Active Perception Interaction System for the Pepper Robot Using Deep Reinforcement Learning

Shih-Huan Tseng, *Member, IEEE*, De-Yu Lin

***Abstract*—This paper aims to design a robot system capable of actively seeking out and initiating interaction with users who have the intention to interact, rather than passively waiting for users to approach. To achieve this, we combined deep reinforcement learning to develop an active perception interaction system applied to the Pepper robot. The system is characterized by multimodal perception integration and virtual environment design. Firstly, the system integrates image, sound, and distance information to determine the user's interaction intention and initiates interaction proactively. Additionally, we designed a virtual environment to simulate the sensing and movement capabilities of the Pepper robot and generated various interaction scenarios within the virtual environment to accelerate the model training process. Finally, we applied the trained model to the Pepper robot and validated the system's performance. This paper utilizes the Deep Q Network (DQN) model in deep reinforcement learning, simulating the perception of human face and body positions, sound direction, and distance from the user as inputs to the model in the designed virtual environment. We designed five different interaction scenarios to train the model and compared the effects of different episodes, neuron numbers, and activation functions on the model training. The experimental results showed that the model achieved an average success rate of 90% across the five scenarios.**

***Index Terms*—Human-Robot Interaction, Multimodal Perception, Deep Reinforcement Learning.**

I. INTRODUCTION

Human-Robot Interaction (HRI) has always been a focal research topic in the field of robotics. HRI refers to the process wherein robots observe the current state of users and respond accordingly. For instance, service robots respond to user queries, while mobile robots follow users around. In the study by Amor et al. [1], imitation learning methods were employed to record sequences of actions performed by two individuals in front of a camera. These sequences served as training data, enabling the trained robot to respond correspondingly when someone performed the recorded actions. Draghici et al. [2] applied the Pepper robot in museum guide services, where users only needed to utter phrases like "Hello!", "Let Go", "Say", and "Bye", prompting Pepper to recognize the user, follow them, and introduce exhibits. Marinov et al. [3] proposed the Pose2Drone HRI framework for drones, which utilized OpenPose to extract human skeletal information and a classifier to interpret these poses, allowing the drone to receive commands based on user movements.

Shih-Huan Tseng is with Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, 824005 Taiwan. (e-mail: shtseng@ nkust.edu.tw).

De-Yu Lin was with Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, 824005 Taiwan. (e-mail: F109110110@nkust.edu.tw).

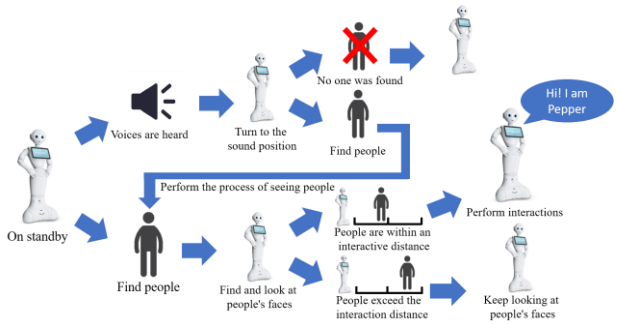

Fig. 1 Flowchart for Actively initiate interactions with willing users

Most HRI predominantly relies on passive responses, lacking the ability to proactively perceive users and initiate interactions. Robots only respond when users signal them, such as asking questions or making specific gestures. To address this limitation, robots must be capable of proactive interaction, autonomously seeking out areas or users of interest. For example, in the study by Duchetto et al. [4], a museum guide robot was designed to roam galleries, actively asking visitors if they needed guidance when it sensed their presence, rather than waiting passively. Foster et al. [5] developed a robotic bartender that evaluated customers' engagement based on facial coordinates, hand positions, torso angle, and head posture, then inquired if highly engaged customers required service.

With the maturation of reinforcement learning technologies, deep reinforcement learning has achieved human-level performance in computer games, thanks to its superior decision-making capabilities, leading to extensive research in robot control. For instance, in Rothbucher et al.'s study[6], a robot used microphones to gather sound direction and camera footage to look at the person speaking, demonstrating that reinforcement learning can achieve good performance with multiple sensors. In Ghadirzedeh et al.'s research[7], a robot played a ball-balancing game with humans, using the robot's hand orientation and the ball's position and acceleration as input states, enabling the robot to learn to control the board and keep the ball centered. Cruz et al. [8] showed that a robot could learn to tidy a desk by interpreting gestures or voice commands, proving that input features from different sources can complement each other.

This paper aims to develop an active perception interaction

system for the Pepper robot using deep reinforcement learning. The system combines image, sound, and distance information to determine user interaction intentions and actively initiate interactions with willing users, as shown in Fig. 1. By utilizing sound direction, the system expands the robot's perception range. When the robot hears a sound and turns towards it, successfully detecting the user's facial and body coordinates, it attempts to look at the user's face and assess the distance to the user. Within interaction range, the robot will attempt to initiate interaction; outside this range, it will continue to observe the user and accurately initiate interaction as the user approaches.

II. RELATED WORK

Numerous studies have explored the application of deep reinforcement learning to control robot actions in human-robot interaction (HRI) scenarios. For example, in a 2016 study by Vazquez et al. [9], the researchers attempted to make a robot move its head to look at the person speaking in a group. They used a virtual environment to simulate multi-person conversation scenarios, where the robot's goal was to locate the speaker and direct its gaze toward them. They collected information using microphones and cameras, adjusting the robot's view to focus on the speaker. However, this study was only tested in a virtual environment, and its feasibility in real-world robots remains to be verified.

In a 2016 study by Qureshi et al. [10], they were among the first to apply deep reinforcement learning to real-world HRI. Their goal was for the robot to actively recognize human intentions, such as initiating greetings or attempting a handshake. They placed the Pepper robot in an open public space to interact with people, using the responses of the interaction partners to determine reward values. In 2017, they introduced an advanced version, the Multimodal Deep Attention Recurrent Q-Network (MDARQN) [11]. This model incorporated an attention mechanism, enabling the robot to more accurately identify areas of interest. They used the Pepper robot's depth camera to obtain depth and grayscale images, combined with convolutional neural networks and the attention recurrent network proposed by Sorokin et al. [12], capturing high-importance feature points and using Long Short-Term Memory (LSTM) to capture temporal relationships. Finally, the DQN model was used to make decisions, selecting actions from waiting, gazing, waving, and handshaking. The overall training time was 14 days, approximately 4 hours per day, with testing conducted in the same public area after training.

In 2019, Ozaki et al. [13] conducted a study where robots used visual images to interact with people. Their goal was for the robot to learn when to initiate interaction, rather than initiating interaction with every passerby. They proposed a User-centered Q-learning (UCQL) model, which used the user's state changes over time as input. The states were divided into seven categories: no one, someone passing by, someone looking at the robot, someone staying nearby, someone approaching the robot, interaction initiated, and someone leaving, resulting in 49 possible state changes. They designed ten actions: waiting, blinking, looking at passersby, making a happy gesture, greeting, saying "I'm sorry" in Japanese, saying "Excuse me" in Japanese, saying "It's rainy today" in Japanese, instructing how to start the service, and saying goodbye. They placed the robot in the lobby exhibition space of a company, trained it for three days to collect data, and tested it in the same scene.

