Stability Control of a Biped Robot on a Dynamic Platform 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; ao.xi@monash.edu
* Correspondence: chao.chen@monash.edu

Received: 14 July 2020; Accepted: 8 August 2020; Published: 10 August 2020

Abstract: In this work, we introduced a novel hybrid reinforcement learning scheme to balance a biped robot (NAO) on an oscillating platform, where the rotation of the platform is considered as the external disturbance to the robot. The platform had two degrees of freedom in rotation, pitch and roll. The state space comprised the position of center of pressure, and joint angles and joint velocities of two legs. The action space consisted of the joint angles of ankles, knees, and hips. By adding the inverse kinematics techniques, the dimension of action space was significantly reduced. Then, a model-based system estimator was employed during the offline training procedure to estimate the dynamics model of the system by using novel hierarchical Gaussian processes, and to provide initial control inputs, after which the reduced action space of each joint was obtained by minimizing the cost of reaching the desired stable state. Finally, a model-free optimizer based on DQN (A) was introduced to fine tune the initial control inputs, where the optimal control inputs were obtained for each joint at any state. The proposed reinforcement learning not only successfully avoided the distribution mismatch problem, but also improved the sample efficiency. Simulation results showed that the proposed hybrid reinforcement learning mechanism enabled the NAO robot to balance on an oscillating platform with different frequencies and magnitudes. Both control performance and robustness were guaranteed during the experiments.

Keywords: biped robot; reinforcement learning; stability control; Gaussian processes; DQN (A)

1. Introduction

Humanoid biped robots have high degrees of freedom that allow them to imitate human motions with the purpose of completing complex tasks under different working environments [1]. Biped robots are also required to be able to adapt to irregular terrains and maintain full functionality when completing tasks, leading to a high requirement of controllers to guarantee the robustness under disturbances. Biped robots normally contain more than twenty degrees of freedom, which leads to a highly sophisticated control strategy to deal with their kinematic complexity to maintain balance [2]. Thus, stability control of humanoid biped robots has always been a difficult task for researchers in the field of robotics and artificial intelligence. The goal of all these studies in these fields is to propose new controllers to provide higher robustness, stability performance, and adaption to more complex terrains [3].

Stability control strategies of biped robots can be divided into conventional control methods and artificial intelligence-based control methods. Conventional control methods are based on the dynamic model of the robot, which involves the position of center of mass of each joint and each link, the length and width of each link, etc. These methods generate control signals for joints according to the deviation between the desired zero moment point (ZMP) and the measured ZMP [4–8]. The problem of conventional control methods is that they highly rely on the accuracy of the dynamics model. Biped
robots normally contain uncertainties in joint friction, contact force with the ground, and other unmeasurable variables. Therefore, it is extremely difficult to obtain an accurate model that contains both the robot dynamics and the environment disturbances. In addition, traditional techniques to obtain the dynamic model of the system are computationally demanding and time consuming. On the other hand, conventional controllers are designed for a specific robot model with particular parameters for that robot. For example, a novel conventional control strategy is originally proposed for an NAO robot [9]. Although it works perfectly on NAOs, it is not applicable to Michigan’s Cassie-series robots as the parameters of these robots are completely different [10]. The designed controller is able to work in different robots only if the parameters of these robots are identical or at least highly similar to one another. Thus, attempting to generalize these controllers to enable most biped robots to be adaptable and flexible to different terrains still remains difficult for researchers.

Machine learning (ML) algorithms have been proposed recently to overcome the above-mentioned limitations, since they do not require the dynamics model of the robot but still achieve the same control performance [11]. Although the integration between ML and robotics is still at their early stage, the huge potentials of ML have been proved to have a promising future in robotic control applications. Valdez, in 2019 [12], proposed an integrated machine learning structure that consists of fuzzy logic control (FLC) and conventional control techniques, which was successfully applied and validated on a real robot to complete the speed control tasks. Juang proposed a fully connected recurrent neural network (RNN) optimization technique to generate a multi-objective gait and applied it to control the gait of a real NAO robot [13]. Ferreira compared the support vector regression (SVR) and neural fuzzy (NF) network based on ZMP error correction to balance the robot [14]. Sun utilized a neural network (NN) to approximate the unknown model of the robot, where it was able to deal with system uncertainties during the task of balancing and posture control [15]. Saputra proposed a biologically inspired RNN to control the biped robot, where the neurons of the network were used to represent the muscular and sensor system of the body [16]. Although these NN structure-based approaches have been proven to be able to replace the classic control methods, and provide an efficient and robust control performance, the performance of these NNs are limited by their teacher controllers, which means if the teacher controller contains errors or the training data are not sufficient enough, the performance of the network will be significantly affected. Compared with approaches utilizing NN to estimate the system model, reinforcement learning (RL) techniques, however, directly employ interactions between the agent (robot) and the environment, where the agent can complete the desired tasks by gathering experience directly from its environment without being trained by target controllers.

Several studies, for example Rodic and co-workers [17], have been conducted involving RL as part of the control scheme to control the biped robot. They proposed a novel hybrid control strategy by combining the model-based dynamic controller with RL feedback controller. Guerrero [18] proposed a supervised RL approach that combined supervised learning (NN) 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. However, none of these methods applied RL as the direct joint level controllers, the actions were simply some discrete predefined action samples. The main limitation of their works is that only discrete states and actions were considered, where large errors or oscillations were introduced to the controller. Many recent attempts have been made by Hwang [19,20] with the purpose of applying Q-learning as the joint controller to maintain the balance of the robot of uneven terrains as well as the imitation of human postures. Although continuous actions were generated using Gaussian distribution in their work, the RL structure still relies on a look-up table, meaning that the convergence time would be extremely long. Wu [21] however, based on Gaussian basis function, employed an abstraction method to reduce the number of state cells needed and applied it on a biped robot to maintain balance on a rotating platform. Simulation results showed that it was able to achieve high precision control with a reduced dimension of state space. One of the major drawbacks to adopting abstraction is that with the generation of the clustering, the actions for each cluster may differ significantly. To achieve better control performance, actions between two nearby clusters (or states) must be transferred smoothly. Seo 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 [22]. Shi, in 2020 [23], employed the deep deterministic policy gradient (DDPG) to improve the precision of attitude tracking control of a biped robot. They proposed an offline pre-training scheme to provide the prior knowledge to the online training in the real physical environment. Garcia, in 2020 [24], proposed a safe RL algorithm for improving the walking pattern of a NAO robot to allow it to walk faster than the pre-defined configuration. The results showed that their approach was able to increase the learning speed while still reducing the number of falls of the robot during the process of learning. The above methods are all considered as model-free reinforcement learning (MFRL). Although steady state performance can be guaranteed after convergence as no model-induced error is introduced to the system, one major challenge of MFRL is that the convergence time (learning time) is extremely long since no model is used to provide prior knowledge of the system.

Model-based reinforcement learning (MBRL), as an alternative approach, is able to reduce the learning time by generating a transition model within the algorithm [10]. The transition model is essentially a mapping \( f(s, a) \rightarrow s' \) from the state \( s \) of the system at a given time step, and the applied action \( a \) (or policy) to the state \( s' \) at the next time step. This model is also known as the stochastic model, where it is defined by a probability distribution over the future state [25]. The most commonly used techniques are Gaussian processes. Deisenroth proposed probabilistic inference for learning control (PILCO) to estimate the transition model of the system, where it only requires very little training data [26]. Policies are consequently improved using the analytical results of the policy gradient. Experimental results show that the algorithm is able to control the physical robotics systems, where the learning procedure is facilitated from scratch in only a few trials [27,28]. Other techniques such as expectation maximization (EM) [29], or dynamic Bayesian networks (DBN) [30], have been proven to successfully estimate the system models. Compared with MFRL, MBRL only requires a small amount of training data, leading to a short convergence time, but it highly relies on the accuracy of the estimated model. Since the policy is computed based on the transition model, any change of model, induced by measurement error or lack of information for estimation, may lead to an exponential change of the resulting policy. Efforts must be made to update the model for each iteration to increase the model accuracy, which is extremely computationally expensive compared to updating the policy directly.

