2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Feature Extraction for Effective and Efficient Deep Reinforcement Learning on Real Robotic Platforms

Peter Böhm[1], Pauline Pounds[2], and Archie C. Chapman[3]

Abstract— **Deep reinforcement learning (DRL) methods can solve complex continuous control tasks in simulated environments by taking actions based solely on state observations at each decision point. Because of the dynamics involved, individual snapshots of real-world sensor measurements afford only partial state observability, so it is typical to use a history of observations to improve training and policy performance. Such intertemporal information can be further exploited using a recurrent neural network (RNN) to reduce the dimensionality of the dynamic state representation. However, using RNNs as an internal part of a DRL network presents challenges of its own; and even then, the improvements in resulting policies are usually limited. To address these shortcomings, we propose using *gated feature extraction* to improve DRL training of real-world robots. Specifically, we use an untrained *gated recurrent unit* (GRU) to encode a low-dimension representation of the state observation sequence before passing it to the DRL training procedure. In addition to dimensionality reduction, this allows us to unroll the RNN by encoding the observations cumulatively as they are collected, thereby avoiding same-length input requirements, and train the RL network on the raw observations at the current step combined with the GRU-encoding of the preceding steps. Our simulation experiments employ gated feature extraction with the TD3 algorithm. Our results show that the GRU-encoded state observations improve the training speed and execution performance of the TD3 algorithm, improving the learned policies in all 19 test cases, exceeding the maximum achieved reward by over 38% in 8 and doubling the maximum achieved reward in three, while also outperforming a baseline implementation of SAC in 17 out of 19 environments. Moreover, the greatest improvement is seen in real-world experiments, where our approach successfully learns to balance a pendulum as well as a complex quadrupedal locomotion task. In contrast, the standard TD3 algorithm not only does not show any learning progress at all, but also repeatedly damages the hardware.**

I. INTRODUCTION

A major challenge in deep reinforcement learning (DRL) is that of bridging the *sim-to-real gap*, by demonstrating that real-world robots and devices can be effectively controlled using policies learned using powerful DRL methods [1], [2]. One way of dealing with this challenge is *dynamics randomization* [2], where robust policies are found by randomizing the simulation parameters during training. This trades robustness for optimality [3], and also decreases sample efficiency, often requiring hundreds of millions of samples [4], [5], [6].

This work was supported by the Advance Queensland Trusted Autonomous Systems Defence CRC Fellowship.

[1]Peter Böhm is a PhD candidate at ITEE, the University of Queensland, Australia, p.bohm@uq.edu.au

[2]Pauline E. I. Pounds is an Associate Professor at the University of Queensland, Australia, pauline.pounds@uq.edu.au

[3]Archie C. Chapman is a Senior Lecturer at the University of Queensland, Australia, archie.chapman@uq.edu.au

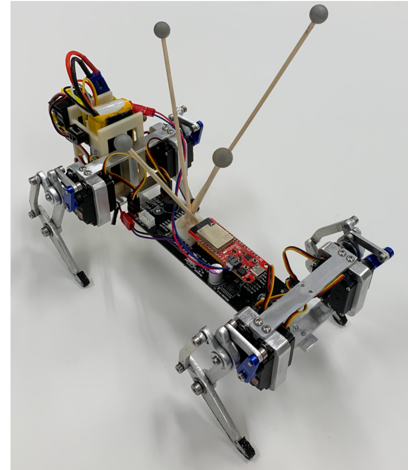

Fig. 1. Quadrupedal robot.

An alternative to domain randomization is training directly on the hardware [7], [8], [9], which avoids the sim-to-real transfer. However, because training on real robots is slow and they are susceptible to hardware failure, training needs to be much more sample efficient for DRL to be considered useful in these settings, lest it lead to excessive wear or even breakdown of the physical components.

In this context, a key aspect of the sim-to-real gap is finding useful state representations for DRL algorithms to learn on using the available sensor data. In particular, a sequence of state observations, measured by sensors, can be transformed into a compact representation of the state using a recurrent neural network (RNN) [10]. Most of the works taking this approach focus on RNNs as a solution to partially-observable Markov decision processes (POMDP) [11], and/or are confined to discrete action spaces or image input from the Atari Learning Environment [12]. However, using RNNs for state *feature extraction*, by pre-processing state observations to reduce their size in settings with continuous action spaces and non-image inputs has been largely unexplored. Thus, this paper considers DRL for continuous control in combination with RNNs for state feature extraction and representation size reduction.

