Reinforcement Learning For Walking Robot

V Akila¹,³, J Anita Christaline¹, A Jothi Mani¹ and K Meenakshi²

¹ Electronics and Communication Engineering Department, SRM Institute of Science and Technology, Chennai India
² Computer Science Engineering Department, SRM Institute of Science And Technology, Chennai India

Abstract. Reinforcement learning namely Deep Q Learning neural network is used to make a robot walk in a controlled environment. The main objective of this paper is to make a robot learn to walk by building a model in hardware with six forms of movement. The basic alignment of robot at any moment depends on the direction of movement of stepper motor of a leg joint by test and trial method without any preordained model. The robot is made to freely walk which is stopped without falling down and is able to move freely over a transportable platform. Hardware robot communicates to the learning model which is run on the local machine through a USB port connection.

Keywords:-Deep Q-Learning; Reinforcement Learning; Hardware Walking robot.

1. INTRODUCTION

In order to achieve the human behaviour movements are based on actions and characteristics of the robot. For achieving complicated control problems in robots that resemble humans the most serious objective to be achieved in a walking robot is to make the robot walk in a similar pattern to human movements. As by previous studies, hardware robots uses either a pretrained or simulated model to make the robot to learn the pattern of walking.

In the previous model uses reinforcement learning algorithm uses divide and conquer approach [1]. Based on a particular learning pattern of repetitive walking [2] successive patterns of walking of biped robot is achieved. Similarly pattern of periodic pattern based on the decision of online measurements [3]. Trajectory learning process based on movements made dynamically for lower limb by removing unpredictable states [4]. Minimising the error between the temporal form of continuous waveform without interference from the surroundings [5]. There is another work which takes much longer time but with simpler neural networks and algorithm [6]. Design can be implemented using 4 actuators and 6 internal degree of freedom and it adapts itself using optimised learning algorithm [7]. The main purpose of this paper is to design a robot in such a way that it can accomplish a system framework that can enable a robot to walk without already used model methods. Robot has the ability to move in 4 directions and can move up and down the stairs and along uneven ground [8].

Gait initiation is adapted according to postural adjustment with an anticipated sequence stair movement [9]. Focussing on the natural dynamic movements of robot biped walking is done using other alternate methods. Two layer biologically inspired learning robot system in which there are 2 modules with upper level of learning model and a lower level of walking patterns by a generator.
called central pattern generator to generate recurring patterns that match with sensory inputs [10]. A nonlinear optimisation process with a probability model in a real and in simulated surrounding with optimise walking and stepping process [11]. Most of the learning system in the previous studies can relate only with intellectual property of the environment. If the surrounding parameters are established correctly for a model to learn it can improve the learning process without the need for physical biped robots. However the above mentioned analysis is not enough for implementing physical biped robots.

In the past 20 years studies are made in order to make an alternate for physical robot that can be made to perform bipedal walking by the advancement in superfast computing technology. Initially there were static walking robot with the design that inside the foot area always had the projection support for centre of gravity on a flat surface. The design had some issues with walking speed and in adapting itself to the environment in making only small steps [11]. As an alternate to this dynamic walking is done where the projection of centre of gravity is outside the area of foot support whereas the total angular momentum point is zero and it cannot be outside the foot area support[12]. Faster running system with hop and run is balanced and maintained by controlling velocity and attitude control part [13]. Modified robot dynamics enables the robot to walk at a relatively high speed based on the interaction of physical robot with virtual learning agent [14].

2. MATERIALS AND METHODS

The hardware component is connected to four jumper cables using a standard 10 cm lead wires. A USB AB cable is used for connecting microcontroller and the local machine. In an external container 3 sets of size 48 mm x 35 mm x 10 mm mini breadboards are enclosed. The actuators takes 2048 steps for completing one full revolution. Motor driver is required for connecting a microcontroller to the stepper motor. This model uses 13-bit high resolution measurements for identifying inclination changes less than one degree. This uses ATMEGA microcontroller for controlling all the operations. The power source, 2 rechargeable batteries, microcontroller and stepper motor drives are mounted inside a container. The body of the robot is 90 mm x 90 mm x 45 mm cuboid shaped 4 mm thick cardboard box. This box encloses the accelerometer breadboard circuit and the LED indication to specify the state of communication with the local machine along with the stepper motors wiring and external container wiring.

