Reinforcement Learning for a Continuous DC Motor Controller

1st Bucur Cosmin
Faculty of Engineering
Ovidius University of Constanta
Constanta, Romania
bcmircea@univ-ovidius.ro

2nd Tasu Sorin
Faculty of Engineering
Ovidius University of Constanta
Constanta, Romania
sorin.tasu@365.univ-ovidius.ro

2023 15th International Conference on Electronics, Computers and Artificial Intelligence (ECAI) | 979-8-3503-2138-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ECAI58194.2023.10193912

***Abstract*— Electric motors control is a very knows topic in research and most of the activities are drawn towards classic PI or model predictive control methods. Implementing reinforcement learning techniques in the field of motor control depends on fidelity environments, types of considered motors and modeled power electronics used for control. Training an RL speed controller means finding an optimal control policy by offline training using an environment, before implementing it in a real-world scenario. Different environments and techniques have been developed for training RL controllers, most of them being extensions of Open AI gym environments. This paper presents a trained RL speed controller, developed through Reinforcement learning techniques, specifically TD3 RL algorithm, applied to permanently excited dc motors. In this work, the open-source Python package gym-electric-motor (GEM) [1] is used for environment setup, and pytorch framework for developing the controller.**

Keywords—Electrical motors, Reinforcement learning, DDPG, dc machine, artificial intelligence, OpenAI gym

I. INTRODUCTION

Applying reinforcement learning in electrical motors control is a new topic and a good candidate for verifying the performance of such techniques other that well knows training environments such Atari games. Implementing the trained algorithms in real-world scenarios like factory floors is not trivial since typical training methos implies simulated environments that can greatly deviate from the real implementation of the industrial application. The RL methods are trained in simulated environments or in HIL (hardware in the loop) setups. Working with simulated environments enables training without actual hardware being available but imposes some risks due to differences and nonlinearities in actual real-world applications.

In the field of motor control, the main control approach is MPC (model predictive control) with mature theory but recently, the reinforcement learning has been promoted with relatively good results. A good comparison between advantages and drawbacks of reinforcement learning and MPC is presented in [1] and [2]. In relation to MPC type of controls, in the power electronics field there is also the approach of imitation, which implies training artificial neural networks to imitate the MPC controller [3].

In the field of power engineering different DRL (deep reinforcement learning) techniques has been tried, depending of the type of electrical machine, environment used and type of action space (discrete or continuous) [4,5,6]. Most of the initial RL algorithms, like DQN, were discrete action space, like ON/OFF thyristors commands, being suitable for a small number of applications. Implementing continuous action space for motor controllers is not trivial and we used for this paper the TD3 variation of Deep Deterministic Policy Gradient (DDPG).

"Deep Deterministic Policy Gradient (DDPG) is an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces". It combines the actor-critic approach with insights from DQNs: in particular, the insights that 1) the network is trained off-policy with samples from a replay buffer to minimize correlations between samples, and 2) the network is trained to approximate a deterministic target policy" [7]. DDPG is a well-known algorithm for extending the typical DQN to continuous actions space, such as in our case, but also has some drawbacks like instability, Q values overestimation or failing in particular applications.

One of the modifications of typical DDPG algorithm is TD3 (*twin delayed DDPG*), that we will use in this paper. TD3 is an actor critic method based on DDPG but improves the learning process and stability by using two Q-value functions for actor and the critic. Stability is improved by delaying the update of the target and policy networks and minimizing the updates of targets through smoothing. There are other variants of the algorithms that try to mitigate some possible problems of TD3, like bias underestimation, one of them being Twin Average Delayed DDPG (TAD3), presented in [8].

This paper presents the results of implementation of the TD3 algorithm for controlling a DC motor, showing the different impacts of parameters in training speed, training stability and total cumulative reward. After establishing an optimal value for those parameters, the controller is compared with a typical tunned PI speed controller for different setpoint profiles.

One of the important choices in training reinforcement learning algorithms is choosing the environment. There are several environments available like gym-electric-motor (GEM) based on OpenAI Gym environments (see [9], [10]) and MATLAB reinforcement learning toolbox.

