Ain Shams Engineering Journal xxx (xxxx) xxx

Contents lists available at ScienceDirect

Ain Shams Engineering Journal

journal homepage: www.sciencedirect.com

Development of Push-Recovery control system for humanoid robots using deep reinforcement learning

Emrah Aslan [a,*], Muhammet Ali Arserim [b], Ayşegül Uçar [c]

[a] Dicle University, Silvan Vocational School, Diyarbakır, Türkiye
[b] Dicle University, Engineering Faculty, Diyarbakır, Türkiye
[c] Fırat University, Engineering Faculty, Elazığ, Türkiye

ARTICLE INFO

Article history:
Received 28 June 2022
Revised 30 December 2022
Accepted 10 January 2023
Available online xxxx

Keywords:
Deep reinforcement learning
deep q network(DQN)
double deep q network(DDQN)
Humanoid robot
Robotis op2
Push-recovery

ABSTRACT

This paper focuses on the push-recovery problem of bipedal humanoid robots affected by external forces and pushes. Since they are structurally unstable, balance is the most important problem in humanoid robots. Our purpose is to design and implement a completely independent push-recovery control system that can imitate the actions of a human. For humanoid robots to be able to stay in balance while standing or walking, and to prevent balance disorders that may be caused by external forces, an active balance control has been presented. Push-recovery controllers consist of three strategies: ankle strategy, hip strategy, and step strategy. These strategies are biomechanical responses that people show in cases of balance disorder. In our application, both simulation and real-world tests have been performed. The simulation tests of the study were carried out with 3D models in the Webots environment. Real-world tests were performed on the Robotis-OP2 humanoid robot. The gyroscope, accelerometer and motor data from the sensors in our robot were recorded and external pushing force was applied to the robot. The balance of the robot was achieved by using the recorded data and the ankle strategy. To make the robot completely autonomous, Deep Q Network (DQN) and Double Deep Q Network (DDQN) methods from Deep Reinforcement Learning (DPL) algorithms have been applied. The results obtained with the DDQN algorithm yielded 21.03% more successful results compared to the DQN algorithm. The results obtained in the real environment tests showed parallelism to the simulation results.

1. Introduction

Since ancient times, human and animal movements have been among the important topics for researchers. For this reason, many researchers have worked on machines that imitate human and animal movements. Technological developments and efforts to facilitate human life have accelerated the studies on robots. These robots are being developed for use in jobs that are dangerous to humans or where human power is insufficient. However, robots can be insufficient to imitate some human movements. In recent years, in robotic studies, more successful humanoid robots have been developed in imitating human movements by using advanced sensors and artificial intelligence technologies.

Humanoid robots come to the forefront compared to other robots in that they resemble and can act like humans in the real environment. The most basic advantage of humanoid robots is to have higher mobility in real environments compared to wheeled and tracked robots. The ability to make movements such as passing over obstacles and climbing stairs makes them more attractive compared to wheeled and tracked robots. There are situations when humanoid robots have some disadvantages compared to traditional wheeled and tracked robots. Keeping humanoid robots in balance is a challenging problem. Since these robots do not have a flexible body structure, they have problems maintaining their balance. They are unstable to external forces, especially when walking on unexpected pushes and rough terrain. In contrast, traditional wheeled robots are more stable than humanoid robots[3].

Humanoid robots need to make decisions actively and quickly to walk successfully in real environments. These robots fail in their

* Corresponding author.
E-mail address: emrah.aslan@dicle.edu.tr (E. Aslan).

Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.asej.2023.102167
2090-4479/

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

walk for two main reasons. The first is obstacles or slopes located on the ground, and the second is the pushing forces from outside. To eliminate these problems, the balance position of the robot should be calculated very quickly with the data coming from the sensors and the results should be transmitted to the actuators to ensure the balance of the robot [8,9].

The main goal of humanoid robot studies is to develop robots that can work in real environments. Therefore, studies are carried out on recovery from external forces to develop humanoid movements in robots. In addition to the own balance problems of these robots, the impacts they receive and the irregularities caused by the ground are also described as external push. Humanoid robots are quickly affected by external pushes, they can lose their balance and fall. Therefore, push-recovery control is very important in humanoid robots. In the literature, this problem is called the bipedal locomotion problem. Various methods have been proposed to solve the walking problem in humanoid robots. These methods consist of traditional control systems, fuzzy control systems, and artificial neural networks [6,7].

In humanoid robots, balanced walking is provided by lower and upper body movements. In robots' lower body movements, the problems of a balanced walking and running are examined. Upper body movements, on the other hand, control the arm and body movements of humanoid robots to make them more balanced and more sensitive to the environment. With these methods, humanoid robots can be recovered from falling. The push-recovery control strategies included in the lower body movements consist of ankle strategy, hip strategy and step strategy. These strategies are mathematical expressions of the reactions of people in real life to be able to stand in balance disorders. In the ankle strategy, the torque value of the ankle joint is controlled to provide balance. In the hip strategy, the body is kept in balance using the angular acceleration of the body. In the step strategy, the body is balanced by changing the position of the support point in the direction of deterioration [11,18,31].

Although they are used theoretically for a proper walk, these strategies are not physically easy to implement on humanoid robots. There are few studies in the literature on how robots move using these three strategies. A machine learning approach should be used to minimize the cost of push-recovery control in humanoid robots [12,14,15].

Part I of this paper provides general information about the research problem. The rest of the paper proceeds as follows: Section II includes related works in the literature. Part III contains the System requirements. In addition, Robotis-OP2 and its hardware structure have been introduced and information about the simulation environment has been provided. Deep Q Network (DQN) and Double Deep Q Network (DDQN) from Deep Reinforcement Learning algorithm and push-recovery controllers are explained. Part IV contains the studies carried out and the results and discussion. Finally, in Part V, the conclusions and future works are discussed.

2. Literature review

In recent years, several studies have been carried out on balance and push-recovery problems of humanoid robots. Many different control methods have been used in these studies. Some of them are as follows.

Huang et al. [29] have studied how to reduce the imbalance that occurs in humanoid robots. People respond to the push from the outside with an opposite force and maintain balance. Using this strategy, they proposed a resistive control system for humanoid robots that reduces the instability caused by the movement away from the equilibrium point as a result of an external push. A virtual mass model is proposed for this system. With the proposed model, a solution is presented against balance disturbances caused by unexpected external forces and inclined ground.

Li et al. [28] used a Fuzzy DDQN (FDDQN) algorithm to construct a linear inverted pendulum model of a humanoid robot. They have designed a real-time walking model with the controller they have created. FDDQN solves the balance problem by correcting the walking pattern of the humanoid robot. In the experiments conducted, it was found that the robot received timely feedback during walking with FDDQN and performed its walking independently.

Yang et al. [25] focused on the push-recovery problem based on the inverted pendulum model of variable height in humanoid robots in their study. The zero-step point of the variable inverted pendulum was determined by a method based on the Hamilton-Jacobi accessibility analysis. They have designed a method that allows the fulcrum step position to be determined for situations where the robot cannot catch the fulcrum point. To use the applied strategy in location-controlled humanoid robots, they used an associated differential inverse kinematics-based controller. Thanks to the developed controller, it has been observed that better results are obtained for balance compared to the classical inverted pendulum model.

Schuller et al. [26], in their study, presented a push-recovery algorithm for humanoid robots by taking advantage of the rotational dynamics of the system. In their proposed method, the humanoid robot generates centripetal momentum based on the magnitude and direction of the push to eliminate and stabilize the incoming external push. By increasing and decreasing the centripetal moment, they ensured that the robot reached the equilibrium position.

Messesan et al. [27] created a controller against balance disturbances by using the step strategy and the ankle strategy. The algorithm is actively working at both single-support and double-support stages to ensure instant balance. They have developed a solution to the robot's balance problem with the controller they have designed.

Chiang and Wang [24] have designed a posture control system for the balanced walking of humanoid robots on uneven and uphill floors. They combined the data obtained from the accelerometer and gyroscope using a complementary filter. Using these filtered data, they found a solution to the robot's balance problem through a fuzzy controller.

