euspen's 23rd International Conference & Exhibition, Copenhagen, DK, June 2023
www.euspen.eu

Balance control of quadruped robot by Deep Q-Network

Shih-Chieh Chen, Jau-Liang Chen*

National Chung Hsing University, Department of Mechanical Engineering, Taichung, Taiwan

**jlchen@dragon.nchu.edu.tw*

Abstract
One of the issues of making an additive robot is posture balance control in a dynamic environment. In traditional method, we would build a kinematic and a dynamic model to estimate the ideal posture to counter the dynamic environment. Nowadays, the development of Reinforcement Learning (RL) attracts the researchers and engineers. It can solve multi-dimensional problems and give optimal solutions. There are already some successful applications of RL in robotics. In this topic, we attempt to use one of the RL methods, Deep Q-Network, to acquire a proper control model to make a quadruped robot balance on a balance board. The main idea to achieve robot balance is knowing the relationship of the robot posture and the ground. Therefore, we use a series of raw data from Inertia Measurement Unit and the position of foot end-point as the input. The control model outputs the final position of the foot end-point in order to adjust the posture. With proper design of the training parameters and train environment, we successfully obtained a control model that balance the quadruped robot when the ground tilt in random direction.

Keywords: Quadruped robot, Deep Q-Network

1. Introduction

In classic robot control, dynamic and kinematic analysis is necessary to get the control model. Even though those techniques could help acquire a robust control model, the process is mathematically complicated for a multi-DOF(degree of freedom) robot. The Machine Learning (ML) technique can categorize input data and generate a result, and the input data can mix with multiple sources. These characteristics make Machine Learning useful for solving a multi-dimensional problem. One of the common, and well-known ML algorithm is Deep Learning. The model learns the desired output with a group of expected input and output pairs (supervised data)[1-2]. Supervised data pair could be a known question and correct answer. But in robotics control, it is difficult to predict the correct answer in complex (multiple input) situations, there could be even no correct answer. Therefore, we prefer unsupervised techniques in robot control. Reinforcement Learning (RL) is a more common application[3-6]. RL does not require labeling the correct output. There is only desired output, by providing a guideline for the training agent. The principle of the training agent is maximizing the reward it can get by taking action.

In this paper, we introduce the procedure of implementing Deep Q-Network, one of the techniques of RL, to acquire a control model in simulation. The objective of the model is to keep the quadruped robot body remain horizontal. It requires direct input of sensory data and minimal information on the robot's state, to maintain a quadruped robot body and remain horizontal during operation.

2. Deep Q-Network

Deep QNetwork (DQN), or Deep Reinforcement Learning, is an algorithm derived from QLearning. The feature of DQN is it uses an artificial neural network(NN) to present a control model (policy π). NN can extract the characteristic of enormous input data, the so-called State (S). The output of NN represent the expected reward (Q) that the training agent can get by taking Action (A). The NN that represents the policy is called Policy Network (PNN). Since it just like a table recording expected reward of each State and Action pair. We can also use $Q_\pi(S, A)$ to represent PNN.

According to Mnih et al.[7], presenting a nonlinear function such as NN is unstable to represent policy. This may be because the correlation between data sequences is high, and the small change of update may significantly change the distribution of the NN. They present two solutions:

1. Experience replay: There is a memory that stores the "experience". Experience record the new State S_{t+1} after a timestep t by taking a specific Action A_t, and the Reward R that training agent received. Thus one experience can represent as a tuple of $(S_t\ ,\ A_t\ ,\ R,\ S_{t+1})$ in each time step t. When in learning stage, the agent randomly samples a number of transitions from memory to perform update

2. Generate the other neural network, called Target Network (TNN). TNN is a copy of PNN, it is used for generating expected best return of new State-Action pair $(maxQ_\pi(S_{t+1}, A))$. TNN is synchronized with PNN every numbers of steps and hold fixed between individual updates. This solution lower the error of supposed expected return and current State-Action value.

