2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Retargeting Human Facial Expression to Human-like Robotic Face through Neural Network Surrogate-based Optimization

Bowen Wu[1,2,*], Chaoran Liu[1,3], Carlos T. Ishi[1,3], Takashi Minato[1,3] and Hiroshi Ishiguro[2,3]

***Abstract*— Facial mimicry is crucial for human-like robots in human-robot interaction. The challenge is that the high diversity of facial expressions proposes difficulties in programming a robotic face to mimic human facial expressions using traditional methods. In this paper, we present a data-driven method to retarget human facial expressions to robotic faces without human effort. Our data collection is fully automatic, where only a robotic face and Apple ARKit are involved to sample actuator commands and record the resulting facial blendshape values. We trained a neural network that predicts blendshape values from commands, which is then used as a surrogate model to optimize command values to resemble given facial expressions. Experiments show that the proposed method has achieved lower error in terms of facial blendshape values than baselines. Moreover, the response time can be reduced to 0.2 seconds via TCP/IP through WiFi, offering great potential for real-time application. Our method is a novel framework for retargeting facial expressions to robotic faces, which can be incorporated into various human-robot interaction systems.**

I. INTRODUCTION

Facial expressions are essential in human-human interaction. People express feelings, emotions, or intentions through various facial expressions, and recognize facial expressions to understand others [1]. One crucial interactive ability based on facial expressions is facial mimicry, which is considered as expressing emotional empathy to others [2]. For an enhanced human-robot interaction (HRI), such ability is also required by human-like robots [3]–[6]. Not only do autonomous robots need to mimic human facial expressions to express sympathy just as humans do [7], but also teleoperated robots have to accurately reproduce the facial expression of the operator [8]. It is imperative to create a system that can accurately mimic human facial expressions for enhanced HRI.

Previous methods tried to choose a facial expression from a predefined set [9]–[15]. Other works developed a fitness function to select the nearest one from the predefined set of facial expressions [16], [17]. In practice, it is also possible to program the robotic face using predefined coefficients by utilizing high-level features such as facial blendshape. However, these methods require human expertise, which is expensive and time-consuming. More importantly, the main challenge is that the variability of human facial expressions is too high to cover using such traditional methods.

This work was supported in part by JST, Moonshot R&D under Grant JPMJMS2011.

[1]Guardian Robot Project, RIKEN, Japan
[2]Department of Engineering Science, Osaka University, Japan
[3]Hiroshi Ishiguro Labs, ATR, Japan
[*]E-mail: wu.bowen.research@gmail.com

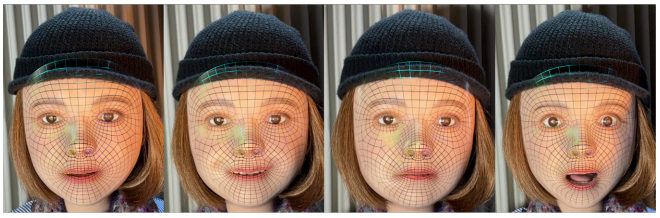

Fig. 1. Deformation in the wire frames indicates that ARKit successfully captures facial expressions of the robotic face used in our experiment. The colored wire frames on the face show the current face geometry captured by ARKit.

In this work, we present a data-driven method with high adaptability that does not require human expertise. Our main idea is to utilize existing facial data extraction systems, such as Apple ARKit[1] for facial blendshape values, to automatically collect facial data from a human-like robotic face. Since we can randomly sample faces using the robotic face, numerous data can be obtained for learning the mapping between command values and facial expressions. Empirically, we found that ARKit works well for the robotic face used in our experiment, as shown in Fig. 1. Moreover, the facial blendshape system used in ARKit shares common action units with the facial action coding system [18], which provides accurate modeling of human facial expressions.

To realize our idea, we first collected a dataset containing pairs of randomly sampled command values and the corresponding facial blendshape values extracted from the resulting robotic faces using ARKit. With this dataset, although a trivial way is to train a neural network to map blendshape values to commands, the random sampling of commands during the data collection can lead to a potential problem that causes the training of such a network to be difficult (V-A). Instead, we trained a neural network that predicts facial blendshape values from input commands. Using the trained network as a surrogate, any command values can be optimized based on the gradients derived from the error between the model's prediction and the target facial blendshape values. Our experiments on a robotic face that has 33 actuators show that the proposed method can successfully retarget human facial expressions to the robotic face, and outperforms other data-driven baselines. In addition, the latency of our system can be reduced to 0.2 seconds for a single face based on TCP/IP via WiFi, showing potential for real-time application.

