. a coordinate system O_H fixed at each hip and parallel to the body; these are given by,

$$\begin{aligned} x_{toe,leg}(t_{leg}) &= x_{c,leg} + a_{leg} - (t_{leg} - T_{swing})v, \quad v = 2a_{leg} / T_{\mathrm{stance}} \\ y_{toe,leg}(t_{leg}) &= y_{c,leg} \end{aligned} \tag{1}$$

where $x_{c,leg}$, $y_{c,leg}$, are the ellipse center coordinates w.r.t. O_H, and a_{leg} is the ellipse horizontal semi-axis, see Figure 2c. The absolute value of $y_{c,leg}$ corresponds to the height of the robot CoM.

Swing phase. The swing phase trajectory is also important to provide adequate ground clearance while a toe is repositioned from the end point of the previous stance phase to the start point of the next one. The general form of an elliptical trajectory is used as described by,

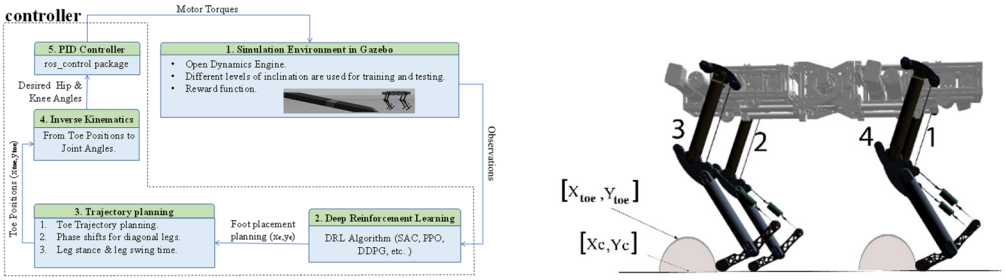
[§] http://wiki.ros.org/xacro
[**] https://bitbucket.org/csl_legged/drl-evaluation-framework/

$$x_{toe,leg}(t_{leg}) = x_{c,leg} + a_{leg}\cos(\theta_{\text{traj}})$$
$$y_{toe,leg}(t_{leg}) = y_{c,leg} + b_{leg}\sin(\theta_{\text{traj}}) \quad (2)$$

where b_{leg} is the vertical ellipse semi-axis corresponding to the maximum ground clearance. In contrast with [18], θ_{traj} does not vary linearly with time, but is given by,

$$\theta_{traj} = \frac{\pi}{2}\left(\cos(\pi t_{leg} / T_{swing}) + 1\right) \quad (3)$$

By using (3), the derivative of $y_{toe,leg}$ as given by (2), varies smoothly from zero, at the beginning of the swing phase, to zero, at the end of the swing phase. This enhances the smoothness of the overall motion since the toes contact the ground with zero vertical velocity.



(a) (b)

Figure 3. (a) Evolution framework architecture, (b) Detailed model of the Laelaps II leg in Gazebo.

3. Proposed Evaluation Framework

The DRL algorithm performs foot placement planning by choosing the trajectory centers for each toe, see Figure 3a. This is passed to the trajectory planner that produces a sequence of toe positions along a semi-elliptical trajectory for every leg. Using the toe position as input and the inverse kinematics of the Laelaps leg as presented in [17], the hip and knee angles θ_1, θ_2 (Figure 2c) are calculated. In turn, these are sent to a PID controller to produce the motor torques to be applied on the simulated model. The evaluation framework was based on Gym [9].

