Facial Expression Cloning with Fuzzy Membership Functions

by:

Patrick John Santos

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of

Master of Applied Science in Computer and Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Electrical Engineering and Computer Science
University of Ottawa

October 2013

Abstract

This thesis describes the development and experimental results of a system to explore cloning of facial expressions between dissimilar face models, so new faces can be animated using the animations from existing faces. The system described in this thesis uses fuzzy membership functions and subtractive clustering to represent faces and expressions in an intermediate space. This intermediate space allows expressions for face models with different resolutions to be compared. The algorithm is trained for each pair of faces using particle swarm optimization, which selects appropriate weights and radii to construct the intermediate space. These techniques allow the described system to be more flexible than previous systems, since it does not require prior knowledge of the underlying implementation of the face models to function.

Acknowledgements

I would like to thank my colleagues, family, and friends for their support in my endeavors that led to the completion of this work. I thank my research supervisor, Dr. Emil Petriu, for providing advice and insights over the course of this project. I thank my family for supporting me throughout my studies. I thank all of my colleagues at the Bio-Inspired Robotics Laboratory, DISCOVER Laboratory, as well as those who worked with me prior to undertaking my studies for sharing their experiences. I also credit all of my friends for their constant reminders that “unconventional” is fun and useful in the right context.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

This chapter introduces the motivation behind the system discussed in this thesis.

1.1. Motivation and Problem Statement

There have been many new efforts into creating computer systems that use facial expressions and other interfaces that are more natural to humans. These interfaces are part of the relatively new field of affective computing [1]. Social robots, which are robots designed to communicate naturally with humans, also implement facial expressions [2]. There are many works on facial expressions, but automated systems that implement them work in relatively specific situations, such as the requirement of similar types of faces. The lack of flexibility in face animation systems hinders their use when creators have limited resources because there are many variations on faces and facial expressions, and creating these animations without a starting point is difficult.

For the faces themselves, there are different implementations because the creation of a face is based on artistic preferences and psychology. In addition to designing a face for a particular audience, the Uncanny Valley problem also affects how humanlike a face should be [3]. The Uncanny Valley Problem is a hypothesis by Mori that stated that people can be uneasy with systems that are almost perceived to be human, but are not exactly human. The idea was that uneasiness occurs because the almost humanlike thing, like a robot or avatar, is perceived as a human with grave problems rather than an artificial entity that is not human. If a platform is perceived to have this property, it is said to have "fallen into" the Uncanny Valley, and there is a potential that users will not want to interact with that platform. This hypothesis was tested to have merit for virtual avatars [4] and robots [5]. Both virtual and robot faces have very different subjective appearances depending on how their creators choose to approach or avoid the Uncanny Valley problem.

In addition to differences between the faces, implementations of the facial expressions themselves are performed in different ways and are culture-dependent. Facial expressions can be represented as mixtures of typical expressions, also known as

"prototypical" or "basic" expressions of emotion [6]. They can also be represented by their underlying movements [7]. Facial expressions are also culture-dependent, with differences significant enough that these differences must be considered for creating media between cultures [8], and databases of facial expressions represent different cultures [9]. Since many different expressions are possible, some flexible notation should be used if a system is to be able to work with new expressions after its creation.

Due to the differences between faces and facial expression implementations, facial expressions created for one platform are difficult to transfer to other platforms. There is a need for facial expression cloning systems that can take facial expressions between multiple type of faces, including virtual avatar and robot implementations.

1.2. Objectives

The objective of the project discussed in this thesis was to build a facial expression cloning system to map facial expressions between parameterized faces regardless of the implementation of the face itself. The intention of this project was to extend it into an interface that allows an animator or actor to generate facial expressions for some virtual face and then transfer these expressions to other faces, whether the faces be robot or virtual faces. Therefore, databases of facial expressions or simple movements of the face can be generated for one platform, and these expressions can be enabled automatically on other platforms using a system such as the one proposed in this thesis.

1.3. Contribution

This thesis extends upon pre-existing concepts in facial expression cloning literature [10] [11]. It proposes the use of separate categories of fuzzy sets to classify both the static structure and dynamic movements of the face. Additionally, it proposes a variant of the particle swarm optimization velocity update, in which particles are repelled off the boundary of a constrained problem space at a randomized velocity. Finally, it extends the idea on the detection of error for cloning results which using a mapping from the source face, then to the target face, then back to the source face (A to B and back to A). It extends upon this idea by proposing a scaling factor to be able to compare results between different works.

Refereed conference publications:

Work leading to the creation of this thesis was published in two refereed conferences, listed below:

P. J. Santos, J. Erbene Maia, M. Goubran, and E. M. Petriu, "Facial Expression Communication for Healthcare Androids," in *2013 IEEE International Symposium on Medical Measurements and Applications (MeMeA 2013)*, 2013.

P. J. Santos and E. M. Petriu, "Facial Expression Cloning with Fuzzy Set Clustering," in *2013 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA 2013)*, 2013.

The conference papers presented the concepts of this system applicable for a robot and the use of fuzzy sets with clustering for facial expression cloning.

1.4. Organization of the Thesis

This thesis discusses background information on facial expressions, the proposed system, and the results of testing the proposed system. Chapter 2 discusses facial expressions and their representations, facial expression cloning, and prior art in the fields relevant to this project. Chapter 3 discusses the proposed system, its design, and the underlying algorithms that it uses to map facial expressions and optimize its parameters. Chapter 4 discusses the experiments and results that led to the final configurations of the system. Chapter 5 contains the conclusion and recommendations for future work in this system and in the field of facial expression cloning.

Chapter 2. Background Information

This chapter defines facial expressions for the purposes of this system and it presents a literature review of the various fields this project covers. The review encompasses the types of face models in both virtual space and robots, and facial expression cloning (FEC) algorithms that map facial expressions between different representations. It also gives a short description on fuzzy membership functions, which are used in this system to create an intermediate model for comparison of facial expressions.

2.1. Definition of Facial Expressions

A facial expression, for humans, is a temporary deformation of the face. Facial expressions are temporary in that the movements in the face have a finite duration. After the expression is over, the face returns to a neutral or "relaxed" state. The movements in the expression itself are caused by actions in the muscles of the face, and humans have 268 muscles that can contribute to facial expressions [12] [13]. These muscles exert forces on tissue on the face, deforming the face. Facial expressions are part of human nonverbal communication, and humans use expressions to send and receive messages.

2.1.1. Signals Conveyed with Facial Expressions

Facial expressions convey many signals in human non-verbal communication. Studies mention that 55% of a message in face-to-face conversations between humans is conveyed through facial expressions [14]. The most commonly studied signal from facial expressions is emotional states [15], but other signals can be studied as well, including conversational cues, physiological states, and non-emotional mental states [16]. The signals conveyed by facial expressions are important for human-computer interaction interaction because these signals can enable more ways the system can communicate with human users, and these signals also convey information that humans do not always explicitly (or knowlingly) send.

Emotional states can be conveyed through facial expressions. The most commonly studied set of expressions are the prototypical expressions of emotion that Ekman and others studied [6]: happiness, fear, disgust, anger, surprise, and sadness.

The study found humans across different cultures could classify these emotional states from still photographs of actors attempting to portray these emotions. These emotions are sometimes called the "six basic emotions" [9], and expressions portraying these emotions are often used in facial expression recognition and animation work. Another more recent addition to this set of prototypical emotions is contempt [17]. Most social robots that implement facial expressions also employ these emotional displays as preprogrammed expressions [18], [19], [20]. A set of the movements generally associated with these prototypical expressions is shown in Table 1.

Table 1: Movements of Prototypical Expressions of Emotion, summarized from [17]

Expression of Emotion	**Movements**
Happiness ("true")	Lip corners "backwards and up"; Wrinkling near eyes
Happiness (posed or fake)	Same as above without movement around eyes
Fear	Eyes widened; Inner eyebrows "upward and together"; Lips corners backwards
Disgust	Nose wrinkled
Anger	Lips tightened; Eyebrows lowered
Surprise	Eyebrows raised; Eyes widened; Mouth open
Sadness	Lip corners down; Inner eyebrows "together and up"
Contempt	One of the two lip corners up

To express or represent more emotions (or mixed emotions), systems which use multidimensional spaces to represent elements of emotion were created. One such space is the Arousal, Valence, Stance (A, V, S) space used in [21]. Each of the prototypical basic expressions are represented as a point in the space to make less intense versions or combinations of the basic expressions possible. Another three-dimensional emotion space is the Pleasure, Arousal, Dominance (PAD) model [22], although it is concerned with temperaments (an average of emotions over longer periods of time). The PAD model also places certain prototypical expressions as points

in the 3D space. These emotion and temperament models are not used as often for facial expression animation or face robots however, due to a lack of mappings from coordinates in these emotional spaces directly to facial expressions. These models are more often used for evaluating human emotional states from more modalities than facial expressions alone. In the PAD model, facial expressions are good for displaying where one lies on the Pleasure dimension, but not the others [23]. High pleasure can be displayed with a smile expression while low pleasure can be displayed with a "sneering" expression or "anger" expression which displays the teeth. The arousal and dominance states are best detected or displayed using pose and body gestures, such as a loose, relaxed torso and upper body to denote low dominance or low arousal.

In addition to emotion, facial expressions can communicate other messages such as non-verbal cues, physiological signals, and mental states not typically associated with emotions. The study in [24] found that humans use facial movements or expressions in conversation as they speak. These expressions were grouped into three categories:

- Semantic: Use of facial movements to convey the meaning of a spoken word (e.g. re-enacting facial expressions of a person while talking about them)
- Syntactic: To emphasize parts of spoken dialogue, similar to italics in text (e.g. eyebrow movements when speaking a word of importance in the conversation)
- Listener response: The listener using facial expressions to respond to the speaker in conversation

Listener responses in particular were studied in HCI. For instance, a "confused" facial expression or shaking of the head can be used in place of some verbal error message that a command or statement was not understood [2], [25]. Conversely, a nod (although more of a head movement than a facial expression) can signify that the last query was understood. These signals from facial expressions can be used to replace or complement error messages for more efficient and pleasing communication in HCI. Facial expressions can also be used to convey physiological signals or mental states [16], [26]. These expressions, including the expressions for pain, tiredness, or attentiveness can be used in robots and virtual characters to enhance their humanlike behaviours. Robots could use a facial expression for instance, to let the user know of

mechanical troubles or a low battery. Virtual avatars and robots alike could use facial expressions of attentiveness (or lack thereof) to show users where to focus their attention [2]. Virtual characters in computer games and other virtual environments use facial expressions for many aspects of interaction with the player [8]. Authors of these virtual environments express virtual characters' emotions through facial expressions to increase immersion and trigger emotional responses of human users towards the virtual characters. Similarly, virtual environments with expressive characters can engage users in a form of social learning that uses facial expressions. The user can read facial expressions of virtual characters to receive hints as to whether they are solving problems in the environment correctly. For instance, an expression of frustration in characters the user is working with (or joy in those the user is working against), notify a user that they may want to try a different approach, without using words. These "non-emotional" facial expressions are equally important in HCI, and should therefore be considered in communication with robots.

For the purposes of animating facial expressions, most works use spaces which are tied to the motion of the face rather than some psychological state. These other spaces are described in section 2.1. Facial expressions can be used to convey emotional states, and in animation and robots research, the main focus is still the display of prototypical expressions. However, facial expressions can have many more uses. Because facial expressions have so many uses and are context dependent, even different between individuals, any adaptive system such as a service robot or affective HCI system should implement some way to incorporate new expressions.

2.2. Classes of Face Models

A face model with animations must be used to generate, display, or manipulate facial expressions in a computing system. Face models can be represented by a 3D mesh if the face is created on a computer, or some statistical (appearance) model if the shape of the face is obtained by scanning humans [27]. The proposed system uses parameterized face models, in which a facial expression is represented by a vector. Unlike generic face models such as a mesh or image that can be deformed in any conceivable way, parameterized face models constrain their movements to the space

defined by their parameters. Depending on the type of parameterized model, the parameters can have different meanings. Parameters can represent deformations directly, or they can represent some forces or simulation of muscles. Robot faces can also be modeled for purposes of animation, with the parameters of the robot being its actuator positions. This section covers different types of parameterized face models used in facial expression recognition or animation.

2.2.1. Facial Action Coding System (FACS)

The Facial Action Coding System, or FACS, was developed by Paul Ekman as a method to measure and quantify facial expressions in humans [7]. Its parameters are called action units (AUs), and each AU roughly corresponds to a single indivisible movement that a human can perform by moving facial muscles. Each FACS AU is referred to by number, but for each AU there is also a short description and detailed instructions for a human to perform the action. A list of some AUs with their number and descriptor is shown in Table 2.

Table 2: FACS AUs, copied from [7]

#	Descriptor
1	Inner Brow Raiser
2	Outer Brow Raiser
4	Brow Lowerer
5	Upper Lid Raiser
6	Cheek Raiser
7	Lid Tightener
9	Nose Wrinkler
10	Upper Lip Raiser
12	Lip Corner Puller
13	Cheek Puffer
15	Lip Corner Depressor
16	Lower Lip Depressor
17	Chin Raiser

18	Lip Puckerer
20	Lip Stretcher
23	Lip Tightener
24	Lip Pressor
25	Lips Part
26	Jaw Drop

FACS was initially intended to be used by human researchers to objectively record the facial expression of other humans, but concepts of FACS (typically the AUs) have been used in automated facial expression recognition and animation. Systems have been developed to automatically estimate FACS AUs from images or video sequences [28], and public databases of humans performing FACS AUs are also available [29]. For animation, several facial animation standards and systems have loosely based their parameters on FACS, including the MPEG4-FA standard [30], the Candide3 face [31], and the Source Engine [32] by Valve Corporation. The first 11 animation parameters in the Candide3 face are based on FACS AUs, and share the same name as FACS AUs: These parameters include Upper Lip Raiser, Jaw Drop, Lip Stretcher, and Brow Lowerer. Animation and recognition systems use subsets of FACS AUs, or they imitate the movements of FACS AUs. The Source engine uses shape-based faces (explained in Section 2.2.5) that have animations with similar names and effects to the FACS AUs. The way AUs identified movements of the face rather than the expressions that are a result of those movements made FACS attractive to artists and researchers wishing to animate a face.

2.2.2. Active Shape Model and Active Appearance Model

The active shape model (ASM) active appearance model (AAM) are parameterized representations of 2D images that are commonly used for facial expressions. Unlike the other parameterized models described in this section, parameters are obtained by measuring differences between images in a training set, rather than by the authors restricting the model to have a certain set of deformations.

An ASM characterizes 2D images by storing some mean shape and variations on this mean shape from a training set of labelled images [33]. These models, although not specific for faces, have been used in face detection and recognition systems [34], either alone or in combination with the AAM. The labelling of the images in a training set is done by selecting a set of points in each image of the training set. In [33], the points were selected using the following criteria:

1. "Application-significant" points, used to mark important features that are present in that particular class of image
2. "Application-independent" points, marking general areas of the image to track, including the boundaries of a region in the image
3. Other points "interpolated" from the other criteria

As an example of this labelling to model a face, the work in [35] labelled points along the following regions:

- The eyebrows
- The boundary of the face region in the image from the eyebrows down to the bottom of the chin
- The boundary between the eyelids and the eye region
- The nose bridge and bottom of the nose
- The outer boundary of the upper and lower lip

In addition to being of significance in the image, the selected points in the training set must be consistent between training images. The order of the points matters, as well as what features they mark in the image. If care is not taken to have consistent labelling (such as labelling the centre of the nose bridge in one image and using the bottom of the nose bridge in another), the training of the model will fail and produce anomalies in which features in certain images are distorted to different locations.

The points in each training image are used to create a point distribution model of the face [34]. In this model, each shape, whether it be a whole face or deformations to add to a face, is represented as a vector $\boldsymbol{s}$ which contains all the 2D vertices on the face in that image: $\boldsymbol{s}=\{x_1,y_1, x_2,y_2, \ldots x_l,y_l\}$. To create a useable face model, all labelled images and their models are processed with principal component analysis (PCA), which is a statistical method that constructs some mean shape and orthonormal shape bases, also

known as eigenfaces, to represent a face. In the ASM algorithm, PCA produces the mean face, called $\boldsymbol{s_0}$, and n eigenfaces, represented as $\boldsymbol{s_i}$, where i is the index for that particular eigenface. The eigenfaces are applied to the mean face to create deformations by multiplying them by parameters, p_i. Any face (deformed or neutral) that fits the model constructed from the training images can be represented in the following form:

$$\boldsymbol{s} = \boldsymbol{s_0} + \sum_{i=1}^{n} p_i \boldsymbol{s_i} \quad \textbf{(1)}$$

From [34]

Since the ASM representation in (1) was generated with statistical methods that do not observe the context of facial expressions, $\boldsymbol{s_0}$ is not necessarily a neutral face representation. Note that each training image has to contain the same number of points, and in the same order because the model relies on deforming the same mean shape. The shape bases can also be non-intuitive to a human user since they are automatically generated. Additionally, since each shape basis must be derived in some way from the training images, an ASM is not permitted to represent a face deformed in a way not represented in the training images, therefore reducing the possibility for false positives. For these reasons, the ASM is very popular as a representation for a facial expression recognition system to use internally due to reliability, but human animators normally choose other models to deform a face manually, since manipulation of the shape bases is not intuitive for a human.