The workflow of DQN is shown in Figure 1. At first, after the initialization of the training environment. The training environment samples an initial current state(St). Then process moves to the execute phase. In the execution phase, the agent decides whether to choose an action from the PNN or randomly samples action from the action space base on ε - $reedy$ mechanism[8]. The agent interacts with the environment by executing the action A_t. The time step move to the next one $(t + 1)$, environment returns the new state S_{t+1} and a reward R. The next step is gathering the experience data $(S_t\ ,\ A_t\ ,\ R,\ S_{t+1})$

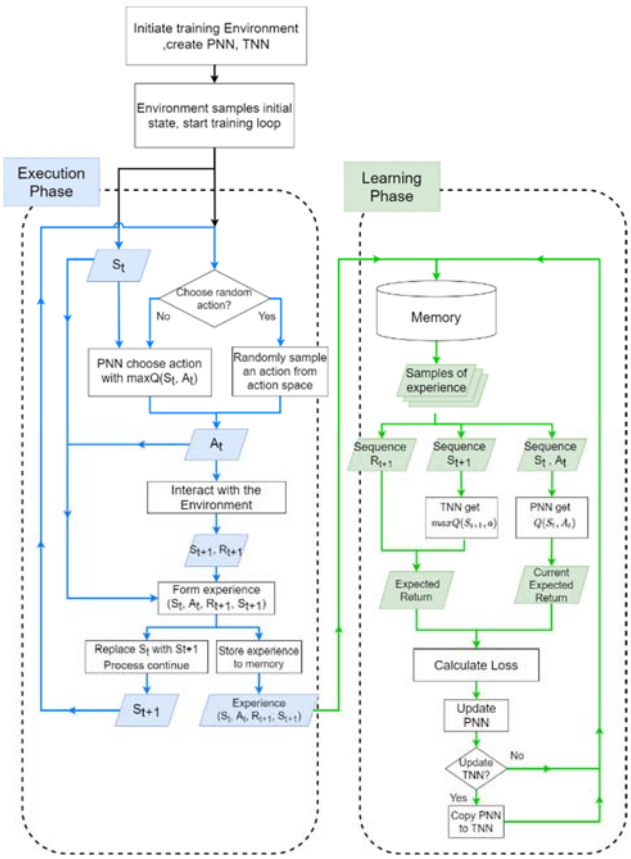

Figure 1. The detailed workflow of Deep Q-Network.

and send the experience to memory. The new state replaces the old current state, and the cycle continues.

In the learning phase, several experience data are randomly picked from the memory (a min-batch). The agent uses these data to calculate the loss for maintaining the weight of PNN. The loss function is defined below:

$$L(\theta) = \mathbb{E}_{(S_t, A_t, R, S_{t+1}) \sim U(D)} \left[\left(R_{t+1} + \gamma max Q_\pi(S_{t+1}, a; \theta') - Q(S_t, A_t; \theta) \right)^2 \right] \quad (1)$$

where L is the loss function, θ represents the weighting parameters in PNN, and θ' are the parameters in TNN; $Q(S_t, A_t; \theta)$ is the state-action value acquired from the PNN; $maxQ_\pi(S_{t+1}, a; \theta')$ is the maximum new state-action value in the TNN; γ is the discount rate; The symbol $\mathbb{E}_{(S_t, A_t, R, S_{t+1}) \sim U(D)}$ denotes a set of experience data pulled uniformly random from a data pool D, where storing experience in each time step t. Note that, it does not matter whether the learning phase should be synchronized with the execution phase. Both can function individually.

