IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 24, 2022, accepted June 13, 2022, date of publication June 17, 2022, date of current version June 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3184002

Improved MM-MADRL Algorithm for Automatic Tuning of Multiparameter Control Systems

HONGMING ZHANG[1], **WUDHICHAI ASSAWINCHAICHOTE**[1], **AND YAN SHI**[2]

[1]Department of Electronic and Telecommunication Engineering, Faculty of Engineering, King Mongkut's University of Technology Thonburi, Bangkok 10140, Thailand

[2]Graduate School of Science and Technology, Tokai University, Kumamoto 862-8652, Japan

Corresponding author: Wudhichai Assawinchaichote (wudhichai.asa@kmutt.ac.th)

This work was supported in part by the Petchra Pra Jom Klao Scholarship; and in part by the Department of Electronic and Telecommunication Engineering, Faculty of Engineering, King Mongkut's University of Technology Thonburi.

ABSTRACT Control systems are widely used in our lives, and good control can be achieved by obtaining the optimal tuning parameters of the control system. The number of parameters that need to be adjusted for different control systems varies. With an increase in tuning parameters, the difficulty of tuning grows. Therefore, this paper proposes an improved monkey multiagent DRL (IMM-MADRL) algorithm and selects 3 test functions to test the setting environment of 2-7 parameters. Thus, these parameters are adjusted. The IMM-MADRL algorithm is based on the modified monkey-multiagent DRL (MM-MADRL) algorithm, and its initialization method, position update method and somersault operation are further improved so that it can perform good parameter tuning for a control system with many parameters. The simulation part of this paper proves the advantage of the IMM-MADRL algorithm in a multiparameter control system.

INDEX TERMS Incomplete differential PID controller, modified monkey-multiagent DRL (MM-MADRL) algorithm, improved modified monkey-multiagent DRL (IMM-MADRL) algorithm, optimization.

I. INTRODUCTION

Currently, many control systems are used in our lives, such as PID control systems, QFT control systems, fractional-order PID control systems, CRONE control systems and incomplete differential PID control systems [1]–[6]. The adjustment parameters of these control systems are in the range of 3-7, and different control methods have different advantages. It is known that the PID control system is the most widely used. On this basis, many researchers have proposed more novel control systems based on PID control, such as fractional-order PID control and incomplete differential PID control. These two systems contain 4-5 adjustment parameters that can achieve better actual control (control effect). The parameter autotuning algorithm will help us to better determine the optimal parameter combination of various control systems. At present, many automatic tuning algorithms have been proposed [7]–[36], and good results have been achieved through experimental tests, such as the hybrid bacteria foraging optimization algorithm and particle swarm optimization (hBFOA–PSO) algorithm, Bat algorithm (BA), and chaotic genetic algorithm (CGA). These swarm intelligence algorithms follow fixed search patterns and automatically tune the parameters of the control system through appropriate parameter initialization settings. However, these swarm intelligence algorithms may fall into a local optimal solution. Therefore, new swarm intelligence learning algorithms that combine the reinforcement learning algorithm [37]–[50] and swarm intelligence algorithm have been proposed by researchers, such as the Q-SLP algorithm and MM-MADRL algorithm [51] and [52]. These algorithms interact with the environment by using reinforcement learning algorithms and reward their users. Feedback reduces the interference of artificially set parameters while taking advantage of the swarm algorithm.

The MM-MADRL algorithm [52] is a swarm intelligence learning algorithm that was proposed in 2020. The algorithm combines the Multiagent Reinforcement Learning Algorithm (MADDPG) and the Monkey Swarm Algorithm (MA) in a nonlinear PID control environment. Combining the parameter environment with the operation of the jump interval can obtain good parameter tuning results. Furthermore, researchers have also expounded some shortcomings and limitations of the MM-MADRL algorithm, and the algorithm itself still has much room for improvement.

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

Therefore, based on the principle of MM-MADRL, this paper improves the four steps of initialization, environment design, position update and hopping interval and proposes the IMM-MADRL algorithm, which renders the preceding algorithm suitable for various types of control systems. In the simulation experiment, the test function and incomplete differential PID control system are used for testing and compared with the original MM-MADRL algorithm. The experimental results prove the advantages of the improved algorithm. Therefore, the contributions of this paper can be summarized as follows:

1. The IMM-MADRL algorithm is proposed for the self-tuning of incomplete differential PID parameters.
2. The IMM-MADRL algorithm control system has more parameter dimensions.
3. We improve the environment design, initialization, location update and somersault operation
4. The code for IMM-MADRL and the code for the incomplete differential PID controller are written in Python.

II. TECHNICAL BACKGROUND

A. INCOMPLETE DIFFERENTIAL PID CONTROLLER

From the basic principle of PID control, we know that the introduction of differential signals can improve the dynamic characteristics of the system, but there is also a problem in that it is easy to introduce high-frequency disturbances, especially when the deviation disturbance changes abruptly. To solve this problem, people introduce low-pass filtering. This is the basic idea of incomplete differential PID control. In the literature [5], the incomplete differential PID controller is compared with the original PID controller, and it has been proven that the incomplete differential PID controller is superior to the original PID controller. This controller contains the following two structures:

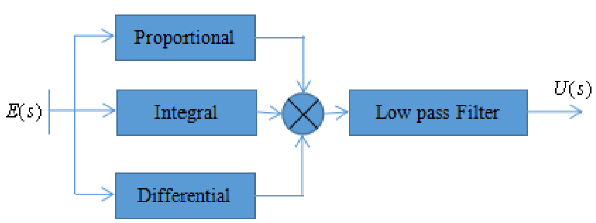

FIGURE 1. First type of control structure.

From Figures 1 and 2, the transfer function of the low-pass filter is:

$$G_f(s) = \frac{1}{T_f s + 1} \tag{1}$$

From Figure 1, this structure is a combination of the differential term and the first-order inertial ring. The calculation formula of the k_p and k_i terms is unchanged, and the transfer function is as follows:

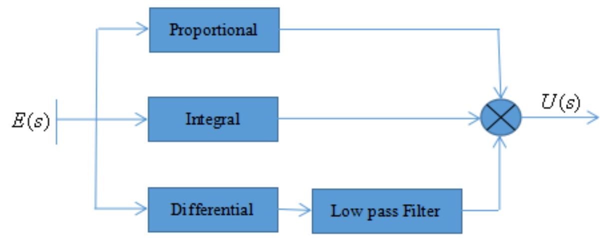

FIGURE 2. Second type of control structure.

$$U(s) = (k_p + \frac{k_p/T_I}{s} + \frac{k_p T_D s}{T_f s + 1})E(s)$$
$$U(s) = U_p(s) + U_I(s) + U_D(s) \tag{2}$$

Note that $U(s)$ is the controller output. In addition, k_p is the proportion number, T_I is the integration time constant, T_D is the differential time constant, T_f is the filter coefficient, $E(s)$ is the bias input and $U_P(s)$, $U_I(s)$ and $U_D(s)$ are the outputs of the proportional term, the integral term and the derivative term, respectively.

