Biped Robot Walking based on Deep Reinforcement Learning

Tomislav Tadić[*] and Petar Ćurković[*]

[*] Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb, Croatia
tomitadic97@gmail.com; pcurkovic@fsb.hr

Abstract **- In this paper a 6 DOF biped robot model is designed, and deep reinforcement machine learning methods are implemented for the robot to learn efficient walking following a straight line. Detailed procedure of the robot design, development of a Simulink model and implementation of learning procedures is presented. Two approaches were compared for motion learning – Deep Deterministic Policy Gradient (DDPG), and Twin-Delayed Deep Deterministic Policy Gradient (TD3). The results show that both approaches are successful in generating a model with free continuous action learning and input to action mapping. Additionally, our results show that the TD3 algorithm outperforms the DDPG algorithm in the problem as formulated in this study.**

Keywords – reinforcement learning; artificial neural network; biped robot model; agent; environment; Matlab; Simulink; DDPG algorithm; TD3 algorithm

I. Introduction

The field of machine learning (ML) has made significant progress in various applications over the last few years, from robotics, image analysis and generation to computer games to name a few. An important aspect of inherently unstable mechanical systems, such as inverse pendulum, or humanoid robotics is finding efficient control laws to enable stable and efficient walking. This is not a simple errand, and many traditional [1-3] and AI-based approaches [4-8] exist trying to efficiently solve it. In this study, we focus on implementing and comparing two ML approaches to a bipedal robot model. The motivation is to get a deeper insight in both methods behavior and suitability to solve the problem of generating a stable and efficient gait for the two-legged motion following a straight line. We further plan to expand the model to include obstacles, stairs and slopes in the environment, based on results of this study.

II. Reinforcement Learning

Reinforcement learning is a goal-oriented computing approach, where a computer learns to perform a task by interacting with an unknown dynamic environment. This approach to learning allows a computer to make a series of decisions to maximize the cumulative reward for a task without human intervention and without explicit programming to accomplish that task [9].

The goal of reinforcement learning is to train an agent to perform a task within an unfamiliar environment. The agent receives observations and rewards from the environment and sends actions to the environment. The reward is a measure of how successful the action is in meeting the goal of the task. The agent contains two components: a policy (a set of behavioral rules) and a learning algorithm. Policy is a mapping that selects actions based on observations from the environment. Typically, a policy is a function approximator with adjustable parameters, such as a deep neural network. The learning algorithm continuously updates policy parameters based on actions, observations and rewards. The goal of the learning algorithm is to find the optimal policy that maximizes the cumulative reward received while performing the task.

III. Implementation of Reinforcement Learning on a Biped Robot Model

A. *Creation of a robot model*

The parts of the robot model, as shown in Fig. 1, were designed in the SolidWorks software, and in the same software they were joined into an assembly, in which the mutual relations of the parts and the revolute joints were defined. A total of 6 joints are defined (hip, knee and ankle for both legs of the robot model).

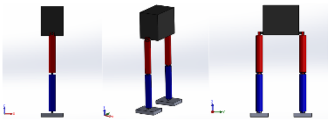
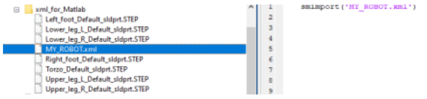


Figure 1. Biped robot model assembly (in SolidWorks)

B. *Transferring the model to the Simulink environment*

The first step in transferring a CAD model to Matlab is to create a translatory .xml object which can be loaded to many physical simulators, Matlab included. To open the .xml file in Simulink it is necessary to write "smimport" in Matlab and put the name of the .xml file in brackets as an argument, which is shown in Fig. 2.


xml_for_Matlab
Left_foot_Default_sldprt.STEP
Lower_leg_L_Default_sldprt.STEP
Lower_leg_R_Default_sldprt.STEP
MY_ROBOT.xml
Right_foot_Default_sldprt.STEP
Torzo_Default_sldprt.STEP
Upper_leg_L_Default_sldprt.STEP
Upper_leg_R_Default_sldprt.STEP

```
smimport('MY_ROBOT.xml')
```

Figure 2. Opening the .xml file in Matlab

This opens a block diagram of the .xml file in Simulink, as shown in Fig. 3.

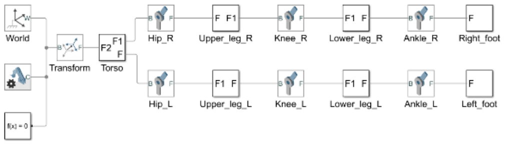


Figure 3. Block diagram of the basic robot model in Simulink

Starting the Simulink model from the figure above displays the basic model of the robot (modeled in SolidWorks) within the Simulink environment, which is shown in Fig. 4. However, that model needs to be adapted to implement reinforcement learning.