[1]https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation

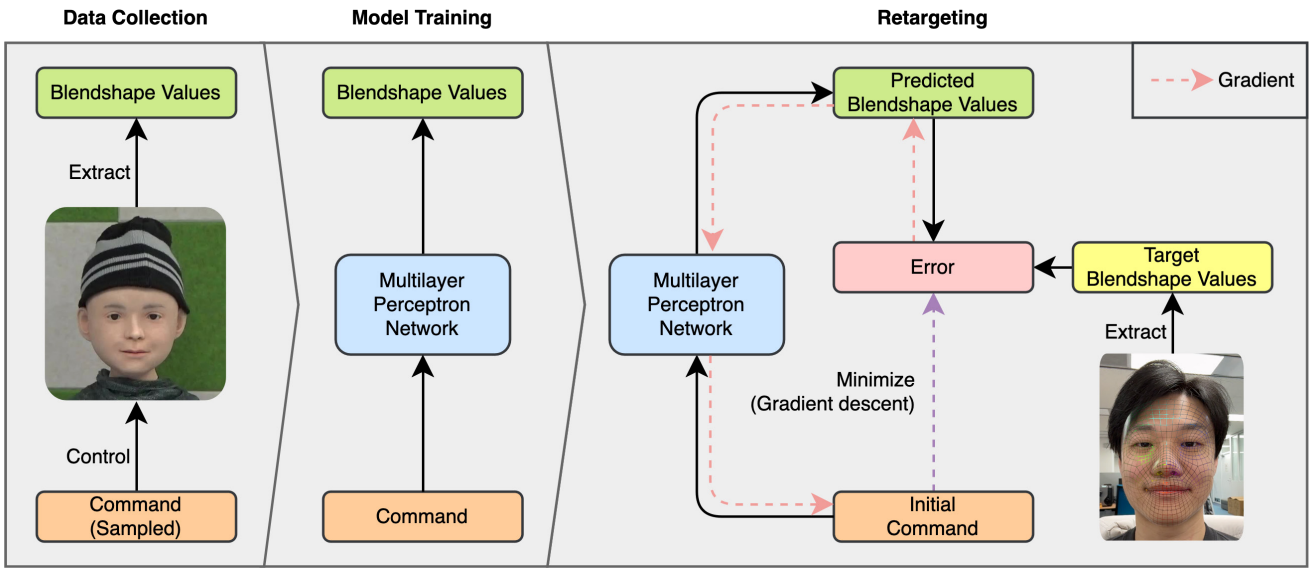

Fig. 2. Overview of our method. The robotic face on the left is used in our experiment (section IV). The colored wire frames on the human face show the current face geometry captured by ARKit, which is considered as being used to predict facial blendshape values (section III-A).

The rest of this paper is organized as follows: In section II, we briefly review research efforts on facial expression retargeting to robotic faces. Our proposed method is detailed in section III. The experiment is described in section IV. We present our discussion on the results in section V. Finally, we conclude our work in section VI.

II. RELATED WORKS

In the early stages, the retargeting from human facial expressions to robotic faces relied heavily on manual programming and human prior knowledge. Oh et al. [9] manually designed facial expressions for Albert HUBO based on a specific person. Itoh et al. [19] hand-crafted robot facial expressions to express certain emotions. Hashimoto et al. [11] used a life mask to allow the robotic face to mimic a real person. A robotic face with only 3 actuators was proposed by Asheber et al. [15] to simplify the programming of facial expressions for higher flexibility. Liu et al. [16] utilize extracted internal states from facial expressions, i.e., emotion, to search in their predefined set of robot facial expressions. Another previous work proposed to search for the best matching robotic face using a genetic algorithm [17]. In contrast to the above-mentioned works, our method is data-driven without human expertise and labeling, providing high adaptability to diverse facial expressions of different people.

The most similar work to ours is the one by Chen et al. [20], where they collected data from a robotic face and trained a generation model to generate robot face images from extracted face landmarks. They also trained an inference model to predict motor commands from a robotic face image. Our work differs from theirs in that: (1) We bridge the gap between the robotic face and human face using Apple ARKit; (2) We train a network to predict facial blendshape values from command values, and optimize the command value using the trained network; (3) The robotic face involved in our experiment has 32 actuators, whereas theirs has 10.