The AAM is a model that builds upon the ASM by adding a texture component as well as the shape component from the ASM [36]. The texture component monitors changes in either greyscale values or colours on the 2D image of a face. In a similar manner to the ASM, the AAM is trained with many face images that have certain feature points manually labelled. During training, the AAM system applies PCA to look at both the shape variation and the texture to produce mean shape information and mean texture information. The shape information (exactly the same as the shape information from the ASM) is represented by the mean shape $\boldsymbol{x_0}$ and n eigenshapes, $\boldsymbol{x_i}$. Similarly, the mean texture and texture bases are represented by $\boldsymbol{g_0}$ and $\boldsymbol{g_i}$, respectively. The shape and texture information for any face that fits the AAM are represented as follows:

$$\boldsymbol{x} = \boldsymbol{x_0} + \sum_{i=1}^{n} p_i \boldsymbol{x_i}$$
$$\boldsymbol{g} = \boldsymbol{g_0} + \sum_{i=1}^{n} p_i \boldsymbol{g_i} \quad \textbf{(2)}$$

The equations in (2) represent the original implementation of the AAM from [36]. Note that there is only one set of parameters (p_i) per face, despite information of the face coming from shape and texture. From this implementation of AAM, there have been later works to refine or add to the AAM to add flexibility to the model or contain 3D information [34] [37]. These further works include the independent AAM, the 2D+3D AAM, and the 3D anthropomorphic muscle-based (AMB) AAM. The independent AAM was introduced as a method to have two different sets of parameters control the information for the deformed model of a face [34]. The original AAM is considered a "combined" AAM, meaning that a single set of parameters (p_i) set the shape and the texture information in a deformed model. With an independent AAM, a set of appearance parameters control the texture component, and a set of shape parameters control the shape component. The 2D + 3D AAM adds limitations on the 3D reconstruction of the face to eliminate cases where a deformed face may be a valid combination of the shape and texture bases, but it creates an unnatural face shape [34]. These 3D shape constraints are represented by a set of 3D shape bases and a mean 3D shape, using a similar form as equation (1). The 3D shape has the same number of vertices as there are feature points in the training images used to create the AAM. To fit the 3D shape back to the 2D image, the 3D shape is projected back into 2D, which normally requires the 3D shape to be rotated or translated to fit the 2D image.

ASMs and AAMs are used in most FEC implementations that use 2D images or video of humans as the source of the facial expressions. AAMs have also been used as the target face for FEC implementations [38], and these implementations have been researched to be able to replace faces of humans in video while maintaining the expressions.

2.2.3. Physically-based Face Models

Physically-based face models simulate the face as some sort of deformable model, with simulated muscles exerting forces on the model. In such a model, the parameters denote the contraction or the force of muscles on different points of the face. In the implementations of [39] and [40], the face itself is modeled as a rigid skeleton (bone) and a mass spring system. To simulate multiple types of layers in the face, the density of the vertices and the stiffness of the springs is different for each layer, with the top layer (epidermis) being the stiffest of the layers. The muscles in this type of physically-based face model exert forces to pull vertices of different layers together. In addition to the muscle forces, constraints for volume preservation were introduced to mimic the way facial tissues contain incompressible water. In [39], the displacements were calculated by integrating the accelerations resulting from the forces over time, meaning that a final deformation (for a given set of muscle forces) is not reached until the system stabilizes. The method to generate a simple deformation in a physically-based face is illustrated in Figure 1.

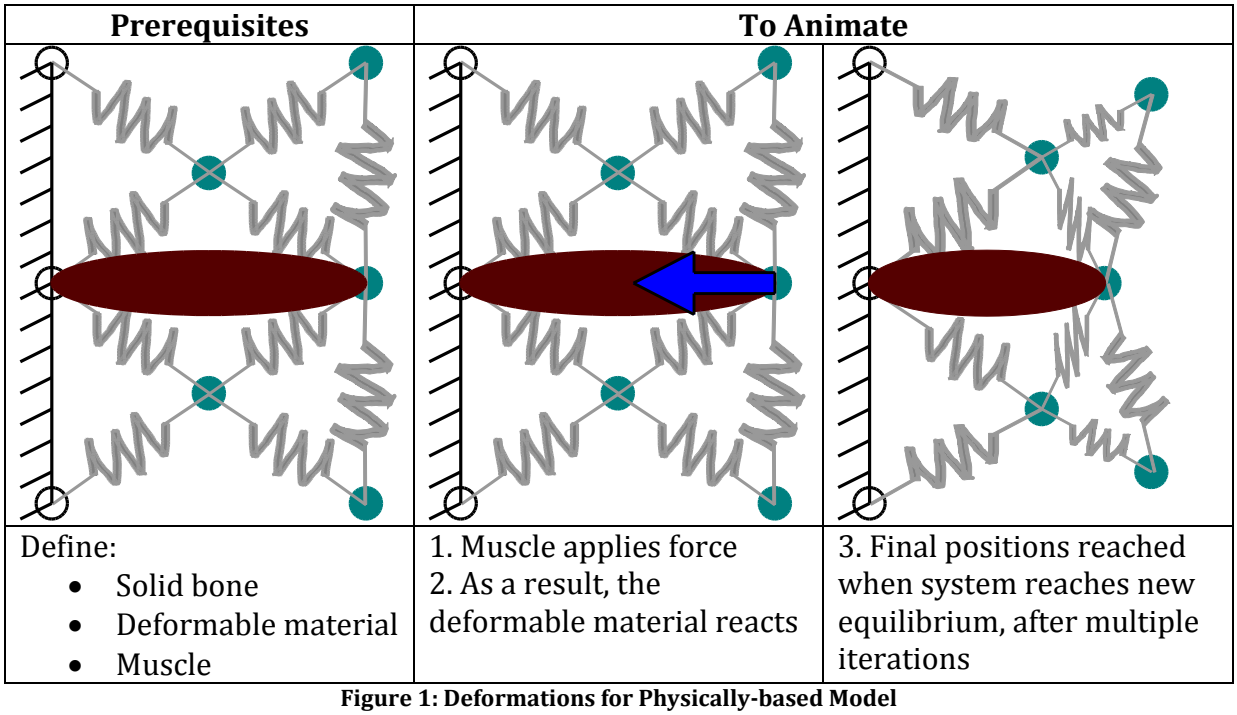


Figure 1: Deformations for Physically-based Model

An alternative physically-based face model was proposed in [34], but for the simulation of a robot face rather than a human face. In the model of [34], the face was modeled as a set of deformable tetrahedral structures. The deformable structures used

an elastic material model, and they were connected to a rigid structure representing the non-deformable parts of the skull. Muscles applied forces which added energy to these structures, and a final deformation was calculated by finding the positions for each of the structures that minimizes the total elastic energy stored in the face. Similar to the mass-spring representations, this elastic model requires multiple iterations to reach the final deformed state. The detailed nature of these models allows them to automatically simulate wrinkles in the skin for facial expressions or aging simulation [35]. Physically-based face models offer a realistic simulation when it is desired to simulate the underlying dynamics of the face, such as forces internal to the face, but they suffer from higher computational complexity compared to other models. As such, simpler models have been developed that maintain a believable output while being able to calculate deformations for facial expressions in a single iteration.

2.2.4. Muscle-based Face Models

Muscle-based face models (or pseudo-muscle based face models) are a simplification on the physically based model. They retain the concept of simulated muscles as in [39], but they do not handle dynamics of the face. Additionally, the face is modeled as a 3D mesh with vertices. To do this, the muscles in this model are represented as objects with an attachment point, an unmovable vertex that acts as a point of reference for the muscle, and a region of influence (ROI), a set of movable vertices in the face model that the muscle can displace [37]. When activated, the muscles move the vertices in the ROI towards the attachment point, or in some trajectory around the attachment point, depending on the type of muscle. To calculate a deformation, each muscle contributes a displacement to each vertex in its ROI, and all of these displacements are summed together to get the final position of a single vertex. This calculation can be performed in one iteration per muscle, plus the sum of all the deformations. Parameters are used to state how much of the full displacement each muscle contributes. For instance a parameter of 0 would make the muscle have no influence (neutral), and a parameter of +1 would add the full displacement of the muscle to each of its vertices. When several muscles act upon the same set of vertices, the neutral positions of the vertices are used to calculate the displacements. Otherwise,

there would be a race condition, where the result would depend on the order of which vertex deformations are calculated. The muscle-based model provides a nonlinear relationship between parameters and deformations, while still being simple enough to process in real-time. For these reasons, this project uses the muscle-based model in [37] as one of the target faces for mapping, to be able to quickly validate the algorithm without using a physical robot.

The muscle-based model defines three different types of muscle [37]: linear muscle, sphincter muscle, and sheet muscle. The linear muscle simulates a muscle that is attached to a bone [39]. For the simulation, the muscle pulls vertices towards an attachment point, which represents the point where the muscle attaches to the bone. The pivot point does not move in the simulation. Similarly, a sphincter muscle is a muscle that pulls towards a deformable location. In the muscle-based model, sphincter muscles pull surrounding vertices towards another vertex. Since the muscle-based model only calculates against neutral vertex positions, the linear and sphincter muscles are simulated using the same equations, but with a different attachment point. The sheet muscle represents a region where muscle fibres pull along some surface, rather than towards some point. The deformations that occur in a muscle-based model for linear or sphincter muscles is shown in Figure 2.

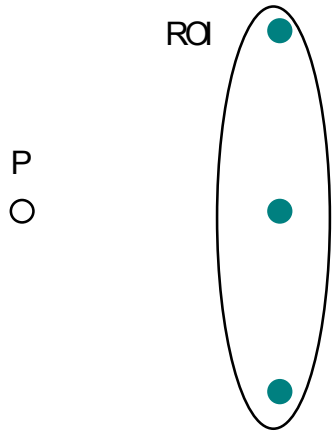
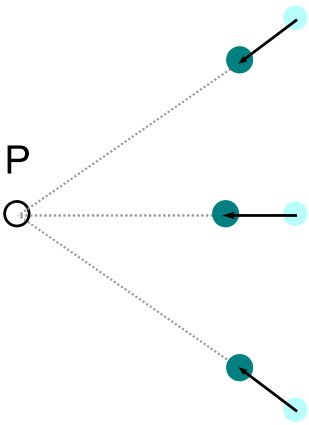
Prerequisites	To Animate
	
Define: • Deformer attached to some point P • ROI for deformer	Deformer makes distance between P and vertices within the ROI smaller

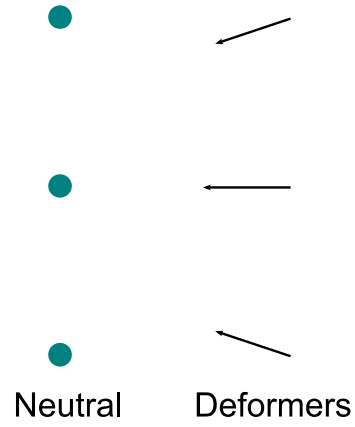
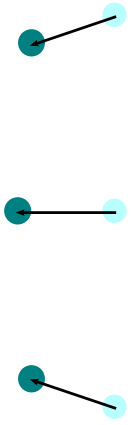
Figure 2: Deformations for Muscle-based Model

2.2.5. Shape-based Face Models

Shape-based models are the simplest and most popular face model for animation [41]. These models are also known by the name of "shape interpolation" or the name of the animation technique in certain software packages, such as "morph targets" or "blend shapes". Unlike the physically-based and muscle-based models, shape-based models do not attempt to follow any physical laws. Instead, they store a neutral face mesh, $\boldsymbol{x_0}$ and N deformers, $\boldsymbol{d_i}$, $i \, \varepsilon \, [1, N]$. The parameters p_i, which take a value in [-1, 1] for most implementations, state how intensely the deformations are added. To create a deformed version of the face, deformers are added to the neutral face according to the parameters as follows:

$$\boldsymbol{x_D} = \boldsymbol{x_0} + \sum_{i=1}^{N} p_i \boldsymbol{d_i} \tag{3}$$

The representation in (3) is from the Candide3 face, which is used to test the proposed system [31]. A parameter with a value of 0 does not apply a deformation, and a face with all parameters set to 0 is the neutral face. Similar to the ASM model, deformations do not change the number of vertices or triangles in the mesh. This process is illustrated in Figure 3.

Prerequisites	**To Animate**
	
Define: • Neutral positions	Add deformed positions to neutral positions

• Deformed positions or displacements	

Figure 3: Deformations for Shape-based Model

Another standardized model that is compatible with shape-based faces in the MPEG-4 FA standard. MPEG-4 FA is a standard which defines the shape of the face, animations affecting the face, and the encoding of a stream of facial animation data [42]. The MPEG-4 FA standard specifies the static model of a face using 84 mandatory points on a face in its neutral position, referred to as feature points (FPs) [43]. FPs declare which areas of the (static) face a point is associated with, and many of the FPs are displaced during an animation. The dimensions and scaling of the face are defined using facial animation parameter units (FAPUs), which state the distance between select facial features. The movements available to facial animations are defined by face animation parameters (FAPs), which are integer values that define for different movements, which FPs move and by how much do they move. Similar to shape-based methods, MPEG-4 FA can define linear combinations of transformations to vertices in a face model using FATs, and the animations created by the FATs have no relationship to the physics of the face model. MPEG4-FA was useful for some cases of FEC because any model that is compatible with the standard could be used as the target face.

2.2.6. Face Robots

Face robots are mechanical structures that use lighting or actuators to emulate facial expressions. The lighting can be used to project parts of the face onto translucent surfaces of the robot or emphasize expressions by making the robot glow different colours associated to emotions [44]. If expressions are produced exclusively by using projected lighting, then such a robot can use one of the aforementioned face models directly, such as a rendering of the shape-based model. The other method to produce facial expressions on a robot is with actuators. Actuators, including pneumatic cylinders or DC motors, push and pull parts of the robot's face to emulate a facial expression [18]. Depending on the construction of the robot, the actuators may move rigid structures such as plates shaped like eyebrows and a mouth, or they may move deformable parts including rubber lips. Robots that use deformable parts can produce humanlike facial

expressions, but simulating these movements is nontrivial, requiring some physically-based model [45]. If no simulation is available, then facial expressions must be captured from the robot using some sensors or cameras. When using a real robot to train a FEC system, the following considerations must be taken into account, which are not typically assumed for FEC to virtual models:

- Hardware limits as to how much the face can move
- Limited speed of the actuators
- Sensor noise while reading back displacements

With these considerations, FEC systems that use robot targets must operate differently than those that use virtual targets. Since movement of the robot face is limited in both position and velocity, training cannot experiment with many thousands of iterations of the robot face, and software must be in place to prevent the robot's actuators from hitting mechanical limits (or else damage or inaccurate readings would result). FEC systems that handle robots must also deal with imperfect sensors and actuators, similar to capturing facial expressions from humans. Figure 4 and Figure 5 show robots with rigid and deformable structures to perform expressions.

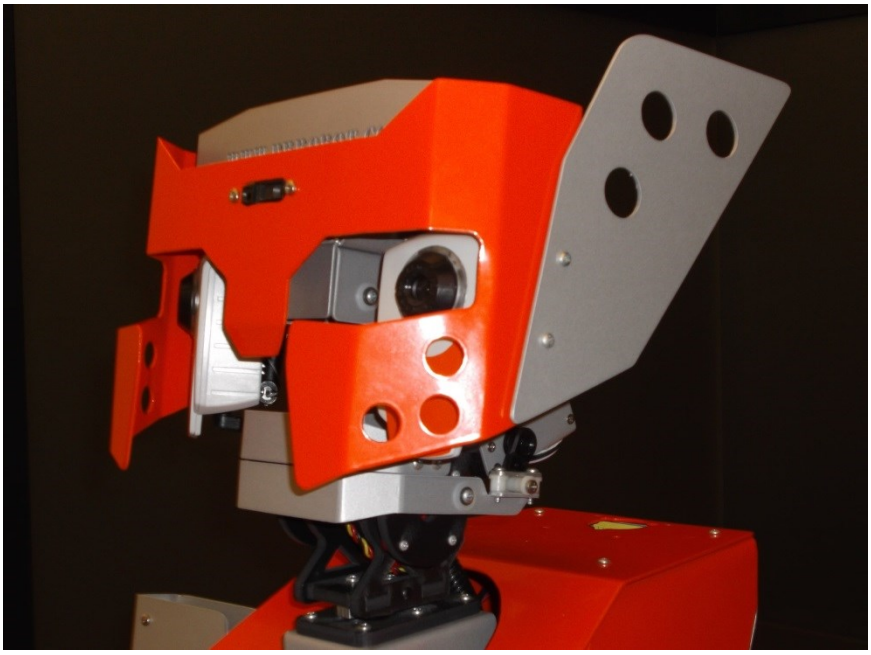


Figure 4: Robot with Rigid Face Structures

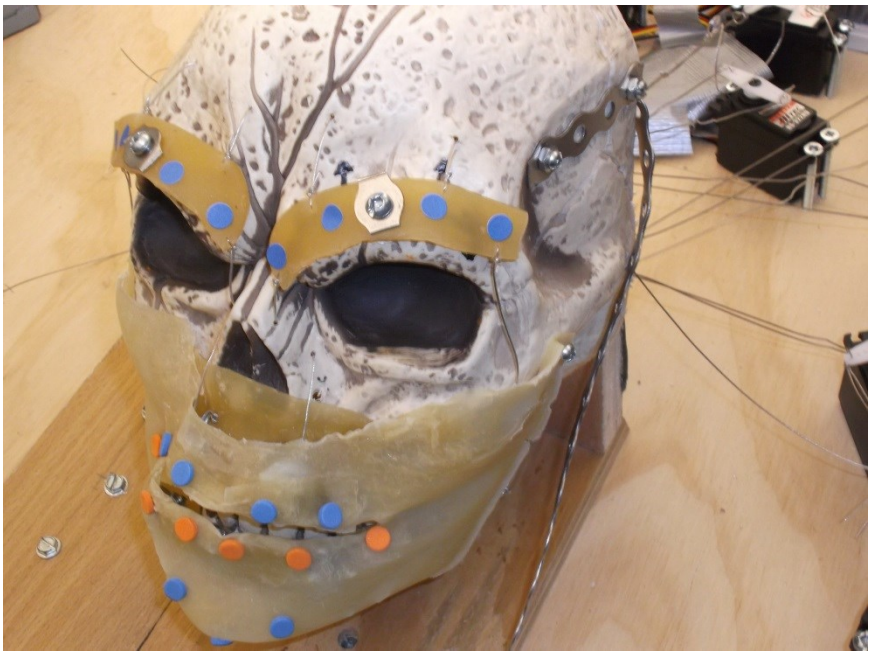


Figure 5: Robot with Deformable Face Structures

2.3. Fuzzy Membership Functions and Fuzzy Sets

In the proposed system, fuzzy sets are used as an intermediate space to compare the two faces regardless of their underlying implementation or construction. Fuzzy sets are used to represent the location of vertices and which movements they can perform. They are also used as values to compare source and target facial expressions. This section describes fuzzy membership functions, fuzzy sets, and the role of these concepts in systems in literature.