Melo et al. [22] have developed a control system that allows humanoid robots to balance against push with control theory based on the Zero Moment Point concept for existing walking motors. They have used in their studies the Proximal Policy Optimization (PPO) algorithm, one of the deep neural network algorithms.

Seo et al. [16] have performed their analysis using the Linear Inverted Pendulum Model in their study. They have applied high-level push-recovery strategies with the DQN algorithm to their training.

Hosseinmemar et al. [20] have used the closed-loop feedback control method with the data obtained from the accelerometer and gyroscope to ensure that the humanoid motors using weak servo motors remain in balance against the pushes. To get rid of the pushes, they conducted three different experiments. These are gyroscope only, accelerometer only, and a combination of both sensors. According to the results, the balancing method created with the combination of two sensors showed a better performance.

Maulana et al. [23] designed a system capable of reducing noise on the data obtained from the tilt sensor information used in the stabilization system of humanoid robots. The data obtained from the gyroscope and accelerometer were passed through the complementary filter and the tilt angle was determined. The robot has

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

performed the balancing act depending on the output of the complementary filter. When the detected tilt angle is in a position to cause it to drop, it notifies the main processor and the servo motors are adjusted to switch to the equilibrium position.

Yang et al. [19] have presented a solution to the balance problems of humanoid robots through deep reinforcement learning using balance disorders control policies such as pushing.

Shafiee-Ashtiani et al. [30] used the combination of the ankle (Center of Pressure) and hip strategy (Center of Moment Axis). Center of Pressure and Center of Moment Axis strategies are used for torque-controlled humanoid robots. But traditional humanoid robots are location-controlled. Shafiee-Ashtiani et al. [30], in this study, designed and balanced a feedback control system using push-recovery strategies without the need for torque sensors on a location-controlled humanoid robot.

Ghassemi et al. [21] have studied the push-recovery problem of humanoid robots with unstable structures. Torque control approaches have been applied to a robot with position control. A Proportional Derivative (PD) controller has been used to control the robot.

Yi et al. [17] have used a robust feedback control method for bipedal humanoid robots to stay balanced in their walking. The stability of the robot has been optimized by using deep reinforcement learning algorithms based on data received from various sensors and motors.

Stephens [1] studied push-recovery strategies in humanoid robots. The study has presented three different push recovery strategies to recover from pushes. It has been shown that humanoid robots will stay in balance by using their ankles, hip joints and support step strategies.

Our study provides a new solution to the push-recovery problems of humanoid robots. We used ankle strategy, which is one of the push-recovery strategies applied to the torque-controlled humanoid robot. The data received from the sensors and motors on the robot were recorded and our control method was designed. Using the data we have recorded, the magnitude of the external force acting on our humanoid robot has been found. To find the magnitude of the applied force, the ankle torque and position data of the robot were calculated statistically by multi-linear regression. According to this force value, the robot determines the magnitude and direction of the incoming external push and decides whether the movement it will make is forward, backward, or stop. In addition, the reward function that the robot will gain during training is created similar to the force.

DQN and DDQN algorithms, which are deep learning methods, are used for the robot to work autonomously and respond with the appropriate movement to the reaction it encounters. Thus, it is ensured that it stays in balance by self-learning against the pushes to be applied from different directions and different forces. Tests were also carried out on real robots with DQN and DDQN algorithms. Both results were compared and it was observed that learning was faster in the DDQN algorithm.

3. System requirements

3.1. Robotis-OP2

In this research, the humanoid robot Robotis-OP2 developed by Robotis company was used. Robotis-OP2 is an open-source robot environment with advanced computational capabilities designed according to the physical structure of humans. This robot is manufactured as a head, body, arms and legs.

There are a total of 20 Degrees of Freedom (DOF) on the robot, including 2 on the head, 3 on each arm and 6 on each leg. The Robotis-OP2 platform includes 20 motors, 20 position sensors, a 3-axis gyroscope, a 3-axis accelerometer, a camera, a speaker, a USB input, a mini HDMI, an ethernet input, and LEDs. Dynamixel MX-28 T servo motors are used for each joint. Also, an Intel Atom N2600@1.6 A GHz dual-core processor is used, and up to 4 GB of DDR3 Ram is supported. The foot base of this robot is designed to be rectangular. This design helps the robot to balance better. Robotis-OP2 has a height of 45.5 cm and a weight of 3.0 kg. The image of the robot is shown in Fig. 1. Robotis-OP2 supports both Windows-based and Linux-based operating systems.

3.2. Webots

Since the cost of robots is high, tests are first carried out in a virtual environment in order not to damage their hardware[13]. In this study, the Webots environment was preferred as the robot simulator. Webots is a free robot development simulator developed by the company Cyberbotics. Khpera Simulator is developed based on open-source code. Webots, which allows making 3D designs, are used in academic studies. Webots is a simple and useful simulator thanks to its plain interface. With the libraries it contains, it allows programming and development of many different robots. It can compile programming languages such as C/C++, Java, and Python in its own editor to perform robot controls. In addition, thanks to the MATLAB interface, it also provides the opportunity to compile the codes here. Learning and processing sensor data in the Webots environment is quite simple. Thanks to its libraries, it contains many functions that will process the data it receives from the sensors. The fact that Webots works independently of the operating system provides convenience to the developers.

3.3. Push-Recovery controllers

Many solutions have been presented for external forces and walking on rough terrain, which are the biggest balance problems

Fig. 1. Robotis-OP2.

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

of humanoid robots. Most of the existing walking algorithms are designed to walk on flat surfaces at a predefined static angle to the floor and with the feet parallel to it. Therefore, with each external push to a humanoid robot while walking, the probability of the robot's falling increases. Many different strategies have been introduced to overcome this problem. Among these strategies are three basic methods at best: 1) Ankle Strategy: It is a Center of Pressure (COP) Strategy that uses the robot's ankles to stabilize by changing the Center of Mass (COM) [2,4,5] 2) Hip Strategy: A Centroidal Moment Pivot (CMP) Strategy that uses both the ankles and hips to keep the robot in balance, allowing the body to change its position and create angular momentum by shaking the trunk[1]; and 3) Step Strategy: It is the Capture Step Approach Strategy that captures external forces by taking one or more steps in an appropriate direction[10]. The push-recovery strategies are given in Fig. 2.

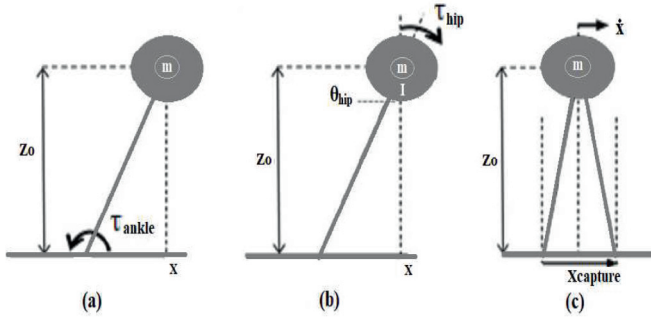

Fig. 3. (a) ankle strategy (b) hip strategy (c) step strategy.

3.3.1. Ankle strategy

The ankle strategy controls the center of the mass of the robot. It is also known as the center of pressure strategy. Mechanically, this strategy consists of rotating the robot's body around the ankle joints to maintain balance. In this strategy, other joints are fixed. To keep the center of mass in balance, a control torque is applied to the ankle joints. Thus, the robot behaves like an inverted pendulum with the applied torque. In this strategy, either the zero moment point on the ankle is controlled to control the ankle torque, or the target angle of the servo located on the ankle can be corrected. The ankle strategy equation,

$$\ddot{x} = \frac{g}{z_0}\left(x - \frac{\tau_{ankle}}{mg}\right), \tag{1}$$

where τ_{ankle} is the torque value, m is the mass of the linear inverted pendulum model, g is the gravity and z_0 is the height of the center of mass. ẍ is the horizontal distance of the center of mass from the current support point. Fig. 3.a shows the model based on the ankle strategy.

