Lightweight Imitation Learning Algorithm with Error Recovery for Human Direction Correction

Mingchi Zhu[1], Haoping She[1,*], Weiyong Si[2], Chuanjun Li[1]

***Abstract*— Existing imitation learning methods for human directional corrections may lead to learning incorrect behaviors due to erroneous artificial teaching, resulting in a significant increase in the required number of iterations and even non-convergence situations, which can affect the system's performance. Additionally, the high computational complexity makes it unsuitable for embedded real-time application scenarios. To address these two issues, this study proposes a lightweight imitation learning algorithm that pre-corrects human-directed corrections. This method utilizes a deep learning network trained on a small dataset to correct human directional corrections and designs a lower-dimensional cost function for imitation learning. The proposed approach is applied to the example of a drone passing through doorways. Through the construction of a simulation platform and conducting simulation verification, the results show that the algorithm incorporating the correction error detection mechanism achieves an accuracy of over 98% in discerning human corrections, reduces training time by 27.87% per iteration, and decreases the average number of rounds by approximately 40%. The results indicate that the algorithm, which combines correction detection based on deep learning and a low-dimensional cost function, improves the accuracy of algorithm iterations, reduces computational complexity, and enhances computational speed.**

***Index Terms*— Learning from demonstrations (LfD), cost function design, lightweight network, error recovery for human correction, small-dataset neural network**

I. Introduction

In today's rapidly modernizing world, intelligent robots are increasingly used in different industries. These robots need to learn how to perform tasks efficiently. Since tasks vary in different situations, traditional reinforcement learning algorithms usually require a lot of trial-and-error learning, with insufficient learning efficiency and robustness.

To address these challenges, imitation learning [1] has been developed. Imitation learning, also known as Learning by Demonstration [2,3] (LfD), refers to the fact that robots learn better strategies faster by obtaining effective guidance and feedback from human experts to quickly respond to different environments and demands [4,5]. This enables closer human-robot collaboration for more efficient cooperation and interaction. Inverse Reinforcement Learning [6,7] (IRL) is a common method in imitation learning, where robots introduce knowledge and feedback from human experts into reinforcement learning algorithms based on observed behaviours, in order to accelerate the learning process and improve the learning efficiency and performance of intelligent systems.

To make IRL more accessible and adaptable for non-experts, a method of learning from human directional corrections is suggested [8]. This learning approach only requires user direction corrections and has strict convergence. Direction correction is a correction involving only directional information and does not necessarily need to be specific to a magnitude [9]. This enables non-expert users to provide effective demonstrations of the robot [10].

However, assuming non-expert user corrections are always accurate can lead to problems if incorrect corrections are not considered [11]. These errors can impact system performance, leading to inefficiencies and affecting user trust and satisfaction. Moreover, most imitation learning algorithms use high-dimensional cost functions [12], which can be computationally complex and prone to overfitting.

To address these challenges, this paper proposes a lightweight imitation learning algorithm that recover human directional corrections using deep learning [13]. Based on user input corrections and data such as the interval time between different corrections, a deep neural network adapted for small datasets [14] is used to determine the accuracy of human directional corrections and eliminate error corrections in advance. Deep neural networks based on small datasets reduce overfitting risks and improves generalization. Additionally, due to the simplicity and fewer parameters of the network, it is easier to interpret and understand, and is more resource-efficient, making it suitable for deployment on devices with limited space. Also, it uses low-dimensional cost functions to compute device trajectories, making optimization easier and more efficient. This cost function is simpler, reducing the risk of overfitting and helping the model converge to better solutions. By incorporating filtered human corrections, the robot can plan optimal trajectories while filtering out erroneous inputs and reducing the learning network's complexity.

A. Related Work

This paper considers the scenario of quadrotor UAVs performing flights in indoor environments, with a focus on the UAV's flight trajectory. The UAV needs to learn to navigate around fixed obstacles and complete transportation tasks.