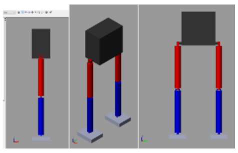


Figure 4. The basic model of the robot in the Simulink environment

C. Adapting the model to implement reinforcement learning

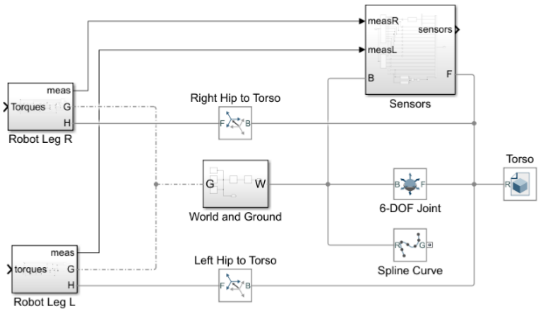


Figure 5. Block diagram of the adapted robot model

Fig. 5 shows the block diagram of the adapted robot model. A subsystem named "World and Ground" was added to the basic model. This subsystem is shown in detail in Fig. 6. It is important to note that the "World Frame" block (the block with the letter "W") was added from the Simulink library, so it is not a copied block that can be seen in the fig. 3 which shows the basic model of the robot from the .xml file (that block has been deleted).

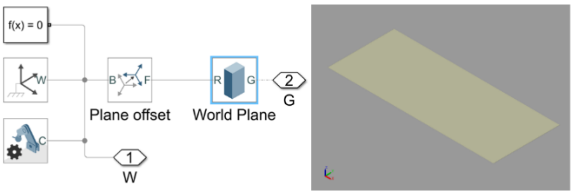


Figure 6. Subsystem "World and Ground"

The "6-DOF Joint" block, which represents a ball joint with 6 degrees of freedom of movement, also from the Simulink library, was added. This joint is connected to the torso of the robot model, because the torso must be allowed to move freely during walking, so the torso must not be fixed. For a nicer display during gait simulation, a curve in the positive direction of the X axis has been added to the base. This is the block that is located below the "6-DOF Joint" block in the model (see fig. 5). Fig. 7 shows the "Robot leg R" subsystem, which represents the modified model of the robot's right leg. For the joints, blocks provided by the Matlab team that created specific blocks for applying reinforcement learning to bipedal robot models were used. These joint blocks, shown on Fig. 8, are controllable via torques (port "t"), which the model will receive from the RL agent in the reinforcement learning process.

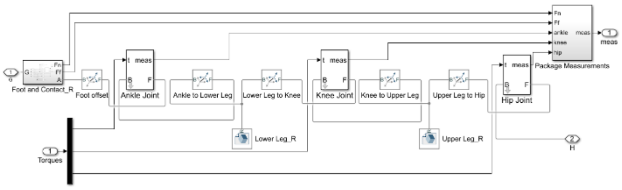


Figure 7. Subsystem "Robot leg R"

In addition, they have an output port ("meas") through which they send information about the current angle, angular velocity and applied torque on each individual joint to the "Measurement Package" block.

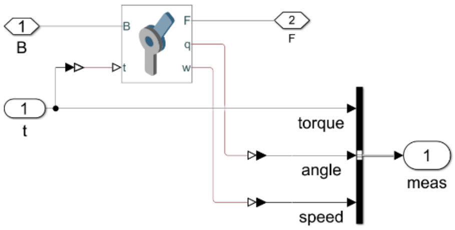


Figure 8. Layout of a custom joint block for the robot model

Fig. 9 shows the "Foot and Contact_R" subsystem. This subsystem serves for the realization of walking, that is, the establishment of contact between the robot and the ground. It shows the implementation of the "Spatial Contact Force" block from the Simulink library, which can be used to model the contact between two surfaces and define the parameters of that contact. In the example of a walking robot, the modeled contact is the contact between the robot's foot and the surface on which the robot walks. In this subsystem, it is visible that four "Spatial Contact Force" blocks are connected to four ball blocks. Four balls are placed in the four corners of the robot's foot model, thus realizing contact at four points of contact between the robot's foot and the surface on which the robot walks.