The environment used is called *gym electric motor* (GEM) due to its open-source nature and the fact that we've implemented the training in pytorch framework. This also enables the exporting of the model in ONNX format and easily integrating in hardware platforms like beaglebone, raspberry for verification on real motors setup.

The GEM environment can be used for different control objectives: speed, current or torque control for dc motors in different configurations. The environment also includes the converters in discrete or continuous action spaces, mechanical loads, and voltage supplies. In our paper we used a two-quadrant converter with continuous action space for a permanently excited dc motor for speed control without constraints. The chosen configuration is suitable for training in typical laptop GPUs because of the small number of states (speed, speed reference).

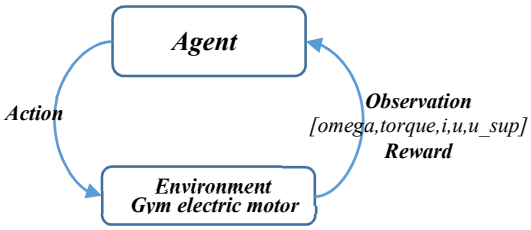

Fig 1. Basic reinforcement learning setting

In our case the environment will be represented by a permanently excited dc motor as part of a Supply Converter Motor Load System (SCML):

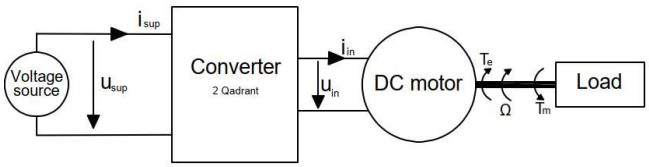

Fig 2. Supply converter motor load system

The control is modelled as a two-quadrant converter:

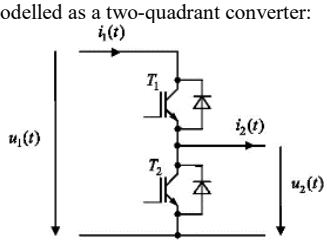

Fig 3. Two Quadrant Converter

For continuous control the action is a number between 0-1 with duty cycle for the upper transistor being *Action* and for lower transistor the duty cycle is *1-Action*.

II. ENVIRONMENT SETUP

For the development of the DRL speed controller we instantiate the environment Continuous Speed Control DC Permanently Excited Motor Environment from the gym package. For this environment the state variables are ['omega', 'torque', 'i', 'u', 'u_sup'] and the simulation step t=1e-4s. The Dc permanently excited motor is a dc motor with a permanent magnet instead of the excitation circuit with following parameters:

- Armature circuit resistance – 16e-3 Ohm
- Armature circuit inductance – 19e-6 H
- Magnetic Flux of the permanent magnet – 0.165 Wb
- Moment of inertia of the rotor – 0.025 kg/m^2

The control policy is suitable for a 2-quadrant converter with continuous action space. The control action is the duty cycle of the transistors. For upper transistor the duty cycle is Action and for lower transistor is 1-Action, with action between 0-1.

Reference generator that generates a reference for speed is a Wiener Process with sigma between (0.001, 0.01) and mean = 0.

A reward is received upon each action, which is calculated as the weighted sum of errors with a certain power:

$$r_{wse} = -\sum_i w_i\left((|\omega_i - \omega_i^*|)/2\right)^2 + b \qquad (1)$$

For refence purposes the values used are ω_i =1 and b=1.

III. TD3 IMPLEMENTATION

The TD3 algorithm is implemented in python using pytorch framework, following the initial paper which introduced the algorithm [11].

The TD3 Agent is composed of four networks, 2 critics and 2 actors (one of each is a deep copy of the other). The actors and critics are neural networks with 3 layers of 128 neurons. The activation functions used are tanh and the actions are clipped between 0 and 1.

The states are preprocessed meaning that the input dimension is 2, (actual speed ω and reference speed ω*). This approach was chosen such as the resulting controller will be easy to implement in practice since the only measurement used is the actual speed. We did not implement any constraints (such as maximum current).

The optimizer used is Adam [12], with a learning rate of 0.0001 and loss function mean squared error. On the optimizer we did not implement any clipping to the network's parameters.

