euspen's 21st International Conference & Exhibition, Copenhagen, DK, June 2021
www.euspen.eu

# Robotic limb gait-tracking using deep-q-network

Eric J. Tzeng[1], S. C. Chen[1], J. L. Chen[1], A. E. Roche[2], J. L. Chen[1]

[1]*Department of Mechanical Engineering of Chung-Hsing University in Taichung, Taiwan*
[2]*Independent Researcher admitted to the Master program in Mechatronics at New York University in New York City, New York*

*jlchen@nchu.edu.tw*

**Abstract**
In the current landscape of robotic limb control for biped and quadruped robots, inverse kinematics and other mathematical methods are used to model the position and torque of the limb. However, due to the complexity of their parameters, inverse kinematic models can result in infinite solutions unless constraints are set. Moreover, force and torque models are required to ensure the multiped robot can maintain stability during operation. This paper explores the adoption of the reinforcement learning technique, Deep Q Network (DQN). The model-free condition of DQN allows the model to learn the gait without the constraints that are required in the inverse kinematic model. Under the DQN model, the robot limb is able to successfully complete a steady cycloid gait within 3 600 cycles. When faced with a randomly generated obstacle, the robot limb under the DQN model is still able to successfully complete the cycloid gait cycles using a raw model. Furthermore, this method shows substantial torque optimization under a given reward function with different load conditions. The model allows additional reward constraints.

Keywords: Deep-Q-Network, gait-tracking, robotics , control, machine learning.

## 1. Introduction

In the general case, to control a biped or quadruped robot, geometric-based forward kinematics, inverse kinematics, and other mathematical methods have been used to build control systems to execute gait-tracking. For real world implementation, it requires dynamic torque/force modeling to ensure the robot operates with minimal interference. Unfortunately, this modeling process is complex and time comsuming.

In recent years, the development of computer technology has substantially increased the efficiency of machine learning in its implementation in robotics. Reinforcement Learning is a popular Deep Learning technique that is frequently used in academia and industry and has recently been used to explore robotic limb control.

One feature of Reinforcement Learning (RL) that makes the method appealing is its ability to be used in the model-free case where a complete model is not required for implementation [1]. RL can exploit the full solution space depending on the reward function and reward factors used. An optimal control policy can be achieved through prioritizing each of these functional requirements.

A classic and well-known method of Reinforcement Learning is the Q-table algorithm, one of the methods of Q-learning. It calculates the cumulative rewards for every action and state the agent can take in an environment. This particular algorithm is fairly intuitive for simple taks and easily adaptable to other algorithms. In the case of robotic limb control, the vast number of combinations of states and actions in this scenario makes Q-tables too simplisitic. However, Deep-Q-Network(DQN)[2] combines neural networks and Q-learning to overcome this limitation and achieve complex tasks like gait-tracking.

## 2. Model design

### *2.1. Robot Limb Model*

The limb model used in this paper is shown in Figure 1. The model is composed of three motors as active joints and three links. The error of distance is the distance between the linkage endpoint and target gait point. The state definition is dependent on the difference in radius and the difference in angle. Since each target point of the endpoint has an infinite amount of solutions, without RL, the mathematical representation of the model can become very complex when given constraints. This situation provides the possibility for the DQN learning system to search for the optimal solution.

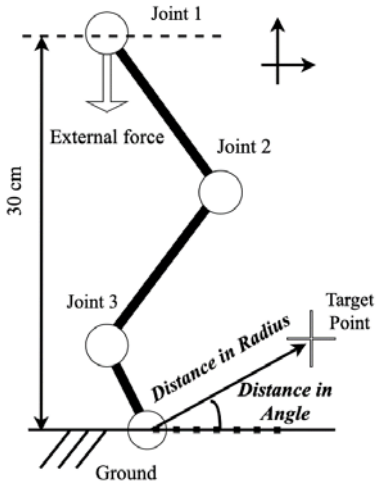


**Figure 1.** This is the basic training setup of the environment where J1, J2, and J3 represent the three linkage joints.

### *2.2. Simulation System*

The first goal is to verify that the presented method can do gait-tracking which can be achieved through the construction of a simple simulator with a basic environment. The environment contains the forward kinematic model and dynamic force analysis functions to generate different properties such as position, velocity, acceleration and torque at each time step. These values describe the current condition of the simulated limb model, which can be evaluated by reward functions to quantify the policy's performance. The static and dynamic force of the limb model is also analysed and put into a physics engine to observe simulated torque. The simulated model also takes external force into consideration for a more realistic simulation.

