International Journal of Computer Trends and Technology
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V69I7P103

Volume 69 Issue 7, 24-28, July 2021

Original Article

Locomotion Control Framework for Snake-like Robot using Deep Reinforcement Learning

Obe Olumide O[1] and Ayogu Thomas O[2]

[1,2]*Department of Computer Science, The Federal University of Technology, Akure, Nigeria*

Received Date: 30 May 2021
Revised Date: 03 July 2021
Accepted Date: 13 July 2021

Abstract *- In many industries today around the globe, robots can be seen carrying out different tasks. These robots have the capabilities to lift heavy loads, move at a very unbelievable speed, and execute tasks at a high level of pinpoint accuracy. But despite their amazing repertoire of tasks, most robots will find it very difficult to adapt themselves to new and environments that are unfamiliar to them. This could be because human environments are so dynamic and unpredictable and very difficult to be programmed, but rather must be learned firsthand by the robot. The desire to build machines that learn behavior based on the environment presented to them is one of the goals of Reinforcement Learning (RL). Reinforcement learning, an aspect of machine learning which is inspired by behavioral psychology, allows an agent – the learner and decision-maker, to automatically and autonomously discover optimal behavior through trial and error interactions with its environments in an attempt to solve problems. We present in this paper a control framework for Snake-like robot locomotion based on Deep Reinforcement Learning.*

Keywords *- Locomotion, Snake-like robot, Framework, Reinforcement learning, Controller.*

I. INTRODUCTION

A snake-like robot inspired by a biological snake is a class of hyper-redundant robots that have the potentials for meeting the ever-increasing need for robotic locomotion in a challenging environment and performing diverse tasks in a difficult and challenging environment such as underwater exploration tasks, industrial pipe inspection, firefighting, etc. The snake-like robot is made up of serially connected modules, and these modules can bend in one or more planes to generate locomotion gaits. The snake-like robot has many degrees of freedom, and this gives snake-like robots great potentials to navigate a wide range of environments by actively changing their overall shape, which surpasses the locomotion ability of more conventional robots with wheels, tracked robots, or robots with legs. However, this attributes of snake-like robot that makes them powerful also make designing locomotion control model for snake-like robot very difficult.

Snake-like robot achieves locomotion by changing their body shape which causes the body to interact with its environment and by so doing, propelling the robot in some direction. Common strategies for controlling snake-like robot locomotion include undulating the joints angle of the robot according to parameterized sine wave [2], central pattern generators [5], and follow-the-ladder controllers [1]. Although many successes have been recorded in the area of developing adaptive and complaint locomotion controllers for snake-like robot locomotion, it remains short of replicating the true versatile locomotion ability of biological snakes. This work, therefore, presents a control framework for Snake-like robot locomotion based on Deep Reinforcement Learning.

II. RELATED WORKS

Biological snakes can achieve diverse and different locomotion gaits by wiggling their bodies on rough or smooth terrains. To acquire similar locomotion gaits, most research work on snake-like robot have employed kinematic-based methods which simplifies the parametric representations of the snake-like robot trajectories. This method can be seen in the works of Hirose, 1993 [1], where the terpenoid curve was used to describe the locomotion properties of the snake. Main [3] modeled the muscle characteristics of snakes and developed a curve known as serpentine curve and used it to describe snake-like robot locomotion [3]. Tesch et al., 2009 in [6] described snake-like robot joint angles as a parameterized sinusoidal function. From the perspective of kinematic-based methods, researchers have also developed locomotion control models based on a central pattern generator (CPG) [7, 8]. Even though these methods have achieved significant results, the gait efficiency achieved by this method is limited to only tuning the parameters manually, which makes it time-consuming and inefficient.

Studies on designing optimized locomotion gaits for snake-like robots have also been carried out. In [9], multidimensional friction, known as evolutionary algorithms, was adopted to design efficient locomotion gait while [10, 11] studied and implemented policy gradient search

algorithms for optimized locomotion gait. However, these algorithms are associated with the problem of local optima, and because of this, the process can be very slow, inefficient, and manually intensive.