3.3.2. Hip strategy

In the hip strategy, the robot uses its trunk to maintain balance, creating a counterforce to the push. In other words, it is a push-recovery controller that uses the angular acceleration of the hip servo to counteract the motion of the robot's center of mass. The hip strategy is also known as the Central Moment Axis strategy. With this method, the servo motors in the hip part of the robot try to stay in balance by producing a torque suitable for the incoming push. They follow a similar path to how people try to stay balanced by using their hip joints against incoming pushes. This strategy is a more effective method than the ankle strategy. Mathematical model equation of hip strategy,

$$\ddot{x} = \frac{g}{z_0}\left(x - \frac{\tau_{hip}}{mg}\right), \tag{2}$$

$$\tau_{hip} = I\theta, \tag{3}$$

where τ_{hip} is the torque applied to the body, m is the mass of the model, g is the gravity, z_0 is the height of the center of mass, and I is the rotational inertia. ẍ is the horizontal distance of the center of mass from the current support point. The model based on hip strategy is given in Fig. 3.b. Humanoid robots cannot rotate around their axes, so the bang-bang control technique is used. The bang-bang control technique applies the maximum torque for the T_{R1} period first for the $\tau_{hip}(t)$ function. Then it applies the maximum negative torque for the T_{R2} period. The mathematical equation of this theory is as follows,

$$\tau_{hip}(t) = \tau_{max}u(t) - 2\tau_{max}u(t - T_{R1}) + \tau_{max}u(t - T_{R2}), \tag{4}$$

$$\theta\,(T_{R2}) = \theta_{max}, \tag{5}$$

where τ_{max} is the maximum torque that the joint can exert, T_{R1} is the moment when the acceleration of the body stops, T_{R2} is the moment when the body stops, u(t) is the unit step function, and θ_{max} is the maximum joint angle. This controller makes the hip servo's position work by applying maximum torque (τ_{max}) until it almost reaches its target angle. To stop the system, torque is applied in the opposite direction. After reaching the target position, the value of the hip angle should be returned to the initial value.

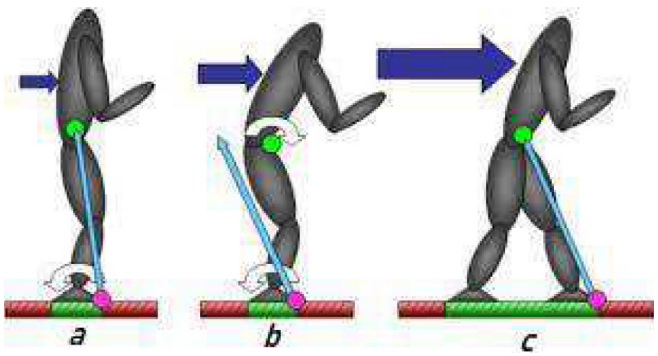

Fig. 2. Three basic balancing strategies. The green dot represents the center of mass, the pink dot represents the center of pressure, and the blue arrow represents the ground reaction force. (**a**) Ankle Strategy (**b**) Hip Strategy (**c**) Step Strategy [1]. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.3.3. Step strategy

As the torque value of the push applied to the robot increases, it becomes difficult for the robot to stay in balance. When the push from the outside exceeds the maximum torque that the robot will withstand, ankle and hip strategies are insufficient and the robot must take steps to remain in balance. This move, which is made by stepping to the support point against the push coming to the robot, is called a step strategy. For the robot to stay in balance without falling, the best step length it will take should be within the equilibrium point. In the step strategy, balance is achieved by taking a support step in the direction where the push comes from. This strategy is also known as the capture step. The capture point is a concept that is also widely used in human movements, and it is the support step taken by the robot to stop itself completely or to stay in balance. The capture point of a robot that is not in equilibrium is proportional to its body speed and is calculated as,

$$X_{capture} = \ddot{x}\sqrt{\frac{z0}{g}}, \tag{6}$$

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

where $X_{capture}$ is the capture point, ẍ is the linear velocity of the body. The step strategy shows that the capture point is proportional to the linear velocity of the robot's body and the distance of the step that must be taken to push-recovery the robot. The step strategy is given in Fig. 3.c.

3.4. Deep Q-Network (DQN) algorithm

DQN is a deep learning algorithm used with artificial neural networks in Q-learning. In Q-learning, the memory structure of the agent consists of Q tables. In DQN, the agent's memory is a convolutional neural network. The input of the DQN neural network is the states and its output is the actions performed by the agent. In the DQN algorithm, it keeps the samples (s, a, r, s^1) obtained in each discovery in its memory. Where s is the state, a is the action, r is the reward, and s^1 is the next state. In the DQN algorithm, first of all, the action a is performed in the case s. As a result of this action, an r prize is won and s^1 is passed to the next state. The flow diagram of the DQN algorithm is given in Fig. 4.

The most important feature of this algorithm is the reminder and replay functions. The agent tends to forget the experiences learned after a certain period of time. For the retraining of the agent, the experience learned is required. With the DQN algorithm, the agent's experiences are kept in memory. During learning, these data are used for training of the agent with random samples.

In our algorithm, the value that gives how far the agent's prediction is from the actual target is expressed as a loss. Here we can find the loss by subtracting the estimate from the target. In the DQN algorithm, the loss function is defined as,

$$L(\theta) = \left[Q(s,a) - (r + \gamma max Q(s^l, a^l)\right]^2. \tag{7}$$

The pseudo-code of the DQN algorithm is as follows.

1: Initialize replay memory(M)
2: Initialize random network weights
3: **for** episode = 1 **to** M **do**
4: s0 state initialization
5: **for** t = 0 to T **do**
6: Use the ε strategy to get a random a_t actions
7: Perform a_t action and advance to s_{t+1} and calculate current reward r_t.8:
(s_t, a_t, s_{t+1}, r_t) save the values to M
9: Calculate the loss value
$L(\theta) = \left[Q(s,a) - (r + \gamma \text{maxQ}(s^l, a^l)\right]^2$
10: $s_t = s_{t+1}$
11: ε = βε, update ε with β
12: **end for**
13: **end for**

When a disruptive input is applied, the system is said to be asyptotically stable if it approaches approximately the equilibrium point over time. In their study, Fan et al. [32] proposed to generate the convergence of the DQN algorithm by examining a simplified version of the fitted Q-iteration algorithm. The difference between this algorithm and DQN is that there is no repeat buffer M to store state-action-reward-state sets. In order to better understand the convergence of the Fitted Q-iteration algorithm, some important neural networks should be defined first [32–33]. The sparse family of ReLu networks is defined as,

$$W_{L,s,dj} = \left\{ fNNwithReLU : \max_{l\epsilon\{0,\cdots,L+1\}} \|\vec{\theta}_l\|_\infty \le 1, \sum_{l=0}^{L+1} \|\vec{\theta}_l\|_0 \le s, \max_{i\epsilon\{d_0,\cdots,d_{L+1}\}} \|f_i\|_\infty \le V \right\}, \tag{8}$$

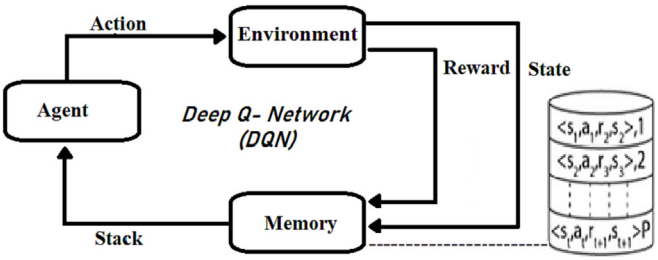

Fig. 4. DQN flow diagram.