## 3. Training setup

### *3.1. State, Action definition*

The following five elements define the state:

i. Error of angle ($E_t^{\theta}$): The absolute angle between end point and target point.
ii. Error of radius ($E_t^{r}$): Distance between end point and target point.
iii. Absolute angle of motor 1 ($\theta_t^{J1}$)
iv. Absolute angle of motor 2 ($\theta_t^{J2}$)
v. Absolute angle of motor 3 ($\theta_t^{J3}$)

The action is defined by the velocity command for each motor (joint 1 ($\omega_{J1}$), joint 2 ($\omega_{J2}$), joint 3 ($\omega_{J3}$))

### *3.2. Reward definition*

Five reward values are:

i. Reach Target: If the distance of the endpoint and target gait point is smaller than a certain value, give a positive reward.
ii. Target radius error: If the distance of the endpoint and target gait point is going down, give a positive reward.
iii. Step Increase: To ensure the endpoint reaches the target gait point in an efficient way, give a small punishment (negative reward) for each iteration of the learning process.
iv. Fail: If there is interference with the robot limb, the distance of the endpoint of robot limb and target gait is too large, or the robot limb hits an obstacle, the largest punishment is given and the agent enters the terminal state.
v. **Torque difference of each motor**: Loading a single motor significantly more than the others is not desired for torque balance. A small reward is given if the motor torque difference decreases.

### *3.3. Gait design*

The designed gait shown in Figure 2 will be split into multiple points and inputted into the learning algorithm one by one. If limb model interference happens or the endpoint is too far from the target gait point, then the limb is reset to the first point of the gait.


**Figure 2.** The designed gait for training. The left one is the normal cycloid gait; The right one is normal cycloid gait with a randomly generated obstacle.

### *3.4. Deep Q Network training process*

The processdure of the training process is shown in Processdure 1. It also shows the interaction between state-action and Deep Q Network algorithm.

Processdure 1. Deep Q Network for gait tracking

Initialize policy network parameters $\vartheta$.
Initialize target network parameters $\vartheta^{-1} \leftarrow \vartheta$.
Initialize learning memory.
Initialize counter C $\leftarrow 0$
Initialize simulated robot model.
Choose target gait.
**Repeat**
>Get the next target gait point, and state
**Repeat**
>Epslon greedy decide to get action from policy network or random action.
>Interprete action as robot model command
>Get new state and reward.
>Store state, action, reward, new state as transition and put into learning memory.
>Sample one mini batch with size N from learning memory and calculate gradient by

$$\mathbf{loss} = \gamma(r + maxQ(s_{t+1}; \vartheta^{-1})) - Q(s_t, a; \vartheta)$$

>Update state $\leftarrow$ new state
>**if** remainder(C/$I_{target}$) == 0 **then** update target network parameters $\vartheta^{-1} \leftarrow \vartheta$
>**end if**
>**if** to next target **then** break
>**end if**
**Until** terminate singal triggered.

## 4. Experiment result

### *4.1. Cycloid gait*

A simple cycloid gait on a flat terrain determines if the trained DQN policy can solve the inverse kinematics of the robot foot and follow the predefined gait trajectory (the light grey cycloid line). This will prove the state definition and reward function are suitable for the model. In Figure 3, the limb model is strictly following the gait trajectory. The left side shows the trajectory of the predefined gait and the initial posture of the limb model. The right side shows how the limb model is moving in the global coordinate aspect.

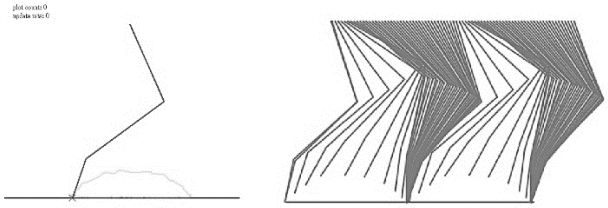

**Figure 3.** Two cycles of the training results for a cycloid gait on flat terrain.

### *4.2. Cycloid gait with random ramp obstacle.*

The random ramped gait adds supplementary information to the environment of the robot limb. In this training scenario, the environment will generate a ramped terrain, 10 cm in width and 3 cm in height, in an arbitrary time and position. The modified gait will adapt its gait trajectory depending on the environment.