III. METHODOLOGY

The proposed method retargets human facial expressions to a human-like robotic face, by mapping the facial blendshape values extracted from the human face to the command values which are used to control the actuators in the robotic face. To achieve this, we trained a neural network that takes as input command values and outputs facial blendshape values. For given command values, one can compute the difference between the network's output and the target blendshape values. Its gradients with respect to the command values can be calculated by applying chain rule through the network parameters, which can then be used to optimize the command values via gradient-based algorithms such as gradient descent. By iterating this process, one can obtain the command values that minimize the difference between the predicted blendshape values and the target values. To this end, we collected a dataset that contains pairs of commands and the resulting facial blendshape values extracted from the robotic face using ARKit. An overview of the proposed method is shown in Fig. 2.

A. Data Collection

Given a human-like robotic face whose movements are controlled by actuators, random sampling in its actuators' command space can be performed to produce numerous faces. Moreover, due to its resemblance to the human face, facial blendshape capture systems such as ARKit can be used to extract blendshape from it. This allows the data collection of actuator commands and their resulting blendshape values on the robotic face.

We developed an iOS application (APP) that uses ARKit to extract blendshape values via the frontal camera, as shown

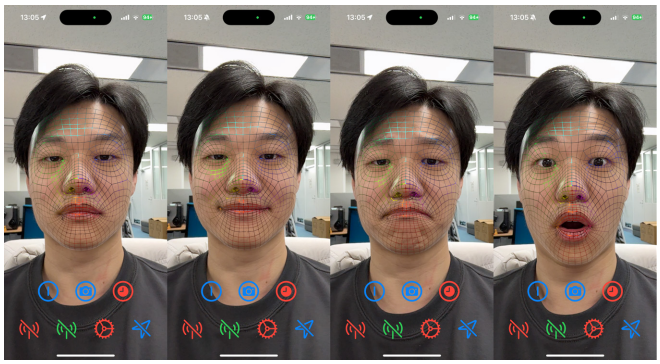

Fig. 3. Interface of the developed iOS application, showed with different facial expressions. The images are screenshots on an iPhone 15 Pro. The colored wire frames on the face show the current face geometry captured by ARKit.

in Fig. 3. An iPad Pro equipped with the APP was placed in front of the robotic face to record the blendshape value. We randomly sampled command values uniformly for all actuators simultaneously within the value range. For human-like robotic faces, there is usually silicone skin attached to the face. If there are actuators that pull the skin in the opposite direction, care must be taken to not tear up the skin during random sampling. To avoid this issue, for each sample, we simply randomly choose one side from the opposite actuators and zero out its command value. Under this constraint, sampled commands were sent to the robot control system via TCP/IP. We wait for the robotic face to reach the desired position before recording the blendshape value. Mostly, it takes less than 0.1 seconds to reach the desired position, but we wait for 1 second to avoid possible network latency.

In total, 20,000 samples are recorded for the robotic face used in our experiment, which has 35 actuators. We did not consider neck rotations thus there are a total of 32 actuators involved. Recorded samples are randomly split into train, validation, and test sets by 8:1:1. We also recorded the default blendshape values for the robotic face when it was at its resting position for calibration.

B. Neural Network Parametrization

A multilayer perceptron (MLP) is trained to predict the resulting blendshape values from command values. The network is trained to output the calibrated blendshape values, i.e., subtracted by the default blendshape values of the robotic face, rather than the raw blendshape value obtained directly from ARKit. This is convenient when matching the network's output to the calibrated blendshape values for different users. The calibrated blendshape values will be re-normalized to be within $[0.0, 1.0]$.

The command value is normalized to $[-1.0, 1.0]$ before inputting to the network. In our experiment, since the command value of our robotic face ranges from 0 to 255, we use the following formula for the normalization:

$$\tilde{x} = x/127.5 - 1, \tag{1}$$

where x stands for the command value of a single actuator. Since the blendshape values are within $[0.0, 1.0]$, we apply a hard sigmoid function on the output of the network to constrain its output to be within $[0.0, 1.0]$. The choice of hard sigmoid instead of frequently used sigmoid is because hard sigmoid provides a constant derivative with respect to its input outside its saturation range, which is preferable when calculating gradient using the network. For completeness, the definition of hard sigmoid is shown below:

$$\text{Hardsigmoid}(x) = \begin{cases} 0 & \text{if } x \leq -3, \\ 1 & \text{if } x \geq +3, \\ \frac{x}{6} + \frac{1}{2} & \text{otherwise.} \end{cases} \tag{2}$$

Formally, denoting command values for all actuators as $\boldsymbol{x} \in \mathcal{R}^C$ and the predicted facial blendshape values as $\hat{\boldsymbol{y}} \in \mathcal{R}^B$, where C represents the number of actuators and B represents the number of blendshapes, the network is defined as follows:

$$\hat{\boldsymbol{y}} \leftarrow \text{Hardsigmoid}(\text{MLP}(\tilde{\boldsymbol{x}})), \tag{3}$$

where $\tilde{\boldsymbol{x}}$ represents the normalized command values defined in (1). The network first projects $\tilde{\boldsymbol{x}}$ to the hidden space and then processes it through hidden layers sequentially. There is a skip connection in each layer for more stable gradients with respect to the network parameters [21]. A final fully-connected layer projects the output of hidden layers to the vector space which has the same dimension as blendshape values. Activated by the hard sigmoid function, it is used as the network output $\hat{\boldsymbol{y}}$.

