Received July 15, 2020, accepted August 5, 2020, date of publication August 10, 2020, date of current version August 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3015506

Walking Control of a Biped Robot on Static and Rotating Platforms Based on Hybrid Reinforcement Learning

AO XI AND CHAO CHEN
Laboratory of Motion Generation and Analysis, Faculty of Engineering, Monash University, Clayton, VIC 3800, Australia
Corresponding author: Chao Chen (chao.chen@monash.edu)

This work was supported by the Australian Research Council under the scheme of LP160101192.

ABSTRACT In this paper, we proposed a novel Hybrid Reinforcement Learning framework to maintain the stability of a biped robot (NAO) while it is walking on static and dynamic platforms. The reinforcement learning framework consists of the Model-based off-line Estimator, the Actor Network Pre-training scheme, and the Mode-free on-line optimizer. We proposed the Hierarchical Gaussian Processes as the Mode-based Estimator to predict a rough model of the system and to obtain the initial control input. Then, the initial control input is employed to pre-train the Actor Network by using the initial control input. Finally, a model-free optimizer based on Deep Deterministic Policy Gradient framework is introduced to fine tune the Actor Network and to generate the best actions. The proposed reinforcement learning framework not only successfully avoids the distribution mismatch problem while combining model-based scheme with model-free structure, but also improves the sample efficiency for the on-line learning procedure. Simulation results show that the proposed Hybrid Reinforcement Learning mechanism enables the NAO robot to maintain balance while walking on static and dynamic platforms. The robustness of the learned controllers in adapting to platforms with different angles, different magnitudes, and different frequencies is tested.

INDEX TERMS Biped robot, reinforcement learning, Gaussian processes, deep deterministic policy gradient.

I. INTRODUCTION

The biped robot is a unique robotic system that has two legs, and it is able to perform static and dynamic postures or walking gait as humans. Biped robots also represent an interesting research topic that involves the dynamics and kinematics, mechanical design, control techniques, walking gait generation, stability, adaptability, rehabilitation, and the recent integration with artificial intelligence. Due to the kinematic complexity that the biped robots have more than twenty degrees of freedom, the control issue has always been the most complicated and challenging task for researchers in the fields of both robotics and artificial intelligence [1], [2].

Many approaches for biped walking based on conventional control theories rely on the mathematical model (the Inverted Pendulum Model) of the robot as well as the trajectory of Zero moment Point (ZMP) or Centre of Pressure (CoP) [3]. These approaches are commonly used for walking gait generation on flat surfaces [4]–[6]. Other implementations based on conventional approaches focus on biped walking control on uneven surface, which involves inclined surface [7], partially inclined surface [8], and grass, snow, sand, burning brush [9]. The main drawback of conventional control methods is that they highly rely on the accuracy of the mathematical model, where the model can be affected by joint friction, contact force with the ground, and other un-measurable uncertainties. Normally, the uneven surface such as the inclined surface, grass or snow ground need to be modelled independently, where, most of the time, the modelling procedure itself is computationally demanding and the obtained model may not be accurate enough. Besides, the conventional methods normally involve many parameters such as the weight of the robot, the position of the Centre of Mass (CoM) for each link, the inertial, the motor speed, etc. Therefore, obtaining the dynamic model using conventional approaches is highly computationally demanding and time consuming.

The associate editor coordinating the review of this manuscript and approving it for publication was Zeyang Xia.

The designed controller for a specific environment is also not general enough for it to adapt to other different types of environment. In other words, the controller designed for the robot to walk on the flat surface is not applicable for the walking control on inclined surfaces, sand surfaces or other types of environment. As a conclusion, attempting to generalize these controllers which enable most biped robots to be flexible and adaptable to different terrains still remains difficult for researchers.

Reinforcement Learning (RL) based control strategy is an alternative way that has been proposed recently to address the above mentioned problems for biped walking [10]. RL is an efficient approach that directly employs the interactions between the robots and the environment, where the robots can complete the desired tasks by gathering experience directly from its environment without computing the mathematical model of the robot or the environment [11]. Although the integration between RL and biped robots is still at their early stage, the huge potentials of RL have been proved to have a promising future in biped robots control applications. In 2010, Stulp *et al.* [12] applied Policy improvement with Path Integrals, which is a model-free RL approach, to control a biped robot to complete some simple tasks, such as passing through a way-point or pushing open a door. The simulation results showed that the proposed algorithm was able to efficiently learn humanoid motor skills to complete tasks. In 2017, Phaniteja *et al.* [13], employed Deep Deterministic Policy Gradient (DDPG) to learn a robust Inverse Kinematic (IK) solver and obtain stable IK solutions. The robot was able to complete the tasks involving reaching a point in the far right, in the left-back side, and blew its knee, respectively. Navarro-Guerrero *et al.* [14] proposed a supervised RL approach that combined supervised learning (Neural Networks) with RL. Experimental results showed that the algorithm was able to be applied on a physical robot to complete the tasks of docking and grasping. Although these papers showed some successful implementation of RL in biped robot control tasks, none of them were applied to biped walking control. Biped walking is much more complicated and requires high dimensional state space and action space, as it involves the precise control of every joint while maintaining the stability.

Many attempts based on model-free RL frameworks have been made recently to involve RL into biped robot walking control to avoid calculating the mathematical model. Gil *et al.* [15] utilized Q-Learning to find a sequence of pose that allows a NAO robot to reach the furthest distance in the shortest time, while still keeping a straight path without falling down. However, the actions were discrete, thus it lacked a smooth transfer between two poses. Lin *et al.* [16] successfully applied Q-Learning, a model-free RL framework, to a NAO robot as the joint controller to maintain the stability of walking on flat surface, inclined and declined surface, and a seesaw. They also proved that the proposed Q-learning scheme was able to imitate human motions captured from cameras while still maintaining the balance without falling down [17]. Although continuous actions were considered in their work by using Gaussian distribution, the Q-Learning framework still relied on a look up table where the action pairs were pre-defined. In 2019, Kim *et al.* [18] utilized Deep Q-Learning (DQN) and the Inverted Pendulum Model (IPM) to control the robot joints to complete the push recovery, where DQN was able to deal with a huge number of states without generating a look-up table or clusters. Although the steady state performance can be guaranteed after convergence using mode-free RL, the major challenge of it is that the convergence time is extremely long since no prior knowledge of the system is provided.

Model-based RL is an alternative approach to reduce the convergence time by generating a transition model and provides information about the dynamics of the system [19]. Deisenroth and Rasmussen [20], Deisenroth *et al.* [21] employed Gaussian Processes, known as the most commonly used mode-based RL approach, and proposed the Probabilistic Inference for Learning Control (PILCO) to estimate the transition model of the system, where the control policies are improved using the analytical result of the policy gradient based on a small amount of training data. Compared with model-free RL, model-based RL relies on the transition model leading to a shorter convergence time with less training data, but it highly relies on the accuracy of the estimated model. The resulting control policy would be changed significantly if the transition model is not learned perfectly or it contains measurement error. Besides, updating the model is extremely computationally consuming.

Many attempts focusing on combining model-based training framework with model-free learning techniques on robotic control tasks have been made in recent years. In 2016, Gu *et al.* [22], proposed a novel hybrid structure that incorporated a particular type of learned model into the proposed Q-learning scheme with Normalized Advantage Functions to improve the sample efficiency. The authors used a mixture of planned iLQG and on-policy trajectories, and then generated additional synthetic on-policy rollouts using the learned model from each state visited along the real-world rollouts. The effectiveness and data-efficiency are guaranteed by refitting the model itself for every n episodes. In 2018, Pong *et al.* [23] introduced a novel Temporal Difference model (TDM) by combining the benefits of model-free and model-based RL frameworks. The proposed algorithm was used for model-based control tasks where it was trained with mode-free learning mechanism. Nagabandi *et al.* [24] employed a HRL framework that combined Model Predictive Control (MPC) with NN Dynamic Models to achieve excellent sample complexity in a model-based RL algorithm. Then, a NN dynamics model was applied to initialize the model free learner. The results showed that the algorithm guaranteed the sample efficiency as well as the control performance. Feinberg *et al.* [25], however, proposed a model-based value expansion method where the predictive system dynamics model was incorporated into the model-free function estimation to reduce the sample complexity.

The predicted dynamics model was fitted prior to the sampling of a trajectory from the replay buffer, after which the actor network was updated. Then, an imagined future transition was obtained and was used to update the critic network. In 2019, Burhan Hafez *et al.* [26] proposed a Curious Meta-Controller which consists of the model-based planner and the model-free learner, where it was capable of alternating adaptively between model-based and model-free control. The approach was validated to be able to improve the sample efficiency and to achieve the near-optimal performance.

However, these methods were only validated with simple fully-observed models (the walker or the swimmer) under ideal conditions (flat surface), where no disturbances or complex terrains were considered as part of the environment. They also required a huge amount of computations as both model-based RL and model- free RL were being updated within the learning iteration during the on-line learning procedure. Besides, none of the above frameworks considered the joint limitations, mechanical constraints, or different terrains. Thus, there is a large variation between the generated walking gait for biped robots (the walker) from their experiments and the real human walking gait. Therefore, their generated gaits are not suitable for complex biped models, such as NAO robots, to complete tasks such as stable walking on different terrains. Hence, a Hybrid Reinforcement Learning (HRL) framework that is suitable for a complex biped robot model (a NAO robot) is investigated in this work.

The proposed HRL framework incorporates the model-based scheme and the model-free structure. The approach is validated on a NAO robot model to maintain balance while walking on static and dynamic platforms. By introducing the pre-training procedure, the proposed control method successfully bridges the gap between model-based RL and model-free RL, where the transition from learning the model to obtaining the best action has been smoothly completed. It also avoids the distribution mismatch problem while integrating the model-based scheme into the model-free framework. Simulation results show that the proposed HRL framework is able to generate joint velocity control inputs to generate stable walking gait on both static and dynamic platforms without falling down at any time step. The robustness of the controllers has also been validated, where the robot is able to adapt to walking on the platforms with different angles, different magnitudes, and different frequencies.