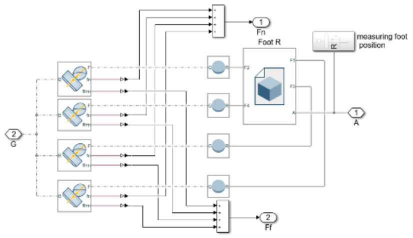


Figure 9. Subsystem "Foot and Contact_R"

Fig. 10 provides a visualization of the defined coordinate systems for the balls ("F1", "F2", "F3" and "F4"), as well as the coordinate system "A", which defines the geometric connection between the foot and the ankle joint. The appearance of the ”Spherical Solid” block is also shown.

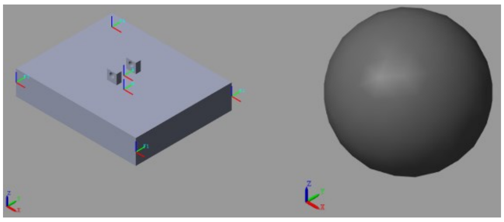


Figure 10. Defined coordinate systems for balls and block ”Spherical Solid”

Fig. 11 shows the " Package Measurements " block to which the contact data (total normal force and friction force between the foot and the surface) and data from the joints (ankle, knee and hip) are sent. The data from the joints come in the packages of three data from each joint (angle, angular velocity and torque), so it is necessary to unpack them into individual signals. "Extract Measurements" blocks are used for unpacking.

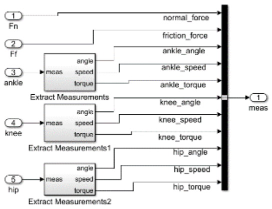


Figure 11. Block " Package Measurements "

All parts of the "Robot leg R" subsystem (see Fig. 7) of the adapted robot model are hereby described. The "Robot Leg L" subsystem is completely analogous, so there is no need to show and describe it. Fig. 12 shows the "Observations" block in which all measurements of the model adjusted for reinforcement learning are combined. These are measurements from the left and right leg of the robot (total of 22 data, 11 per leg), and data on the state of the torso of the robot model connected to the ball joint (6 degrees of freedom of movement) which are extracted using the Simulink block " Transform sensor". This is 12 additional data (position of the torso along the x, y and z-axis, velocity of the torso in the directions of the x, y and z-axis, acceleration of the torso in the direction of the x, y and z-axis, angular velocity of the torso around the x, y and z-axis) . In addition, 3 more data are considered, which are obtained using the Matlab function "calcAltitude" and 1 data on the total applied power to actuate the joints of both legs of the robot model (see Fig. 13). So the observations consist of a total of 38 (22+12+3+1=38) data on the robot model.

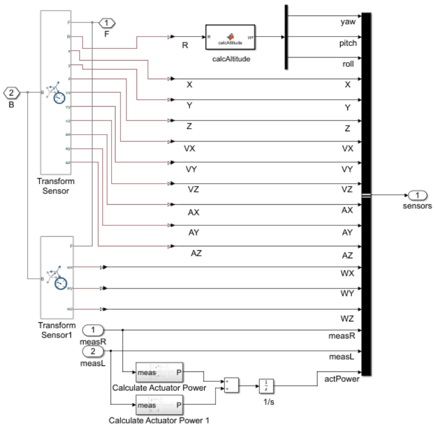


Figure 12. “Observations“ block

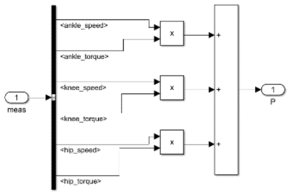


Figure 13. "Calculate Actuator Power" subsystem

Fig. 14 shows the appearance of the robot model in the initial position on the ground and at the beginning of the curve in the x-axis direction. The positions of the balls on the robot's feet, which are connected to the "Spatial Contact Force" blocks and through which the detection of the foot's contact with the ground was achieved, enabling the simulation of walking are visible.

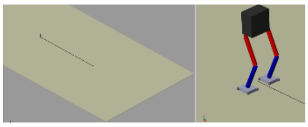


Figure 14. Initial position of the robot model for learning

Fig. 15 shows the behavior of the robot model when starting the simulation in the Simulink environment. A simulated gravitational force acts on the robot, and since the robot's joints are not controlled by anything, the robot falls after a while. Since the contact between the foot and the surface is defined, the robot remains in a "hanging position". Contact is not defined on other parts of the robot model, so the torso and legs fall through the base.

