Research Article

A Novel Model-Based Reinforcement Learning Attitude Control Method for Virtual Reality Satellite

Jian Zhang and Fengge Wu

1Institute of Software, Chinese Academy of Science, University of Chinese Academy of Sciences, Beijing, China
2Institute of Software, Chinese Academy of Science, Beijing, China

Correspondence should be addressed to Jian Zhang; 16141191@qq.com

Received 20 May 2021; Revised 1 June 2021; Accepted 16 June 2021; Published 1 July 2021

Copyright © 2021 Jian Zhang and Fengge Wu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Observing the universe with virtual reality satellite is an amazing experience. An intelligent method of attitude control is the core object of research to achieve this goal. Attitude control is essentially one of the goal-state reaching tasks under constraints. Using reinforcement learning methods in real-world systems faces many challenges, such as insufficient samples, exploration safety issues, unknown actuator delays, and noise in the raw sensor data. In this work, a mixed model with different input sizes was proposed to represent the environmental dynamics model. The predication accuracy of the environmental dynamics model and the performance of the policy trained in this paper were gradually improved. Our method reduces the impact of noisy data on the model’s accuracy and improves the sampling efficiency. The experiments showed that the agent trained with our method completed a goal-state reaching task in a real-world system under wireless circumstances whose actuators were reaction wheels, whereas the soft actor-critic method failed in the same training process. The method’s effectiveness is ensured theoretically under given conditions.

1. Introduction

SpaceVR’s Overview 1 virtual reality satellites enable us to experience space firsthand using any mobile, desktop, or virtual reality device. Observing the universe with virtual reality is an exciting application of satellites. As one of the core subsystems in satellite, the attitude control system needs to be developed to meet the requirement for space virtual reality applications. An intelligent method of attitude control plays a role part in the subsystem. Reinforcement learning gives a remarkable achievement to prove potentiality to realize intelligent control.

In recent years, a significant progress has been made in the research of reinforcement learning (RL) [1]. Researchers have proposed many RL methods, such as the deep deterministic policy gradient (DDPG), trust region policy optimization (TRPO), proximal policy optimization (PPO), soft actor-critic (SAC), and twin-delayed deep deterministic policy gradient (TD3) methods [2–6]. These methods have been applied for playing Atari games [7] completing control tasks in OpenAI Gym [8] and defeating human players in Go [9]. However, compared with simulation environments, successful cases in real-world systems are relatively rare. Application of RL methods directly in real-world systems faces a series of problems, such as sample limitations, exploration of the state space leading to system damage, unspecified reward functions, unknown delays in the system actuators, and unstable measurements from sensors [10].

In a real-world system, goal-state reaching is a task in which the control actuators adjust the system to reach a specified goal state. For example, attitude control of a satellite, including attitude maneuvering and attitude stabilization, can be defined as a task to reach a goal attitude state. The control tasks, such as balancing a car or an inverted pendulum, are also goal-state reaching problems in which the balancing point is specified as the goal state.

Approaches to tackle the challenges of real-world systems without expert knowledge can be roughly divided into two categories: “simulation to real” and “real experience data to model to real.” “Simulation to real” methods build a
simulation environment based on the real-world system and use RL methods for training. After training, the policies obtained from the simulation environment are applied in the real-world system. A number of previous studies have examined this type of method to learn robotic skills in real-world systems, including locomotion [11], robotic control [12], and vision-based autonomous flight [13].

“Real experience data to model to real” methods learn the model of the environment dynamics based on experience data from real-world systems and interact with a learned dynamics model using RL methods, and then, they apply the learned policy in a real-world system. Nagabandi et al. [14] built an environment dynamics model with neural networks, used a model predictive controller to simulate the transition trajectory in the learned environment model, and selected the best actions for the real-world system, thus realizing four-legged robot motion along a specified path. Lambert et al. [15] built an environment model with real-world system data, simulated the transition trajectory in the environment model running on multiple computing nodes, obtained the best action, and realized the attitude stability of a four-wing aircraft with low-level control, where the input was sensor data and the output was a pulse-width modulation signal.