1 M. Zhu, H. She and C. Li are with the School of Aerospace Engineering, Beijing Institute of Technology.

2W. Si is with the School of Computer Science and Electronic Engineering, University of Essex.

*Corresponding author is H. She (Email: shehp@bit.edu.cn)

The aim of this study is to transfer flight strategies to multi-rotor UAVs based on optimized processing of human corrections, enabling the UAVs to autonomously plan flight trajectories. Initially, when the algorithm receives user-input corrections, it preprocesses the human corrections using a trained deep neural network to obtain corrections that approximate the user's true expectations. Subsequently, during the imitation learning phase, after the UAV receives the desired human corrections, it iterates parameters in the cost function and solves for the optimal trajectory based on the current cost function to ultimately achieve convergence towards the desired flight trajectory. The overall algorithm overview diagram is shown in Figure 1.

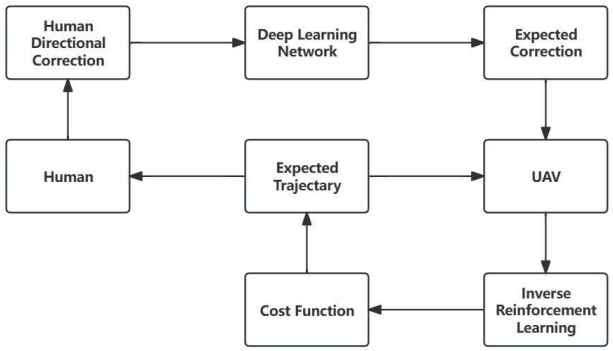

Figure 1 Algorithm flow

B. Contributions

The main contributions of this paper are as follows:

1) Based on user input teaching data, for inputs that are very close in time and seem like possible errors, a deep neural network based on a small dataset is used to filter them, ensuring that the input corrections are as close as possible to the user's expected corrections, while also reducing incorrect iterations.

2) By using a low-dimensional cost function, on the basis of stable convergence in simple scenarios like the example in Part IV, a significant reduction in computational complexity is achieved, resulting in increased iteration speed, lightweight network implementation, and suitability for deployment on small devices or devices with limited storage space, such as drones.

II. FORMULATION

The proposed framework aims to iterate UAV cost functions by human corrections to plan trajectories. Therefore, it is necessary to discuss the modeling issues of UAV dynamics, cost function settings, and the expectation handling of input corrections.

A. UAV Dynamics

Considering the dynamics equation of a quadrotor UAV as follows:

$$\begin{aligned} \dot{r}_I &= v_I \\ m\dot{v}_I &= mg_I + f_I \\ \dot{q}_{B/I} &= \frac{1}{2}\Omega(\omega_B)q_{B/I} \\ J_B\dot{\omega}_B &= \tau_B - \omega_B \times J_B\omega_B \end{aligned} \tag{1}$$

Here, the subscripts I and B represent the parameters of the quadrotor UAV in the body and Earth frames, respectively. m is the UAV mass, r and v are the UAV's position and velocity, J is the moment of inertia, q is the quaternion, $\Omega(\omega_B)$ is the matrix form of quaternion multiplication, τ is the UAV's torque, and f is the total force applied at the UAV's center of mass.

The total force f and the torque τ applied at the UAV's center of mass are generated by the thrust provided by the four rotors $T = [T_1 \quad T_2 \quad T_3 \quad T_4]^{\mathrm{T}}$.

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -l/2 & 0 & l/2 \\ -l/2 & 0 & l/2 & 0 \\ \kappa & -\kappa & -\kappa & -\kappa \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \tag{2}$$

Defining the state vector of the quadrotor UAV as:

$$x = [r_I \quad v_R \quad q_{B/I} \quad \omega_B]^{\mathrm{T}} \tag{3}$$

The control input is:

$$u = [T_1 \quad T_2 \quad T_3 \quad T_4]^{\mathrm{T}} \tag{4}$$

Thus, the state matrix of the quadrotor UAV can be represented as:

$$\begin{aligned} x(t+1) &= f(x(t), u(t)) \\ x(0) &= x_0 \end{aligned} \tag{5}$$

where $t = 1, 2...$ denotes the time step, and time interval $\Delta = 0.1s$.