Reinforcement learning has also been explored for designing effective locomotion gaits for snake-like robots. Reinforcement learning is an intelligent trial and error learning method that is particularly useful for tasks in which it is easy to assess if a goal was reached, but the best ways to reach it are hard to determine. Reinforcement learning brings an entirely new solution for free gait generation tasks without having prior knowledge of the models. At the initial stage, reinforcement learning was not widely used in the robotics domain due to the fact that robots often require high dimensional, continuous states and actions [12]. However, with recent advancements in reinforcement learning algorithms, complicated tasks like locomotion gaits generation [13], dexterous manipulation [14], and autonomous driving [15] can be handled by robots. Schuhmann et al. in [16] used RL-based methods Proximal Policy Optimization (PPO) algorithm to generate locomotion gait for robots and equally used the same algorithm to learn energy-efficient gaits. Wu et al. proposed a novel triplet-average Deep Deterministic Policy Gradient (DDPG) algorithm while investigating the underestimation problem in Q-learning methods in order to reduce estimation bias. Their proposed method improved the performance of many robotic control tasks [17]. Cully et al. in [4] carried out two prototype experiments using reinforcement learning methods and showed in their results that Reinforcement Learning methods could help robots to recover from damage and quickly adapt as animals do. In their experiments, a robotic arm was able to learn to reach the given target with one or more stuck joints, and a hexapod robot was able to learn how to walk very fast and straight with broken or missing legs.

III. DESCRIPTION OF SNAKE-LIKE ROBOT MODEL

A case study of a snake-like robot model adapted from the Active Cord Mechanism (ACM) snake-like robot by Hirose [1] was used. The snake model is a planner snake-like robot consisting *of n* number of links with *l* lengths and interconnected by *n-1* joints. The kinematics of the snake-like robot used is defined according to the symbols in Figure 1. The *n* links have the same mass *m* and moment of inertia J, with each link having a mass that is uniformly distributed, and as such, the link Center of Mass (CM) is located at its center point. The snake-like robot moves in a horizontal plane and has n+*2* degrees of freedom. The position of the Center of Mass of the robot is denoted by

$$p = [p_x p_y]^T \in R^2 \quad (1)$$

The absolute angle represented as θ_i of link *I* have expressed with respect to the global x axis with positive counterclockwise direction. The relative angle between link *i* and link *i* + 1 (i.e., the angle of joint *i*) is defined as

$$\theta_i = \theta_i - \theta_{i+1} \quad (2)$$

The Center of Mass of the links is subjected to a Coulomb ground friction force, having anisotropic friction coefficients μ_t and μ_n, which describes the Coulomb friction force in the tangential and normal direction of the links, respectively. With the snake-like robot, it is common to assume that $\mu_t > \mu_n$, which is also a property found in biological snakes [1].

The motion equation of the snake-like robot in terms of the joint angle $\theta \epsilon R^{n-1}$, absolute angle of the robot head link $\theta_N \epsilon R$, and the Center of Mass p position is expressed as equation (3).

$$\acute{\emptyset} = u, \acute{\theta_N} = g\left(\emptyset, \theta_n, \acute{\emptyset}, \acute{\theta_N}, \acute{p_x}, \acute{p_y}, u\right),$$
$$N\acute{mp_x} = \sum_{i=1}^{N} f_{x,i}, N\acute{mp_x} = \sum_{i=1}^{N} f_{y,i} \quad (3)$$

Where $u \in R^{N-1}$ is a transformed control input that corresponds to the acceleration of the joints angle, $g\left(\emptyset, \theta_n, \acute{\emptyset}, \acute{\theta_N}, \acute{p_x}, \acute{p_y}, u\right) \in R$ is the nonlinear function of the state vector, and the control input, *fx, i* and *fy, i* are the Coulomb friction force components on the link *i* in the global *x* and *y* direction respectively.

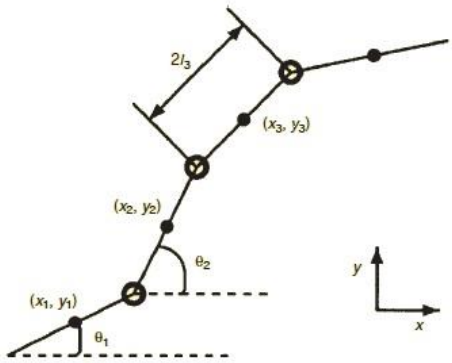

Fig. 1 Kinematics definition of Snake-like Robot model

IV. PROPOSED RL-BASED MOTION FRAMEWORK

A. Reinforcement Learning Setup

To solve a Reinforcement Learning problem, it is critical to segregate the agent from its environment and define the actions, states, and rewards. In the proposed RL-based motion controller (Figure II), the Agent is the entity that controls the snake-like robot. It implements the learning algorithm, takes actions, perceives the state, and receives the

rewards. The Environment consists of the snake robot, the target object, and other obstacles. The environment reacts to the actions taken by the agent.