The “simulation to real” methods depend on knowledge of the real-world system, the accuracy of the simulator, and the effectiveness of shifting the experience distribution between the simulator and the real-world system. The existing “real experience data to model to real” methods face two major challenges. First, sufficient experience data must be collected initially to model the environment dynamics. Second, the state space visited by the untrained policy without expert knowledge is insufficient for modelling the environment dynamics. In this paper, we present a method that overcomes the shortcomings of both classes of methods. Our method iterates in the loop of “collecting data using the latest policy-learning environment dynamics model-learning policy.” By using the latest policy to collect data from the real environment, it can gradually improve the model’s accuracy regarding the dynamics and policy’s performance without building a simulator. Although SAC was used for learning the latest policy, the difference from the original SAC method is that when exploring the unknown state space, we use the verified policy in the learned dynamics model, thus making the exploration more effective and safer.

Compared with the “simulation to real” methods, our method does not require knowledge of the real-world system. Existing “real experience data to model to real” method collects training samples at one time. The policy used to collect data should cover all the state space to support modelling the environment dynamics precise. But it is impossible. The difference between our method and existing “real experience data to model to real” is that the data used to train the model in our method is collected gradually, which reduces the dependence on the initial random data and thus reduces the probability of falling into a state space that will lead to disasters of the system. At the same time, it can cover a wider state space, which makes the dynamics model more precise.

The proposed method was tested using Cubli, a reaction-wheel-based inverted pendulum system. Cubli [16] is a cube whose height, width, and depth are 15 cm. It is equipped with three reaction wheels in three different planes. It is a typical nonlinear, unstable, and multidegree of freedom control system. The three reaction wheels are driven by motor rotation to generate torque to achieve attitude adjustment. Research on Cubli’s control is mainly based on classical methods of control, such as proportional–integral–derivative (PID), linear–quadratic regulator (LQR), and nonlinear control theory [16–19]. These controllers operate based on analytical models for Cubli’s dynamics, manual parameter tuning, and long design times. In our paper, we address the challenge of generating controllers without system knowledge in an end-to-end learning method.

The main work of this paper includes the following: (1) A learning system for Cubli is constructed in Section 2.1. The policy learned with our method achieves motor control from raw sensor data on board to drive the Cubli reaction wheels. The required torque produced by changing the spinning speeds of the reaction wheels leads the system to reach the goal state. (2) To solve the problems of an unknown delay of the actuator and inaccurate state information from the sensor data, a mixed environment dynamics model whose input window size varies is proposed (Section 2.3). (3) A method that can be used to learn the environment dynamics and policy based on the SAC method is presented. The convergence of the method was ensured theoretically (Section 2.5), and the effectiveness of the method was verified by experiments (Section 3).

2. Methodology

2.1. Learning System Construction. The state is generally composed of Euler angle and angular velocity around axis in a real-world system. Especially, in a goal-reaching system, the state is given by a measurement unit on board. The policy to learn outputs the control signal to the execution unit which is regarded as an action in a given state.

This section mainly describes how the learning system was built. As shown in Figure 1, the learning system was composed of four parts. (1) A real-world system called Cubli was used to collect state-action pair trajectory data. (2) The execution node executed the latest policy and interacted with Cubli via Bluetooth. (3) Real trajectory data was used to train an environmental dynamics model represented by a neural network. (4) In the environment modelled by the trained neural network, SAC method was used to train agents to obtain the latest policy. The latest policy was verified at the execution node.

At any time step $t$, Cubli obtained the state from an onboard sensor and submitted the state $s_t$ to the execution node. The execution node used the latest policy neural network for inference and obtained the current action $a_t$. After Cubli executed the action, the execution node collected the next state data $s_{t+1}$, judged whether the current trajectory had ended through the feedback state data, and saved the trajectory into the trajectory buffer. The trajectory buffer was used to train the dynamics model $P(s_{t+1} | s_t, a_t)$. 