This paper is organized as follows: Section II shows the simulation environment of the proposed NAO robot as well as the rotating platform. Then, we illustrate the calculation of centre of pressure (CoP), generation of walking gait cycle, and the desired CoP, after which the state space and action space of the system are defined accordingly. In Section III, the overview of the proposed HRL is presented, as well as the basic principle of GP and the corresponding optimization techniques. Then, we demonstrate the structure of the proposed Hierarchical Gaussian Processes (HGP) and briefly explain the function of each layer, after which we explain how the initial control input is obtained according to the HGP model. After that, we explain the necessity of applying the Pre-training procedure based on the previously generated control input. Finally, we present Deep Deterministic Policy Gradient (DDPG) as the model-free optimizer to improve the policy based on the pre-trained actions. Section IV describes how we set up the experiments, where all simulation results are provided. In section V we present conclusions and discuss possible extensions of the proposed HRL algorithm.

II. FORMULATION OF THE PROBLEM

As shown in Figure 1, the NAO robot stands on a platform, where the platform is able to rotate along $x - axis$ and $y - axis$. The platform is employed here to simulate the complex and dynamic environment, where oscillations are introduced to the system to imitate the real external disturbances. Compared with other studies where the robots are trained on a flat surface or a platform with fixed angle under the experimental environment [15]–[17], the proposed platform is able to provide a more complex and dynamic environment where the robot is capable of learning a more robust and efficient controller. Thus, the experiments in this paper will not only show the convergence of the learning procedure, but also involve the robustness of the learned controller to adapt to different complex and dynamic environments.

At the beginning of each experiment, a NAO robot will be initialized to its default standing posture. The actions of the upper body, including the torso, arms, hands, and head, are ignored as it can be considered as a point mass using the Inverted Pendulum Model (IPM) [27]. The control actions are provided by ankle joints, knee joints, and hip joints. As shown in Figure 1, the robot is at its initial position, where the red and blue frames are the world frame and the robot frame, respectively. P is the rotating platform which can rotate along $x - axis$ (roll) and $y - axis$ (pitch) in the world frame.

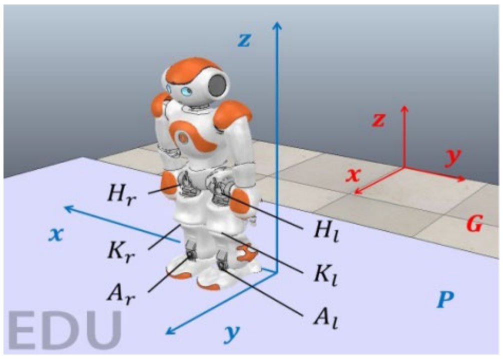


FIGURE 1. The simulation environment of the NAO robot with the platform.

H_R and H_L represent the right hip and the left hip, respectively. K_R (K_L) and A_R(A_L) indicate the knee joint and the ankle joint, respectively. The ankle joint has two Degree of Freedoms (DoFs) which can rotate along $x - axis$ (roll) and $y - axis$ (pitch). The knee joint has one DoF that allows it to rotate along $x - axis$. The hip joint has two DoFs that are pitch and roll, which allows it to rotate along $x-axis$ and $y-axis$, respectively. The actions applied on NAO at any time step, known as the action space, is a 10 dimensional array which can be expressed as

$$A = [\Delta q^p_{A_R}, \Delta q^p_{A_L}, \Delta q^r_{A_R}, \Delta q^r_{A_L}, \Delta q^p_{K_R}, \Delta q^p_{K_L}, \Delta q^p_{H_R}, \Delta q^p_{H_L}, \Delta q^r_{H_R}, \Delta q^r_{H_L}]^T$$

where the control inputs for hip joints are the change of pitch angles ($\Delta q^p_{H_R}, \Delta q^p_{H_L}$), and the change of roll angles ($\Delta q^r_{H_R}, \Delta q^r_{H_L}$), respectively. For knee joints, the control inputs are the change of pitch angles ($\Delta q^p_{K_R}, \Delta q^p_{K_L}$). For the ankle joints, the control inputs are the change of pitch angles ($\Delta q^p_{A_R}, \Delta q^p_{A_L}$), and the change of roll angles ($\Delta q^r_{A_R}, \Delta q^r_{A_L}$), respectively. In order to generate training data for the mode-based estimator, the action space is further specified as several discrete action combinations. For each dimension in **A**,the action is ranging from -0.1 to 0.1 and can be written as $\Delta q \in [-0.1, 0.1]$ which contains 11 evenly distributed discrete actions.

The walking gait for a biped robot is a periodic walking cycle, where the robot executes the same walking pattern from one cycle to another. The periodic gait can be further separated into the double support phase (DSP) and single support phase (SSP), where the instant transition between the DSP and SSP is ignored in this work. The simplified walking gait for one cycle is shown in Figure 2. As can be seen from Figure 2, the red dot indicates the desired CoP, and the black quadrilateral is the robot's foot. We assume that the biped robot starts this walking cycle from the DSP at the beginning, as shown in Figure 2 (a), where the left foot is in front of the right foot. Then it lifts its right foot (Figure 2 (b)) and takes a step forward (Figure 2 (c)), where the black dash quadrilateral indicates that the foot is lifted and is in the sky. The right foot contacts the ground when it completes a step forward, and at this moment, the desired CoP is shifted to the right foot immediately, as shown in Figure 2 (d). Then, the left foot takes the same step forward as the right foot, after which it contacts the ground, as shown in Figure 2 (g). The desired CoP is consequently shifted to the left foot at this moment. After the robot completes this walking cycle, it will execute the same cycle from the same beginning posture, where the posture shown in Figure 2 (g) is the same as that in Figure 2 (a).

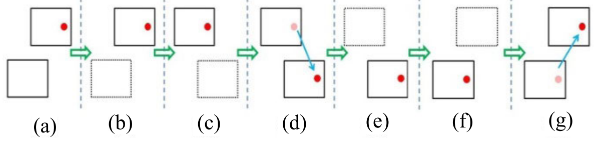


FIGURE 2. **Biped robot simplified walking gait for one cycle.**

In this paper, we utilize foot sensors to measure the centre of pressure (CoP) of the robot and determine whether it is stable. The detailed position of foot sensors and the calculation of CoP for DSP and SSP are shown in Figure 3.

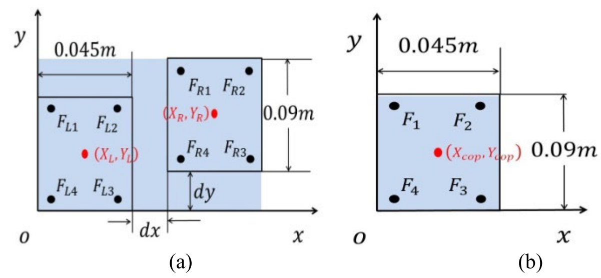


FIGURE 3. **The position of foot sensors and the center of pressure for (a) Double support phase, (b) Single support phase.**

As shown in Figure 3, the blue area indicates the support polygon (stable region), the black quadrilateral is the robot's foot, and the black and red dot show the position of foot sensors and the centre of pressure (CoP), respectively. For SSP, as shown in Figure 3 (b), the position of CoP in $x-axis$ and $y-axis$ can be calculated as follows

$$X^{L/R}_{CoP} = \frac{x_1F_1 + x_2F_2 + x_3F_3 + x_4F_4}{F_1 + F_2 + F_3 + F_4} \tag{1}$$

$$Y^{L/R}_{CoP} = \frac{y_1F_1 + y_2F_2 + y_3F_3 + y_4F_4}{F_1 + F_2 + F_3 + F_4} \tag{2}$$

where $x_i i \in [1, 4]$ and $y_i i \in [1, 4]$ indicate the x and y coordinates for the i^{th} sensor, respectively, and F_i is the sensor reading of the i^{th} sensor. The length and width of the foot are 0.09m and 0.045m, respectively. For DSP, as shown in Figure 3 (a), the position of CoP in $x-axis$ and $y-axis$ can be expressed as

$$X_{CoP} = X^L_{CoP} \times \left(\frac{F_L}{F_L+F_R}\right) + \left(X^R_{CoP}+d_x\right) \times \left(\frac{F_R}{F_L+F_R}\right) \tag{3}$$

$$Y_{CoP} = Y^L_{CoP} \times \left(\frac{F_L}{F_L+F_R}\right) + \left(Y^R_{CoP}+d_y\right) \times \left(\frac{F_R}{F_L+F_R}\right) \tag{4}$$

where F_L and F_R are the sum of sensor readings of the left and right foot, respectively. The superscript L and R in (3), (4) indicates the left foot and right foot, respectively. F_L and F_R are the sum of all four sensor readings for left and right foot, respectively. $[X^L_{CoP}, Y^L_{CoP}]$ and $[X^R_{CoP}, Y^R_{CoP}]$ are the position of CoP of the left and right foot, which can be calculated using (1) and (2). d_x is the offset of the foot in $x-axis$, which indicates the length of one step forward. d_y is the offset in $x-axis$, which represents the distance between the left and right feet. As shown in Figure 3 (a), both d_x and d_y will be calculated and refreshed at each step. The state space contains the position of CoP as well as the joint angles and can be presented as

$$\mathbf{S} = [X_{CoP}, Y_{CoP}, q^p_{A_R}, q^p_{A_L}, q^r_{A_R}, q^r_{A_L}, q^p_{K_R}, q^p_{K_L}, q^p_{H_R}, q^p_{H_L} q^r_{H_R}, q^r_{H_L}]^T$$

As this paper only focuses on the static stability walking control of the NAO robot for both SSP and DSP, the goal of the proposed RL algorithm is to make the measured CoP as close to the desired CoP ($X_{CoP}^{desired}$, $Y_{CoP}^{desired}$) as possible at any time steps under any environment. In other words, the objective of the proposed approach is to generate an adaptive walking gait based on the existing gait on the flat surface, which allows the NAO robot walking steadily on both static and dynamic platforms.