B. The State (observation) Space

The Reinforcement Learning (RL) Agent receives information from the environment through the state (observation) space at each time step. Choosing the right state space is a very critical task in Reinforcement Learning as agents require accurate information to be able to learn its behavior. The state-space used in this work is made up of images from the vision sensor and the angles of the robot joints, as well as the target angles that each joint is currently rotating towards and the current speed of the head module.

The state space S_i^t is given in Table 1. The vision sensor has an angle of 90° and a resolution of 32 x 32 pixels. For the snake-like robot to learn locomotion, it requires the joints position $\emptyset$ and the angular joints velocity $\dot{\emptyset}$. The head link velocity v_1 is used to sense the global velocity, which offers the snake-like robot better movement awareness. For the snake-like robot to be able to locomote and move forward, actuated joints τ are utilized. To control the velocity of the robot, a target velocity v_t is specified and passed to the environment, which can be changed dynamically. Therefore, the state space (observation) size used in this work is 29-DOF.

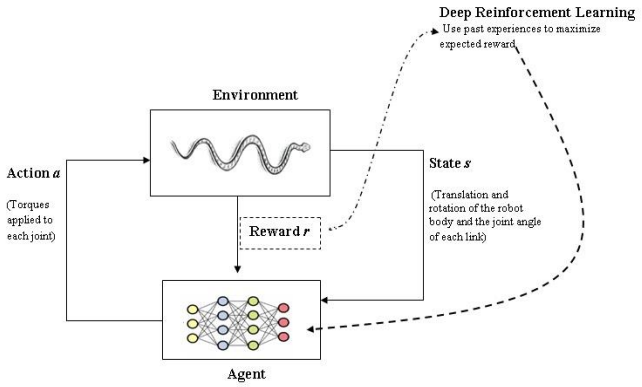

Fig. 2 Snake-like Robot Motion Control Model

Table 1. State (observation) Space for the RL-Motion Controller

Parameter	Description
$\emptyset_{1-9}$	Joint angle position
$\dot{\emptyset}_{1-9}$	Joint angle velocity
v_1	Head module velocity
τ_{1-9}	Actuator torque
v_t	Target velocity

C. Action Space

The snake-like robot is on a one-dimensional track, which is positioned between some obstacles. The goal of the robot is to move to a set target while learning a snake locomotion gait. Therefore, the action space is in a vector [snake robot position, snake robot velocity: moving forward or stop]. Since the snake-like robot is moving in a curve-like feature, the position is given by a continuous value [-1.2, 0.6], and the velocity is a bounded continuous value of [-0.07, 0.07]. Hence, the action space a_i^t of the RL-based motion controller corresponds to the nine joints positions of the snake robot, which linearly translates to a finite continuous value in the range of [-1.5, 15] to [-90o, 90o].

D. Reward Function

Reward function plays a very vital role in a reinforcement learning model. It is an immediate response sent back from the environment to evaluate the last action taken by the agent. In this work, the behavior that the reward function should incite in the agent is to move towards a target object. Therefore, it compares the location of the snake-like robot before and after a time step. It rewards any movement towards the right direction and penalizes movement towards the wrong direction. Figure 2 shows the schematic representation of the reward function used in this work.

Let p_x and p_y denote the location of the snake-like robot before the time step respectively in x and y coordinates, q_x and q_y the position after the time step in the respective x and y coordinates as well as $t_x t_y$ and the target coordinates before the time step respectively in x and y.

To obtain the distance the robot has moved over the course of a one-time step, the distance between the robot and the goal before the time step (equation 4) is compared to the distance between the robot and the goal after the time step (equation 5). During the time step, the robot moved, whereas the target position stayed at a constant position and is updated after calculating the reward. The reward signal, therefore, is equal to the difference between the distance before and after the time step (equation 6).

$$d_i = \sqrt{(t_x - p_x)^2 + (t_y - p_y)^2} \quad (4)$$

$$d_{i+1} = \sqrt{(t_x - q_x)^2 + (t_y - q_y)^2} \quad (5)$$

$$r_i = d_i - d_{i+1} \quad (6)$$

From (6), the agent gets a positive reward for any movement towards the goal, penalizes movement away from the target. However, movement away from the goal that would eventually result in a better position can also yield a higher overall reward. Because the discount factor γ is set to

0.99, expected future rewards are less important to the algorithm than immediate rewards but can still strongly influence its policy.