B. Human Directional Corrections

In the algorithm, we use direction corrections as input corrections. A notable feature of methods based on such corrections[8] is the assumption that each correction satisfies:

$$\left\langle -\nabla J(u_{0:T}^{\theta_k}, \theta^*), a_k \right\rangle > 0, k = 1, 2, 3, \ldots \tag{6}$$

where,

$$a_k = [0^{\mathrm{T}} \quad \ldots \quad a_{t_k} \quad \ldots \quad 0^{\mathrm{T}}]^{\mathrm{T}} \in R^{m(T+1)} \tag{7}$$

$-\Delta J(u_{0:T}^{\theta_k}, \theta^*)$ represents the gradient descent around $J(\theta^*)$ of $u_{0:T}$ in the current UAV trajectory. Here, no specific magnitude is required for a_k , but the direction should roughly align with the gradient descent around $J(\theta^*)$. Algorithms based on this assumption can ensure strict convergence.

However, in practice, imperfect corrections are inevitable. Therefore, we process the direction corrections a_k provided by users into expected corrections $E(a_k)$, transforming a_k into $\overline{a_k} = E_{a_k}(a_k)$. $\overline{a_k}$ is the expected correction obtained after the human correction is processed by a trained deep neural network which will be introduced in the following chapter.

III. Main Algorithm

To enable the unmanned aerial vehicle (UAV) to plan its flight trajectory according to specific requirements in different scenarios, utilizing the concept of Inverse Reinforcement Learning (IRL) is a feasible and effective approach. However, the imperfection of human corrections and the complexity of the cost function limit the application of this method. The following will focus on using deep learning networks to preprocess human corrections and designing low-dimensional cost function.

A. Neural Network for Error Recovery of Human Directional Corrections

In this study, a deep neural network is employed to learn the accuracy of human corrections, filtering out erroneous input corrections in advance.

Like other commonly used deep neural networks, a deep neural network with input layer, multiple hidden layers, and output layer is used to learn the quality of human corrections, retaining correct inputs and filtering out human errors. Each layer of the neural network is fully connected to its adjacent layers.

When training the deep neural network, a dataset of human correction containing 288 data points is used. Every data includes its input state (up, down, left, right, forward, backward), the next input state, and the time interval between two inputs. Each one has a label indicating whether it is correct.

Principal Components analysis [15] (PCA) is utilized to preprocess the original input dataset, and normalization [16] is applied to the input and target variables before being fed into the network. The dataset is then randomly divided into training, validation, and testing sets in a 70%:15%:15% ratio to evaluate the performance of the neural network.

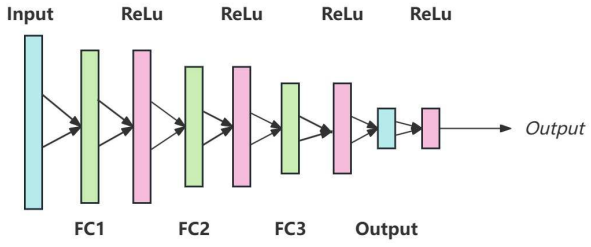

Figure 2 DNN model architecture

Due to the limited amount of correction of user input corrections in the demonstration learning scenario presented in this paper, it is a difficult challenge to collect and assemble large data sets. Therefore, a DNN network based on small data set [14] are chosen as the preprocessing neural network. The neural network structure is a 13-(5-4-3)-1 DNN with 3 hidden layers, trained with random initialization, SAE pre-training, and fine-tuning, like Figure 2. The maximum training epochs are set to 200. All computations are performed in Python. The DNN prediction accuracy is calculated as the indicator of network performance.