The training process is described in detail in Section 2.4. The training of the agent was carried out through the interaction with the environment represented by the trained model. At time step $t$, $s_t$ and $a_t$ are submitted to generate $s_{t+1}$ by the model of the environment dynamics, and the reward signal is given based on $s_{t+1}$. In this study, the agent was trained by the SAC method, and the gradient of the value network and the policy network parameters estimated with batch samples were used to update the parameters. The execution node, the training of the agent, and the model of the environment dynamics were run on the upper machine. The entire learning process could operate automatically and independently, except that manual intervention was required in cases of Cubli power exhaustion or abnormalities. Although the new trajectory stored in the buffer was paused, the training of the agent did not stop. The tasks of training can be distributed between multiple machines at the same time, the learning system can be connected with multiple systems to provide more trajectories and allow the dynamics model, the agent’s value network, and policy network’s parameters to converge faster.

2.2. Definition of Goal-State Reaching under Reinforcement Learning Paradigm. Reinforcement learning is a machine learning paradigm that maximizes expected cumulative return. The problem to be solved is defined as a Markov process. A Markov process is defined by the five-tuple $<S, A, r, P, \gamma>$, where $S$ is the state space, $A$ is the action space, and $r$ is the reward function. In the problem of goal-state reaching, the reward function is often related to the current state $s_t$ and the goal state $s_{goal}$. It can be written as $r(s_t, s_{goal})$. $P$ is the state transition probability function. $P : S \times A \rightarrow S$ is a function that maps the state-action pair space to the state space, which we also call it in this paper the environmental dynamics model as $P(s_{t+1} \mid s_t, a_t)$. $\gamma$ is the discount factor used to calculate the discount cumulative return. The policy function is represented by $\pi : S \rightarrow A$, which is a function that maps the state space to the action space. The agent uses the policy function to obtain the current action according to the current state $a_t = \pi(s_t)$ and interacts with the environment to get the next state $s_{t+1}$. This process continuously loops to form a state-action trajectory $\tau = <s_0, a_0, \cdots, s_T>$. When using a neural network to represent the policy function, $\pi$ is determined by the neural network architecture and the parameter $\theta$, so it can be written as $\pi_{\theta}$, and the policy network training is to adjust the parameter $\theta$. The process can be formalized as

$$\operatorname{argmax}_{\theta} E_{\tau \sim \pi_{\theta}} \left[ \sum_{t} r(s_t, a_t, s_{goal}) \right]$$

where $r(s_t, s_{goal}) = -\|s_t - s_{goal}\|$.  

(1)

The SAC is an RL method based on maximum entropy, which emphasizes the policy’s randomness when exploring an unknown state space. While maximizing the cumulative discount return, SAC prevents the parameters of the policy network from converging to the local optimum. The following equations describe the loss functions that need to be optimized:

$$J_Q(\phi_j) = E_{(s,a,s')} \left[ Q_{\phi_j}(s,a) - (r + \gamma \min_{j=1,2} Q_{\phi_{target_j}}(s', a')) \right] - \alpha \log \pi_{\theta}(a' \mid s')$$

$$J_V(\theta) = E_{\tau \sim \pi_{\theta}} \left[ \sum_{t} r(s_t, a_t, s_{goal}) - Q_{\phi_j}(s_t, a_t) \right]$$

(2)
functions

the from current state

\( \pi \)

relative reward following policy

action pair at one timestep as the input of the model will

the noise in the onboard sensor data, only using the state-

2.3. Learning the Environmental Dynamics Model.

Because of

exploration and the utilization and to ensure the stability

of the policy in the later stage of training.

these techniques were

bellman error (MSBE), Equation (3) is the loss function of

caused by using the same set of

network parameter to solve the training instability problem

pendent

tion. The SAC takes the minimum value of two sets of inde-

Q-function parameters to solve the problem of

-estimating the

Q

value, and Equation (4) is the loss function

of the temperature parameter \( \alpha \), which defines the explore-

exploit tradeoff, with a higher \( \alpha \) corresponding to more explo-

ration, and a lower \( \alpha \) corresponding to more exploitation.