where L is the number of hidden layers, s is the sparsity, dj is the width of the j-th layer, V ∈ (0, ∞), and $\vec{\theta}_l$ are all the weights and bias of the layer l. Here, we restrict the maximum weight to be 1 and the number of important neurons to be s. Hölder functions,

$$W_{L,s,dj} = \left\{ f : \mathbf{C} \to \mathbf{R} : \sum_{\alpha:|\alpha|<\beta} \|\partial^\alpha f\|_\infty + \sum_{\alpha:|\alpha|_1=\lfloor\beta\rfloor} \sup_{\substack{x,y\in C \\ x\neq y}} \frac{|\partial^\alpha f(x) - \partial^\alpha(y)|}{\|x-y\|_\infty^{\beta-\lfloor\beta\rfloor}} \le K \right\}, \tag{9}$$

where β, K > 0 and $\partial^\alpha = \prod_{i=1}^n \partial^{\alpha i}$. We will denote by G_q all the valid[1] compositions of q Hölder functions. Hölder functions $\widehat{G_q}$ and the extended sparse ReLu networks $\widehat{W_{L,s,dj}}$ are defined as,

$$\widehat{G_q} = \{f : SXA : \forall\alpha \in A, f(.,\alpha) \in G_q\}, \tag{10}$$

$$\widehat{W_{L,s,dj}} = \{f : SXA : \forall\alpha \in A, f(.,\alpha) \in W_{L,s,dj}\}. \tag{11}$$

Two assumptions have been made for the fitted Q-iteration algorithm to be valid. These,

- For all, f $\in W_{L,s,dj}$, we have that,

$$r(s,a) + \gamma \mathrm{E}\left[\max_{a' \in A} Q(S',a) | S'\ P(s,a)\right] \in \mathrm{Gq}\ . \tag{12}$$

- Let ν1, ν2 ∈ D[S × A] two absolutely continuous probability measures and μ = (μ1, ⋯, μi, ⋯) a policy for the a Markov decision process (MDP). Dentone $P_T^\mu V_1 = P^{\mu_T} \cdots P^{\mu_1 \nu_1}$ the probability distribution of states $\{(S_i, A_i)\}_{i=1,\ldots,T}$ of the MDP that has $(S_0, A_0) \sim \nu_1$.

Let π_t be the greedy policy in step t of the fitted Q-iteration algorithm. There exists a constant C ∈ R^+ such that,

$$\|Q^* - Q^{\pi_t}\|_{1,\nu} \le \frac{C}{(1-\gamma)^2}(\nu,\sigma)\gamma|A|\left(\ln n\right)^{1+\frac{2\xi n(\alpha-1)}{2}} + \frac{4\gamma^{t+1}R_{max}}{(1-\gamma)^2}, \tag{13}$$

where α is calculated by the following equation,

$$\alpha = max_j = 1, \cdots, q \frac{t_j}{2\beta_j \prod_{l=1}^{j+1} min(1, \beta_l) + t_j}, \tag{14}$$

where, n is the sample size, β_j is the constant defining the j-th Hölder functions in the composition of q Hölder functions, tj is the maximum number of input variables on which a j-th Hölder function depends, and ξ satisfies,

$$max\left\{\sum_{j=1}^q \left(t_j + \beta_j + 1\right)^{3+t_j}, \sum_{j=1}^q ln\left(t_j + \beta_j\right), \max_{j \in \{1,\cdots q\}} P_j\right\} \le \widetilde{C}(\log n)^{\frac{\xi-1}{2}}. \tag{15}$$

For some $\widetilde{C} > 0$, and P_j the deminsion of domain of f_i, the j-th Hölder function. The two terms in this bound correspond to: first, a statistical error,

$$\frac{C}{(1-\gamma)^2}(v,\sigma)\gamma|A|\left(\ln n\right)^{1+\frac{2\tilde{c}n(\alpha-1)}{2}}, \tag{16}$$

and second, an algorithmic error,

$$\frac{4\gamma^{t+1}R_{max}}{(1-\gamma)^2}, \tag{17}$$

that decreases by a factor of γ every iteration t. Thus, for large t, where the statistical error is dominant, the error rate achieved by the fitted Q-iteration algorithm will be,

$$\|Q^* - Q^{\pi_t}\|_{1,v}\infty|A|\left(\ln n\right)^{1+\frac{2\tilde{c}n(\alpha-1)}{2}}. \tag{18}$$

3.5. Double deep Q-Network (DDQN) algorithm

DDQN uses two identical neural network models (A and B). One of these neural networks learns during the repetition of experience (A), as DQN does, and the other is a copy of the last part of the first model where the value of Q is calculated by the second model (B). The flow diagram of the DDQN algorithm is shown in Fig. 5.

The loss function in the DDQN algorithm is defined as,

$$L(\theta) = \left[r_{i+1} + \gamma Q^T\left(s_{t+1} \cdot argmaxQ^E(s_{t+1}, a^l)\right)\right]. \tag{19}$$

We find the index of the highest Q value from the main model and use this index to derive the action from the second model. Q^E and Q^T represent the evaluation network and target network, respectively. The pseudo-code of the DDQN algorithm is as follows.

1: Initialize replay memory(M)
2: Initialize random network weights
3: Initialize the target network as Q^T (θ)= Q^E(θ)
4: C, target network equalization counter
5: **for** episode = 1 **to** M **do**
6: s0 state initialization
7: **for** t = 0 to T **do**
8: Use the ε strategy to get a random a_t actions
9: Perform a_t action and advance to s_{t+1} and calculate current reward r_t.10:
(s_t, a_t, s_{t+1}, r_t) save the values to M
11: Calculate the loss value
$L(\theta) = r_{i+1} + \gamma Q^T\left(s_{t+1}.argmaxQ^E(s_{t+1}, a^l)\right]$
12: s_t = s_{t+1}
13: ε = βε, update ε with β
14: **if** mod(T, C)==0 **then**
15: Q^T (θ)= Q^E(θ)
16: **end if**
12: **end for**
13: **end for**

A benefit from this variant is a better estimate of the true Q(s, a) values. It was shown in that, given some unbiased estimates Qt(s, a) such that,

$$\sum_{a}(Q_t(s,a) - V^*(s)) = 0, \quad \frac{1}{N}\sum_{\alpha}(Q_t(s,a) - V^*(s))^2 = k > 0, \tag{20}$$

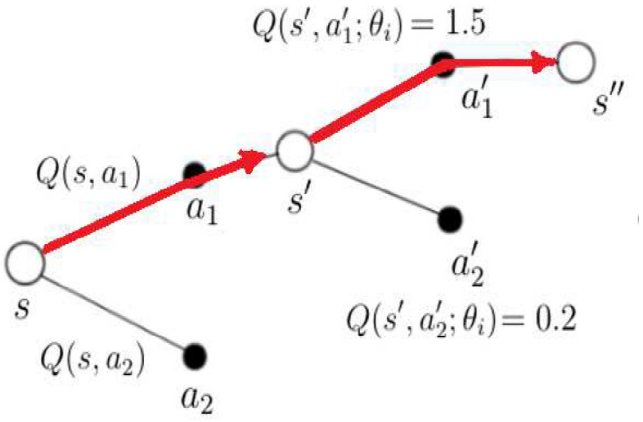

Fig. 5. DDQN Flow Diagram.

$Q_t(s,a)$ estimates have a strict lower bound given by,

$$\max_{\alpha} Q_t(s,a) - V^*(s) \geq \sqrt{\frac{k}{N-1}}, \tag{21}$$

whilst the DDQN estimates have a zero-valued lower bound,

$$\left|Q_t^A\left(s, argmax_a Q_t^B(s,a)\right) - V^*(s) \geq 0\right|. \tag{22}$$

4. Results and discussion

Our application has been carried out in real and simulation environments. The Webots development tool was used for the simulation environment. The humanoid robot that we performed our operations with is Robotis-OP2. A deep reinforcement learning algorithm was used to train the robot, and Python and C++ languages were used for coding.