B. Low Dimensional Cost Function

Similar to other common cost function forms [17,18], the cost function of the quadrotor UAV is defined as follows:

$$J(u_{0:T},\theta)=\sum_{t=0}^{T}\theta^{T}\varphi(x(t),u(t))+h(x(T+1)) \tag{8}$$

where φ: $R^n\times R^m\to R^r$ is the basis function, $\theta\in R^r$ is the iterative weight vector, and $h(x(T+1)$ is the terminal index.

By seeking the minimum value of the cost function, the UAV plans the current optimal control sequence $u_{0:T}$ to obtain the optimal flight trajectory

$$\xi_\theta=\left\{x_{0:T+1}^{\theta},u_{0:T}^{\theta}\right\} \tag{9}$$

The error between the target attitude and the current attitude of the quadrotor UAV is defined as:

$$e\left(q,q^*\right)=\frac{1}{2}trace(I-R^{\mathrm{T}}(q^*)R(q^*)) \tag{10}$$

The final cost is set as:

$$h(x_{T+1})=10\left\|r_I-r_I^*\right\|^2+10\left\|v_I\right\|^2+10e(q_{B/I},q_{B/I}^*)+10\left\|\omega_B\right\| \tag{11}$$

The task of a quadcopter is to ultimately land at a target position. Here, denotes the position of the quadcopter in the world coordinate system. Considering that the algorithm needs to be deployed on UAVs with limited computational space, this study opts for a lower-dimensional cost function to reduce the number of iterative parameters. Thus, the weight-feature cost term is set as:

$$\begin{aligned}\phi&=\left[r_x^{\,2}\quad r_y^{\,2}\quad r_z^{\,2}\quad \|u\|^2\right]^{\mathrm{T}}\in R^4\\ \theta&=\left[\theta_1\quad\theta_2\quad\theta_3\quad\theta_4\right]\in R^4\end{aligned} \tag{12}$$

here, the cost function composed of the feature vector ϕ and general polynomial features θ will determine the specific trajectory of the quadcopter to the target. $r_I=\left[r_x,r_y,r_z\right]^{\mathrm{T}}$ represents the position of the quadcopter in the world coordinate system, and u represents the control input of the drone. The quadcopter's task is to ultimately land at the target position, so we choose the squared term r^2 representing the spatial position of the drone and the squared term $\|u\|^2$ representing the attitude control information of the drone as the feature vectors of the cost function. This cost function will determine the specific trajectory of the quadcopter drone to the target.

For the initialization of parameters in the cost function, different entries of θ are allowed to have their own lower and upper bounds, which can be derived from prior knowledge or simply initialized as:

$$\Omega_0=\left\{\theta\in R^r\left|\left\|\theta\right\|_\infty\le R\right.\right\} \tag{13}$$

where

$$R=\max\left\{\underline{c}_i,\bar{c}_i,i=1,...,r\right\} \tag{14}$$

By default [19], the initial weight search space is

$$\Omega_0 = \{\theta_1, \theta_2, \theta_3 \in [-8, 8], \theta_4 \in [0, 0.5]\} \quad (15)$$

IV. Simulation

In this section, a six-degree-of-freedom quadrotor maneuver simulation platform, as depicted in Figure 4, was established in a Python environment. A pre-trained deep neural network in Python was utilized to preprocess the input corrections. The simulation objective is to enable the unmanned aerial vehicle (UAV) to learn a cost functional by receiving preprocessed directional corrections from the deep neural network. The UAV needs to learn how to navigate from its initial position through two differently sized and positioned square frames and finally land successfully at the target location. Additionally, the UAV has no prior information about the flight scenario (square frames) and, therefore, cannot accomplish the task successfully without receiving manual corrections.

User directional corrections are input via the keyboard using the keys "up," "down," "W," "S," "A," and "D," representing ascent/descent, forward/backward, and left/right movement for the UAV. Multiple correction inputs are allowed during the iteration process.