The SAC takes the minimum value of two sets of inde-

Q-function parameters to solve the problem of

overestimating the Q-function, and it uses the target Q value

network parameter to solve the training instability problem

caused by using the same set of Q value network parameters

booststrap. These techniques were first proposed by van

Hasselt et al. [20]. The optimization of the temperature

parameter \( \alpha \) is used to control the trade-off between the

exploration and the utilization and to ensure the stability of

the policy in the later stage of training.

2.3. Learning the Environmental Dynamics Model. Because of

the noise in the onboard sensor data, only using the state-

action pair at one timestep as the input of the model will

affect the accuracy of the environmental dynamics model.

We propose a model that predicts the next state by combing

state-action pairs of different sizes as the input. As shown

in Figure 2, for a state-action pair trajectory, the state at a cer-
tain moment is assumed to be \( S_t \), which is marked by blue

boxes with solid lines. The environmental dynamics model

needs to predict the state at this moment with the historical

state-action pairs. The red boxes with dashed lines in the fig-

ure represent the input, and we define the number of state-

action pairs as the size of the input window. The sizes of

the input windows in the first, second, and third parts in

Figure 2 are 1, 2, and 3, respectively. A single model is represen-
ted by a fully connected neural network. Each model uses

data with different sizes of the input windows as inputs to

predict the state change value \( \Delta s_t \) for the next state, and

the output of the K models are averaged as the actual pre-
dected state change value. The parameters of the ith model

are defined as \( \varphi_i \), and a single-environment dynamic model

is expressed as

\[
\begin{align*}
 P_{\varphi_i}(s_t \mid s_{t-1}, a_{t-1}, \ldots, s_{t-1-k}, a_{t-1-k}) + s_{t-1}.
\end{align*}
\]

When the size of the input window is \( K \), a state-action pair trajectory of length \( N \) can construct \( N - K + 1 \)

“prediction-truth” pairs represented by \( \langle s_t', s_t \rangle \) where

\( s_t' = (1/K) \sum_{i=1}^{K} P_{\varphi_i}(s_t \mid s_{t-1}, a_{t-1}, \ldots, s_{t-1-k}, a_{t-1-k}) + s_{t-1} \). The training

of the model is minimizing the mean square error between

the prediction and the truth. The process is formalized as

\[
\begin{align*}
 \argmin_{\varphi_i} E_{c_{t-1}, a_{t-1}, \ldots, s_{t-1}, a_{t-1}, s_{t-1}, s_t} D \cdot
\left| 1 \sum_{i=1}^{K} P_{\varphi_i}(s_t \mid s_{t-1}, a_{t-1}, \ldots, s_{t-1-k}, a_{t-1-k}) + s_{t-1} - s_t \right|^2.
\end{align*}
\]

2.4. Training Environment Construction. In this section, we

describe the construction of the learning environment. The

state consisted of Euler angles (roll, yaw, and pitch) and the
three-axis angular velocities. The state was directly measured by the MPU-6050 inertial measurement unit. Action outputs the pulse-width modulation to control the rotation of the motor.

A state \( s_t = [\phi, \theta, \varphi, \dot{x}, \dot{y}, \dot{z}]^T \) is a six-dimensional vector, where \( \phi, \theta, \) and \( \varphi \) represent roll, yaw, and pitch, and \( \dot{x}, \dot{y}, \) and \( \dot{z} \) represent the angular velocity of the rotations around the \( x-, y-, \) and \( z- \) axes, respectively. An action \( a_t = [a_1, a_2, a_3]^T \) is a three-dimensional vector, where \( a_1, a_2, \) and \( a_3 \) represent the values of the pulse-width modulation (PWM) to control the rotational speeds of the motors.