Current approaches that apply RNNs *within* the DRL training process to reduce the size of the state representation face three technical challenges: (i) there is a considerable computational cost of unrolling the sequence of observations at each training step, (ii) for training, the same sequence length within a mini-batch is required, and (iii) in many situations, the raw and exact sensory observations are required

to decide the next action rather than the RNN representation of the past n steps. Yet the improvements resulting from using RNNs are usually very limited; for example, DRQN outperforms standard DQN only on two Atari games [10]. This paper addresses these issues by proposing a novel approach to feature extraction for DRL.

To address the first two issues above, we use an untrained GRU network [13] as a variable sequence encoder for feature extraction, and place it outside the DRL network. This allows the observations to be unrolled only once per training, thereby avoiding the repeated computational cost. In addition, the observations are encoded cumulatively as they are collected without any fixed sequence length limitation. The DRL network is then trained to interpret this GRU-based encoding. Our approach is motivated by the long-standing use of untrained layers in neural networks for classification and regression tasks, proposed as early as [14] to improve the training of multi-layer feed-forward networks and widely exploited since then (e.g. [15]). Moreover, recent results extend universal approximation of some classes of RNNs with randomly generated internal weights to reservoir systems [16], which are a class of dynamics similar to those that arise in continuous control and robotics applications. We choose to use GRUs in particular as they are computationally more efficient than Long Short-Term Memory (LSTM), another RNN architecture, thanks to GRU's simpler structure. GRUs have also been shown to outperform the LSTM on non-language modeling tasks with shorter sequences [17].

To address the third issue, we train the RL network on the raw observations at the current step combined with an encoding of the preceding steps. This gives the RL network access to both less detailed historical information as well as exact and unprocessed current data. In contrast, existing DRL approaches that employ preprocessing RNNs use the output of the RNN layer as an input to the Q-Network, or actor/critic networks in continuous action spaces [10], [18]. By doing this, part of the observation input is lost or diluted in the history encoding. Our results demonstrate that the next action inference benefits from both exact sensory observations at the current step (i.e. encoder reading, heading, etc.) as well as historical context provided by the RNN hidden state. Additionally, this historical context provides a proximal smoothing effect in state observations which in turn reduces abrupt action changes between steps and stress on the physical components, which is an important consideration in DRL training on real-world robots.

To demonstrate the effectiveness of our gated feature extraction approach, we implement it with TD3 [19] and rigorously evaluate it across a comprehensive suite of 19 challenging simulated control tasks. Our empirical results show that our algorithms substantially outperform the current state of the art on both maximum reward and sample efficiency. We further test our approach on real-world, low-cost pendulum and quadrupedal locomotion tasks. These are simple tasks in simulation, however, they can become very challenging with limited and noisy sensory observations and limited precision control. Given these conditions, standard TD3 did not show any learning progress. Furthermore, the aggressive exploration repeatedly lead to physical damage of the motors. In contrast, training the same TD3 algorithm on the GRU-encoded state representations combined with state observations showed steady learning progress, and successfully solved both environments without damaging the physical hardware. Taken together, these simulated and real-world experiments demonstrate that our untrained GRU feature extraction approach reduces the sim-to-real gap and moves us towards the goal of using DRL to train and deploy field devices.

The structure of the paper is as follows: Section II discusses related work, followed by the technical preliminaries in Section III. Section IV introduces the gated feature extraction. We report results from simulated experiments in Section V, and real-world experiments in Section VI. Section VII concludes.

II. Related Work

Using RNNs to improve the performance of DRL has been around for some time, and its various aspects have been studied in several publications. The majority of these works focus on *deep Q-networks* (DQNs) [20] within the Atari Learning Environment [12], with many using the RNN in a partially-observable Markov decision process (POMDP) framework.