A typical replay buffer is used for store trajectories in the form of a tuple *(state, action, reward, next_state, done)*. We didn't use the done flag from the environment since we didn't impose any restrictions on motor currents. For training purposes, the replay buffer size was 1e6 and we randomly sampled a batch size of 256 samples.

Each episode consists of 2000 steps in the environment. On each episode we save the total reward for tracking the learning curve.

On training sequence, we apply noise as random number between 0 and 20% of maximum action value, clipping the final value on maximum 50% of maximum action value.

We trained the networks with different values for training interval and τ to observe the impact on final cumulative reward. Training interval is a parameter that defines the frequency of training on a running episode. This parameter has a big impact on convergence and final cumulative reward.

An example we ran three training sessions with 90 epochs, τ=0.1, maximum noise of 20% and training interval of 3, and we obtained the following cumulative reward, figure 4:

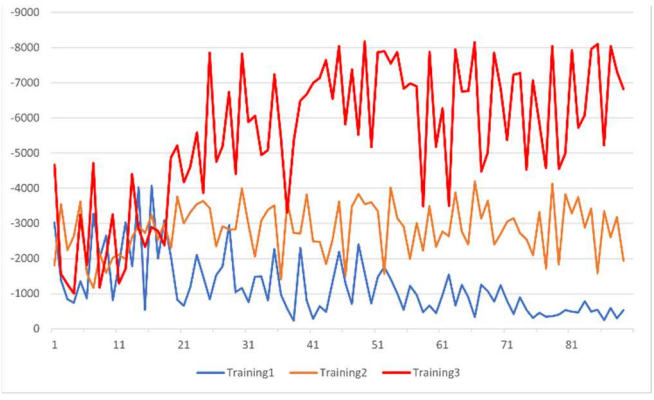

Fig.4 Cumulative reward for training interval 3

As shown the training is not consistent, as generally known for DDPG algorithms. In one case the cumulative reward diverges, on another it oscillates and the final one it converges at around -250 cumulative reward.

Another test involves determining the impact of different training interval values. Running the tests with 90 epochs, τ=0.1-0.005, maximum noise of 20% and training interval 4, results being presented in figure 5 below:

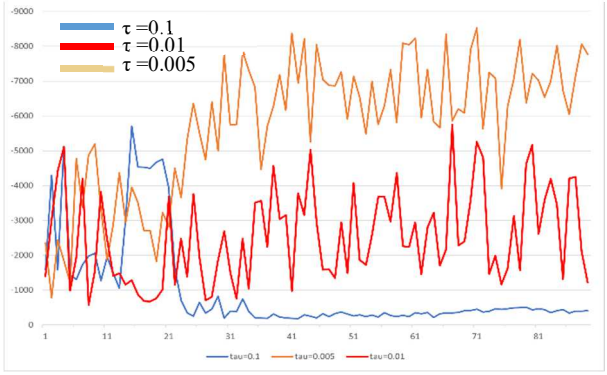

Fig.5 Cumulative reward for τ=0.1, 0.01 and 0.005

As seen, the best cumulative reward is obtained for τ=0.1 with a final value after 40 epochs around -200.

We ran other trainings to determine the influence of training interval on cumulative reward considering τ=0.1 and the obtained results are presented in figure 6:

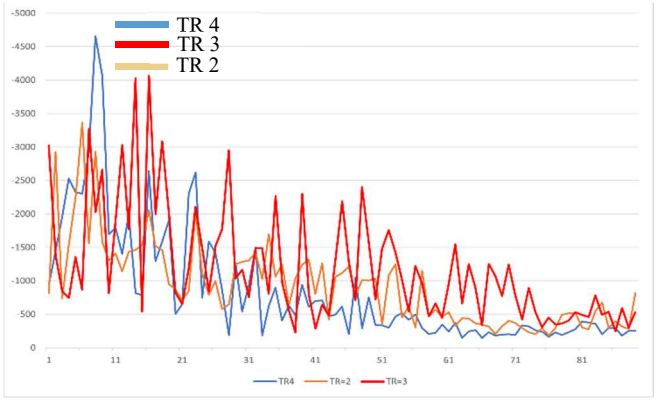

Fig.6 Cumulative reward for training interval 2,3 and 4

As seen from the above picture the training interval has an impact on the cumulative reward even if in all cases it converges. The most stable case is for training interval 4 and we will choose it as the final value for this parameter. Considering the above training tests, we have chosen for the final implementation the following values for parameters: τ=0.1, training interval=4 and maximum noise of 20% in the training sequence. We trained the networks, several times, for 110 episodes on an Nvidia RTX3080 GPU. Each episode consists of 2000 steps. We used for training a replay buffer of 1e6 samples, sampled randomly. After 5 episodes the training is activated at each 4-step interval. In the training we update the actor and the frozen networks at 100 step intervals.

