Inverse Kinematics Solver for an Android Face using Neural Network

*Emarc MAGTANONG, Akihiko YAMAGUCHI, Kentaro TAKEMURA, Jun TAKAMATSU and Tsukasa OGASAWARA (Nara Institute of Science and Technology)

1. Introduction

An important part of natural human-robot interaction is the ability of robots to display facial expressions. One of the main goals in android research is to enable androids to show facial expressions for a more natural human-robot interaction [1]. Many researches have been done to incorporate facial expressions on android robots by designing androids equipped with deformable faces [2] [3] [4]. However, controlling facial expressions on androids is still an open problem due to the lack of an inverse kinematics solver for android faces.

The purpose of this research is to provide a method for controlling facial expressions of androids by specifying target feature point positions. A feature point is defined as a specific position on the android's facial skin surface that moves with displacing facial actuators. In general, the goal of the research is to create an inverse kinematics solution to android faces. In this research, we define the inverse kinematics as the process of determining the actuator displacements given the target feature point positions.

The problem in solving for the inverse kinematics of android faces is that the facial skin surface is deformable. This results to the feature points to move with each other. Thas is, there is *coupling* between the feature points. Therefore, it is difficult to model the forward kinematics of the face since feature points are not fixed on a rigid link. Also, specifying target feature points for the inverse kinematics is complicated because of the coupling problem.

Two ideas are presented in this paper for the solution to the inverse kinematics of android faces. Initially, we model the forward kinematics of the face by using a neural network. The forward kinematics model is defined as determining the feature point positions given the actuator displacements. We use neural networks because it is capable of learning the complex forward kinematics due to the feature points being on a deformable surface. The feature point positions are captured using a motion capture system. The output of the motion capture system is the feature point position in space. Next, using an iteration technique, we compute for the actuator displacements that satisfies the specified target feature point positions. In addition, we introduce a weighting method for computing the error between the target and computed feature point positions to handle the coupling problem by emphasizing the contribution of each feature points. Both the neural network and the iteration technique are used to create an inverse kinematics solver for an android's face.

To test the proposed inverse kinematics model, we conducted several experiments using the constructed inverse kinematics solver. We verified the performance of the inverse kinematics solver using two types of target feature points. First, target feature point positions are created using random actuator displacements and captured using a motion capture system. This will test the performance of the inverse kinematics solver when the coupling of feature points is maintained. Next, target feature point positions are created manually by displacing a single feature point or several feature points in order to assess if the weighting error function is suitable for handling the coupling problem.

Several researches have been done to control android facial expressions from target feature point positions. Jaeckel et al. presented a method for controlling facial expressions on androids by mapping facial expressions from video footages [5]. They used active appearance models to capture 2D positions of feature points and multilinear Partial Least Squares regression for converting feature point positions to actuator displacements. Another research done by Wilbers et al. used a technique in computer animation called *blendshapes* for mapping human facial motion capture data to an android [6]. In their method, target feature point positions are computed as linear transforms of key feature point positions called blendshapes models. The intermediate feature point positions are interpolated from the blendshape models. Then actuator displacements are obtained by computing the weights of each of the blendshapes. The difference of our proposed method with the previous methods is that, our inverse kinematics solver can provide a proper solution even to infeasible target feature point positions.

Researches have also been done to model the inverse kinematics of manipulators using machine learning techniques. A research done by Xia et al. uses neural networks to model the inverse kinematics between the target end effector position and manipulator joint angles [7]. For the neural network to learn the inverse kinematics, it is trained using sets of target end effector positions as the input and joint angles as the output. After training, the network can solve for the joint angles given the target end effector position.

The rest of this paper is organized as follows. Section 2 discusses the method we use to solve the inverse kinematics of an android face. Section 3 describes the details of the experiments we performed for evaluating the forward kinematics model and inverse kinematics solver. Lastly, section 4 concludes the paper.

2. Inverse Kinematics Solver for an Android Face

As stated in section 1, the purpose of this research is to provide a method for solving the inverse kinematics of android faces. In order to create the model, we first employ a neural network to learn the inverse kinematics between the actuator displacements and feature point positions wherein the network's input are the actuator displacements and the output are the feature point positions. We collect training data using a motion capture system to track the feature point positions.

Next, using an iteration technique, we compute the actuator displacements that minimizes the error between the target and computed feature point positions solved by the neural network. Here, in order to handle the coupling problem, localized weights for each of the feature points are introduced into the error.

2·1 Forward Kinematics using Neural Network

To model the forward kinematics of the face, we train a neural network to learn the forward kinematics between the actuator displacements and feature point positions. The neural network has 3 layers which is comprised of the input, hidden, and output layer. The activation function of the hidden layer neurons is the logistic sigmoid function.