To standardize the state value, we replaced the Euler angles with their cosine and sine values. For the axis angular velocity, the measured data was obtained from the IMU and divided by the maximum value of each measurement, and each state component was limited to the value range of \([-1, 1]\). Therefore, the actual state is expressed as follows:

\[
\begin{align*}
\pi &= \begin{pmatrix}
\cos (\varphi), 
\sin (\varphi), \cos (\theta), 
\sin (\theta), 
\cos (\phi), 
\sin (\phi)
\end{pmatrix} \\
&= \begin{pmatrix}
\frac{\dot{x}}{\text{MAX}_X}, 
\frac{\dot{y}}{\text{MAX}_Y}, 
\frac{\dot{z}}{\text{MAX}_Z}
\end{pmatrix}
\end{align*}
\]

The control task of balancing Cubli on an edge is defined as follows. When the Cubli flips, the error from the balance point is \( e \), and only the IMU measurement data and PWM signal of the control motor are used to maintain balance on the edge. Based on the position of the momentum wheel, three edges that can achieve the balance task are determined. Euler angles (roll, yaw, and pitch) can be used to describe the distance from the balance point. As described above, the reward function of the balance task is given as follows:

\[
V^{\pi_{k+1}, P^*} = V^{\pi_{k+1}, P^*} - D_{TV, \pi^*}(P, P^*) (\text{Condition 1})
\]

\[
V^{\pi_{k+1}, P^*} = V^{\pi_{k+1}, P^*} - D_{TV, \pi^*}(P, P^*) (\text{inequality 1}).
\]

2.5. Theoretical Analysis. Theory 1: when the reward function satisfies \( r(s, s_{goal}) = -\|s - s_{goal}\| \), the value function \( V^{\pi_{k+1}, P^*} \) of the policy \( \pi_{k+1} \) acts on the real environment model \( P^* \) will be greater than or equal to the value function \( V^{\pi_{k+1}, P^*} \) while the policy \( \pi_{k+1} \) is obtained from \( \pi_{k+1} = \arg \max_{P^*} (V^{\pi_{k+1}, P^*} - D_{TV, \pi^*}(P, P^*)) \).

\[
r(s_t, s_{goal}) = -\|s_t - s_{goal}\|.
\]

Proof.

\[
V^{\pi_{k+1}, P} = V^{\pi_{k+1}, P^*} - D_{TV, \pi^*}(P, P^*)
\]

\[
= E_{s_t, P^*} [ - \sum_{t=0}^{\infty} y_t (\|s_t - s^*\| - \|s_t - \hat{s}_t\|) ]
\]

\[
\geq E_{s_t, P^*} [ - \sum_{t=0}^{\infty} y_t (\|s_t - \hat{s}_t\|) ]
\]

\[
\geq -E_{s_t, P^*} [ \sum_{t=0}^{\infty} y_t \max (\|s_t - \hat{s}_t\|) ]
\]

\[
= -D_{TV, \pi_{k+1}} (P^*, P^*)
\]

\[
\geq -D_{TV, \pi^*} (P^*, P^*) (\text{inequality 1}).
\]

From inequality 1,

\[
V^{\pi_{k+1}, P^*} \geq V^{\pi_{k+1}, P^*} - D_{TV, \pi^*}(P^*, P^*) (\text{According to condition 1})
\]

\[
\geq V^{\pi^*, P^*} (because D_{TV, \pi^*}(P^*, P^*) = 0).
\]

Verification.
Input: initialize three input length environment dynamic model parameters $\varphi_1, \varphi_2, \varphi_3.$
  Initialize policy parameters $\theta,$ Q-function parameters: $\varnothing_1, \varnothing_2$
  Empty replay buffer $D,$ empty trajectory buffer $T.$
  Set target Q-function parameters: $\varnothing_{target,1} \leftarrow \varnothing_1, \varnothing_{target,2} \leftarrow \varnothing_2$