Lathuilière et al. [14] proposed a method to enhance the robot's active perception capabilities using auditory signals. They combined visual and auditory information, enabling the robot to turn its head towards areas of interest, and applied this method to a real robot. For model input, they used OpenPose to capture human skeletal information in images, including 18 human feature points, and microphones to capture sound direction information and the robot's head joint posture. This information was fed into an LSTM-based deep learning architecture, with the output being the robot's head rotation angle. In this study, the authors used a virtual environment to accelerate training, using the AVDIAR dataset[15]for training.

In the studies by Qureshi et al. [11] and Ozaki et al. [13], the primary goal was for the robot to actively initiate interaction with willing passersby, using visual images as input and focusing on determining user states. However, this limited the perception range, as the robot could not sense users outside its field of view or initiate interaction with them, even if they made a sound or called the robot. In the studies by Vazquez et al. [9] and Lathuilière et al. [14], sound direction was used to complement visual information, addressing the issue of blind spots in the robot's field of view. This allowed the robot to turn its view towards areas or people of interest, but the robots in these studies only gazed at people without interaction capabilities.

Therefore, we aim to combine the strengths of the aforementioned studies, using auditory signals to assist visual signals and overcome the limitations of the visual range. To determine user interaction intentions, we incorporate sonar distance to judge whether to initiate interaction based on user distance, enabling the robot to actively perceive areas of interest and proactively initiate interactions, maximizing interaction success rates. Considering that Qureshi et al.'s study [11] required 14 days to train the robot in a real environment, with the need for retraining if parameter adjustments were suboptimal during the process, we refer to Lathuilière et al.'s concept of virtual environment design [14]. This allows the robot to explore sounds and users in a large global map, designing a dedicated virtual environment to accelerate training time. Given that this study controls the Pepper robot by invoking built-in movement APIs to issue discrete commands, we chose the more implementable Deep Q Network (DQN) model in deep reinforcement learning.

III. METHODOLOGY

A. Programming for Pepper Robot

The Pepper robot is equipped with various sensors, including a 3D camera, a depth camera, and a 2D camera for visual tasks, such as facial recognition, pedestrian detection, and gaze detection. For audio tasks, Pepper's microphones can detect sound, locate its direction, measure its volume, and recognize speech. Additionally, Pepper is equipped with sonar sensors to capture the distance between the robot and surrounding objects or people. Despite the numerous sensors available, this study primarily utilizes the aforementioned cameras, microphones, and sonar sensors as sources of sensory information.

The robot's basic movements include turning the body left or right, turning the head left or right, looking up, and looking down—six actions in total. We added three more actions: initiating interaction (where the robot proactively greets the person), resetting (returning to the initial position if the robot does not see anyone after turning towards the sound), and waiting in place, making a total of nine actions, as shown in Table I. Given the larger movement scope for body rotation, each body turn is designed to rotate 5°, head turns rotate 1°, and head tilts (up and down) move 3°. The reset action in the virtual environment directly ends the current iteration, while in actual robot operations, the robot records the number of rotations and returns to the initial position, executing the stand-at-attention API. The interaction action involves the robot proactively greeting the person.

This study primarily uses Python for programming and employs the NAOqi and qi Framework to handle sensor data acquisition and action design. The relevant APIs include ALFaceDetection, ALPeoplePerception, ALSoundLocation, ALLaser, and ALMotion.

Table I
Action Set

Action No.	Actions
a_0	Waiting
a_1	Turning body left
a_2	Turning body right
a_3	Turning head left
a_4	Turning head right
a_5	Look up
a_6	Look down
a_7	Reset
a_8	Initiate interaction

B. Virtual Environment Design

1) Environment Design

This paper utilizes the OpenAI Gym [16] toolkit to develop a customized virtual environment that simulates the visual and auditory perspectives of the Pepper robot. This enables the model to be trained in a virtual environment, and the trained model is subsequently transferred to a real Pepper robot for testing.

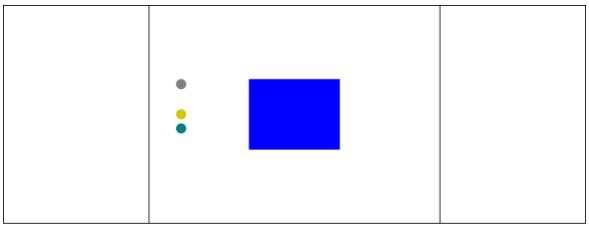

Fig. 2 Virtual environment design

To replicate the actual sensing environment of the robot, we measured Pepper's visual field of view. According to official documentation, Pepper's visual field of view is $55.7^\circ \times 43.7^\circ$, but it cannot capture a 180° range like humans. Although Pepper's head can pivot up to 119° on either side, we limited the head rotation range to 90° to ensure the robot faces the user directly, using body rotation to adjust the view if necessary. We measured the highest and lowest angles Pepper can see when looking up and down, resulting in an extreme angle of about 135°, representing the vertical range. As the robot can rotate its body a full 360°, the auditory field of view is also 360°. Based on this information, we used the concept of a camera panorama to flatten the robot's field of view into a 2D plane map, setting the virtual environment size to $360^\circ \times 135^\circ$, with the center point $(0^\circ, 65^\circ)$ as the reference. Within this range, a rectangular frame simulating the robot's visual field of $55.7^\circ \times 43.7^\circ$ explores the environment, representing changes in the robot's view. We converted the 3D space task of searching for people into a 2D plane task.

The designed environment, as shown in Fig. 2, features a blue rectangle at the center representing the robot's current visual field. The visual field center point (μ_x, μ_y) indicates the center of the robot's view, and the closer (μ_x, μ_y) is to an object, the more the robot is gazing at it. When the blue rectangle is centered, the robot is in its standard standing posture, looking straight ahead, with two lines on either side representing the head movement limits. The blue rectangle can only move within these two lines, indicating the robot's head cannot turn beyond this range. If the target is beyond this range, the robot needs to turn its body to bring the target into the head's movable range, then move its head to face the target directly.

In the environment, three different colored dots represent the positions of people and sounds. The gray dot is the facial coordinate (F_x, F_y), covering the entire virtual environment, representing the person's actual location. The green dot is the body coordinate (B_x, B_y), the same as and attached to the facial coordinate. The yellow dot is the sound direction θ_{sound}, representing the sound source's location.