To train the network, smooth L1 loss [22] is adopted since it penalizes outliers less than other Euclidean distances such as L2 loss, thus can mitigate gradient explosion. The error between the predicted blendshape values $\hat{\boldsymbol{y}}$ and the ground truth blendshape values, denoted as $\boldsymbol{y}$, is calculated as follows:

$$\boldsymbol{l}_i = \begin{cases} 0.5(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^2/\delta & \text{if } |\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i| < \delta, \\ |\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i| - 0.5 \cdot \delta & \text{otherwise,} \end{cases} \tag{4}$$

where i represents the index of each blendshape and δ is set to 1 in our experiment. The network parameters are optimized via gradient descent by taking the average over all samples in a batch and over all dimensions as follows:

$$\mathcal{L} = \frac{1}{NB} \sum_{n}^{N} \sum_{i}^{B} l_i^{(n)}, \tag{5}$$

where n represents the n-th sample.

The network is trained using Adam optimizer [23]. The learning rate decayed by 0.1 when the validation loss reached a plateau for 40 epochs consecutively. Hyperparameters for model architecture and training are determined by performing a grid search to minimize the validation loss. We used early stopping to prevent overfitting.

C. Optimization of Command Value

Using the trained network described in the previous section, one can predict the resulting blendshape values from command values. For given command values and target blendshape values, the difference between the prediction based on the command values and the target blendshape values can be computed. The gradients of the difference with respect to the command values can be calculated, which can then be used to update the command values via gradient descent. By iterating this process, the command values that minimize the difference between the predicted blendshape values and the target can be obtained.

Formally, denoting the target blendshape values as $\boldsymbol{y}$ and the given command values as $\tilde{\boldsymbol{x}}$, the difference can be defined as follows:

$$d = dist(\boldsymbol{y}, \text{Hardsigmoid}(\text{MLP}(\tilde{\boldsymbol{x}}))), \tag{6}$$

where d is a scalar and $dist$ represents an arbitrary distance function between two vectors. In our experiment, we use mean absolute distance defined as follows:

$$dist(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{N}\sum_{i}^{N} |\boldsymbol{x}_i - \boldsymbol{y}_i|, \tag{7}$$

where i represents the index of one dimension of a vector. A gradient descent step with respect to the command values can be defined as:

$$\tilde{\boldsymbol{x}} \leftarrow \tilde{\boldsymbol{x}} - \alpha \nabla_{\tilde{\boldsymbol{x}}} d, \tag{8}$$

where α is the learning rate. Empirically, we found that using advanced gradient-based optimization methods such as Adam [23] can improve the performance and reduce the updating steps. Finally, the optimized command values are un-normalized and quantized to the original command value space by performing the inverse of (1). If any produced command value exceeds the valid range, we clip it to be within $[0, 255]$.

We use the default command values where the robotic face is at the rest position as the starting point for the optimization. The learning rate is determined by trying out different values to select the one that minimizes d defined in (6). During the optimization process, it is possible that command values exceed the valid range before reaching the maximum step. We compared clipping them at intermediate steps and clipping the final result and found that the latter performs better. The choice of the number of optimization steps will be discussed in section IV-C.

D. Retargeting Human Face to Robotic Face

Following our data collection method, the iPad frontal camera along with ARKit is used to extract facial blendshape values from the human face. The obtained raw blendshape values cannot be directly used to optimize the command value because people have their own default blendshape value offset, which can cause the retargeting to be biased. For example, for a person who has small eyes, the extracted blendshape values for closing eyes[2] will be larger than zero, which will result in command values that control the robotic face to close its eyes. As a result, although the person is not closing eyes, a closing eyes facial expression will be performed on the robotic face, which is not desired.