III. FRAMEWORK

A. THE STRUCTURE OF THE PROPOSED HYBRID REINFORCEMENT LEARNING

Many existing Hybrid RL structures attempt to integrate the model-based structure into model-free learning [22]–[26]. The model itself is being updated leading to a faster and more accurate value function estimation, where the sample efficiency is significantly improved. Inspired by their structure, we believe by reducing the dimension of action space with a reduced state space will also improve the sample efficiency with less computational load. In addition, avoiding the distribution mismatch problem [22]–[26] is much more straightforward with less computation than finding a method to address it. Thus, alternatively, a novel structure is proposed in this work which establishes a serial connection between model-based and model-free structures, where the distribution mismatch issue can be avoided. The model-based (MB) estimator is used to estimate the system model and obtain initial control inputs during the off-line training procedure. The model-free (MF) optimizer utilizes the control inputs as a reduced action space and fine tunes the action during the learning procedure.

As shown in Figure 4, at the beginning of off-line training, the predefined discrete actions are randomly applied to the robot to collect training data x_i. All training data will be stored as an array $\mathbb{D}_1$ where it contains N training tuples ($x_i \in \mathbb{D}_1$, $i = 1, 2, \cdots, N$). Each tuple includes three elements that $x_i = [s_i, a_i, s_{i+1}]^T$, where s_i and s_{i+1} represent the current and the successor state respectively, and a_i in the current random action. Then the proposed Hierarchical Gaussian Processes (HGP) is employed to predict the initial model of the system and generate initial control inputs. Detailed

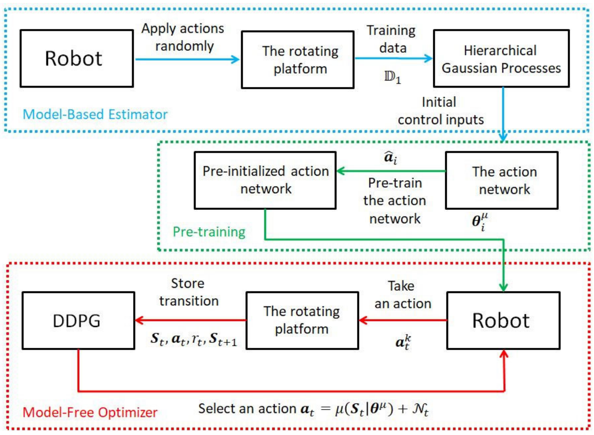


FIGURE 4. The structure of the proposed Hybrid Reinforcement Learning.

implementation of HGP is demonstrated in Section III C. The generated initial control inputs are essentially a reduced action space, where the elements in this space are action ranges. Every action range can be redefined as Gaussian distribution specified by mean μ_i^A and variance σ_i^A, where A indicates the action space, and i represents the corresponding state.

Before the model-based off-line training, the action space is defined as action ranges for all states, where the range is identical for each corresponding state. Assuming that the range for ith state can be defined as $a_i|s_i \in [-C, C]$, where the total number of actions is K. The goal of the proposed HGP is to significantly reduce the number of actions for each state, that is, to reduce the range of action space. The HGP will now generate a new action space for each state, i.e. the reduced action space, where it only contains k actions ranging from c to c ($k \ll K$ and $c \ll C$). Now, the action space for ith state can be written as $a_i|s_i \in [-c, c]$, which indicates that the actions within this range can better control the robot than the actions without this range. Suppose that the best action for ith state can be presented as a_i^b. Given that the objective of RL is to find a_i^b for ith state, and the searching space is reduced from $[-C, C]$ to $[-c, c]$. The sample efficiency is consequently improved, and thus, the convergence time is also reduced.

However, the parameters θ^μ for the actor network are randomly initialized [29], thus, it does not know that the actions from the reduced range $a_i|s_i \in [-c, c]$ can control the biped robot with better performance. As a result, the next step is to pre-train the actor network and to obtain a pre-initialized actor network parameterized by $\hat{\theta}^\mu$ that allows the model-free optimizer to start learning from a relatively stable state. The ground-truth for the pre-training procedure is defined as the mean μ_i^A for each state, and the parameters θ^μ are randomly initialized. The loss function is defined as the mean squared error (MSE) between the desired action μ_i^A and the current action. Consequently, the pre-trained network will be directly used as the initialized network for the Deep Deterministic Policy Gradient (DDPG) based model-free optimizer. DDPG is a model-free off-policy RL structure which is able to be applied to robotic systems with high dimensional continuous action space [29]. DDPG not only utilizes a critic network to approximate the value function but employs the actor network to generate action directly. Given that the action of the proposed NAO robot is the change of angle for each joint that contains the ankle joints, knee joints and hip joints. Also consider that the principle of the proposed HRL resembles a fine tuning scheme based on the original walking gait, where the robot is able to walk on different terrains. Thus, we believe that generating the policy directly through the deep neural network (Deterministic Policy) is much more efficient than employing gradient descent on Stochastic Policy. Many attempts based on Stochastic Policy gradient, such as Actor Critic and A3C [30], samples training data from both state and action spaces, leading to a low sampling efficiency and long convergence time. DDPG, on the other hand, samples training data only from the state space since the mapping between the state space and the action space is determined [29].

B. HIERARCHICAL GAUSSIAN PROCESSES

The Hierarchical Gaussian Processes (HGP) that consists of two layers, shown in **Algorithm 1**, is employed to estimate the dynamics of the system. The training data of the first layer can be written as $\mathbb{D}_1^j = \left\{s_t^j, a_j^j, s_{t+1}^j\right\}_{t\in\{1,2,\ldots,T\}}^{j\in\{1,2,\ldots,N\}}$. The input is $\mathbb{X}_1^j = \left\{s_t^j, a_j^j\right\}_{t\in\{1,2,\ldots,T\}}^{j\in\{1,2,\ldots,N\}}$, and the training target is $\mathbb{Y}_1^j = \left\{s_{CoP_{t+1}}^j\right\}_{t\in\{1:T\}}^{j\in\{1,2,\ldots,N\}}$, where j indicates jth action pair from the action space A. The state in $\mathbb{D}_1^j$ is specified by the current position of CoP and the current joint information, which can be written as

$$s_t^j = [X_{CoP}^j, Y_{CoP}^j, \boldsymbol{q}_A^{pj}, \boldsymbol{q}_A^{rj}, \boldsymbol{q}_K^{pj}, \boldsymbol{q}_H^{pj}, \boldsymbol{q}_H^{rj},]^T \tag{5}$$

Here, we use the bold letters to represent joint information as the robot has two legs. After obtained the estimated model, the training results of the first layer will be sued as the training inputs of the second layer. The training input of the second layer is $\mathbb{X}_2^i \in \mathbb{D}_2^i$ where $\mathbb{X}_2^i = \left\{s^i, a_j^i\right\}_{j\in\{1,2,\ldots,N\}}^{i\in\{1,2,\ldots,M\}}$ and the training target ($\mathbb{Y}_2^i \in \mathbb{D}_2^i$) is the successor state obtained from the first layer $\mathbb{Y}_2^i = \left\{s_{(t+1)j}^i\right\}_{j\in\{1,2,\ldots,N\}}^{i\in\{1,2,\ldots,M\}}$. Given the test input and the corresponding hyper-parameters, the estimated training outputs of HGP are the Gaussian Distributions of the successor states which can be calculated using Multiple Inputs Multiple Outputs Gaussian Processes in (6), as shown at the bottom of the page. The Detailed implementation of HGP can be found in our previous paper [33].

C. INITIAL CONTROL INPUT GENERATION

After obtaining the estimated model of the system, the initial control inputs will be consequently calculated by minimizing the predicted immediate cost, which can be presented by the following binary saturating function

$$\begin{aligned}\mathbb{E}\left[c\left(s_{t+1}\right)\right] &= \left|I + \Sigma\Lambda^{-1}\right|^{-\frac{1}{2\sigma_e^2}} \\ &\times exp\left(-\frac{1}{2\sigma_e^2}\left(\mu - s_{target}\right)^T \Omega^{-1}\left(\mu - s_{target}\right)\right)\end{aligned} \tag{7}$$

where Λ^{-1} is the diagonal precision matrix where the elements are in unity. $\Omega^{-1} = \Lambda^{-1}\left(\mathrm{I} + \Sigma\Lambda^{-1}\right)^{-1}$. As a result,

Algorithm 1 Implementation of HGP

```
1 :  **Initialize:** state space $\boldsymbol{S}$, and action space $\boldsymbol{A}$
2 :  Apply discrete actions randomly to the robot and collect
       data set $\mathbb{D}_1$
3 :  Using $\mathbb{D}_1$ to generate $\mathbb{D}_2$, and then obtain the transition
       model $\mathcal{M}$
4 :  Calculate the finial guesses $\boldsymbol{a}_{guess}^i$ by evaluating the
       predicted cost
5 :  **Repeat forever**
6 :    Obtain current state $\boldsymbol{s}_t$ from sensors' reading
7 :    Learn the transition model $\mathcal{M}$
8 :    Calculate the finial guesses $a_t$ according to $\mathcal{M}$
         Using $\boldsymbol{a}_t = \mathbb{E}\left[c\left(\boldsymbol{s}_{t+1}\right)\right]$
9 :    Apply $\boldsymbol{a}_t$ observe the next state $\boldsymbol{s}_{t+1}$, and receive an
         immediate reward $r_{t+1}$
10 :   Store $\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}$ as $\mathbb{D}_{new}$
11 :  $\mathbb{D}_2 = \mathbb{D}_2 \cup \mathbb{D}_{new}$
12 :  **until** $s$ is terminal
```

for any given state s_t after taking an action a_t, the expected immediate cost $\mathbb{E}\left[c\left(s_{t+1}\right)\right]$ can be computed using (7), where μ and Σ are the predicted mean and covariance of $p\left(s_{t+1}\right)$ respectively. Thus, the initial control inputs are defined as the actions for each state where the expected immediate costs are minimum and can be expressed as $a \sim \mathcal{N}(\mu^A, \sigma^A)$. The superscript of mean and variance with capital letter indicate that a is calculated from the original action space A. The initial control inputs can be generalized as an array, where each element represents a best action range corresponds to a certain state. Detained calculation of the initial control inputs can be found in our previous paper [33].