As a result, many attempts aiming at combining model-based approaches with model-free learning techniques have been made in recent years. Nagabandi in 2018 proposed a hybrid RL structure that combined NN dynamics models with model predictive control (MPC) to achieve excellent sample complexity in a model-based RL algorithm. Then, they further proposed a NN dynamics model to initialize a model-free learner, where the algorithm guarantees the sample efficiency as well as the control performance [31]. Pong instead combined MFRL with MBRL by introducing the temporal difference models (TDMs) [32]. The proposed RL structure combined the benefits of model-free and model-based RL, where it can be trained with model-free learning and used for model-based control tasks. Feinberg, in 2018, proposed a model-based (MB) value expansion method that incorporated the predictive system dynamics model into model-free (MF) function estimation to reduce sample complexity. According to their algorithm, the predicted dynamics were fitted first, then a trajectory was sampled from the replay buffer to update the actor network, after which an imagined future transition was obtained and was used to update the critic network. Both of the MB and MF parts were used to update the actor and critic network during the learning process [33]. Gu incorporated a particular type of learned model into the proposed Q-learning scheme with normalized advantage functions to improve sample efficiency. They used a mixture of planned iterative LQG 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. At the same time, the model itself was being refit for every \( n \) episodes to obtain an effective and data-efficient model learning algorithm [34]. Hafez employed a novel curious meta-controller, where it alternated adaptively between model-based and model-free control. In the end, both the MB planner and MF learner were improved mutually. The method improved the sample efficiency.
and achieved near-optimal performance [35]. However, these methods were only validated on simple fully-observed models (the walker or the swimmer) under ideal conditions, where no disturbances were added to the environment. They also required a huge amount of computations as both MBRL and MFRL were updated within the iteration during the online learning procedure. In addition, none of the above approaches consider the motor limitations, mechanical constraints, or disturbances. Thus, they are not applicable for physical biped robot joint control tasks to maintain balance as it requires precise joint velocity control at any time step to achieve the minimum steady state error under a dynamic environment.

As a conclusion, first of all, for the model-based RL to achieve a better control performance, the estimated model must be precise enough to contain all information of the system. The problem of it is that it is highly computationally demanding and time consuming to obtain such precise models. Secondly, for the model-free RL, the convergence time is extremely long as there is no prior knowledge of the system for the pure model-free RL. Thirdly, the distribution mismatch problem always exists in other types of HRL frameworks [33-36]; therefore, to overcome this problem further calculations are required to determine which policy is better, leading to less efficiency of these HRL frameworks. Fourthly, the recent HRL frameworks only validated on simple fully-observed robotic models in ideal environments; thus, they are not suitable for complex biped models such as NAO robots. Thus, a hybrid RL structure has the potential to overcome stability control challenges for biped robots, which has not been fully explored.

To guarantee the fast convergence of MFRL and to achieve the sample efficiency of MBRL, a novel hybrid reinforcement learning framework (HRL) incorporating the inverse kinematics (IK) is introduced in this work, where it successfully establishes a serial connection between MBRL and MFRL. This framework is applied to a complex biped model (NAO robot) to maintain sagittal and frontal balance on a dynamic platform with disturbances. We combined the hierarchical Gaussian processes (HGP) as the model-based estimator with DQN(\(\lambda\)) as the model-free fine tuning optimizer, where HGP is utilized to predict the dynamics of the system and DQN(\(\lambda\)) is utilized to find the optimal actions. The benefit of using this framework is that the model-based estimator does not need to obtain a very precise dynamic model of the system, which significantly reduces the number of training data as well as the computational load. Secondly, this hybrid structure successfully avoids the distribution mismatch problem (mentioned in ref. [33-36]) while combining model-based RL with model-free RL. Thirdly, as the estimated model is obtained, and the action space is significantly reduced; therefore, the convergence time of the model-free optimizer is consequently reduced. As a conclusion, the efficiency of the proposed HRL is improved compared with pure model-based RL, pure model-free RL, and other types of HRL approaches. Simulation results show that the proposed algorithm was able to be applied on a NAO robot to maintain balance on a dynamic platform under different oscillations at all times. The adaptability was also guaranteed for the robot on a platform with different rotating frequencies and magnitudes.

This paper is organized as follows: Section 2 provides the model of the NAO robot and the corresponding platform. Several assumptions and constraints are elaborated in detail. The state space and action space of the system are defined accordingly. In Section 3, the overview of RL is presented, as well as the basic principle of GP and the corresponding optimization techniques. Then, we illustrate the structure of the proposed hierarchical Gaussian processes (HGP) and briefly explain the function of each layer, after which we explain how the reduced action space is defined for each state according to the immediate cost function. Finally, we present the MFRL to improve the policy based on the temporal difference learning method. Section 4 is the experiment setup as well as all simulation results. In Section 5 is the conclusion of this work and the discussion about the potential future extension of the proposed HRL.

2. Formulation of the Problem

As shown in Figure 1, a NAO robot stands on a platform, where the platform is rotating along the x-axis and y-axis at the same time. The platform is employed here to simulate the uneven surface of the real environment, where oscillations are introduced to the system to imitate real
external disturbances. Compared with the flat surface under the experimental environment, such a platform is able to provide a more complex environment to make the robot learn a more robust and efficient controller. This paper will conduct experiments to test not only the efficiency of learning procedures (including the convergence time, number of training data, etc.), but also the robustness of the learned controller to deal with complex environments.

At the beginning of all experiments, NAO is initialized to the default standing posture. The control task of the upper body, including the torso, arms, hands, and head, is ignored as it can be considered as a point mass using the inverted pendulum model (IPM) [37]. Both of the robot’s foot soles are always attached to the platform and will not slip at any times. The ankle joint has two degree of freedoms (DoFs) which can rotate along the x-axis (roll) and y-axis (pitch). The knee joint has one DoF that allows it to rotate along the x-axis. The hip joint has two DoFs that are pitch and roll, which allows it to rotate along the x-axis and y-axis, respectively.

Figure 1. (a) The overview of NAO robot and the platform. (b) The simplified mechanism of the proposed NAO robot.

As can be seen from Figure 1, the robot is at its initial position, where the red and black frames are the world frame and the robot frame, respectively. P is the rotating platform which can rotate along the x-axis (roll) and y-axis (pitch) in the world frame. \(H_R\) 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. When the robot is at its default standing posture, the distance between the hip and the ankle is \(l_{HR}^{i} = l_{AR}^{i} = 0.2019\ m\), the distance between two hip joints are \(l_{HR}^{H} = 0.1\ m\), and the length of thigh and tibia are \(l_{HR}^{T} = l_{AR}^{T} = 0.1\ m\) \(l_{HR}^{A} = l_{AR}^{A} = 0.1029\ m\), respectively. The distance between the two hip joints is assumed to be fixed as \(l_{HR}^{H} = 0.1\ m\) at all times. There are 10 control inputs applied at hips (four control inputs), knees (two control inputs), and ankles (four control inputs). For hip joints, the control inputs are the change of pitch angles \((\Delta q_{HR}^{P}, \Delta q_{HL}^{P})\), and the change of roll angles \((\Delta q_{HR}^{R}, \Delta q_{HL}^{R})\), respectively. For knee joints, the control inputs are the change of pitch angle \((\Delta q_{KR}^{P}, \Delta q_{KL}^{P})\). For the ankle joints, the control inputs are the change of pitch angles \((\Delta q_{AR}^{P}, \Delta q_{AL}^{P})\), and the change of roll angles \((\Delta q_{AR}^{R}, \Delta q_{AL}^{R})\), respectively. Thus the control inputs, or in our case, the action applied on NAO at any time step is a 10 dimensional array that can be written as \(A = [\Delta q_{HR}^{P}, \Delta q_{AR}^{P}, \Delta q_{HL}^{P}, \Delta q_{AR}^{P}, \Delta q_{HR}^{R}, \Delta q_{AR}^{R}, \Delta q_{HL}^{R}, \Delta q_{AR}^{R}]^T\). The action space for each element is defined as \(\Delta q \in [-0.1, 0.1]\). Assuming that each dimension of the action space contains 10 discrete actions, the total action space has \(10^{10}\) possible combinations. If the dimension of action space can be reduced, the computation load for RL can be reduced significantly, leading to a reduced training complexity for MB system estimator and an increased learning speed for the MF optimizer. Thus, we employed the most commonly used and the most straightforward approach in robotics, i.e., the inverse kinematics (IK) technique, to reduce the dimension of applied action by defining the mechanical constraints. The roll action and pitch action of the left ankle are equal to the right ankle at
all times \((\Delta q_{AR} = \Delta q_{AL}, \Delta q_{AB} = \Delta q_{AL})\); the left knee always executes the same action as the right knee \((\Delta q_{AR} = \Delta q_{AL})\); the change of pitch angle of the left and right hip joints are identical for any time step \((\Delta q_{HI} = \Delta q_{HI})\); the roll angle of both hip joints are fixed to default and will not change during the entire experiment \((\Delta q_{HI} = 0)\). As a result, the action space is reduced to \(A = [\Delta q_{AP}, \Delta q_{AP}, \Delta q_{AP}, \Delta q_{AP}]^T\). Since the length of the thigh and tibia are fixed at all times, we can further reduce the dimension of the action space by applying the inverse kinematics (IK) constraints. Given the pitch and roll action of the ankle joints \((\Delta q_{AR}, \Delta q_{AR})\), the resulting change of knee joint and hip joint can be calculated as:

\[
\Delta q_k = \arccos \frac{l_{\text{thigh}}^2 + l_{\text{tibia}}^2 - (l_{\text{thigh}}^2 - l_{\text{thigh}} \cdot \sin(\Delta q_{AP}))^2}{2l_{\text{thigh}}l_{\text{tibia}}}
\]

\[
\Delta q_{HI} = \arccos \left(\frac{(l_{\text{thigh}}^2 - l_{\text{thigh}} \cdot \sin(\Delta q_{AP}))^2 + l_{\text{thigh}}^2 - l_{\text{tibia}}^2}{2 \left(l_{\text{thigh}}^2 - l_{\text{thigh}} \cdot \sin(\Delta q_{AP})\right)} + \Delta q_{AP}\right)
\]