2) State Design

This study uses four types of sensor data: facial position, body position, sound angle, and sonar distance.

a) Facial Position (f_x, f_y): This is the facial information captured when the robot sees a person in its field of view. In the virtual environment, this is calculated by the difference between the facial coordinate (F_x, F_y) and the visual center point (μ_x, μ_y) divided by the size of the visual frame ($56^\circ \times 44^\circ$), as shown in equation (1). The range is between $(0.5\sim-0.5, 0.5\sim-0.5)$, with a default of (-1, -1).

$$(f_x, f_y) = \left(\frac{(\mu_x - F_x)}{56}, \frac{(\mu_y - F_y)}{44}\right) \quad (1)$$

b) Body Position (b_x, b_y): This is the body information captured when the robot sees a person in its field of view. In the virtual environment, this is calculated by the difference between the body coordinate (B_x, B_y) and the visual center point (μ_x, μ_y) divided by the size of the visual frame ($56^\circ \times 44^\circ$), as shown in equation (2). The range is between $(0.5\sim-0.5, 0.5\sim-0.5)$, with a default of (-1, -1).

$$(b_x, b_y) = \left(\frac{(\mu_x - B_x)}{56}, \frac{(\mu_y - B_y)}{44}\right) \quad (2)$$

c) Sound Angle θ: This is the relative angle between the sound direction θ_{sound} and the robot's orientation θ_{robot}. In this study, the sound is retained until the interaction is completed or new sound occurs. When the robot moves its body, the sound direction remains constant, using the relative angle. When the robot turns towards the sound, the angle between the sound and the body decreases $\theta\downarrow$; conversely, when turning away, the angle increases ($\theta\uparrow$), indicating the sound state. The sound direction on the X-axis is randomly generated within the range of -180° to 180°. To reduce the state size, we normalized the sound range to between -1 and 1, avoiding significant differences from other state values.

d) Sonar Distance d : This measures the distance between the robot and the object in front (not visible in the environment), used as a reference for judging the user's distance. The range is between 0 and 1. When someone is in front of the robot and the sonar value is less than 0.5, it indicates the person is interested in the robot; if it exceeds 0.5, it suggests the person is farther away and the robot should continue to wait.

C. Scenario Design (Training Data)

Based on the aforementioned sensor data, we designed different scenarios to train the model. To ensure smoother model training, we excluded some rare or meaningless situations, such as a face being below the body, the body height exceeding the robot's field of view, the face and body having an excessive X-axis distance, and situations where the sound and person are on opposite sides of the robot. This is because the robot cannot find a person when exploring sound directions or locate the head near the body under these conditions. Therefore, we defined the following rules:

- The Y-axis of the body B_y must be within the robot's visual range $(\mu_y \pm 22)$ to ensure the robot can see the person when it turns to the person's location.
- The face must be above the body, and the X-axis offset between the face and body should not exceed 2°.
- The sound source location does not necessarily have to coincide with a person's presence, but if a person is present, the body's X-axis B_x must align with the sound source θ_{sound}.

Based on these rules, we identified five scenarios:

1) **Scenario 1: Sound with a Person Present**

This is the most basic scenario where a person near the robot makes a sound. The robot should turn to the sound's location, find the person, look at their face, and decide whether to initiate interaction or wait and continue tracking the face based on the sonar distance. (Illustrated in Fig. 3.)

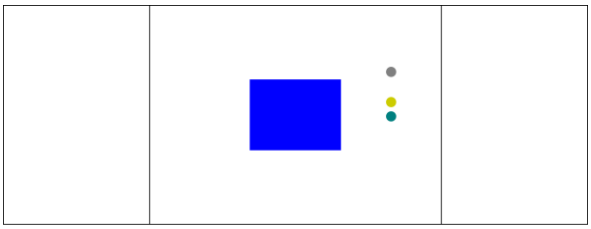

Fig. 3 Illustration for sound with a person present.

2) **Scenario 2: Sound but Person in Front**

There are two possibilities: a person is in front of the robot, and there is noise or another person generating the sound nearby, or the robot sees a person when turning towards the sound source. If no sonar distance is available, the robot should prioritize the person in front, look at their face, and initiate interaction. If sonar assistance is available, the robot can determine the distance to the person in front and decide whether to abandon the person in front to pursue the sound source. (Illustrated in Fig. 4.)

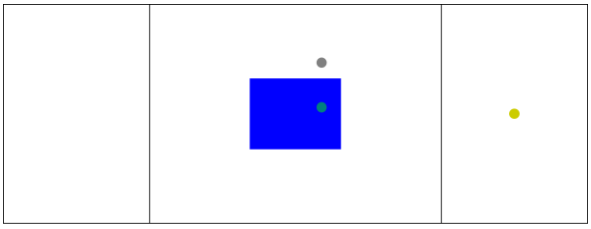

Fig. 4 Illustration for sound but person in front.

3) **Scenario 3: Sound but No Person at Sound Source**

This scenario represents non-human generated sound or a person who has left the robot's sensing range, such as closing a door or being obstructed by an obstacle. When the robot turns to the sound's location and does not see anyone, it should return to its original position and wait for a new signal. (Illustrated in Fig. 5.)

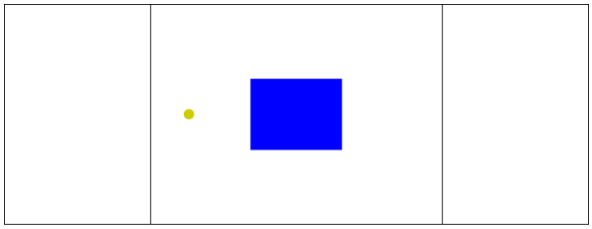

Fig. 5 Illustration for sound but no person at sound source.

4) **Scenario 4: No Sound but Person in Front**

In this scenario, a person is within the robot's visual range. If the sonar distance is within the interaction range, the robot should look at the person and decide whether to initiate interaction based on the distance. (Shown in Fig. 6)

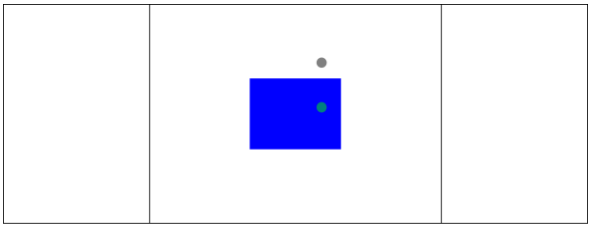

Fig. 6 Illustration for no sound but person in front

5) **Scenario 5: No Sound and No Person**

This scenario represents the absence of any triggering signals, so the robot should wait in place for new signals to appear. (Illustrated in Fig. 7)

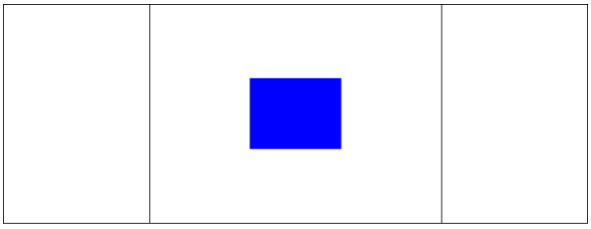

Fig. 7 Illustration for no sound and no person.

D. Reward Mechanism Design

The reward function in this study is divided into two parts: sound searching and face searching. Initially, the system prioritizes sound searching before finding a person; once a sound is detected and the robot turns towards it to find a person, the system enters the face searching mode. The reward function design is based on the optimal configuration determined through iterative experimental testing.