In the constructed virtual environment, the initial position of the quadrotor UAV is located at $r_I^{(0)} = [-16 \quad 0 \quad 4]^T$ (the lower left corner), and the target position is at $r_I^* = [16 \quad 0 \quad 16]^T$ (the upper right corner), passing through pink and blue boxes along the way. The initial attitude of the UAV is $q_{B/I}^{(0)} = [1 \quad 0 \quad 0 \quad 0]^T$, while its target attitude is $q_{B/I}^* = [1 \quad 0 \quad 0 \quad 0]^T$. The time horizon for this simulation is $T = 50$, i.e., $T\Delta = 50s$.

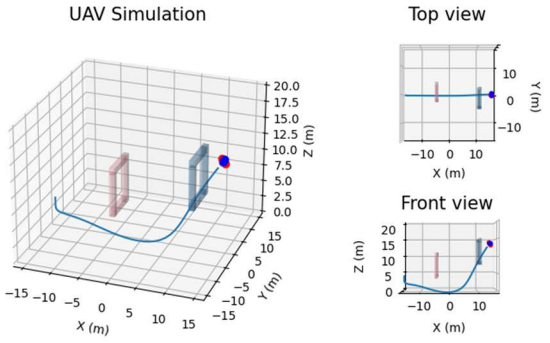

Figure 4 A six-degree-of-freedom quadrotor maneuver simulation platform

Here are the basic parameters for the quadcopter UAV and the constructed scenario.

Table 1 UAV parameters

UAV Parameters			
*Inertia(x/y/z) (kg*m^2)*	*Mass (kg)*	*Length (m)*	*Rotor Para*
1	1	1	1

Table 2 Scenario parameters

Scenario Parameters							
Region			*Obstacle*				
X	Y	Z	Thick	Height	Width	Reference Point	
(-16,16)	(-16,16)	(0,20)	1	6.5	6.5	P: (-5,-2.5,3.5)	B: (10,-5,7.5)

A. Neural Network for Error Recovery of Human Directional Corrections

In this section, the neural network described in the third section was constructed using Python and trained.

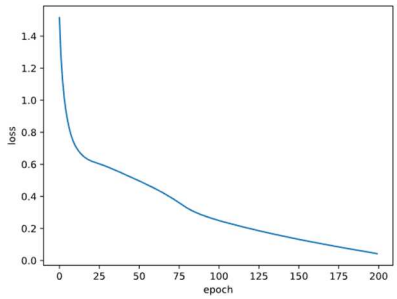

Figure 3 Iterative performance

Figure 3 shows the iterative performance of the deep neural network. It can be observed from the figure that the network has approached convergence after 200 iterations.

Table 3 presents the training results, indicating that the neural network performed well in this scenario.

Table 3 Neural network training results

Neural Network Training Results			
	Amount	*MSE*	*R*
Training	202	0.0080	0.9981
Test	86	0.0113	0.9885

Table 4 Training process demonstration with Error Recovery Neural Network

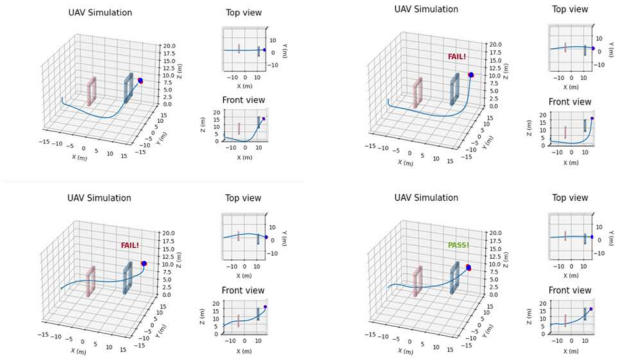

Table 4 illustrates the complete process of multiple rounds of training for the quadrotor UAV in a virtual environment to ultimately accomplish the task.

Table 5 displays the corrections provided by the user for each round and the time of correction.

Table 5 Corresponding correction information. The red correction in the third iteration was identified as an error input by the user due to negligence and was subsequently removed by the neural network.