By creating the environment in the Webots simulation tool, Robotis-OP2 has been integrated into the created environment. Information such as acceleration, gyro, torque and position from the robot's sensors were instantly received and recorded. The data recorded enabled checking the balance position range of the robot's body. A physics library was created to apply force to the robot from within the simulation and its connection with the robot was realized. Forces were randomly applied to the robot from the front and back. The developed controller algorithms in the simulation environment were transferred into real robot via of an internet cable. In both simulation and real environment tests of the study, the forces of random magnitudes between 10 N and −10 N were applied. The applied forces are constantly controlled and the robot is brought to the balance position by interfering with the ankle when it goes out of balance position. DQN and DDQN algorithms were used in the training process of the robot. The states in our algorithm are defined as,

$$S = (tSign,\ abs(Joint\_tork),\ anklePos,\ fSign,\ abs(Force)),$$

where tSign is the sign of the torque, abs(Joint_Tork) gives the absolute value of the torque, anklePos is the position of the ankle, fSign is the sign of the applied push, and abs(Force) is.
the absolute value of the magnitude of the applied push. To calculate the value of the applied push, we recorded the torque and position information of the ankle and used multi-linear regression. The following equation was obtained,

$$F = 20.9255 - (16.0364 * T) - (40.5437 * P), \tag{23}$$

where F is the magnitude of the applied push force, T is the torque of the ankle, and P is the position information of the ankle. A multi-linear regression graph of 120 randomly received positions and torque information is given in Fig. 8. R-squared was taken as the statistical measurement metric 1. There are three actions that the robot will take against the incoming push. These are respectively;

- stop (0),
- forward movement (1),
- back movement (2). The reward function to be gained according to action is expressed as,

$$R = 0.25 - T^2. \tag{24}$$

The reward is calculated with the equation (24). The parameters we used in DQN and DDQN algorithms are given in Table 1 in detail.

The maximum push forces that the system we have established can resist to remain in balance have been calculated. Then, push force was applied to the robot in different ways from the front and back, and the results were compared. Fig. 6 shows the push force coming from the back, and Fig. 7 shows the push force coming from the front. The flowchart of the humanoid robot movement is presented in Fig. 9.

7500 episodes were performed using the DQN algorithm. The robot has learned to stay in balance after about 7200 episodes. According to the reward function, the maximum score to be obtained from a fully successful section is 7,5. The robot, which was trained using the DQN algorithm, received a total score of 37483.51 at the end of the training. The results obtained with the DQN algorithm are shown in the graph in Fig. 10.

7500 episodes were performed with the DDQN algorithm. The robot has learned to stay in balance after about 5500 episodes. It was observed that a total of 45369.33 points were obtained with the DDQN algorithm during the training period. The results of the DDQN algorithm are presented in Fig. 11.

Table 1
DQN and DDQN algorithm parameters.

Parameters	Value
Epısodes	7500
Memory	2000
Gamma	0.95
Epsilon	1.0
Epsilon_Min	0.005
Epsilon_Decay	0.995
Batch_Size	64
Train_Start	100
Action_Size	3
State_Size	5
Maxstep	30

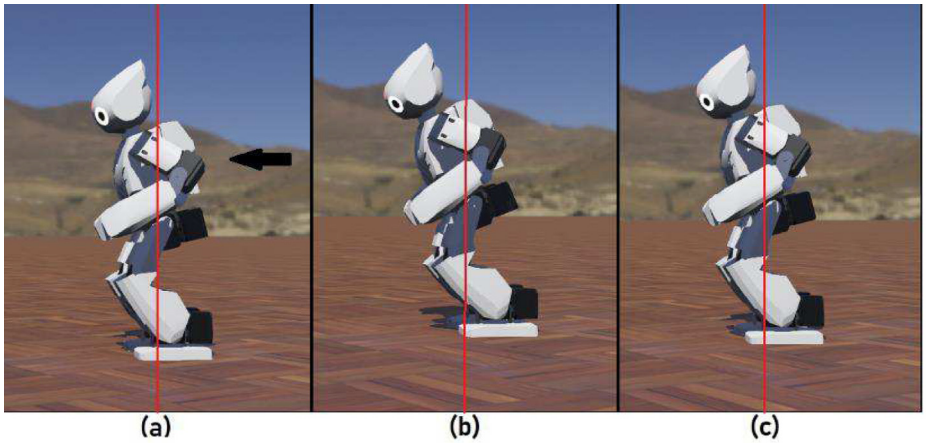

Fig. 6. (a) Applying the push from behind (b) Position taken as a result of the push (c) Reaching the equilibrium position.

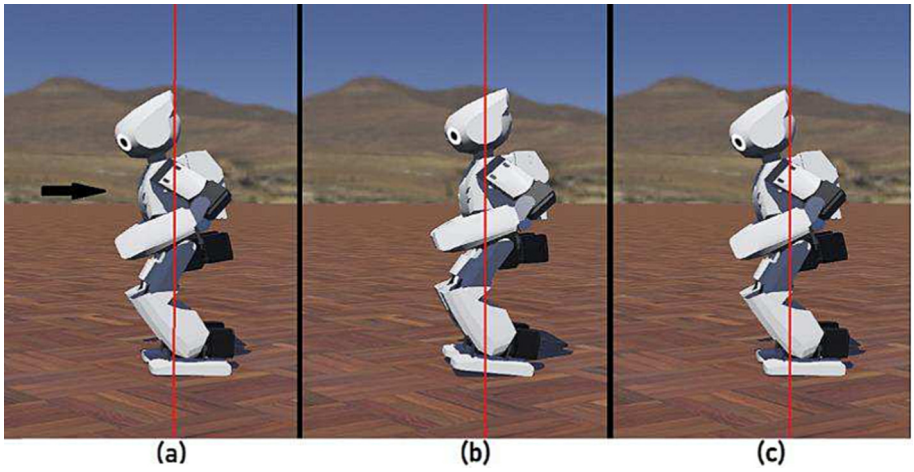

Fig. 7. (a) Implementation of the front push (b) Position taken as a result of the push (c) Reaching the equilibrium position.

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

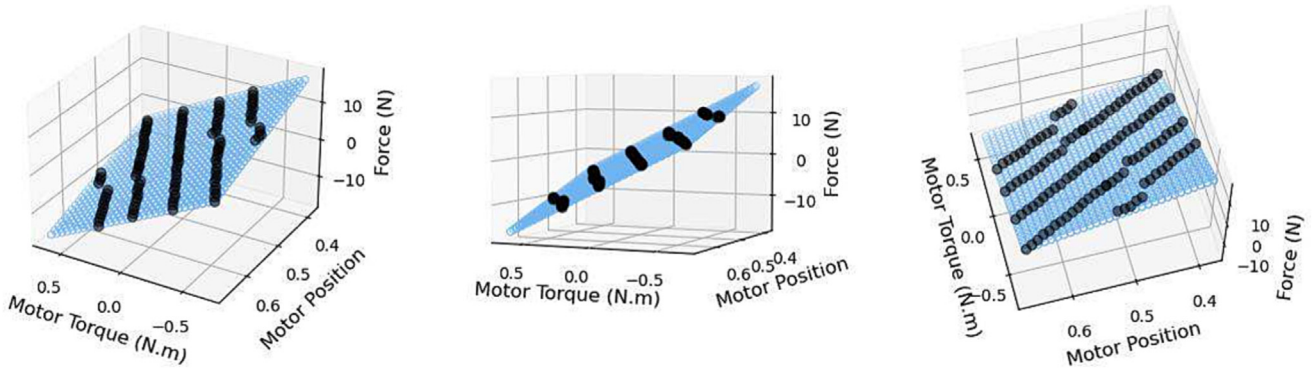

Fig. 8. Force graph.