(1)

Thus, the action space for the proposed RL algorithm is now reduced as \(a = [\Delta q_{AP}, \Delta q_{AP}]^T\). By combining the IK technique with RL structure, the dimension of action space is significantly reduced, leading to a significantly reduced convergence time. The gyroscope and accelerometers were disabled during the experiments; thus, the only measurable information we were able to obtain was the foot sensor readings. The position of the center of pressure (CoP) in the world frame was defined as two elements of the state space. As shown in Figure 2, the position of CoP is located at the frame, where the support polygon, shown as the blue region, ranges from 0.02 to 0.12 in the x-axis and 0.08 to 0.17 in the y-axis, respectively. Each foot has four foot sensors located at the bottom. The width and length of each foot are 0.045 and 0.09 m, respectively. The distance between the feet in the x direction, \(dx\), is measured in every time step as it may be slightly varying during the experiment.

![Figure 2. Position of foot sensors and the stable polygon.](image)

As can be seen from Figure 2, each foot has four sensors, denoted by \(F\). Thus, the CoP for the left foot along the x-axis and y-axis can be calculated as:

\[
X_{CoP} = \frac{x_1 F_{L1} + x_2 F_{L2} + x_3 F_{L3} + x_4 F_{L4}}{F_{L1} + F_{L2} + F_{L3} + F_{L4}}
\]

(2)

\[
Y_{CoP} = \frac{y_1 F_{L1} + y_2 F_{L2} + y_3 F_{L3} + y_4 F_{L4}}{F_{L1} + F_{L2} + F_{L3} + F_{L4}}
\]

(3)

where \(x_i\) and \(y_i\) denote the x and y coordinates, respectively, for the foot sensors. \(F_{Li}\) is the corresponding sensor readings of the left foot. The CoP of the right foot \((X_{CoP}, Y_{CoP})\) can be consequently computed using the same equations. As a result, the position of CoP for the entire robot can be computed as:
where \( F_L \) and \( F_R \) are the sum of sensor readings of the left and right feet, respectively. The desired CoP is measured \( (X_{\text{CoP}}^{\text{desired}} = 0.065 \text{ m}, Y_{\text{CoP}}^{\text{desired}} = 0.116 \text{ m}) \) while the robot is at its initial position without applying any actions or external disturbances. The state space of the system is now defined as \( S = [X_{\text{CoP}}, Y_{\text{CoP}}, q^p, q^r, \dot{q}^p, \dot{q}^r]^T \), where \( \dot{q} \) denotes the angular velocity of the ankle joint in pitch and roll. The goal of the proposed RL algorithm is to make the measured CoP as close to the desired CoP \( (X_{\text{CoP}}^{\text{desired}}, Y_{\text{CoP}}^{\text{desired}}) \) as possible while the robot is in any state with any disturbance.

3. Framework

3.1. Framework of the Proposed Hybrid Reinforcement Learning

Many existing hybrid RL structures attempt to integrate the model-based structure into model-free learning [33-36]. 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 reducing the dimension of action space with a reduced state space will also improve the sample efficiency with a lesser computational load. In addition, according to previous studies, the distribution mismatch problem always exists in their HRL frameworks [33-36]. This is because these frameworks essentially establish parallel connections between the model-based RL and the model-free RL. Both of the model-based and model-free parts are being updated within the learning loop; thus, two policies are generated parallel from the model-based and model-free parts, respectively. Sometimes, the policy obtained from the predicted model (model-based RL) is completely different from the randomly sampled policy (model-free RL); therefore, further calculations are required to determine which policy is better, leading to less efficiency of the HRL framework. We believe that avoiding the distribution mismatch problem directly 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 the 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 offline 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 3, at the beginning of offline training, the predefined discrete actions are randomly applied to the robot to collect training data \( x_i \). However, the number of states is still too large for HGP as it requires a huge amount of training data to obtain the estimated model. The computational load will be consequently increased exponentially. Therefore, to reduce the amount of all possible states, we reduced the dimension of the state space by ignoring \( X_{\text{CoP}}, Y_{\text{CoP}} \). Thus, the state space for the model-based offline training structure (HGP) was redefined as \( S_{\text{HGP}} = [q^p, \dot{q}^p, q^r, \dot{q}^r]^T \). For simplicity, we use \( s \) to represent \( S_{\text{HGP}} \) in this section. All training data is stored as an array \( \mathbb{D} \) where it contains \( N \) training tuples \( (x_i \in \mathbb{D}, i = 1, 2, \cdots, N) \). 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 states, respectively, and \( a_i \) in the current random action.

Then, the HGP was employed to estimate the dynamic model of the system, after which the reduced action space was generated by finding the actions that minimized the immediate cost. The reduced action space here was essentially a four dimensional array, where the first dimension indicated the total state, the second and third dimensions represented the pitch \( (\Delta q^p) \) and roll \( (\Delta q^r) \) actions, respectively, and the fourth dimension indicated the elements for each action. It is called “reduced” because the control input of each dimension \( (\Delta q^p) \) and \( (\Delta q^r) \) for each state was not a specific value, but a vector containing several possible actions. This was where the fourth dimension
came from. For example, we assumed that the original action space of the change of pitch angle \( \Delta q_A \) was defined by \( K \) discrete actions ranging from \(-c\) to \( c\). Thus, the action space could be written as \( \Delta q_A^p \in [-c, c] \). The goal of the proposed HGP was to significantly reduce the number of actions for each state, that is, to reduce the range of action space. The HGP could generate a new action space for each state, i.e., the reduced action space, where it only contained \( k \) actions ranging from \(-c\) to \( c\) \((k \ll K\) and \( c \ll C\)). Now, the action space could be written as \( \Delta q_A^p \in [-c, c] \).

As a result, the model-free online learning could directly search for the best policy for each state within the reduced action space above. The MBRL structure provided the MFRL with a dramatically reduced action space, leading to a significantly reduced convergence time. Consequently, this novel structure bridged the gap between MBRL and MFRL, which overcame the disadvantages of MFRL by utilizing the advantages of MBRL. Considering the large size of state space, deep Q-learning (DQN) was employed to generate a network to evaluate the value function with respect to all possible actions. DQN utilized the current measured state \( S_t \) as the input to generate a hidden layer using a feature function \( \phi(s) \); the output were the value functions \( Q(S,a\theta) \) weighted by unknown parameters \( \theta \). Note that the input state of the network is written in capital letters as \( S \) as it includes not only the state for HGP \((s)\) but the position of CoP \((X_{COP}, Y_{COP})\). The outputs were state–action value functions \( Q \) with respect to all possible actions. A detailed explanation is found in Section 3.4.

### 3.2. Hierarchical Gaussian Processes

The proposed HGP contained two layers. The first layer generated the initial model of the system w.r.t. each predefined action. The second layer utilized the predicted results from the first layer as an input to generate the final model. We assumed that the action space was defined as \( A = \{a_1, a_2, \ldots, a_N\} \), where \( a_i = \left[ \Delta q_A^p, \Delta q_A^r \right]^T \), \( N \) is the total number of action pairs. The first layer of HGP was to generate \( N \) individual predictions w.r.t. each action pair. The training samples of the first layer could be presented as \( \mathcal{D}_1 = \{\mathcal{D}_1^1, \mathcal{D}_1^2, \ldots, \mathcal{D}_1^K\} \). The training input \( (X_1 = [X_1^1, X_1^2, \ldots, X_1^N], X_1 \in \mathcal{D}_1) \) was the state pairs \( X_1^j = \{s^j_1, a^j_1\}^{j=\{1,2,\ldots,N\}} \) and the training target \( (Y_1^j \in \mathcal{D}_1^j) \) was the successor state \( \mathcal{Y}_1^j = \{s^j_{COP1}, \ldots, s^j_{COP1+1}\}^{j=\{1,2,\ldots,T\}} \), where \( j \) indicates \( j \)th action pair from the action space \( A \), \( s_1^j = \left[ \Delta q_A^p, \Delta q_A^r \right] \), \( s^j_{COP1} = \left[ X_{COP1}, Y_{COP1} \right]^T \), and \( s^j_{COP1+1} = \left[ X_{COP1+1}, Y_{COP1+1} \right]^T \). For simplicity, we use \( s_{t+1} \) to represent \( s_{COP(t+1)} \) in this section. Given the test input as \( X_1^j = \{s_1^j, a_1^j\}^{j=\{1,2,\ldots,M\}} \), the estimated training outputs were the Gaussian distributions of the successor states \( s_{COP(t+1)} = E[f(X_1^j)]X_1^j, \mathcal{D}_1^j \) calculated from a multiple outputs GP, corresponding to each action pair \( j \), shown in Equation (8). \( L \) indicates the dimension of output, which, in our case, is 2. \( \mu_t \) and \( \varphi_t \) in Equation (8) are the predicted mean and variance, which can be calculated using Equations (6) and (7), respectively.