To mitigate the above issue, calibration is performed whenever a user starts to use the system. In the developed iOS APP, there is a button for starting calibration. Once the button has been clicked, 10 frames of facial blendshape values will be recorded within 1 second with an interval of 0.1 seconds. If the variance of the blendshape values among these frames does not exceed a certain threshold, the average of them will be taken and used as the default blendshape values for the current user. The user is also required to look at the frontal camera during the calibration. For each obtained raw facial blendshape value, we subtract the calibration values from it and use the resulting values as the optimization target to optimize command values using the method described in the previous section.

Before sending the obtained command values to the robotic face, constraints on the robotic face must be considered. As mentioned in section III-A, actuators that pull the silicone skin on the robotic face in the opposite direction must not be activated together, otherwise it will tear up the skin. In such cases, we subtract the larger one by the smaller one and zero out the smaller one. Finally, the processed command values can be safely sent to the robotic face control system to realize the movement.

IV. EXPERIMENT

The robotic face used in our experiment is called Nikola, which is capable of performing diverse facial actions [24]. It has 32 pneumatic actuators in its face for facial expressions, and 3 in its neck to control head raw, pitch, and yaw. A silicone skin with a human-like appearance is covered on its face. Images of Nikola are shown in Fig. 1.

A. Baselines

Several data-driven baseline models were used to contrast our method, namely nearest neighbor, linear regression, and a neural network trained to predict command values from blendshape values. Firstly, the nearest neighbor is realized by calculating the L1 distance between the target blendshape values and the blendshape values of all samples in the training set. The minimum one will be selected as the prediction. Secondly, the linear regression fits the coefficients for command values by minimizing the residual sum of squares between its prediction and the ground truth blendshape values. Thirdly, a neural network is trained to predict command values from blendshape values. A grid search was conducted to find the optimal hyperparameters for the network, where the goal is to minimize validation loss. Finally, random sampling of command values is also included as a reference.

[2]The corresponding coefficients are called eyeBlinkLeft and eyeBlinkRight. See `https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation`

B. Evaluation Metric

It is difficult to measure the error at the command level such as command accuracy as in previous works [20]. For real human faces, the ground truth command value is unknown. Even for our robotic face dataset where we have the ground truth, there are unpredictable command values and thus cannot be used as ground truth to compute the error, as discussed in section V-A.

To quantify the error of the retargeting, we relied on ARKit for facial blendshape value extraction to measure the error at the blendshape level, which is similar to landmark distance used in [20]. For target blendshape values, the predicted command values are sent to control Nikola. The blendshape values of the controlled face are extracted to be compared with the target. The absolute errors on all blendshapes for one sample are summed and denoted as blendshape distance (BD). However, this error will be biased by the blendshapes with lower variance since different blendshape values vary at different extents. Therefore, we weight the error for each blendshape by their standard deviation as opposite to the summation of standard deviations of all blendshapes, based on the ground truth data. This weighted error is denoted as weighted-BD (wBD). In addition, we report the 95% confidence interval of the error.

C. Number of Optimization Steps

The number of optimization steps is set to 200 in our experiment. There is a trade-off between the number of optimization steps and the time cost as shown in Fig. 4. Although with more steps the performance can be improved, we want to keep the latency as low as possible for the system to run in real-time. From our observation, a latency of 0.2 seconds is acceptable for real-time retargeting, which is similar to 0.18 seconds in a previous work [20].

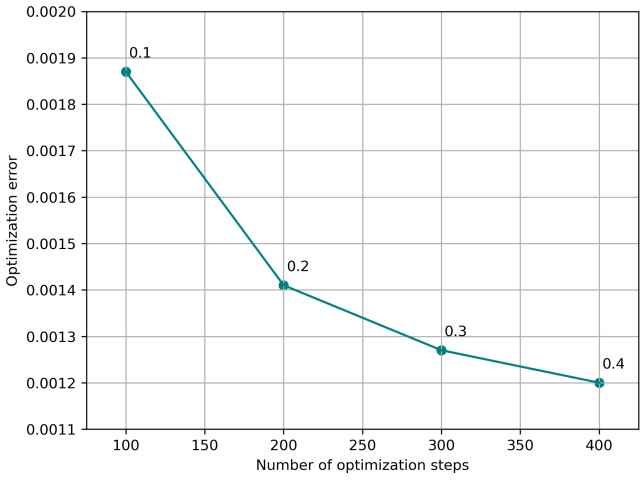

Fig. 4. Optimization error when using different numbers of optimization steps on the validation set. The time cost of the optimization for different number of optimization steps is annotated on its top.

D. Evaluation of Robotic Faces

We evaluated our model using our collected robotic face dataset described in section III-A. There are 2000 samples in the test set. For blendshape values in the dataset, different methods were used to predict the command values, which will then be sent to Nikola to control its face. We extracted the blendshape values from the resulting face and calculated the error with respect to the target. The errors of all samples were averaged and listed in TABLE I.