E. Network Architecture

To map the state space s_i^t (input) to the action space a_i^t (output), *MlpPolicy* from OpenAI Baselines will be used as the policy network. *MlpPolicy* configured as a fully connected 2-hidden layer neural network with the hidden layer size of 64 will be used as a non-linear function approximator to the policy π_θ. Both hidden layers have standard Rectified Linear Units (ReLu), and the final layers output the joint position commands for the robot. To train the network, the DDPG algorithm described in the next section will be used.

F. Training Algorithm

DDPG, a deep reinforcement learning algorithm proposed by Lillicrap et al. 2016 [18], will be used for training. Both Q-learning and policy gradient are combined in Deep Deterministic Policy Gradient frameworks and use neural networks as a function approximator. Q-learning is basically a method to learn using Bellman Equation (equation 7).

$$Q(s,a) = Q(s,a) + \alpha[(s,a) + \gamma maxQ'(s',a') - Q(s,a)] \quad (7)$$

where α = learning rate, γ = discount factor, a = action, s = state

DDPG maintains actor and critic networks (Fig. 4). The actor, which is represented as $\mu(S/\theta^\mu)$maps, states actions whereθ^μ represents the network parameters for the actor-network. The critic network represented as $Q(S/\theta^Q)$ outputs the value of the action under the state whereθ^Q represents network parameters for the critic network. An experience replay buffer is used by the DDPG framework to store transitions and to update the model. A target actor-network $Q'\mu'$ and are created by copying the actor and critic networks respectively so that a consistent temporal difference backup is provided.

An agent takes action $a_t S_t$on and then receives a reward based onS_{t+1}, the transition (S_t, a_t, S_{t+1}, r_t) is then stored in a replay buffer R. N sample transitions are drawn from R, and expected return is calculated by equation 8, and the critic network is then updated by minimizing the loss function$L(\theta^Q)$ between outputs of the target critic network Q' and the critic network Q (equation 9).

$$y_i = r_i + \gamma Q'(S_i + 1, \mu'(S_{i+1} | \theta^{\mu'}) | \theta^{Q'} \quad (8)$$

$$L(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q(S_i, a_i | \theta^Q))^2 \quad (9)$$

The actor-network is updated by using a sampled policy gradient (equation 10), and the target actor-network and the target critic network are updated as in Equations 11 and 12.

$$\nabla\theta^\mu J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a | \theta^Q) | S = S_i, a = \mu(S_i) \nabla\theta^\mu \mu(s|\theta^\mu)|s_i \quad (10)$$

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'} \quad (11)$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'} \quad (12)$$

Where τ denotes learning rate.

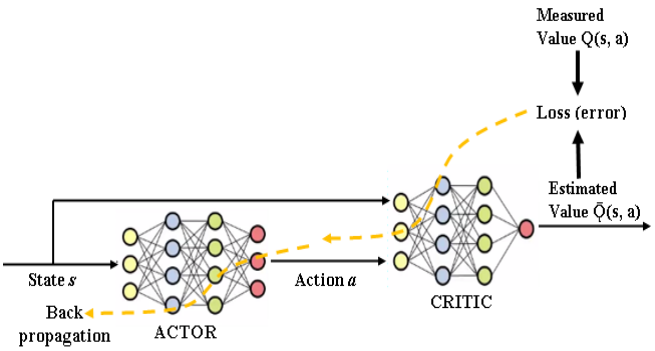

Fig. 3 Actor-Critic Model (DDPG Algorithm)

V. PROPOSED SYSTEM ARCHITECTURE

The implementation and evaluation of this work will be based on simulations. A robotic simulator V-REP by Coppelia Robotics will be used. V-REP can be used as a simulator as well as a means for scene creation, and it provides different choices of the physics engine to choose from. A robotic scene in V-REP consists of different objects that can be controlled individually with the use of embedded Lua scripts. The script connected to an object is called a child script. The snake-like robot is controlled partially with embedded Lua child script in V-REP. In addition to providing some control directly, the Lua script also performs message exchange with Reinforcement Learning (RL) algorithm. The RL algorithm runs in python and controls the robot via the V-REP RemoteAPI, through which it also receives the environment states.

Each scene in V-REP has an environment file written in Python that is needed to control dynamic factors and as well as serve as an interface between the RL Algorithm and the software used for simulation. Python was used to implement all the interfaces and will be registered as the environment in OpenAI Gym. The gym will serve as an interface and abstraction layer between the Reinforcement Learning agent and its simulation environment. The agent will be able to call methods such as step (action) on a gym environment regardless of the implementation details of the environment.