\[
\mu_t := m_f(X_t) = E[f(X_t)X_t, \mathcal{D}] = K_f(X_t, X_t)(K + \sigma_n^2 I)^{-1}y
\] (6)
\[ \Sigma := \text{var}[f(X)]X_\Sigma \mathbb{D} = K_f(X_\Sigma, X_\Sigma) - K_f(X, X)(K + \sigma_n^2 I)^{-1} K_f(X, X) \]  

where \( K_f \) is the kernel and is defined by the squared exponential function. Then, the second layer multiple outputs GP was employed as the final estimator to predict the final model of the system, where the training input was exactly the predicted output from the first layer. We defined the state space as \( s \in [s_1, s_2, \ldots, s_M] \), where \( s_i = \left[ q_A^i, q_A^i, q^i_A, q_A^i \right]^T \), \( i \in M \) is the state pair, and \( M \) indicates the total number of state spaces. The second layer of HGP was to generate \( \lambda \) possible state pairs was still too large, generating a look-up table (Q-table) and updating it was first layer of HGP. Then, the estimated model was obtained from the second layer of HGP, where the states w.r.t. each state and action pair was obtained. 

\[ \text{Equation (8)} \]

3.3. Initial Control Inputs

The initial control inputs were calculated by find an action range for each state that minimized the following cost function:

\[ \mathbb{E}[c(s_{t+1})] = 1 - |I + \Sigma^{-1}|^{-1} \frac{1}{2\sigma_e^2} \times \exp \left( -\frac{1}{2\sigma_e^2} (s_{t+1} - s_d)^T \Lambda^{-1} (s_{t+1} - s_d) \right) \]  

where \( \Lambda^{-1} \) is the diagonal precision matrix and the elements in \( \Lambda^{-1} \) are unity. \( \Omega^{-1} = \Lambda^{-1}(I + \Sigma \Lambda^{-1})^{-1} \). \( s_d \) is the desired state. Here we utilize \( \mathbb{E}[c(s_{t+1})] \), as the expectation of the cost at \( s_{t+1} \) is essentially equals to the expectation of the cost at \( s_t \) after an action \( a_t \) is applied. Thus, for any given current state \( s_t \) after taking an action \( a_t \), the expected immediate cost \( \mathbb{E}[c(s_{t+1})] \) can be computed using Equation (9), where \( \mu \) and \( \Sigma \) are the predicted mean and covariance of \( p(s_{t+1}) \), respectively. As a result, the reduced action space is defined as \( a_{\text{reduced}} = [\Delta q_p^i, \Delta q_A] \) for any state \( s_i, i \in M \). Detailed implementation of obtaining the initial control inputs can be found in our previous paper [38].

3.4. Model-Free Fine Tuning

To obtain an accurate policy with minimum variance, the model-free tuning scheme must consider not only the joint information as the state space but also the position of CoP. As a result, the state space here was extended to \( S = [X_{\text{CoP}}, Y_{\text{CoP}}, q_A^P, q_A^P, q_A^P, q_A^P] \). However, as the number of all possible state pairs was still too large, generating a look-up table (Q-table) and updating it was impractical. Thus, we employed deep Q(λ)-network (DQN(λ)) [39] to find the best policy each state. The traditional Q-learning updates the Q-function \( Q(s_t, a_t) \) at time \( t + 1 \) after taking an action \( a_t \); the updating rule can be represented as:

\[ Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a \in A} Q(s_{t+1}, a) - Q(s_t, a_t)) \]  

where \( \alpha \) is the learning rate, \( r_t \) is the immediate reward, and \( \gamma \) is the discount factor.
where \( r_t \) is the reward at time \( t \), \( \alpha \in (0,1] \) is the learning rate, and \( \gamma \in (0,1] \) is the discount factor [40]. But with the increasing number of state pairs, the traditional algorithm was no longer feasible as it required an extremely large look-up table to store the value, where the convergence speed was significantly reduced. DQN replaced the original Q-table with a deep network, where a mapping between the state space and the value function (Q-Function) was generated. Updating the parameterized network was much more time efficient than updating the look-up table. The inputs of the network were the current measured states, and the outputs were the state-action function values. The goal of the model-free fine-tuning algorithm was to find the best action for any given state by updating the parameters. The value of the \( j \)th Q-function can be written as:

\[
Q(s, a_j) = \sum_{i=1}^{n} \phi(s_i) \theta_i
\]  

(11)

where \( \phi(s_i) \) is the feature, \( n \) represents the total number of states. DQN is a successful implementation of deep reinforcement learning (DRL), which resembles a deep neural network and learns a parameterized function \( Q(s_t, a_t | \theta) \) instead to generalize the state pairs. DQN utilizes replay memories \( D \) to store transitions \( (s_t, a_t, r_t, s_{t+1}) \) and employs gradient descent to find the parameters \( \theta \) that minimize the following lose function:

\[
L(\theta) = \mathbb{E}_{(s_t,a_t,r_t,s_{t+1}) \sim D} \left( r_t + \gamma \max_{a \in A} Q(s_{t+1}, a | \theta^-) - Q(s_t, a_t | \theta) \right)^2
\]  

(12)

where the parameter \( \theta^- \) is a scale copy of \( \theta \) that fixes the Q-value targets temporarily to prevent divergence or oscillations. On the other hand, the changes of parameters do not impact \( \theta^- \) immediately; thus, the training inputs do not have to be 100% independent and identically distributed [41]. DQN essentially contains two NN structures, where both \( \theta^- \) and \( \theta \) are being updated during the process of learning. With both target network and experience replay, DQN has a more stable input and output to train the network, where the entire scheme of learning behaves like supervised learning. However, one drawback of DQN is that it only uses the current reward to determine the loss function, that is, only the 1-step temporal difference (TD) [40] is used to update the value function:

\[
Q(s_t, a_t | \theta) \leftarrow Q(s_t, a_t | \theta) + \alpha (R_t^{(1)} - Q(s_t, a_t | \theta))
\]  

(13)

As can be seen from Equation (13), the 1-step TD return only utilizes the current \( r_t \) reward where \( R_t^{(1)} = r_t + \gamma \max_{a} Q(s_{t+1}, a | \theta) \). Only considering the current reward is not enough to generate a better policy as it may introduce overshoot and oscillation in the context of stability control. Thus, we utilized the n-step return to update the value function Equation (13), which can be presented as:

\[
R_t^{(n)} = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^{n} \max_{a} Q(s_{t+n}, a | \theta)
\]  

(14)

where \( 1 \leq n \leq T - n \). If \( n \) is large, it means that many rewards are considered to evaluate the value function, since the n-step return is equivalent to the Monte Carlo (MC) return, where huge numbers of samples are required for convergence to avoid variance. Sutton provided a tradeoff between the number of rewards and the sampling efficiency, known as the \( \lambda \)-return [39]. The \( \lambda \)-return is defined as the exponential average of all n-step returns:

\[
R_t^\lambda = (1 - \gamma) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t^{T-t}
\]  

(15)

If \( \lambda \), in Equation (15), equals to 1, the \( \lambda \)-return is the same as the MC algorithm. On the other hand, if \( \lambda = 0 \), the \( \lambda \)-return is the same as the 1-step TD algorithm. Thus, it can be treated as a smooth interpolation between TD and MC. The \( \lambda \)-return in Equation (15) is actually the “forward view” [41], which is not directly implementable in practice as it is extremely difficult at each step to predict what will happen many steps later. In addition, it is computationally expensive as it requires n-step trajectory, where \( O(N^2) \) operations must be done. It is particularly impractical for a long
trajectory. As a result, the “backward view” of \(\lambda\)-return is employed to overcome this drawback, which provides a more efficient updating rule [39]:

\[
R_t^\lambda = r_t + \gamma \left( \lambda R_{t+1}^\lambda + (1 - \lambda) \max Q(s_{t+1}, a|\theta) \right)
\]

Therefore, by applying the “backward” scheme, the computational load is reduced to \(O(N)\), which is much more efficient for calculating a long trajectory of returns [40]. In Equation (16), each \(n\)-step return becomes a sum of discounted rewards and a final maximization over action values. This is equivalent to Peng’s \(Q(\lambda)\) [42]. Consequently, the integration of \(\lambda\)-return with DQN yields DQN(\(\lambda\)) that allows the \(\lambda\)-return to be updated periodically and stored in the corresponding replay memory. Compared with DQN that stores the current reward \(r_t\) in replay memory \(D\), the DQN(\(\lambda\)) instead stores several steps of discounted rewards \(R_t^\lambda\) in \(D\), where transitions are appended to a temporary table \(T\). The \(\lambda\)-return is computed along \(T\) and stored in \(D\) if the state is within the stable region \(S\). The loss function Equation (12) is now rewritten as:

\[
L(\theta) = \mathbb{E}_{(s, a, r)|-D} \left( R_t^\lambda - Q(s, a|\theta) \right)^2
\]

where

\[
R_t^\lambda = r_t + \gamma \left( \lambda R_{t+1}^\lambda + (1 - \lambda) \max_{a^\prime} Q(s_{t+1}, a^\prime|\theta) \right)
\]

Note that the parameter \(\theta^-\) in Equations (17) and (18) is eliminated, compared with Equation (12), as all of the \(\lambda\)-returns are periodically updated using the current Q-function. Thus, it is not necessary to copy \(\theta\) to \(\theta^-\) in our case. Instead, all of the \(\lambda\)-return will be updated periodically using the present Q-function, shown in Algorithm 1.