First, boundary conditions must be defined. If the visual field exceeds the boundaries, i.e., beyond the range the head can turn, the iteration ends immediately and a penalty is given. Similarly, if the robot turns more than 180°, the iteration ends immediately with a penalty, as the robot should use the smallest angle to find the sound. As shown in Equation (3), μ_x is the X-axis of the visual field, μ_y is the Y-axis of the visual field, and θ_robot is the robot's turning angle.

$$\mathrm{r} = -10 \text{ if } \begin{cases} 90 < \mu_x \; or \; \mu_x < -90 \\ 135 < \mu_y \; or \; \mu_y < 0 \\ 180 < \theta_{robot} \; or \; \theta_{robot} < -180 \end{cases} \quad (3)$$

1) **Sound Searching Reward Design:**

Although reducing the angle between the robot and the sound direction should always yield a reward, to encourage the robot to turn its body rather than just its head, a reward is given only when the body turns and the angle between the robot and the sound direction decreases. As shown in Equation (4), when the previous sound angle θ' decreases to the current sound angle θ by 5°, it indicates the robot has turned 5° towards the sound, thus earning a positive reward. If the robot reaches the vicinity of the sound source but does not find a person, it should return to the initial position and end the iteration. Therefore, if the angle between the sound and the body is less than 15° and no person is detected, the robot should perform the return action, receiving a completion reward, while other actions receive a -1 penalty to increase the attractiveness of the correct actions.

$$\mathrm{r} = \begin{cases} 2 & if \; \theta' - \theta = 5 \\ 100 & if \; \theta \leq 15 \; and \; a = a_8 \\ -1 & otherwise \end{cases} \quad (4)$$

2) **Face Searching Reward Design:**

When the robot's visual field moves towards the sound source and detects a body or face, it enters the face searching stage. Face searching can be divided into three situations: only seeing the body, only seeing the face, and seeing both the body and face. The latter two situations are treated the same, thus simplifying to two modes: seeing a face and only seeing a body. When only the body is seen, the robot's view is too low or too close, requiring an upward adjustment. If the Y-axis of the visual field μ_y increases compared to the previous moment μ_y', it indicates an improved view and earns a reward, while other actions receive a -1 penalty. As shown in Equation (5), when a person is close, to prioritize face searching over continued sound searching, the upward adjustment reward value is adjusted based on the distance. If the distance is less than 0.5, it indicates the person is within interaction range, prompting the model to prioritize face searching and initiate interaction.

$$\mathrm{r} = \begin{cases} 3 - (d * 2) & if \; \mu_y > \mu_y' \\ -1 & otherwise \end{cases} \quad (5)$$

When a face enters the visual field range, the system attempts to move the face to the center of the view. Using the Euclidean distance formula (Equation (6)), the distance l_f between the face coordinates and the visual center is calculated. If l_f decreases compared to the previous moment l_f', indicating the face is closer to the center, a reward is given. To prioritize looking at the face, the reward for reducing the distance is higher than the previous rewards for sound and body. Once the robot successfully centers the face, it decides whether to initiate interaction based on the sonar distance d. If d is less than the interaction distance of 0.5, indicating the person is willing to interact, successfully initiating interaction completes the task, earning a completion reward and ending the iteration. If d > 0.5, indicating the person is still far from the robot, the robot should keep the person in view without initiating interaction, waiting

to see if the person enters the interaction range. As shown in Equation (7).

$$l_f = \sqrt[2]{(\mu_x - F_x)^2 + (\mu_y - F_y)^2} \tag{6}$$

$$\mathrm{r} = \begin{cases} 5 & if\ l_f < l_f' \\ 100 & if\ l_f < 0.25\ and\ d \le 5\ and\ a = a_7 \\ 100 & if\ l_f < 0.25\ and\ d > 5\ and\ a = a_0 \\ -1 & otherwise \end{cases} \tag{7}$$

In another scenario, if no sound occurs and no person is detected, the robot should choose to wait until a new signal appears. In the virtual environment, the coordinates of the sound and the person do not change, so simply executing the wait action completes the task and earns a completion reward, as shown in Equation (8).

$$\mathrm{r} = \begin{cases} 100 & if\ a = a_0 \\ -1 & otherwise \end{cases} \tag{8}$$

E. Deep Q Network Model Construction

In our study, since the coordinates of the person do not change, and the robot's actions vary by fixed angles, the virtual environment can be converted into a discrete space, where each state exists independently. Based on this, we chose the most well-known algorithm for discrete environments in deep reinforcement learning-the Deep Q Network (DQN) as the decision model. We also experimented with the PPO model [17] used for continuous spaces, but in our test results, PPO performed significantly worse than DQN in our virtual environment. Therefore, we decided to use DQN as the decision model and did not compare it with other models further.

The Deep Q-Network (DQN) was proposed by Mnih et al. [18] from Google DeepMind in 2013. In their paper, they demonstrated that DQN achieved better performance than human players in Atari 2600 games, gaining widespread attention. We adopted the Double DQN model proposed by Van Hasselt et al. in 2016[19], which significantly improves the stability and convergence speed of DQN. Double DQN introduced two key methods:

- Experience Replay: This method stores the state, action, reward, and next state of each training instance as experiences. During training, a random batch of data is sampled from the Replay Memory. The Replay Memory is continuously updated with new data and old data are discarded as the agent interacts with the environment. This reuse of old data not only maximizes the value of the data but also enhances DQN's training speed and breaks the temporal correlation of the data by random sampling, preventing the model from being influenced by the sequential order of the data.
- Target Network: Having only one neural network that updates both the target and output values simultaneously can cause instability, as the model would be "chasing its own tail". To address this, the training neural network and the Target Network used to calculate target values are separated. The parameters of the training neural network are updated as usual, while the Target Network's parameters are copied from the training network at regular intervals and remain unchanged otherwise, significantly improving DQN's stability.

Algorithm 1 Double DQN

Algorithm 1: Double DQN Algorithm

Input : $\mathcal{D}$ − empty replay buffer; θ − initial network parameters; θ^- − copy of θ
Input : N_r − replay buffer maximum size; N_b − training batch size; N^- − target network replacement freq.
for episode $e \in \{1, 2, \dots, M\}$ **do**
  Initialize frame sequence **x** ← ()
  for t ∈ {0, 1, … , } **do**
    Set state s ← **x**, sample action a ~ π_B
    Sample next frame x^t form environment ε given (s, a) and receive reward r, and append x^t to **x**
    if |**x**| > N_f **then** delete oldest frame $x_{t_{min}}$ from **x** **end**
    Set s' ← **x**, and add transition tuple (s, a, r, s') to $\mathcal{D}$, replacing the oldest tuple if $|\mathcal{D}| \ge N_r$
    Sample a minibatch of N_b tuples (s, a, r, s')~Unif($\mathcal{D}$)
    Construct target values, one for each of the N_b tuples:
    Define $a^{max}(s'; \theta) = \arg\max_{a'} Q(s', a'; \theta)$
    $y_j = \begin{cases} r & if\ s'\ is\ terminal \\ r + \gamma Q(s', a^{max}(s'; \theta); \theta^-), & otherwise \end{cases}$
    Do a gradient descent step with loss $\|y_i - Q(s, a; \theta)\|^2$
    Replace target parameters $\theta^- \leftarrow \theta$ every N^- steps
  end
end

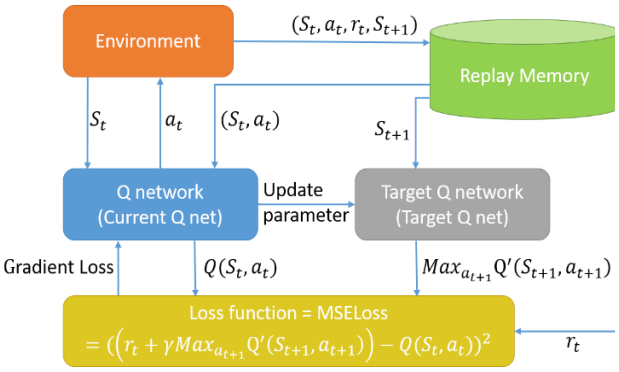

Fig. 8 Double DQN architecture.