Based on the dimensions of the stepper motor, body and the ball bearing two software Blender 2.81 and Autodesk Inventor Professional 2020 are used to develop 3D models. Each leg is divided into components such as upper and lower leg in Figure.1

![Figure 1. Upper and lower leg 3D image](image)

The foot is shown as in figure 2. The interface of body and leg is shown in figure 3.
2.1 Reinforcement Learning

In this paper, for dynamic learning which requires continuous feedback from the learning requires Reinforcement learning and it does not require labelled input/output pairs. For implementing reinforcement learning [15], this model uses Deep Q Learning in the environment that is stated in the form of a Markov decision process for implementing dynamic programming techniques.

2.1.1. Markov Decision Process.

A Markov Decision Process (MDP) is a tuple (S, A, T, R) where;

- S is the set of different states, in this case state is represented by the 3 - Dimensional orientation values plus its sign-inverted values (x, y, z, -x, -y, -z) of the robot at a given instant t.
- A represents a set with different actions that can be taken at each time t. There are twelve actions that the robot can take, move each of the six stepper motor in clockwise or anti-clockwise direction
- T is called the transition rule:
  
  \[
  T : (a_t, s_t, s_{t+1}) \rightarrow P(s_{t+1} | s_t, a_t)
  \]

  where P(st+1 | st, at) is the probability that the future state is st+1 given that the current state is st and the action played is at. The distribution of probability of the future states at time t + 1 is given by T given the current state and the action taken at time t. Hence, we can predict the future state st+1 by a drawing a random value from the distribution
  
  \[
  T : s_{t+1} \sim T (a_t, s_t, .)
  \]

- R is the reward function:

  Reward gained for choosing action at in the state st is given by rt. After defining the Markov Decision Process, it is important to remind that it relies on the following assumption: the probability of the future state st+1 only depends on the current state st and action at, and doesn’t depend on any of the previous states and actions. That is:

  \[
  P(st+1 | s0, a0, s1, a1, ..., st , at ) = P(st+1 | st , at )
  \]

Many complex MDP’s are used in robot navigation such as [16].

2.2.2. Policy Function

The policy function \( \pi \) is exactly the function that, given a state st , returns the action at:

\[
\pi : st \rightarrow A
\]

Let’s denote by \( \pi \) the set of all the policy functions. Then the choice of the best actions to play becomes an optimization problem. Indeed, it comes down to finding the optimal policy
\( \pi \) * that maximizes the accumulated reward:

\[
\pi^* = \arg\max_{\pi \in \pi} \pi . t \geq 0 R(\pi(st), st)
\]  

- When temporal difference \( t(at, st) \) is high, it receives "good surprise".
- When temporal difference \( (at, st) \) is low, it receives "frustration".

The temporal difference calculated is used in the last next step of the Q learning algorithm to reinforce \((a,s)\) pair from time \( t-1 \) to \( t \), with the help of the equation:

\[
Q_t (at, st) = Q_{t-1} (at, st) + \alpha TD_t (at, st)
\]

In this point of view, the Q-values measure the accumulation of the positive or negative temporal difference associated with the state action pair \((at, st)\). In the surprise or positive case, reinforcement takes place, and in the frustration or negative case, weakening of the learning takes place. The objective is to learn Q-values that will fetch more positive values. Based on this, the decision of action to be taken usually relies on the Q-value \( Q(a, s) \). If the action \( a \) is taken in the state

2.2.3. Future Cumulative Reward.

However we can still improve the model. The elements \( r_t, r_{t+1}, \ldots, r_n \) are values we are trying to estimate with the reward function \( R \).