2.3.1. Overview of Fuzzy Membership Functions and Fuzzy Sets

The concept of fuzzy sets is from the field of fuzzy logic, which is a system that can assign infinitely many values of truth or falsehood to some variable [46]. For purposes of explanation, the variable will be called *a*. If *a=0*, then whatever *a* represents is absolutely false. If *a=1*, then *x* is unquestionably true. Any value in (0, 1), e.g. *a=0.5* or $a=1\times10^{-20}$ denotes a partial truth. This "partial" truth is also what gave fuzzy sets and fuzzy logic the name "fuzzy". This is compared to a two-valued or "classical" logic in which a variable can only take 0 or 1. Likewise, a fuzzy set is a grouping in which elements are assigned a value between [0,1] that states how much an element belongs to that set [46]. If an element's value in the set is 0, then it absolutely does not belong to

the set. Likewise, a value of 1 denotes that the element fully belongs in that set. Intermediate values denote a partial belonging to the set. In a fuzzy set, the "belongingness" of any element *x* is assigned by use of a membership function $\mu(x)$, which assigns a value between 0 and 1 to any *x* inputted into the system. A membership function can have any shape as long as it assigns the input to a single value in [0,1]. This type of membership function is also referred to more as a type-1 fuzzy membership function because each *x* is assigned to a single membership value. Type-2 fuzzy membership functions and sets also exist, where the fuzzy membership function assigns a fuzzy range of values to an inputted *x*, rather than a single value [47]. The use of a range of values for type-2 fuzzy logic is meant to represent the notion that the problem has uncertainty in the definition of truth or belongingness to a set. Although such a concept could theoretically be applied to this project, it exclusively uses type-1 fuzzy membership functions for practical reasons. The use of type-1 fuzzy logic allows for repeatable results in the mappings given a set of input parameters. In addition, the uncertainty factor for the representation of the faces is handled by weights that are varied by a particle swarm, described in section 3.9. Therefore, this thesis uses fuzzy logic terms without specifying the type, since there is no type-2 fuzzy logic implemented for this project in any component.

Fuzzy logic and fuzzy sets are more often used in facial expression recognition than in FEC, since the implementations mainly use fuzzy sets to represent high-level ideas such as emotions or prototypical expressions. Halder et. al. used type-2 fuzzy sets that represented the measurements of different regions of the face in a system to classify the six basic emotions from facial expressions [48]. The fuzzy set values obtained from the measurements are then used as the input to higher-level membership functions to classify facial expressions. Zhao et. al. used type-1 fuzzy membership functions and fuzzy logic to classify emotional states from several modalities including facial expressions [49]. The implementation used fuzzy sets to represent the following high-level prototypical facial expressions: happy, angry, and sad. The work in [10] used fuzzy sets to characterize low level motions of faces similar to the proposed system, which is explained further in section 2.4.4.

2.4. Facial Expression Cloning

Facial expression cloning (FEC) is the process of taking existing animations and mapping those animations to another model which is not identical to the first model. The term was first mentioned in [50]. A special case of FEC is performance-based facial animation, where the source model is taken directly from recordings of a human's facial expressions through video or other motion capture technology [51]. In performance-based facial animation, the system takes a human's recording (performance) of facial expressions and uses algorithms to map these facial expressions to a computer-generated model, such as a 3D mesh or 2D avatar face.

FEC has several applications, ranging from animation to privacy. Since it allows transferring the animation between models, its primary use is to save animators time by allowing them to create one set of facial animations and cloning the animations to many different models. This is in contrast to creating specific animations for each and every character in an animated work. Certain implementations do not require pre-programmed deformations [52], which save animators from having to define moveable parts and motions on the face (known as rigging). FEC can also be used to map the facial expressions of a person who does not want their face revealed. In online conversations, FEC would allow humans to have a video-based conversation with their face replaced by an avatar face. Additionally, camera data online, such as Google Street View could replace faces rather than blur them out for censoring [53]. The typical components and data flow of the discussed implementations is shown in Figure 6. A FEC implementation starts with a model of a face (source model) that has static deformation or animation information attached. In the case of performance-based facial animation implementations, the FEC component of the system uses a model gathered from video or motion capture data. The input model (and its animations) is then processed through a fitting or training stage, in which a mapping algorithm is created. This mapping algorithm is used to take animations between the source and target models. The output of the system is the target model, which is a model which is not identical to the source model (although the models in many cases are similar). This subsection outlines recent FEC systems.

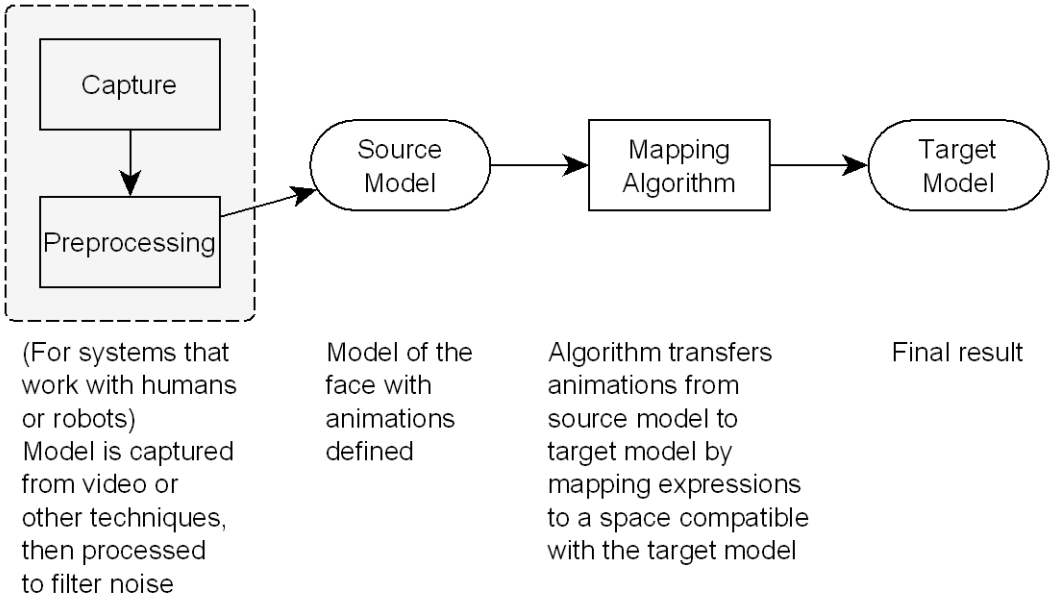


Figure 6: Facial Expression Cloning Process

2.4.1. Video and Images of Humans to Virtual Models

The first FEC systems captured expressions from pictures or video of humans rather than animated 3D models. Almost all of the implementations used standard 2D video from a single colour camera, but there are solutions that used depth information from an infrared camera and projector as well as a colour camera [54]. These cloning methods typically required some training process to generate or adapt parameters to the video to map to virtual face models. After training, a mapping algorithm used the trained images to clone facial expressions to the target face. The cloning methods that used 2D image or video data as a source predominantly used AAM in their capture and preprocessing stage.

The implementation of Cordea in [37] used video captured from a colour camera and mapped the pose, face shape, and facial expressions of the captured face to a muscle-based 3D model. It measured face shape and pose using an extended Kalman filter. The filter used predictions from the muscle-based model and motion data from the camera data as the streams of information. The pose information was gathered by tracking points which were not actuated by the 3D model. Facial expressions were tracked and transferred to a muscle-based model using an AAM search, similar to the

2D+3D AAM. In this implementation, the muscle-based model, which was constructed separately from the training of the system, was used as the constraint for the shape of the face in 3D. This is opposed to the 2D+3D AAM in [55] where the 3D constraints were created during training. The AAM search directly mapped from the captured video to the muscle-based model. The 3D shape was constrained by the range of contraction of the muscles in the model. After training, which required marking 64 points on the face, was complete, the mapping process was completely automated.

Stoiber, Seguier, and Breton [38] created a system to learn and clone facial expressions from a database of videos performing facial expressions to a generic 2D or 3D model. The videos of humans were processed using the AAM fitting algorithm, and the parameter space generated for the AAM was used for the cloning of expressions. This method used the independent AAM to be able to clone only shape parameters if desired (e.g. for a 3D face where shading performed on the target) or to clone the shape and texture when cloning to a 2D model. A deformation space is also created for the target using principal component analysis. Mapping between the AAM parameter space and the target space is performed using weighted sum, where weights were determined using linear regression. This method required a collection of source expressions and target expressions to function.

Two studies cloned facial expressions from images of humans to MPEG-4 FA compatible faces. Jia, Zhang, and Cai [56] cloned expressions from a 2D image database of facial expressions to 2D and 3D generic models. The system divided each image into the following sub-regions:

- Eyeball
- Eyelid
- Upper lip
- Bottom lip
- Corner lip

These sub-regions were chosen to match the animation parameters (FAPs) of the MPEG-4 FA model. The movement of individual vertices on the face were clustered with hierarchical clustering. The cluster that had movement with the greatest

correlation to the movement of the FAPs were matched together to determine the FAP parameter for each source face. A cloning was then performed using these FAPs to copy the animation to 2D and 3D face models compatible with the FAPs. Lei and Miao [57] performed a similar mapping from video data to FAPs, using a trained AAM to track 68 points on the source face. The mapping from displacement of the points to FAPs was also performed with a weighted linear transformation. The system aligned the movement of the face with the movement specified in each FAP to find out which FAPs to use, and the intensity of the FAP was determined by dividing the magnitude of the displacement in the source movement by a measurement of the source face. The measurements included the width and the height of the head in the 2D image. The mapping from the movement of points to FAPs in both studies was automated, but the study in [56] used a database where the points were already labelled (instead of recovering the points from the image data).

Kholgade, Matthews, and Sheikh [58] cloned expressions from video data to shape-based 3D models. This study attempted to classify high level expressions, rather than low-level movements in their model. Motions for the mapping were separated into three layers: emotion, speech, and eye blinks. The method was used to give artists more control over the reproduced expression when cloning to cartoon faces: depending on what the character was to portray, the expression could be biased towards the desired components of the expression. The emotion layer could control most of the face, and it was separated into an emotion space which could represent 29 different intense emotions (and the blending of these emotions). The speech layer was segmented into a 12-dimensional space representing 12 visemes (the shape of a face when speaking specific syllables). Finally, the eye blink layer represented fully open, partially open, and closed eyes. Points on the human performer's face were captured using 2D+3D AAM fitting. During capture, vertices were selected to be contributing to certain layers using weights, and a different set of weights were applied when cloning the expressions to the source. Manual steps were required for the creation of all the weights.

Kim et. al. proposed a system to clone expressions from a human performance in video to the Candide3 shape-based face [59], [31]. Similar to the work in [37], separate

mechanisms were used to track full head movements and expressions. In [59], the head pose was tracked using a sequential Monte Carlo algorithm, while facial expressions were tracked using 2D+3D AAM fitting. The system tracked 56 control points directly, and it interpolated an additional 13 points using an internal muscle-based face implementation.

2.4.2. 3D Scans of Humans to Virtual Models

Another approach to take expressions directly from humans is to use a 3D scan of the human performance. These methods measure displacement of points on the scanned data and map these displacements to the target face.

Sucontphunt et. al. [60] cloned expressions from a 3D scan of a human face to a 2D image styled similarly to a portrait sketch. Live performance by a human was captured using a 3D scan with markers on the human's face, which tracked 90 points. As a preprocessing step, the image was centred on a specific point and singular value decomposition was used to remove any undesired rotation. Subsets of these points were mapped to both an intermediate 3D model and a 2D image. These points were transferred to the model by "motion propagation", a linear transformation of the velocity of points from the source capture to the target images. The system then generated "strokes" corresponding to the location of the detected points on the 3D model and the edges of the 2D image. These strokes combined to make the target face model look like a hand-drawn sketch.

Weise et. al [61] performed FEC from a live 3D scan of a human to virtual models. It fitted a template 3D scan of the human, and it transferred the deformations of the template to the target model. The source data from the 3D scan came in as a point cloud. To adapt the template to a particular human subject, points were manually selected in the point cloud to correspond with points on the template. The template was fitted to the human's 3D scan by minimizing distances from points on the template to points on the point cloud. The human subject also performed prototypical expressions for the system to be able to record which template vertices move during expressions and the displacements of these vertices. The system then performed principal

component analysis (PCA) on the displacements for each expression performed by the human to create a model describing the motion of the face. This model was used for the real-time mapping of the human's facial expressions to the target 3D model.

Lin et. al [62] proposed a system to perform FEC between 3D scans of humans performing prototypical expressions. The authors used sparse coding, a technique to generate basis functions that can handle spaces with higher order relationships. The sparse coding was used to define a dictionary for each expression across all faces, and a coefficient matrix specific to each face. Using this method, FEC was performed by a dot-product of the entry in the dictionary for the expression to be cloned and the coefficient matrix for the target face. The system was shown to also perform with incomplete or noisy data sets, which is expected with scanned data. To create the dictionary, each scanned expression had to be labelled.

The Kinect for Windows SDK as of version 1.5 [63] was able to use 2D video plus depth data (from an infrared camera and projector) of human performance to copy to a shape-based face in real time. This commercially-available system uses the proprietary Microsoft Kinect sensor and algorithms to operate. The target face is a shape-based face based on Candide3. This solution is available out-of-the-box for users with demos available.

2.4.3. Replacing Pictures of Humans in Video

Certain implementations cloned expressions from video or images to another 2D image or video of a human. Some of these implementations map back to a 2D image despite making use of 3D data (e.g. a form of 2D+3D AAM) in their intermediate stages. The applications of these systems were for security and privacy, to be able to hide the identity of a human in a picture without removing their facial expressions.

De la Hunty, Asthana, and Goecke transferred expressions between images of humans by manipulating their AAMs directly, with no other intermediate models [53]. It selected AAM parameters on both faces to minimize the following:

- Change in the target face's AAM parameters compared to neutral vs .change in source face's AAM parameters compared to neutral

- How far candidate expression is from expressions the target face is capable of

In addition to this minimization, the system also accounted for full head movements using a similarity transform. It compared points which did not move much during expressions to find the appropriate transformation to the source shape and texture to apply to the target when performing the cloning.

Huang and Torre [64] cloned expressions from video to another face image using bilinear regression. Similar to the AAM-based methods, their system mapped the changes in shape and the changes in texture (appearance) separately. The system required the source face to have expressions, and the target face was the 2D image of a neutral face. To transfer the shape, it tracked 66 points from the source face to produce 106 triangles, and the deformations of these triangles from neutral were transferred to the target face. Next, under the assumption that the shape and appearance changes were similar, the system transferred changes in intensity for individual pixels depending on the expression.

2.4.4. Between Virtual Models

This subsection describes systems in literature that are capable of cloning from face models created by animators (and not automatically from capture). Systems that also map from video capture are listed here as well, if their intermediate model uses parameters generated by humans.

Khanam and Mufti [10] implemented a system that mapped faces between MPEG4-FA compatible models. The first stage of that system defined how points on the source face moved using two fuzzy sets. One set described the movement in the X direction and the other set described movement in the Y direction. These sets took the movement in each direction, and it classified them using the following overlapping triangular functions:

- Positive low, Positive medium, Positive high
- Negative low, Negative medium, Negative high

In this fuzzy set notation, a movement that was closer to zero received a higher membership value in “low”, while a value closer to -1 or 1 received a higher value for

the “high”. This implementation of fuzzy sets is similar to that of the proposed system, except for the use of Cartesian rather than polar coordinates. That system also used other fuzzy sets to describe the six basic emotions, which were used to animate mixed facial expressions (by setting values for more than one emotion at once). The cloning was completely automated as the fuzzy membership values were inputted into the subsystem that animated the target face.