Deep recurrent Q-networks (DRQN) were introduced in [10] as a combination of a *long short-term memory* (LSTM) [21] and a DQN, to address partial observability in Atari 2600 games. In the original DRQN implementation, the output of the convolutional layer is passed through the LSTM and its outputs become Q-values (after passing through a fully-connected layer). Because LSTM requires a sequence of multiple frames, two types of updates were examined: (i) bootstrapped sequential updates and (ii) bootstrapped random updates. The first is trained on entire (randomly selected) episodes and the RNN is unrolled from the beginning to the conclusion of the episode. The second is trained over $n = 10$ consecutive frames selected randomly within (randomly selected) episodes. Neither is ideal because unrolling the full episode violates DQN's sampling policy and unrolling over n frames limits the history used by the RNN. Both DRQN variants performed is similarly to the plain DQN.

Another work extending DRQN, [22], introduced the *action-specific deep recurrent Q-network* (ADRQN), which couples actions and observations before passing them through an LSTM. We use a similar approach in one of our innovative variants, coupling current observations and actions before passing them to the GRU.

The idea of separating the RNN from the RL network is investigated in [23]. The RNN/LSTM is trained to predict next observations and rewards and the learned hidden states are then passed as the input to DQN. The RNN and DQN are backpropagated in two separate steps. As this architecture involves backpropagation of the RNN at each training step, it requires history sequences within a minibatch to be of equal size and is computationally expensive.

The use of RNNs in continuous action spaces is investigated in [18]. Their architecture extends the TD3 algorithm [19] with the RNN implemented as a GRU [13] layer in the actor and critics. As such, the architecture closely mimics DRQN, but extends TD3 instead of DQN. As a result it suffers from the same drawbacks and requires a fixed-size history sequence as well as high computational cost.

In all previous work, the observation sequence unrolled by the RNN is limited to 10 frames to improve computational efficiency. This, however, limits the RNNs ability to learn situations that span more time steps. It also presents a significant computational cost. In all cases the hidden state of the RNN is used as a direct input of the RL network, which may affect the accuracy of the sensory observations used in training. Our approach overcomes these shortcomings.

III. Technical Preliminaries

A. Reinforcement Learning and Twin Delayed DDPG (TD3)

In reinforcement learning (RL), an agent interacts with an unknown environment $\mathcal{E}$ while trying to maximize its received reward. At each time step t, the agent selects an action $a \in \mathcal{A}$ based on the state observation $s \in \mathcal{S}$ with respect to its policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. Based on this action, it receives a reward $r \in \mathcal{R}^N$ and a new state observation s'. The agent's goal is to learn a policy π that maximizes the expected discounted cumulative reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, where γ is a discount factor.

Estimates of the optimal action values can be learned using Q-learning, a form of temporal difference learning [24]. However, most interesting problems are too large to learn all action values in all states separately. Instead, we can learn a parameterized value function $Q(s, a; \theta_t)$. The standard Q-learning update for the parameters after taking action A_t in state S_t and observing the immediate reward R_{t+1} and resulting state S_{t+1} is then $\theta_{t+1} = \theta_t + \alpha(Y_t^{\mathrm{Q}} - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t)$ where α is a scalar step size and the target Y_t^{Q} is defined as $Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t)$.

In continuous action spaces, the goal is to optimize a parameterized policy π_ϕ by maximizing its expected return $J(\phi) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_0]$. This can be done using the *deterministic policy gradient* algorithm [25]:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s \sim p_\pi}\left[\nabla_a Q^\pi(s, a)|_{a=\pi(s)} \nabla_\phi \pi_\phi(s)\right].$$

The *deep deterministic policy gradient* (DDPG) algorithm extends deterministic policy gradients by using deep neural networks [26]. Building on this, *twin delayed DDPG* (TD3) improves on DDPG by addressing the over-estimation of Q-values in the critic network, specifically by innovating in three areas [19]: (i) using two critic networks, (ii) using delayed updates of the actor to reduce per-update error, and (iii) applying action noise regularization. We use TD3 in Sections V and VI to demonstrate the value of adding gated feature extraction to solving robot control problems.