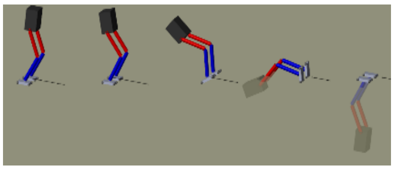


Figure 15. Behavior of the robot model without control when starting the simulation

Now it is necessary to find a control law that will enable the robot to walk on the surface without falling and with as little deviation as possible from the black line shown. As previously mentioned, it is a complex problem to solve using traditional control methods, which motivates the implementation of ML methods presented in this paper.

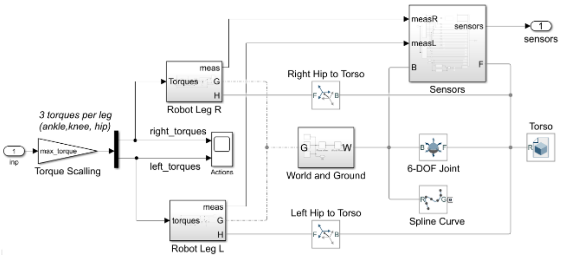


Figure 16. Model of a walking robot ready for addition of a control system

Fig. 16 shows model of a walking robot ready for addition of a control system. Input and output ports for communication with the control system were added to the previous model (see Fig. 5), and the input port was scaled (since the actions sent by the RL agent in reinforcement learning will be limited to values between -1 and 1). In addition to scaling, the input will be a vector of 6 elements (torques to drive the 6 joints of the robot model), so it is necessary to separate them into two vectors of three elements each for the left and right legs of the robot.

Fig. 17 shows the appearance of the control system inherent in reinforcement learning. Such systems generally consist of an RL (Reinforcement Learning) Agent consisting of an RL algorithm and artificial neural networks, the number of which depends on the type of applied RL algorithm. DDPG (Deep Deterministic Policy Gradient) Agent, which consists of two artificial neural networks ("Critic" and "Actor") and TD3 (Twin-Delayed Deep Deterministic Policy Gradient) Agent, which consists of three artificial neural networks ("Actor" and two "Critic" networks) were applied in this work. Network structures are selected by the user. In addition to the Agent, such systems must also have an environment in which the Agent operates and tries to learn to perform a certain task. In this example, the environment is represented by a walking robot model. The means of communication between the Agent and the environment must also be defined, observations (from the environment) and actions (from the Agent) are used for this. The figure below shows the layout of the implemented reinforcement learning system, which is used to control the movement of the robot model (saved under the name "MY_ROBOT.slx").

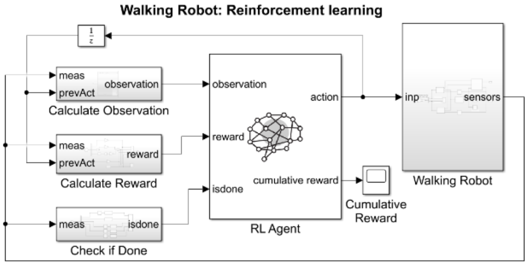


Figure 17. A complete model for implementing reinforcement learning

From a total of 38 data from the robot model, 25 data are selected for the final observations. Some of the selected observations are further processed. The initial position of the torso along the z-axis is subtracted from the current position of the model's torso along the z-axis, because the deviation of the torso's position from the initial position is important for gait learning (the goal is the smallest possible deviation along the z-axis), and this difference is scaled to have a greater influence on the behavior of the robot. In addition, the actions generated by the RL Agent in the previous learning step are added to the observations, that's 6 more pieces of data. So, in total, 31 (25+6) observations are continuously sent to the RL Agent during the reinforcement learning process.

Fig. 18 shows the "Calculate Reward" subsystem, where the reward function is formulated, that is, the reward signal is calculated. For this example of a walking robot, the reward function in mathematical form reads:

$$r_t = v_x + 0.0625 - 25 * \hat{z}^2 - 0.05 * \sum_i (u_{t-1}^i)^2 - 3 * y^2 \quad (1)$$

Whereby $\hat{z} = z - z_0$; z_0–initial position of the torso along the z-axis

The desired behavior of the robot when walking can be deduced from the reward function shown in (1). The first term of the equation rewards the speed of movement along the x-axis, the second term rewards the duration of the walk (for each time step of the simulation "Ts", the Agent receives a reward of 0.0625, until the simulation is interrupted, e.g. due to the fall of the robot), the third term penalizes the deviation from the initial position of the torso along the z-axis, the fourth term penalizes the total power required to move the robot, excessive torques in the joints are not desirable (this prevents the robot's legs from being thrown and uneven movement of the robot's left and right legs) and the fifth term penalizes the lateral deviation of the robot from x-axis (along the y-axis in both directions). The image below shows the implementation of the above mathematical reward function modeled in Simulink.

