Robot Walking via Deep Deterministic Policy Gradient

Simon Kang
Chung-Ang University
Seoul, South Korea
simonkang@cau.ac.kr

Seung Tae Choi, Advisor
Chung-Ang University
Seoul, South Korea
stchoi@cau.ac.kr

***Abstract*—Utilizing MATLAB's standard robot model as our base, we applied the Deep Deterministic Policy Gradient (DDPG) for reinforcement learning. With a defined reward function, stopping criteria, and sidewalk conditions, our robot autonomously learns its environment, striving to optimize actions based on long-term rewards.**

***Index Terms*—Reinforcement Learning, Deep Deterministic Policy Gradient (DDPG), MATLAB, Robot Modeling, Autonomous Learning, Reward Function, Stopping Criteria.**

I. Introduction

Research on humanoid robots has traditionally emphasized the optimization of static leg postures with the aim to minimize the torque generated by motors at the joints. However, such approaches do not extend seamlessly into dynamic activities, such as walking, which involve complex biomechanical interactions and constant shifts in the center of mass.

To extend the scope from a static to a dynamic context, this work adopts Reinforcement Learning (RL) algorithms to optimize the walking posture of humanoid robots. Specifically, the Deep Deterministic Policy Gradient (DDPG) algorithm serves as the RL agent, tasked with learning the most effective leg movements for walking. The environment for the agent is a modeled sidewalk, where the state is defined as the robot's current location and posture, and actions correspond to walking steps. The objective function is framed as maximizing the distance walked before the robot deviates from the sidewalk, thereby optimizing the overall walking efficiency and stability.

II. Background

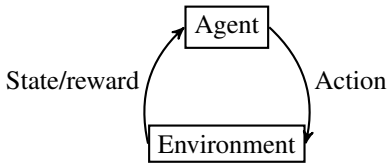

Fig. 1. Reinforcement Learning Diagram

Reinforcement Learning (RL) is a paradigm in which an *agent* learns through continuous interaction with its *environment*, using feedback to make optimal decisions. Refer to Figure 1 for a schematic representation of this process. The key components of this interaction are detailed as follows:

- **Agent**: The decision-making entity. For instance, in a self-driving car context, the car acts as the agent.
- **Action**: A specific decision or move the agent executes in its environment.
- **Environment**: The external context within which the agent operates.
- **Observation**: The data perceived by the agent post-action. It comprises:
  - **State**: Represents the agent's current understanding or perception of the environment. In the self-driving car analogy, it could be the data from cameras and LiDARs.
  - **Reward**: The feedback received after taking a particular action. For a self-driving car, it might be the distance covered without an accident.

In essence, the primary objective of RL is to determine an action A that maximizes the expected reward at a given state S.

III. Methodology

A. Distinguishing Reinforcement Learning from Traditional Control Theory

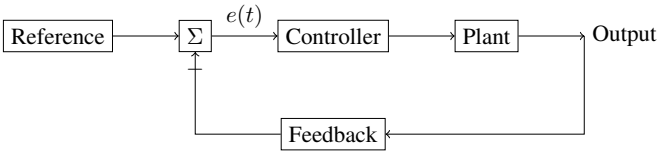

Fig. 2. Traditional control theory block diagram

Reinforcement Learning (RL) fundamentally diverges from conventional control theory paradigms in its approach to control and optimization. In the context of RL, the *Agent* encapsulates the functionalities of both the controller and the feedback elements, as seen in traditional control systems.

One crucial distinction to underscore is that the *Plant* in control theory does not directly correlate to the *Environment* in RL. This highlights the model-free characteristic intrinsic to RL. Not being bound by the specifics of different plant configurations means that the same RL agent can potentially be applied to an array of different plant types, underscoring its versatility and adaptability.

B. Deep Deterministic Policy Gradient

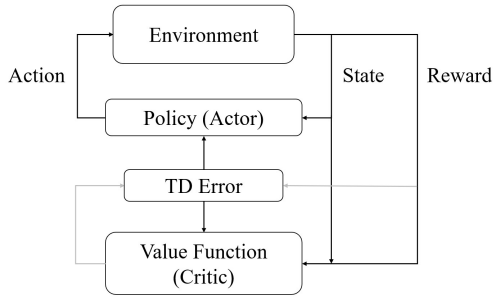

Fig. 3. DDPG Actor-Critic Model

In the present study, the chosen agent for reinforcement learning is the Deep Deterministic Policy Gradient (DDPG). DDPG is an off-policy algorithm and operates in a model-free environment, utilizing two primary components: the Actor and the Critic.

The *Actor* employs a neural network to compute the optimal action based on a given observation, aiming to maximize the cumulative reward over time. On the other hand, the *Critic* evaluates the action chosen by the Actor by computing the Q-value for that action in a particular state.

The interaction between the Actor and Critic is symbiotic. The Actor's decisions are informed by feedback from the Critic in the form of the Temporal Difference (TD) Error. This error is calculated as the difference between the predicted Q-value and the actual reward obtained. By using this TD Error, the Actor adjusts its policy to make better future decisions. This iterative feedback mechanism ensures the convergence of the system towards optimal policies.