**Algorithm 1 Updating \(R_t^\lambda\)**

1: For transition \((S_t, a_t, r_t, R_t^\lambda, S_{t+1})\) from back to front do
2: If \(S_{t+1}\) is terminal
3: Update \(R_t^\lambda \leftarrow r_t\)
4: Else
5: Obtain the next transition from \(T\)
6: Update \(R_t^\lambda \leftarrow r_t + \gamma \left[ \lambda R_{t+1}^\lambda + (1 - \lambda) \max_{a^\prime} Q(s_{t+1}, a^\prime) \right]\)
7: End If
8: End For

This guarantees the fast convergence speed while retaining a stable TD target. For any time step \(t\), the transition will be stored in \(T\) first until the current state is within the stable region \(S\). Then, the \(\lambda\)-return for each time step will be calculated from back to front using Algorithm 1. Finally, the temporary table \(T\) will be stored in \(D\), after which it will be refreshed as empty. The algorithm will then sample a random minibatch from the replay memory to improve the Q-function, where the parameters \(\theta\) will be updated using Equation (19).

\[
\theta \leftarrow \theta + \alpha \nabla_{\theta} \left( R_t^\lambda - Q(s, a|\theta) \right)^2
\]

Finally, all \(\lambda\)-returns \(R_t^\lambda\) within the replay memory will be calculated by applying Algorithm 1 again. According to the Section 3.3, the reduced action space is computed by Hierarchical GP for all states and can be defined as \(a_{\text{reduced}}^i \in \mathbb{R}^{M}, i = 1:M\), where \(i\) indicates all possible states belonging to the state space. As illustrated in Section 3.3, \(a_{\text{reduced}}^i\) is an 4D array containing all possible action pairs for all state pairs \((s_{t+1}^i, q_{t+1}^i, q_{t}^i, q_{t}^i)\), where each state pair corresponds to a 2D vector that contains all possible action pairs for this state. For simplicity, we use \(\mathcal{A}_i\) to represent the reduced action space \(a_{\text{reduced}}^i\). For any censored state \(s_i\), all possible action pairs at this state can be presented as \(\mathcal{A}_i = [a_1^i, a_2^i, \cdots, a_k^i]\), where \(k\) is the total number of action pairs in \(\mathcal{A}_i\), and \(a_k^i = \)
The state space of network is now extended to \( S = [X_{\text{CoP}}, Y_{\text{CoP}}, q_{\text{CoP}}^p, q_{\text{CoP}}^r, q_{\text{CoP}}^q] \). For any time step, the algorithm calculates the difference between the censored state \( S_i \) and all state pairs in \( s_{\text{HGP}} \), and finds the specific \( S_i \) with minimum difference. Then, the action space \( \mathcal{A}_i \) that corresponds to \( S_i \) will be selected for DQN (Algorithm 2) to evaluate the corresponding Q-function values, after which the parameter \( \theta \) will be updated by applying gradient descent scheme. Detailed implementation of the proposed algorithm is shown in Algorithm 2.

Algorithm 2 Hybrid Reinforcement Learning Algorithm

1: Initialize: State space \( S \), action space \( \mathcal{A} \)
2: Apply discrete actions randomly to the robot and collect data set \( \mathcal{D}_1 \)
3: Using \( \mathcal{D}_1 \) to generate \( \mathcal{D}_2 \), and then obtain the transition model \( \mathcal{M} \)
4: Obtain the reduced action space \( a_{\text{reduced}} \)
5: Initialize \( \mathcal{A} \) using \( a_{\text{rough}} \in [a_{\text{rough(min)}}, a_{\text{rough(max)}}] \)
6: Initialize replay memory \( \mathcal{D} \) to capacity \( N \), parameter vector \( \theta \)
7: For episode = 1, K do
8: \hspace{1cm} Reset the robot and the platform to their initial position, transition temporary table \( T = \emptyset \)
9: \hspace{1cm} For \( t = 1, T \) do
10: \hspace{1.5cm} Obtain current state \( S_t \) from sensors’ reading
11: \hspace{1.5cm} Select a random action from \( \mathcal{A}_i \) with probability \( \epsilon \), otherwise select \( a^*_i \)
12: \hspace{1.5cm} Append transition \( (S_t, a_t, r_t, R^3_t, S_{t+1}) \) to \( T \)
13: \hspace{1.5cm} If \( S_{t+1} \) is within stable region \( S_s \)
14: \hspace{1.5cm} Update \( R^3_t \) using Algorithm 1
15: \hspace{1.5cm} Store \( T \) in D, refresh \( T \): \( T = \emptyset \)
16: End If
17: Sample random minibatch of transitions from \( \mathcal{D}(S_j, a_j, r_j, R^3_j, S_{j+1}), j = 1, 2, \cdots, P \)
18: Apply a gradient descent on \( \theta \) to improve Q-function \( \theta \leftarrow \theta - \alpha \nabla_{\theta} \left( R^3_j - Q(S_j, a_j|\theta) \right)^2 \)
19: Every C step, update \( R^3_0 \) using Algorithm 1
20: End For
21: End For

4. Experiment and Discussion

4.1. Experiment Setup

The experiment environment was built in V-REP. We assumed that all sensor readings contained measurement noises at all times. The total time of offline training procedure for collecting training samples for HGP was 500 episodes, with 20,000 steps for each episode. The training data for the model-based estimator were collected by applying pre-defined discrete actions randomly to the robot, and allowing it to interact with the dynamic platform. The training data were collected as tuples which could be represented by \( \mathcal{D}_1 = [\mathcal{D}_1^1, \mathcal{D}_1^2, \cdots, \mathcal{D}_1^{121}] \), indicating that the total number of action pairs was 121. For each training sample \( \mathcal{D}_1^j \), the training input and training target could be represented as \( X_i^j = \{s_i^j, a_i^j\}_{t \in \{1,T\}} \) and \( Y_i^j = \{s_{\text{CoP}+1}^j\}_{t \in \{1,T\}} \), respectively, where \( j \) and \( T \) are the action index and the total training time, respectively. The probability distribution of successor states was obtained using the first layer of HGP. The training input is given by \( X_i^j = \{s_i^j, a_i^j\}_{t \in \{1,14441\}} \), and the probability distributions of the successor states \( \mathbb{E}[f(X_i^j)|X_i^j, \mathcal{D}_i^j] \) can be analytically computed. The results were the estimated distributions of the successor states specified.
by 121 individual transition functions. For the second layer GP, the training data were the estimated outputs from the first layer. The training data for the second layer can be defined as $D^2_k = \{s^i_k, a^k, s^i_{k+1}\}_{k \in \{1,2,\ldots,121\}}$, where there were 14,641 state pairs in total. The output was 14,641 individual transition functions, indicating the estimated distributions of the successor states $\mathbb{E}[f(X^2_k) | X^2_k, D^2_k]$. As mentioned in Section 3.2, the cost is defined as the geometric distance between the predicted state, i.e., the predicted CoP: $(x_p, y_p)$ and the desired CoP $(x_d = 0.065 \text{ m}, y_d = 0.116 \text{ m})$, which can be written as:

$$
\hat{c} = 1 - \exp\left(-\frac{1}{2 \sigma_x^2} \sqrt{(y_p - y_d)^2 + (x_p - x_d)^2}\right)
$$