Dutreve, Meyer, and Boukaz [65] proposed a system to perform FEC from a video recording or animated 3D faces to a generic 3D face model. Expressions were cloned from the source to the target using a radial basis function (RBF) neural network. This system worked using the concept of having 15 to 30 manually selected points or vertices as feature points, and the other vertices were used to calculate weights in the RBF neural network. The weights for the neural network were set with equations meant to create an inverse relationship between the influence of a vertex and its distance to feature points (if the vertex itself is not a feature point). This type of training avoided having to study target expressions. If the source face was a video, the system used a pyramidal Lucas Kanade algorithm to track feature points. This system required manual selection of 15 or 30 feature points to track on the source and target faces.

The implementation of Song et. al [11] mapped expressions between shape-based models using statistical and machine learning methods. It attempted to map both the shape of the faces and the timing of the expressions together using a combination of two regression methods: a radial basis function neural network (RBF) and kernel canonical correlation analysis (kCCA). The RBF and kCCA results were combined using a weighted sum of the output of both learning mechanisms. Error was measured by mapping from the source to the target and back to the source. Compared to the proposed system, the method in this literature is only semi-automated, requiring a human to generate some trained expression pairs. It also assumed that both the source and target are shape-based.

Seol et. al. [66] proposed a full system to capture expressions from video with markers, but the intermediate part of the system cloned facial expressions between

shape-based models. The source and target face models were matched by manually selecting 54 points on each face to be paired together. The cloning itself was performed by minimizing the difference between the velocities of the points on each face during the animation. Similar to the proposed system, the mapping component of [66] required regions on both faces to be labelled and mapping of facial expressions pairs was completely automated. However, it required the same number of points to be labelled on both faces, while the proposed system only requires regions, which can have different numbers of vertices between models, to be labelled.

2.4.5. Cloning to Robots

There have been fewer implementations of FEC systems that clone to robots, and these techniques either have many manual steps or require a robot face to be very close to the source face. The implementations of FEC to robots were used to allow humans to create pre-programmed expressions for the robot or to allow the robot to mimic a human's facial expressions.

Gotoh et. al. [67] created a facial expression cloning system that mapped from frontal face images to a simple cartoon-like robot, dubbed "Ifbot". The robot only had motors for the eyes and eyelid (two degrees of freedom for each eye, another two for each eyelid). The rest of the robot's facial features were controlled by LED lighting. In particular, the mouth was a projection. The image processing stage extracted 23 points on the face, and the displacement of points from neutral were used to calculate the magnitude of "AUs" loosely based on FACS. The AU values were then used to train a multilayer perceptron (MLP) neural network to map the AU values from the images to control parameters controlling the motors and the LED lighting on the robot. The training inputs to the neural network were created manually.

Jaeckel et. al. [68] proposed a system to clone facial expressions from a video to a humanlike robot face, dubbed "Jules" using the partial least squares method. A video of a human performance was fitted to an AAM, and the AAM shape parameters were used as the input to the learning stage, which outputted appropriate servo positions to the robot face. The robot used 34 servo motors to produce facial expressions. The

learning stage used partial least squares (PLS) analysis. To produce values to train the system, an animator manually mapped the frames in the video recording of the actor to sets of (34) servo motor positions. For new frames not used in the training, the implementation used partial least squares (PLS) analysis on the AAM parameters gathered from the new frame to predict the servo positions.

Tscherepanow et. al. [69] configured a robot, dubbed "iCAT", to mimic human expressions from video. The captured video was preprocessed by performing regression on the change in greyscale pixel intensity values in linear regions. The authors studied three different statistical and machine learning methods to perform mapping from the regression output run on the source video to the robot space: an MLP, a cascade correlation network, and a support vector machine (SVM). The SVM was the least sensitive to the feature extraction method used, but the none of the methods showed significant differences in the accuracy. Similar to the other mappings to a robot, humans manually created the robot expressions in training, and the learning system was only predicted mappings of unknown inputs.

2.4.6. Mapping Accuracy

There is a lack of comparisons or benchmarks to show or compare the mapping accuracy of FEC methods. The study of [37] took error measurements for the angle of the whole head, but the mapping quality for the expressions themselves were only compared by showing the output parameters with a filtering stage versus the same output without the filtering stage. In [53] [65] [56] [38], the authors posted screenshots of their mapping inputs and outputs without any mention of mapping errors. In [66], there was a comparison of the velocity of vertex motions between methods, but the authors were not able to use the same models as the works they were comparing against. In [11], the authors measured the error by mapping expressions to a target and then back to the source, and in [61], error was measured by subtracting the distance from 3D tracking data and the target face. These two measures rely on properties of the algorithm however, since in [11], the system would have to be capable of doing an inverse mapping, and in [61], the error measure of simple distance measurements would have to be scaled in some standardized manner if there was a comparison with a

target that is very dissimilar from the source. The work in [70] used a questionnaire answered by human test subjects to say if the expressions were accurate reproductions, or that the humans could classify the reproduced expressions. The issue with using humans to measure the error is that there could be bias in the human subjects, for instance if the humans have seen a similar model before. The work in [68] used the difference between the output of the system and training data from a human expert. In a similar manner, this type of error measure depends on the opinion of the humans evaluating the facial expressions. There is a need in the field of FEC to have some standard faces (and possibly expressions) to compare the mapping accuracy between works. In this project, error is measured with the same method as in [11] since the proposed system has the same capability to map back from a target to the source face.

2.4.7. Summary of Implementations

The implementations of FEC systems are summarized in Table 3. The table shows the models, mapping technique, and manual steps used in each technique. Note that all of the AAM-based methods specify a trained database as the manual step, since the AAM is built from such a database. If an AAM-based method requires other steps on top of creation of the AAM, then those steps are listed in addition to the requirement of the database. The implementations are listed in the order they were presented in this section.

Table 3: FEC Methods Summary

	Source Model	Target Model	Mapping or Learning	Manual Steps
[37]	Video (2D+3D AAM)	Muscle-based 3D model	Extended Kalman filter, AAM Ssarch	Trained database
[38]	Video (AAM)	Parameterized 3D model	Expression space reduction	Creation of expression pairs
[56]	2D Images	Generic 2D/3D model	Hierarchical clustering	Trained database
[57]	AAM	MPEG4-FA	Linear	Trained database
[58]	Video (2D+3D AAM)	Generic 3D model	Linear	Trained database, Creation of weights
[59]	Video (2D+3D AAM)	Shape-based face (Candide3)	Shape-based face (Candide3)	Trained database
[60]	3D scan	Generic 3D model	Linear	Labelled points (individually)
[61]	3D scan	Generic 3D model	Principal component analysis	Labelled points (individually)
[62]	3D scan	Generic 3D model	Sparse coding	Trained database
[63]	RGB + depth	Shape-based face	Proprietary	None
[53]	Video (AAM)	AAM	Linear, similarity transform (weighted sum)	Trained database
[64]	Video (AAM)	AAM	Linear (transfer deformations)	Trained database
[10]	MPEG 4-FA	MPEG 4-FA	Fuzzy sets	None
[65]	Manually-processed video or 3D model	Generic 3D model	Radial basis function	Labelled points (individually)
[11]	Parameterized 3D model	Parameterized 3D model	Radial basis function, kernel canonical correlation analysis	Creation of expression pairs
[66]	Video (marker tracking)	Parameterized 3D model	Radial basis function (scaling), linear (movement)	Labelled points (individually)
[67]	2D Images	Robot	Multilayer perceptron	Creation of expression pairs
[68]	Video (AAM)	Robot	Partial least squares	Creation of expression pairs
[69]	Video	Robot	Support vector machine	Creation of expression pairs

2.5. Summary

This chapter discussed several implementations of face models and methods that were used in literature to clone facial expressions between models. It reviewed FACS as the system for quantifying all human facial expressions, and how different facial expression representations were based off FACS. Several types of face models were used to represent or generate expressive faces. The ASM and AAM models were

created as models for generic 2D images, but implementations specific to facial expressions used them to be able to automatically create a parameterized face model from training images. In addition, this chapter looked at manually created models of faces: the physically based, muscle-based and shape-based model. The physically-based model was created to most accurately represent the anatomy and dynamics of a human face, by simulating layers of skin using mass-spring or elastic material equations and creating expressions by having simulated muscles apply forces to the model. The muscle-based model was a simplification of the physically-based model, in that it only simulates the *final result* of applying forces to deformable structures in the face, without actually performing calculations on the forces involved. The shape-based model is the most common face model, and it only defines deformations on a neutral face which are added together to animate the face and perform expressions. These virtual face models can be taken into physical space using a face robot. Face robots' expression spaces are defined by their actuators and lights and the control parameters to use these devices. FEC algorithms have worked with humans and virtual models as a source model, and output to other virtual models, 2D images of humans, and robots. Many of the FEC algorithms in literature used the AAM model or variations of it when capturing expressions from humans. A common target face was a shape-based face or generic face model. FEC with robot target faces is less user-friendly than systems that work with virtual models, requiring more manual steps from human animators or the creators of the face model. A large issue in FEC, which is lack of consistency in error measurement, was also discussed. Most implementations of FEC publish images or videos of the final result of the cloning, but they give no numerical data as to how accurate or precise the result is. So far, the FEC implementations that do quantitatively publish error used values specific to their system or face models, making it difficult to compare FEC methods.

Chapter 3. FEC System

The main project in this thesis is a FEC system that generates pairs of mappings from the source face to the target face. This chapter describes the workflow of the system, the face models used in testing, and the individual components of the system.

3.1. System Workflow and Design

The proposed system takes in specifications of the movements and the regions of the source and target faces to generate a mapping function that maps known source expressions to target expressions. A chart showing the steps to generate an output from the system, with two parameterized faces as the starting point, is shown in Figure 7. The user must manually label regions of the face and select a set of expressions to be used for the mapping. All of the other stages are automated. The output of the system is a lookup table of source parameter sets and target parameter sets.

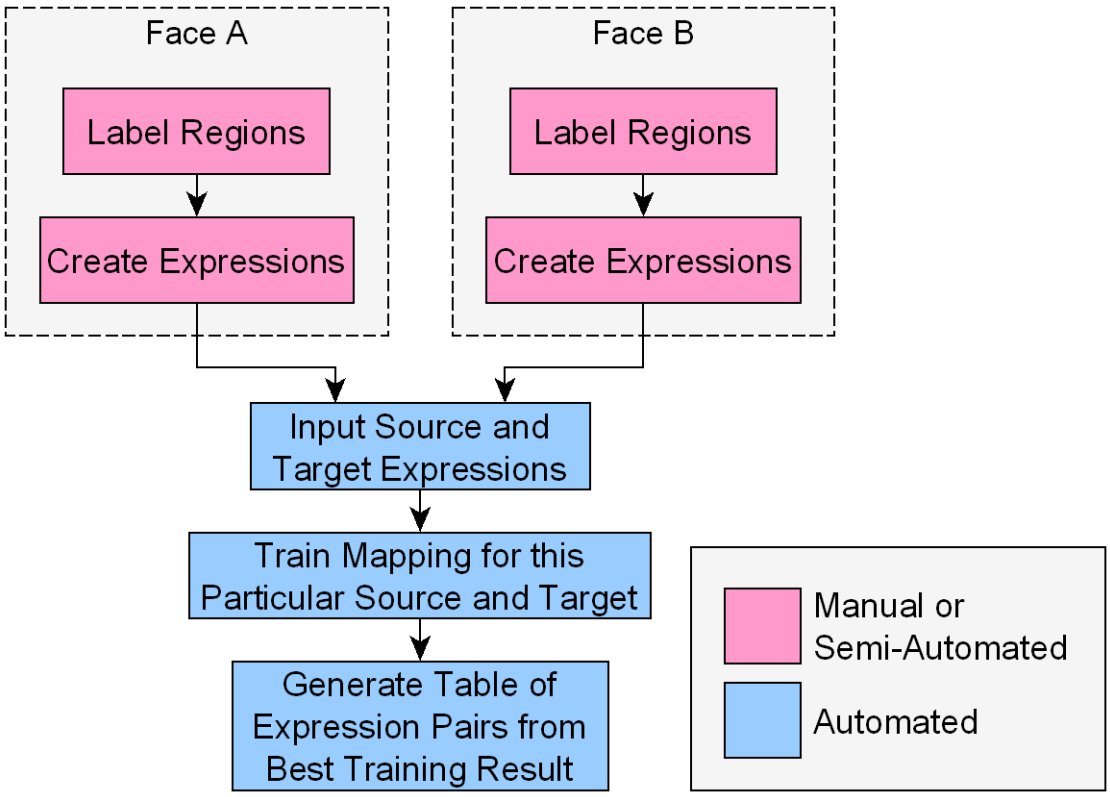


Figure 7: Steps to Create Mappings

The manual steps in the system are the labelling of regions in the face and the manual creation (or semi-automated generation) of expressions or movements to be mapped. The first manual step is to select which vertices belong to which parts of the face, called region labelling. It allows the system to focus on certain regions of the face and adjust the training to best match specific regions of the face, rather than having to be used for the entire face. This step is discussed further in Section 3.3. The other step requiring manual work is expression creation or extraction, where the user must select expressions for the mapping system to consider during the mapping. For this project, a semi-automated expression extraction is used, where the user selects which animation parameters are to be used and at which intensities. The system automatically generates a set of expressions (or basic movements) that correspond to the parameters the user chose. The expression extraction is discussed in Section 3.4.

The other stages of the system are completely automated, meaning the user does not need to give any information related to the faces. The system reads in the specifications for the face as well as the labelling and expression information provided by the user. This information is processed using fuzzy membership functions to convert the information on both faces into an intermediate space to compare the vertices and the expressions. This intermediate space, generated using fuzzy sets and membership functions, is discussed in Section 3.5. Since each face is different, the system performs unsupervised learning to set weights and other parameters in the fuzzy membership function component, and this learning stage is discussed in Section 3.9. Additional information as to the design of the system is in Appendix 1.

3.2. Specific Face Models used in this Project

To test and tune the proposed system, two specific face models were used, each with different implementations. The first model, to represent a simple low resolution model, was the Candide3 face [31]. The second model is a higher resolution 3D anthropomorphic face model from [37]. These models were selected because they are of different classes: shape-based and muscle-based, respectively. Additionally, these models have different characteristics, including different movements available to the

mouth, so mapping between them allows for a comparison as to what happens if mappings are performed between very dissimilar models.

3.2.1. Candide3 Face

The Candide3 face is an intentionally low-resolution shape-based face proposed in [31]. It contains 113 vertices and 168 triangles, with parameters that control the following movements and characteristics of the face:

- Action Units: 65 parameters that control expressions
- Shape Units: 14 parameters that control the shape of the face
- 3 parameters for global rotation of the whole model along X, Y, and Z axes

The Action Units and Shape Units are defined as linear deformations. Only the Action Units of this face are used, since this project had no need to vary the shape of the face or move the entire model. The Candide3 model was chosen because of its ease of use, availability, and because the Microsoft Kinect for Windows outputs are compatible with the definition of the Candide3 face [63]. Future works would therefore be able to use Kinect for Windows sensors to generate expressions for this system. Additionally, with 65 different parameters for expressions, the Candide3 face is able to recreate many expressions. A render of the Candide3 face is shown in Figure 8.

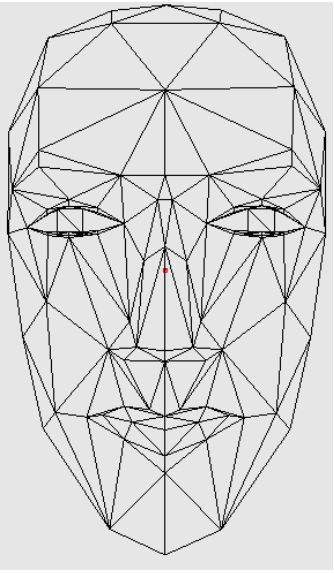


Figure 8: Neutral Candide3 Face

3.2.2. 3D Anthropomorphic Muscle-based Face

The second face used in this system is the muscle-based face of [37]. It is a much higher resolution face, with 512 vertices and 934 triangles. Despite its higher vertex count, this face has much fewer parameters, with only 7 deformations for expressions. It also contains 7 parameters for expressions and 11 parameters, which are not used in this project, for altering the shape of the face. This face was chosen because its deformers behave similarly to a potential face robot configuration without the computational complexity of a physically-based model, and it is very different from the Candide3 face, making use of the proposed algorithm's ability to map between different face types. The main differences between the Candide3 face and the muscle-based face are the following:

- Resolutions: The muscle-based face has more vertices.
- Scale: The two faces are of different sizes from each other.
- Possible actions: The muscle-based face cannot control its eyes, lip raisers, and horizontal movement of the jaw, while the Candide3 face can.
- Different implementations of animation: The muscle-based face moves lip corners individually, while the Candide3 face moves them together in its main lip animation.
- Internal implementations: The muscle-based face uses nonlinear relationships between parameter value and vertex movements, while Candide3's expressions are linear.

A render of the muscle-based face is shown in Figure 9.