After discretizing Equation (2) and deriving the differential term, we have:

$$U_D(s) = \left(\frac{k_p T_D s}{T_f s + 1}\right) E(s) \tag{3}$$

If we rewrite Equation (3) as a differential equation, then we obtain:

$$u_D(t) + T_f \frac{du_D(t)}{T} = k_P T_D \frac{de(t)}{dt} \tag{4}$$

After discretizing Formula (4), we finally obtain:

$$u_D(k) = \frac{T_f}{T_s - T_f} u_D(k-1) + k_P \frac{T_D}{T_s - T_f}(e(k) - e(k-1)) \tag{5}$$

Note that k is the time count, $k = 0, 1, 2 \ldots, n$ and e is the error. T_s is the sampling time. If we set,

$$a = \frac{T_f}{T_s + T_f}, \quad \text{then} \frac{T_s}{T_s + T_f} = 1 - a$$

As a result, the calculation formula of the final differential term is:

$$u_D(k) = k_D(1-a)(e(k) - e(k-1)) + a u_D(k-1) \tag{6}$$

From Figure 2, in this structure, a low-pass filter is connected in series after the main body of the PID parameter calculation. The purpose of this setup is to suppress the influence of high frequency.

Therefore, the final output calculation formula of the controller is as follows:

$$U_{final}(s) = \frac{U(s)}{T_f s + 1} \tag{7}$$

The calculation formula of this structure is very similar to the calculation method of the original position PID control.

Because this formula can be directly written according to Formula (7) combined with the position PID formula, it will not be written in detail here.

The purpose of the two kinds of incomplete differential PID controllers is to smooth the error and accumulate the effect. The value of a is a number between 0 and 1. When the two limit values are 0, we actually have an ordinary differential link without filtering, and when these two values are 1, there is no differential effect. Therefore, the value of a is crucial to the effect of incomplete differentiation.

B. MODIFIED MONKEY-MULTIAGENT DRL (MM-MADRL)

The MM-MADRL algorithm is a swarm intelligence learning algorithm that combines the advantages of the multiagent reinforcement learning algorithm and the monkey group algorithm. This algorithm makes each monkey interact with the environment to obtain its climbing step size, which reduces the influence of artificially set parameters and uses rewards as the correct direction in the monkey group search. According to the basic concept of the somersault operation in the literature [52], the somersault operation's primary notion is to make the search less likely to fall into the local optimal solution while also making the algorithm converge faster.The pseudocode of the algorithm is as follows [52]:

Based on the literature [52], the MM-MADRL algorithm still has some weaknesses as follows:

1. The design of its environment is too biased toward the controller optimization environment with 3 or fewer parameters.
2. In the design of the number of agents, the number of agents is also within the artificial control range, and the number of agents in the reinforcement learning algorithm will also affect the efficiency.
3. In the somersault operation, the optimal solution for each step is used to jump the parameter range. In fact, the use of the current optimal solution has not been maximized.
4. In each step of the update, the state is actually reinitialized. This step slightly wastes the optimization efficiency of the agent that is already close to the optimal position.

Algorithm 1 Modified Monkey – Multiagent Deep Reinforcement Learning Algorithm (MM-MADRL)

Set cycle search times T
for episode = 1 to T do
Initialize a random process β for action exploration, receive initial state information x.
Set the number of search rounds episode_length = T_C
split search space $M = (M_1, \ldots M_N)$
for $t = 1$ to max_episode do
  For each monkey m, select the action $a_m = u\theta_m(o_m) + \beta(t)$
  Return to the collection of all of the monkeys' actions $\overrightarrow{a} = (\overrightarrow{a_1}, \ldots \overrightarrow{a_N})$
  Store reward $r = (r_1, \ldots .r_N)$ and new state X' (x, $\overrightarrow{a}$, r, x') in relay buffer $\mathcal{D}$
  Record the best position $best\_x'$ for each episode_length
  Set the stop condition of the loop episode >10 && r'-$r < 0.001$
  do somersault ()
  Start interval update after NC rounds from $best\_x'$ to obtain (MAX_x, MIN_x),
   $c \leftarrow x - (x - MAX\_x) * 0.1 d \leftarrow x - (x - MIN\_x) * 0.1$
  Return to the main loop
  Reach the set goal
  Output the global $Best\_x$
  for monkey i = 1 to N do
    Sample random min-batch of S samples (X^m, a^m, r^m, X'^m) from $\mathcal{D}$.
   Set $y^m = r^m + \gamma Q_m^{u'}(X'^m, a_1', \ldots, a_N') \mid_{a_k' = u_k'(o_k^m)}$
  Update the critic by minimizing the loss:

$$\mathcal{L}(\theta_i) = \frac{1}{S} \sum_j \left(y^j - Q_i^{\mu} \left(\mathbf{x}^j, a_1^j, \ldots, a_N^j \right) \right)^2$$

  Update the actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S} \sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i \left(o_i^j \right) \nabla_{a_i} Q_i^{\boldsymbol{\mu}} \left(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j \right) \Big|_{a_i = \boldsymbol{\mu}_i \left(o_i^j \right)}$$

  end for
  Update target network parameters for each monkey i
  $\theta_i' \leftarrow \tau \theta_i + (1 - \tau) \theta_i'$
  end for
end for

III. IMM-MADRL DESIGN PRINCIPLES AND IDEAS

First, in the initialization process, because the original algorithm is within the search range M_n, the agents are randomly and uniformly distributed. Still, the optimal position that was found may also be included. To make the work of the algorithm more accurate, in the initialization part of the IMM-MADRL algorithm, a repulsion factor is added; its purpose is to exclude the optimal position during initialization, distribute the agents in the range M_{2n} far from the optimal position, and at the same time move 20% of M_{2n} to a random uniform distribution within 80% of the range. The principle of the repulsion factor is that the reward of the agent in the initial position is not included in the range of less than 1.

At the same time, when initializing the boundary, a new boundary operation instruction is added, and an ac parameter is added. The function of the ac parameter is that when the wall is hit, the agent will no longer directly reach the boundary but move to the boundary according to the value of the action; for example, when ac=0.5, the agent moves half of the current margin to the boundary. This behavior makes the search coverage of the agent more accurate, and it is not easy for the agent to fall into the local optimal solution position in the process of shrinking the scope.

Because the initialized search range is enlarged, corresponding changes are also made in the termination condition, and a new termination rule is added. After enlarging the random range of the starting point, it is still possible to finally reach a local optimum. Hence, we add a new item to the Convergence Judgment Conditions:

If the convergence condition of Modification 1 is reached and max_reward is still less than the expected threshold, the exploration range is reset to the full space, and the entire update process is repeated. The upper limit of the number of cycles can be specified.

At the same time, changes are also made in each position update, and the optimal solution position of the previous step is taken as the initial point. The original MM-MADRL algorithm is initialized to the initial position in every position update but records every time the optimal position of the agent is obtained according to the movement of the environment. The optimal position data are used as the basis for the algorithm to operate the jump interval, and the space is reduced and jumped according to the operation. There are some repetitive search operations of the agent.

The most important change in the IMM-MADRL algorithm is the change in the somersault operation. In the somersault operation, the idea of a multiagent reinforcement learning algorithm is also introduced. The basic ideas are as follows:

1. Record the final optimal solution obtained by the previous search.
2. Using the search boundary as the starting point, pull all agents back to the boundary.
3. Take the optimal solution that has been obtained as the target and let the agent make a quick jump in the target direction.
4. Set a pseudogradient between the agent and the target, and let the agent use the pseudogradient direction as its search direction.

The specific implementation is as follows:

Space boundary setting: starting from the center point/zero point and extending to both ends in any dimension, two boundary points are obtained when the boundary is touched. Hence, we have a total of $2 \times n\_states$ boundary points.