Algorithm 1 R-TD3

1: Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
2: Initialize target networks $\theta_1' \leftarrow \theta_1$, $\theta_2' \leftarrow \theta_2$, $\phi' \leftarrow \phi$
3: Initialize replay buffer $\mathcal{B}$
4: Initialize the RNN with state observation s received from $env.reset()$ and h_t initialized to 0s
5: Combine the state observation s with the last layer of the returned output as s_o
6: **for** $t = 1$ **to** T **do**
7: Select action with exploration noise $a \sim \pi_\phi(s_o) + \epsilon$,
8: $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
9: Unroll the RNN for additional step using the s'
10: Combine the observation s' with the last layer of the returned output as s'_o'
11: Store transition tuple (s_o, a, r, s'_o') in $\mathcal{B}$
12: Standard TD3 training (actor/critic gradient clipped at 0.1)
13: **if** episode ended **then**
14: Re-set the hidden state of the RNN with state observation s received from $env.reset()$ and hidden state h_t initialized to 0s
15: Combine the state observation s with the last layer of the returned output as s_o
16: **end if**
17: **end for**

B. Recurrent Neural Networks for State Observation Encoding

A recurrent neural network (RNN) uses its internal state, referred to as hidden state, $\mathbf{h}$, to process a variable-length sequence $\mathbf{x} = (x_1, \dots, x_T)$ input. At each time step t, this hidden state $\mathbf{h}_{\langle t \rangle}$ is updated through

$$\mathbf{h}_{\langle t \rangle} = f\left(\mathbf{h}_{\langle t-1 \rangle}, x_t\right), \tag{1}$$

where f is a non-linear activation function. By training to predict the next symbol in a sequence, the RNN learns a probability distribution of the sequence.

A GRU [13] is a network architecture that encodes a variable-length sequence into a fixed-length vector representation, and then decodes this vector back into a variable-length sequence. The GRU encoder reads each symbol of an input sequence $\mathbf{x}$ sequentially. Processing each symbol updates the the hidden state of the RNN according to (1). Thus, the hidden state, $\mathbf{h}_{\langle t \rangle}$, is a summary of the processed input sequence. In RL, as the agent moves through the episode, a sequence of observations or action-observation pairs can be considered as such a sequence, and the hidden state, $\mathbf{h}_{\langle t \rangle}$, is a compact representation of the state information.

IV. Dimensionality Reduction Through RNN

This section contains our main technical innovation, capturing information from an observation history by feature extraction using an untrained GRU that is decoupled from the RL network. Our framework is outlined in Algorithm 1.

The simplest way of adding an observation history to DRL training is to stack a sequence of preceding observations. However, this quickly increases the dimensionality of the input layer, which motivates our investigation into the dimesion-reduction power of RNNs. Previous approaches

incorporate an RNN *within* the RL network, usually using LSTM as the first layer before the Q-network or before the actor/critic networks. This has several consequences. RNNs require a sequence of observations as their input, and because the training is done using mini-batch gradient descent, each sequence within the mini-batch must be of equal length. In addition, to reduce the computational cost caused by unrolling the entire sequence at each training step, the sequence length is usually limited to 10 observations. However, many tasks, even only moderately complex ones, span (and depend on) many more steps, so this limits the effectiveness of the RNN layer.

A. GRU Encoding

Our approach decouples the RNN layer from the RL network by using a separate GRU network. Unlike in other approaches, this network is not used for prediction but rather for encoding of the variable-length sequence of observations into a fixed-length vector representation.

The observations are encoded cumulatively, with each step further unrolling the hidden state of the GRU (Algorithm 1, line 9). As a result, the transition for step 10 will contain the encoded history of observations from steps 1 - 10, the transition for step 15 contains observations from steps 1 - 15, etc. Because each sequence is encoded individually, there is no same-length-batch requirement. The hidden state is reset at the start of each episode (lines 4 and 14). The GRU is initialized at the start of the training, and this initialization will produce a consistent encoding of the same (or similar) sequence of observations throughout the training. The encoding generated by the GRU network is then used as an additional input to the RL network. The actor and critic within the RL network are trained to interpret this encoding and use it for prediction.

We achieve best results with 3 layer GRU and the hidden state of 128 (256 for more complex environments).

B. RL Network Input and RNN Input

In other approaches (e.g. [10]), the hidden state of the RNN becomes the entire input of the Q-network. This causes a part of the current step's observation to be lost or combined with the rest of the sequence. This can be counter-productive in multi-dimensional, non-image based situations where the next action depends on the exact observations at the current step.