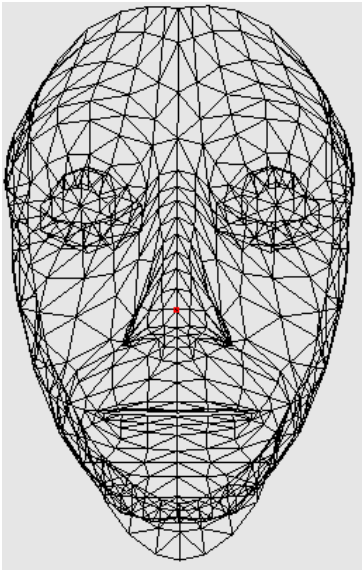


Figure 9: Neutral Muscle-based Face

3.3. Region Labelling

The purpose of the manual region labelling is to prevent mappings between unrelated portions of the face, like the mapping of an eyebrow movement to a jaw muscle movement. In addition, having specific regions of the face isolated allows the system to adjust parameters for each portion of the face during training, rather than having a set of parameters that work on the whole face. This is important because different parts of the face have very different types of movements, and because the fuzzy membership functions scale to the largest displacement in a particular region for better precision.

Labelling a vertex in this system adds that vertex to a list of vertices that are in that region. There is no order of vertices, nor are there any strict criteria that must be used in the tagging process. The region tagging function in this system was designed to be simpler than other implementations. Unlike tagging for ASM or AAM based systems such as those described in [34], vertices do not have to be labelled in any particular order. In addition, since this system automatically clusters vertices, there can be a different number of vertices in each face. The ability to use faces with different numbers of labelled vertices is an advantage over AAM-based and some shape-based FEC systems, which require the same number of vertices in both the source and target faces. Vertices that do not actually move during the expressions can also be labelled without affecting performance, because the system automatically rejects vertices that only have movements below a threshold. For the simulated faces, this threshold is set to a nonzero value which is lower than the distance moved by any vertex.

To perform the labelling, the user must manually select vertices on the face and add them to the list of vertices for each region. For ease of use, this process is performed using a graphical interface, shown in Figure 10. Since certain vertices are very close to each other or overlapping for the neutral face, the labelling interface also allows deformations to be performed while labelling. The labelled faces are shown in Figure 11 and Figure 12.

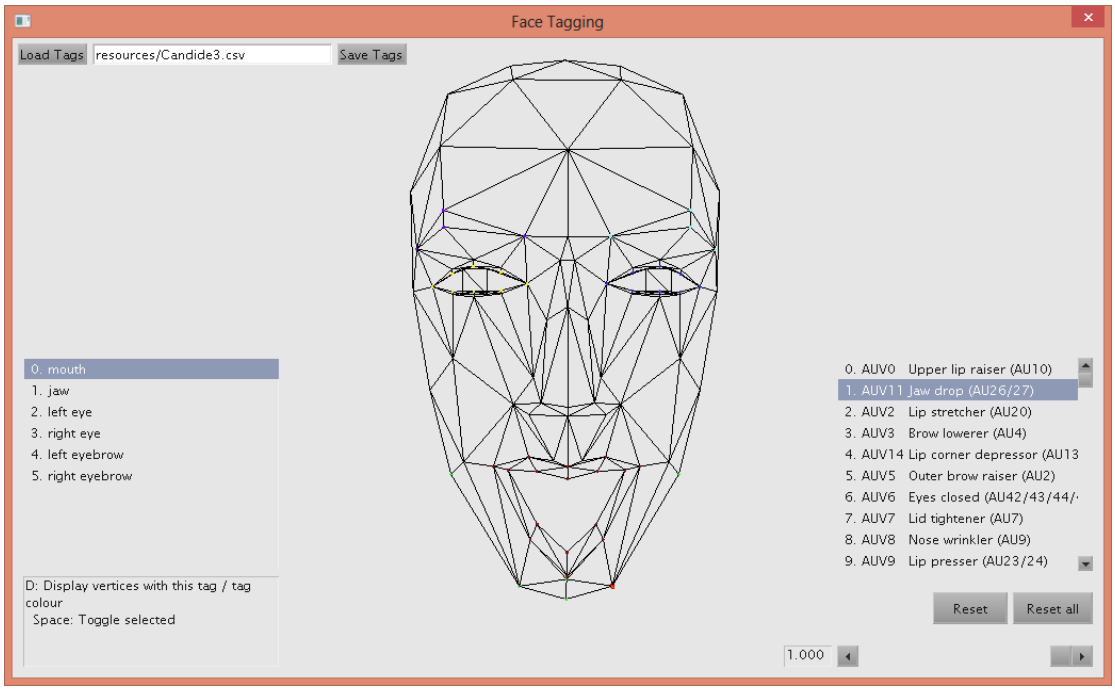


Figure 10: Labelling Interface Displaying Candide3 Jaw Drop

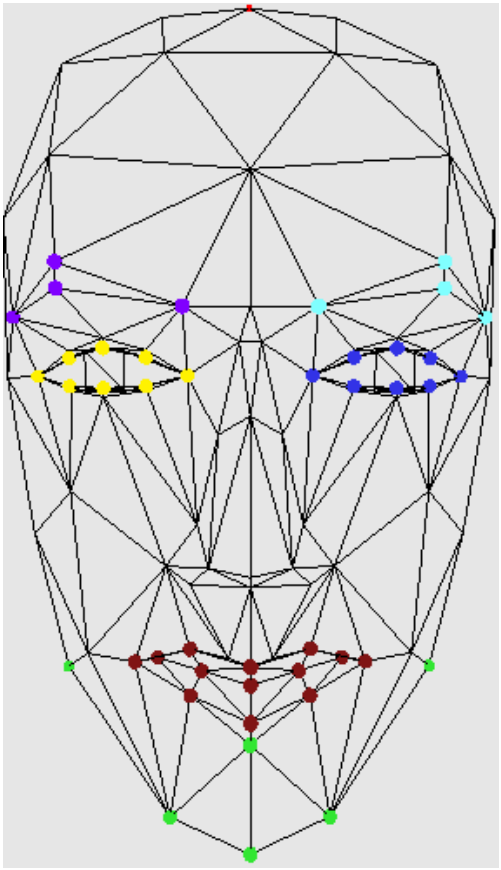


Figure 11: Candide3 Face Labelled Regions

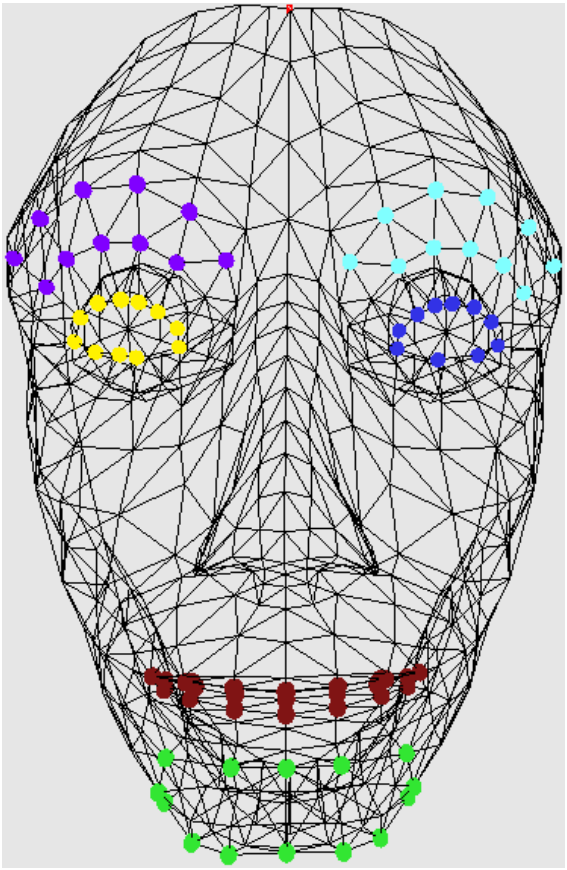


Figure 12: Muscle-Based Face Labelled Regions

In Figure 11 and Figure 12, the following regions are labelled, along with the colour used to display them:

- Lips (dark red)
- Chin (green)
- Left eye / eyelids (blue)
- Right eye / eyelids (yellow)
- Left eyebrow (cyan)
- Right eyebrow (purple)

3.4. Expression Extraction

The expression extraction stage generates the lists of expressions for the system to use. In the testing of this system, the expressions are basic movements created by deforming each face with a controlled set of parameters. The use of basic movements instead of "prototypical" expressions such as in [17] was to allow a wider set of movements to be tested, and it allows testing of possible movements that are not seen in the prototypical expressions, such as movement of single lip corners. The expression extraction stage generates lists of parameters and the corresponding vertex locations, created by altering a single parameter at a time. For each parameter, the expression extraction stage iterates through a set number of non-neutral positions. It saves the value of the parameters used as well as the position of every vertex in the face. This information is used in the mapping stages to match the movements together and then recover the parameters used to generate these movements. Note that the FEC algorithm does not calculate any new face expressions by moving the actual face; it only works with recorded expressions or with the list of expressions created in this stage. Restricting the system to use recorded expressions (without extrapolating new ones in training) allows it to be ported to a robot face in the future. If the target face is a robot, the system will not have the liberty to attempt thousands of different candidates for cloning due to the time it would take for the actuators to move the face. Additionally, only single parameter expressions were generated because the memory usage to enable more parameters per expression had factorial growth. For instance, there are ${}_{65}C_4 = 677040$ four parameter combinations on the Candide3 face (with 65 parameters to choose from). To record all the vertex positions for these movements, and for multiple values per parameter, would require tens or hundreds of gigabytes of memory.

The movements were selected by setting which parameters were to be used, and the range of values to be used for each parameter. For the Candide3 face, any parameter that moved parts of the faces that used for expressions (the animation unit vectors) were used. Similarly for the muscle-based face, the parameters to move the muscles used for expressions were used. Although both faces support parameters that change the shape of the face, including the size of the jaw and nose, these parameters were not used.

3.5. Facial Expression Mapping

The mapping component of the proposed system is the component that performs the FEC. It takes the labelled regions and recorded expressions from the previous components for both faces as an input. It then converts the expression information for selected face regions into an intermediate space using fuzzy membership functions. From the intermediate space, it compares the source and target expressions, selecting the best target expression for each source expression. This component's output is lookup table which states which target face parameters correspond to each set of source face parameters for expressions. The mapping component's workflow is sequential, and the flowchart for the steps required to perform the mapping is shown in Figure 13.

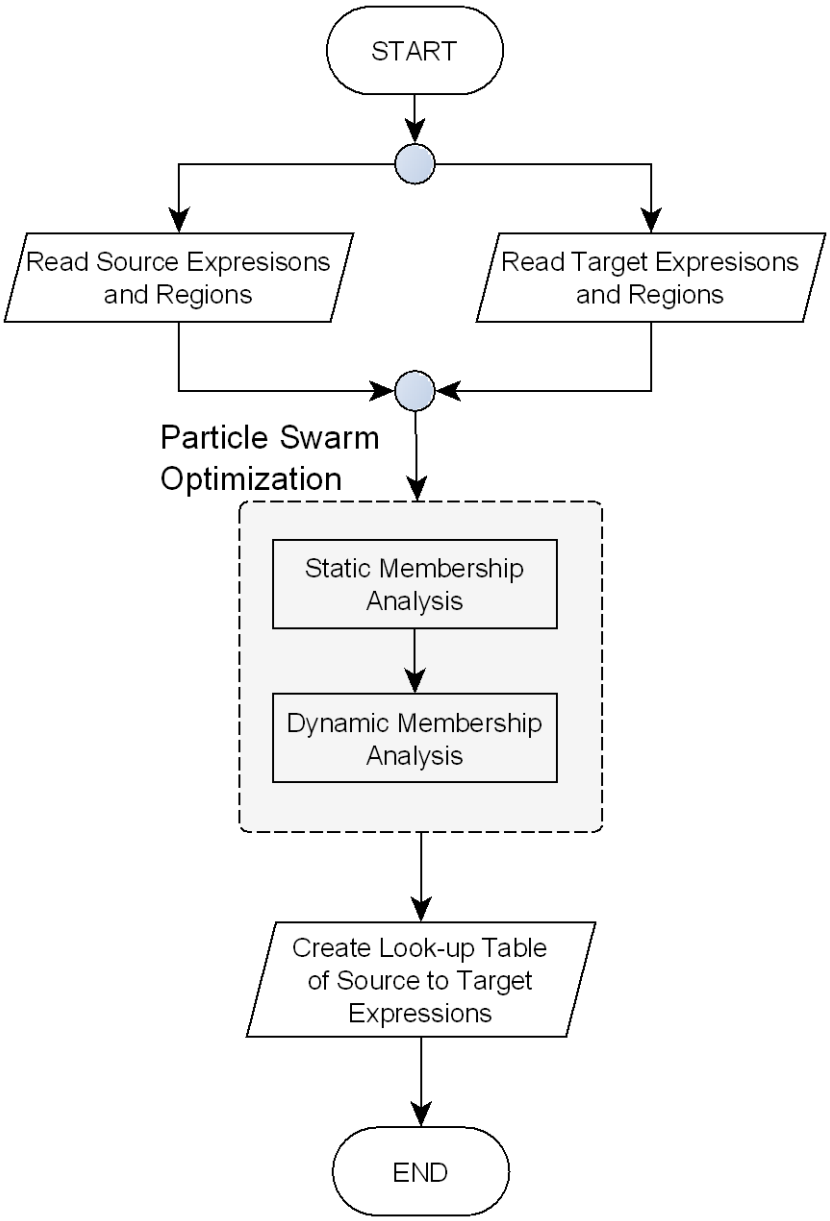


Figure 13: Flow Chart for Mapping

The system processes the information of the two faces in several stages. The first stage is the static membership analysis, which examines where vertices are in the labelled regions and which movements each vertex can perform. The static membership analysis generates weights to use in the dynamic membership analysis, which generates the lookup tables of source to target expressions. Both the static and dynamic membership analysis have tunable parameters that are automatically

optimized using particle swarm optimization. The following subsections discuss these stages in more detail.

3.6. Representation of Faces Using Fuzzy Membership Functions

The static and dynamic membership analysis use fuzzy membership functions to take measurements from the faces inputted into the system and represent the measurements in a fuzzy space. Vertices between the source and target face, as well as expressions that move these vertices can be compared in this fuzzy space regardless of scale or variations in the way vertices on both faces move. These functions are broken up into the following categories depending on what they take in as an input value:

- **Static position functions:** Read scaled position of vertices
- **Movement direction functions:** Read angle of vertices during expressions
- **Movement distance functions:** Read distance vertices travelled

There are six static position functions broken up into two groups, related to where a vertex is inside a labelled region when the face is in its neutral state. The first group is the horizontal position group which states whether a vertex is in the left, centre, or right the labelled region. This group of functions reads the normalized position of in X of a vertex, scaled such that the leftmost vertex is -1, the rightmost vertex is +1, and the centre of the region in X is 0. Similarly, the other group is the vertical position group that reads in the Y value of neutral vertices scaled such that the lowest vertex would take a value of -1 and the highest vertex in Y would take a value of +1. Note that there does not necessarily have to be a vertex at the normalized position of (0, 0). The static position functions are all trapezoidal functions, and they are shown graphically in Figure 14.

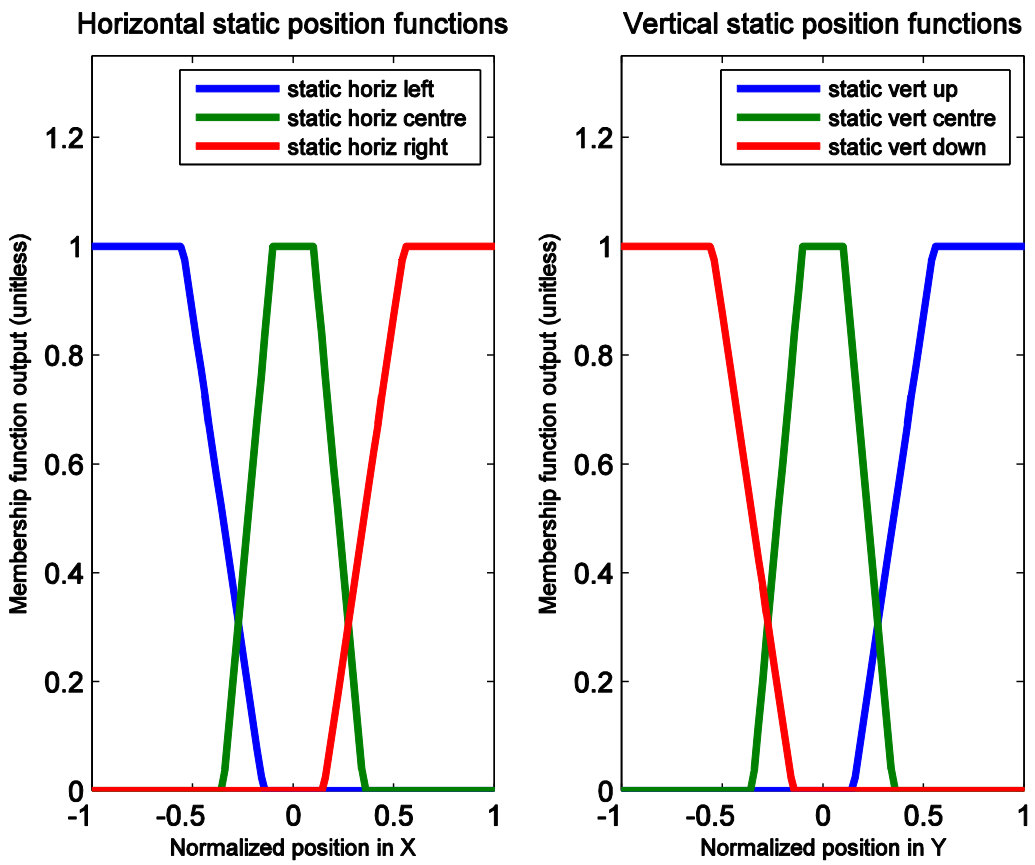


Figure 14: Static Membership Functions

To measure movements of vertices during expressions, 11 more membership functions are used. These functions are divided into the movement distance and movement angle functions, which take the measurements of the magnitude and angle of the displacements of individual vertices, respectively. The X and Y direction component for movements of each vertex are converted into polar coordinates. Note that in this implementation, the depth component (Z) is not used because the faces tested had very little movement in Z, and for robot target faces, low-cost measurement systems such as the Microsoft Kinect sensor have low precision in Z [71].