The forward kinematics neural network function is defined as,

$$\mathbf{x} = NN(\mathbf{u}, \Theta), \tag{1}$$

where, Θ denotes the weights between the neurons and is optimized during training. In addition, $\mathbf{u}$ represents the input vector of actuator displacements and $\mathbf{x}$ defines the output vector of feature point positions.

For training the neural network, we collect data using a motion capture system to record the feature point positions. That is to say,

$$\begin{aligned} \mathcal{D} &= \{\mathbf{u}_n, \mathbf{x}_n | n = 1, 2, ..., N_s\}, \\ \mathbf{u} &= (u_1, u_2, ..., u_{N_a}), \\ \mathbf{x} &= \left(x_1, y_1, z_1, ..., x_{N_f}, y_{N_f}, z_{N_f}\right). \end{aligned} \tag{2}$$

where, N_{s} is the total training samples for the network and N_{a} and N_{f} represent the number of actuators and feature point vectors respectively.

To train the network, we input these training data to the neural network to minimize the error function,

$$E(\Theta) = \frac{1}{2}\sum_{n=1}^{N_s} \|NN(\mathbf{u}_n, \Theta) - \mathbf{x}_n\|^2, \ \text{w.r.t.}\ \Theta. \tag{3}$$

To avoid the problem of overfitting, we separate the data into a training and a validation set and apply an early stopping method during the training iterations [8].

2·2 Solving for the Inverse Kinematics

This section discusses our proposed method to solve for the inverse kinematics of the android's face. We aim to deal with two problems. First, the difficulty of deducing an analytic solution to the inverse kinematics when using a neural network. Second, is the problem of handling the coupling of the feature points. To consider these problems, we formulate the inverse kinematics as the minimization of the weighted square error of the feature point positions with respect to the actuator displacements, that is,

$$\begin{aligned} &\min_{\mathbf{u}} \quad \sum_{i=1}^{N_f} \mathbf{w}_i \left[NN(\mathbf{u}) - \mathbf{x}_i^*\right]^2, \\ &\text{where,} \quad u_{\min_{\mathrm{N_a}}} \le u \le u_{\max_{\mathrm{N_a}}}, \end{aligned} \tag{4}$$

$\mathbf{x}^*$ denotes the target feature point positions and $\mathbf{w}$ is the weight vector for each feature points. The weight vector handles the coupling problem by emphasizing the error contribution of each feature points.

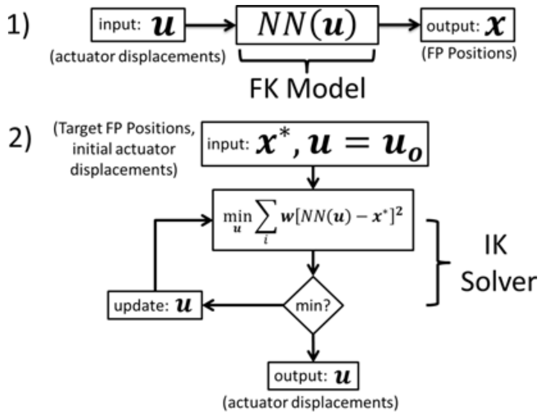

Fig.1 System diagrams of forward kinematics (FK) model (1) and the inverse kinematics (IK) solver (2) for android faces.

Fig.2 The Actroid-SIT android.

The error function is minimized using a gradient descent method with multiple starting points for $\mathbf{u}$. Using multiple starting points increases the chance of finding the global minimum.

Combining the two methods proposed in section 2·1 and 2·2, we can design the inverse kinematics solver that computes for the actuator displacements given target feature point positions. The diagrams of the proposed forward kinematics model and inverse kinematics solver is shown in **Fig.**1 where $\mathbf{x}^*$ is the vector of target feature point positions and $\mathbf{u}$ is the vector of actuator displacements.

3. Experiments of Controlling an Android Face

3·1 Actroid-SIT

The android we used for the evaluation our method is the Actroid-SIT android manufactured by Kokoro, Co. The skin is constructed from silicone. The android is shown in **Fig.**2.

It has 42 pneumatic actuators that can move the robot from the waist up. The lower part of the android is not actuated. The android's face has 13 actuators. The input of each actuator is a target value of the actuator displacements which ranges from 0 to 255.