3.1. *RL Task*

The RL problem is formulated as a Markov Decision Process (MDP) that is described for each time step t with a tuple (S, A, P, r, γ), where S is the current robot state, A is the action applied, P is the transition probability function from the current state s_t to the next state s_{t+1}, r is a reward value obtained due to the transition, and $\gamma \in [0,1]$ is a discount factor for the long term reward. The robot starts by exploring a stochastic environment to find an optimal behavior and increase cumulative reward values over subsequent timestamps t throughout the robot trajectory [18], where r_{a_t} is a reward function under action a.This function is defined so as to promote robot forward motion and punish divergence from its goal (e.g.: climb up a ramp)

$$r_{a_t} = w_f(x_t - x_{t-1}) - w_d(y_t - y_{t-1}) \quad (4)$$

where w_f and w_d are the weights for the forward and drifting terms respectively, with $w_f = 1$ and with the drift weight set to $w_d = 2$, to penalize more drifting motions. More, x and $y$the cordinates of the Laelaps CoM along the x and y axis.

The control algorithm receives from Gazebo proprioceptive sensory information s, regarding the Laelaps II body, i.e. roll/pitch/yaw angles and angular rates, building a compact observation space. The outputs are adjustments in the toe trajectories in the form of a vector of actions $[x_{C_1}, y_{C_1}, .., x_{C_4}, y_{C_4}]^T$, where $x_c \in [-0.1, 0.1]$ and $y_c \in [-0.55, -0.52]$. In the implemented algorithm, the toe trajectories are chosen individually for every leg as the robot climbs up/down

the ramp. In this way, the algorithm adjusts the CoM height accordingly for slope climbing, stabilizes the robot by bounding the roll and pitch angle of the body, and keeps the robot moving straight by bounding the yaw angle of the body.

This section gives an insight of the different categories of DRL algorithms. A focus is given to model-free algorithms as usually they are easier to train without the need to build descriptive models of complex environments. Model-free algorithms are classified in three main families, i.e. in *policy-based* for policy estimation, *value-based* for approximating value functions, or *combined methods* [16]. We will focus on the combined methods which provide the strengths of the other two methods and are popular in addressing continuous control tasks [16, 15]. These methods are based on the Actor-Critic neural network architecture. The actor network uses the policy μ to predict actions, while the critic network approximates the quality of the action applied given the current state [13]. The learning mechanism strategy follows an on-policy or off-policy manner. In turn, the policy can be trained using a stochastic or deterministic actor. In this work, we have chosen to evaluate algorithms able to handle continuous control problems and summarize the previously mentioned categories. Specifically, the selected algorithms are: Deep Deterministic Policy Gradient [14] (off-policy, deterministic), Proximal Policy Optimization [19] (on-policy, stochastic) and Soft Actor-Critic [18] (off-policy stochastic).

3.2. *Deep Deterministic Policy Gradient (DDPG) Algorithm*

DDPG is an algorithm with a deterministic actor network with a policy to predict a single action value. The actor learns to maximize the critic estimated action-value function *Q(s,a)* by following a policy gradient with *N* batch size as the objective J [19]

$$\nabla_{\theta^{\mu}} J = 1/\mathrm{N} \sum_{i} \nabla_{\alpha} Q(s, a \mid \theta^{Q}) \mid_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s \mid \theta^{\mu}) \mid_{s_i} \tag{5}$$

DDPG is trained using a reply buffer of fixed size saved experiences. The Ornstein-Uhlenbeck process as in [19] was used as the exploration strategy and it was implemented by adding noise to the actor output actions with temporally correlated values. For enhancing training stability, at the end of every training step the target network weights are updated by a soft update strategy that copies a small ratio of the learned updates in the main network [19].

3.3. *Soft Actor-Critic (SAC) Algorithm*

SAC is an algorithm with a stochastic actor network with policy $\pi_\theta(a_t \mid s_t)$ combined with a critic network for the approximation of the state value function *V(s)*, and a soft Q-function network [18]. The algorithm uses a modified objective function *J*:

$$J(\pi) = \sum_{t=0}^{T} E_{(s_t, a_t) \sim \rho_\pi} [r(s_t, a_t) + \alpha H(\pi(\square \mid s_t))] \tag{6}$$

where *H* is the entropy term and *a* is a weight parameter. This modification provides advantages for enforcing the exploration strategy, the robustness, and the stability in training [18]. Like DDPG, SAC is trained using a replay buffer of previously sampled data. In SAC, the target network trick is used for the critic network to stabilize training with soft update of target weights.