The algorithm for the Double DQN model is illustrated in Algorithm 1. In combination with the network architecture shown in Fig. 8, the working process of Double DQN can be described as follows:

1) **Initial Stage:**
The environment generates an initial state S_t and passes this state to the Q network.
2) **Action Decision:**
The Q network determines the action a_t to be executed based on its policy and feeds this action back to the environment.
3) **Reward and State Update:**
Upon receiving the action a_t, the environment evaluates its effect, generating the corresponding reward r_t and the next state S_{t+1}.
4) **Experience Storage:**
The information regarding the state, action, reward, and next state is combined and stored in the Replay Memory.

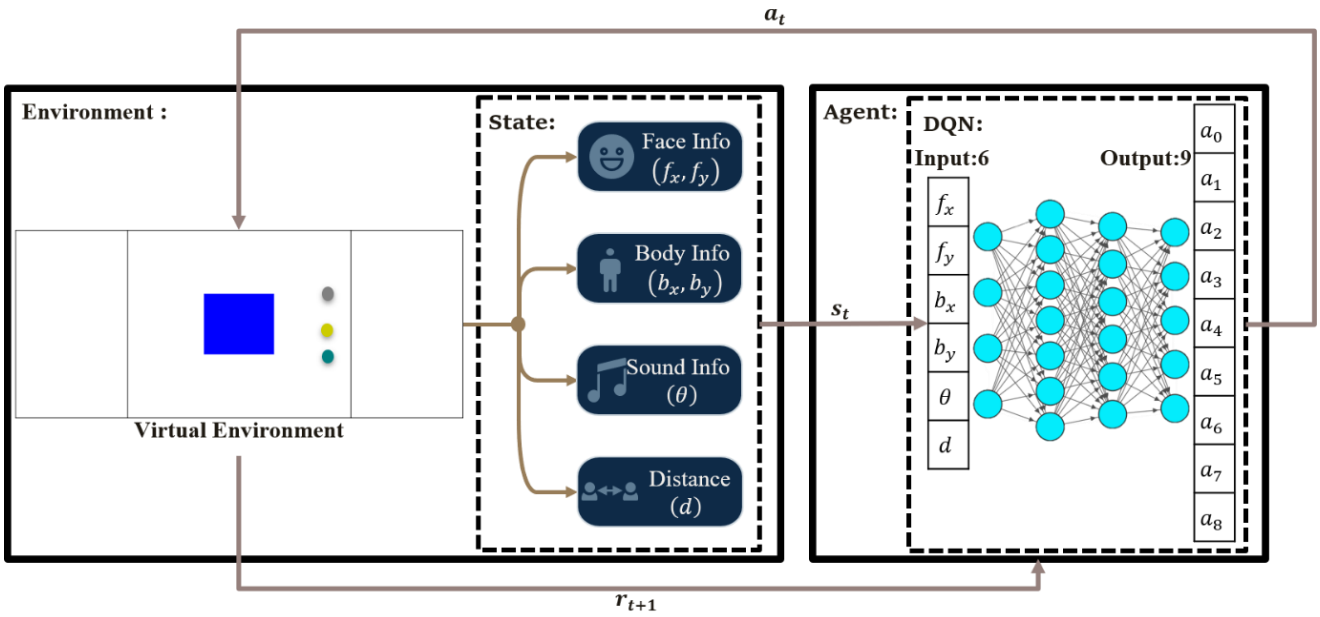

Fig. 9 Training architecture of the Double DQN model

5) **Batch Calculation:**
A specific batch of data is extracted from the Replay Memory and passed to the Q network for computation, which generates an action. The next state is also passed to the Target Q network to calculate the maximum Q-value for the next state.
6) **Loss Function Calculation:**
The Q-value generated by the Q network, the maximum Q-value for the next state generated by the Target Q network, and the reward r_t are used to calculate the loss function.
7) **Parameter Update:**
The calculated loss value is backpropagated through the Q network using gradient descent, leading to parameter updates.
8) **Parameter Copying:**
Every few steps, the parameters of the Q network are fully copied to the Target Q network.

This training process continues until the model converges or the predefined maximum step limit is reached.

We designed the virtual environment, defined the states, actions, and reward functions, and constructed the training architecture of the Double DQN model, as shown in Fig. 9. In this training architecture, the model acquires the facial position (f_x, f_y), body position (b_x, b_y), sound angle θ, and sonar distance d from the virtual environment. Based on this information, it inputs the data into a DQN model with two hidden layers, generating nine different actions $a_0 \sim a_8$, enabling the robot to actively seek out users of interest and proactively initiate interaction with willing users.

IV. EXPERIMENTS AND RESULTS

A. Experimental Environment

The experimental environment in this study comprises both a server-side and a Pepper robot-side, with the frameworks and equipment detailed in Table II and Table III. Both the server and Pepper robot programs were developed using Python. The DQN model was developed in the server-side environment using the PyTorch deep learning library proposed by Paszke et al. [20] Due to the Pepper robot's processor version not supporting Python 3, the information received by Pepper is transmitted via a socket to the server for model computation, after which the results are sent back to the Pepper robot. This configuration ensures that the server can efficiently handle model training and computation, while the Pepper robot can smoothly receive and execute commands from the server, thereby achieving specific human-robot interaction functions.

Table II
Server environment

Central Processing Unit (CPU)	Intel Core i7-8700
Graphics Processing Unit (GPU)	NVIDIA GeForce RTX3060
Random Access Memory (RAM)	64GB DDR3
Operating System	Windows 10
Programming Language	Python 3.7.3
Development Environment	Pytorch 1.13.1 CUDA 11.8 GYM 0.21.0

Table IV
Pepper environment

Central Processing Unit (CPU)		Atom E3845 Quad core
Operating System		Gentoo Linux
Programming Language		Python 2.7.18

B. Experimental Design

In this subsection, we will introduce the design of the DQN model, including the model architecture and parameter settings. Additionally, we will discuss the training method, process, and execution flow for the Pepper robot within the virtual environment.