The DDQN algorithm is faster than the DQN algorithm because it uses a double neural network. As a result of the experiments conducted, it has been observed that the DDQN algorithm is faster than the DQN algorithm. When the rewards are evaluated, it is

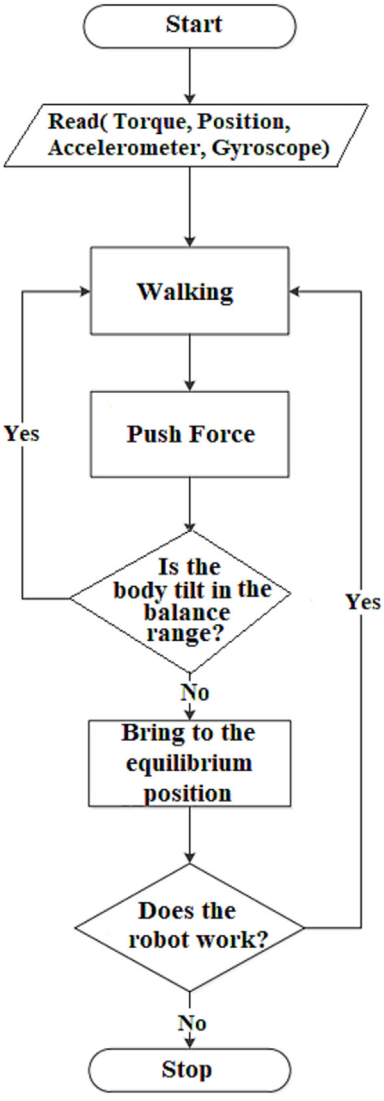

Fig. 9. Flow chart of the push-recovery controlled humanoid robot.

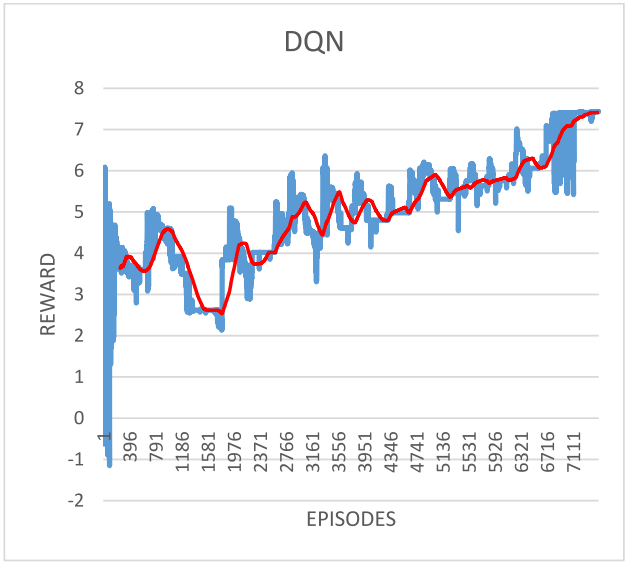

Fig. 10. Results of the DQN algorithm.

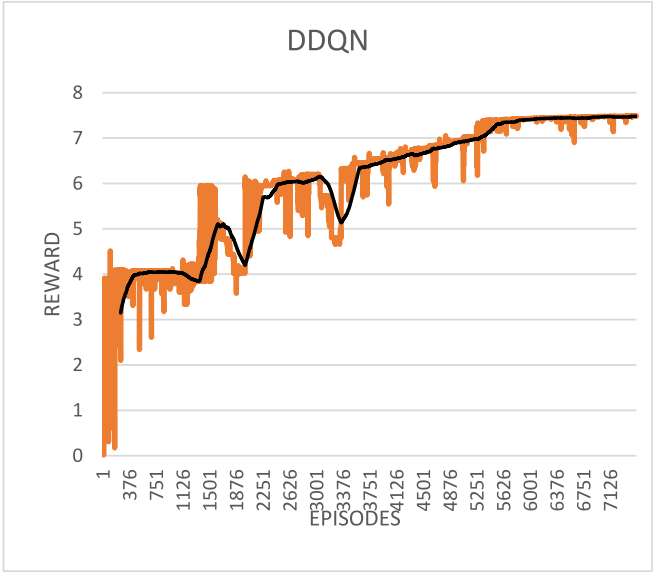

Fig. 11. Results of the DDQN algorithm.

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

understood that the rewards received with the DDQN algorithm are 21.03 % better than the DQN algorithm. The comparison graph of the results of DQN and DDQN algorithms is given in Fig. 12. It is seen in the graphic results that the DDQN algorithm is better than the DQN algorithm. The results obtained in both applications are given in Table 2.

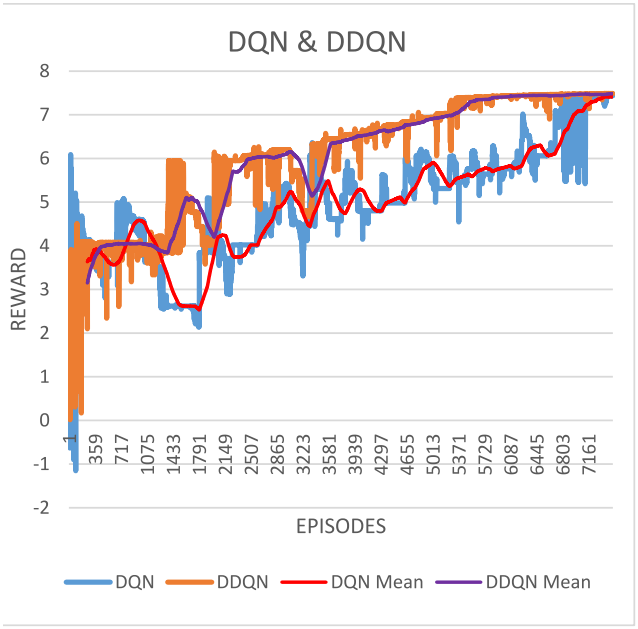

Fig. 12. Comparison results of DQN and DDQN algorithms.

The same tests performed in the simulation environment were also performed in the real environment. You can access the test video made in the real world via the link https://youtu.be/nWG53hVrAE8. In order to apply the application in the Webots simulation environment to the robot in the real world, a connection must be established with the robot. The robot control screen opens with the "Show Robot Window". On this screen, the connection is made by entering the IP address, username, password information and pressing the start remote control button. Thus, the application running in the simulation environment is transferred to the real robot. In the real environment, the results were recorded by applying pushes to the robot with the help of a stick. It has been observed that the robot is balanced against the pushes coming from the front and the back. Similar results to those in the simulation environment were obtained. Fig. 13 shows the real-world tests.

The designed push-recovery controller was developed using the ankle strategy. The ankle torque values produced by the robot against external forces were recorded. It has approached the equilibrium point where it is stable by reacting depending on the size of the external force. The torque values produced by the ankle according to the different forces applied to the robot are given in Fig. 14 and Fig. 15. The vertical axis the torque and the horizontal axis refers the step. Fig. 14 shows the torque values produced by the ankle as a result of the forces applied from behind. Fig. 15 shows the torque values produced by the ankle as a result of the forces applied from the front.

The positions taken by the ankle after the force applied to the robot are presented in Fig. 16 and Fig. 17. The vertical axis the ankle and the horizontal axis refers the step. Fig. 16 shows the force applied from behind, and Fig. 17 shows the force applied from the front. As seen in the figures, the robot approaches the equilibrium point after the disruptive force and shows stability.

Table 2
Comparison of the results.