The model can easily be implemented in a realtime-generated gait trajectory because the state definition is independent of

gait definition. In Figure 4, the model's endpoint makes contact with the ramp and moves forward. In the case where the model does not accomplish moving forward, it will overfit and remain in a fixed gait trajectory that will fail to follow all gait points.

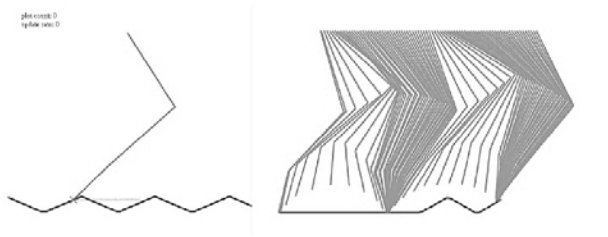

**Figure 4.** Two cycles of the training results for a cycloid gait with a random ramp obstacle. On the right, the gait commences from the ground for the first gait cycle, then steps on the ramp obstacle in the second cycle. The left shows the model's capability of following the dynamic gait.

### *4.3. Torque balance optimization*

Using a DQN model allows the possibility of including additional requirements to the control system that would otherwise mathematically overwhelm other robotic limb control systems.

Figure 5 exhibits the torque distribution history of the cycloid gait with and without a torque reward. The comparison between the model with torque reward and the model with no torque reward shows a significant reduction in the torque difference between the middle motor and lower motor.

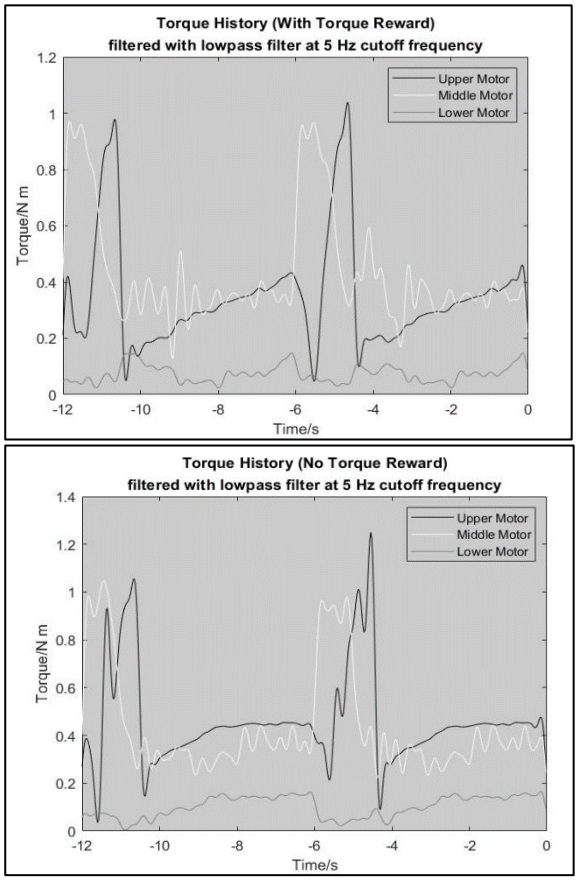


**Figure 6.** Comparison of torque distribution with (top) and without(bottom) torque balance reward. In both graphs, the lower motor exhibits no visible change in behavior when a torque reward is imposed. However, the upper motor shows red fluctuations in value with the torque balance reward.

From a visual standpoint, there is less fluctuation in the torque of the upper motor when the robot limb is airborne under the reward function. The middle and upper motor achieve closer values in torque as well. Figure 5 also exhibits a small increase in torque for the middle motor when the model includes a torque reward.

The distribution in the average torque difference in raw data can be observed in Table 1. The optimization can achieve up to 10 percent improvement under 3Kg to 7Kg external loading applied. The inclusion of torque rewards exhibits the model's ability to change its behaviour according to the additional reward function without failing its original gait-tracking task.

**Table 1.** Detail average torque difference in different conditions.

| Loading/kg | Average torque difference without torque difference reward/N-m | Average torque difference with torque difference reward/N-m | Optimization rate percentage |
|---|---|---|---|
| 3 | 1.7443 | 1.5692 | 10 % |
| 4 | 2.4338 | 2.2146 | 9 % |
| 7 | 4.5181 | 4.0751 | 10% |

### *4.4. Real robot limb implementation.*

After training the model in simulation. The model was apply to the real robot limb and make it track the same cycloid gait in simulaiton. The result is shown in Figure 6. From Figure 6, the robot limb can track the desire cycloid gait, this shows that trained model can be applied in real world. The video record of real robot implementation is mentioned in reference[3].