D. THE ACTOR NETWORK PRETRAINING

Gradient based model-free RL algorithms apply the calculation of policy gradient (Reinforce), value function gradient (DQN), or both (A3C) [30]. The principle of these methods is to utilize deep neural networks to obtain a mapping $\pi_\theta(a|s)$ between the input and output. The policy is being updated by relying on an estimated return by Monte-Carlo (MC) using episode samples to update the policy parameters θ. However, MC algorithm receives the reward and learns value functions until the end of an episode, which is time consuming. DQN and its variations on the other hand, generates a mapping between the input states and the Value Function, where the network is employed to estimate the function values with respect to each action. Although it performs well

$$\begin{aligned}&p(\boldsymbol{s}_{t+1}{}^j|\mathbb{X}_1, \mathbb{X}_{1*}, \mathbb{Y}_1) \\ &\sim N\left(\begin{bmatrix}\mathbb{E}_f\left[s_{1,t+1}^j|\mathbb{X}_1, \mathbb{X}_{1*}, \mathbb{Y}_1\right] \\ \vdots \\ \mathbb{E}_f\left[s_{L,t+1}^j|\mathbb{X}_1, \mathbb{X}_{1*}, \mathbb{Y}_1\right]\end{bmatrix}, \begin{bmatrix}var_f\left[s_{1,t+1}^j|\mathbb{X}_1, \mathbb{X}_{1*}, \mathbb{Y}_1\right]\cdots 0 \cdots & & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots var_f\left[s_{L,t+1}^j|\mathbb{X}_1, \mathbb{X}_{1*}, Y_1\right]\cdots 0 & \end{bmatrix}\right)\end{aligned} \tag{6}$$

with high dimensional state space, the estimated function values are only capable of evaluating discrete action space. A3C employs Actor Critic Structure that involves the critic network and the actor network. The critic network updates the Value Function parameters w based on Temporal Difference (TD) error, while the actor network updates the policy parameters θ for $\pi_\theta(\mathrm{a}|\mathrm{s})$, in the direction suggested by the critic network.

In the above-mentioned approaches, the policy function $\pi_\theta(\mathrm{a}|\mathrm{s})$ is typically modeled as a probability distribution over actions A with respect to the current state and thus it is a stochastic policy. The problem of stochastic policy gradient is that after the policy is learned, the probability distribution of optimal policy must be sampled to obtain the real action. The action is normally a high dimensional array, which means that frequent sampling is required in this high dimensional space leading to high computational load. As a result, during the process of learning, the policy gradient is calculated by integrating the entire action space. Alternatively, Deterministic Policy based algorithm considers the policy as a deterministic decision $\mathrm{a} = \mu(\mathrm{s})$ which does not require the integration in action space. Consequently, it improves the learning efficiency with less sampling procedures. However, performing gradient descent on deterministic policy usually starts from a randomly initialized actor network as the algorithm does not have prior knowledge of the system. On the other hand, updating the parameters that change the policy too much while using the deterministic policy gradient may induce training instability. Many approaches, such as TRPO [31] and PPO [32], that attempt to avoid this problem employ the KL Divergence that constrains the size of the policy update at each iteration. They improve the sample efficiency and thus improve the learning stability, but those approaches are still based on stochastic policy gradient. To bridge the gap between the mode-based HGP and the mode-free DDPG, we employ the Actor Network Pre-training scheme.

According to the previous sections, the HGP estimates the dynamic model of the system and also provides the initial control inputs. The control input is the possible actions for each that can better control the robot, where these actions can be stored as an array for each state. As a conclusion, the HGP essentially provides the exact prior knowledge of the system showing which actions can better control the robot and guarantee the stability, where the model-free optimizer is able to directly utilize it. We assume that the actor network is parameterized by θ^μ, and the output of it is the real actions and can be expressed as $\bar{\mathrm{a}} = \mu(\mathrm{s}|\theta^\mu)$. Since the initial control input only provides an action range and the exact best action for each state is still unknown, we extract the mean of the estimated actions for each state μ^{A} as the prior knowledge of the target actor network which can be written as $\hat{\mathrm{a}}$. The objective of pre-training is to train the actor network that minimizes the following Mean Squared Error (MSE) loss function

$$L(\bar{a},\hat{a}) = \frac{1}{N}\sum_{i=1}^{N}(\bar{a}_i - \hat{a}_i)^2 \tag{8}$$

The resulting actor network is able to generate actions for any given state where the robot is relatively stable. Thus, the model-free optimizer can perform gradient descent from a better beginning where the action is relatively close to the best action given the state. Consequently, the convergence time for the on-line learning procedure will be reduced.

E. MODEL FREE FINE TUNING

We employ DDPG as the model-free optimizer for the on-line learning procedure. DDPG is a model-free off-policy based on AC framework, which combines Deterministic Policy Gradient (DPG) with DQN [29]. DPG utilizes deterministic policy gradient to replace the importance sampling and thus avoid the mismatch between behavior and target policies introduced by stochastic policy gradient. DQN is an expansion of Q-Learning, where a deep neural network (Q-Network), together with experience replay, are used to replace the Q-Table and estimate the Q-Function [30]. The original DQN works in discrete action space, and DDPG extends it to continuous space to address the high dimensional action space. The objectives of DDPG are to learn a Q-Function, and to learn a policy. For the first objective, we assume that a deep neural network is used to approximate the Q-Function and can be expressed as $(\mathrm{s_i}, \mathrm{a_i}|\theta^{\mathrm{Q}})$, which is parameterized by θ^{Q}. The objective is to minimize the following Mean Squared Bellman Error (MSBE) loss function

$$\begin{aligned}&\mathcal{L}\left(\theta^{\mathrm{Q}},\mathcal{D}\right)\\&= \mathbb{E}_{(\mathrm{s,a,r,s',d})\sim\mathcal{D}}\\&\quad\times\left[\left(\mathrm{Q}\left(\mathrm{s},\mathrm{a}|\theta^{\mathrm{Q}}\right)-\left(\mathrm{r}+\gamma\left(1-\mathrm{d}\right)\max_{\mathrm{a'}}\mathrm{Q}\left(\mathrm{s'},\mathrm{a'}|\theta^{\mathrm{Q}}\right)\right)\right)^2\right]\end{aligned} \tag{9}$$

where $(\mathrm{s,a,r,s',d})$ is the collected transitions from D, and d indicates whether $\mathrm{s'}$ is terminal. The target term of this loss function is $\mathrm{r}+\gamma\,(1-\mathrm{d})\max_{\mathrm{a'}}\mathrm{Q}\left(\mathrm{s'},\mathrm{a'}\mid\theta^{\mathrm{Q}}\right)$, where the algorithm is trying to make the Q-Function be more like this target. However, it depends on the same parameters θ^{Q} and the learning process might be unstable when minimizing the loss function. Thus, DDPG introduces the target network to avoid this issue

$$\theta^{Q^{targ}} \leftarrow \rho\theta^{Q^{targ}} + (1-\rho)\,\theta^{Q} \tag{10}$$

where ρ is a hyperparameter ranging from 0 to 1. In DDPG, the target network is copied from the main network and is updated once per main network update $(\theta^{\mathrm{Q^{targ}}} \leftarrow \theta^{\mathrm{Q}})$. Thus, the MSBE loss function can be written as

$$\begin{aligned}&\mathcal{L}\left(\theta^{Q},\mathcal{D}\right)\\&= \mathbb{E}_{(s,a,r,s',d)\sim D}\\&\quad\times\left[\left(Q\left(s,a|\theta^{Q}\right)-\left(r+\gamma\left(1-d\right)Q^{targ}\left(s',\mu^{targ}|\theta^{Q^{targ}}\right)\right)\right)^2\right]\end{aligned} \tag{11}$$

where $\mathrm{Q^{targ}}$ represents the target Q-Network, and μ^{targ} is the target policy. For the second objective, DDPG is trying to learn a policy that maximizes the Q-Function $Q(s_i, a_i|\theta^Q)$. We assume that the actor function, parameterized by θ^μ can be written as $\mu(s|\theta^\mu)$. The objective is to perform gradient ascent with respect to θ^μ only to solve the following loss function

$$\mathcal{L}\left(\theta^\mu, \mathcal{D}\right) = \mathbb{E}_{s\sim\mathcal{D}}\left[Q(s_i, \mu(s|\theta^\mu)|\theta^Q)\right] \tag{12}$$

where the parameters in Q-Function are considered as constants in the above equation. The actor network is updated by introducing the target network and following the same principle in (10) as $\theta^{\mu^{\text{targ}}} \leftarrow \rho\theta^{\mu^{\text{targ}}} + (1-\rho)\,\theta^\mu$. Detailed implementation of the proposed HRL that combines HGP, pre-training, and DDPG is shown in **Algorithm 2**.