3.4. *Proximal Policy Optimization (PPO) Algorithm*

PPO uses a stochastic actor network for the policy prediction and a critic network for the value function V(s) approximation. To update the policy network, data is sampled using the current policy for a set of environment interactions. In turn, training is performed for several epochs on mini batches of the sampled data. The training sampled data is discarded after the policy update. PPO uses a clipped objective function $L^{CLIP}(\theta)$

$$L^{CLIP}(\theta) = \hat{\mathrm{E}}_t[\min(r_t(\theta)\hat{\mathrm{A}}_t, clip(r_t(\theta), 1-e, 1+e)\hat{\mathrm{A}}_t], \quad r_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta old}(a_t \mid s_t) \tag{7}$$

where the objective limits the ratio $r_t(\theta)$ between the old policy $\pi_{\theta(old)}$ and the new policy π_θ with a parameter ε. A_t is the advantage function [19]. Hence, the clipped objective function aims to reduce large updates avoiding the performance collapse by stepping away from the old policy.

4. Results

4.1. *Experiment*

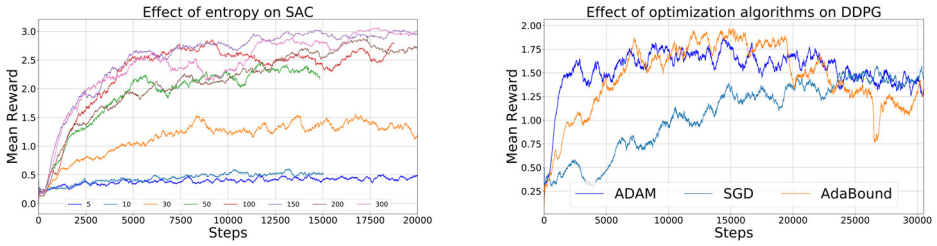
This work highlights the impact of different categories of state of the art DRL algorithms in the training of a detailed 3D Laelaps II model in the task of trotting up a slope of +10° utilizing the trajectory planner described in Section 2.3. The evaluated algorithms are compared with respect to the sample efficiency, stability, reproducibility, and generalization. More, they are tested after training in a different environment of a varying inclination ramp.

4.2. *Tuning the hyperparameters of the RL algorithms*

Concerning the implementation of the algorithms starting from DDPG, two hidden layers where chosen for the actor neural network (NN) and two hidden layers for the critic NN with learning rates of 10^{-4} and 10^{-3} respectively. Each NN consists of 500 neurons in the first layer and 400 neurons in the second. The batch size is 128 and the replay buffer size is 10K. Concerning SAC, the same hyperparameters as those in DDPG were applied plus the entropy parameter of $\alpha^{-1} = 150$. For the SAC soft Q-function network, the learning rate value is 10^{-3} and two hidden layers were used with 500 & 400 neurons. Concerning PPO, a fixed trajectory size of 256 environment steps was used. For the policy update, mini batches of size 64 were sampled from the trajectory, on which training of 30 epochs is performed. For actor and critic NNs, two hidden layers with 64 neurons each and learning rates of 10^{-3} were used. All algorithms were implemented using the PTAN package and are based on the baseline implementation of [15] and Openai Spinup[††].

4.3. *Discussion*

Hyperparameter tuning can have a dramatic influence on the performance of the algorithm. However, each algorithm is affected differently by varying hyperparameter values. Indicative results are: SAC is mainly sensitive to the entropy parameter α (see Figure 4a), the DNN optimizers have an impact on the performance of the algorithms. A comparative study of ADAM[‡‡], SGD[§§], and AdaBound[***] algorithms influence on DDPG is shown in Figure 4b. Adam optimizer helps DDPG to quickly converge to the maximum possible reward and stay at this value, AdaBound is a bit slower and SGD is third.