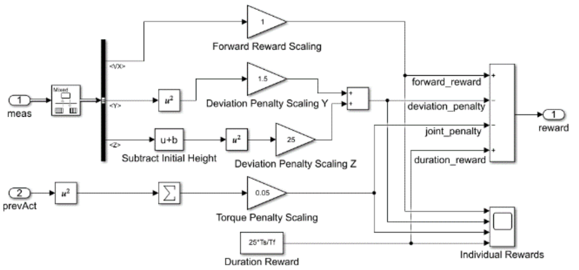


Figure 18. "Calculate Reward" subsystem

Fig. 19 shows the subsystem "Check if Done". Here, the conditions for breaking an episode when training an RL Agent are defined. The maximum allowed lateral deviation from the x-axis in the y-axis direction is defined. The minimum amount of position of the robot's torso along the z-axis for detecting the robot's fall is also set. In addition, the maximum allowed rotation angles of the robot's torso around the x, y and z-axis are set, because they indicate a loss of balance and a fall of the robot, that is, an unnatural movement of the robot that we want to avoid.

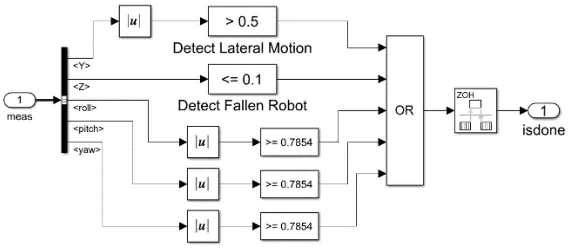
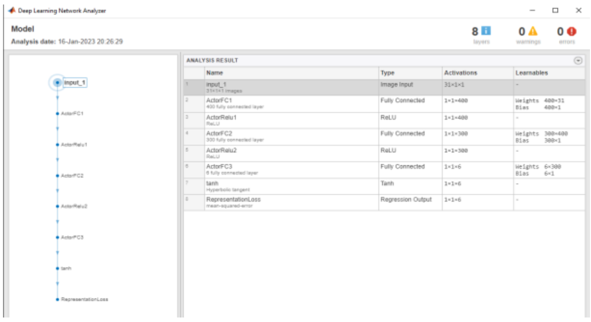


Figure 19. “Check if Done“ subsystem

This describes the created Simulink environment for training RL agents. Now follows a brief description of the created artificial neural networks and options for training RL agents, as well as displays of training results and simulation displays.

Fig. 20 shows the appearance and properties of the "Actor" neural network (created in the Matlab script "my_neural_networks.m") for reinforcement learning purposes on the example of a bipedal walking robot.



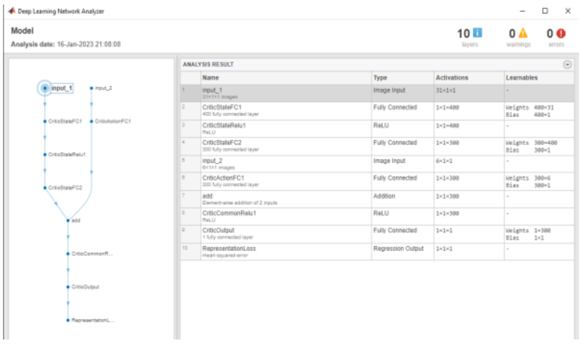
Model
Analysis date: 16-Jan-2023 20:26:29

ANALYSIS RESULT

	Name	Type	Activations	Learnables
1	input_1 31×1×1 images	Image Input	31×1×1	-
2	ActorFC1 400 fully connected layer	Fully Connected	1×1×400	Weights 400×31 Bias 400×1
3	ActorRelu1 ReLU	ReLU	1×1×400	-
4	ActorFC2 300 fully connected layer	Fully Connected	1×1×300	Weights 300×400 Bias 300×1
5	ActorRelu2 ReLU	ReLU	1×1×300	-
6	ActorFC3 6 fully connected layer	Fully Connected	1×1×6	Weights 6×300 Bias 6×1
7	tanh Hyperbolic tangent	Tanh	1×1×6	-
8	RepresentationLoss mean-squared-error	Regression Output	1×1×6	-

Figure 20. Appearance and properties of the created "Actor" neural network

Fig. 21 shows the appearance and properties of the "Critic" neural network (created in the Matlab script "my_neural_networks.m") for reinforcement learning purposes on the example of a bipedal walking robot.