Algorithm 2 Hybrid Reinforcement Learning Algorithm

```
1 : Initialize: state space S, and action space A, randomly
initialized the critic network $Q(\boldsymbol{s}, \boldsymbol{a}|\boldsymbol{\theta}^Q)$ and the actor
network $\mu(\boldsymbol{s}|\boldsymbol{\theta}^\mu)$ with the weight $\boldsymbol{\theta}^Q$ and $\boldsymbol{\theta}^\mu$
2 : Apply Algorithm 1 to generate the initial control inputs
      $\boldsymbol{a}|\boldsymbol{s}$ with the mean $\boldsymbol{\mu}^{\boldsymbol{A}}$ and the variance $\boldsymbol{\sigma}^{\boldsymbol{A}}$
3 : Pre-train the actor network and obtain the pre-trained
    parameters $\hat{\boldsymbol{\theta}}^\mu$
4 : Initialize: the actor network $\mu(\boldsymbol{s}|\hat{\boldsymbol{\theta}}^\mu)$ with the weight $\hat{\theta}^\mu$
5 : Initialize target network $Q'$ and $\mu'$ with weights
      $\boldsymbol{\theta}^{Q^{targ}} \leftarrow \boldsymbol{\theta}^Q, \boldsymbol{\theta}^{\mu^{targ}} \leftarrow \hat{\theta}^\mu$
6:Initialize replay buffer $\mathcal{D}$
7: For $episode = 1, K$ do
8:   Reset the robot and the platform to their initial position
9:     For $t = 1, T$ do
10:      Select an action $\boldsymbol{a}_t = \mu\left(\boldsymbol{s}_t \,|\, \hat{\theta}^\mu\right) + \epsilon$, where $\epsilon \sim \mathcal{N}$
11 :       Execute action $\boldsymbol{a}_t$ in the environment
12:      Observe next state $\boldsymbol{s}_{t+1}$, reward $r_t$
13:      Store transition $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ in replay buffer $\mathcal{D}$
14:      If $\boldsymbol{s}_{t+1}$ is the terminal state
15:        Reset everything and back to step 8
16:      Else
17:        Randomly sample a minibatch of $\mathcal{N}$ transitions
18:        $(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}_{i+1})$ from $\mathcal{D}$
19:        Compute the targets as
20:    $r_i + \gamma Q^{targ}\left(\boldsymbol{s}_{i+1}, \mu^{targ}(\boldsymbol{s}_{i+1}|\boldsymbol{\theta}^{\mu^{targ}})|\boldsymbol{\theta}^{Q^{targ}}\right)$
21:        Update Q-function by one step of gradient
           descent
22:           $\nabla_{\boldsymbol{\theta}^Q} = \frac{1}{N}\sum_{\mathrm{i}}\left(y_i - Q\left(\boldsymbol{s}_i, \boldsymbol{a}_i|\boldsymbol{\theta}^Q\right)\right)^2$
23:        Update policy by one step of gradient ascent
24:    $\nabla_{\hat{\theta}^\mu} = \frac{1}{N}\sum_{\mathrm{i}}\nabla_{a_i}Q\left(\boldsymbol{s}_i, \boldsymbol{a}_i|\boldsymbol{\theta}^Q\right)|_{\boldsymbol{a}_i=\mu(\boldsymbol{s}_i)}\nabla_{\hat{\theta}^\mu}\mu\left(\boldsymbol{s}_i|\hat{\theta}^\mu\right)$
25:        Update target networks with
26:          $\boldsymbol{\theta}^{Q^{targ}} \leftarrow \rho\boldsymbol{\theta}^{Q^{targ}} + (1-\rho)\,\boldsymbol{\theta}^Q$
27:          $\boldsymbol{\theta}^{\mu^{targ}} \leftarrow \rho\boldsymbol{\theta}^{\mu^{targ}} + (1-\rho)\,\hat{\theta}^\mu$
28:      End If
29:    End For
19:  End For
```

IV. EXPERIMENT

A. EXPERIMENT SETUP

The experiment environment is built in V-REP, as shown in Figure 1, which contains a rotating platform and a NAO robot. We also assume that all sensor readings contain measurement noises at all times. The total time of off-line training procedure for collecting training samples for HGP is 800 episodes. During the process of training, if the robot falls, the training procedure will be paused and the robot as well as the platform will be reset to their initial position, after which the process will start again and continue collecting training data. After the estimated model is obtained, the algorithm will find the actions where the predicted immediate cost is minimal. The cost is defined as the Euler distance between the current CoP and the desired CoP. To avoid overshoot, we employ the generalized binary saturating function to evaluate this distance penalty, which can be expressed as

$$c = 1 - exp\left(-\frac{1}{2\sigma_e^2}\sqrt{\left(y_p - y_0\right)^2 + \left(x_p - x_0\right)^2}\right) \tag{13}$$

where (x_p, y_p) is the current position of CoP, and (x_0, y_0) is the desired position of CoP. The result of HGP model-based off-line training is the action range for each state, where the estimated immediate cost is minimal. After it, the mean of the previous obtained action range for each state will be selected as the training data for the pre-training procedure, where the actor network of the model-free on-line learning framework will be pre-trained. This process does not require the interaction between the robot and the environment, thus, it can be considered as a separated procedure between the off-line training and the on-line learning where the NAO robot is not involved. Finally, DDPG is employed as the model-free optimizer to find the best action pair for each state. The reward function is defined as

$$\mathrm{r} = \begin{cases} -10 & \text{FallsDown} \\ 10 & \text{StableRegion} \\ \exp\left(-\frac{1}{2}\,(\mathrm{s}-\mathrm{s}_0)^{\mathrm{T}}\,\Psi^{-1}\,(\mathrm{s}-\mathrm{s}_0)\right) & \text{Otherwise} \end{cases}$$

where Ψ is the diagonal weighting matrix with the elements of 0.5, where each element determines the impact of the corresponding state. s and s_0 indicate the current position of CoP and the desired CoP, respectively. Considering that the final goal of the algorithm is to maintain the robot balance where the CoP must be located within the desired region. The joint angles may vary from each state as the angle of the platform is changing; thus, joint angles are not considered as part of the desired state.

Instead of allowing the simulation to run forever, we employ the episodic tasks in our work. At the beginning of each episode, the robot and the platform will be reset to their initial positions. This is because the entire experiment, including the off-line training process and the on-line learning procedure, assumes that the Robot Frame (local frame) is translated from the World Frame without rotation, which means that the $x-axis$, $y-axis$, and $z-axis$ of the Robot Frame are pointing the same directions as the World Frame. However, during the experiment, some of the intense actions may result in large instant velocity of the robot joints, leading the robot to deviate from the original local frame. The resulting Robot Frame may rotate from the World Frame, which consequently introduces errors after applying the control inputs. Episodic tasks allow the robot to be able to recalibrate the consistency between the Robot Frame and the World Frame, which avoids the error and increases the efficiency of the algorithm. In addition, as the platform is designed with finite length and width, the robot may walk outside of the platform and falls to the ground. Therefore, during the process of model-free on-line learning, if the robot falls on the ground from the rotating platform, the simulation will be paused and the platform will be set to their initial position. For the robot, the posture will be set to the initial posture but the starting position will be set to a random position within the range of the platform. This is because if we set the platform, as well as the robot, to their initial position and posture, the robot will always start learning from the same point and the platform will always execute the same rotating behaviour. Consequently, the robot will only explore actions where the environment only varies within a very small range, and thus the learned controller will be much less robust. To enable the robot to explore more actions where it is able to walk in a more complex environment, we select a random position within the range of the platform as the new starting position of the robot.

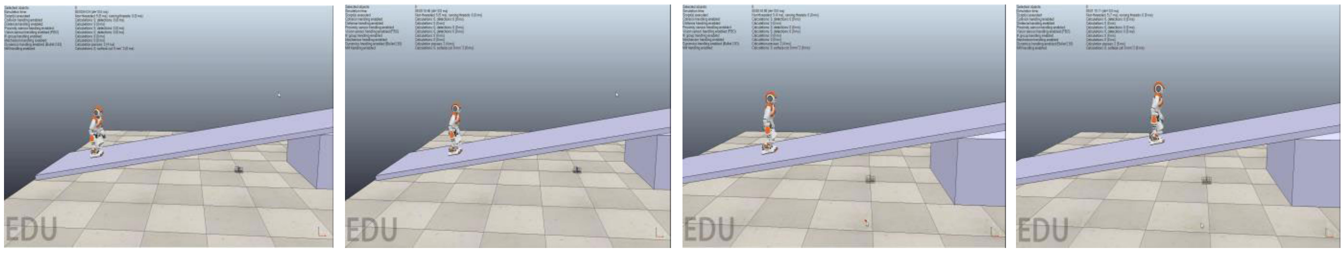
The benefit of employing simulation is that it can speed up the training and learning process, which will save a lot of time in reality. It is still hard to say that the system is exactly the same as the real one. One challenge is the difference between the simulation and the real platform. In the simulation, the dynamics of the platform is pre-defined and assumed not to be affected by the robot. However, in reality, the rotation may change slightly due to the contact force generated by the interaction between the robot and the platform. We will therefore consider the deviation between the simulated and the real platform in our future works.

B. EXPERIMENT RESULTS

We separate our experiment into three stages: uphill, downhill, and walking on the rotating platform. The on-line learning curve (i.e. the average error with respect to the learning episode), the real time position of CoP at the beginning and the end of learning, and the snapshots of the simulation are shown respectively for each experiment stage. For the uphill task, the on-line learning episode is 500. During the process of learning, if the robot falls on the ground from the platform, the simulation will be paused and the robot will be reset to its initial position, after which the simulation will continue.

Figure 5 shows 4 simulation snapshots of the NAO robot walking uphill on Episode 500 during the experiment on 4s, 18s, 43s, and 1min 15s, respectively. As can be seen from the figure, the robot walks uphill with steady steps without falling down. The robot is also able to walk in a straight line and no left or right deviation in walking occurs. Since we ignore the DSP and consider it as an instant transition between two SSPs, to show the stability of walking, the position of CoP for SSP is shown in our experiments. Since the stability of the robot is controlled by the model-free optimizer, the walking pattern may differ from one gait circle to another. The total time of completing each gait circle is consequently different. Thus, we collect 100 sampling points of CoP during the experiment for each foot to show the performance of walking before and after using the model-free optimizer. These sampling points may not be obtained from a single gait circle, and most of the time they are collected from two nearby gait circles.