In our approach, the GRU encoding is added to the raw observations at the current step (line 10). This provides the RL network with both the memory of the previous steps as well as the exact current observations (line 12). In addition, the combined input is cached at each step with the rest of the transition data in the replay buffer (line 11). Because each step is encoded only once, the additional computations required are limited and the resulting speed is on par with the baseline algorithms (i.e. TD3) without the RNN.

When it comes to RNN input, we are presented with two options, to pass in: (i) state observations only, or (ii) state observations and the action that lead to those observations. Most of the previous works use the past ten state observations as the RNN input and the RNN hidden state becomes the input of the Q-network. The causal relationship between the action the resulting observation is explored in [22], in which they concatenate both before passing them to the RNN, in an attempt to improve performance. We test both approaches. In the remainder, the first variant is denoted as *R-* and the latter as *RA-* in the name of the algorithm.

V. Experiments in Simulation

To ensure robustness of our results, we evaluate and compare the performance of the algorithms across selected tasks from three simulation environments: (i) MuJoCo continuous control tasks [27] interfaced through OpenAI Gym [28], (ii) PyBullet [29], and (iii) the DeepMind Control Suite [30]. To allow for reproducible results, we use the tasks without any modifications to the publicly available environment or reward. For each algorithm/environment combination, we train 10 different instances using seeds $0-9$. Following the TD3 paper, the reward is calculated as an average of 10 test runs without exploration every $5,000$ training frames. The more complex tasks, Humanoid, HumanoidStandup, and DM Humanoid: Walk are trained for 10M frames, and the rest of the tasks for 1M frames. We benchmark against TD3 [19], DDPG [26] and SAC [31].

A. Results

Table I shows average returns of the top 5 performing seeds for each batch. We use the top 5 seeds to eliminate the seeds that fail to learn at all, which is inevitable in very complex tasks such as the Humanoid or HumanoidStandup, regardless of the training algorithm. Results for DDPG are omitted as it was consistently outperformed. Figure 2 shows learning curves for several environments with variants of our modifications outperforming the base TD3 algorithm in both sample efficiency and final performance.

Overall, these simulation results demonstrate a performance improvement from our GRU pre-processing modifications to the TD3 algorithm. Our algorithms yields solution quality improvements of over 38% in 8 out of 19 test cases, and doubles the maximum achieved reward in three (HumanoidStandup, Minitaur and DM Hopper: Hop) when compared to TD3. It also outperform a baseline implementation of the SAC algorithm in 17 out of 19 environments.

VI. Real-world Experiments

To confirm the effectiveness of our approach in real-world conditions, we tested it with a real-world implementation of a swing up pendulum environment and quadrupedal locomotion task. Despite most of the DRL research focusing on high-cost and high-compute robotics [4], we use low-cost and commodity grade hardware to show that our approach generates learning progress even in the challenging conditions of low precision and variable delays where off-the-shelf DRL approaches fail.

TABLE I

AVERAGE RETURNS OF TOP 5 PERFORMING SEEDS. ± CORRESPONDS TO A SINGLE STANDARD DEVIATION.