TABLE I
EVALUATION RESULTS ON COLLECTED ROBOTIC FACES. NN: NEURAL NETWORK.

Method	BD ↓	wBD ↓
Random	2.762 ± 0.168	4.756 ± 0.354
Nearest neighbor	1.363 ± 0.065	1.769 ± 0.124
Linear regression	1.331 ± 0.095	1.788 ± 0.216
NN for blendshape to command	2.598 ± 0.078	4.425 ± 0.207
Ours	1.233 ± 0.065	1.52 ± 0.112

E. Evaluation of Human Faces

To evaluate the performance in the real world, we collected a human face dataset from people in our laboratory, using the iOS APP we developed. After calibration, users click a button on the screen to record the current blendshape values. The users are instructed to perform natural facial expressions with noticeable movements. Totally, we collected 181 facial expressions from 7 people (5 males and 2 females). Using the collected blendshape, the same procedure in the robotic face evaluation was performed to evaluate the model. The obtained results are listed in TABLE II. We show some retargeting results of our method in Fig. 5.

TABLE II
EVALUATION RESULTS ON HUMAN FACES. NN: NEURAL NETWORK.

Method	BD ↓	wBD ↓
Random	4.634 ± 0.385	5.977 ± 0.525
Nearest neighbor	3.348 ± 0.393	4.152 ± 0.534
Linear regression	3.146 ± 0.401	3.85 ± 0.555
NN for blendshape to command	4.169 ± 0.35	5.292 ± 0.462
Ours	2.937 ± 0.381	3.581 ± 0.511

F. Real-time Performance

We set up a real-time system based on TCP/IP through WiFi for the environment in our laboratory. After calibration, our iOS APP initiates a loop, within which the APP extracts blendshape values from the user's face and sends them to the GPU machine for optimization, and the GPU machine sends the solved command values back to the APP after the optimization has finished. The APP will not initiate the next loop until it receives all responses from the GPU machine to avoid jamming the channel due to the occasional instability of the network. In addition, we incorporated an exponential moving average to eliminate high-frequency perturbations in the solved command values. For a smoother transition, we

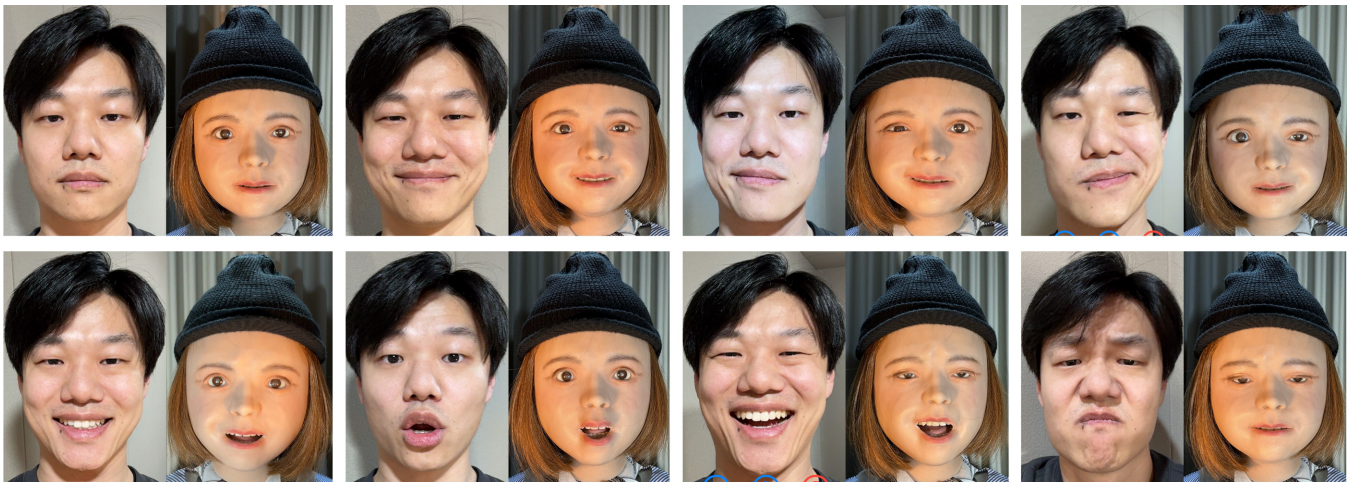

Fig. 5. Retargeting results of our method. In each pair, the target face is on the left and the retargeted face is on the right. The most left upper one was used for calibration.

applied a linear interpolation from the previous command values to the next command values 4 times. Thus, the actual interval of controlling the robotic face is 0.05 seconds. Under this setting, the duration between the initiation of each loop is around 0.2 seconds, which is the response time of our system.