Figure 8 (a) and (b) shows the position of CoP on the left and right feet, respectively. The blue circles indicate the position of CoP in Episode 1, which is the beginning of the model-free on-line learning procedure, and the red star markers are the position of CoP at the end of learning (Episode 500). The black dash lines indicate the desired CoP in $x-$ axis and $y-$ axis, which are $x = 0.0225$m and $y = 0.07$m, respectively. The more the sample points are close to the intersection of two dash lines, the more the robot is stable. For the left foot, shown in Figure 8(a), at the beginning for on-line learning, the sampling points are scattered ranging from 0.005m to 0.045m in $x-$ axis and 0.027m to 0.09m in $y-$ axis. The blue curve shows the CoP boundary of Episode 1. During the process of learning, the sampling points tend to gradually concentrate at the desired steady state $[0.0225m, 0.07m]$, and the average error between the measured CoP and the desired



(a) (b) (c) (d)

FIGURE 5. Simulation snapshots of the NAO robot walking uphill on a static platform, where the incline angle is 7 degrees.

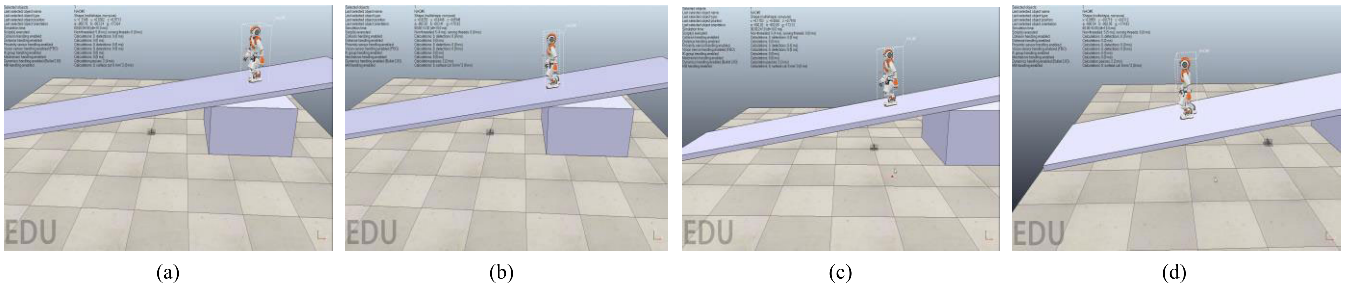


FIGURE 6. Simulation snapshots of the NAO robot walking downhill on a static platform, where the incline angle is 7 degrees.

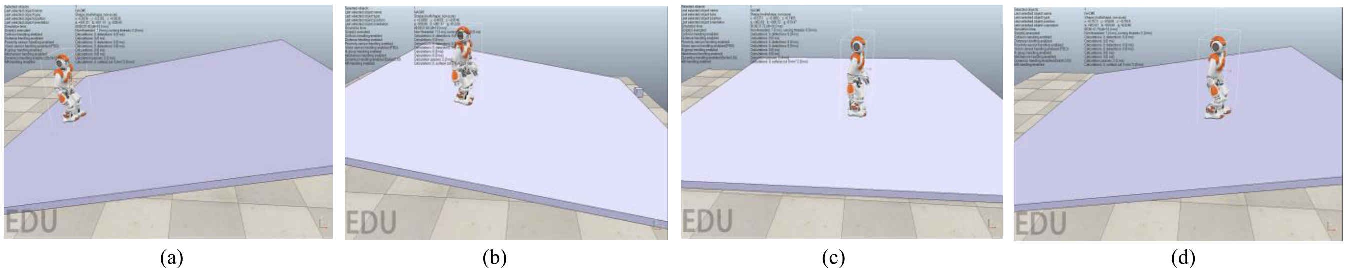


FIGURE 7. Simulation snapshots of the NAO robot walking on a dynamic platform with the frequency of 0.013Hz and the magnitude of 7 degrees.

CoP is consequently reduced. At the end of learning (episode 500), indicated by the red boundary, the sampling points are scattered ranging from 0.0109m to 0.0384m in $x-$axis and 0.058m to 0.084m in $y-$axis.

Figure 8 (b) shows the convergence of the sampling CoP of the right foot, where it has the same convergence pattern as the left foot.

At the beginning of learning, the CoP boundaries are $[0m, 0.041m]$ in $x-$axis and $[0.029m, 0.09m]$ in $y-$axis. At the end of learning, represented by the red boundary, the sampling points are converged to $[0.007m, 0.031m]$ in $x-$axis and $[0.055m, 0.084m]$ in $y-$axis. Figure 8 (c) shows the average errors from 50 individual trails with respect to learning episodes while applying the proposed HRL and applying HGP as the pure model-based RL, respectively. The error is defined as the Euler distance from the desired CoP to the measured CoP, which can be calculated using (13). The error at each episode is computed by averaging the values from 40 randomly initialized trails at the same episode.

The episode reward is not listed as it only shows the convergence of applying the RL algorithm. Here we show the error since it not only provides the convergence of the proposed model-free optimizer, but also indicates the control performance of the controller as the CoP error is the most significant criteria of stability for biped robots. As shown in Figure 8 (c), the red curve represents the average error while applying the proposed Hybrid RL framework which contains the mode-based estimator, actor network pre-training, and the mode-free optimizer. The blue curve shows the average error while applying the pure model-based RL without the model-free optimizer. The shaded area indicates the maximal

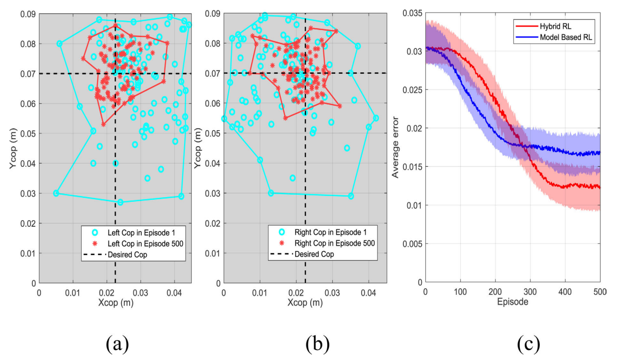


FIGURE 8. (a) 100 random sensor readings and boundaries of CoP at the left foot at episode 1 and 500 while the robot is walking upwnhill, (b) the sensor readings and boundaries of CoP at the right foot, (c) the average episode CoP errors of the proposed HRL compared with the model-based RL from 50 individual trails.

and minimal average error of 50 individual learning processes. At the beginning, the error of both frameworks is approximately 0.03m, with the increase of learning episode, the mode-based RL descents faster than HRL at the early stage of learning. The blue curve drops to 0.018m at Episode 260, after which it drops slightly to 0.017m at Episode 410 and does not change anymore. The red curve however, takes 330 Episodes to descend from 0.03m to 0.0125m, and the error is stable at 0.0125m after Episode 330. Although the convergence speed of mode-based RL is faster than HRL, due to the disturbance induced model uncertainty, the steady state error is larger than the proposed HRL.

Figure 6 shows 4 simulation snapshots of the NAO robot walking uphill on Episode 500 during the experiment on

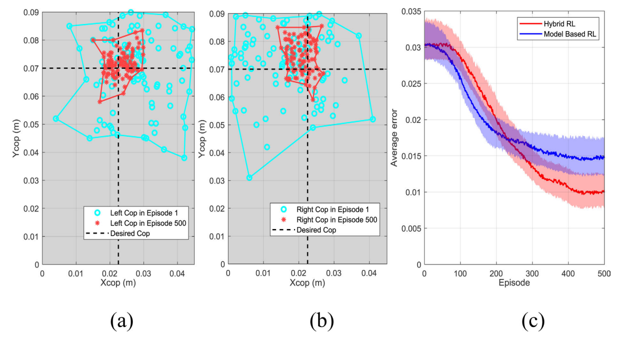


FIGURE 9. (a) 100 random sensor readings and boundaries of CoP at the left foot at episode 1 and 500 while the robot is walking downhill, (b) the sensor readings and boundaries of CoP at the right foot, (c) the average episode CoP errors of the proposed HRL compared with the model-based RL from 50 individual trails.

5s, 13s, 24s, and 43s, respectively. The robot is also able to walk in a straight line steadily without left or right deviation or falling down. Figure 9 (a) and (b) shows the position of CoP in Episode 1 and Episode 500 on left and right foot, respectively. At the beginning of on-line learning, the boundaries for the left foot in $x-$axis and $y-$axis are $[0.004m, 0.045m]$ and $[0.038m, 0.09m]$, respectively. The boundaries for the right foot in $x-$axis and $y-$axis are $[0.001m, 0.041m]$ and $[0.031m, 0.09m]$, respectively. At the end of learning, the boundaries for the left and right feet are : $[0.015m, 0.03m]\,y$: $[0.058m, 0.084m]$, and x : $[0.014m, 0.0265m]\,y$: $[0.059m, 0.085m]$, respectively. As can be seen from Figure 6 and as compared with Figure 8, walking downhill is steadier than walking uphill as the position of CoP after on-line learning is more concentrated. Also, comparing Figure 9 (c) with Figure 8 (c), the steady state average error of walking downhill is less than walking uphill. The convergence speed of pure mode-based RL, represented by the blue curve, is faster than HRL at the beginning, but the final error $(0.015m)$ is larger than HRL $(0.01m)$ after it reaches the steady state, which shows the same result as walking uphill.

Both of the positions of sampled CoP and the average error for walking downhill after employs the HRL model-free optimizer show better results than walking uphill. This is because the robot walks against the gravity while walking uphill, thus it is harder for the robot to maintain balance.