\[
R_t = R(at, st) + R(at+1, st+1) + \ldots + R(an, sn) = r_t + r_{t+1} + \ldots + r_n
\]

At time \( t \) we are unsure of the reward in this future time, the more we look into the future the more uncertain it is. In other words, the larger is \( t' \), the larger is the variance of the estimated reward \( r_{t+j} \). So, in order to fix that we have to discount each of the single rewards in the future, and the discount has to subsequently increase based on how far we are in to the future. So to do this we have to take the discounted sum of rewards:

\[
R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n r_n
\]

where \( \gamma \in [0, 1] \). That way the higher is \( t' \), the smaller is \( \gamma t' \), and therefore the more \( r_{t+j} \) is discounted. \( \gamma \) is called the discount factor. The closer \( \gamma \) is to 0, the more the learning will try to optimize the current reward \( r_t \). The closer \( \gamma \) is to 1, the more the learning rate will aim to optimize the future reward.

2.2.4. Q-Value

Each state and action pair \((a,s)\) we have an associated numeric value \( Q(a,s) \):

\[
Q : (at A, st S) Q(at, st) R
\]

We will say that \( Q(a, s) \) is "the Q-value of the action \( a \) played in the state \( s \)."

2.2.5. Temporal Difference

At the beginning \( t = 0 \), all the Q-values are initialized to 0. Now let's suppose we are at time \( t \), in a certain state \( st \). We play the action \( a \) and we get the reward \( r_t \). Then we take a random draw from the \( T(at, st, .) \) distribution, which leads us to the next state \( st+1 \):

We can now introduce the temporal difference, which is at the heart of Q-Learning. The temporal difference at time \( t \), denoted by \( TD_t (at, st) \), is the difference between:

- \( r_t + \gamma \maxa (Q(a, st+1)) \), \( \gamma \in [0, 1] \), that is the reward \( r_t \) obtained by playing the action at in the state \( st \), plus a percentage (which is our previous discount factor \( \gamma \)) of the Q-value of the best action played in the future state \( st+1 \),
- \( Q(at, st) \), that is the Q-value of the action at taken in the state \( st \), thus leading to

\[
TD_t (at, st) = r_t + \gamma \maxa (Q(a, st+1)) Q(at, st)
\]

\( TD_t (at, st) \) is like an intrinsic reward. The Q-values will be learned in such a way that:

- \( st \) is has a large Q value \( Q(at, st+) \) associated to it, the has higher chances of choosing \( at \). And if the Q value associated is low the learning rate will have less inclination in considering the action. There are several ways of obtaining the best action to take. First, when being in a certain state \( st \), we could simply take a with which we have the maximum of Q(a, s):

\[
a = \arg\maxa (Q(a, s))
\]
But experience has shown that this is not the best option. A better solution is the softmax method. We get the action to take by picking a random value from that distribution: \( a \sim Ws(.) \)

3. PROPOSED SYSTEM

The idea is to let the robot learn to walk without any simulations or pre-trained models. This is ideal because it overcomes bias in learning trajectories and policies that may occur in a simulation environment. And also taking into consideration that all the dynamics of the real world cannot be recreated in a simulation, this approach seems to be a potential solution. However, letting the physical robot to learn without any heuristic data comes with a price. The problem is the time that the physical robot takes to fall, get up and learn to walk compared to the incredibly fast simulations that achieves the same in significantly less time. In order to compensate for this time factor, the proposed system involves a robot that learns to walk in a constrained environment which will speed up the learning process since, the process (actions) of falling and getting up is eliminated as described below, this makes the system comparatively faster than a free roaming robot learning to walk by falling and getting up. The system consists of two parts, as depicted in Figure 4, one the neural network model that exists in the local machine and second the physical robot and its associated components.