Environment	R-TD3	RA-TD3	SAC	TD3
OpenAI Gym (MuJoCo) Environments				
Ant-v3	5,669.3 ± 103.3	**5,705.8 ± 173.3**	5,026.6 ± 220.9	5,437.6 ± 89.6
HalfCheetah-v3	**11,006.9 ± 653.8**	10,251.7 ± 963.4	8,291.8 ± 643.7	9,854.8 ± 346.7
Hopper-v3	**3,844.3 ± 62.4**	3,801.5 ± 40.7	3,603.1 ± 33.4	3,710.2 ± 37.7
Humanoid-v3	6,913.7 ± 456.9	7,209.7 ± 357.9	**7,744.3 ± 510.6**	5,681.6 ± 27.8
HumanoidStandup-v2	181,946.4 ± 37,170.7	**220,533.4 ± 53,563.0**	203,756.8 ± 39,379.3	161,700.4 ± 2,671.1
Swimmer-v2	**98.9 ± 37.0**	84.0 ± 33.3	78.8 ± 9.3	65.0 ± 16.3
Walker2d-v3	**6,413.4 ± 195.4**	6,044.1 ± 305.5	4,606.2 ± 151.7	4,920.3 ± 284.1
PyBullet Environments				
AntBulletEnv-v0	3,142.7 ± 160.5	**3,176.5 ± 109.1**	1,396.8 ± 227.7	2,424.1 ± 97.1
HalfCheetahBulletEnv-v0	**3,189.7 ± 193.5**	3,130.5 ± 265.4	2,008.8 ± 194.9	2,490.7 ± 155.6
HopperBulletEnv-v0	2,667.2 ± 70.5	**2,717.6 ± 94.9**	2,457.6 ± 58.1	2,614.3 ± 52.0
MinitaurBulletEnv-v0	**20.6 ± 4.5**	14.6 ± 4.4	16.3 ± 3.2	5.4 ± 4.4
Walker2DBulletEnv-v0	2,254.2 ± 74.1	**2,370.1 ± 53.6**	1,744.0 ± 55.2	2,060.7 ± 48.8
DeepMind Control Suite Environments				
Finger: Turn Hard	827.4 ± 94.5	757.1 ± 40.5	**909.4 ± 54.0**	767.7 ± 48.0
Fish: Swim	**771.8 ± 22.9**	743.5 ± 34.9	344.5 ± 35.4	335.8 ± 37.7
Hopper: Hop	**192.7 ± 28.3**	164.9 ± 22.7	144.9 ± 42.1	103.2 ± 16.8
Humanoid: Walk	**610.9 ± 355.8**	424.4 ± 385.9	556.7 ± 45.1	210.6 ± 278.6
Swimmer: 15	**654.0 ± 65.7**	614.9 ± 52.6	521.9 ± 34.0	534.4 ± 35.9
Walker: Run	**674.6 ± 33.9**	642.7 ± 87.5	625.0 ± 34.4	582.8 ± 73.3
Walker: Walk	972.7 ± 3.7	**974.3 ± 3.4**	973.0 ± 3.5	970.3 ± 2.4

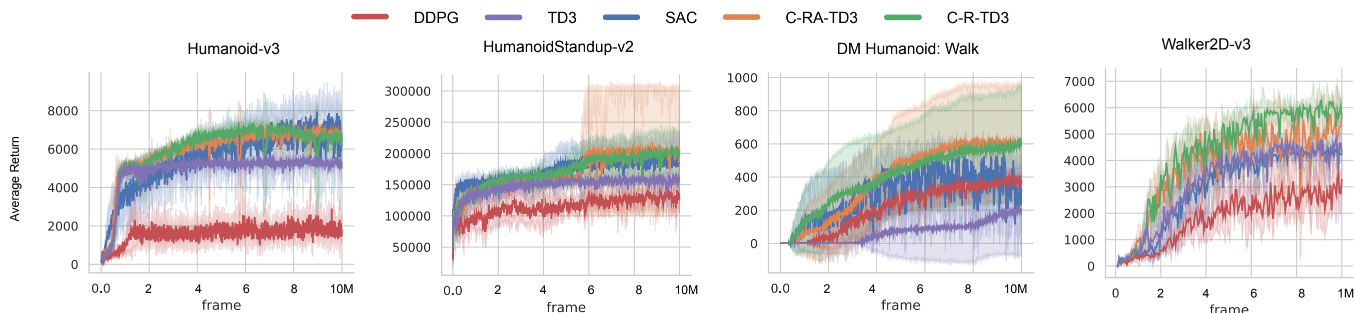

Fig. 2. Learning curves for the continuous control tasks.

A. Pendulum Apparatus

We use a minimal version of a swing-up pendulum. The machine consists of a rectangular aluminum extrusion mounted on a servo using a servo block. At the end of the extrusion is an encoder with the pendulum. The machine is controlled by an MCU (ESP32-S2). The communication between the MCU and the training machine is via WiFi, using a Pub/Sub protocol over an MQTT message broker. All components are low-cost and commodity grade and the design is open sourced.

The continuous space action is a value from interval $[-1, 1]$, representing the leftmost and rightmost positions of the servo. To prevent too large movements, this value u_t is passed through an exponential filter. In the discrete setting, the action is a number from interval $[0, 4]$ representing one of the possible actions: (i) left fast, (ii) left slow, (iii) stop, (iv) right slow, and (v) right fast. A safety loop halts the current action if either the right-most/left-most position is reached or the pendulum spins at 2 or more rotations per second.