(a) (b)

Figure 4. (a) Effect of entropy parameter in SAC, (b) Optimizer effect on DDPG.

[††] https://spinningup.openai.com/
[‡‡] Adaptive Moment Estimation.
[§§] Stochastic gradient descent.
[***] Adaptive Gradient Methods with Dynamic Bound of Learning Rate: https://github.com/Luolc/AdaBound

DRL algorithms suffer from high variance in performance, and poor generalization [16, 20]. The performance of the algorithms is examined by running the same experiment several times with different random seeds as recommended in [20], with a validation scheme every 500 steps. The mean reward and 95% confidence bound over three runs are reported to have a sanity comparison between the algorithm's performance and their reproducibility capabilities. All algorithms were trained 3 times for 20K steps except from PPO which needed to train for 5K more steps to stabilize the mean reward (see Figure 5a after the blue vertical line). Figure 5a presents the training performance of DDPG, PPO, and SAC for all runs. SAC shows stability, speed and sample efficiency in training compared to PPO and DDPG. Using SAC, the robot learned to reach the top of the ramp after 8K steps, achieving reward ~3. In contrast, on-policy algorithms like PPO, suffer from sample efficiency and need more interaction with the environment in order to reach better results (25K steps). In addition, PPO has linear improvement in performance reaching DDPG after 19K steps. Last, notice that DDPG suffers the most from reproducibility (see shadowed part for every line in Figure 5a) while PPO doesn't and the 3 runs performance coincide at the beginning of every training.

Generalization of the trained policies are tested in trotting upward a varying inclination ramp (from 10° to 8°, ending with a 15° inclination, Figure 5c). Figure 5b presents the CoM position of the quadruped during this task. Laelaps II is an appropriate testbed to evaluate the algorithms in this scenario, since it cannot perform hip abduction which would help to stabilize the robot in cases of increasing body roll/yaw angles.

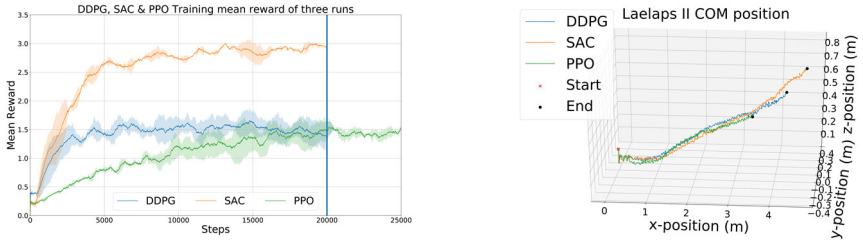


Figure 5. (a) Reproducibility for all algorithms over 3 runs, the shaded area is the 95% confidence bound, (b) Generalization scenario: Laelaps is trotting upwards a varying inclination ramp. SAC only succeeded to reach the top in this scenario, (c) The ramp of figure 5b as shown in Gazebo.

Last, Figure 6a presents that the body roll/yaw angles remain bounded when the quadruped is trotting upwards a ramp twice the length of the one used in training, while Figure 6b shows the applied saturation torque limit of 50Nm.

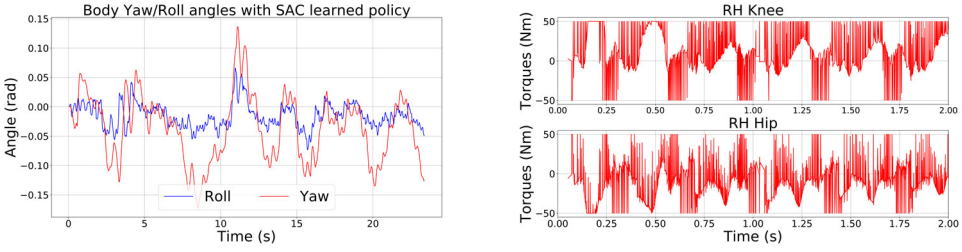


Figure 6 . (a) Bounded yaw and roll body angles for the test scenario of trotting upwards a ramp of +10°, (b) Right hint (RH) knee and hip torques for the same scenario.