1: repeat
2:  Collect state-action trajectory $\tau = (s_0, a_0, s_1, a_1, \cdots, s_T), $ store trajectory into replay buffer $T.$
3:  Randomly sample a batch $B_t$ of trajectory from replay buffer $T,$ construct $|B_t|$ $(N - 2)$ (prediction, truth) pair training samples.
4:  Update dynamic model by gradient descent using
\[
V_{\theta}(s_t, a_t) = \frac{1}{1/3} \sum_{i=1}^{3} P_{\theta}(s_{t+1}, a_{t+1} = s_{t+1}, a_{t+1}) + s_{t+1} - s_t \|^2
\]
5:  until convergence
6:  repeat
7:    for $i$ in range(three dynamic models) do
8:      Based on current state $s_{t-1},$ and sample action $a_{t-1} \sim \pi_{\theta}(s)$
9:      calculate next state $s'$ with executing action in dynamic models $P_{\theta}$ with previous state-action pair
10:   end for
11: Record latest action $a,$ observe latest two state $s, s'$, reward $r,$ and done signal $d$
12: Store $(s, a, s', d)$ into replay buffer $D$
13: if done signal $d$ is True then
14:   Reset environment state.
15: end if
16: for $j$ in range(update times) do
17:   Randomly sample a batch $B_t$ transitions from replay buffer $D$
18: Update Q-functions by gradient descent using
\[
\nabla_{Q}(1/|B_t|) \sum_{(s,a,s',d) \in B_t} (Q_{\theta}(s, a) - (r + \lambda (\min_{j=1,2} Q_{target,j}(s', a') - \alpha \log \pi_{\theta}(a'|s'))))
\]
19: Update policy network by gradient descent using
\[
\nabla_{\theta} (1/|B_t|) \sum_{(s,a) \in \tau} \alpha \log \pi_{\theta}(a|s) - \min_{j=1,2} Q_{target,j}(s, a)
\]
20: Update target Q-function networks with
\[
Q_{target,j} \leftarrow \rho Q_{target,j} + (1 - \rho) Q_{\theta, j}, \text{ for } j = 1, 2
\]
21: end for
22: if it is time to test in real system then
23: Run latest policy in real system, collect state-action trajectory, collect state-action trajectory $\tau = (s_0, a_0, s_1, a_1, \cdots, s_T), $ store trajectory into replay buffer $T.$
24: Observe the result running in real system
25: if task is completed then
26: completed $\leftarrow$ true
27: else
28: Update dynamic model as step 4.
29: end if
30: end if
31: until completed

Algorithm 1: Learning algorithm with hybrid window's length dynamic model.

Therefore, according to the steps described in Theory 1, the policy sequence $\pi^0, \pi^1, \cdots, \pi^k$ with a monotonically increasing value function will be generated such that $V^{\pi^0, p^c} \geq \cdots \geq V^{\pi^1, p^c} \geq V^{\pi^k, p^c}.$

The procedure is presented in Algorithm 1. The environment dynamics model represented by mixed models with different sizes of the input window is trained to convergence using trajectory data from the real environment. The agent is trained by the SAC. The update rule of the parameters in the policy function’s network is given by Equation (3), and the value function is given by Equation (2). During the application of the algorithm, Condition 1, which considers all the policy and environment dynamics models, is not satisfied. In our algorithm, we assume that the model represented by the neural network architecture includes the real environment dynamics model and that the error between the learned model trained with more real data and the real environment model is decreasing.

3. Results and Discussion

A Cubli is shown as Figure 3. The yellow area marks the brake components, the blue area marks the control unit and the onboard measurement unit, and the green area marks 3 sets of motors. It is mainly composed of three brushless motors with encoders, three sets of brake devices, an MPU6050 digital motion processor, and a main control chip STM32F103RCT6. The communication with the host computer depends on the Bluetooth module. The communication protocol between Cubli and the upper computer
consisted of seven floating-point numerical data (roll; pitch; yaw; angular velocities of rotation around the x-, y-, and z-axes; current motor speed code value; and current motor PWM value). The data were read from the DMP register in the MPU6050. The upper computer sent the target PWM value of the motor to the STM32F103RCT6 on Cubli, and the instruction was executed by the motor after it was written into the register. The frequency of setting the PWM value was 62.5 Hz, and the frequency of obtaining the onboard data was also 62.5 Hz, i.e., the period of the PWM value and obtaining onboard data were both 16 ms. The control loop is shown as Figure 4.