Model	Total Episode	Section Achieved	Max Reward Per Episode	Total Reward
DQN	7500	7200	7,5	37483,5
DDQN	7500	5500	7,5	45369,3

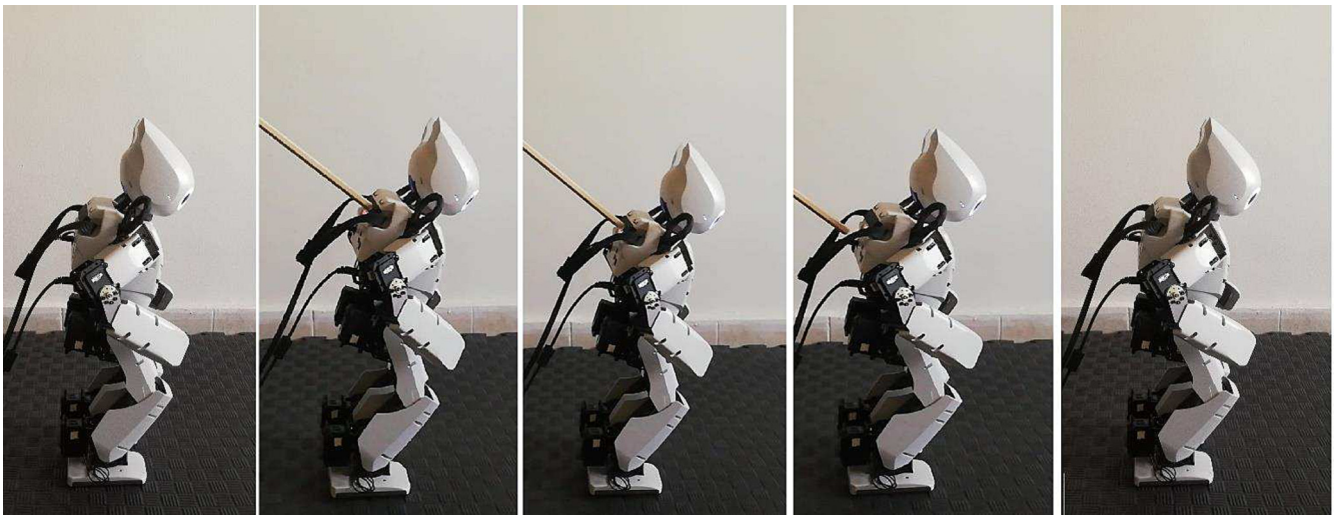

Fig. 13. Real-world testing.

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167

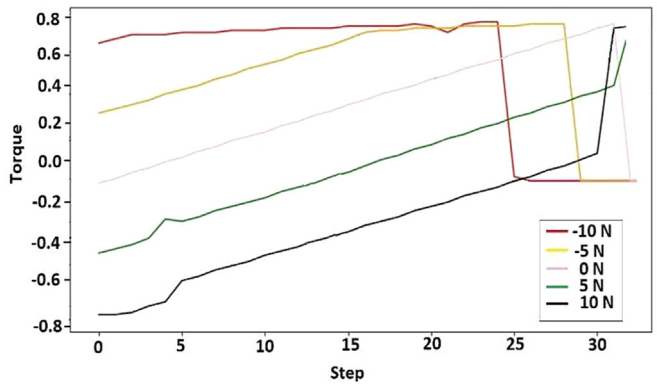

Fig. 14. Torque values produced by the ankle according to the different forces applied from behind.

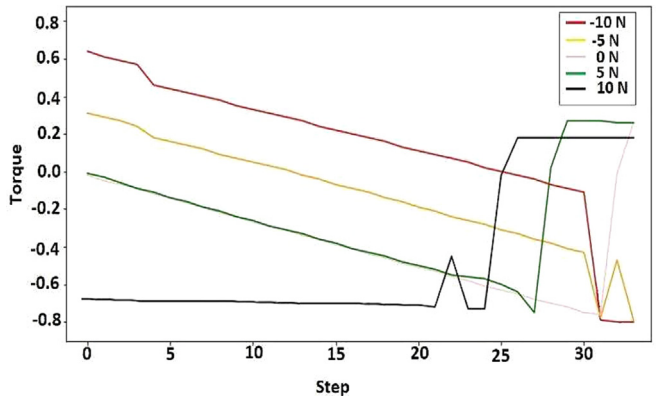

Fig. 15. Torque values produced by the ankle according to the different forces applied from the front.

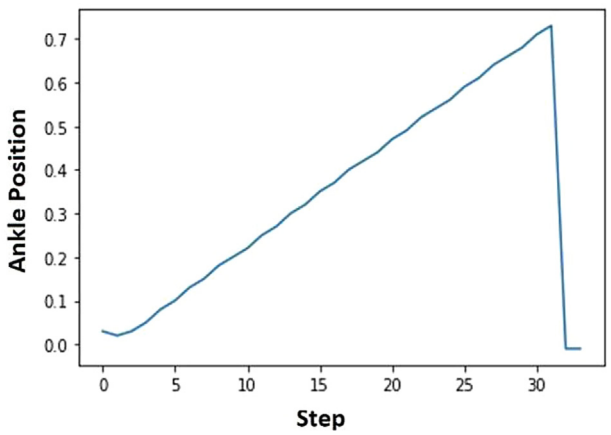

Fig. 16. Ankle position after force applied from behind.

5. Conclusions

In this study, a control system has been implemented that will allow a humanoid robot with 20 DOF to remain in balance against the push to be applied. A push-recovery controller has been designed using DQN and DDQN from deep reinforcement learning algorithms. The application was carried out based on the ankle strategy from the push-recovery controllers. Random pushing forces were applied to the robot from the front and back to keep it in balance. Forces between 10 N and −10 N were applied.This application was carried out in both simulation and real environments. The external push was applied to the robot with the help of a stick in the real environment by creating a library in the simulation. It has been observed that the robot has learned to stay balanced against front and rear pushes. Simulation and real environment results show parallels.

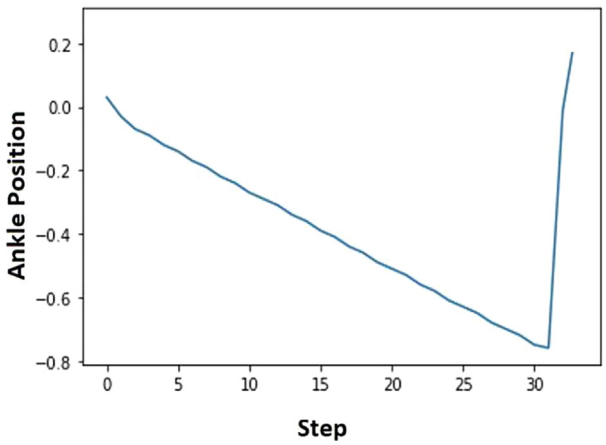

Fig. 17. Ankle position after force applied from front.

Learning to stay in balance with the DDQN algorithm was realized in the 5500th episode. Learning to stay in balance with the DQN algorithm was realized in the 7200th episode. In the proposed controller, it has been observed that the DDQN algorithm is better than the DQN algorithm. When the rewards are evaluated, it is understood that the rewards received with the DDQN algorithm are 21.03 % better than the DQN algorithm.

In our future studies, it is planned to design a more powerful controller by using the hip strategy, step strategies and their combinations from the push-recovery controllers. In addition, it is considered to compare push-recovery controllers by using different methods such as D3QN, FDQN, A2C, A3C, and PPO, which are deep reinforcement learning algorithms.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] B. Stephens. (2007) Humanoid push recovery. In Humanoid Robots, 2007 7th IEEE-RAS International Conference on, pp. 589–595. IEEE.

[2] Stephens B. Push recovery control for force-controlled humanoid robots. Pennsylvania USA: Carnegie Mellon University Pittsburgh; 2011. PhD thesis.

[3] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Aral, N. Koyachi, and K. Tanie. (2001) Planning walking patterns for a biped robot. IEEE Transactions on Robotics and Automation, 17(3) pp. 280–289. ISSN 1042296X. doi: 10.1109/70.938385.

[4] Graf C, Röfer T. A Center of Mass Observing 3D-LIPM Gait for the RoboCup Standard Platform League Humanoid. Berlin Heidelberg, Berlin, Heidelberg: Springer; 2012. p. 102–13.

[5] T. Assman, H. Nijmeijer, A. Takanishi, and K. Hashimoto. (2012) Biomechanically motivated lateral biped balancing using momentum control. Eindhoven: Eindhoven University of Technology, Traineeship report. - DC 2011.035.

[6] R. Neelapu, G. Lavanya Devi, K.S. Rao, (2018) Deep learning based conventional neural network architecture for medical image classification, Traitement du signal, pp.169-182.