4.4. *Conclusion*

This work employed DDPG, SAC, and PPO in a comparative study covering a wide area of DRL algorithms. A trajectory planning algorithm was used enabling the robot to move forward at a constant velocity performing a smooth car-like motion. Utilizing ROS and Gazebo, training and evaluation scenarios could be easily loaded. As a result, training speed, robustness to parameter/environment changes could be tested easily. The chosen training tasks were oriented towards evaluating DRL algorithms for quadrupedal slope handling. It was found that the SAC algorithm was less sensitive to parameter tuning than PPO and DDPG, and it succeeded in training and generalization. However, a lot of test runs were needed to tune the entropy parameter.

Acknowledgments

This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: 2182).

References

1. Taylor A., Patrick C., Kevin G., Alan F., and Jonathan W. H., "Fast online trajectory optimization for the bipedal robot cassie," *Robotics: Science and Systems (RSS)*, 2018.
2. G. Bledt, et al., "MIT cheetah 3: Design and control of a robust, dynamic quadruped robot," *Int. Conf. on Intelligent Robots and Systems (IROS)*, Madrid, 2018.
3. C. Gehring, et al., "Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot," *IEEE RAM magazine*, 23(1):34–43, 2016.
4. M. Hutter, et al. "Anymal-a highly mobile and dynamic quadrupedal robot," *Int. Conf. Intelligent Robots and Systems (IROS)*, Daejeon, Korea, Oct. 9-14, 2016.
5. Jemin Hwangbo, et al., "Learning Agile and Dynamic Motor Skills for Legged Robots," *Science Robotics*, 4(26), 2019.
6. Jie Tan, et al., "Sim-to-Real: Learning Agile Locomotion for Quadruped Robots," *arXiv:1804.10332*, 2018.
7. Joonho, L., Hwangbo, J., and Hutter, M., "Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning," *ArXiv abs/1901.07517*, 2019.
8. Xie, Z. et al., "Feedback Control for Cassie With Deep Reinforcement Learning.," *Int. Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, Oct. 1-5, 2018 1241-1246.
9. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540v1*, 2016
10. Duan Y., Chen X., Houthooft R., Schulman J., and Abbeel P., "Benchmarking deep reinforcement learning for continuous control". *Int. Conf. on Machine Learning*, 2016.
11. Mahmood, A. Rupam et al. "Benchmarking Reinforcement Learning Algorithms on Real-World Robots." *CoRL* (2018).
12. Lillicrap, et al., "Continuous control with deep reinforcement learning, CoRR," *arXiv:1509.02971*, 2015.
13. Sewak, Mohit, "Deep Reinforcement Learning: Frontiers of Artificial Intelligence," *10.1007/978-981-13-8285-7*, 2019.
14. Silver, David & Lever, Guy & Heess, Nicolas & Degris, Thomas & Wierstra, Daan & Riedmiller, Martin, "Deterministic Policy Gradient Algorithms," *Int. Conf. on Machine Learning, ICML*, 2014.
15. Maxim Lapan., *Deep Reinforcement Learning Hands-On*, (2nd ed.) *Packt Publishing*, 2020.
16. Laura Graesser, Wah Loon Keng., *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Addison-Wesley Professional, 2019.
17 Machairas, K. and Papadopoulos, E., "An Active Compliance Controller for Quadruped Trotting," *Mediterranean Conf. on Control and Automation (MED '16)*, Athens, Greece, June 21-24, 2016.
18 Haarnoja, T., Zhou, A., Abbeel, P. & Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". *Int. Conf. on Machine Learning*, 2018.
19 Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal Policy Optimization Algorithms." *ArXiv abs/1707.06347* (2017).
20 Islam, R., Henderson, P., Gomrokchi, M., Precup: D., "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control." *CoRR abs/1708.04133* (2017).