The movement distance functions take the magnitude of the movement of a vertex, $||(x_{ij}, y_{ij})||$, where i and j are the vertex and expression being evaluated, respectively. The each individual displacement is normalized by dividing it by the

largest displacement recorded for any vertex in that region of the face (and for any expression):

$$n_{ij} = \frac{||(x_{ij},y_{ij})||}{\max(||(x_{ij},y_{ij})||)} \quad \textbf{(4)}$$

Note that n_{ij} term from (4) takes a value of 0 for no movement, and a value of 1 for the largest movement of the vertex. This term is used as the input for the movement distance functions, shown in Figure 15. There are three trapezoidal movement distance functions, corresponding to subtle, moderate, and extreme displacements of vertices. There are multiple classes of displacements because smaller movements are equally important in facial expressions. For instance, most facial expressions used for conversational cues are subtle [24].

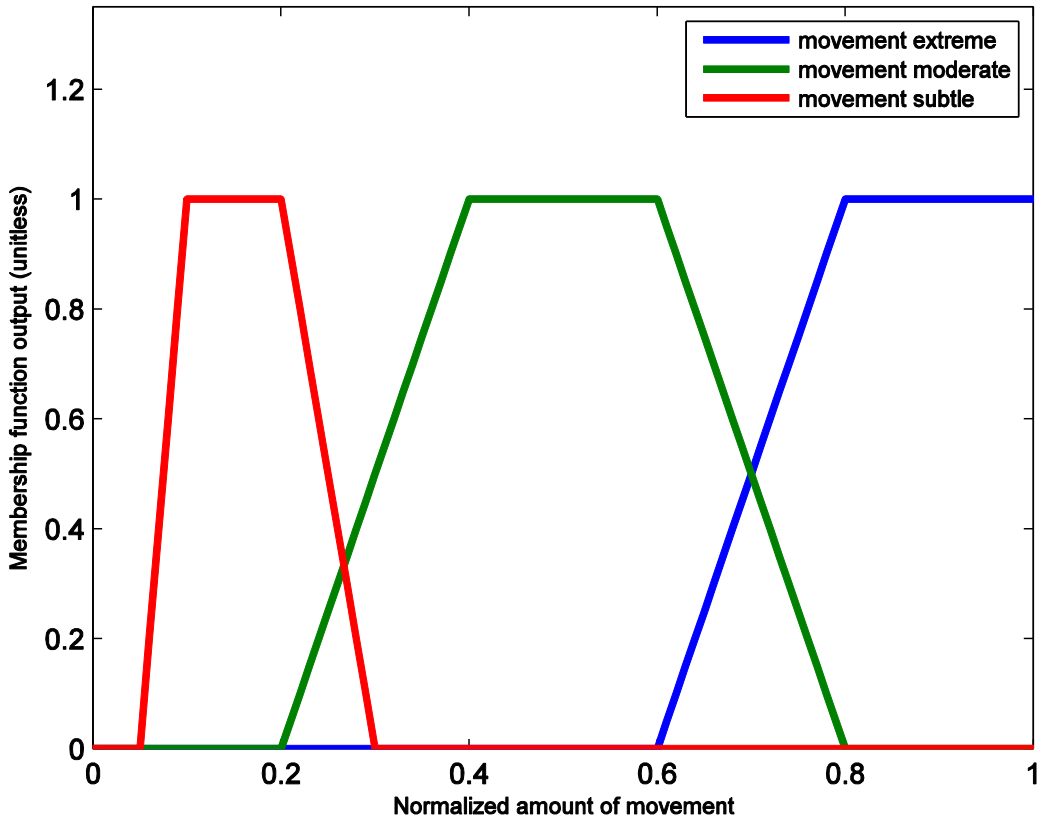


Figure 15: Movement Distance Functions

The direction (angle) component of the displacement converted to polar coordinates is processed by the movement direction functions. The direction is

important to consider because there are movements that have similar distance but a different direction. For instance, there is the eyebrow raiser movement and the eyebrow lowerer movement, both of which control the same eyebrow but they move in opposite directions. There are eight functions, each corresponding to a range of angles that the vertex is moving in. The angle is inputted directly into the functions, shown in Figure 16, to be evaluated. Note that in Figure 16, the functions are separated into two charts to make the presentation clearer; all eight functions work on the same angles.

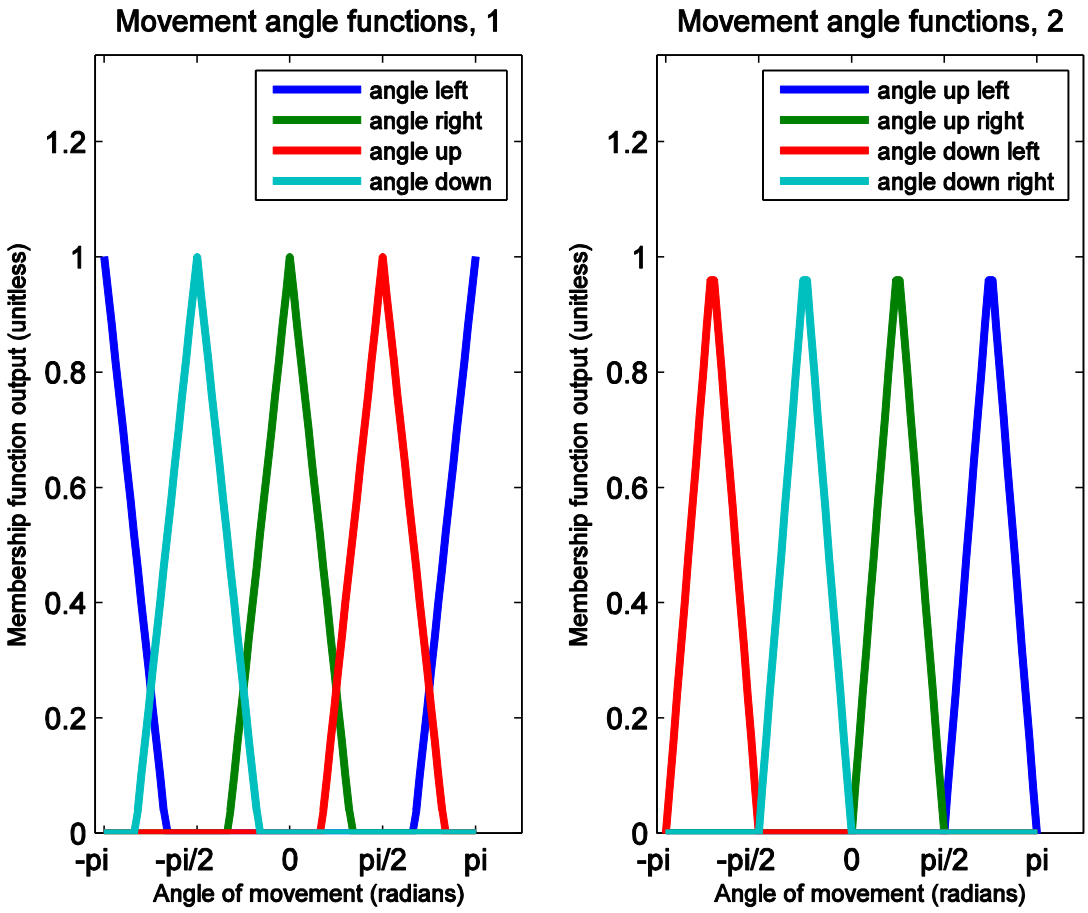


Figure 16: Movement Angle Functions

These functions allow vertex positions to be represented by six fuzzy values and movements to be represented by 11 fuzzy values. In total, this allows each vertex during each expression to have a 17-dimensional vector of these fuzzy values to be used for comparison. Since scaling is performed before evaluating these functions, this representation allows faces of vastly different shapes to be compared. The fuzzy

membership functions are used in both the static and dynamic membership analysis to convert measurements from the face into an intermediate form for comparison.

3.7. Static Membership Analysis

The static analysis step analyzes each vertex to determine its relative position in the labelled region and any movement the vertex is capable of. It therefore classifies the vertices. The steps in the static analysis are the following:

1. Run membership functions on each vertex
2. Multiply the membership values for each vertex by a set of weights
3. Cluster the source or target vertices to group together very similar vertices
4. Compare the cluster centres with the source vertices
5. From the comparison, produce weights to influence the dynamic analysis stage

The first step of the static membership analysis is to run the membership functions on each vertex. This stage uses all 17 functions from Section 3.6. The neutral positions of the vertices are used as inputs to the static position functions. The movement distance and movement angle functions are also taken for each vertex and for every expression. To collect the information of what movements each vertex is capable of, a union across all of the expressions is performed for each movement distance and movement angle output. The union is performed by taking the max for each function output across all expressions. If the movement of a vertex never goes above a certain value, the noise threshold, the vertex's information is discarded. For virtual faces, the noise floor was set to a value below any specified movement in the face, since the noise in the virtual faces was effectively 0 (aside from rounding errors). After the membership functions are run, the position and possible movements of each vertex i in the face is represented by a 17-dimensional vector, $\boldsymbol{s_i}$. Since $\boldsymbol{s_i}$ is a collection of fuzzy logic values, each element is in the range [0,1].

For each pair of faces, certain positions and capabilities may be more important than others. This was discovered in early testing since emphasizing or even removing certain elements from the 17-dimensional vectors had a positive impact on the final mappings between the Candide3 face and the muscle-based face. Therefore, one of the adjustable parameters to the static membership analysis is the diagonal matrix of weights $\boldsymbol{W_S}$ that multiplies each vector ($\boldsymbol{s_i}$) of position and capabilities:

$$\boldsymbol{v_i} = (\boldsymbol{s_i})(\boldsymbol{W_S}) \quad \textbf{(5)}$$

3.7.1. Clustering of Vertices

Before being used in the dynamic membership analysis, all of the $\boldsymbol{v_i}$ vectors are grouped together with subtractive clustering. Subtractive clustering finds groups of vectors (called clusters) in the data where the distance between vectors in each cluster is less than a certain radius. In this implementation, the *subclust* function of MATLAB is used [72]. Although the algorithm was designed to help generate fuzzy rule bases, it was useful for this system because it is a clustering method with a continuous parameter (the radius). This is unlike K-means clustering, which has a discrete parameter (the number of clusters) [73]. A continuous variable to modify for the clustering allows the training described in Section 3.9 to alter the behaviour of the clustering stages, since the training works in a continuous parameter space.

The clustering stage reduces the number of vertices to compare against. If the source face has more vertices in the tagged region than the target, clusters are calculated against the source's weighted vectors. Otherwise, the target's weighted vectors are used for the clustering (including if source and target have the same number of vertices). Clustering also allows very similar vertices to be counted as one element for the dynamic analysis.

The cluster centres are used to produce weights for dynamic analysis. Two sets of weights are produced and are stored as matrices:

- Source vertex to cluster centre $\boldsymbol{W_{SaC}}$
- Target vertex to cluster centre $\boldsymbol{W_{SbC}}$

The matrix $\boldsymbol{(W_{SaC})}$ is of size $Sa \times C$, and the matrix $\mathbf{(W_{SbC})}$ is of size $Sb \times C$, where C is the number of cluster centres found by the clustering, *Sa* is the number of labelled vertices in the source, and *Sb* is the number of labelled vertices in the target. The distances from each weighted vector $\boldsymbol{v_i}$ to each cluster centre $\boldsymbol{c_j}$ ($j \in [1,C]$) is measured. Note that all of the $\mathbf{v_i}$, from both the source and target, are used. The elements of $\boldsymbol{W_{SaC}}$ and $\mathbf{W_{SbC}}$, denoted below as $\underline{\boldsymbol{l_{ij}}}$, are computed by normalizing each distance $||\boldsymbol{v_i} - \boldsymbol{c_j}||$ to the smallest and largest distance per cluster centre:

$$\boldsymbol{l_{ij}} = \left(1 - \frac{||\boldsymbol{v_i} - \boldsymbol{c_j}|| - min_j}{max_j - min_j}\right)^4 \quad (6)$$

$max_j = \max(||\boldsymbol{v_i} - \boldsymbol{c_j}||)$ for any *i* in the face model

$min_j = \min(||\boldsymbol{v_i} - \boldsymbol{c_j}||)$ for any *i* in the face model

Note that the value is taken to the fourth power simply because there were too many vertices with a high weight if the distance alone was used. Each column of $\boldsymbol{W_{SaC}}$ and $\boldsymbol{W_{SbC}}$ is then divided by the sum of that column, so the weights in each column of the matrix adds to 1. For the source face, the sum is over S_a rows:

$$w_{ij} = \frac{||\boldsymbol{v_i} - \boldsymbol{c_j}||}{\left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{Sa}} ||\boldsymbol{v_i} - \boldsymbol{c_j}||\right)} \quad (7)$$

Equivalently, for the target, the sum is over S_b rows:

$$w_{ij} = \frac{||\boldsymbol{v_i} - \boldsymbol{c_j}||}{\left(\sum_{\boldsymbol{i=1}}^{\boldsymbol{Sa}} ||\boldsymbol{v_i} - \boldsymbol{c_j}||\right)} \quad (8)$$

The values of the $\boldsymbol{W_{SaC}}$ and $\boldsymbol{W_{SbC}}$ are passed to the dynamic membership analysis to determine how much influence each target vertex in an expression should have when the expressions are compared with each other.

3.8. Dynamic Membership Analysis

The dynamic analysis produces the final mappings, which is a lookup table between source and target parameters. The steps in dynamic analysis are:

1. Apply weights from static analysis to expressions' sets
2. For each pair of source and target expression, subtract the weighted membership function of the target from the source to find the error
3. Select pairs with the smallest error

The dynamic analysis first analyzes the movements of every vertex in every expression using the movement angle and movement distance membership functions. There are 11 of these membership functions. The other six functions, the static position functions, are not used directly in this stage because the weighted values influenced by these functions come from the static analysis, and applying these values again would be redundant. The outputs from these functions are stored in matrices. Source expression data is stored in one *Sa×11* matrix $\boldsymbol{DYNa_{ea}}$ per source expression *ea*, and target

expression data is stored in a *Sb×11* matrix, referred to as $\boldsymbol{DYNb_{eb}}$, where *eb* is a target expression. The weight matrices from the static analysis are combined with the membership function outputs in this step with matrix multiplication. The multiplications for the source and target vertices are performed in (9) and (10), respectively.

$$\boldsymbol{WDYNa_{ea}} = (\boldsymbol{W_{SaC}}')(\boldsymbol{DYNb_{ea}}) \quad \textbf{(9)}$$

$$\boldsymbol{WDYNb_{eb}} = (\boldsymbol{W_{SbC}}')(\boldsymbol{DYNb_{eb}}) \quad \textbf{(10)}$$

The multiplications in (9) and (10) apply the weights from the static analysis, producing the matrices $\boldsymbol{WDYNa_{ea}}$ and $\boldsymbol{WDYNb_{eb}}$. These matrices allow the selection of expressions to be biased by the similarity of source and target vertices (dictated by the static analysis). Additionally, these matrices are of size *C×11*, regardless of the differences in number of vertices between the two faces. Since these matrices are now the same size, a simple subtraction can be used to compare any two facial expressions between the source and the target:

$$\boldsymbol{ERR_{ea,eb}} = \boldsymbol{WDYNb_{eb}} - \boldsymbol{WDYNa_{ea}} \quad \textbf{(11)}$$

The $\boldsymbol{ERR_{ea,eb}}$ can then be analyzed column-by-column, or by the sum of the columns to extract characteristics about any pair of expressions (ea, eb). In particular, the matrix can be observed for the following characteristics for each potential mapping:

- Each column of $\boldsymbol{ERR_{ea,eb}}$ represents the output for a certain membership function on its own for that expression
- Negative entries mean a new action is being introduced by the target expression, while positive values mean not all of the source expression could be reproduced
- If the sum of the absolute value of all elements in $\boldsymbol{ERR_{ea,eb}}$ is larger than $\boldsymbol{WDYNa_{ea}}$, then the error is greater than simply not making the target face move at all

3.8.1. Creation of Source-Target Parameter Pairs

The final step in the system is the selection of which source expression is the best match for a target expression. To perform this selection, the $\boldsymbol{ERR_{ea,eb}}$ of each pair of

source and target expressions is looked at. The selection occurs with the procedure below:

1. For each *ea*, create matrix $\boldsymbol{EPTS_{ea}}$ that collects the sum of all errors with regards to the angle and magnitude of the movement over all vertices for each target expression *eb*
2. Perform subtractive clustering along the rows of $\boldsymbol{EPTS_{ea}}$
3. Reject any clusters with very poor suitability
4. Of all remaining clusters from $\boldsymbol{EPTS_{ea}}$, select the cluster with the lowest angle error
5. Select the single candidate expression with the lowest magnitude error

This procedure is repeated for every source expression, until an attempt to map every source expression to a target expression was found.