V. DISCUSSION

A. Challenges in Training Neural Networks to Predict Command from blendshape

The most straightforward way of developing a data-driven retargeting system from facial blendshape value to command value is to train a neural network to predict the command value from the facial blendshape value. However, we found that this is not ideal theoretically and empirically. Although we collected data for all 32 actuators that control the robotic face, we noticed that some of them do not affect facial blendshape values. Considering that we randomly sampled command values from a uniform distribution for all actuators during the data collection, training a neural network to predict these uncorrelated command values from blendshape values is the same as training the network to predict random noise. While we can pick them out and remove them, they are kept intact because we want a generic method that involves as little human effort and prior knowledge as possible.

A disadvantage of this is that the loss value will be uninformative. Since the network cannot accurately predict the noise, the loss of these command values will always be large. Therefore, we cannot know the real performance of the model by looking at the loss. This also adds difficulty when comparing different models. Even if there is a difference in predicting commands that can be predicted, it will not be perceivable because the loss for the commands that can be predicted will be much lower than those that cannot.

In practice, one can still use a converged model trained to predict command values from blendshape values without caring much about the loss. We also tried but the results are not satisfying as shown in our experiments. Several trained models were attempted directly on the robotic face. However, visualization of the robotic face shows that each model has its own bias, i.e., they produce some unnatural faces regardless of the input blendshape values. One possibility is that these models have overfitting yet it is not noticeable due to the magnitude of the loss of those predictable command values. This is further supported by our experimental results, where even a simple linear regression outperforms the trained network.

Although it is difficult to directly train a network to predict command values from blendshape values, it is easy to train a network to predict blendshape values from command values akin to forward kinematics, because blendshape values can only change when command values change. When the network is trained to predict blendshape values from command values, it can ignore those actuators that are of less importance, e.g., assign lower weights to them. A lower weight implies that the gradients with respect to them will be around zero when calculating gradients through the trained network. This allows gradient-based optimizations with respect to command values to automatically ignore those actuators that do not affect blendshape values. For example, in our robotic face, there is an actuator that controls the tongue. However, even when the command value for sticking out the tongue reaches its maximum, it is still in the mouth and thus cannot be reflected in the recorded blendshape value[3]. In our experiment, we found that the derivative for the command value of the tongue is near zero regardless of the target blendshape values. Although this is indeed a limitation that our model cannot predict tongue-out movement, it is acceptable since the used robotic face cannot perform tongue-out in the first place. Due to the above analysis, we decided to develop an optimization-based method to retarget facial blendshape values to command values.

[3]https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation/2968190-tongueout

B. Limitations and Future Works

The most significant limitation of our work is the use of Apple devices, which are expensive and the algorithms are not open-sourced. There are also other systems for extracting facial blendshape values, such as MediaPipe [25]. Our future plan involves testing the compatibility between our system and their outputs. Furthermore, there are also face geometric extraction systems in MediaPipe that can be adapted for our purpose, which can potentially allow us to model more intricate facial expressions. We leave these developments as future works.

User calibration is necessary for using our method. Although calibration is important for retargeting systems, in real robot-human interaction scenarios, it is not possible to ask the user to calibrate. We will seek methods that can automatically calibrate to realize seamless real-time interaction.

Our method may suffer from facing direction, due to the instability of the facial blendshape extraction algorithm of ARKit. Other methods like face landmarks or geometric also suffer from this. One important future work is to increase the robustness of the system for various facing directions.

We are not able to use more optimization steps for improved performance due to real-time requirements. An ideal way would be to have a network doing the inference only one time instead of an optimization approach. Unfortunately, we have proved that direct training of such networks is not feasible under the current settings. In the future, we will continue research on realizing fast and accurate retargeting.

VI. CONCLUSION

Facial mimicry is a crucial ability for human-like robots in HRI. In this paper, we presented a data-driven optimization-based method that retargets human facial expressions to a robotic face in real-time. Experimental results show that the proposed method is superior to other baseline models. Our method shows great potential in avoiding great efforts of involving human expertise since the training data are collected using only the robotic face. While currently, our method is facing limitations, there is also evidence that it has the potential to achieve a greater performance, which is promising to be integrated into real-world HRI systems. Furthermore, the present method acts as a novel framework for facial expression retargeting to robotic faces, which provides high adaptability and can be incorporated for various purposes in HRI.