Pseudogradient setting: when exploring the i step to the known optimal point *Best_state*, record the current position as $state_i$, and then use $state_{i-1}$ and $state_i$ to calculate the previous drive gradient Δpre. The postdrive gradient Δpro can be calculated by using $state_i$ and *Best_state*, and the linear combination of these two gradients satisfies the requirement of the pseudogradient. Currently, the linear combination coefficient is 0.5 (the parameters have room for adjustment), i.e., $\Delta = 0.5 \times \Delta pre + 0.5 \times \Delta pro$

Furthermore, the update step size λ is also reduced from 0.9 to 0.5 with the number of explorations:

$state_i = state_{i-1} + \lambda \frac{\Delta}{\|\Delta\|_2}$ (the parameters have room for adjustment).

Termination condition: the distance between the exploration points from all directions and the previous optimal point is less than a certain threshold (the parameters have room for adjustment), namely, one thousandth of the original space width.

In this way, our process is equivalent to making good use of the final optimal solution obtained after the subject search and using it as the goal to let the agent perform a fast jump search of global scope. This process permits the algorithm to eliminate the local optimal solution if possible. If there is no better solution than the previous optimal solution in the final search, this solution will be output. If there is a better position than the previous optimal solution and there is no significant change after several consecutive runs of the algorithm, this solution will be output as the global optimal solution.

The pseudocode of the IMM-MADRL algorithm is as follows:

Therefore, IMM-MADRL works as follows:

1. Initialization
2. Execute the subject search operation and record the optimal position each time.
3. Continue the search with the optimal position as the initial point, and repeat the operation of 2.
4. Repeat operations 2 and 3 until the termination condition is reached or the total number of iterations is met. Then, the current optimal position is recorded.
5. Take the current optimal position as the target point, pull all of the agents back to the boundary, and distribute them evenly on the boundary.
6. Use the pseudogradient direction as the search direction to perform a full-scale fast search.
7. End the algorithm and output the final solution.

IV. PARAMETER TUNING BY THE IMM-MADRL ALGORITHM

This section redesigns the environment for the control system with more parameters and analyzes the environment design by taking the incomplete differential PID controller used in this paper as an example. First, when using the algorithm, the number of parameters must correspond to the state in the algorithm, which would imply that the search spaces formed by different numbers of parameters are different. In short, we have 3-7 possible dimensions. The parameter adjustment environment of the control system for each tuning parameter is regarded as a 3-7 dimensional environment. Consequently, the first step is to normalize the parameter range. The so-called normalization process gives a total parameter range. For example, we have 4 parameters such that each one has a different range, but each one is within (0,2). This range can include the range applicable to all parameters; then, when using the algorithm, only one overall (0,2) range will be given as the overall search space. This operation can make the algorithm applicable to more dimensional adjustment. The parameter system is not limited to a control system of a certain dimension.

Here, *semi_diameter* is used to calculate the position of the remaining *n_agent*-1 other points around the starting point, and the current value is 1/3 of the size of the current

Algorithm 2 Improved Modified Monkey – Multiagent Deep Reinforcement Learning Algorithm (IMM-MADRL)

Set cycle search times T
Enter shrinkage factor *lamb* $(lamb \in (0,1))$
Entry boundary (a, b)
Set somersault max steps = 200
for episode = 1 to T do
Initialize a random process β for action exploration,
receive initial state information x.
Set the number of search rounds episode_length = T_C
split search space $M_n = (M_{1n}, \ldots M_{Nn})$, repel the range M_{2n} far from the optimal position.
for $t = 1$ to max_episode do
  For each monkey m, select the action $a_m = u\theta_m(o_m) + \beta(t)$
  Return to the collection of all of the monkeys' actions $\overrightarrow{a} = (\overrightarrow{a_1}, \ldots \overrightarrow{a_N})$
  Store reward $r = (r_1, \ldots . r_N)$ and new state *state'* $(x, \overrightarrow{a}, r, x')$ in relay buffer $\mathcal{D}$
  Record the best position *state* for each episode_length
  if episode >0:
    search_range = [
      [search_range[i][a] ∗ *lamb* + *state* ∗ (1 - *lamb*),
      search_range[i][b] ∗ *lamb* + *state* ∗ (1 - *lamb*)]
      for i in range(n_states)
      ]
  else:
    search_range = Initialize state
    state = (Best_state, n_agents, search_range)
    episode+ = 1
Termination condition setting()
now_reward =max_reward
reward_increase_ratio = |(now_reward-pre_reward)| / |max(pre_reward)|, a, b)
if reward_increase_ratio <=0.01:
  final_done_episode+ = 1
else:
  final_done_episode = 0
if final_done_episode > 10:
  if now_reward > -0.1:
    finally_episode = True
  else:
    search_range = Initialize state
    final_done_episode = 0
    print('reset range')
if episode >100:
  finally_done = True
somersault ()
Enter the optimal solution obtained in the previous step *Best_state*
Return all individuals back to the boundary and distribute them equally
across the boundary
  now_set_pre = State_set =Best_state

Algorithm 2 *(Continued.)* Improved Modified Monkey – Multiagent Deep Reinforcement Learning Algorithm (IMM-MADRL)

  now_set =state_set + pseudo_gra
  pseudo_gra_pro = np.array(
  [[(env.reward(state_set[i]) - max_reward) / (state_set[i][j] -
  max_state[j]) for j in range(n_states)] for i in range(2 ∗ n_states)]])
  pseudo_gra_pre = np.array([[(env.reward(state_set[i]) - env.reward(state_set_pre[i]))
  / (
  state_set[i][j] - state_set_pre[i][j]) for j in range(n_states)] for i in
  range(2 ∗ n_states)]])
Termination condition setting()
Output the global *Best_state*
for monkey i = 1 to N do:
  Sample random min-batch of S samples (X^m, a^m, r^m, X'^m) from $\mathcal{D}$.
   Set $y^m = r^m + \gamma Q_m^{u'}(X'^m, a'_1, \ldots, a'_N) |_{a'_k = u'_k(o^m_k)}$
  Update the critic by minimizing the loss:

$$\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\mu}\left(\mathbf{x}^j, a_1^j, \ldots, a_N^j\right)\right)^2$$

  Update the actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i\left(o_i^j\right)\nabla_{a_i}Q_i^{\boldsymbol{\mu}}\left(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j\right)\Big|_{a_i=\boldsymbol{\mu}_i\left(o_i^j\right)}$$

   end for
  Update target network parameters for each monkey i
  $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
  end for
end for

exploration space. This is a vector of *n_state* dimensions because the exploration space may not be the same size in each dimension.

When constructing other points, the method used is to start from the starting point and move the *semi_diameter* length in the positive or negative direction of a dimension to obtain a newpoint. Therefore, considering that each dimension can obtain two new points for a total of $2\times n\_state$ new points, the number of agents is set to $2\times n\_state+1$. After the construction is completed, if a point jumps out of the boundary, it falls directly onto the boundary.

After this change, we can only change the value of n according to the number of parameters, where n=1,2,3…n, which corresponds to the number of parameters and the number of agents. The algorithm will make its own decision based on the formula and then proceed to the initialization operation in the previous section.

The basic parameters are set as follows:

It can be seen from Table 1 that the process of setting the basic parameters has greatly reduced the need to manually input the initial values of the parameters from the table, which will further reduce the impact of the manual settings on the algorithm search.

TABLE 1. Basic parameter list.