The first step in the selection is to create a matrix that collapses all the error information from the $\boldsymbol{ERR_{ea,eb}}$ matrices for each expression. For each source expression *ea,* this error information is collected in a matrix called the error points matrix, called $\boldsymbol{EPTS_{ea}}$. This matrix is of size $N_{eb}\times 2$, where N_{eb} is the number of target expressions being considered as candidates for mapping to a given source expression. Each row represents the cumulative error across all vertices that would result if the source expression being considered was to be mapped to the target expression represented by that row. Each row has two columns because the error is considered separately in terms of angle and in terms of magnitude of movements of the vertices. The elements in the first column represent the angle error, taken by performing a double sum across all elements of the $\boldsymbol{ERR_{ea,eb}}$ matrix that represent the error in the movement angle functions. Likewise, the elements in the second column are created by performing a similar double sum over the elements that represent the error in the movement distance functions.

A subtractive clustering is performed on the rows of the $\boldsymbol{EPTS_{ea}}$ matrix, using the same algorithm as the clustering in Section 3.7.1. The clustering is performed so the target expressions can be selected by their angle first, and then their magnitude. Clustering helps group expressions which are the same movement but of different intensities. This allows the algorithm to more appropriately pick the correct intensity

for a movement and have more mappings in general. The radius of the clusters in this step is an adjustable parameter controlled in the optimization process. The cluster centres are then evaluated to find which one has the lowest cumulative angle error. This is also the stage where target expressions can be rejected if there really is no target expression that has similar movements to the source expression being analyzed. Two rejection criteria were used:

1. Angle error: Reject any clusters with a negative angle error lower than a certain threshold: This criterion makes sure that a completely different movement is not mapped to a source expression.

2. Worse than neutral: Reject any clusters where the total error would be larger than mapping to the neutral target face: This criterion also rejects poor mappings that would result in undesirable movements in the target face.

For most of the mappings, the first criterion was used, although for certain mappings, to prevent situations where there are large magnitude errors because the algorithm was attempting to find a low angle error, the second criterion was used. The differences in the use of the error criteria are discussed in the next section.

If all the clusters were rejected, this source expression is recorded as a failed mapping, which means the algorithm failed to find a proper match for a particular source expression. Otherwise, the cluster which has the cluster centre with the lowest *angle error* is selected, which selects from target expressions with the largest amount of similar movements in terms of the direction. Of that cluster, the final mapping is selected as the target expression in the cluster with the lowest magnitude error.

Although the algorithm only does a one-to-one mapping for each region, it can combine mappings of the same source expression that were mapped to different target expressions for each region. For instance, the dual eyebrow raiser of the Candide3 face was mapped to a left eyebrow raiser on the left eyebrow region, and similarly for the right eyebrow region. This combination was performed for each parameter separately by taking the average of all nonzero values for that parameter across mapping pairs.

For instance, if a target parameter was used in three pairs with the value of 0, 0.5, and 0.25, then the combined parameter value is the average of 0.5 and 0.25 (0.375). This simple combination worked well since there were no conflicting actions being recorded across the different face regions.

3.9. Training using Particle Swarm Optimization

This system uses particle swarm optimization (PSO) as part of its training stage to adjust different parameters and attempt to create a reasonable set of source to target parameter pairs. PSO was chosen for training of this system because it is an unsupervised learning technique and the behaviour of particles in PSO is a good way to explore the space of the weights and radii used in this problem. If supervised learning techniques (that require examples or counter-examples to operate) were used, the system would require users to suggest suitable pairs of expressions or give information as to the algorithm behind animation of the source and target faces. Unsupervised learning allows the proposed system to operate without this prior knowledge of mappings, allowing the system to be more flexible.

A genetic algorithm (GA) was also considered as it is another unsupervised learning technique [74]. In a GA, exploration of the problem space is created by "birth" of new entities (children) by combining traits from entities in the current generation (parents). However, the static analysis weights cannot be randomly combined together in this manner due to the normalization performed by Equation (6). If the combination of weights chosen for a child affected the normalization terms, the child's position in the problem space could be completely different from the parents. This could happen for example, if the parents had a combination of large and small weights, but the child was given all of the small weights. The child's weights would all become large after the normalization of (6), since the term in the denominator would be smaller than the term used in the parents. The child would therefore *not* be similar to its parents. To use a GA to select new static analysis weights, an additional verification or normalization would have to be added to make children have similar normalized weights as their parents after being processed by Equation (6). The PSO is not affected by this issue because no

such randomized pairing of weights is performed. Therefore, the PSO was selected over the GA for the simpler selection of candidates per iteration.

PSO is an unsupervised learning method that optimizes some fitness value for a problem by having many entities, called particles, explore the problem space [75]. The particles are all part of a group, called a swarm. Particles inside a swarm share information, including the best solution found by the swarm at any given time. There are many variants of PSO, although they all run in the general way shown below:

```
Initialize set of particles P with randomly chosen positions P.x and velocities P.y
Loop until some stopping condition is met:
    Calculate the fitness of each particle p in P
        If a particle has found a better fitness than before, designate it p_best
    For each p in P, point p.y towards p_best.x, plus some random variations
End loop
```

The general behaviour of PSO is that particles start randomly distributed in the space, and then they collect close together near the best solution found by the swarm. As with any learning procedure, the system can get stuck in local minima and generate suboptimal solutions. The specific PSO variants used in this project are discussed in Sections 3.9.3 and 3.9.4.

In this system, the PSO explores the space of tunable parameters during the mapping of a pair of faces by repeatedly mapping facial expressions from the source face, to the target face, and back to the source face. Similar to the work in [11], the original source expressions are compared to the reconstructed source expressions. The cloning of source to target and then back to source is referred to in this paper as *reflexive mapping*. Note that since the Candide3 face has 65 parameters and the muscle-based face has 7 parameters, theoretically the best reconstruction that can occur with this algorithm is to have $\frac{7}{65} \cong 10.8\%$ of the source expressions back from the reflexive mapping. The workflow for training and error management is shown in Figure 17.

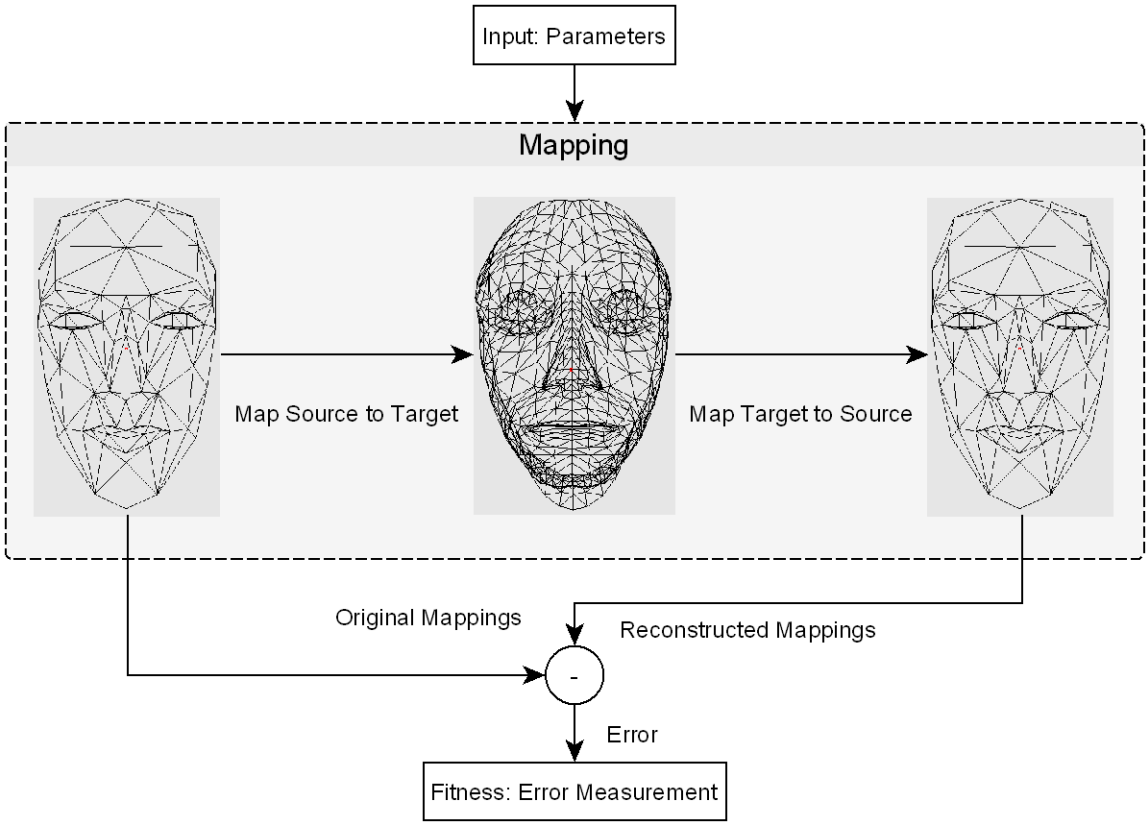


Figure 17: Workflow for Training and Error Measurement (Reflexive Mapping)

3.9.1. Controlled Parameters

During training, the PSO explores the space of the facial expression mapping problem for a pair of faces. The particles in the PSO vary several parameters in the mapping, including the weights used for the matrix $\boldsymbol{W_S}$ in the static analysis and the radii used for the subtractive clustering. Each particle had 32 dimensions to its position and velocity, corresponding to 14 weights for the static analysis + 2 radii from clustering stages. The PSO was tested against older results found in earlier versions of this system in [76], shown in the next chapter. The description of each dimension of the position of the particles is shown in Table 4.

Table 4: PSO-Controlled Parameters

Index	Direction	Description
1-10	⟶	Static Analysis, Static Position Weights Static Analysis, Movement Angle Weights: Up, Down, Left, Right

11	⟶	Static Analysis, Movement Angle Weights: Diagonals
12-14	⟶	Static Analysis, Movement Distance Weights
15	⟶	Static Analysis, Clustering Radius
16	⟶	Dynamic Analysis, Clustering Radius
17-26	⟵	Static Analysis, Static Position Weights Static Analysis, Movement Angle Weights: Up, Down, Left, Right
27	⟵	Static Analysis, Movement Angle Weights: Diagonals
28-30	⟵	Static Analysis, Movement Distance Weights
31	⟵	Static Analysis, Clustering Radius
32	⟵	Dynamic Analysis, Clustering Radius

Where ⟶ represents source to target mapping and the ⟵ represents target back to source mapping

Note that the diagonal movement angle sets were all controlled by the same weight to slightly reduce dimensionality, and because the other movement angle sets overlap these sets. Additionally, the PSO only affected static analysis weights since it would not be productive to apply the weight to the dynamic analysis. Doing so would greatly increase the dimensionality of the problem and it would be redundant as dynamic analysis is already influenced by the static analysis weights. With this set of parameters, the PSO is able to automatically select parameters for the FEC algorithm to minimize mapping errors.

3.9.2. Fitness Criteria

As this is unsupervised learning, the particles require a fitness criteria, which is a single number that states how good a result is. The PSO uses the criteria to find if a result is better or worse than what was previously found. The fitness criteria for this system is measured by the error per vertex, the number of perfectly restored source mappings, and the number of remaining original mappings.

The error per vertex is the difference between the original vertex position and the reconstructed vertex position for every mapping. For each mapping *j* and for each vertex *i*, the original expression is denoted by $\boldsymbol{o_{ij}}$, and the reconstructed expression is

denoted by $\boldsymbol{r_{ij}}$. To make the error comparable between face models, the error per vertex is scaled to the largest deformation d_i recorded for that vertex. Note that completely failed mappings are not counted in this error since no difference can be calculated for these mappings. The errors are averaged over all mappings, M_1, across all vertices V, that the system was able to create. The average of these errors is the first fitness criterion, the mapping error component, denoted by fit_{map}:

$$fit_{map} = \frac{1}{V}\sum_{i=1}^{V}\left(\frac{1}{M_1}\sum_{j=1}^{M_1}\frac{||\boldsymbol{r_{ij}} - \boldsymbol{o_{ij}}||}{\max(d_i)}\right) \quad \textbf{(12)}$$

Since the proposed system works with recorded or generated expressions only and does not create new expressions, its mapping quality can also be measured by finding how many expressions were recreated perfectly, and how many of the original mappings were lost. Mappings can be lost completely if multiple source expressions map to the same target expression, therefore losing the information from the source expressions. In other words, the number of mappings from the source to target mapping can be greater than the number of mappings from the target back to the source. The perfectly restored mappings criterion, $fit_{restored}$, is inversely related to the ratio between perfectly restored mappings, $N_{restored}$, and the number of all mappings, M_1:

$$fit_{restored} = \frac{M_1 - N_{restored}}{M_1} \quad \textbf{(13)}$$

Note that $fit_{restored}$ gives the minimum value of 0 if all of the mappings had no error.

The final fitness criterion, fit_{lost}, describes how many original mappings the system generated. It is inversely related to the ratio between the number of unique mappings, M_2, and the total number of mappings:

$$fit_{lost} = \frac{M_1 - M_2}{M_1} \quad \textbf{(14)}$$

The fit_{lost} criteria is designed to give the minimum value of 0 if there were no duplicated mappings.

All of the criteria are added to form a fitness single value for the PSO to use:

$$fitness = fit_{map} + fit_{restored} + 2(fit_{lost}) \quad \textbf{(15)}$$

The lost criterion has a higher weight than the other criteria because in previous tests, the PSO would generate unfavourable results with many lost mappings in order to have a low mapping error.

3.9.3. Original PSO Algorithm

In this project, multiple variations on the PSO algorithm were attempted. The first one tested was the original PSO algorithm proposed by Kennedy and Eberhart [75]. The testing with the original PSO is shown in Section 4.1. The second variant of the PSO is a small modification on the original PSO, where the position of particles is bounded.

The original PSO algorithm is shown below. It gives a random starting position and velocity to the particles, and particles are moved with random variations towards both the best fitness found by the particles themselves and to the best position found by the entire swarm.

Initialization

Let *P* be a particle swarm with *n* particles

Initialize $P.x_{best}$ to store best position in the whole swarm

$P.f_{best}$ <= worst possible fitness (e.g. +∞ or -∞)

For each *p* in *P*:

p.x <= rand(d)

p.y <= rand(d)

$p.f_{best}$ *<= worst possible fitness*

$p.x_{best}$ *<= p.x*

Running

Begin loop:

(Fitness update)

For each *p in P:*

Compute fitness: *f = fitness for p.x*

If *f* is better than p.f_{best}:

$p.f_{best}$ <= *f*

$p.x_{best}$ <= *p.x*

End if

If *f* is better than P.f_{best}:

$P.f_{best}$ <= *f*

$P.x_{best}$ <= *p.x*

End if

End for loop

(Velocity update)

For each *p in P:*

p.v <= p.v (old) + rand(d) × ($P.x_{best}$ *- p.x) + rand(d) × (*$p.x_{best}$ *- p.x)*

p.x <= p.x + p.v;

End for loop

Loop until some stop condition is met

Where d is the number of dimensions in the problem rand(d) is a vector containing d random numbers between 0 and 1, x is the position; and v is the velocity; <= is the assignment operator; all multiplications are component-wise vector multiplications

Figure 18: Original PSO

Note that the original PSO behaves in a way that the particle that found the best position is only influenced by its old velocity, and other particles move towards the best position found by the whole swarm, while still exploring around their individual best. There are no limits for the position or velocity, so particles can drift towards infinity in the negative or the positive directions. To demonstrate the behaviour of the PSO on a simple problem, the behaviour of the original PSO was tested on the following function:

$$z = 2x^2 + 2y^2 - \cos(2\pi x) - \cos(4\pi x) - \cos(3\pi y) \quad \textbf{(16)}$$

While not representing any meaningful problem, (16) allows the PSO solver to be visualized. Particles search for the lowest *z* by moving along *x* and *y*. It has local minima due to the cosine terms, while the parabolic terms place a global minimum at

the origin. A surface plot of (16) is in Figure 19. The movement of the particles at different intervals is shown in Figure 20.