A single model of the environmental dynamics was represented by a fully connected neural network. The neural network had two hidden layers, each with 256 neurons, and it used random values to initialize each parameter. In the experiment, we used input windows with different sizes to test the actual effect. The neural network architecture, initialization parameters, learning method, batch size, and learning rate were fixed, and it was ensured that the experimental training conditions were as constant as possible. Table 1 shows the specific conditions. At the beginning of the experiment, a random policy was used to collect 200 state-action pair trajectories at the execution node by interacting with the real environment, corresponding to about 40,000 state-action-state transition sample data. These data were used to train the three models. As shown in Figure 5, the collection of these training data began from the initialization state.

As shown in Figure 6, using these 40,000 samples to train the environment dynamics model, we randomly initialized the parameters of each model and trained each model multiple times to evaluate the loss. After 50 times training with batch size of 256, all three models converged.

Theoretically, the larger the input window length, the more mixed models, and the higher the prediction accuracy of the final environmental dynamics model. However, as the size of the input window and the number of mixed models increase, the number of final model parameters will increase, and the training and inference time will be extended.

There was a trade-off between the size of the input window and the training time, and we found through experiments that the accuracy of the model and the requirements of the task could be satisfied when the size of input window was 3.

Algorithm 1 was used to train the agent, where the SAC was used as the controller training method to interact with the model of the environment dynamics. The agent’s policy and value functions were represented by a fully connected neural network, and it had two hidden layers, each with 256 neurons. To compare the model of the environment dynamics and verify the effectiveness of the method proposed in this paper, for each model, the same method was used to train the agent with the same setting for 100,000 steps, and the trained controller was used for verification in the real-world system. Each controller interacted with the environment for 100 epochs, and the results are shown in Figure 7. The controller trained using the model when the size of the input window was 1 was unable to complete the task in the real environment. The controller trained with the environment dynamics model with a window length of 2 could complete the task in some initial states of the real environment, but it still encountered initial states for which it could not complete the task. According to the method proposed in this paper, the controller trained using the model of the environment dynamics with an input window length of 3 could complete the task in the random initial state and quickly stabilize at the target point.

In addition, the method described in this paper was compared with the SAC. After 140,000 training steps, the controllers trained with our method and the SAC were used in a real environment. The PWM value output by the two controllers
Table 1: Parameters used in training the environmental dynamics model.

| Parameter                  | Size of input window: 1 | Size of input window: 2 | Size of input window: 3 |
|----------------------------|--------------------------|--------------------------|--------------------------|
| Size of input layer unit   | 12                       | 12 + 24                  | 12 + 24 + 36             |
| Size of output layer unit  | 9                        | 9                        | 9                        |
| Initialization of parameters | Randomly generated       | Randomly generated       | Randomly generated       |
|                            | from truncated normal distribution | from truncated normal distribution | from truncated normal distribution |
| Size of hidden unit        | 256                      | 256                      | 256                      |
| Size of hidden layers      | 2                        | <2, 2 >                  | <2, 2, 2 >               |
| Learning method/learning rate | Adam/0.003              | Adam/0.003               | Adam/0.003               |
| Batch size                 | 256                      | 256                      | 256                      |

Figure 4: The control loop of Cubli.
is shown in Figure 8. The controller trained by our method completed the task and kept the PWM value around 600 in 15 steps, but the controller trained by the SAC failed, and the motor remained in a saturated state with a PWM value of 7200.

4. Conclusion

As a promising method, reinforcement learning could be used to realize intelligent attitude control for virtual reality satellites in which the method of attitude control needs to have sufficient adaptability for meeting the various creative virtual reality satellite applications. To cope with the challenges of a high sample data acquisition cost and noise of the raw sensor data when using RL methods in real-world systems, a new method to learn the model of the environment dynamics and train the controller was developed in this study. Compared with the two main current methods, our method does not require knowledge of the system and has high coverage of the state space. Our method was verified to learn to reach the goal-state in a real-world system, Cubli, without prior knowledge. In our work, we found that the mixed model describing the environment model with different sizes of the input was effective for reducing the impact of noisy data on the model’s prediction accuracy. In our experiments, the models representing the environment dynamics with different architectures were trained to converge with the same samples, and the controllers were trained in environments with different models. Based on the results, the effectiveness of the model was verified. We found that our method of training based on the environmental dynamics and the policy gradually achieved better performance than...
Figure 8: (a) Pulse-width modulation (PWM) value output by trained controllers; (b) roll value when running trained controllers; and (c) gyro value for roll when running trained controllers (note: SAC: soft actor-critic).
the method of collecting data at a single time to train with the same number of training samples.