(a ,b)	Normalized Search Scope
n_state = n	n =1,2,3,...,n the number of parameters
n_actions = n_state	(The dimension of the action, such as n_state = 6 , has 6 parameters, and the only action of the body is a 6-dimensional movement method)
n_agents = 1+2 × n_states	Automatically calculate the number of agents
shared_reward = True , Actor network learning rate and critic network learning rate 0.0001 , Maximum iteration episode = 10000, T_C *= 200.*	

Taking the Ackley test function used in the simulation in this paper as an example, based on the setting method mentioned above, the basic parameters are set as follows:

It can be seen from TABLE 2 that when running the IMM-MADRL algorithm in different environments, it is only necessary to change the search range of the parameters and the number of parameters for everything to work. In addition, the specific environment for adjusting parameters needs to be in the setting where the algorithm begins.

The design principle of the reward function is the same as the design principle in the literature [52]. This paper uses IAE as the reward function, and its expression is:

$$IAE = \int_{o}^{\infty} |e(t)|dt \tag{8}$$

In part of the somersault operation, in addition to IAE, the design of the reward function also sets a pseudogradient direction and uses double judgment criteria, which is equivalent to setting a new search environment based on the optimal solution of the previous operation during the flip operation. Therefore, the agent can perform a full-range search more accurately and comprehensively in the somersault operation.

In this paper, the environment is replanned and designed for the control system of a multiparameter environment. Moreover, the idea of reinforcement learning is also added to the somersault operation, and a separate search environment for the somersault operation is designed to make better use of the best results found in the previous steps. The optimal solution can also better solve the problem of falling into a local optimal solution.

TABLE 2. Parameters list for ackley test function.

(a ,b)	Normalized Search Scope (-32 , 32)
n_state = n	n =1,2,3,...,n Depends on how many parameters you want to run
n_actions = n_state	(The dimension of the action, such as n_state = 6 , has 6 parameters, and the only action of the body is a 6-dimensional movement method)
n_agents = 1+2 × n_states	Automatically calculate the number of agents
shared_reward = True , Actor network learning rate and critic network learning rate 0.0001 , Maximum iteration episode = 10000, T_C *= 200.*	

V. SIMULATION AND RESULTS ANALYSIS

The simulation uses Jupiter in Anaconda Navigator, Pycharm 2020.3.3, Python 3.6 and PyTorch to edit the IMM-MADRL code and simultaneously edit the code of the incomplete differential PID controller used in the simulation. Ornstein–Uhlenbeck process (OU) noise has also been added to the simulation, and the system running on the simulation is an Intel(R) Core(TM) i7-6700T CPU @ 2.80 GHz, Windows 10, Alienware.

A. TEST FUNCTION SIMULATION

This paper uses three test functions to test the IMM-MADRL algorithm. The three test functions are the Ackley function, Rastrigin function and Griewank function. They run in a 2-7 dimensional environment and obtain results. The algorithms are compared, and the resulting graph after the comparison is obtained. The maximum number of iterations is 1000, dim= 2-7 and lamb=0.5. The results are as follows:

From Figures 3, 4 and 5, it can be seen that the IMM-MADRL algorithm can perform well in different parameter dimensions of the three test functions. In Figure 5, to make the curve easier to observe, the rewards of different dimensions are changed. The threshold has no effect on the final result. It can clearly be seen from the image that the IMM-MADRL algorithm can exactly find the optimal solution for parameter tuning in different parameter dimensions.

To intuitively see the advantages of the improved IMM-MADRL algorithm, this paper compares it with the MM-MADRL algorithm and obtains the corresponding curves. The parameters are optimized, and the image is as follows:

Figures 6, 7 and 8 show that the green and brown curves in the figure mainly describe the state of a single optimization, while the blue and pink curves mainly describe the average curve state after multiple optimizations. The improved IMM-MADRL algorithm has better performance than the original MM-MADRL algorithm in the optimization environment of each dimension. It can clearly be seen that

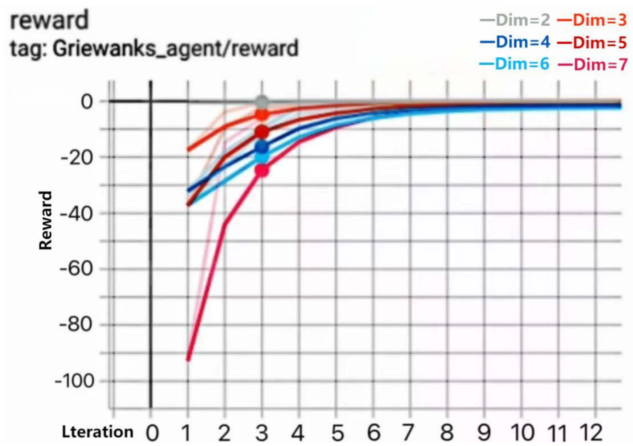

FIGURE 3. Griewank function convergence curve.

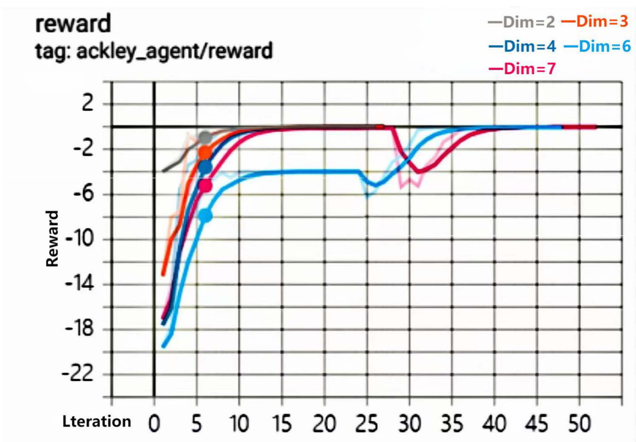

FIGURE 4. Ackley function convergence curve.

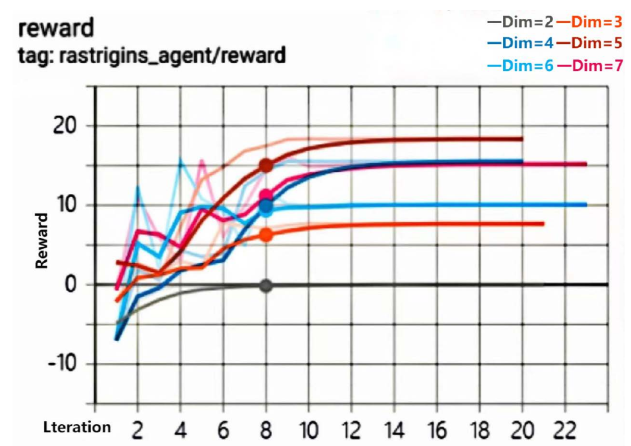

FIGURE 5. Rastrigin function convergence curve.

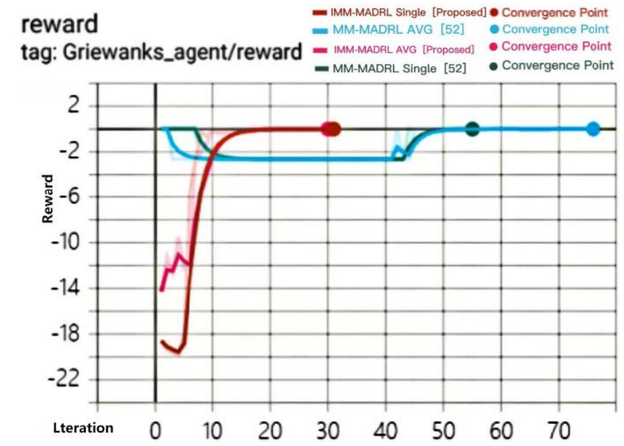

FIGURE 6. Griewank function contrast curve.