To simplify the modeling for the inverse kinematics solver, we limit the number of actuators to control. Specifically, we designed an inverse kinematics solver to control five actuators, A6, A7, A8, A11, and A12, which controls the lips. Actuators A9 and A10 were

Table 1 Movement of Actroid Facial Actuators

Actuator	Movement
A1	eyebrows raise (both)
A2	eyelids close (both)
A3,A4	left/right horizontal eye gaze
A5	vertical eye gaze/eyelids raise
A6	jaw open
A7,A8	pull upper/lower lip upward
A9,A10	pout upper/lower lip
A11,A12	left/right lip corner pull
A13	raise cheeks

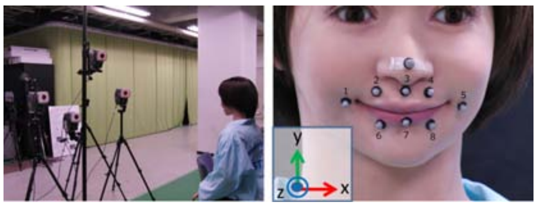

Fig.3 Motion capture setup (left) and marker layout (right) used in the motion capture experiments.

not included since they have very small contributions to the movement of lip feature points (see **Table** 1).

3·2 Capturing Feature Point Positions

We capture feature point positions using a motion capture system. These data are used to train the neural network in section 2·1.

Several passive reflective markers are attached to the android's face which serves as the feature points (shown in **Fig.**3). The markers are placed on the face where significant feature point movement occurs when the actuators are displaced. The marker attached on the nose is used as the origin point to have a uniform reference coordinate for all captured data.

We place the android robot in the center of the capture volume to ensure maximum coverage for each markers.

We record the position of the feature points while keeping the actuator displacements stationary. To consider different actuator configurations, we captured 51 samples of single actuator displacements, 243 samples for combinations of actuator displacements, and 300 samples of random actuator displacements. An additional 1000 samples of random actuator displacements are captured to be used for testing the generalization of the neural network.

3·3 Evaluation of the Forward Kinematics Model

Since the forward kinematics of the actuator displacements and feature point positions is modeled using a neural network, it is important to test the generalization ability of the network. The generalization ability of the neural network is the measure on how well the neural network estimates the output when presented with input data not used during training.

In the case of the forward kinematics model in section 2·1, we test the generalization ability by inputting target feature point positions created using the random actuator displacements. This will test if the neural network overfits the training data.

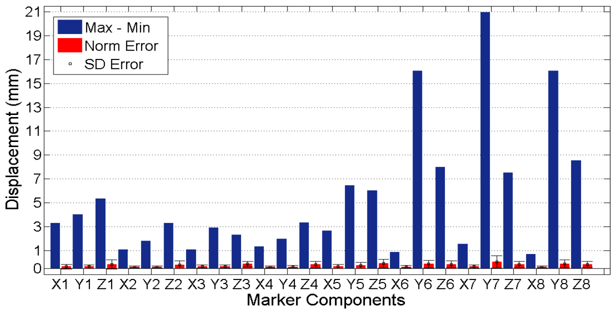

Fig.4 Bar plot of the movement range of the feature points (left bar) and the norm error with standard deviation of feature point positions computed using the forward kinematics model (right bar).

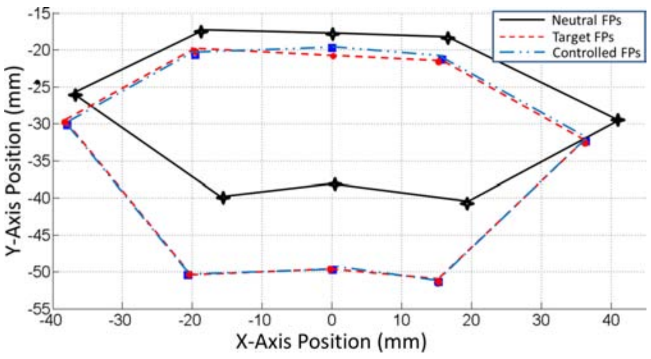

Fig.5 Plot of the target feature points (target FP) using data captured from random actuator displacements (dashed) and controlled feature points (controlled FP) from solver output applied to the android (dotted dashed). The Z-axis is omitted.

Fig.6 Image comparison of the resulting face when the solver output are applied to the android for the case of target feature points captured from random actuator displacements.

We then computed the norm error of the neural network output with respect to the target feature point positions. Next, we plot the results of the norm error and compare them with the displacement range of each of the feature points. As shown in **Fig.**4 the error of the estimation is significantly smaller than the range of displacement of the feature points. This suggests that the neural network for the forward kinematics model has good generalization ability.

3·4 Evaluation of the Inverse Kinematics

This section discusses the experiments to evaluate the inverse kinematics model.