In figure 7 we present the total reward after each iteration:

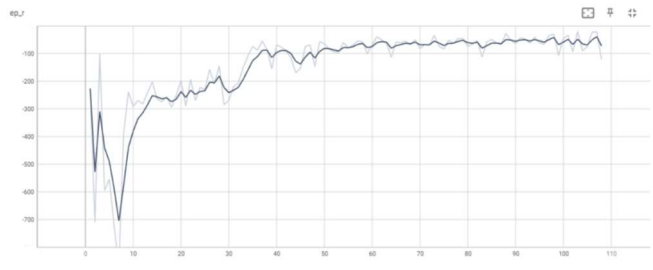

Fig.7 Episodes total reward

Considering the reward function definition, the total reward is negative. As seen from the chart after 90 episodes the total reward is stabilized around -50.

We found out, as already seen in the DDPG algorithm, that total reward and actual learning is not the same on different training sessions. We observed worse performance on different sessions using the same training parameters. This is one drawback in the algorithm and shown results are the best that we achieved. In our case the biggest impact on performance is generated by training intervals which can cause training instability.

After each episode we tests the policy for 2000 steps and the results are in figure 8 and figure 9 (any test is done without any noise applied to the control action):

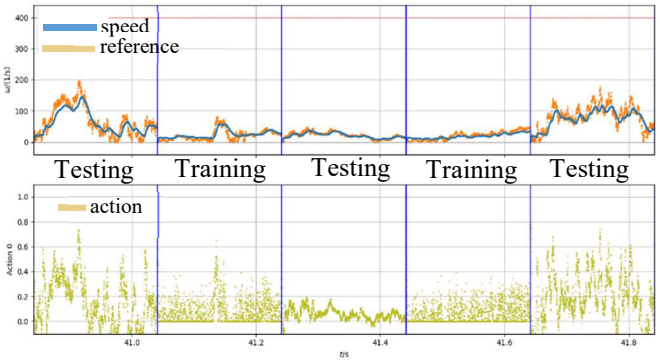

Fig8. Training and testing sequences 40sec-42sec.

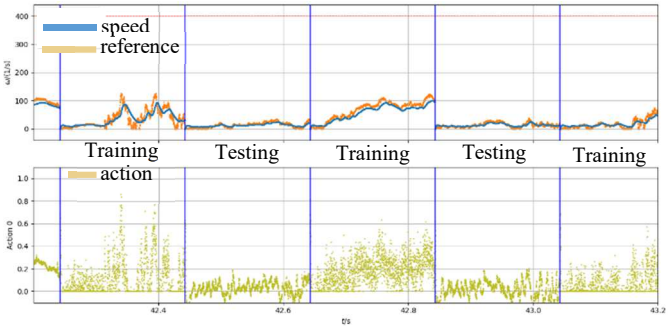

Fig9. Training and testing sequences 42-43 sec.

To verify the final controller, we tested against typical references. With model in evaluation mode first we tested against step reference and the results are below, figure 10:

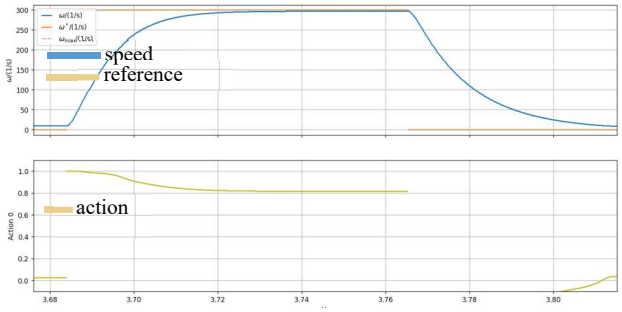

Fig.10 Testing with step reference.