Figure 7 shows the snapshots of the third stage of the experiment, where the NAO robot is walking on the rotating platform while the platform is rotating along the y − axis. The snapshots are taken on Episode 1000 at 7s, 21s, 31s, and 41s, respectively. Figure 10 (a), (b) show the CoP position before and after the learning. At the beginning, the boundaries are the same as walking uphill and downhill as the initial control input is obtained from the same model-based estimation procedure. However, since the platform is rotating during the learning process, the robot is essentially interacting with a dynamic environment, the control task for the RL algorithm to maintain balance is more challenging. Thus, the CoP boundaries of the left and right feet on stage 3 is slightly larger than those of uphill and downhill, which are x : [0.013m, 0.039m], y : [0.048m, 0.086m], and x : [0.008m, 0.032m], y : [0.048m, 0.085m], respectively. Figure 10 (c) shows the average error with respect to learning episode, which shows almost the same convergence pattern as that in Figure 8 (c) and Figure 9 (c), respectively. The convergence speed for mode-based RL is faster at the beginning, but it slows down after Episode 410 and does not change any longer. The error of applying HRL is reducing gradually at the beginning, and it continues decreasing when the error of model-based no longer changes. The steady state error of HRL (0.0155m) is much less than that in the model-based RL (0.019m).

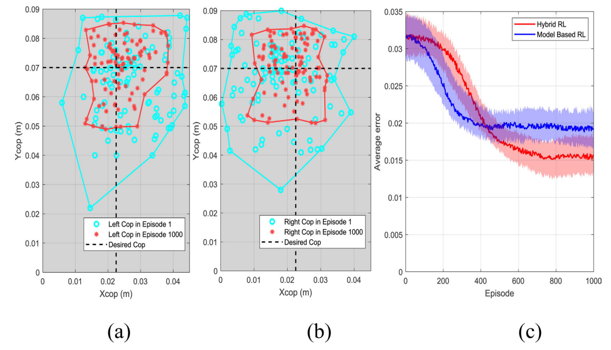


FIGURE 10. (a) 100 random sensor readings and boundaries of CoP at the left foot at episode 1 and 500 while the robot is walking on a dynamic platform, (b) the sensor readings and boundaries of CoP at the right foot, (c) the average episode CoP errors of the proposed HRL compared with the model-based RL from 50 individual trails.

Table 1 shows the detailed comparisons between the proposed HRL and pure model-based RL in terms of the on-line learning time, the average error, and the walking velocity. Although the model-based RL has faster convergence speed, the overall control performance is much worse than the proposed HRL framework.

TABLE 1. Comparisons Between HRL and model-based RL.

Case	On-line learning time	Average error	Walking velocity
Uphill (HRL)	330 episodes	0.017m	0.037m/s
Uphill (Model-based)	260 episodes	0.0125m	0.028m/s
Downhill (Model-based)	426 episodes	0.01m	0.069m/s
Downhill (Model-based)	373 episodes	0.015m	0.061m/s
Platform (HRL)	633 episodes	0.0155m	0.054m/s
Platform (Model-based)	400 episodes	0.019m	0.047m/s

Further experiments are conducted to test the robustness of the controller obtained by the proposed HRL framework. We utilize the learning controllers obtained from Stage 1 and Stage 2 and apply them to the robot, where the robot is walking on the platform with different slope angles. By gradually

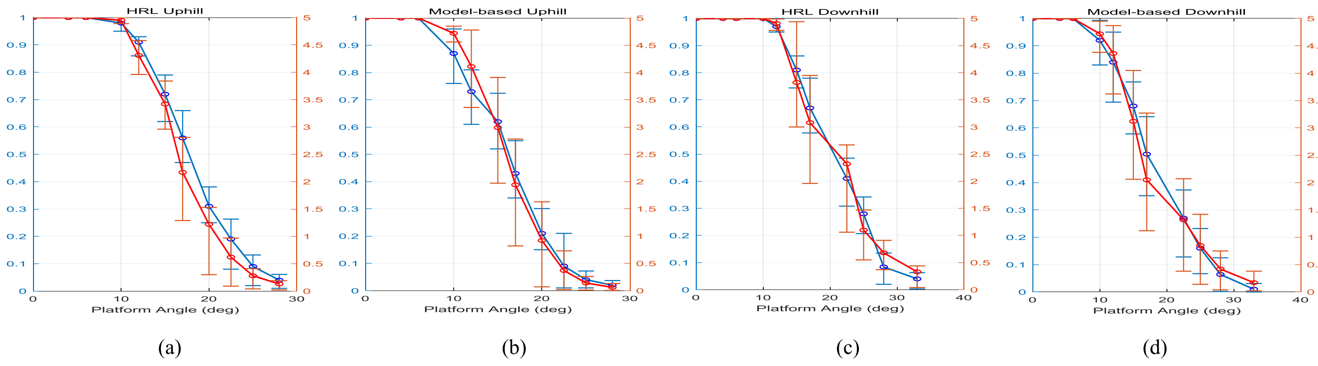


FIGURE 11. The x − axis is the angle of the static platform, the blue and red y − axises represent the Success Rate and the Maximal Walking Distance, (a): the Success Rate and Maximal Walking Distance of the proposed HRL while the robot is walking uphill, (b): the results of mode-based RL for uphill walking, (c): the results of the proposed HRL for downhill walking, (d): the results of mode-based RL for downhill walking.

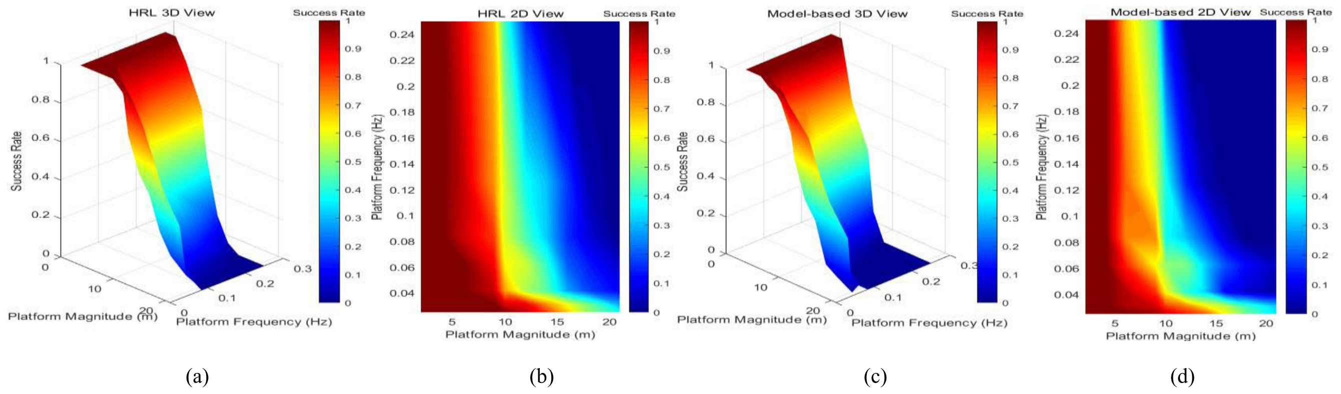


FIGURE 12. The x − axis and y − axis represent the magnitude and the frequency of the dynamic platform, respectively, the z − axis is the Success Rate, (a) (b): the Success Rate of the proposed HRL while the robot is walking on a dynamic platform, (c) (d): the Success Rate of the mode-based RL.

changing the slope angle, the robustness of the controller is tested by evaluating the maximal Walking Distance as well as the Success Rate. We fix platform angle before each experiment, and then apply the learning controllers from States 1 and 2 to uphill and downhill tasks, respectively. The robot tries to walk on the platform with a fixed slope angle 100 times. If the robot falls down, the simulation will be paused and the robot will be reset to the position where it falls down, after which the simulation will continue running and the robot will continue walking until it reaches to the end of the platform. The walking distance is defined as the distance between the start of walking to the position where the robot falls. Thus, the maximal distance for a certain slope angle is calculated by finding the maximal value within 100 rounds. The success rate is defined as the average walking distance divided by the length of the platform. We compared the robustness of two controllers obtained by HRL and model-based RL respectively from Stage 1 on a rotating platform to enable the robot walking uphill, where the slope angles of the platform are 0, 4, 6, 10, 12, 15, 17, 20, 22.5, 25, 28 degrees, respectively. For the downhill task, the angles are 0, 4, 6, 10, 12, 15, 17, 22.5, 25, 28, 33 degrees, respectively.

As shown in Figure 11 (a), (c), the Success Rates of HRL for both uphill and downhill tasks are dropping dramatically if the platform angle increases. The blue and red curves indicate the Success Rate and the maximal Walking Distance from 100 individual tests, respectively. For uphill walking control, if the angle of the platform is smaller than 10 degrees, the Success Rate is guaranteed to be 1 and the robot is able to walk from the beginning to the end of the platform without falling down. If the angle of the platform is 28 degrees, the Success Rate reduces to 0.04 with the threshold of $[+0.021, -0.03]$ and the maximal walking distance decreases to 0.13m with the threshold of $[+0.06\text{m}, -0.11\text{m}]$. Compared with walking uphill, both the Success Rate and Walking Distance begin to drop from 12 degrees. If the angle of the platform is 33 degrees, the Success Rate and Distance are reduced to 0.041 and 0.33m, with the threshold of $[+0.023, -0.037]$, and $[+0.11\text{m}, -0.29\text{m}]$, respectively. For the model-based RL, both the Success Rate and the Distance drop faster, particularly for the angles between 10 degrees to 20 degrees for uphill, and between 8 degrees to 22 degrees for downhill, respectively. The variance of model-based RL is also larger than that of HRL.

To test the robustness of the controller obtained from Stage 3, the platform is rotating along y − axis with different frequencies and magnitudes. The environment is now considered to be a dynamic environment during the experiment, where it is much more challenging for the robot to complete the stable walking tasks. Considering that the robot may not walk in a straight line during the experiment as the platform is continuously rotating, we only show the Success Rate and ignore the Walking Distance. The frequencies of the platform are selected as 0.025, 0.031, 0.0417, 0.0625, 0.0833, 0.125, 0.25Hz, respectively, while the magnitudes are chosen as 1, 2, 4, 6, 8, 10, 12, 14, 15, 18 degrees, respectively.