![Figure 4. Biped Robot with the walking environment](image)

The neural network model uses a simple deep Q-learning algorithm with three hidden layers. The first hidden layer has twenty-four nodes, the second with eighteen nodes and the third with fourteen nodes. The input and output layer had six and twelve nodes respectively. The neural network was fully connected with random initial weights and Adam optimizer [16]. The model parameters such as the learning rate, discounting factor (gamma) and temperature parameter are determined by experimentation. The modal uses Smooth L1-loss as the loss function. The temperature parameter is a random value obtained by trial and error, it determines how sure the RL agent is in taking the action. It is multiplied with the softmax value of the actions. The 3-Dimensional orientation of the robot at a particular instant is obtained from an accelerometer sensor and it along with its sign inverted values (positive to negative and vice versa) is used as the input state to the deep neural network. The accelerometer readings are of the form \((x, y, z, -x, -y, -z)\) which represent the acceleration of the robot in the respective axis expressed in \((m/s^2)\) or \(G\)-forces \((g)\). Since the robot doesn’t undergo very fast movements the value is usually in the range of \(-2\) to \(2\) Table 1.

| Xa  | Ya  | Za  |
|-----|-----|-----|
| 1.05| 0.04| 0.11|
| 1.02| -0.07| 0.11|
| 1.04| -0.02| 0.19|
| 0.98| 0.01| 0.07|
Using the orientation data the level of tilt of the robot is determined. If the tilt exceeds a particular threshold value, which again is determined by experimentation, the reinforcement learning agent is given the maximum negative reward as going beyond the threshold tilt value corresponds to falling of the robot however, the robot doesn’t fall as it is suspended and continues to walk. This is done overcome the significant delay of the robot actually falling down and getting back up. Maintaining the tilt within a certain range provides the agent with a small positive reward enabling to understand that it is supposed to maintain the balance. The output layer of the deep neural network consists of twelve neurons, corresponding to the twelve actions or Q-values, where each Q-value represents the rotation of the corresponding stepper motor by ten steps in that particular direction, six clockwise and six anti-clockwise respectively for each stepper motor. Each time the best action is determined by the action selection policy which takes the softmax of the twelve values which is then multiplied with the temperature parameter.

The hardware biped robot consists of two physical sections, one is the body of the biped walking robot with the two legs housing all the six stepper motors and the accelerometer sensor, second is the external container housing the Arduino microcontroller and the power source which is connected to local machine via USB Serial port communication at 250000 bit baud rate. The six joints of the legs i.e. hip, knee and ankle joints of both the leg are connected to the respective stepper motor on one side and a small 6 mm (inner diameter) ball bearing on the other side. The ball bearing is used to distribute the load of body on the leg thereby reducing the amount of strain on the stepper motor. The robot is suspended from the top support and stands on top of two discs. These discs are freely movable, hence when the robot tries to walk the discs acts as a moving platform. The suspension prevents the robot from falling and stepping out of the discs. In order to prevent still state where the robot uses the suspension to stay still and not move there by ensuring continuous gain of positive reward, a small negative reward is given i.e. the robot is punished if it stays still instead of trying to walk.

4. RESULT
A gradual increase in the rewards obtained by the RL agent can be seen after experimenting with different network parameters. Figure 5 shows the reward window score curve after 3000 epochs starting with random initial weights and with temperature parameter (T) 700.

![Figure 5. 3000 epochs T=700](image)

The reward window score is value that can be used to visually represent whether the Reinforcement Learning agent is learning the right set of actions, in this case learning to walk. It is the average of the last n rewards obtained by the RL agent, here the value of n used is 1000. This score is plotted against each epoch.
The initial spike in all the curves are due to the stable standing state before the first action performed by the biped robot. In order to identify global maxima, several times the model was run with different initial weights, number of epochs and temperature parameters. Figure 6 shows the reward curve with 4000 epochs and temperature parameter set to 7000.

As we can see from Figure 7 and Figure 8 setting the temperature parameter $T$ to smaller value decreases the reward scores and also results in not so stable action selections, even though the number of epochs is higher (6000).
After exploring and tweaking other parameters such as the learning rate and discounting factor gamma, a model with a close to stable action selection and some gradual increase in the reward widow was obtained, as seen in Figure 9.