Algorithm 1 DDPG Algorithm

Initialize actor π, critic Q, targets π', Q'.
Initialize replay buffer R.
for episode $= 1$ to M **do**
  Init noise $\mathcal{N}$, get state s_1.
  for $t = 1$ to T **do**
    $a_t = \pi(s_t) + \mathcal{N}_t$
    Execute a_t, get r_t, s_{t+1}.
    Store (s_t, a_t, r_t, s_{t+1}) in R.
    Sample mini-batch from R.
    Compute $y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}))$.
    Update Q by minimizing: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i))^2$.
    Update π using policy gradient.
    Update target networks.
  end for
end for

C. Reward Function and Environment Design

1) Reward Function Parameters: The following parameters define the state and actions of the robot:

```
v_y = 'Forward velocity';
p = 'Power consumption';
Delta_z = 'Vertical displacement';
Delta_x = 'Lateral displacement';
w_1 = 0.05;  % Encourages forward motion
w_2 = 1;     % Standard weighting factor
w_3 = 5e-4;  % Minimizes power consumption
w_4 = 10;    % Minimizes vertical displacement
w_5 = 10;    % Minimizes lateral displacement
```

The reward function r_{func} is given by:

$$r_{\text{func}} = w_1 v_y + w_2 t_s - w_3 p - w_4 \Delta z - w_5 \Delta x \qquad (1)$$

2) Stopping Criteria: The stopping criteria to ensure safety and efficiency are:

```
params.stoppingCriteria.heightChange = 0.5;
params.stoppingCriteria.lateral = 1;
params.stoppingCriteria.angle = 30;
params.stoppingCriteria.timeoutTime = 2;
params.stoppingCriteria.timeoutDistance = 1;
```

3) Environment Design: The Sidewalk: Parameters for the designed environment, considering real-world scenarios, are:

```
params.display.tileThickness = 0.005;
params.display.planeWidth = 1.5;
params.display.planeLength = 25;
params.display.planeHeight = 0.6;
```

If the robot deviates from the sidewalk by more than 0.5 m, the episode is terminated.

IV. Results

During the optimization process, the Episode Reward Graph demonstrated an increasing trend as the number of episodes progressed. High deviations in rewards were observed, attributable to the agent's selection of unknown actions. Such behavior aligns with the trial-and-error nature of RL.

TABLE I
Episode and Average Results

Metric	Value
Episode Number	2227
Episode Reward	846.6369
Episode Steps	571
Episode Q0	235.2353
Total Number of Steps	727529
Average Reward	590.704
Average Steps	554.65

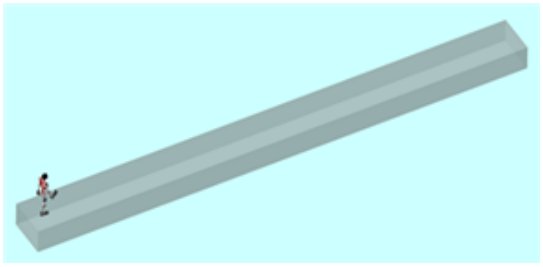

Fig. 4. Robot's trajectory in the MATLAB Simulink simulation.

In a simulated environment via MATLAB Simulink, the robot consistently achieved its target location without significant deviations, as illustrated in Figure 4.

V. Analysis & Implications

Through the research, several observations and deductions can be made:

1) **Fluctuations in Rewards**: Even post the learning phase, reward fluctuations persisted. This behavior, though expected in initial episodes, continuing into later episodes might indicate the agent's constant exploration or unforeseen environmental intricacies.
2) **Convergence to Local Maximum**: There's an evident risk that our agent might be converging to a local maximum instead of a global maximum. This could potentially mislead the optimization process. Augmenting the observation space, possibly through integrating more sensors, might alleviate this. However, this comes with its own set of challenges, predominantly being the increased computational overhead.

VI. Concluding Remarks

The application of DDPG for reinforcing humanoid robots to achieve optimized walking postures has showcased promising outcomes. Through a defined reward function, the robot, in a simulated environment, learned to walk efficiently, adhering to real-world sidewalk conditions.

Despite its successes, the potential for further improvement remains. The persistent fluctuations in rewards, even in later episodes, suggest a need for refined exploration strategies. One such strategy could be the incorporation of Genetic Algorithms (GAs). GAs, with their diverse exploration capabilities and ability to handle non-differentiable spaces, can complement traditional RL methods, helping the agent to avoid local optima and explore a broader solution space. Furthermore, the parallelization property of GAs can speed up the learning process, especially beneficial in computationally expensive environments. Also, ensuring convergence to a global maximum, rather than local peaks, is pivotal for the real-world deployment of such models.

Future work should look into enhancing the observation space, fine-tuning the algorithm, and perhaps integrating other RL strategies for better robustness and efficiency. Furthermore, conducting real-world tests will provide more insights into the practicality and reliability of the proposed approach.

References

[1] T. Haarnoja, et al., "Learning to Walk via Deep Reinforcement Learning," ArXiv abs/1812.11103, 2019.

[2] J. Kober, J. Bagnell, and Peters, "Reinforcement Learning in Robotics: A Survey," The International Journal of Robotics, 2013.

[3] T. P. Lillicrap, et al., "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2016.

[4] S. Fujimoto, H. V. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," arXiv preprint arXiv:1802.09477, 2018.

[5] G. Barth-Maron, et al., "Distributed Distributional Deterministic Policy Gradients," arXiv preprint arXiv:1804.08617, 2018.