REFERENCES

[1] K. Scherer, "What does a facial expression express," 1992.

[2] H. Drimalla, N. Landwehr, U. Hess, and I. Dziobek, "From face to face: the contribution of facial mimicry to cognitive and emotional empathy," *Cognition and Emotion*, vol. 33, no. 8, pp. 1672–1686, 2019.

[3] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and autonomous systems*, vol. 42, no. 3-4, pp. 143–166, 2003.

[4] M. Blow, K. Dautenhahn, A. Appleby, C. L. Nehaniv, and D. Lee, "The art of designing robot faces: Dimensions for human-robot interaction," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pp. 331–332, 2006.

[5] R. Hakli, "Social robots and social interaction," in *Sociable robots and the future of social relations*, pp. 105–114, IOS Press, 2014.

[6] S. Saunderson and G. Nejat, "How robots influence humans: A survey of nonverbal communication in social human–robot interaction," *International Journal of Social Robotics*, vol. 11, pp. 575–608, 2019.

[7] C. Mayer, S. Sosnowski, K. Kühnlenz, and B. Radig, "Towards robotic facial mimicry: system development and evaluation," in *19th International Symposium in Robot and Human Interactive Communication*, pp. 198–203, IEEE, 2010.

[8] S. Nishio, H. Ishiguro, and N. Hagita, "Geminoid: Teleoperated android of an existing person," *Humanoid robots: New developments*, vol. 14, no. 343-352, pp. 10–1109, 2007.

[9] J.-H. Oh, D. Hanson, W.-S. Kim, Y. Han, J.-Y. Kim, and I.-W. Park, "Design of android type humanoid robot albert hubo," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1428–1433, IEEE, 2006.

[10] T. Hashimoto, S. Hitramatsu, T. Tsuji, and H. Kobayashi, "Development of the face robot saya for rich facial expressions," in *2006 SICE-ICASE International Joint Conference*, pp. 5423–5428, IEEE, 2006.

[11] T. Hashimoto, S. Hiramatsu, and H. Kobayashi, "Dynamic display of facial expressions on the face robot made by using a life mask," in *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pp. 521–526, IEEE, 2008.

[12] H. S. Ahn, D.-W. Lee, D. Choi, D.-Y. Lee, M. Hur, and H. Lee, "Designing of android head system by applying facial muscle mechanism of humans," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pp. 799–804, IEEE, 2012.

[13] D. Loza, S. Marcos Pablos, E. Zalama Casanova, J. Gómez García-Bermejo, and J. L. González, "Application of the facs in the design and construction of a mechatronic head with realistic appearance," 2013.

[14] C.-Y. Lin, C.-C. Huang, and L.-C. Cheng, "An expressional simplified mechanism in anthropomorphic face robot design," *Robotica*, vol. 34, no. 3, pp. 652–670, 2016.

[15] W. T. Asheber, C.-Y. Lin, and S. H. Yen, "Humanoid head face mechanism with expandable facial expressions," *International Journal of Advanced Robotic Systems*, vol. 13, no. 1, p. 29, 2016.

[16] N. Liu and F. Ren, "Emotion classification using a cnn_lstm-based model for smooth emotional synchronization of the humanoid robot ren-xin," *PloS one*, vol. 14, no. 5, p. e0215216, 2019.

[17] H.-J. Hyung, H. U. Yoon, D. Choi, D.-Y. Lee, and D.-W. Lee, "Optimizing android facial expressions using genetic algorithms," *Applied Sciences*, vol. 9, no. 16, p. 3379, 2019.

[18] P. Ekman and W. V. Friesen, "Facial action coding system," *Environmental Psychology & Nonverbal Behavior*, 1978.

[19] K. Itoh, H. Miwa, M. Zecca, H. Takanobu, S. Roccella, M. C. Carrozza, P. Dario, and A. Takanishi, *Mechanical design of emotion expression humanoid robot WE-4RII*. Springer, 2006.

[20] B. Chen, Y. Hu, L. Li, S. Cummings, and H. Lipson, "Smile like you mean it: Driving animatronic robotic face with learned models," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2739–2746, IEEE, 2021.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pp. 630–645, Springer, 2016.

[22] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

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

[24] W. Sato, S. Namba, D. Yang, S. Nishida, C. Ishi, and T. Minato, "An android for emotional interaction: Spatiotemporal validation of its facial expressions," *Frontiers in psychology*, vol. 12, p. 800657, 2022.

[25] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, *et al.*, "Mediapipe: A framework for building perception pipelines," *arXiv preprint arXiv:1906.08172*, 2019.