Human Directional Corrections and the Time		
Iteration k	***Correction*** a_k	***Correction Time*** t_k
1	"W", "S"	17,31
2	"UP"	6
3	("W"), "S"	(23),26

To test the effectiveness of error recovery, we conducted a comparison between this algorithm and one without a recovery neural network in teaching drone flight trajectories in the same scenario. Table 6 shows the training process demonstration.

Table 7 shows the corrections of the algorithm without the error recovery neural network.

Table 6 Training process demonstration without error recovery neural network

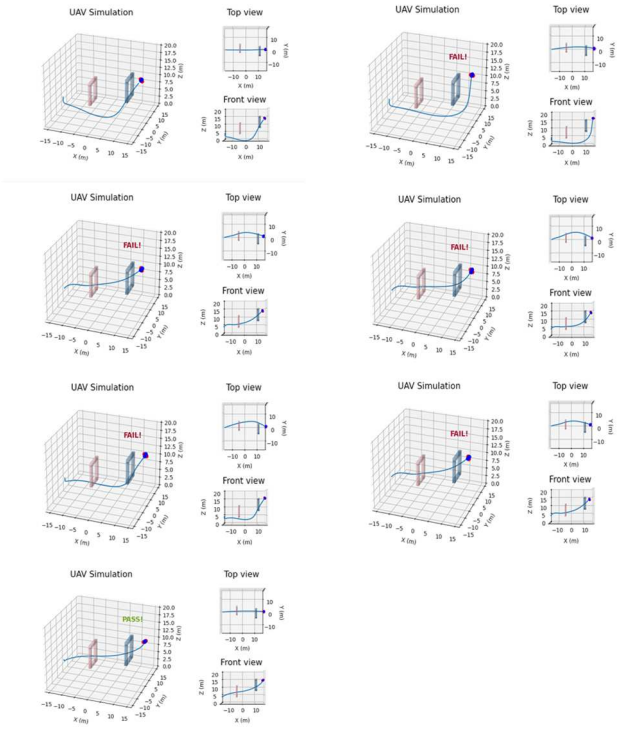

Table 7 Corresponding correction information. The red correction in the third iteration was an error which had not been deleted.

Human Directional Corrections and the Time		
Iteration k	***Correction*** a_k	***Correction Time*** t_k
1	"W", "S"	17,31
2	"UP"	6
3	"W", "S"	23,26
4	"S"	24
5	"UP", "S"	11,28
6	"S"	29

In a scenario where the initial human directional corrections were the same, the second experiment, which lacked the error recovery neural network, introduced an incorrect correction into the algorithm during the second iteration. This led the algorithm to learn incorrect behavior, necessitating more rounds of training compared to the algorithm with the error recovery neural network to rectify the error. The additional rounds of training also increased the risk of introducing more error corrections.

B. Low Dimensional Cost Function

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

To better evaluate the iterative performance of the low-dimensional cost function used in this paper, a model using a 7-dimensional cost function is introduced for reference. Both models run in the same environment.

Table 8 shows the average time per training cycle for the model using a 4-dimensional cost function and a 7-dimensional cost function. It is evident that in this training environment, the training time corresponding to the low-dimensional cost function is approximately half of that for the high-dimensional cost function.

Table 8 Average time per iteration of algorithm with four-dimension and seven-dimension cost function

Average Time per Iteration of Algorithm		
Four dimensions(4D)	***Seven dimensions(7D)***	***Ratio***
0.066	0.122	54.10%

Furthermore, with the inclusion of the error recovery neural network, the proposed model (4D+DNN) still requires less computation time compared to the model with the high-dimensional cost function without preprocessing in Table 9. The average time per training round is 0.088 s, approximately 72.13% of the time required for the high-dimensional cost function.