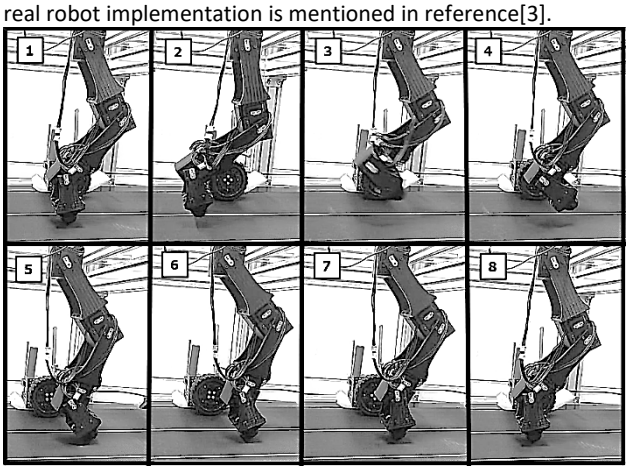


Figure 5 The result of real robot implementation, from No.1 to No.8 are the gait tracking process history.

### *4.5. Video record of cycloid gait, extra gait experimentation.*

To further investigate the effect of reward design, the same model was applied to different gaits and randomized obstacles. The trained model achieved the gait tracking successfully as predicted in the simulated model. Video links of the various scenarios tested are provided in the references[4-9].

## 5. Conclusion

In this paper, a control model was successfully constructed to achieve stable gait-tracking using DQN model training instead of the geometric-based inverse kinematic model. The model can be trained into a stabilized state in 3600 cycles of gait execution, successfully tracking the gait points and optimizing torque performance simultaneously.

At the phase of model execution, it can achieve gait-tracking accuracy up to 1 mm and has shown an observable decrease in joint-wise torque optimization up to 17% in various loading conditions. In the result of random ramped simulations, it shows that this model is not only capable of tracking static finite gait points but also realtime-generated gait points, which proves the robustness of the trained model. Due to the methodology of reinforcement learning, a DQN model can encounter overfitting issues. In the context of a robotic foot following a gait, an overfit model will learn to follow a predetermined trajectory. However, incorporating an obstacle shows the model can adopt real-time adjustments to its gait.

This paper demonstrates the potential of using reinforcement learning techniques such as DQN to simplify the inverse kinematic modelling process, and achieve multiple targets by reward design. Such technique can be used in multi-linkage robot control policies for fitting multiple function requirements without complex mathematical derivation.


## Acknowledge

We thank to National Center for High-performance Computing (NCHC) for providing computational and storage resources.


## References


[1] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press. p22.

[2] 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.

[1] Chen, S C. “Robotic limb gait-tracking using deep-q-network: Real robot leg implementation.” *YouTube*, YouTube, 29 Oct. 2020, https://www.youtube.com/watch?v=Re-sKyV8oxc

[2] Chen, S C. “Robotic limb gait-tracking using deep-q-network: Cycloid gait tracking.” *YouTube,* Youtube, 29 Oct. 2020, https://youtu.be/4Ean3-Scw6E.

[3] Chen, S C. “Robotic limb gait-tracking using deep-q-network: Variational Cycloid gait with random ramp ground.” *YouTube,* YouTube, 29 Oct. 2020, https://www.youtube.com/watch?v=-M0E8V_FQj8.

[4] Chen, S C. “Robotic limb gait-tracking using deep-q-network: Straight line gait tracking.” *YouTube,* YouTube, 29 Oct. 2020, https://www.youtube.com/watch?v=yY56sAFnmhE.

[5] Chen, S C. “Robotic limb gait-tracking using deep-q-network: Straight line gait tracking with an obstacle..” *YouTube*, YouTube, 29 Oct. 2020, https://www.youtube.com/watch?v=7RoSvj8fjAY.

[6] Chen, S C. “Robotic limb gait-tracking using deep-q-network: Online training.” YouTube, YouTube, 29 Oct. 2020, www.youtube.com/watch?v=Vmk7U4cMfvc.