[7] Gedik O, Demirhan A. Comparison of the Effectiveness of Deep Learning Methods for Face Mask Detection. Traitement du signal 2021:947–53.

[8] S. Behnke. (2006) Online trajectory generation for omnidirectional biped walking. Proceedings - IEEE International Conference on Robotics and Automation.

[9] M. Missura and S. Behnke. (2013) Omnidirectional capture steps for bipedal walking. In Proceedings of Humanoids, pp. 401–408, Atlanta, GA, 2013. ISBN 9781479926190.
[10] M. Missura and S. Behnke. (2015) Online learning of foot placement for balanced bipedal walking. IEEE-RAS International Conference on Humanoid Robots. Pp. 322–328, ISSN 21640580. doi: 10.1109/HUMANOIDS.2014.7041379.
[11] Pratt J, Carff J, Drakunov S, Goswami A. Capture point: A step toward humanoid push recovery. Proc IEEE-RAS Int Conf Humanoid Robots 2006:200–7.
[12] H. Y. Ong, K. Chavez, A. Hong, (2015) Distributed Deep Q-Learning, Computer Science.
[13] C. Robert, T. Sotiropoulos, H. Waeselynck, J. Guiochet, S. Vernhes, (2020) The virtual lands of Oz: testing an agribot in simulation, *Empirical Software Engineering*, Springer Verlag.
[14] Prakash VR, Saran. An Enhanced Coding Algorithm for Efficient Video Coding. Journal of the Institute of Electronics and Computer 2019:28–38.
[15] V. Mnih, K. Kavukcuoglu, D. Silver, et al. (2015) Human-level control through deep reinforcement learning, Nature, pp. 529-533.
[16] H. Kim, D. Seo and D. Kim, (2019) Push Recovery Control for Humanoid Robot Using Reinforcement Learning, 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, pp. 488-492, doi: 10.1109/IRC.2019.00102.
[17] Yi, Seung-Joon, et al. (2011) Online learning of a full-body push recovery controller for omnidirectional walking. Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on. IEEE.
[18] Yi S-J et al. Whole-body balancing walk controller for position controlled humanoid robots. Int J Humanoid Rob 2016;13(01):1650011.
[19] C. Yang, T. Komura and Z. Li, (2017) Emergence of humancomparable balancing behaviours by deep reinforcement learning, 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), Birmingham, pp. 372- 377, doi: 10.1109/HUMANOIDS.2017.8246900.
[20] Hosseinmemar, J. Baltes, J. Anderson, M. C. Lau, C. F. Lun, and Z. Wang. (2019) Closed-loop push recovery for inexpensive humanoid robots. Applied Intelligence. The International Journal of Research on Intelligent Systems for Real Life Complex Problems, 49(173):pp. 1–14. ISSN 0924-669X (Print), 1573-7497 (Online). doi: 10.1007/s10489-019-01446-z.
[21] P. Ghassemi, M.T. Masouleh, A. Kalhor, (2014) Push Recovery for NAO Humanoid Robot, 2014 Second RSI/ISM International Conference on Robotic and Mechatronics(ICRoM), IEEE.
[22] D. C. Melo and M. R. Omena Albuquerque Maximo, A. M. Da Cunha, (2020) Push Recovery Strategies through Deep Reinforcement Learning, 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), Natal, Brazil.
[23] R. Maulana, W. Kurniawan, H. Z. Fahmi, (2018) Noise Reduction on the Tilt Sensor for the Humanoid Robot Balancing System Using Complementary Filter, 2018 The 2nd International Conference on Mechanical, System and Control Engineering (ICMSC).
[24] Shu-Yin Chiang, jin-Long Wang, (2020) Posture Control for Humanoid Robot on Uneven Ground and Slopes Using Intertial Sensors, Advances in Mechanical Engineering.
[25] S. Yang, f. Chen, L. Zhang, Z. Cao, P. M. Wensing, Y. Liu, J. Pang, W. Zhang, (2021) Reachability-based Push Recovery for Humanoid Robots with Variable-Height Inverted Pendulum, 2021 IEEE International Conference on Robotics and Automation (ICRA).
[26] R. Schuller, G. Messesan, J. Englsberger, J. Lee, C. Ott, (2021) Online Centroidal Angular Momentum Reference Generation and Motion Optimization for Humanoid Push Recovery, IEEE Robotics and Automation Letters.
[27] G. Messesan, J. Englsberger, C. Ott, (2021) Online DCM Trajectory Adaptation for Push and Stumble Recovery during Humanoid Locomotion, 2021 IEEE International Conference on Robotics and Automation (ICRA).
[28] Tzuu-Hseng S. Li, Ping-Huan Kuo; Lin-Han Chen, Chia-Ching Hung, Po-Chien Luan, Hao-Ping Hsu, Chien-Hsin Chang, Yi-Ting Hsieh, Wen-Hsun Lin, (2022) Fuzzy Double Deep Q-Network-Based Gait Pattern Controller for Humanoid Robots, IEEE Transactions on Fuzzy Systems.
[29] Huang Q, Dong C, Yu Z, Chen X, Li Q, Chen H, et al. Resistant Compliance Control for Biped Robot Inspired by Humanlike Behavior. IEEE/ASME Trans Mechatron 2022.
[30] M. Shafiee-Ashtiani, A. Yousefi-Koma M. Shariat-Panahi M. Khadiv. (2016) Push Recovery of a Humanoid Robot Based on Model Predictive Control and Capture Point. Proceedings of the 4th International Conference on Robotics and Mechatronics pp. 26-28.
[31] M. Shafiee-Ashtiani, Milad Yousefi-Koma, Aghil Mirjalili, Reihaneh Maleki, Hessam Karimi, Mojtaba. (2017). Push Recovery of a Position-Controlled Humanoid Robot Based on Capture Point Feedback Control. pp. 126-131. 10.1109/ICRoM.2017.8466226.
[32] J. Fan, Z. Wang, Y. Xie, and Z. Yang, arXiv:1901.00137 [cs, math, stat] (2020), arXiv: 1901.00137.
[33] L.C. Mantilla Calderon, M. J. Junca Pelaez, éDeep Q-Learning" Universidad de Los Andes, Dogota, Colombia, 2021.

Emrah ASLAN received a BS degree the Computer Engineering and Electrical and Electronics Engineering Department at the University of Harran of Turkey in 2013 and 2019 respectively. He is the MS degree in electrical and electronics engineering from Harran University in 2016. He is currently a Ph.D student in electrical and electronics engineering from Dicle University. He is an Lecturer with the Silvan Vocational School, Dicle University since 2016. His research interests include humanoid robots, deep learning and renewable energy.

Muhammet Ali ARSERİM (Member, IEEE) received the B.S. degree in electrical and electronics engineering from Çukurova University, Adana, Turkey, in 1997, the M.S. degree in electrical and electronics engineering from Dicle University, Diyarbakır, Turkey, in 2001, and the Ph.D. degree in electrical and electronics engineering from Fırat University, Elâzığ, Turkey, in 2009. He is currently an Assistant Professor with the Electrical and Electronics Engineering Department, Dicle University. His current research interests include signal processing, embedded systems, and FPGA.

Ayşegül UÇAR received a BS degree, MS degree, and PhD degree from the Electrical and Electronics Engineering Department at the University of Firat of Turkey in 1998, 2000, and 2006, respectively. In 2013, she was a visiting professor at Louisiana State University in the USA. She has been a professor in the Department of Mechatronics Engineering since 2020. She has more than 21 years background in autonomous technologies and artificial intelligence, its engineering applications, robotics vision, teaching and research. Ucar is active in several professional bodies, in particular, she is an associate editor of IEEE Access and Turkish Journal Electrical Engineering and Computer Sciences and a member of European Artificial Intelligence Alliance Committee.

Please cite this article as: E. Aslan, M.A. Arserim and Ayşegül Uçar, Development of Push-Recovery control system for humanoid robots using deep reinforcement learning, Ain Shams Engineering Journal, https://doi.org/10.1016/j.asej.2023.102167