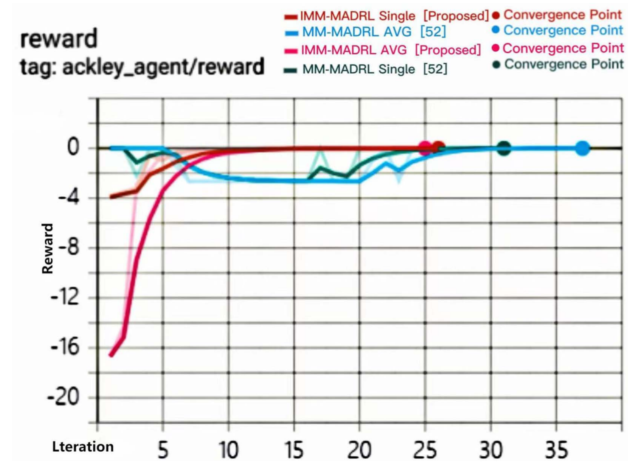

FIGURE 7. Ackley function contrast curve.

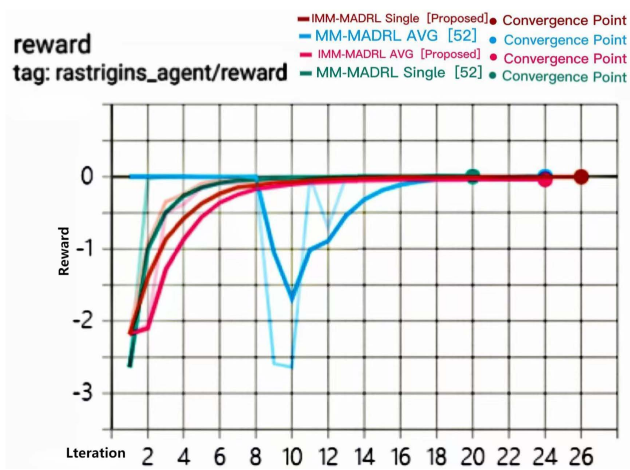

FIGURE 8. Rastrigin function contrast curve.

IMM-MADRL's convergence speed is faster and the accuracy rate is guaranteed with higher efficiency. After using the new initialization method designed in this paper, the initial position of the algorithm optimization is very far from the optimal position, while the original MM-MADRL algorithm sometimes randomly selects the initial point close to the optimal position at the beginning of the initialization. In this way, the shrinking of the range needs to be carried out quickly; otherwise, the algorithm will ignore the real optimal position, causing the algorithm to fall into a local optimal solution.

A random distribution can make the shrinking area of the range smooth, and the shrinking of the range can force a shrinkage coefficient to adjust according to different search space ranges, which would improve the algorithm. Moreover, changing the control system environment can make the algorithm more suitable.

The new design method of the somersault operation can be seen from the experimental results, and it has achieved good performance. The specific comparison results of this step are as follows: the threshold of the reward is set to -1, and the number of steps of the flip operation is approximately 50-200 steps with dim=4, 5 and 6. The results are as follows:

The values in Tables 3, 4 and 5 are the reward values. The smaller the value of the reward is, the better the performance will be. It can be seen from the tables that the steps of the somersault operation are well adapted to multidimensional parameter optimization, which can make the algorithm more accurate. As the search dimension increases, the effect of the somersault operation is more obvious, and the global optimal solution can be found more ideally. Thus, the algorithm has been substantially improved.

TABLE 3. Somersault operation effect data with dim =4.

Function name	Before Somersault	After Somersault
Griewanks	-0.26	-0.06
Rastrigins	-0.01	-0.006
Ackley	-0.18	-0.049

TABLE 4. Somersault operation effect data with dim =5.

Function name	Before Somersault	After Somersault
Griewanks	-0.09	-0.09
Rastrigins	-0.0029	-0.0001
Ackley	-0.09	-0.0149

TABLE 5. Somersault operation effect data with dim =6.

Function name	Before Somersault	After Somersault
Griewanks	-0.74	-0.38
Rastrigins	-0.43	-0.034
Ackley	-0.097	-0.024

B. CONTROL SYSTEM SIMULATION

Based on the first part, this paper describes an incomplete differential PID controller for the simulation of this part. The code of the algorithm's main function is as follows:

From Python example code 1, it can be seen that the incomplete differential PID controller used in this article contains 5 parameters. This controller's structure combines the 2 structure diagrams in the first part of this article and adds 2 low-pass filters. After the simulation test and verification, the five parameters can run normally.

The transfer function is as follows:

Python Example Code 1 Incomplete Differential PID Controller

```
error = pid.set_val - out_now
Res = pid.Kp ∗ (error - pid.error_last) + pid.Ki ∗ error +\
   pid.Kd ∗ (error - 2 ∗ pid.error_last + pid.error_prev) +
   pid..lam ∗ math..fabs(error - pid.error_last) +
   pid..mu ∗ math..fabs(error - 2 ∗ pid.error_last + pid.error_prev)
   pid.error_prev = pid.error_last
   pid.error_last = error
   Return Res
```

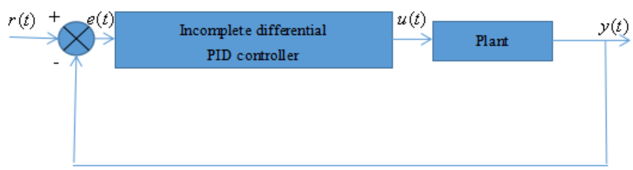

FIGURE 9. Control diagram.

Python Example Code 2 First-Order Relay System Transfer Function

```
SystemFunc = lambda x: 5 ∗ x + np.random.normal (0, 0.5, 1)[0]
```

From Python example code 2, the edited formula is converted into a mathematical formula as follows:

$$G(s) = Ks + b \tag{9}$$

Note that K is time constant, b is noise value.

It is similar to a PWM speed control system in that it is a simple linear control system. Simultaneously, Gaussian noise is introduced to the transfer function, causing the system to become unstable and forming a linear unstable system. The following is its control principle:

The search range of the controller parameters is (0,1), n = 5, setpoint = 100 and the operation results are as follows

For more forensics, next change the target and set the target to 200 and 400. The results are as follows:

Figures 11 and 12 show that using the IMM-MADRL algorithm to change the setting height of the PID system with incomplete differentiation of 5 parameters can also obtain the optimal parameter combination, and the simulation curve shows that the effect is good.

The reward part's judgment requirements were then adjusted, and ISE, ITAE, and ITSE [53] were employed to make the control system operate at setpoint=200, with the following results:

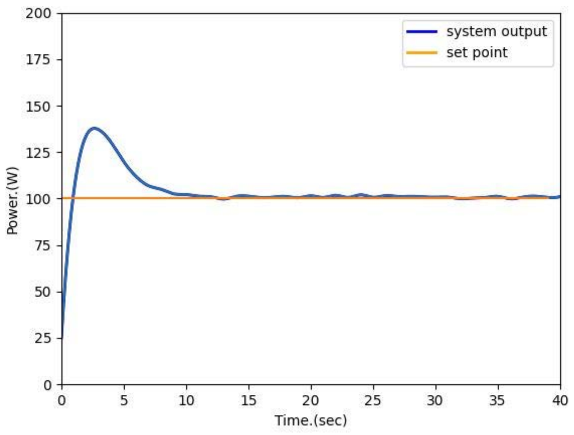

FIGURE 10. Incomplete differential PID controller result.

TABLE 6. Result data with setpoint=100.