The steady state error is zero achieved without overshoot and a rise time of 28 ms. Also testing with a sinusoidal reference generator with frequency range (10,20)Hz shows good reference tracking performance. The results are presented in figure 11, below:

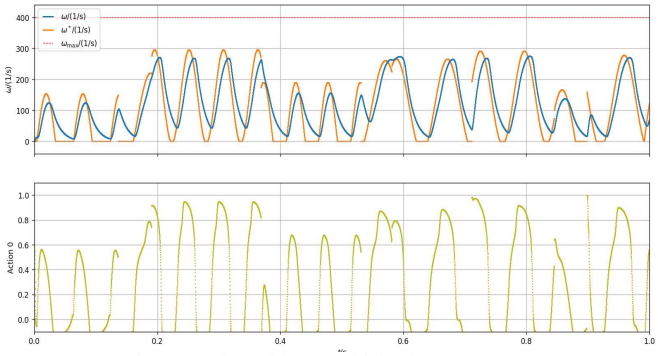

Fig.11 Testing with sinusoidal reference generator.

IV. CONCLUSIONS

In this paper we presented a novel speed controller for permanently excited dc motor based on reinforcement learning TD3 algorithm. As seen from the simulation results, after 110 training episodes the results are promising, with very good speed tracking performance. Tests with different reference generators is provided and the performance is obtained across all types of references.

The paper focused not only on obtaining an optimal controller but also identifying the impact of different tunning parameters on cumulative reward, training speed and stability for this specific application.

Implementation of this controller on real time hardware is relatively easy through pytorch framework on using the ONNX runtime. Further work consists of implementation of this controller in a physical setup and evaluating the results.

REFERENCES

[1] D. Görges - *Relations between model predictive control and reinforcement learning*, *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920-4928, Jul. 2017

[2] M. Glavic, R. Fonteneau and D. Ernst, - *Reinforcement learning for electric power system decision and control: Past considerations and perspectives*, *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6918-6927, Jul. 2017.

[3] M. Novak and T. Dragicevic - Supervised imitation learning of finite set model predictive control systems for power electronics, IEEE Trans. Ind. Electron., Jan. 2020.

[4] G. Bujgoi and D. Sendrescu - *DC Motor Control based on Integral Reinforcement Learning*, *2022 23rd International Carpathian Control Conference (ICCC)*, Sinaia, Romania, 2022, pp. 282-286, doi: 10.1109/ICCC54292.2022.9805935.

[5] Bucur C., Burlacu P. - *Artificial intelligence driven speed controller for DC motor in series, Scientific Bulletin" Mircea cel Batran" Naval Academy 24 (2), 1-6*, 2021

[6] M. Schenke, W. Kirchgässner and O. Wallscheid - *Controller design for electrical drives by deep reinforcement learning: A proof of concept*, IEEE Trans. Ind. Informat., vol. 16, no. 7, pp. 4650-4658, Jul. 2020

[7] Hado van Hasseltl, Arthur Guez and David Silver - *Deep Reinforcement Learning with Double Q-learning*, arXiv:1509.06461v3, 2015;.

[8] Teckchai T., Ismail S., Kenneth T. Herwansyah L., *Deep Reinforcement Learning with Robust Deep Deterministic Policy Gradient*, 2nd International Conference on Electrical

[9] A. Traue, G. Book, W. Kirchgässner, O. Wallscheid - Toward a Reinforcement Learning Environment Toolbox for Intelligent Electric Motor Control, *IEEE Transactions on Neural Networks and Learning Systems*, 2020

[10] A. Traue, G. Book, W. Kirchässner and O. Wallscheid, *Gym Electric Motor (GEM)*, 2020, [online] Available: https://github.com/upb-lea/gym-electric-motor.

[11] 12, Herke van Hoof, Dave Meger - Addressing Function Approximation Error in Actor-Critic Methods, arXiv:1802.09477

[12] Diederik P. Kingma, Jimmy Lei Ba - *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*, Published as a conference paper at ICLR 2015.