Model

Analysis date: 16-Jan-2023 21:08:08

ANALYSIS RESULT

	Name	Type	Activations	Learnables
1	input_1 31×1×1 images	Image Input	31×1×1	-
2	CriticStateFC1 400 fully connected layer	Fully Connected	1×1×400	Weights 400×31 Bias 400×1
3	CriticStateRelu1 ReLU	ReLU	1×1×400	-
4	CriticStateFC2 300 fully connected layer	Fully Connected	1×1×300	Weights 300×400 Bias 300×1
5	input_2 6×1×1 images	Image Input	6×1×1	-
6	CriticActionFC1 300 fully connected layer	Fully Connected	1×1×300	Weights 300×6 Bias 300×1
7	add Element-wise addition of 2 inputs	Addition	1×1×300	-
8	CriticCommonRelu1 ReLU	ReLU	1×1×300	-
9	CriticOutput 1 fully connected layer	Fully Connected	1×1×1	Weights 1×300 Bias 1×1
10	RepresentationLoss mean-squared-error	Regression Output	1×1×1	-

Figure 21. Appearance and properties of the created "Critic" neural network

D. *Matlab script for training DDPG (Deep Deterministic Policy Gradient) Agent*

A Matlab script containing the code for training the RL DDPG Agent was created. Information about observations and actions are created. As mentioned earlier, there is a total of 31 observations for Agent training and 6 actions which are limited to values between -1 and 1 (for better/faster/easier training of artificial neural networks). An environment is created, which is connected to the Simulink model of the robot (see Fig. 17). For training purposes, a reset function is defined, i.e. returning the robot to its initial position after each training episode. Then the Matlab script "my_neural_networks" is called, which creates artificial neural networks for the RL Agent in the desired structure. After that, the Matlab script "createDDPGOptions" is called, which creates the desired RL Agent options.

An Agent is then created and its training begins according to the defined training options from the "createDDPGOptions" script. During training, those Agents that meet the saving condition (which is defined in the "createDDPGOptions" script) are saved in the "savedAgents" folder. After training, saved Agents can be simulated within a Simulink robot model (see Fig. 17) or with Matlab's Reinforcement Learning Designer app.

E. *Training the RL Agent that controls the gait of the robot model*

By starting the script for loading the parameters of the robot model, and then the script for creating and training the DDPG Agent, the window shown in Fig. 22 opens and the training of the DDPG Agent, which operates within the environment of the Simulink model of the walking robot named "MY_ROBOT.slx", begins. The goal of the training is to maximize the total reward, and in the Matlab script "createDDPGOptions" the conditions for the maximum number of episodes are defined, as well as the conditions for saving good Agents (all episodes that reach a reward of 120 or more), as well as the condition for terminating Agent training (if the average reward reaches a value of 110).

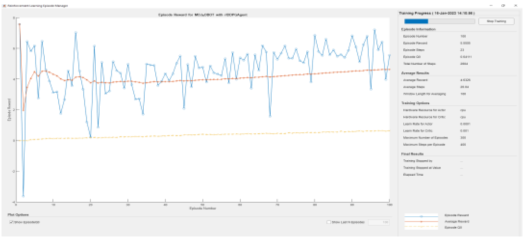


Figure 22. Training of DDPG Agent

Fig. 23 shows the movement of the robot model at the beginning of training. It is obvious that the robot does not walk correctly, but this is normal considering that the RL DDPG Agent does not yet have experience and does not have developed rules of behavior (that is, weight factors in neural networks) to achieve large amounts of the reward function.

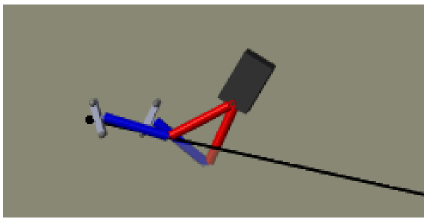
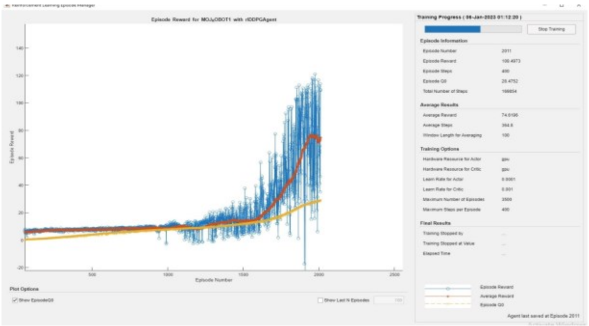


Figure 23. Movement of the robot model at the beginning of training

Fig. 24 contains a graph showing the layout of the value of the reward function for individual episodes during Agent training. The blue lines and circles indicate the rewards of individual episodes, the orange curve shows the average reward (there is a positive trend during the learning of the Agent), and the yellow curve represents the "Critic" neural network's assessment of the quality of behavior, i.e. the predicted reward at the start of each individual learning episode.