Parameter name	Result
k_p	0.010622
k_i	0.104101
k_d	0.022496
lam	0.010879
Mu	0.063705
Calculate time	14.24s
Rise time	3s
Settling time	11s
Overshoot	38.19%

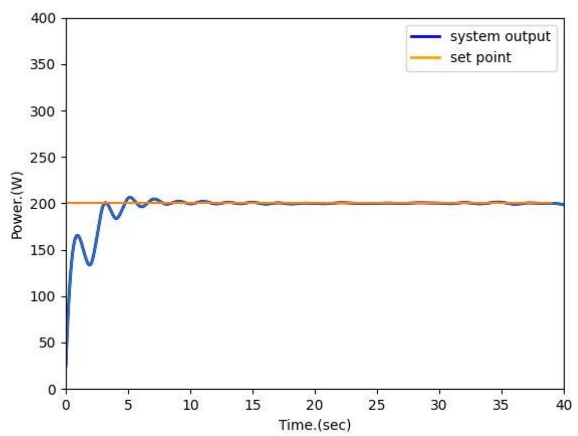

FIGURE 11. Result with setpoint = 200.

Figures 13-15 depict the control system's output curves when ITSE, ISE, and ITAE are utilized as rewards. The data findings are shown in Tables 9, 10, and 11. The results reveal that when different indicator functions are employed as rewards, the derived parameters are varied, as are the computation time and overshoot.

TABLE 7. Result data with setpoint=200.

Parameter name	Result
k_p	0.014355
k_i	0.108231
k_d	0.029522
lam	0.008564
Mu	0
Calculate time	14.49s
Rise time	5s
Settling time	7s
Overshoot	3.40%

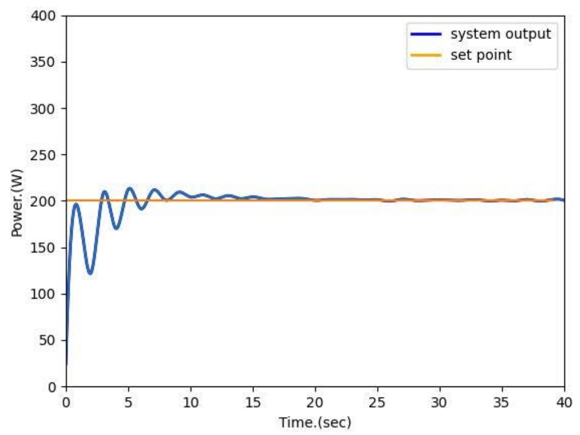

FIGURE 12. Result with setpoint = 400.

TABLE 8. Result data with setpoint=400.

Parameter name	Result
k_p	0.082212
k_i	0.056675
k_d	0.019567
lam	0.029578
Mu	0.002341
Calculate time	15.38s
Rise time	5s
Settling time	17s
Overshoot	7.67%

In the comprehensive simulation, the figure shows that the improved IMM-MADRL algorithm in this paper can be applied to the parameter setting of the control system in a multiparameter environment, and the effect is better than that of the MM-MADRL algorithm. The figure shows that the IMM-MADRL algorithm is used for parameter tuning of the actual PID control system with incomplete differentiation of 5 parameters, and good results have been achieved.

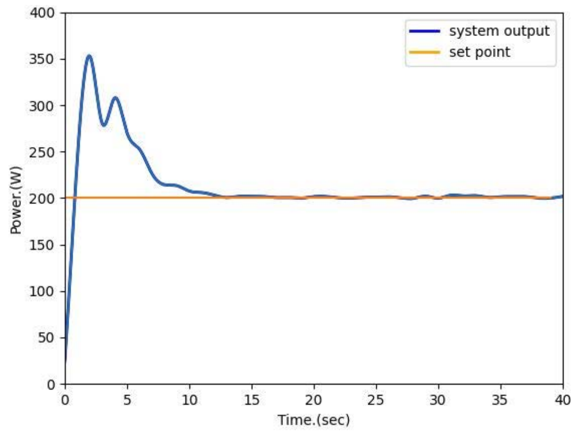

FIGURE 13. Result with reward for ITSE.

TABLE 9. Result data with reward for ITSE.

Parameter name	Result
k_p	0.021371
k_i	0.139186
k_d	0.0
lam	0.007969
Mu	0.078949
Calculate time	12.20s
Rise time	2s
Settling time	11s
Overshoot	75.82%

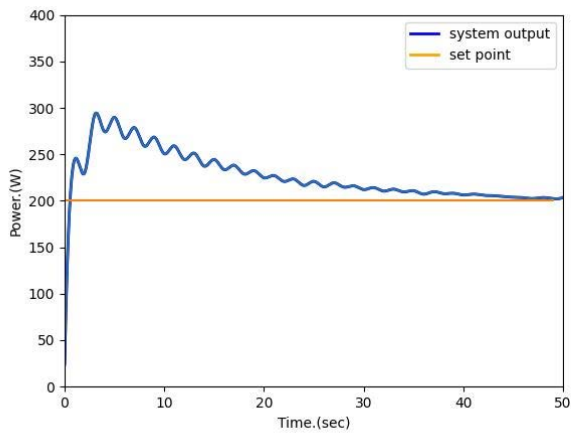

FIGURE 14. Result with reward for ISE.

TABLE 10. Result data with reward for ISE.

Parameter name	Result
k_p	0.055159
k_i	0.046385
k_d	0.037686
lam	0.099031
Mu	0.0112861
Calculate time	13.37s
Rise time	3s
Settling time	44s
Overshoot	45.68%

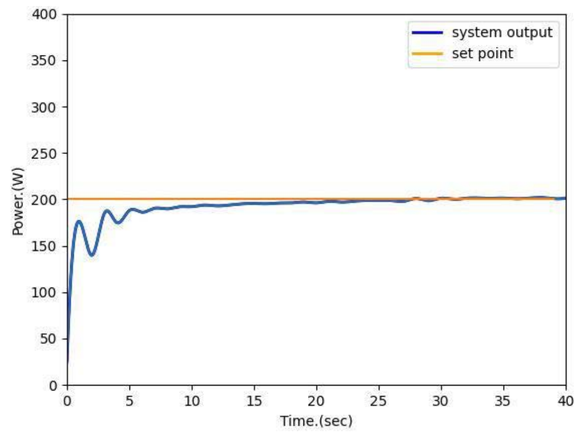

FIGURE 15. Result with reward for ITAE.

TABLE 11. Result data with reward for ITAE.

Parameter name	Result
k_p	0.092007
k_i	0.017972
k_d	0.009105
lam	0.045001
Mu	0.007455
Calculate time	14.73s
Rise time	16s
Settling time	20s
Overshoot	0.96%

The IMM-MADRL proposed in this research combines the benefits of the monkey group algorithm. According to the literature [52], the original MM-MADRL algorithm is suitable for 3-dimensional nonlinear PID due to the functioning of the somersault step. The IMM-MADRL algorithm in this study is a further refinement of the MM-MADRL algorithm, which broadens the algorithm's application area and allows it to optimize the control system with additional parameters. Because the IMM-MADRL algorithm simultaneously blends the somersault operation with the concept of reinforcement learning, the control system's parameter self-tuning produced good results; it is difficult to fall into the local optimal solution, and it can also converge faster when compared to the MM-MADRL algorithm and the traditional Actor-Critic algorithm. According to the comparison curve in Figure 6-8 of this work, the IMM-MADRL algorithm has more advantages than the MM-MADRL algorithm, thus when improving the control system, the advantage is that the ideal solution can be discovered more accurately, and the convergence time is also enhanced.