3. Training environment and DQN implementation

3.1. Training environment

Figure 2 shows the view of the simulated quadruped robot and environment. The ground (Figure 3) has the capability of rotating in the X, Yaxis, which is the distrublance source. The following parameters are the setup of the training process and the training environment:

- **Simulator minimal time resolution:** 4 ms; We design the parameters considering the working frequency of the IMU sensor
- **Initial robot height:** 35 cm. This indicate the initial distance between the robot body and the ground. For the foot endpoint position y of each leg.
- **Robot execute action frequency:** 5 Hz; This is the operating rate of the robot execute

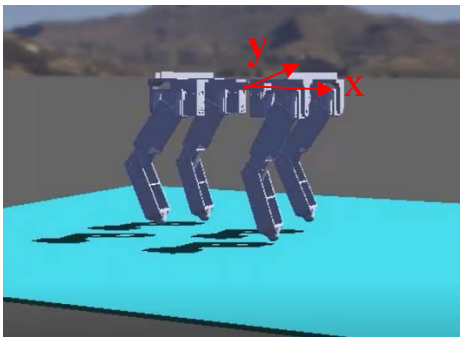

Figure 2. Simulated quadruped robot

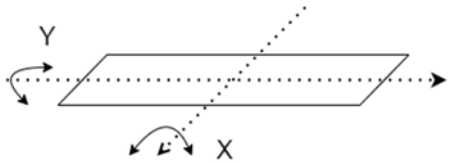

Figure 3. Configuration of ground rotation

an action. It is also the same as the decision rate of the learning agent.

- **Ground max incline angle:** 7 degrees; The ground tilt angle is random in our design. This parameter defines the maximum angle of the ground incline in any direction.
- **Ground angular velocity:** 3.5 degrees/second; The parameter refers to the lean rate of ground on axis X and Y.
- **Ground change period:** 10 second; This is the time that must pass before ground lean to another direction and angle.
- **Fail definition:** When the robot tilt angle is greater than 6 degrees, we define it as a fail operation. We will reset the simulation while keeping the learning system alive.

3.2. Deep Q-Network parameters design

The strategy is to let the learning agent identify the relation of robot's posture and ground. The action is the tilt direction along the robot body. Finally, the robot adjust the robot leg based on the action, and make the body of the robot remain balance.

- State:

  The state contains the information of robot posture and lean angle. In our configuration, we put the vertical distance of foot endpoint correspond to the robot body (H), and a series of raw angular velocity (V_t) and acceleration (Acc_t) data from simulated IMU sensor. We use a series of data since we expected that it contains characteristics to represent the robot posture (Acc_t) and dynamic motion (V_t). Equation below is the state design:

$$\begin{aligned} S_t = \{ & H_{LF}, H_{RF}, H_{LH}, H_{RH}, \\ & V_{x,t}, V_{x,t-1}, \dots V_{x,t-5}, V_{y,t}, V_{y,t-1} \dots, V_{y,t-5} \\ & Acc_{x,t}, Acc_{x,t-2} \dots . Acc_{x,t-14}, Acc_{x,t-15} \\ & Acc_{y,t}, Acc_{y,t-2} \dots . Acc_{y,t-14}, Acc_{y,t-15} \} \quad (2) \end{aligned}$$

  Where S_t is the state at time t; $H_{LF}, H_{RF}, H_{LH}, H_{RH}$ represent the height of left front, right front, left hind and right hind foot; $V_{x,t}$, $V_{y,t}$ represent the angular velocity along with robot axis X and Y at time t. $Acc_{x,t}$, $Acc_{y,t}$ denote the acceleration along robot axis X and Y respectively. Note that 1 time step is 4 ms.

- Action:

  We define two actions that denote the height change command on the endpoint of the robot legs, for X (roll)

direction and Y (pitch) direction. We set the motion in both directions to be symmetric. For example, if X action is up, the left side will push up while the right side will pull down in the next step. The detailed action setup shows below:

$$A_x \in \{-3, \dots k_n, \dots 3\ mm\ ; k = -3 + \frac{6*n}{20}; n = 0{\sim}20\}$$
$$A_y \in \{-3, \dots k_n, \dots 3\ mm\ ; k = -3 + \frac{6*n}{20}; n = 0{\sim}20\} \quad (3)$$

Where n can also viewed as the resolution of action.