The Cubli, whose actuators are reaction wheels, is a simple prototype of an attitude controller subsystem in a satellite. To the best of our knowledge, the model-based RL method proposed in this paper was applied to balance Cubli on its edge for the first time. Without prior knowledge, we learned the control policy of the PWM signal whose input was raw sensor data. Our method is an end-to-end learning method.

Nevertheless, our work has limitations, which we will address in future work. (1) For the task of balancing on an edge, the effective action for Cubli is one-dimensional. In future work, we plan to expand the action space to three dimensions representing the PWMs of the three motors that control the spinning speed of three reaction wheels simultaneously. (2) The task of reaching the goal state in this work was to balance on an edge based on the mechanical abilities of Cubli. As such, in future work, we will enhance the mechanical ability to balance Cubli on a single corner and verify our methods on more complex tasks.

**Data Availability**

Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

**Conflicts of Interest**

The authors declare that they have no conflicts of interest.

**References**

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, 2018.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., *Continuous Control with Deep Reinforcement Learning*, Y. Bengio and Y. LeCun, Eds., ICLR, 2016.

[3] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, *Trust Region Policy Optimization*, F. R. Bach and D. M. Blei, Eds., ICML, JMLR.org, 2015.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, http://arxiv.org/abs/1707.06347.

[5] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning, PMLR*, pp. 1582–1591, Stockholm, Sweden, 2018.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning, PMLR*, pp. 1856–1865, Stockholm, Sweden, 2018.

[7] V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing Atari with deep reinforcement learning," 2013, http://arxiv.org/abs/1312.5602 Comment: NIPS Deep Learning Workshop 2013.

[8] G. Brockman, V. Cheung, L. Pettersson et al., "OpenAI Gym," 2016, http://arxiv.org/abs/1606.01540.

[9] D. Silver, J. Schrittwieser, K. Simonyan et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[10] G. Dulac-Arnold, D. J. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," 2019, http://arxiv.org/abs/1904.12901.

[11] J. Hwangbo, J. Lee, A. Dosovitskiy et al., "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, article eaau5872, 2019.

[12] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, Vancouver, BC, Canada, September 2017.

[13] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6008–6014, Montreal, QC, Canada, May 2019.

[14] A. Nagabandi, G. Yang, T. Asmar, G. Kahn, S. Levine, and R. S. Fearing, "Neural network dynamics models for control of under-actuated legged millirobots," 2017, http://arxiv.org/abs/1711.05253.

[15] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. J. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[16] M. Gajamohan, M. Merz, I. Thommen, and R. D’Andrea, "The Cubli: a cube that can jump up and balance," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3722–3727, Vilamoura-Algarve, Portugal, October 2012.

[17] M. Gajamohan, M. Muehlbach, T. Widmer, and R. D’Andrea, "The Cubli: a reaction wheel based 3D inverted pendulum," in *2013 European Control Conference (ECC)*, pp. 268–274, Zurich, Switzerland, July 2013.

[18] M. Muehlbach and R. D’Andrea, "Nonlinear analysis and control of a reaction-wheel-based 3-D inverted pendulum," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 1, pp. 235–246, 2017.

[19] S. Trimpe and R. D’Andrea, "Accelerometer-based tilt estimation of a rigid body with only rotational degrees of freedom," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2630–2636, Anchorage, AK, USA, May 2010.

[20] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, D. Schuurmans and M. P. Wellman, Eds., pp. 2094–2100, AAAI Press, 2016.