Therefore, the reduced action space could be obtained by minimizing the expected immediate cost using Equation (9). Then, DQN($\lambda$) was employed as the model-free fine-tuning scheme to find the best action pair for each state. The total training episode was 322 with 20,000 steps for each episode. Instead of allowing the simulation to run forever, the episodic tasks were considered in our work. At the beginning of each episode, the robot and the platform were reset to their initial positions, and the temporary table $T$ was initialized to empty. This is because the entire experiment, including the offline training process and the online learning procedure, assumed that the robot frame (local frame) was translated from the world frame without rotation, which meant that the $x$-axis, $y$-axis, and $z$-axis of the robot frame were pointing the same directions as the world frame. However, during the experiment, some of the intense actions produced large instant velocity of the robot joints, leading the robot to deviate from the original local frame. The resulting robot frame may have rotated from the world frame, which consequently introduced errors after applying the control inputs. Episodic tasks allowed the robot to be able to recalibrate the consistency between the robot frame and the world frame, which avoided the error and increased the efficiency of the algorithm. The total number of online learning episodes was 2200. The reward function is defined as:

$$
r = \begin{cases} 
-10 & \text{Falling Down} \\
10 & \text{Stable Region} \\
\exp\left(-\frac{1}{2}(S - S_d)^T \Psi^{-1} (S - S_d)\right) & \text{Otherwise}
\end{cases}
$$

where $\Psi$ is the diagonal weighting matrix with the elements of 0.5, where each element determines the impact of the corresponding state. Considering that the final goal of the algorithm was to maintain the robot balance where the CoP must be located within the desired region, the joint angles had to vary from each state as the angle of the platform was changing; thus, joint angles were not considered as part of the desired state. Furthermore, to reduce the oscillation, the angular velocities had to be as small as possible. As a result, the stable state only contained the position of CoP as well as the angular velocities of the ankle joint, which can be presented as $S_d = [0.065, 0.116]^T$. The stable region is defined as $[0.065 \pm 0.01, 0.116 \pm 0.005]^T$.

### 4.2. Experiment Results

After obtaining the reduced action space using the proposed HGP, the robot and the platform were reset to their initial positions, after which the online learning scheme was conducted.

The goal of implementing the proposed model-free online learning algorithm to NAO robot on a rotating platform was to obtain a controller that is able to control the robot and maintain its balance on the rotating platform with different frequencies and different maximum magnitudes at all times. As can be seen from Figures 4 and 5, the sampling points (CoP) of episode 1, indicated by blue circles, are located within the predefined support region at the beginning. This meant that HGP had effectively provided an initial control strategy, where the robot was able to maintain balance but with a relatively large CoP error. Consequently, this rough control strategy led to a reduced action space which enabled the MF optimizer to find the optimal within a small range of actions. Both
figures show the procedure of online learning where the platform is rotating along the x-axis and y-axis at the same time. The maximum rotating magnitudes of the platform along x-axis and y-axis were both 20 deg. The frequency of the platform along x-axis ($f_x$) and y-axis ($f_y$) were 0.033 and 0.0833 Hz, respectively.

As shown in Figure 4a, we collected 100 random sensor readings from the beginning of learning (episode 1) to the end of learning (episode 322) from a single trail to show the process of convergence. The gray area represents the stable region of the biped robot. The blue circles and red cross markers are the real time measured position of CoP. The black dash lines are the desired CoP in the y-axis and x-axis, respectively. The more the sample points are close to the intersection of two dash lines, the more the robot is stable. The algorithm is trying to make the CoP of the robot as close to the intersection as possible.

![Figure 4](image)

**Figure 4.** (a) One hundred random sensor readings and boundaries at episodes 1 and 322 for a single trail when $\lambda = 0.6$, $f_y = f_x = 0.0333$ Hz, (b) the average episode error from 40 individual trails with respect to learning episode for different $\lambda$ ($f_y = f_x = 0.0333$ Hz), (c) 100 random sensor readings and boundaries at episodes 1 and 322 for a single trail when $\lambda = 0.6$, $f_y = f_x = 0.0833$ Hz, (d) the average episode error from 40 individual trails with respect to learning episode for different $\lambda$ $f_y = f_x = 0.0833$ Hz.

At the beginning of learning, the sampling points were scattered ranging from 0.04 to 0.09 m in the x-axis and 0.1 to 0.13 m in the y-axis. The boundaries, represented by the blue curve in Figure 4a, in the x-axis and y-axis were $(0.045 \text{ m}, 0.086 \text{ m})$ and $(0.103 \text{ m}, 0.126 \text{ m})$, respectively. During the

![Figure 5](image)

**Figure 5.** (a) One hundred random sensor readings and boundaries at episodes 1 and 322 using only deep Q-learning (DQN($\lambda$)) without hierarchical Gaussian processes (HGP) for a single trail when $\lambda = 0.6$, $f_y = f_x = 0.0333$ Hz, (b) 100 random sensor readings and boundaries at episodes 1 and 322 using only HGP when $f_y = f_x = 0.0333$ Hz, (c) the average episode errors from 40 individual trails using the proposed hybrid reinforcement learning (RL) and only using the HGP, respectively.
process of learning, the sampling points tended to concentrate at the desired steady state, the average error was consequently reduced. At the end of learning (episode 322), the sampling points were scattered ranging from 0.058 to 0.085 m in the x-axis and 0.1055 to 0.125 m in the y-axis. The boundaries, indicated by the red curve, in the x-axis and y-axis were reduced to (0.058 m, 0.068 m) and (0.108 m, 0.125 m), respectively. Figure 4b shows the average episode errors from 40 individual trials with respect to learning episodes when $\lambda$ is different, where the error for each episode is calculated by averaging the values from 40 randomly initialized trials at the same episode. As can be seen from episode 90 to episode 150, the convergence speed was relatively faster if $\lambda$ was larger ($\lambda = 0.8$). However, small $\lambda$ ($\lambda = 0.6$) led to small steady state error and thus the control performance was better. Compared with DQN ($\lambda = 0$), DQN($\lambda$) showed better overall performance, where the convergence speed was increased and the steady state error was reduced. The trained controller was consequently defined as the obtained actions for all states after the learning process when $\lambda = 0.6$. The robustness of this controller will be further tested in the next several experiments.

Then, we increased the rotating frequency of the platform to 0.0833 Hz to imitate the increase of the external disturbances, where the system was initialized and the policy was re-trained from the beginning. The trained policy can maintain the balance of the robot that all sampling points are within the support polygon; however, compared with Figure 4a, the collected sampling points are more dispersed meaning that the steady state performance is worse than before. As can be seen from Figure 4c, at the beginning of the training process, the boundaries are increased to (0.038 m, 0.1 m) and (0.083 m, 0.147 m) for the x-axis and y-axis, respectively. During the process of learning, the error in Figure 4c was always larger than that in Figure 4a. At the end of learning, the boundaries were increased to (0.05 m, 0.078 m) and (0.108 m, 0.126 m) in the x-axis and y-axis, respectively. The average errors shown in Figure 4d were also increased for all $\lambda$, compared with those in Figure 4b. As a conclusion, training the policy on the rotating platform with the minor disturbance was easier than with intense disturbance; the training outcome was highly dependent on the platform frequency.

Figure 5a shows the CoP positions and boundaries at episode 1 and episode 322 without applying HGP. Compared with Figure 4a, the robot was unstable at many time steps as the corresponding positions of CoP were outside of the stable region. Although the robot was eventually stable at episode 322, the steady state errors, $\pm 0.03$ and $\pm 0.025$ m in the x-axis and y-axis, respectively, were still much larger than the results in Figure 4a. Figure 5b shows the CoP positions and boundaries before and after the learning process using HGP without DQN($\lambda$). At the beginning, as indicated by the blue circles in Figures 4a and 5b, there is almost no difference between using HGP only and using HGP+ DQN($\lambda$). However, after the learning process, the CoP boundary of using HGP ((0.055 m, 0.073 m) in the x-axis and (0.1059 m, 0.125 m) in y-axis) was larger than using HGP+ DQN($\lambda$) ((0.058 m, 0.068 m) in the x-axis and (0.108 m, 0.125 m) in y-axis), which are presented by red star markers in Figures 4a and 5b, respectively. Figure 5c also indicates the average error during the learning procedure. The red curve is the average errors using HGP as the estimator and DQN($\lambda$) as the optimizer, which is exactly the same red curve in Figure 4b. The blue curve shows the convergence of only using HGP. The convergence speed was faster at the beginning, but it gradually reduced after episode 140. The steady state error was eventually larger than using DQN($\lambda$), since the model contained measurement noises leading to an inaccurate control input to the robot. As a result, Figure 5a proves the effectiveness of using HGP as the estimator and Figure 5b,c indicates the effectiveness of using DQN($\lambda$) as the optimizer. The proposed hybrid RL structure was able to improve the sample efficiency and reduce the steady state error.