Training Progress (06-Jan-2023 01:12:20)

Episode Information

Episode Number	2011
Episode Reward	109.4973
Episode Steps	400
Episode Q0	28.4752
Total Number of Steps	169854

Average Results

Average Reward	74.6196
Average Steps	364.8
Window Length for Averaging	100

Training Options

Hardware Resource for Actor	gpu
Hardware Resource for Critic	gpu
Learn Rate for Actor	0.0001
Learn Rate for Critic	0.001
Maximum Number of Episodes	3500
Maximum Steps per Episode	400

Final Results

Training Stopped by	—
Training Stopped at Value	—
Elapsed Time	—

Figure 24. Graph with the values of the reward function for individual episodes during DDPG Agent training

Fig. 25 shows the gait simulation of a bipedal robot model controlled by a trained Agent. The robot crosses the marked path and continues to walk without falling (if the simulation time was allowed to be longer, the robot would have traveled a greater distance). This indicates a well-found control law. It is also noticeable that the robot does not deviate much from the x-axis during walking and does not swing too much left-right during walking, nor does its torso lean back, that is, it does not deviate much from the initial position of the torso along the z-axis. This is consistent with the reward function model.

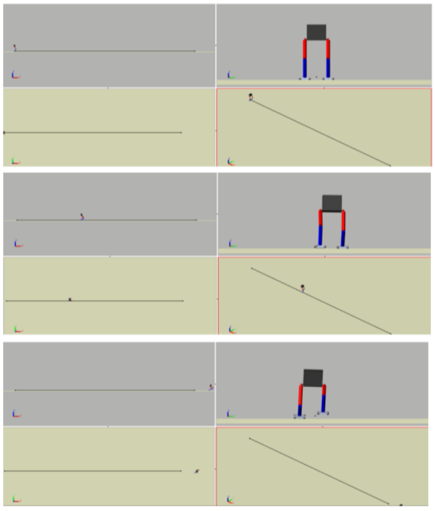
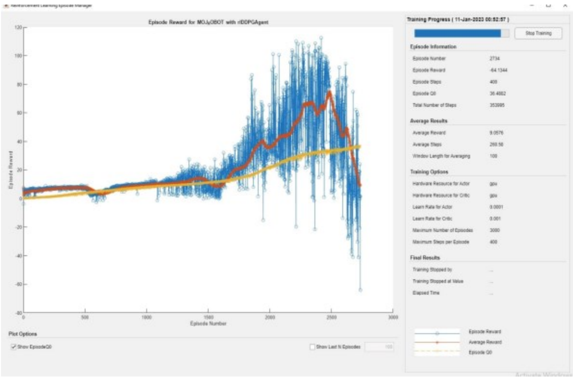


Figure 25. Flow of gait simulation of a bipedal robot model

Fig. 26 shows the DDPG Agent training graph for the case when learning was not very successful. Given that DDPG as an RL algorithm is quite stochastic, quality learning of the Agent cannot be guaranteed every time. This graph is shown to illustrate the described nature of the DDPG algorithm.



Training Progress (11-Jan-2023 00:52:57)

Episode Information

Episode Number	2734
Episode Reward	-64.1344
Episode Steps	400
Episode Q0	36.4882
Total Number of Steps	353995

Average Results

Average Reward	9.0576
Average Steps	259.58
Window Length for Averaging	100

Training Options

Hardware Resource for Actor	gpu
Hardware Resource for Critic	gpu
Learn Rate for Actor	0.0001
Learn Rate for Critic	0.001
Maximum Number of Episodes	3000
Maximum Steps per Episode	400

Final Results

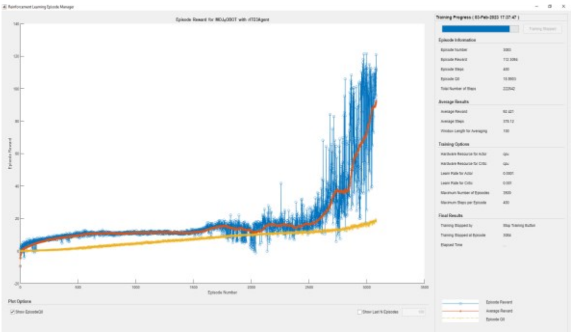
Training Stopped by	--
Training Stopped at Value	--
Elapsed Time	--

Plot Options

☑ Show EpisodeQ0 □ Show Last N Episodes 100

Figure 26. Training graph of the DDPG RL Agent for the case of failed learning

Training of the robot model was also done using TD3 algorithm after making the necessary adjustments in Matlab scripts and its graph is shown in Fig. 27.