1) **DQN Parameter Design**

The training architecture of the DQN model used in this study is shown in Fig. 9. The model features two hidden layers and uses Sigmoid [21] and ReLU [22] as activation functions. The optimizer used is Adam, proposed by Kingma et al. [23] in 2014. The number of neurons and episodes were varied in three different parameter combinations to evaluate the model's performance under different settings. The action selection process uses Epsilon Greedy to decide between exploration and exploitation. The exploration probability ε gradually decays from ε_max to ε_min according to Equation (9). This allows the model to start with random actions to accumulate experience, then gradually rely more on the model's decisions as experience increases, while retaining some randomness to avoid local optima. The complete model parameter settings are detailed in Table IV.

$$\varepsilon = \varepsilon_{\min} + (\varepsilon_{\max} - \varepsilon_{\min}) * \varepsilon_{decay} \quad (9)$$

2) **Training Process Design**

Each training session involves running a fixed number of simulation episodes. At the start of each simulation, it is randomly determined whether a person, sound, and sonar distance are present. If sound is present, a sound source is randomly generated within a range of 0° to 360°, followed by determining whether the person is associated with the sound source (i.e., the X-axis coordinates align) or appears in the robot's field of view. Once the person, sound, and sonar data are generated, it is checked whether the person is within the robot's visual field. If so, facial and body states are generated; otherwise, the default is set to -1. The sound source is then normalized to a range of -1 to 1, and if no sound source is generated, it is set to -2. Once all states are generated, they are passed to the DQN model to begin training.

Table III
DQN model parameter settings

Number of input states	6
Number of output actions	9
Number of neurons in the first and second layers	(150、75)，(200、100)，(250、125)
Number of training episodes	5000，8000，10000
Size of replay memory	3000
Batch size for training	128
Learning rate	0.001
Discount factor	0.9
Frequency of target network updates	100
Maximum number of interaction steps per episode	300
Activity Function	Sigmoid，ReLu
Optimizer	Adam
Loss Function	MSE
ε_max	0.9
ε_min	0.05
ε_decay	Exponential decay[1]

At the start of each iteration, the DQN collects information without immediately beginning the learning process, waiting until the Replay Memory is filled before learning begins. Each iteration continues until the target is reached or the maximum number of interaction steps for the environment is met. The total reward value, number of interactions, and estimated error values are stored. Every 100 episodes, a sample is drawn and plotted as a line graph to evaluate the model's performance.

3) **Testing Process Design**

During the testing phase, the primary goal is to evaluate the model's performance and generalization with different parameters. The trained model is tested across the five proposed virtual scenarios, with a total of 1,000 episodes and 200 episodes for each scenario. The success rate for each scenario and the overall success rate are calculated, along with the percentage of total positive rewards obtained. The success rate represents the number of times the model successfully completed the task. The success criteria vary by scenario; for example, in Scenarios One, Two, and Four, success is defined as the robot successfully turning towards a person, with the person within interaction range, and the robot initiating interaction, or if the person is out of interaction range and the robot chooses to wait. In Scenario Three, success is defined as the robot turning towards the sound source but not finding a person, choosing to return to its original position. In Scenario Five, success is defined as the robot choosing to wait.

Table VI
Success Rate of Different Parameter Combinations

Neurons	Episodes	Success Rate (%)	
		Sigmoid	ReLu
(150, 75)	5000	90.1	41.8
	8000	83.3	52.0
	10000	76.2	48.9
(200, 100)	5000	79.0	69.9
	8000	83.7	67.6
	10000	**91.8**	58.3
(250, 125)	5000	84.9	67.4
	8000	73.8	57.7
	10000	83.5	43.1

We calculate the success rate for each scenario and the overall success rate, analyzing the model's performance under various conditions. The percentage of total positive rewards indicates how often the robot performed the correct actions, and we assess whether the model exhibits overfitting or underfitting. We tested models with different parameters to identify the optimal combination. These parameter combinations include iteration counts of 5,000, 8,000, and 10,000, and neuron counts of (150,75), (200,100), and (250,125) in combination with ReLU and Sigmoid activation functions, resulting in a total of 18 combinations. We compare these to determine which combination achieves the highest success rate.

C. Experimental Results

1) **Comparison of Different Parameter Combinations**

We first compared the test results of 18 different parameter combinations, varying the number of neurons, episodes, and activation functions, as shown in Table VI. From the test results, we identified the best parameter combination as the one with 200 and 100 neurons, 10,000 episodes, and the Sigmoid activation function. The line charts depicting the cumulative reward, steps, and loss values during training are shown in Fig. 10. After identifying the best parameter combination, we used this model to compare the success rate in each

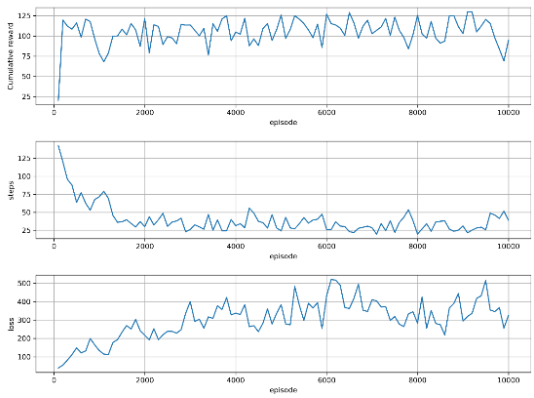

Fig. 10 Cumulative reward, steps, and loss value with 200 and 100 neurons, 10,000 episodes and Sigmoid.

Table V
Test results with 200 and 100 neurons, 10,000 episodes and Sigmoid.

Scenario	Success Counts	Success Rate (%)
1	186/200	93.0
2	184/200	92.0
3	184/200	92.0
4	164/200	82.0
5	200/200	100.0
Total	918/1000	91.8
Positive Reward Rate (%)	90.5%	

scenario and the overall success rate, analyzing the model's performance across various scenarios, with the test results presented in Table V.

The second-best parameter combination was identified as 150 and 75 neurons, 5,000 episodes, and the Sigmoid activation function. The corresponding line charts for reward values, interaction counts, and estimated error values during training are shown in Fig. 11. The model's performance in various scenarios with the second-best parameter combination is summarized in Table VII.

From the comparison between the two combinations, we observed that although the second-best parameter combination utilized fewer episodes and neurons and achieved a success rate slightly lower than the best combination, its success rate across various scenarios was not as consistent as the best combination. This indicates that the second-best parameter combination lacked sufficient scenario generalization, likely due to the insufficient number of episodes, which may have led to incomplete training across all situations, and the neuron count being inadequate to represent all state variations in the scenarios. Therefore, while using fewer resources may yield decent results, it could result in incomplete model representation or only partial training across certain scenarios.