We compared our results (Figures 4 and 5) with Wu’s [21] results, whose work also focused on the stability control of a biped robot on a rotating platform. The average CoP error of our work was 0.0125 m, while the error of their paper was approximately 0.025 m on average. In addition, our model was more complex (2 DoF of the platform) than that in their study (only 1 DoF), which meant it was much more difficult and challenging for the controller to balance the robot since more disturbances were introduced in our work. Detailed comparisons are listed in Table 1 below.
Table 1. Results comparisons between the proposed HRL method and Wu’s method [21].

| Evaluation Index         | Our Method | Wu’s Method |
|--------------------------|------------|-------------|
| ZMP Error                | 0.0125 m   | 0.025 m     |
| DoF of the platform      | 2          | 1           |
| Maximum angle            | ±38 deg    | ±30 deg     |
| Maximum frequency        | 0.0333 Hz  | 0.25 Hz     |
| Success rate (0.0333 Hz, ±30 deg) | 100%  | 82%         |

As a result, it made sense that to obtain a controller that can better control the robot to maintain stability, the training procedure had to be done on a platform with minor rotating frequencies, where both control performance and convergence speed could be guaranteed. Further experiments were consequently being conducted using the trained controller obtained from Figure 4 under the condition of \( \lambda = 0.6, f_y = f_x = 0.0333 \text{ Hz} \). More complex environments were introduced in the following experiments by increasing \( f_x \) during the experiment. The robustness of the trained controller was tested under 6 different fixed rotating frequencies. We extracted 100 sampling points from each of these experiments to show the stability of the robot, as well as the control performance of the controller.

As can be seen from Figure 6, the blue region indicates the support polygon of the robot, where the robot is stable if the sampling point is within this area but unstable if the point is outside this region. The red circles indicate the location of CoP, where the robot was controlled by the trained controller (same as Figure 4, episode 2200). As the frequency increased, the error between the desired stable state with the measured state was increased accordingly. The figure on the right side indicates the boundaries of CoP of different frequencies. The robot fell down if the frequency of the platform was larger than 0.0588 Hz, indicated by the yellow line; thus, the success rate of the policy was not 100% anymore. We further tested the robustness as well as the control performance of the controller under a more complicated environment by increasing \( f_x \) and \( f_y \) at the same time.

![Figure 6](image)

**Figure 6.** One hundred sampling points and the corresponding boundaries where \( f_x \) is ranging from 0.0333 to 0.25 Hz.

As shown in Figure 7, the figure on the left side shows the sampling points of CoP under different platform frequencies, the right figure indicates the boundaries of these points. Compared with Figure 6, it can be easily seen that the sampling points are more scattered from the desired stable state and the corresponding boundaries are extended. Therefore, it can be concluded that the robot tends to be more unstable if \( f_x \) and \( f_y \) are larger. According to Figures 6 and 7, the resulting
success rate is highly dependent on the frequency of the platform. If the frequency was large, it meant that the external disturbance was relatively intense; thus, the robot was not able to maintain balance perfectly.

Both of the pitch and roll frequencies affected the overall performance of the robot. To explore the relation between the success rate and the platform frequencies, we utilized the trained controller (obtained via the experiment shown in Figure 4) to test how both frequencies affected the success rate. Figure 9 shows the area of success rate ranging from 60% to 100% with respect to different frequencies.

![Figure 7](image)

**Figure 7.** One hundred sampling points and the corresponding boundaries where $f_y$ and $f_x$ are both ranging from 0.0333 to 0.25 Hz.

As shown in Figure 8, $f_y$ and $f_x$ both range from 0 to 0.27 Hz. The red area indicates the bandwidth of the platform that allows the robot to maintain balance with a 100% success rate (no falling detected). The maximum $f_y$ was 0.25 Hz, while the maximum $f_x$ was 0.21 Hz in this experiment. With the increase of $f_x$ from 0 to 0.1 Hz, the $f_y$ was decreased significantly from 0.25 to 0.009 Hz, after which it gradually reduced to zero. The maximum $f_y$ of 90% success rate was increased to 0.26 Hz, while the maximum $f_x$ was still 0.21 Hz. With the reduced success rate, the maximum $f_x$ was maintained at 0.27 Hz. For 70% and 60% success rates, the maximum $f_x$ was increased to 0.23 and 0.26 Hz, respectively. As a conclusion, the system was more sensitive to $f_x$ than $f_y$, since the action at the knee joints and hip joints were calculated using the predefined constraints equations, a subtle change of roll angle of ankle joins led to a significant change in other joints, and the resulting CoP would consequently change dramatically.
During the experiment, we found that most falls were detected while the platform was at its maximum rotating magnitude, the robot was more stable if the magnitude was relatively small. Thus, we believe that the performance of the controller was also dependent on the magnitude of the platform, where the robot was more likely to fall if the magnitude was large, and the success rate was consequently reduced. As a result, we conducted two experiments individually to show how magnitude influenced the success rate with different frequencies. For simplicity, the maximum magnitude of pitch and roll are represented by $m_\gamma$ and $m_\alpha$, respectively. The maximum magnitude in one direction was fixed while the other was changing during each experiment. Only 100% success rate cases were considered.

As shown in Figure 9, with the increased platform frequencies, both $m_\gamma$ and $m_\alpha$ are reduced. To guarantee the 100% success rate, the maximum magnitude was sensitive to the change of frequencies, particularly when $f_\gamma$ ranged from 0.009 to 0.25 Hz, while $f_\alpha$ ranged from 0 to 0.1 Hz.

Figure 9 indicates the change of $m_\gamma$ while $f_\alpha$ and $f_\gamma$ are changing simultaneously. It reaches 38 deg if both frequencies are small, represented by the red area. Then it drops drastically to 15 deg with the increasing frequencies, after which $m_\gamma$ reduces smoothly to 13 deg. The same result for $m_\alpha$ can be found in Figure 9 (c) and (d), where $m_\alpha$ equals to 32 deg for small frequencies and it is also sensitive for moderate frequencies. The difference between $m_\gamma$ and $m_\alpha$ is that $m_\alpha$ was more sensitive for high frequencies and, indicated by blue area, it continued dropping significantly and
reduced to 10.5 deg at an end. The reason why the magnitude could induce great influence on the system is that with the increased magnitude, the angle of the platform was also increased, where the support polygon projected on the flat ground was decreased. According to the principle of ZMP [4], for static biped robots, CoP had to be within the support polygon so that the robot could be stable. However, it was extremely hard for the controller to apply actions at the ankle joints to adjust the Centre of Gravity (CoG) and make the CoP located within the reduced region. A subtle change of joint angle could have induced a drastic change of CoG, where the resulting CoP would be changed significantly.

5. Conclusions

In this paper, a hybrid reinforcement learning framework consisting of the model-based scheme and the model-free framework was proposed and applied to a NAO robot to maintain balance on a dynamic platform. By adding the inverse kinematics, the dimension of action space was reduced before offline training. Then, a hierarchical Gaussian processes containing two layers was employed as the MB estimator to estimate the transition model between the robot and the environment, after which the initial control inputs, i.e., a reduced action space, were obtained for online tuning. Finally, the model-free online learning method-DQN (\(\lambda\)) was applied as the model-free optimizer to evaluate the reduced action space, and find the best action for each state. Compared with other GP-based model-based RL algorithms that utilize state action pairs to predict the transition model, the proposed hierarchical GP employed two layers of prediction that estimated the transition model twice, where it enabled the algorithm to estimate the system model with few training samples, and consequently increased the training time. Moreover, after obtaining the rough model of the system, the model-based estimator was no longer being updated during the online learning procedure. As a result, the computational load for online learning was reduced. The sample efficiency for online learning was improved as the model-free optimizer only needed to evaluate a small number of possible actions for each state. The proposed hybrid reinforcement learning structure also avoided the distribution mismatch problem when integrating model-free RL into model-based RL. Simulation results showed that the proposed HRL algorithm guaranteed the efficiency of model-based RL while it still achieved the steady state performance of model-free RL. The algorithm was able to control NAO to maintain balance on a rotating platform with different frequencies and magnitudes, where the robustness of the controller was also guaranteed. Future works will involve the combination between the inverse kinematics and RL as well as the extension of the state and action space to generate the walking gait for NAO robots to imitate human walking patterns. The algorithm will also be improved and be further applied on walking gait generation to enable the biped robot to be able to walk on a flat surface, inclined or declined surfaces, and on a dynamic platform. Further experiments for biped walking control will involve the comparisons between the improved HRL and other algorithms, such as soft actor critic, asynchronous advantage actor critic, and deep deterministic policy gradient. The algorithm will also be implemented into a physical robot to test the performance of the learning algorithm in the real environment with real external disturbances.

Author Contributions: A.X. contributed to algorithm development, training data collection, data analysis, dynamic model estimation, policy improvement, programming, simulation, results collection, and writing; C.C. provided significant recommendations and suggestions on the experiments and contributed to writing review and editing.

Funding: This research received no external funding.

Acknowledgments: We acknowledge Dacheng Tao for his assistance in providing significant suggestions for the development of the algorithm and for potential future applications.

Conflicts of Interest: The authors declare no conflict of interest.