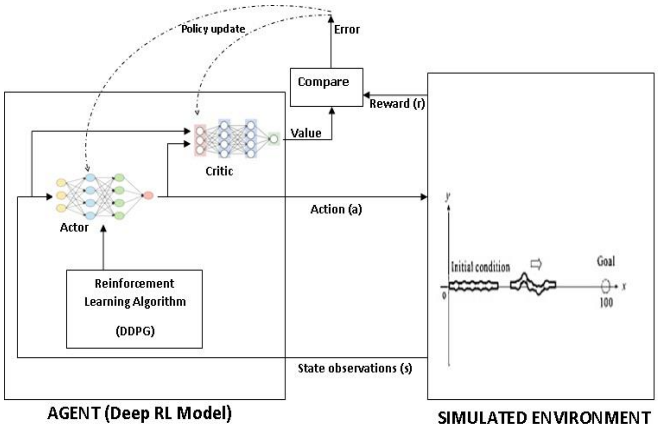

Fig. 4 Snake-like robot motion control system architecture

VI. CONCLUSION

Developing a locomotion control model for a snake-like robot remains a difficult and very challenging task. This is due to the fact that snake-like robots naturally come with redundant degrees of freedom and have very complicated interactions with their environments. The essence of this research work is to develop a locomotion control framework for snake-like robots based on deep reinforcement learning. The developed framework will contribute immensely to the ever-growing quest to build a more sophisticated and complex model for snake-like robot locomotion in real and challenging environments. Our future work will be to implement the framework and evaluate the performance of the model in a simulation environment and a real snake-like robot.

REFERENCES

[1] S. Hirose., Biologically inspired robots: snake-like robot locomotors and manipulators, Oxford University Press, Oxford., 1093 (1993).

[2] H. Ohno and S. Hirose., Design of slim slime robot and its gait of locomotion. Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems.,(2001) 707-715.

[3] S. Ma., Analysis of snake movement forms for the realization of snake-like robots. In Proc. IEEE Int. Conf. Robotics and Automation, Detroit, MIUSA., (1999) 3007–3013.

[4] A. Cully, J. Chune, D. Tarapire, J. B. Mouret (2015). Robots that can adapt like animals, Nature 521(7553) 503.

[5] A. J. Ijspeert., Central Pattern Generators for locomotion control in animals and robots: a review. Neural Networks, 21 (2008) 642-649.

[6] M. Tesch, J. Schneider, and H. Choset., Using response surface and expected improvement to optimize snake robot gait parameters. In 2001 IEE/RSJ International Conference on Intelligent Robots and Systems., (2011) 1069-1074.

[7] N. M. Nor, S. Ma., Smooth transition for CPG-based body shape control of a snake-like robot. Bioinspiration & biomimetics 9(1) (2013) 016003.

[8] Z. Bing, L. Cheng, G. Chen, F. Rohrbein, K. Huang, A. Knoll., Towards autonomous locomotion. CPG-based control of smooth 3D slithering gait transition of a snake-like robot. Bioinspiration and Biomimetics. 12(3) (2017) 035001.

[9] S. Chernova, M. Veloso., An evolutionary approach to gait learning for four-legged robots. IEEE/RSJ International Conference on Intelligent Robots and Systems., 3(2004) 2562-2567.

[10] M. S. Kim, W. Uther., Automatic gait optimization for quadruped robots. Australian Conference on Robotics and Automation. Citeseer. (2003) 1-3.

[11] N. Kohl, P. Stone., Machine learning for fast quadrupedal locomotion. AAAI. 4(2004) 611-616.

[12] J. Kober, J. A. Bagnell, J. Peters., Reinforcement Learning in Robotics: A survey. International Journal of Robotics Research. 32(11) (2013) 1238 – 1274.

[13] X. B. Peng, G. Berseth, K. Yin, M. Van De Panne., Deeploco: Dynamic Locomotion Skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics. 36(4) (2017) 41.

[14] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, S. Levine., Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. arXiv Preprint 1709.10087., (2017).

[15] P. Long, T. Final, X. Liao, W. Liu, H. Zhang, J. Pan., Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. IEEE International Conference on Robotics and Automation (ICRA)., (2018) 6252-6259.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimor (2017). Proximal Policy Optimization Algorithms. arXiv: 1707.06347.

[17] D. Wu, X. Dong, J. Shen, S. C. H. Hoi., Reducing estimation bias via triplet-average deep deterministic policy gradient. IEEE Transactions on Neural Networks and Learning Systems. (2020) 1-1.

[18] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Zrez, Y. Yassa, D. Silver, D. Wierstra (2015). Continuous control with deep reinforcement learning. CoRR abs/1509.02971.