We performed two types of evaluation for the performance of the inverse kinematics solver. First, we

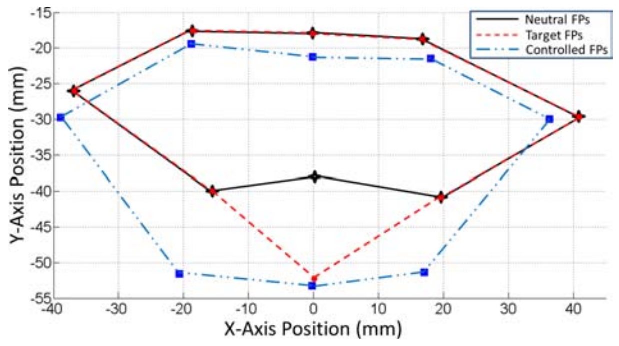

Fig.7 Plot of the target feature points (target FP) using a manually displaced feature point (dashed) and controlled feature points (controlled FP) from solver output applied to the android (dotted dashed). The Z-axis is omitted.

set the target feature point positions as feature points captured from random actuator displacements. In this test, the weight of each feature point is set to 1 since it can be assumed that such feature point positions are feasible. The target feature point positions are then input to the inverse kinematics solver and the actuator displacements output is applied to the android. The resulting feature point positions are captured in order to be compared with the target feature points. The feature points comparison is shown in **Fig.**5. Inspecting the plot, we see that the target feature point positions are close to the controlled feature point positions indicating that the inverse kinematics solver can perform well in computing for the actuator displacements.

In **Fig.**5, the neural feature point positions are also shown for the reference of the displacements of the feature points. The neutral feature point positions refer to the feature point positions when actuator displacements are 0. The difference of the positions of the neutral feature points with the target and controlled feature points is due to some noise and the reattachment of markers when the markers drop during the motion capture. However, referring to **Fig.**6, we can see that appearance of the android for the target and controlled feature point positions are similar.

The second test we conducted is to input target feature point positions when we move only a single or few feature points from the neutral feature point positions. This will verify if the weighting method used in the inverse kinematics solver can handle the coupling of the feature points. We assign the moved feature point weight to be 1 and the rest to be 0.01. We then input these target feature point positions and the weight vector to the inverse kinematics solver and apply the output to the android. Then we capture the resulting feature point positions. The plot of the controlled feature point positions in **Fig.**7 shows that the weighting method employed in the inverse kinematics model is able to handle the coupling of the feature points.

In the first evaluation, the resulting feature point positions is similar to the target feature point positions. As for the second scenarios, the coupling of the feature points is sufficiently handled by the weighting vector. This indicates that the solver can perform well on both cases.

4. Conclusion

This paper presented a method for modeling for the inverse kinematics of android faces. We proposed a solution to the problem of modeling the inverse kinematics of android faces such as the complexity in modeling the deformable facial skin surface and handling coupling of the feature points. To solve for the inverse kinematics, we first modeled the forward kinematics using a neural network to estimate the feature point positions from actuator displacements. Then we defined the inverse kinematics as the minimization of the error between the target feature point position and computed feature point positions estimated by the neural network. To handle the coupling of the feature points, we introduced a weighting method to emphasize the error contribution of each of the feature points. This solution to the coupling, enables the input of infeasible target feature point positions.

To evaluate our inverse kinematics method, we considered two cases for the inverse kinematics solver we implemented. The first scenario verifies the performance of the solver for estimating actuator displacements from target feature point positions that maintains the coupling of feature points. This assures that the target feature point positions are feasible for the android. The second case evaluates for the inverse kinematics solver performance when only specific feature points are moved. The displaced feature points are emphasized in the error computation using the weighting method we proposed. In both cases, the performance of the solver is acceptable based on the comparison of the target and resulting feature point positions when the solver output was applied to the android.

References

[1] R. Stiefelhagen, C. Fugen, R. Gieselmann, H. Holzapfel, K. Nickel, and A. Waibel, "Natural human-robot interaction using speech, head pose and gestures," in *Int. Conf. on Intelligent Robots and Systems*, vol. 3, pp. 2422–2427, IEEE, 2004.

[2] J. Oh, D. Hanson, W. Kim, Y. Han, J. Kim, and I. Park, "Design of android type humanoid robot albert hubo," in *Int. Conf. on Intelligent Robots and Systems*, pp. 1428–1433, IEEE, 2006.

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

[4] S. Nishio, H. Ishiguro, and N. Hagita, "Geminoid: Teleoperated android of an existing person," *Humanoid robots-new developments. I-Tech*, 2007.

[5] P. Jaeckel, N. Campbell, and C. Melhuish, "Facial behaviour mapping–from video footage to a robot head," *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1042–1049, 2008.

[6] F. Wilbers, C. Ishi, and H. Ishiguro, "A blendshape model for mapping facial motions to an android," in *Int. Conf. on Intelligent Robots and Systems*, pp. 542–547, IEEE, 2007.

[7] Y. Xia and J. Wang, "A dual neural network for kinematic control of redundant robot manipulators," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 31, no. 1, pp. 147–154, 2001.

[8] C. Bishop, *Pattern recognition and machine learning*, vol. 4. Springer New York, 2006.