Table 9 Average time per iteration of algorithm with four-dimension cost function and error recovery neural network

Average Time per Iteration of Algorithm		
4D +DNN	***Seven dimensions(7D)***	***Ratio***
0.088	0.122	72.13%

Table 10 Training process demonstration with low dimensional cost function

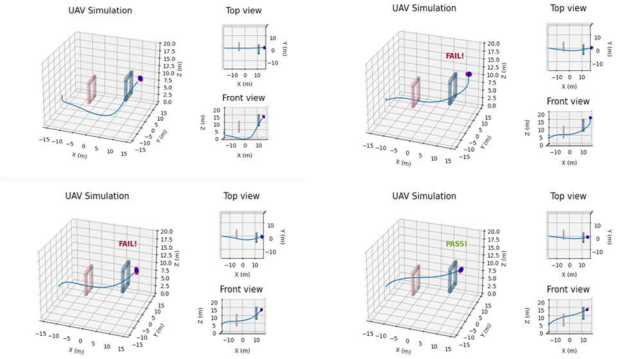

Table 10 and Table 11 illustrate the iteration results of flight trajectories completing the same task in the same scenario

under two types of cost functions. It is evident that when using a low-dimensional cost function, the trajectories are more sensitive to human directional corrections, leading to a significant reduction in the number of iterations.

Table 11 Training process demonstration with high dimension cost function

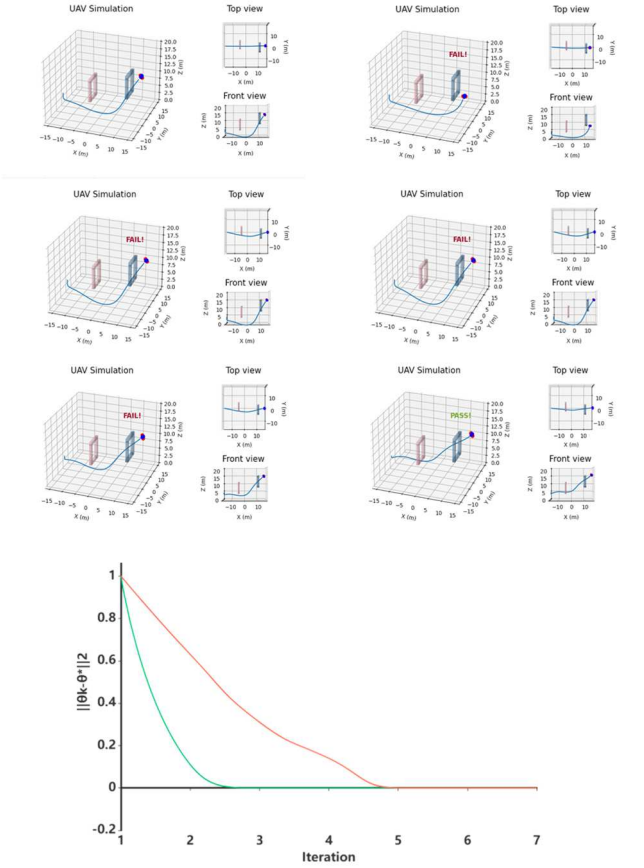

Figure 5 Relationship between parameter convergence and iteration rounds. The green line represents the proposed algorithm in this paper, while the red line represents the high-dimensional cost function algorithm.

Figure 5 depicts the relationship between the convergence of cost function parameters in two dimensions and the number of iterations. The data used in the graph has been standardized for comparison. It can be observed that the low-dimensional cost function requires fewer rounds to achieve convergence relatively. Overall, the model presented in this paper demonstrates promising path learning performance.

V. Conclusion

This paper has developed a lightweight imitation learning algorithm that utilizes neural networks for error recovery in human directional corrections. The algorithm is suitable for simple scenarios with high repetition and is designed to be deployed on intelligent devices with limited storage space, making it convenient for novice users to debug and use. The proposed method has been validated through simulation experiments with a quadrotor UAV. The results confirm the convergence of the proposed method and indicate that it can effectively filter error corrections, leading to faster iteration speeds and fewer iteration rounds. This study shows promise for practical applications in industrial transportation, such as logistics, and contributes to an in-depth understanding of human decision-making thinking and behavioural patterns, providing useful insights for the design and optimization of intelligent systems. In the future, We will continue to study the deep combination of reinforcement learning and human demonstration, consider the introduction of multimodal information fusion technology to achieve human-robot interaction (HRI) for more complex tasks, and further experimentally validate the algorithm effectiveness on actual UAV platforms.