VI. CONCLUSION AND DISCUSSION

In this paper, a new IMM-MADRL algorithm is proposed based on the MM-MADRL algorithm. This new algorithm focuses on the somersault step to design the somersault operation based on reinforcement learning, and good results are achieved. In addition, the environment is replanned and designed so that the algorithm can be applied. For more control system parameter tuning, our system has made a certain contribution to the swarm intelligent learning algorithm for controller parameter optimization and makes the use of the algorithm more diverse. However, because the algorithm is based on Python for code editing and simulation and there is very little Python-related controller code, such as some QFT-based controllers and fractional-order PID controllers, the Python codes of these controllers have no relevant literature.

Hence, in this paper, based on both PID control theory and incomplete differentiation, a similar control system with 5 parameters has been written, the algorithm has been tested and simulated, and good results have also been achieved. It is proven that the IMM-MADRL algorithm can achieve good results in a control system environment with more parameters.

In future work, researchers can use Python to edit the code of more complex control systems, such as the CRONE controller mentioned above, so that the test of the algorithm can be more diversified. Furthermore, in the algorithm itself, some new ideas for agent action and position updating can be set to further improve the algorithm itself.

Finally, researchers have read and understood more operation modes of the swarm intelligence algorithm, and researchers have summarized some concepts, which can be combined into the algorithm proposed in this article. The proper use and combination can also improve the working effect of the algorithm itself.

REFERENCES

[1] *The Control Handbook*, PID Control, IEEE Press, Piscataway, NJ, USA, 1996, pp. 198–209.

[2] A. Hoyo, J. C. Moreno, J. L. Guzman, and F. Rodriguez, "Robust QFT-based feedback linearization controller of the greenhouse diurnal temperature using natural ventilation," *IEEE Access*, vol. 7, pp. 64148–64161, 2019.

[3] P. S. Rao and I. Sen, "Robust tuning of power system stabilizers using QFT," *IEEE Trans. Control Syst. Technol.*, vol. 7, no. 4, pp. 478–486, Jul. 1999.

[4] J. Z. Shi, "A fractional order general type-2 fuzzy PID controller design algorithm," *IEEE Access*, vol. 8, pp. 52151–52172, 2020.

[5] G. Hu, X. Q. Fu, M. Y. Nie, and H. Li, "Temperature control system of chamber electromechanical based on incomplete differential PID algorithm," *Adv. Mater. Res.*, vols. 1044–1045, pp. 885–888, Oct. 2014.

[6] A. Morand, X. Moreau, P. Melchior, M. Moze, and F. Guillemard, "CRONE cruise control system," *IEEE Trans. Veh. Technol.*, vol. 65, no. 1, pp. 15–28, Jan. 2016.

[7] J. Pongfai, X. Su, H. Zhang, and W. Assawinchaichote, "A novel optimal PID controller autotuning design based on the SLP algorithm," *Expert Syst.*, vol. 37, no. 2, pp. 1–15, Apr. 2019.

[8] M. Jamil, A. Waris, S. O. Gilani, B. A. Khawaja, M. N. Khan, and A. Raza, "Design of robust higher-order repetitive controller using phase lead compensator," *IEEE Access*, vol. 8, pp. 30603–30614, 2020.

[9] W. Assawinchaichote, "A non-fragile H_∞ output feedback controller for uncertain fuzzy dynamical systems with multiple time-scales," *Int. J. Comput., Commun. Control*, vol. 7, no. 1, pp. 8–19, 2012.

[10] N. Kaewpraek and W. Assawinchaichote, "H_∞ Fuzzy state-feedback control plus state-derivative-feedback control synthesis for photovoltaic systems," *Asian J. Control*, vol. 18, no. 4, pp. 1441–1452, Jul. 2016.

[11] J. Pongfai, W. Assawinchaichote, P. Shi, and X. Su, "Novel D-SLP controller design for nonlinear feedback control," *IEEE Access*, vol. 8, pp. 128796–128808, 2020.

[12] A. Sungthong and W. Assawinchaichote, "Particle swam optimization based optimal PID parameters for air heater temperature control system," *Proc. Comput. Sci.*, vol. 86, pp. 108–111, Jan. 2016.

[13] S. Ruangsang and W. Assawinchaichote, "A novel robust H_∞ fuzzy state feedback plus state-derivative feedback controller design for nonlinear time-varying delay systems," *Neural Comput. Appl.*, vol. 36, pp. 6303–6318, Oct. 2019.

[14] J. Günther, E. Reichensdörfer, P. M. Pilarski, and K. Diepold, "Interpretable PID parameter tuning for control engineering using general dynamic neural networks: An extensive comparison," 2019, *arXiv:1905.13268*.

[15] J. Fišer and P. Zítek, "PID controller tuning via dominant pole placement in comparison with ziegler-nichols tuning," *IFAC-PapersOnLine*, vol. 52, no. 18, pp. 43–48, 2019.

[16] N. P. Putra, G. J. Maulany, F. X. Manggau, and P. Betaubun, "Attitude quadrotor control system with optimization of PID parameters based on fast genetic algorithm," *Int. J. Mech. Eng. Technol.*, vol. 10, no. 1, pp. 335–343, 2019.

[17] J. Xu, "An expert PID control algorithm based on anti-integration saturation," in *Proc. IEEE 2nd Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Mar. 2017, pp. 1536–1539.

[18] F. Kang and Y. B. Liang, "Research on modeling and simulation of expert_PID controlled servo system based on MATLAB/S-function," *Appl. Mech. Mater.*, vols. 347–350, pp. 604–609, Aug. 2013. [Online]. Available: https://www.scientific.net/Home/Contacts

[19] B. Zhou, S. Xie, and J. Hui, "H_∞ control for T-S aero-engine wireless networked system with scheduling," *IEEE Access*, vol. 7, pp. 115662–115672, 2019.

[20] M. Farahani, S. Ganjefar, and M. Alizadeh, "Intelligent control of SSSC via an online self-tuning PID to damp the subsynchronous oscillations," in *Proc. 20th Iranian Conf. Electr. Eng. (ICEE)*, May 2012, pp. 336–341.

[21] I. Carlucho, M. De Paula, S. A. Villar, and G. G. Acosta, "Incremental Q-learning strategy for adaptive PID control of mobile robots," *Expert Syst. Appl.*, vol. 80, pp. 183–199, Sep. 2017.

[22] A. G. Alexandrov and M. V. Palenov, "Adaptive PID controllers: State of the art and development prospects," *Autom. Remote Control*, vol. 75, no. 2, pp. 188–199, Feb. 2014.

[23] Y. Liao, L. Wang, Y. Li, Y. Li, and Q. Jiang, "The intelligent control system and experiments for an unmanned wave glider," *PLoS ONE*, vol. 11, no. 12, Dec. 2016, Art. no. e0168792.

[24] Z. Jing, "Application and study of expert PID intelligent control," in *Proc. IOP Conf. Mater. Sci. Eng.*, vol. 563, no. 4, Jul. 2019, Art. no. 042084.

[25] C. Vorrawan, W. Assawinchaichote, Y. Shi, and X. Su, "Fuzzy-modeled prescribed performance integral controller design for nonlinear descriptor system with uncertainties," *IEEE Access*, vol. 8, pp. 89520–89533, 2020.

[26] S. Ruangsang and W. Assawinchaichote, "Control of nonlinear Markovian jump system with time varying delay via robust H_∞ fuzzy state feedback plus state-derivative feedback controller," *Int. J. Control, Autom. Syst.*, vol. 17, no. 9, pp. 2414–2429, 2019.