Training Progress (03-Feb-2023 17:37:47)

Episode Information

Episode Number	3083
Episode Reward	112.3086
Episode Steps	400
Episode Q0	15.8603
Total Number of Steps	222642

Average Results

Average Reward	92.427
Average Steps	378.12
Window Length for Averaging	100

Training Options

Hardware Resource for Actor	gpu
Hardware Resource for Critic	gpu
Learn Rate for Actor	0.0001
Learn Rate for Critic	0.001
Maximum Number of Episodes	3500
Maximum Steps per Episode	400

Final Results

Training Stopped by	Stop Training Button
Training Stopped at Episode	3084
Elapsed Time	--

Plot Options

☑ Show EpisodeQ0 □ Show Last N Episodes 100

Figure 27. Training graph of a TD3 RL Agent

IV. Conclusion

If we compare the reinforcement learning graphs for the DDPG and TD3 RL algorithms, we can conclude that the DDPG algorithm is more aggressive, which is reflected in the fact that it has larger oscillations in episodic rewards in earlier learning episodes. With the DDPG algorithm, reward amounts of around 100 were achieved already around the 1800th learning episode, which is much faster compared to the TD3 algorithm, which only achieves reward amounts of around 20 around the 1800th episode. Reward amounts of around 120 were achieved with the DDPG algorithm around 1900 episodes, and with the TD3 algorithm only around the 2900th learning episode. On the other hand, the obvious advantage of the TD3 algorithm is that the average reward per episode has a continuous and stable growth, which promises better control laws proportional to the learning time. The same cannot be said for the DDPG algorithm, because even though it reaches good control laws faster, it does not guarantee a continuous growth of the average reward, that is, with a longer learning time, there may be a decline in the quality of the control laws. This flaw of the DDPG algorithm is very well shown by the graph in Fig. 26. Therefore, the conclusion is that the TD3 algorithm has an advantage in the application of reinforcement learning for finding control laws for the gait of a bipedal humanoid robot. However, it would not be impossible to find quality solutions using the DDPG algorithm as well, since it allowes greater exploration.

References

[1] Reher, J. and Ames, A.D., 2021. Dynamic walking: Toward agile and efficient bipedal robots. *Annual Review of Control, Robotics, and Autonomous Systems*, *4*, pp.535-572.

[2] Gong, Y., Hartley, R., Da, X., Hereid, A., Harib, O., Huang, J.K. and Grizzle, J., 2019, July. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. In *2019 American Control Conference (ACC)* (pp. 4559-4566). IEEE.

[3] Kim, I.S., Han, Y.J. and Hong, Y.D., 2019. Stability control for dynamic walking of bipedal robot with real-time capture point trajectory optimization. *Journal of Intelligent & Robotic Systems*, *96*, pp.345-361.

[4] Z. Li et al., "Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots," 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 2021, pp. 2811-2817, doi: 10.1109/ICRA48506.2021.9560769.

[5] Rudin, N., Kolvenbach, H., Tsounis, V. and Hutter, M., 2021. Cat-like jumping and landing of legged robots in low gravity using deep reinforcement learning. *IEEE Transactions on Robotics*, *38*(1), pp.317-328.

[6] Siekmann, J., Green, K., Warila, J., Fern, A. and Hurst, J., 2021. Blind bipedal stair traversal via sim-to-real reinforcement learning. *arXiv preprint arXiv:2105.08328*.

[7] Li, T., Geyer, H., Atkeson, C.G. and Rai, A., 2019, May. Using deep reinforcement learning to learn high-level policies on the atrias biped. In *2019 International Conference on Robotics and Automation (ICRA)* (pp. 263-269). IEEE.

[8] T. Tiong, I. Saad, K. T. K. Teo and H. b. Lago, "Deep Reinforcement Learning with Robust Deep Deterministic Policy Gradient," *2020 2nd International Conference on Electrical, Control and Instrumentation Engineering (ICECIE)*, Kuala Lumpur, Malaysia, 2020, pp. 1-5, doi: 10.1109/ICECIE50279.2020.9309539.

[9] Russell, S.J., 2010. *Artificial intelligence a modern approach*. Pearson Education, Inc.