References
1. Nwokah, O.D.; Hurmuzlu, Y. The Mechanical System Design Handbook Modeling, Measurement, and Control; CRC Press: Boca Raton, FL, USA, 2001.

2. Chevallereau, C.; Bessonnet, G.; Abba, G.; Aoustin, Y. Bipedal Robots: Modeling, Design and Walking Synthesis, 1st ed.; Wiley-ISTE: London, UK, 2008.

3. Gil, C.R.; Calvo, H.; Sossa, H. Learning an Efficient Gait Cycle of a Biped Robot Based on Reinforcement Learning and Artificial Neural Networks. Appl. Sci. 2019, 9, 502.

4. Vukobratovic, M.; Borovac, B. Zero-moment point thirty-five years of its life. Int. J. Hum. Robot. 2004, 1, 157–173.

5. Strom, J.; Slavov, G.; Chown, E. Omnidirectional Walking Using ZMP and Preview Control for the NAO Humanoid Robot. In RoboCup 2009: Robot Soccer World Cup XIII; Springer: Berlin/Heidelberg, Germany, 2009; pp. 378–389.

6. Yi, J.; Zhu, Q.; Xiong, R.; Wu, J. Walking Algorithm of humanoid Robot on Uneven Terrain with Terrain Estimator. Int. J. Adv. Robot. Syst. 2016, 13, 35.

7. Lee, H.; Yang, J.; Zhang, S.; Chen, Q. Research on the Stability of Biped Robot Walking on Different Road Surfaces. In Proceedings of the 2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII), Jeju, South Korea, 23–27 July 2018; pp. 54–57.

8. Yoshida, Y.; Takeuchi, K.; Sato, D.; Nemchev, D. Balance control of humanoid robots in response to disturbances in the frontal plane. In Proceedings of the 2011 IEEE International Conference on Robotics and Biomimetics, Karon Beach, Phuket, Thailand, 7–11 December 2011; pp. 2241–2242.

9. Zhong, Q.; Chen, F. Trajectory planning for biped robot walking on uneven terrain—Taking stepping as an example. CAAI Trans. Intell. Technol. 2016, 1, 197–209.

10. Gong, Y.; Hartley, R.; Da, X.; Hereid, A.; Harib, O.; Huang, J.K.; Grizzle, J. Feedback Control of a Cassie Bipedal Robot: Walking, Standing, and Riding a Segway. In Proceedings of the 2019 American Control Conference (ACC), Philadelphia, PA, USA, 10–12 July 2019.

11. Wang, S.; Chavalitwongse, W.; Robert, B. Machine Learning Algorithms in Bipedal Robot Control. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. 2012, 42, 728–743.

12. Valdez, F.; Castillo, O.; Caraveo, C.; Peraza, C. Comparative Study of the Conventional Mathematical and Fuzzy Logic Controllers for Velocity Regulation. Axioms 2019, 8, 53.

13. Juang, C.F.; Yeh, Y.T. Multiobjective Evolution of Biped Robot Gaits Using Advanced Continuous Ant-Colony Optimized Recurrent Neural Networks. IEEE Trans. Cybern. 2018, 48, 1910–1922.

14. Ferreira, J.P.; Crisóstomo, M.M.; Coimbra, A.P. SVR Versus Neural-Fuzzy Network Controllers for the Sagittal Balance of a Biped Robot. IEEE Trans. Neural Netw. 2009, 20, 1885–1897.

15. Sun, C.; He, W.; Ge, W.; Chang, C. Adaptive Neural Network Control of Biped Robots. IEEE Trans. Syst. Man Cybern. Syst. 2017, 47, 315–326.

16. Saputra, A.A.; Botzheim, J.; Sulistijono, I.A.; Kubota, N. Biologically Inspired Control System for 3-D Locomotion of a Humanoid Biped Robot. IEEE Trans. Syst. Man Cybern. Syst. 2016, 46, 898–911.

17. Katic, D.M.; Vukobratovic, M.K. Hybrid Dynamic Control Algorithm for Humanoid Robots Based on Reinforcement Learning. J. Intell. Robot. Syst. 2008, 51, 3–30.

18. Guerrero, N.N.; Weber, C.; Schroeter, P.; Wernmter, S. Real-world reinforcement for autonomous humanoid robot docking. Robot. Auton. Syst. 2012, 60, 1400–1407.

19. Hwang, K.S.; Jiang, W.C.; Chen, Y.J.; Shi, H. Gait Balance and Acceleration of a Biped Robot Based on Q-Learning. IEEE Access 2016, 4, 2439–2449.

20. Hwang, K.S.; Jiang, W.C.; Chen, Y.J.; Shi, H. Motion Segmentation and Balancing for a Biped Robot’s Imitation Learning. IEEE Trans. Ind. Inform. 2017, 13, 1099–1108.

21. Wu, W.; Gao, L. Posture self-stabilizer of a bipedal robot based on training platform and reinforcement learning. J. Robot. Auton. Syst. 2017, 98, 42–55.

22. Seo, D.; Kim, H.; Kim, D. Push Recovery Control for Humanoid Robot using Reinforcement Learning. In Proceedings of the 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 25–27 February 2019.

23. Shi, Q.; Ying, W.; Lv, L.; Xie, J. Deep reinforcement learning-based attitude motion control for humanoid robots with stability constraints. Ind. Robot 2020, 47, 335–347.

24. Garcia, J.; Shafie, D. Teaching a humanoid robot to walk faster through Safe Reinforcement Learning. Eng. Appl. Artif. Intell. 2020, 88, 103360.
25. Polydoros, A.S.; Nalpantidis, L. Survey of Model-Based Reinforcement Learning: Application on Robotics. *J. Intell. Robot. Syst.* **2017**, *86*, 153–173.

26. Deisenroth, M.P.; Rasmussen, C.E. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 465–472, Washington, USA, June, 2011.

27. Englert, P.; Paraschos, A.; Peters, J.; Deisenroth, M.P. Model-based imitation learning by probabilistic trajectory matching. In Proceedings of the 2013 IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, 6–10 May 2013; pp. 1922–1927.

28. Deisenroth, M.P.; Fox, D.; Rasmussen, C.E. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 408–423.

29. Coates, A.; Abbeel, P.; Ng, A.Y. Apprenticeship learning for helicopter control *Commun. ACM* **2009**, *52*, 97–105.

30. Nguyen, T.T.; Li, Z.; Silander, T.; Leong, T. Online feature selection for model-based reinforcement learning. *Proc. Int. Conf. Int. Conf. Mach. Learn.* **2013**, *28*, 498–506.

31. Nagabandi, A.; Kahn, G.; Fearing, R.S.; Levine, S. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 7579–7586.

32. Pong, V.; Gu, S.; Dalal, M.; Levine, S. Temporal Difference Model: Model-Free RL for Model-Based Control. *arXiv* **2018**, arXiv: 1802.0908v1.

33. Feinberg, V.; Wan, A.; Stoica, I.; Jordan, M.I.; Gonzalez, J.E.; Levine, S. Model-Based Value Expansion for Efficient Model-Free Reinforcement Learning. *arXiv* **2018**, arXiv: 1803.00101v1.

34. Gu, S.; Lillicrap, T.; Sutskever, I.; Levine, S. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv* **2016**, arXiv: 1603.00748v1.

35. Hafez, M.B.; Weber, C.; Kerzel, M.; Wermter, S. Curious Meta-Controller: Adaptive Alternation between Model-Based and Model-Free Control in Deep Reinforcement Learning, *arXiv* **2019**, arXiv: 1905.01718v1.

36. Nagabandi, A.; Kahn, G.; Fearing, R.S.; Levine, S. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning, *arXiv* **2017**, arXiv: 1708.02596v2.

37. Alcaraz-Jiménez, J.J.; Herrero-Pérez, D.; Martínez-Barberá, H. Robust feedback control of ZMP-based gait for the humanoid robot Nao. *Int. J. Robot. Res.* **2013**, *32*, 1074–1088.

38. Xi, A.; Mudiyanselage, T.W.; Tao, D.; Chen, C. Balance Control of a Biped Robot on a Rotating Platform Based on Efficient Reinforcement Learning. *IEEE-CAA J. Autom. Sin.* **2019**, *6*, 938–951.

39. Daley, B.; Amato, C. Efficient Eligibility Traces for Deep Reinforcement Learning. *arXiv* **2018**, arXiv:1810.09967v1.

40. Sutton, R.S.; Barto, A.G. *Reinforcement Learning an Introduction*; The MIT Press: Cambridge, MA, USA, 2018.

41. Feng, J.; Fyle, C.; Jain, L.C. Experimental analysis on Sarsa(lambda) and Q(lambda) with different eligibility traces strategies. *J. Intell. Fuzzy Syst.* **2009**, *20*, 73–82.

42. Peng, J.; Williams, R.J. Incremental Multi-Step Q-Learning. *Mach. Learn.* **1996**, *22*, 283–290.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).