- Reward:

  In this research, we only implement 3 reward:

  i. **Instant incline angle (degree)**: The value is based on the instant robot incline angle (degree)in new state (φ):

  $$R = 0.4 * e^{-1.5*abs(\varphi)} \quad (4)$$

  Figure 4 shows the distribution of this reward function.

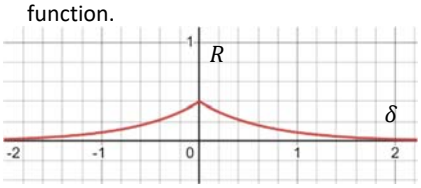

Figure 4. Reward distribution of instant incline angle

  ii. **Toward horizontal:** The purpose of this reward is further encourage agent stay around horizontal. It gives reward when the change of incline angle ($\Delta\varphi$) is more horizontal and give punishment if φ become larger after action execution. The distribution is shown in Figure 5.

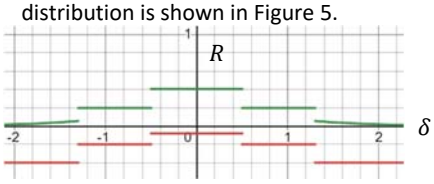

Figure 5. The reward distribution of different situation. Green line represent reward, while red line is punishment.

  iii. **Fail:** If φ is larger than 6 degree. Give -0.5 as punishment.

- Other parameters:

  There are 2 important parameters. One is learning rate, which refer to the change rate of weight pararmeters in PNN. The value we set is 10^{-5}; The other parameter is discount rate γ, the value is 0.1. The value is relatively low since we expect the agent choose action depent on current situation instead of unknown future.

4. Training result

The training process took about 200k decisions to become stable. Then we save the model and deploy it. Figure 6 Shows the deploy result. We can observe that when the ground lean to a random direction. The control model can maintain the robot body to horizontal. We also put a video link below the figure.

Figure 7 shows the data record of the deployment. We focus on the ground incline angle on axis X, Y, and the distribution of roll, pitch angle of the robot. The dot lines represent the timestep when the ground tilt to another random direction. The result shows that the learning agent not only learns to maintain the robot body to horizon, it also learns to reduce movement or even stop moving when the robot body's inline angle is very close to 0.

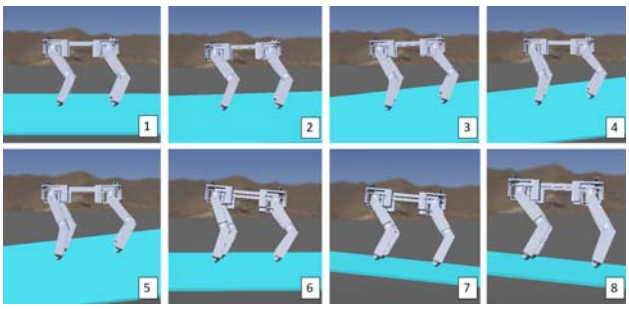

Figure 6. Execution result of the trained control model. The video record link: https://youtu.be/SSIKzoDHBqM