Between the episodes, the servo motor is stopped for 2 seconds. Most of the time, this leaves the pendulum swinging with some residual momentum and at a random starting position. This randomized starting state is important to train the swing-up part of the task.

State Observations: There are two physical measurements: (i) the pendulum angle measured by an absolute encoder, and (ii) arm angle given by servo position. Other state observations are calculated based on changes in position and time since the last measurement. These calculations run on the MCU at a streaming frequency to keep them consistent.

Reward Function The reward function used is: $R = -\theta^2 - 0.5 \times \omega^2$, where θ is an angle measured from the upright position in radians and ω is revolutions per second.

B. Quadrupedal Robot

To confirm that our architecture can scale to more complex problems and continuous action spaces, we tested it on a complex task of quadrupedal locomotion in a specific

Fig. 3. Pendulum apparatus.

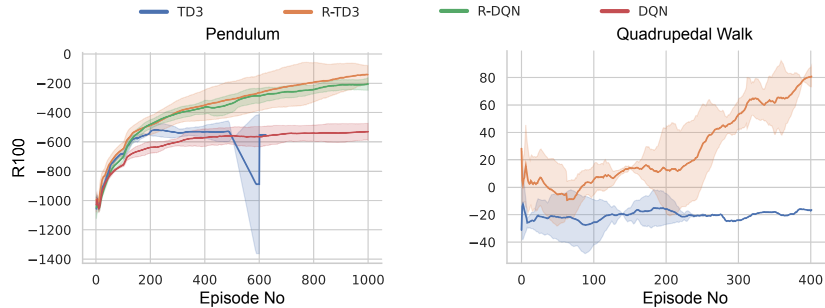

Fig. 4. Learning curves of the real-world experiments.

direction. The robot, shown in Figure 1, is driven by 8 servos, each leg is controlled by two servos. The servos are controlled by an on-board MCU ESP32-S2, and the communication setup is the same as with the pendulum.

Action Space: There are 2 servos per leg resulting in continuous action space size of eight inputs. The actions are real numbers $[-1, 1]$ and these values are mapped to the allowed range of each servo. The mapped value is passed through an exponential filter and the resulting position is then actuated.

Observation Space: The state observations combine readings from multiple sensors, including a position of each servo, orientation yaw, pitch, roll, Δyaw and distance Δd traveled in the direction given by the control signal. Unlike in many simulated environments, the distance is measured with respect to the control signal represented by the yaw of 0.

Reward Function The reward function $R = \Delta d + \Delta yaw - 0.1 \times yaw - 0.1 \times pitch - 0.1 \times roll$ rewards forward motion and marginally penalizes differences from level orientation.

For both R-TD3 and TD3, to reduce the initial random exploration, the replay buffer was seeded by the first 100k transition collected during the first set.

C. Pendulum Results

Each experiment was run for 3 sets, 1000 episodes of 500 steps each, except for plain TD3, which damaged the servo every time after around 500 episodes. We consider the environment solved at reward of -200, at which point it takes several swings to bring the pendulum upright and it remains upright until the end of the episode. Because of the limited precision sensors and variable delays in communication channel and actuation, this problem is considerably more difficult than simulated versions that are easily solvable by most algorithms in minutes. This is evidenced by neither plain TD3 nor plain DQN making any learning progress. Both R-TD3 and R-DQN managed to solve the environment.

Of special note: The aggressive exploration of the plain TD3 lead to the metal gears on the servo being stripped during each run. It also caused extensive wear to the servo block and it needed to be replaced. In contrast, the R-ed algorithms were training almost continuously for weeks (as part of a different study) without any damage.

D. Quadrupedal Locomotion Results

The R-TD3 experiments were run for 3 sets, each taking 400 episodes of 500 steps. In each of the R-TD3 sets, the robot learns a quasi galloping gate that allows it to move as much as 265cm per episode during the no exploration test. As evident from the supplementary video, the robot also makes course correction steps throughout to keep moving in the required direction.

We could only finish two of the TD3 experiments, during which 4 servos as well as the battery elimination circuit were damaged. There was no evidence of learning during the training, with the policy generating the same actions during the non-exploration tests and the robot not moving at all.