As can be seen from Figure 12 (a), (b), the increase in the frequency and the magnitude results in a significant decrease in the Success Rate. The Success Rate can be guaranteed to be 1 if the magnitude is smaller than 8 degrees and the frequency is smaller than 0.0417Hz, which means the robot is robust enough to walk on the platform from the beginning to the end without falling down under these dynamic environments. When the frequency equals 0.25Hz, the success rate drops to 0 if the magnitude is selected as 14, 15, and 18 degrees. It means that the robot is not able to complete one step walking under the above dynamic environment. For model-based RL, shown in Figure 12 (c), (d), the Success Rate drops much faster than HRL. The 100% Success Rate area, indicated by the red area, is also smaller than that of HRL. In addition, model-based RL is highly sensitive to the change of platform angle. As a result, the robustness of HRL is better than that of model-based RL.

V. CONCLUSION

In this paper, a novel Hybrid Reinforcement Learning algorithm that consists of MBRL and MFRL frameworks was proposed and applied to a NAO robot to maintain stable walking behaviors on static and rotating platforms. HGL was proposed as the model-based off-line estimator to predict the dynamic model of the system. Compared with other Gaussian Processes based MBRL framework, the proposed HGP employs two layers of prediction that estimates the transition model twice, where it enables the algorithm to estimate the system model with few training samples and consequently increases the training time. The result of HGP provided a rough model of the system and generated an initial control input, after which the Actor Critic Pre-training scheme was proposed to pre-train the Actor Network by considering the obtained initial control input as the ground-truth. The Pre-training procedure bridged the gap between the HGP based mode-based estimator and the DDPG based mode-free optimizer. Finally, DDPG was applied as the mode-free optimizer to directly generate joint angular velocity actions to the biped robot. The sample efficiency for online learning was improved as the model-free optimizer is only needed to evaluate a small number of states and actions. The proposed HRL framework also avoided the distribution mismatch problem when integrating model-free RL into model-based RL. The simulation results showed that the proposed HRL algorithm guaranteed the efficiency of MBRL while it still achieved the steady state performance of MFRL. The algorithm was able to control a NAO robot to achieve stable walking gait on static and dynamic platforms with different frequencies and magnitudes, where the robustness of the controller was also demonstrated. Future works will involve the stable walking on a dynamic platform with 2 degrees of freedom, as well as the physical implementations on real robots.

REFERENCES

[1] C. Chevallereau, G. Bessonnet, G. Abba, and Y. Aoustin, *Bipedal Robots: Modeling, Design and Walking Synthesis*, 1st ed. Hoboken, NJ, USA: Wiley, 2008.

[2] Y. Hurmuzlu and O. D. I. Nwokah, *The Mechanical System Design Handbook Modeling, Measurement, and Control*. Boca Raton, FL, USA: CRC Press, 2001.

[3] M. Vukobratovic and B. Borovac, "Zero-moment point—Thirty five years of its life," *Int. J. Human Robot.*, vol. 1, no. 1, pp. 157–173, 2004.

[4] J. H. Park and K. D. Kim, "Biped robot walking using gravity-compensated inverted pendulum mode and computed torque control," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 4, May 1998, pp. 3528–3533.

[5] Q. Huang, K. Yokoi, S. Kajita, K. Kaneko, H. Arai, N. Koyachi, and K. Tanie, "Planning walking patterns for a biped robot," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 280–289, Jun. 2001.

[6] E. Ohashi, T. Sato, and K. Ohnishi, "A walking stabilization method based on environmental modes on each foot for biped robot," *IEEE Trans. Ind. Electron.*, vol. 56, no. 10, pp. 3964–3974, Oct. 2009.

[7] J. Yi, Q. Zhu, R. Xiong, and J. Wu, "Walking algorithm of humanoid robot on uneven terrain with terrain estimation," *Int. J. Adv. Robotic Syst.*, vol. 13, no. 1, p. 35, 2016.

[8] J.-Y. Kim, I.-W. Park, and J.-H. Oh, "Walking control algorithm of biped humanoid robot on uneven and inclined floor," *J. Intell. Robotic Syst.*, vol. 48, no. 4, pp. 457–484, Mar. 2007.

[9] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J.-K. Huang, and J. Grizzle, "Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2019, pp. 4559–4566.

[10] S. Wang, W. Chaovalitwongse, and R. Babuska, "Machine learning algorithms in bipedal robot control," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 5, pp. 728–743, Sep. 2012.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[12] F. Stulp, J. Buchli, E. Theodorou, and S. Schaal, "Reinforcement learning of full-body humanoid motor skills," in *Proc. 10th IEEE-RAS Int. Conf. Humanoid Robots*, Dec. 2010, pp. 405–410.

[13] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna, "A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots," in *Proc. IEEE Int. Conf. Robot. Biomimetics*, Dec. 2017, pp. 1818–1823.

[14] N. Navarro-Guerrero, C. Weber, P. Schroeter, and S. Wermter, "Real-world reinforcement learning for autonomous humanoid robot docking," *Robot. Auto. Syst.*, vol. 60, no. 11, pp. 1400–1407, Nov. 2012.

[15] C. Gil, H. Calvo, and H. Sossa, "Learning an efficient gait cycle of a biped robot based on reinforcement learning and artificial neural networks," *Appl. Sci.*, vol. 9, no. 3, p. 502, Feb. 2019.

[16] J.-L. Lin, K.-S. Hwang, W.-C. Jiang, and Y.-J. Chen, "Gait balance and acceleration of a biped robot based on Q-Learning," *IEEE Access*, vol. 4, pp. 2439–2449, 2016.

[17] K.-S. Hwang, W.-C. Jiang, Y.-J. Chen, and H. Shi, "Motion segmentation and balancing for a biped Robot's imitation learning," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1099–1108, Jun. 2017.

[18] H. Kim, D. Seo, and D. Kim, "Push recovery control for humanoid robot using reinforcement learning," in *Proc. 3rd IEEE Int. Conf. Robotic Comput. (IRC)*, Feb. 2019, pp. 488–492.

[19] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. Intell. Robotic Syst.*, vol. 86, pp. 153–173, May 2017.

[20] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 465–472.

[21] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, Feb. 2015.

[22] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-Learning with model-based acceleration," 2016, *arXiv:1603.00748*. [Online]. Available: http://arxiv.org/abs/1603.00748
[23] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep RL for model-based control," 2018, *arXiv:1802.09081*. [Online]. Available: https://arxiv.org/abs/1802.09081
[24] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 7579–7586.
[25] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," 2018, *arXiv:1803.00101*. [Online]. Available: http://arxiv.org/abs/1803.00101
[26] M. Burhan Hafez, C. Weber, M. Kerzel, and S. Wermter, "Curious meta-controller: Adaptive alternation between model-based and model-free control in deep reinforcement learning," 2019, *arXiv:1905.01718*. [Online]. Available: http://arxiv.org/abs/1905.01718
[27] J. J. Alcaraz-Jiménez, D. Herrero-Pérez, and H. Martínez-Barberá, "Robust feedback control of ZMP-based gait for the humanoid robot nao," *Int. J. Robot. Res.*, vol. 32, nos. 9–10, pp. 1074–1088, Aug. 2013.
[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*. [Online]. Available: http://arxiv.org/abs/1509.02971
[30] L. Tai, J. Zhang, M. Liu, J. Boedecker, and W. Burgard, "A survey of deep network solutions for learning control in robotics: From reinforcement to imitation," 2016, *arXiv:1612.07139*. [Online]. Available: http://arxiv.org/abs/1612.07139
[31] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proc. 31th Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.
[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347
[33] A. Xi, T. W. Mudiyanselage, D. Tao, and C. Chen, "Balance control of a biped robot on a rotating platform based on efficient reinforcement learning," *IEEE/CAA J. Automatica Sinica*, vol. 6, no. 4, pp. 938–951, Jul. 2019.

AO XI received the B.Eng. degree in automation from Northwestern Polytechnical University, Xi'an, in 2014, and the Postgraduate Diploma degree in advanced control and systems engineering from The University of Manchester, Manchester, in 2015. He is currently pursuing the Ph.D. degree with Monash University, Melbourne. Since 2017, he has been a Research Assistant with the Laboratory of Motion Generation and Analysis, Monash University. His research interests include humanoid biped robot, reinforcement learning, deep reinforcement learning, biped robot gait generation, flight control systems, and control theories. He received awards, including the 2012 Technology Star First Class Scholarship, the 2012 and 2013 Merit Student First Class Scholarship from Northwestern Polytechnical University, and the 2013 Endress and Hauser First Class Scholarship.

CHAO CHEN received the B.Eng. degree in mechanical engineering from Shanghai Jiao Tong University, Shanghai, in 1996, and the M.Eng. and Ph.D. degrees in mechanical engineering from McGill University, Montreal, in 2002 and 2006, respectively.

He was a Visiting Professor with the Ecole Central de Nantes, IRCCyN. From 2006 to 2007, he was a Postdoctoral Fellow with the University of Toronto. From 2007 to 2010, he was a Lecturer with the Department of Mechanical and Aerospace Engineering, Monash University, Melbourne, where he has been a Senior Lecturer, since 2011. He is currently an Adjunct Associate Professor with The Chinese University of Hong Kong. His research interests include robotic design and control, theory of mechanisms, robotic exoskeleton, robotic surgery, agricultural robots, and humanoid robots.

Dr. Chen received awards, including the 1996 Dean's Honor List from Shanghai Jiao Tong University, the 2004 ASME International Scholarship, the 2005 FQRNT Doctorate Fellowship, the FQRNT Postdoctoral Fellowship from 2006 to 2007, and the 2017 Years Innovation Award from the Australia hand Therapy Association.