2) **Evaluation of Training Speed in Different Environments**

To demonstrate that training and testing the model in a virtual environment is faster than in a real-world

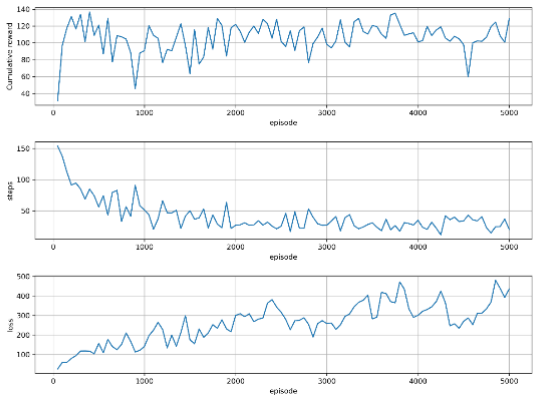

Fig. 11 Cumulative reward, steps, and loss value with 150 and 75 neurons, 5,000 episodes and Sigmoid.

Table VIII
Test results with 200 and 100 neurons, 10,000 episodes and Sigmoid.

Scenario	Success Counts	Success Rate (%)
1	177/200	88.5
2	173/200	86.5
3	200/200	100.0
4	151/200	75.5
5	200/200	100.0
Total	901/1000	90.1
Positive Reward Rate (%)	90.1%	

Table IX
Execution time of each test scenario

Scenario	Real	Virtual with visualization	Virtual without visualization
1	29.28(s)	0.498(s)	0.009(s)
2	16.79(s)	0.155(s)	0.004(s)
3	31.46(s)	0.304(s)	0.003(s)
4	16.14(s)	0.158(s)	0.003(s)

environment, we compared the runtime across three different environments: the real-world environment, a visualized virtual environment, and a virtual environment without displaying the running process, excluding Scenario 5. Due to the difficulty in calculating training time in the real-world environment, we tested the trained model and compared the runtime in each scenario. The runtime results are shown in Table VIII.

We calculated the time taken for the robot to execute a single scenario test. In the real-world environment, timing started from the moment the user emitted a sound or the robot saw the user until the robot initiated interaction or completed the return to the original position. In the virtual environment, the time was calculated from the completion of environment initialization to the receipt of the completion command, with the average taken over 20 runs. The results indicated that the virtual environment's runtime was significantly faster than the real-world environment, especially in scenarios requiring turning, such as Scenario One and Scenario Three. The main reason is that the virtual environment does not need to account for speech and motor rotation times, resulting in much faster speeds. If the virtual environment's operation process is not displayed, the runtime can be further reduced. This runtime was calculated based on a single run, whereas actual training requires thousands of episodes, leading to a more significant difference in overall runtime. Therefore, we concluded that using a virtual environment for training can effectively improve training time.

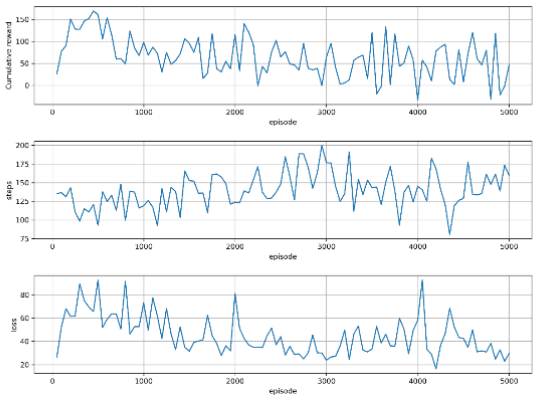

Fig. 12 Cumulative reward, steps, and loss value with 200 and 75 neurons, 5,000 episodes and ReLu.

Table VII
Test results with 200 and 75 neurons, 5,000 episodes and ReLu.

Scenario	Success Counts	Success Rate (%)
1	112/200	56.0
2	122/200	61.0
3	112/200	56.0
4	153/200	76.5
5	200/200	100.0
Total	699/1000	69.9
Positive Reward Rate (%)	58.5%	

D. Discussion

In the experimental results, we found that the parameter combinations with higher success rates all used Sigmoid as the activation function, which contrasts with the traditional perception that ReLU generally performs better. In tests using the ReLU activation function, the highest success rate was achieved with the combination of 200 neurons, 75 neurons, and 5,000 episodes. The corresponding line charts for cumulative reward, steps, and loss value during training are shown in Fig. 12, and the model's performance across various scenarios is summarized in Table 7.

We found that ReLU underperformed compared to Sigmoid in our experiments. In the test with 200 neurons, 75 neurons, and 5,000 episodes, the success rate was only about 70%. We hypothesized that this could be due to the "dying ReLU" problem, where ReLU outputs 0 in the negative range. To test this hypothesis, we examined the output of the first layer of neurons before activation and found that approximately 50% of the values were negative. This means that around 50% of the neurons' outputs would be set to 0 after ReLU activation, leading to information loss. In contrast, Sigmoid transforms negative values into a range between 0 and 0.5, preserving the differences among negative values and thus performing better.

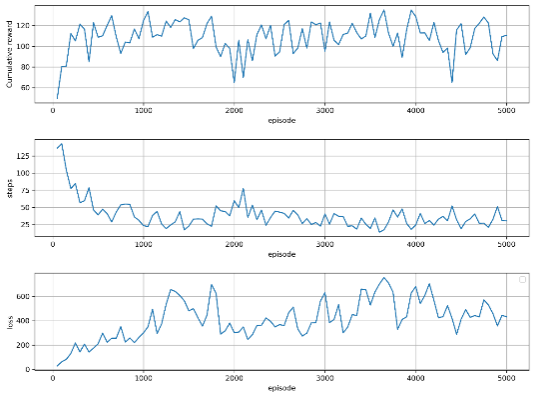

Fig. 13 Cumulative reward, steps, and loss value with 150 and 75 neurons, 5,000 episodes and Leaky ReLu.

Table X
Test results with 200 and 75 neurons, 5,000 episodes and ReLu.

Scenario	Success Counts	Success Rate (%)
1	107/200	53.5
2	163/200	81.5
3	113/200	56.5
4	164/200	82.0
5	200/200	100.0
Total	747/1000	74.7
Positive Reward Rate (%)	76.4%	

To address the issue with ReLU in the negative range, we experimented with the Leaky ReLU activation function. Leaky ReLU[24] is an improved version of ReLU that addresses the issue by assigning a small weight to negative values, allowing them to be expressed. As shown in Figure 13, we trained using the best parameter combinations from Sigmoid and ReLU and selected the best results. The best results were achieved with 5,000 episodes and neuron number of 150 and 75. The line charts for cumulative reward, steps, and loss value during training are shown in Fig. 13, and the model's performance across various scenarios is summarized in Table X.

The experimental results showed that Leaky ReLU increased the success rate from 69.9% with ReLU to 74.7%, an improvement of about 5%. The proportion of positive rewards increased from 58.5% to 76.4%, indicating that the model was better at achieving the target. Overall, Leaky ReLU outperformed ReLU, confirming that ReLU had limitations in our study, leading to suboptimal performance.

Finally, we compared the performance of ReLU, Leaky ReLU, and Sigmoid with 5,000 episodes and 150 and 75 neurons, as shown in Table XI. We concluded that Sigmoid performed better than Leaky ReLU and ReLU for our study. We speculate that this is because Sigmoid assigns the same weight changes to negative inputs as it does to positive ones, whereas Leaky ReLU assigns only a 0.01 weight to the negative range. Therefore, in cases where there are many negative values and they hold some significance, Sigmoid slightly outperforms Leaky ReLU.