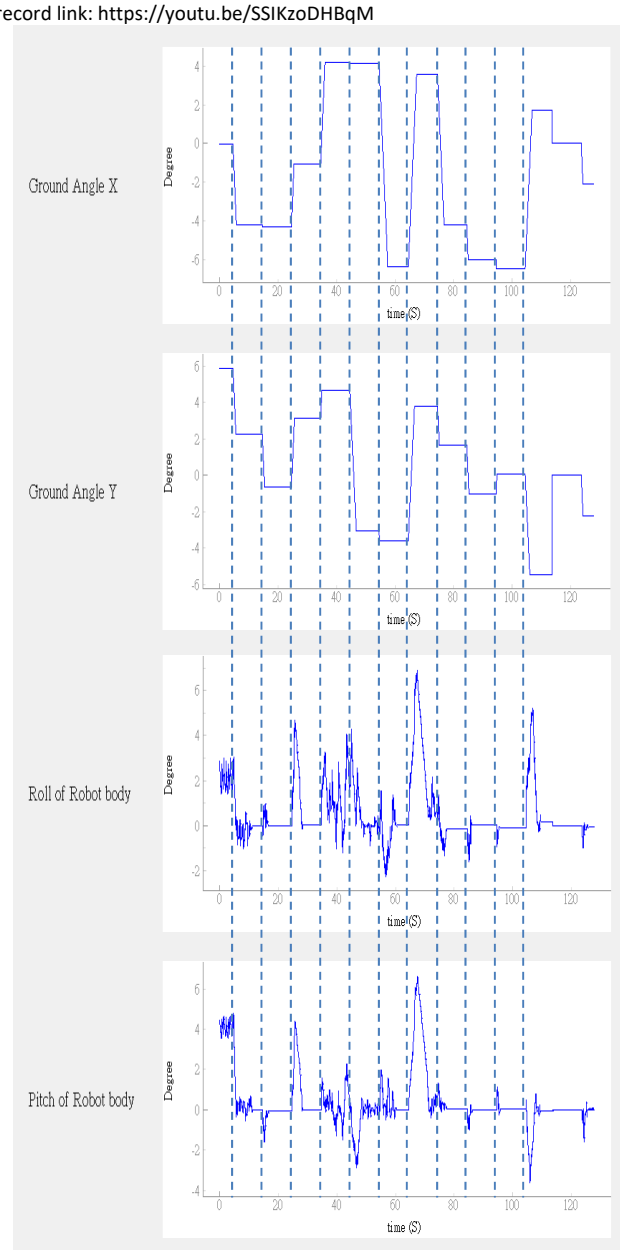

Figure 7. Detail record of ground rotate angle (Top 2) and the roll and pitch angle of the robot body. The dot lines mark the timestep when ground change its incline direction

Even though the control model can maintain the robot body to the horizon, we still observe (timestep 60~80 second) a high peak when the ground rotates both axis X, and Y significantly at once.

5. Conclusion

We successfully use one of the Reinforcement Learning techniques, Deep Q-Network, to acquire a control model in simulation. The control model can maintain the robot body when the ground rotates in a random direction. The State design allows the model to identify the relationship between the robot's posture and the incline angle, and the direction of the ground. It is also possible for a NN to extract characteristics of the robot's posture from the series of simulated raw sensor data. The current parameter design cannot handle the condition when ground rotate both axis X, and Y significantly at once.

Acknowledgement

This research was founded by Minister of Science and Technology (MOST) of Taiwan, ROC, under contract number MOST 109-2221-E005-078. We also sincerely thank the Taiwan Computing Cloud (TWCC) for giving essential support for this research.

References

[1] Crisci, C., Ghattas, B., & Perera, G. 2012. A review of supervised machine learning algorithms and their applications to ecological data. Ecological Modelling, **240**, 113-122.

[2] Joulin, A., Maaten, L. V. D., Jabri, A., & Vasilache, N. 2016. Learning visual features from large weakly supervised data. In European Conference on Computer Vision, 67-84.

[3] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V., 2018, Sim-to-real: Learning agile locomotion for quadruped robots, Robotics: Science and Systems. IEEE.

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

[5] Ananthakrishnan, A., Kanakiva, V., Ved, D., & Sharma, G. 2018. Automated Gait Generation for Simulated Bodies Using Deep Reinforcement Learning. In 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT). IEEE., 90-95.

[6] Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., & Hutter, M. 2019. Learning agile and dynamic motor skills for legged robots. Science Robotics, **4**(26), eaau5872.

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. 2015. Human-level control through deep reinforcement learning. nature, **518**(7540), 529-533.

[8] Tokic, M., & Palm, G. 2011. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In Annual conference on artificial intelligence. 335-346.