![Figure 9. 1000 epochs T=15000](image)

5. Future Work
The results from this approach shows that there is a potential of physical robots learning to walk without pre-trained models and simulations, however to be able to establish it concretely many factors have to be ensured and validated about the system. The motors and microcontroller used in this system are those that are available for many other use cases and not custom manufactured for this purpose. Also, there are several stages that lack precision and completeness in this implementation, when it comes to the hardware environment, as most of them were built with commercially available tools and materials. With respect to the reinforcement teaching agent several different aspects are yet to be explored, such as different optimizers, network architecture, discounting factor, etc. Improving on these factors while also considering methods that can scale this approach well, allowing it to fit into larger and complex systems to achieve autonomous learning.

6. References

[1] Chew CM, Pratt GA. Dynamic bipedal walking assisted by learning. *Robotica*. 2002 Sep;20(5):477-91.

[2] Morimoto J, Nakanishi J, Endo G, Cheng G, Atkeson CG, Zeglin G. Poincare-map-based reinforcement learning for biped walking. *InProceedings of the 2005 IEEE International Conference on Robotics and Automation* 2005 Apr 18 (pp. 2381-2386). IEEE.

[3] Morimoto J, Cheng G, Atkeson CG, Zeglin G. A simple reinforcement learning algorithm for biped walking. *InIEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04*. 2004 2004 Apr 26 (Vol. 3, pp. 3030-3035). IEEE.

[4] Yuan Y, Li Z, Zhao T, Gan D. DMP-based motion generation for a walking exoskeleton robot using reinforcement learning. *IEEE Transactions on Industrial Electronics*. 2019 May 17;67(5):3830-9.

[5] Doya K. Reinforcement learning in continuous time and space. *Neural computation*. 2000 Jan 1;12(1):219-45

[6] Benbrahim H, Franklin JA. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*. 1997 Dec 1;22(3-4):283-302.

[7] Tedrake R, Zhang TW, Seung HS. Stochastic policy gradient reinforcement learning on a simple 3D biped. *In2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*(IEEE Cat. No. 04CH37566) 2004 Sep (Vol. 3, pp. 2849-2854). IEEE.
[8] Hirai K, Hirose M, Haikawa Y, Takenaka T. The development of Honda humanoid robot. InProceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146) 1998 May 20 (Vol. 2, pp. 1321-1326). IEEE.

[9] Gelat T, Brenière Y. Adaptation of the gait initiation process for stepping on to a new level using a single step. Experimental brain research. 2000 Aug 1;133(4):538-46.

[10] Morimoto J. Humanoid Locomotion and the Brain. InHumanoid Robotics and Neuroscience: Science, Engineering and Society 2015 Jun 12. CRC Press/Taylor & Francis.

[11] Morimoto J, Atkeson CG. Nonparametric representation of an approximated Poincaré map for learning biped locomotion. Autonomous Robots. 2009 Aug 1;27(2):131-44.

[12] Takanishi A, Naito G, Ishida M, Kato I. Realization of plane walking by the biped walking robot WL-10R. InTheory and Practice of Robots and Manipulators 1985 (pp. 383-393). Springer, Boston, MA.

[13] Raibert MH. Hopping in legged systems—modeling and simulation for the two-dimensional one-legged case. IEEE Transactions on Systems, Man, and Cybernetics. 1984 May(3):451-63.

[14] Geng T, Porr B, Wörgötter F. Fast biped walking with a sensor-driven neuronal controller and real-time online learning. The International Journal of Robotics Research. 2006 Mar; 25(3):243-59.

[15] Szepesvári C. Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning. 2010 Jul 7;4(1):1-03.

[16] Theocharous G, Rohanimanesh K, Maharlevan S. Learning hierarchical observable Markov decision process Models for robot navigation. InProceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164) 2001 May 21 (Vol. 1, pp. 511-516). IEEE.