References

[1] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR),* vol. 50, no. 2, pp. 1-35, 2017.

[2] W. Si, N. Wang, and C. Yang, "A review on manipulation skill acquisition through teleoperation - based learning from demonstration," *Cognitive Computation and Systems,* vol. 3, no. 1, pp. 1-16, 2021.

[3] Z. Jin, W. Si, A. Liu, W.-A. Zhang, L. Yu, and C. Yang, "Learning a flexible neural energy function with a unique minimum for globally stable and accurate demonstration learning," *IEEE Transactions on Robotics,* 2023.

[4] W. Si, Y. Guan, and N. Wang, "Adaptive compliant skill learning for contact-rich manipulation with human in the loop," *IEEE Robotics and Automation Letters,* vol. 7, no. 3, pp. 5834-5841, 2022.

[5] W. Si, N. Wang, Q. Li, and C. Yang, "A framework for composite layup skill learning and generalizing through teleoperation," *Frontiers in Neurorobotics,* vol. 16, p. 840240, 2022.

[6] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence,* vol. 297, p. 103500, 2021.

[7] X. CHEN, L. CAO, and M. HE, "Overview of deep inverse reinforcement learning [J]," *Computer Engineering and Applications,* vol. 54, no. 5, pp. 24-35, 2018.

[8] W. Jin, T. D. Murphey, Z. Lu, and S. Mou, "Learning from human directional corrections," *IEEE Transactions on Robotics,* vol. 39, no. 1, pp. 625-644, 2022.

[9] J. Y. Zhang and A. D. Dragan, "Learning from extrapolated corrections," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7034-7040: IEEE.

[10] A. Jain, B. Wojcik, T. Joachims, and A. Saxena, "Learning trajectory preferences for manipulators via iterative improvement," *Advances in neural information processing systems,* vol. 26, 2013.

[11] J. K. Flake and E. I. Fried, "Measurement schmeasurement: Questionable measurement practices and how to avoid them," *Advances in Methods and Practices in Psychological Science,* vol. 3, no. 4, pp. 456-465, 2020.

[12] A. H. Marblestone, G. Wayne, and K. P. Kording, "Toward an integration of deep learning and neuroscience," *Frontiers in computational neuroscience,* vol. 10, p. 94, 2016.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature,* vol. 521, no. 7553, pp. 436-444, 2015.

[14] S. Feng, H. Zhou, and H. Dong, "Using deep neural network with small dataset to predict material defects," *Materials & Design,* vol. 162, pp. 300-310, 2019.

[15] R. May, G. Dandy, and H. Maier, "Review of input variable selection methods for artificial neural networks," *Artificial neural networks-methodological advances and biomedical applications,* vol. 10, no. 1, pp. 19-45, 2011.

[16] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*. Martin Hagan, 2014.

[17] W. Jin, D. Kulić, S. Mou, and S. Hirche, "Inverse optimal control from incomplete trajectory observations," *The International Journal of Robotics Research,* vol. 40, no. 6-7, pp. 848-865, 2021.

[18] W. Jin, D. Kulić, J. F.-S. Lin, S. Mou, and S. Hirche, "Inverse optimal control for multiphase cost functions," *IEEE Transactions on Robotics,* vol. 35, no. 6, pp. 1387-1398, 2019.

[19] A. Bobu, A. Bajcsy, J. F. Fisac, and A. D. Dragan, "Learning under misspecified objective spaces," in *Conference on Robot Learning*, 2018, pp. 796-805: PMLR.