[27] F.-J. Lin, H.-J. Shieh, L.-T. Teng, and P.-H. Shieh, "Hybrid controller with recurrent neural network for magnetic levitation system," *IEEE Trans. Magn.*, vol. 41, no. 7, pp. 2260–2269, Jul. 2005.

[28] V. Kachitvichyanukul, "Comparison of three evolutionary algorithms: GA, PSO, and DE," *Ind. Eng. Manage. Syst.*, vol. 11, no. 3, pp. 215–223, Sep. 2012.

[29] E. Anene and G. K. Venayagamoorthy, "PSO tuned flatness based control of a magnetic levitation system," in *Proc. IEEE Ind. Appl. Soc. Annu. Meeting*, Oct. 2010, pp. 1–5.

[30] G. Chen, Z. Li, Z. Zhang, and S. Li, "An improved ACO algorithm optimized fuzzy PID controller for load frequency control in multi area interconnected power systems," *IEEE Access*, vol. 8, pp. 6429–6447, 2020.

[31] S. Kansit and W. Assawinchaichote, "Optimization of PID controller based on PSOGSA for an automatic voltage regulator system," *Proc. Comput. Sci.*, vol. 86, pp. 87–90, Jan. 2016.
[32] S. Kiong Nguang, W. Assawinchaichote, P. Shi, and Y. Shi, "H_∞ fuzzy filter design for uncertain nonlinear systems with Markovian jumps: An LMI approach," in *Proc., Amer. Control Conf.*, 2005, pp. 1799–1804.
[33] S. Panda, B. Mohanty, and P. K. Hota, "Hybrid BFOA–PSO algorithm for automatic generation control of linear and nonlinear interconnected power systems," *Appl. Soft Comput.*, vol. 13, no. 12, pp. 4718–4730, Dec. 2013.
[34] K. Premkumar and B. V. Manikandan, "Fuzzy PID supervised online ANFIS based speed controller for brushless DC motor," *Neurocomputing*, vol. 157, pp. 76–90, Jun. 2015.
[35] R. K. Sahu, S. Panda, and N. K. Yegireddy, "A novel hybrid DEPS optimized fuzzy PI/PID controller for load frequency control of multi-area interconnected power systems," *J. Process Control*, vol. 24, no. 10, pp. 1596–1608, Oct. 2014.
[36] K. Premkumar and B. V. Manikandan, "Bat algorithm optimized fuzzy PD based speed controller for brushless direct current motor," *Eng. Sci. Technol., Int. J.*, vol. 19, no. 2, pp. 818–840, Jun. 2016.
[37] L. J. Ke and X. Q. Wang, *Reinforcement Learning*, 1st ed. Beijing, China: Tsinghua Univ. Press, 2019, pp. 1–176.
[38] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. ICML*, Jun. 2014, pp. 387–395.
[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. ICLR*, 2016, pp. 1–14.
[40] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2017, *arXiv:1706.02275*.
[41] B. Luo, Y. Yang, and D. Liu, "Adaptive Q-learning for data-based optimal output regulation with experience replay," *IEEE Trans. Cybern.*, vol. 48, no. 12, pp. 3337–3348, Apr. 2018.
[42] C.-F. Juang, "Combination of online clustering and Q-value based GA for reinforcement fuzzy system design," *IEEE Trans. Fuzzy Syst.*, vol. 13, no. 3, pp. 289–302, Jun. 2005.
[43] Q. Wei, L. F. Lewis, Q. Sun, P. Yan, and R. Song, "Discrete-time deterministic Q -learning: A novel convergence analysis," *IEEE Trans. Cybern.*, vol. 47, no. 5, pp. 1224–1237, May 2017.
[44] H. Mao, Z. Zhang, Z. Xiao, and Z. Gong, "Modelling the dynamic joint policy of teammates with attention multi-agent DDPG," *arXiv:1811.07029v1*, 2018.
[45] E. Wei, D. Wicke, D. Freelan, and S. Luke, "Multiagent soft Q-learning," *arXiv:1804.09817*, 2018.
[46] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.
[47] R. E. Wang, M. Everett, and J. P. How, "R-MADDPG for partially observable environments and limited communication," 2020, *arXiv:2002.06684*.
[48] J. Han, C.-H. Wang, and G.-X. Yi, "Cooperative control of UAV based on multi-agent system," in *Proc. IEEE 8th Conf. Ind. Electron. Appl. (ICIEA)*, Jun. 2013, pp. 96–101.
[49] K. Shao, Y. Zhu, and D. Zhao, "StarCraft micromanagement with reinforcement learning and curriculum transfer learning," *IEEE Trans. Emerg. Topics Comput. Intell.* vol. 3, no. 1, pp. 73–84, Feb. 2019.
[50] J. E. Summers, J. M. Trader, C. F. Gaumond, and J. L. Chen, "Deep reinforcement learning for cognitive sonar," *J. Acoust. Soc. Amer.*, vol. 143, no. 3, p. 1716, Apr. 2018.
[51] J. Pongfai, X. Su, H. Zhang, and W. Assawinchaichote, "PID controller autotuning design by a deterministic Q-SLP algorithm," *IEEE Access*, vol. 8, pp. 50010–50021, 2020.
[52] H. Zhang, W. Assawinchaichote, and Y. Shi, "New PID parameter autotuning for nonlinear systems based on a modified monkey–multiagent DRL algorithm," *IEEE Access*, vol. 9, pp. 78799–78811, 2021.
[53] Y. J. Huo, "Auto-turning of optimum PID controller parameters based on quantum-behaved particle swarm optimization," *Microelectron. Comput.*, vol. 29, no. 10, pp. 195–197, 2012.

HONGMING ZHANG was born in Kunming, Yunnan, China, in 1993. He received the B.S. degree in information and communication engineering from the University of Mea Fah Luang, Chiang Rai, Thailand, in 2018, and the M.S. degree in electronic and telecommunication engineering from the King Mongkut's University of Technology Thonburi, Bangkok, Thailand, in 2020, where he is currently pursuing the Ph.D. degree. From 2018 to 2020, he began research on PID control systems, mainly in the aspects of intelligent algorithms and neural networks, to optimize the PID system.

WUDHICHAI ASSAWINCHAICHOTE received the B.S. degree (Hons.) in electrical engineering from Assumption University, Bangkok, Thailand, in 1994, the M.E. degree in electrical engineering from Pennsylvania State University (Main Campus), University Park, PA, USA, in 1997, and the Ph.D. degree in electrical engineering from The University of Auckland, New Zealand, in 2004. He is currently an Associate Professor at the Department of Electronic and Telecommunication Engineering, King Mongkut's University of Technology Thonburi (KMUTT), Bangkok. He has published a research monograph and more than 20 research articles in international refereed journals indexed by SCI/SCIE (Clarivate Analytics). His current research interests include fuzzy control, robust control, optimal control, system and control theory, computational intelligence, and PID controller design. He also serves as an Associate Editor for the *International Journal of Innovative Computing, Information and Control*, and serves as a Reviewer for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON FUZZY SYSTEMS, the IEEE TRANSACTIONS ON CYBERNETICS, *Neural Computing and Applications*, and IEEE ACCESS.

YAN SHI received the Ph.D. degree in information and computer sciences from Osaka Electro-Communication University, Neyagawa, Japan, in 1997. He is currently a full-time Professor with the Graduate School of Science and Technology, Tokai University, Kumamoto, Japan. His research interests include fuzzy reasoning and data mining.