Table XI
Test results with 150 and 75 neurons, 5,000 episodes for different activate functions

Scenario	Success Rate (%)		
	ReLu	Leaky ReLu	Sigmoid
1	9.5%	53.5%	88.5%
2	11.0%	81.5%	86.5%
3	70.5%	56.5%	100.0%
4	18.0%	82.0%	75.5%
5	100.0%	100.0%	100.0%
Total	41.8%	74.7%	90.1%
Positive Reward Rate (%)	55.6%	76.4%	90.1%

V. CONCLUSION

This study proposed the use of the Deep Q-Network (DQN) model within deep reinforcement learning to implement an active human-robot interaction system on the Pepper robot. We utilized the OpenAI Gym toolkit to design a custom virtual environment that simulates the sensory inputs Pepper would encounter in a real-world setting. These inputs included facial states, body states, sound direction, and sonar distance. We modeled nine different robot actions, such as waiting, turning the body left or right, turning the head left or right, looking up, looking down, interacting, and resetting. Training the DQN model within the virtual environment significantly accelerated the training process.

We compared the impact of different model parameters on the performance of the DQN model, including the number of episodes, the number of neurons, and the choice of activation functions. The trained DQN model was tested across five different scenarios we designed: the presence of sound with a person at the sound source, the presence of sound but a person directly in front, the presence of sound but no person at the sound source, no sound but a person directly in front, and no sound and no person. The experimental results demonstrated that the model performed best under the parameter combination of 10,000 episodes, 200 and 100 neurons, and the Sigmoid activation function.

To validate the model's applicability in real-world environments, we deployed it on the Pepper robot. Using the SDK provided by Pepper, we acquired sensory data, which were then input into the model to generate actions. This enabled the robot to actively search for people and initiate interactions. The experimental results confirmed the feasibility of the proposed model in a real-world setting.

In the future, we plan to experiment with comparing different Deep Q-learning algorithms, which will allow us to evaluate their effectiveness in handling more complex and dynamic interaction scenarios. By exploring these alternative algorithms, we aim to identify those that can improve the model's performance, particularly in terms of accuracy and reliability, thereby contributing to the advancement of human-robot interaction technology.

For long-term research, we aim to enhance the design of the interaction process to improve the overall human-robot interaction experience. While this study focused on locating individuals and initiating interaction, the subsequent communication following interaction initiation has not been thoroughly explored. Additionally, we hope to integrate other sensory inputs, such as gaze direction and sound volume, to strengthen the model's decision-making capabilities. We also plan to experiment with implementing continuous state, action, and virtual environments for training, allowing the model to operate in a manner closer to real-world conditions rather than moving at fixed angles, thereby avoiding stuttering. These improvements will contribute to enhancing the model's accuracy and reliability, further advancing the field of human-robot interaction technology.

REFERENCE

[1] H. B. Amor, D. Vogt, M. Ewerton, E. Berger, B. Jung, and J. Peters, "Learning responsive robot behavior by imitation," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013: IEEE, pp. 3257-3264.

[2] B. G. Draghici, A. E. Dobre, M. Misaros, and O. P. Stan, "Development of a human service robot application using pepper robot as a museum guide," in *2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 2022: IEEE, pp. 1-5.

[3] Z. Marinov, S. Vasileva, Q. Wang, C. Seibold, J. Zhang, and R. Stiefelhagen, "Pose2drone: A skeleton-pose-based framework for human-drone interaction," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021: IEEE, pp. 776-780.

[4] F. Del Duchetto, P. Baxter, and M. Hanheide, "Lindsey the tour guide robot-usage patterns in a museum long-term deployment," in *2019 28th IEEE international conference on robot and human interactive communication (RO-MAN)*, 2019: IEEE, pp. 1-8.

[5] M. E. Foster, A. Gaschler, and M. Giuliani, "Automatically classifying user engagement for dynamic multi-party human–robot interaction," *International Journal of Social Robotics*, vol. 9, no. 5, pp. 659-674, 2017.

[6] M. Rothbucher, C. Denk, and K. Diepold, "Robotic gaze control using reinforcement learning," in *2012 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE 2012) Proceedings*, 2012: IEEE, pp. 83-88.

[7] A. Ghadirzadeh, J. Bütepage, A. Maki, D. Kragic, and M. Björkman, "A sensorimotor reinforcement learning framework for physical human-robot interaction," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016: IEEE, pp. 2682-2688.

[8] F. Cruz, G. I. Parisi, J. Twiefel, and S. Wermter, "Multi-modal integration of dynamic audiovisual patterns for an interactive reinforcement learning scenario," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016: IEEE, pp. 759-766.

[9] M. Vázquez, A. Steinfeld, and S. E. Hudson, "Maintaining awareness of the focus of attention of a conversation: A robot-centric reinforcement learning approach," in *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2016: IEEE, pp. 36-43.

[10] A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro, "Robot gains social intelligence through multimodal deep reinforcement learning," in *2016 IEEE-RAS 16th international conference on humanoid robots (humanoids)*, 2016: IEEE, pp. 745-751.

[11] A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro, "Show, attend and interact: Perceivable human-robot social interaction through neural attention Q-network," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017: IEEE, pp. 1639-1645.

[12] I. Sorokin, A. Seleznev, M. Pavlov, A. Fedorov, and A. Ignateva, "Deep attention recurrent Q-network," *arXiv preprint arXiv:1512.01693*, 2015.

[13] Y. Ozaki, T. Ishihara, N. Matsumura, and T. Nunobiki, "Can user-centered reinforcement learning allow a robot to attract passersby without causing discomfort?," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019: IEEE, pp. 6986-6992.

[14] S. Lathuilière, B. Massé, P. Mesejo, and R. Horaud, "Deep reinforcement learning for audio-visual gaze control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018: IEEE, pp. 1555-1562.

[15] I. D. Gebru, S. Ba, X. Li, and R. Horaud, "Audio-visual speaker diarization based on spatiotemporal bayesian fusion," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 5, pp. 1086-1099, 2017.

[16] G. Brockman *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[17] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[18] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[19] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, 2016, vol. 30, no. 1.

[20] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[21] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115-133, 1943.

[22] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807-814.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, 2013, vol. 30, no. 1: Atlanta, GA, p. 3.

Shih-Huan Tseng received the B.S. and M.S. degrees in computer science from the Department of Computer Science, Tsing Hua University, Hsinchu, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in computer science from the Department of Computer Science & Information Engineering, National Taiwan University, Taiwan, in 2016.

He is currently an Assistant Professor with the Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan. His research interests fall into human-robot interaction, artificial intelligent, machine learning.

De-Yu Lin received the B.S. degree from the Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan, in 2021. He received the M.S. degree from the Department of Computer and Communication Engineering, National Kaohsiung University of Science and Technology, Kaohsiung, Taiwan, in 2023.

He has rich experience in machine learning, computer programming, system integration. His main areas of research are human-robot interaction.