Our conjecture is that there are two reasons behind the less wear on the hardware when using the GRU encoding: (i) the network actually learns and a good policy requires smooth movements and smaller action changes, and (ii) the history encoding makes state observation/feature changes more gradual, effectively applying a proximal smoothing to the next action inference.

VII. Conclusion

This paper investigates the use of RNNs to improve DRL training in robotic control tasks. We propose a novel architecture that extends the existing TD3 algorithm by combining the current step state observations with a GRU encoding of the sequence of all the preceding observations. By using the GRU network as a variable sequence encoder placed outside of the RL network, our approach addresses three challenges present in using RNNs in DRL training: (i) computational cost of unrolling the observation sequence at each training step, (ii) restricting the observation sequence to a fixed length within a mini-batch requirement, and (iii) dilution of the exact observations from the current step in the RNN output. Our experiments on a broad set of robotics RL tasks across three control suites show that employing our architecture improves on the state of the art DRL algorithms, especially in test environments with high observation and action space dimensionality. Moreover, our experiments in real-world pendulum and quadrupedal locomotion environments indicate that the same approach helps with DRL training in challenging real-world conditions characterized by low precision, noisy observations, limited control and latency.

References

[1] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, H. Kress-Gazit, S. S. Srinivasa, T. Howard, and N. Atanasov, Eds., 2018. [Online]. Available: http://www.roboticsproceedings.org/rss14/p10.html

[2] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[3] J. Luo and K. Hauser, "Robust trajectory optimization under frictional contact with iterative learning," *Autonomous Robots*, vol. 41, no. 6, pp. 1447–1461, 2017.

[4] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Science Robotics*, vol. 5, no. 47, 2020.

[5] Z. Xie, X. Da, M. van de Panne, B. Babich, and A. Garg, "Dynamics randomization revisited: A case study for quadrupedal locomotion," in *2021 IEEE Int. Conf. Robotics and Automation (ICRA)*. IEEE, 2021, pp. 4955–4961.

[6] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, 2019.

[7] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

[8] S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan, "Learning to walk in the real world with minimal human effort," in *Conference on Robot Learning*. PMLR, 2021, pp. 1110–1120.

[9] M. Bloesch, J. Humplik, V. Patraucean, R. Hafner, T. Haarnoja, A. Byravan, N. Y. Siegel, S. Tunyasuvunakool, F. Casarini, N. Batchelor, *et al.*, "Towards real robot learning in the wild: A case study in bipedal locomotion," in *5th Ann. Conf. Robot Learning*, 2021.

[10] M. J. Hausknecht and P. Stone, "Deep recurrent Q-Learning for partially observable MDPs," in *2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015*. AAAI Press, 2015, pp. 29–37. [Online]. Available: http://www.aaai.org/ocs/index.php/FSS/FSS15/paper/view/11673

[11] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva, "Deep attention recurrent Q-network," *arXiv preprint arXiv:1512.01693*, 2015.

[12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[13] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[14] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Systems*, vol. 2, no. 3, pp. 321–355, 1988.

[15] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, 2006.

[16] L. Gonon, L. Grigoryeva, and J.-P. Ortega, "Approximation Bounds for Random Neural Networks and Reservoir Systems," *arXiv e-prints arXiv:2002.05933*, Feb. 2020.

[17] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International conference on machine learning*. PMLR, 2015, pp. 2342–2350.

[18] K. Zhang, Z. Hou, C. W. de Silva, H. Yu, and C. Fu, "Teach biped robots to walk via gait principles and reinforcement learning with adversarial critics," *arXiv preprint arXiv:1910.10194*, 2019.

[19] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, 2018, pp. 1587–1596.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for POMDPs," *arXiv preprint arXiv:1704.07978*, 2017.

[23] X. Li, L. Li, J. Gao, X. He, J. Chen, L. Deng, and J. He, "Recurrent reinforcement learning: a hybrid approach," *arXiv preprint arXiv:1509.03044*, 2015.

[24] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[25] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 387–395. [Online]. Available: http://proceedings.mlr.press/v32/silver14.html

[26] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning." in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html

[27] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[29] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2019.

[30] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[31] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.