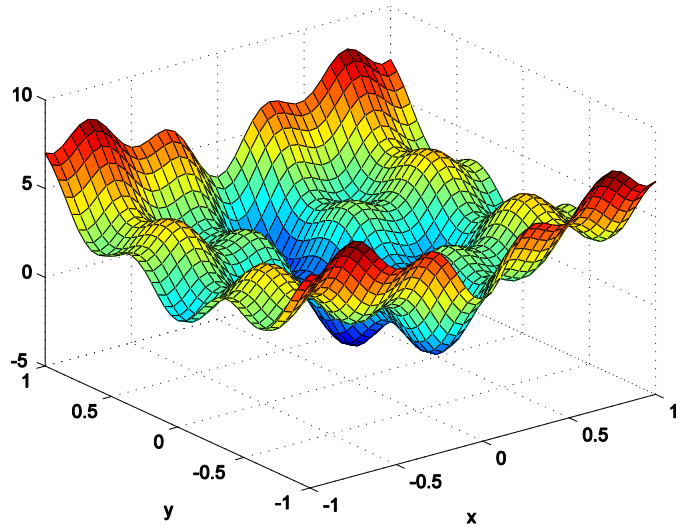


Figure 19: Test Fitness Function for PSO

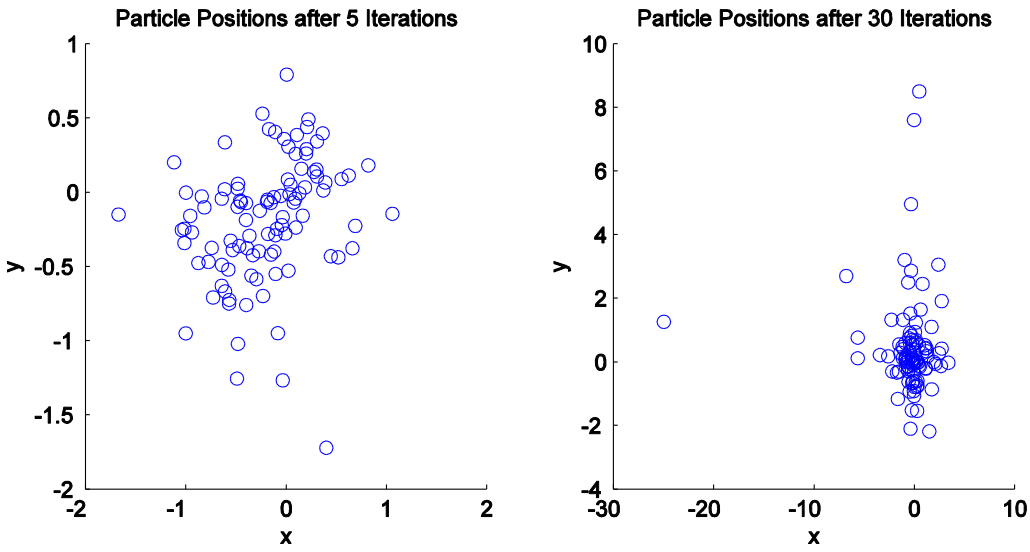


Figure 20: Original PSO Particle Behaviour

These figures demonstrate how the original PSO algorithm behaves by flocking towards the best found solution. In this case, the problem was simple enough that even after 5 iterations most of the particles were moving towards the global minimum at the origin. Note however that after 30 iterations, some particles were diverging. While this

does allow more of the problem space to be searched, if the particles move into areas that are not meaningful to the problem, then they do not contribute to solving of the problem. This behaviour was also noticed for this system (see Section 4.1), so an alternative PSO with bounded position was tested as well.

3.9.4. PSO Constrained by Random Inversion of Velocity Components

To place bounds on the position of the particles, this project tested a variation of the PSO that simply places hard limits on the positions. The changes to the original PSO are as follows:

- Particles are given a limit on their position for each dimension
- Pseudorandom initialization of particles occurs within these limits, with a uniform random distribution
- During the velocity update, if a particle would go out of bounds given its updated velocity and current position, velocity components to prevent the particle from going out of bounds in the appropriate dimensions, called adjustments, are added
- The magnitude of the adjustment is enough to make the particle touch the boundary limit, plus some pseudorandom amount

For each dimension i, the user sets a range $[min_i, max_i]$ to bound the particles. For instance, a user can limit particles to [-1, 1] in one dimension, and [0, 1] in another. These ranges are also used in the initialization as the bounds for the pseudorandom numbers when initializing the particle swarm. In addition, the initial velocity v_i is scaled to be half the magnitude of the ranges, in both the positive and negative directions:

$$v_i \, \varepsilon \left[-\frac{max_i - min_i}{2}, \frac{max_i - min_i}{2} \right] \quad \textbf{(17)}$$

During the execution of the PSO, the velocity update is altered to enforce the position boundaries. The fitness and position update remains the same as the original PSO. The variation on the velocity update is shown below:

(Velocity update)

For each *p in P:*

$s_{adjust} = 2 \times rand(1) + 1$

$p.v <= p.v\ (old) + rand(d) \times (P.x_{best} - p.x) + rand(d) \times (p.x_{best} - p.x)$

temp.x <= p.x + p.v;

If *temp.x* out of bounds

Calculate $x_{overshoot}$ as the amount that temp.x is out of bounds

$v_{adjust} <= - x_{overshoot}$

$p.v <= p.v + s_{adjust} \times v_{adjust}$

End if

End for loop

Where d is the number of dimensions in the problem, rand(d) is a vector containing d random numbers between 0 and 1, x is the position; and v is the velocity; s_{adjust} is a scalar multiplying the adjustment; v_{adjust} is the adjustment velocity vector; temp.x is a temporary position used to calculate v_{adjust} <= is the assignment operator; all multiplications are component-wise multiplications or multiplication by a scalar (for adjustStrength)

Figure 21: Modification to Velocity Update

The adjustment velocity v_{adjust} is calculated as the difference between the particle's position if the velocity update is applied as in the normal PSO and the boundary. Components of v_{adjust} are only calculated for the components of *temp.x* that are outside the boundaries. The components that are outside the boundaries are called $x_{overshoot}$, which is the *temp.x* minus any boundary position that *temp.x* is outside of. To illustrate the effect of this difference, the vectors for a particle overshooting in one of two dimensions is shown in Figure 22.

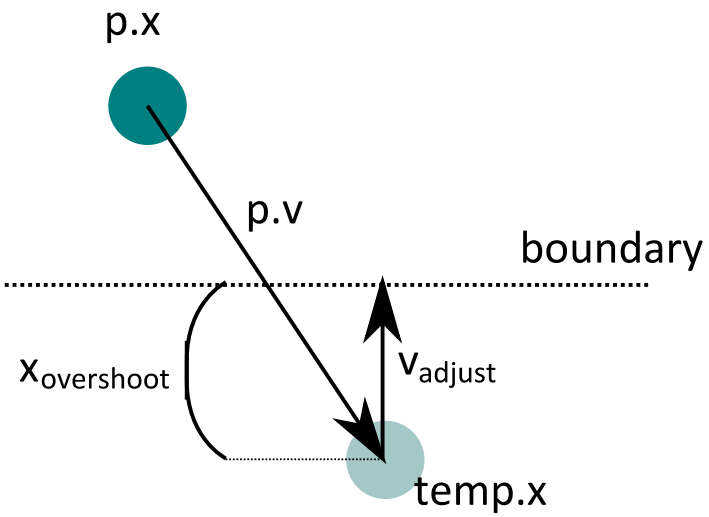


Figure 22: Velocity Adjustment

The adjustment multiplier s_{adjust} is there to prevent the particle swarm from losing energy if particles impact against a boundary. Without this variable (effectively setting s_{adjust} to 1), test runs ended with particles collecting at the boundaries with no velocity remaining. An s_{adjust} of 1 means that the particle will end up exactly at the boundary, while $s_{adjust} > 1$ will make the particle move away from the boundary (and not hit the boundary exactly).

Although there are more complex implementations of constrained PSO [77] [78], this variation was used because the goal was to check how limiting the position of particles affects solving the problem. The constrained PSOs in literature generally use complex methods of deciding where out-of-bounds particles are to go, or they do not guarantee that particles do not go out of bounds. By placing a hard limit on particle positions, the effect of constraining the PSO can be isolated without dealing with more parameters like penalty functions for going out of bounds or particle relocation algorithms.

3.10. Summary

This chapter summarized the algorithms in the proposed system. This system creates facial expression mapping pairs, using expressions recorded from the source and target faces. It requires the user to specify regions of the face for the algorithm to focus on, and each region can have a different set of parameters. To perform mappings, the faces and their respective expressions are represented in an intermediate space of fuzzy membership values. The fuzzy membership values are compared in two stages, the static membership analysis and the dynamic membership analysis. The static membership analysis uses the positions of vertices on the face and the union of movements to see what role each vertex performs in expressions. The dynamic membership analysis looks at expressions individually to pair them up, using weights generated by the static membership analysis. To handle multiple types of faces, subtractive clustering is run on the intermediate space. The clustering enables source and target faces to have different numbers of expressions and different resolutions. To automatically generate usable sets of parameters, PSO is used to explore the target space for each pair of source and target parts of the face.

Chapter 4. Experiments and Results

Different variations of components of the proposed system were tested to find what good solutions for the mappings in terms of error as shown in Figure 17. The experiments examined variations on the particle count for the PSO, the configuration of the PSO, and where sources of error remain. This chapter also discusses the results and where improvements can be made to the proposed system.

4.1. Testing Using Original PSO Equations

The first tests were conducted to find how many particles should be used in the PSO and which variations of the PSO could be used in the system for further testing. The original PSO equations from [75] were used so the results could be observed to find which variations of the PSO could be used for this problem. In addition, the original PSO has the simplest velocity update and its potential issues have been documented [79]. Therefore, undesirable results from running the original PSO can be identified and variations on either the fuzzy mapping or the PSO can be used to address the issues.

In this test, the PSO was varied to find which configuration of PSO could reach the same or a better solution than found in the previous work with this system in [76]. The result from that work, of the possible 10.8% of restored mappings, was a mean mapping error of 0.124535, 7 perfectly restored mappings, and 8 mappings recovered in total. Note that in the mouth region, there are only 24 target face deformations generated by the expression extraction, so the best result could theoretically be 24 restored mappings. Plugging that result into (15), the fitness for that result is 3.0054. Therefore, the PSO was set to have this fitness number as part of its stopping criteria, the other being a number of iterations limit which was never reached in testing. For each PSO configuration, 10 independent particle swarms were used to simulate 10 runs with different initializations. When the median of the best result from all 10 swarms was better than or equal to the previous fitness result, the number of iterations of each swarm to get to this point was recorded. The use of multiple swarms (or runs) ensures that the result was not simply due to some favourable or unfavourable condition with regards to the pseudorandom number generator.

The pseudorandom numbers for both initialization of the position of particles was set to have a uniform random distribution between 0 and 1. For velocity, pseudorandom numbers between -0.5 and 0.5 were used. The result of running this configuration with different particle counts is shown in Table 5.

Table 5: Number of Runs for Different Particle Counts for Original PSO

# of Particles	# of Iterations to Desired Result	# of Mapping Runs
10	21	210
50	18	900
100	18	1800
250	9	2250
500	9	4500
1000	10	10000

For each run, the iterations were stopped when the system reached the desired value fitness as noted above. The number of mapping system runs is the number of times the mapping component of the system was used. It is equivalent to # of iterations × # of particles. The number of mapping runs is significant because most of the calculations are performed in by the mapping function, to the point where the PSO velocity update calculations can be considered negligible.

As expected, a larger number of particles seemed to help bring down the number of iterations to reach the fitness criteria result of 3.0054, but the number of calculations favoured small numbers of particles. There were two possibilities as to why the small particle system found the result quickly: either this result is easy to find, or the larger swarms are less efficient at exploring the problem space. Both of these hypotheses were explored by looking internally at the particle swarm and the results. To check these hypotheses, the 10-particle PSO was run for many more iterations until it got to 400 iterations, the 50-particle PSO was run for 80 iterations, and the 100-particle PSO was run for 40 iterations. The plots for the results for each swarm vs. the iteration number is shown in Figure 23 to Figure 25.

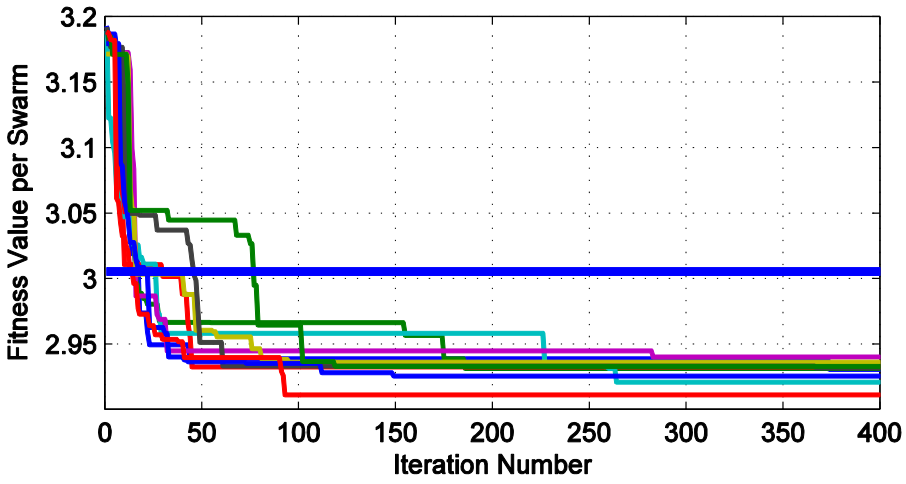


Figure 23: Result Per Iteration, 10 Particles Per Swarm × 10 Swarms

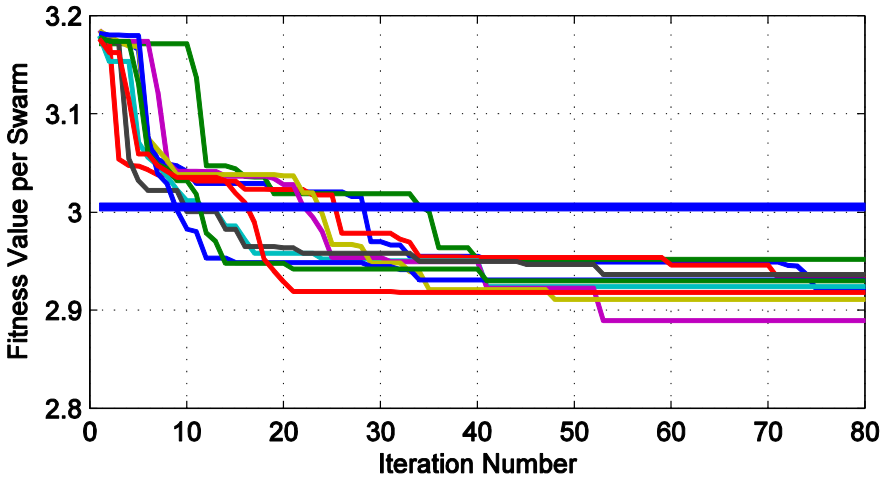


Figure 24: Result Per Iteration, 50 Particles Per Swarm × 10 Swarms

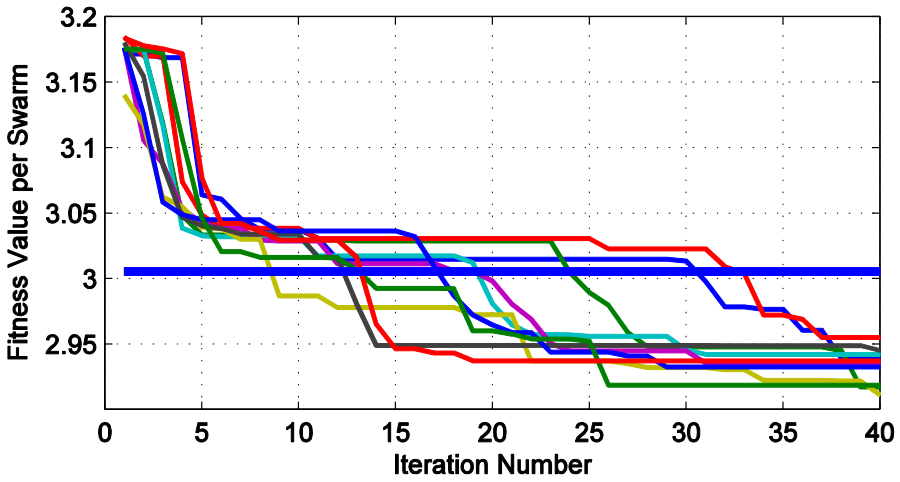


Figure 25: Result Per Iteration, 100 Particles Per Swarm × 10 Swarms

For these runs, the plot of the best position per swarm at each iteration is shown. Note that in these figures, lower results are better. The goal fitness of 3.0054 is shown with the horizontal blue line. There are large drops in the curves for the graphs because of the large influence given by the restored mappings and unique mappings. If there is a new restored or unique mapping, the fit_{lost} or $fit_{restored}$ component of the fitness function of the system changes significantly in steps that appear discrete. In a similar manner, these components do not change over many iterations, the curve looks flat because the other fitness component, fit_{map}, has much less influence over the overall fitness of the system. All of the swarm results after 400, 80, or 40 iterations settle with most of the results being below 2.95 per swarm. These results show that large swarms are not necessary to find better results and the result of 3.0054 from previous work with this system is relatively trivial for the system to reach. In addition, it takes more iterations to find even better results after dipping below a certain point.

To find out how the particles behave during the simulation when a good result is reached, the speeds of the particles and their distance from each other was observed. Since the problem cannot be graphed (due to dimensionality), a histogram of the velocity and the distances from each particle to the mean of their respective swarms (neighbourhoods) was created. Note the histograms show the particles from all swarms added together. This analysis is shown for the 10, 100, and 1000 particle runs at the

point when the particles reach the desired fitness as mentioned beforehand. The histograms are shown in Figure 26 to Figure 28.

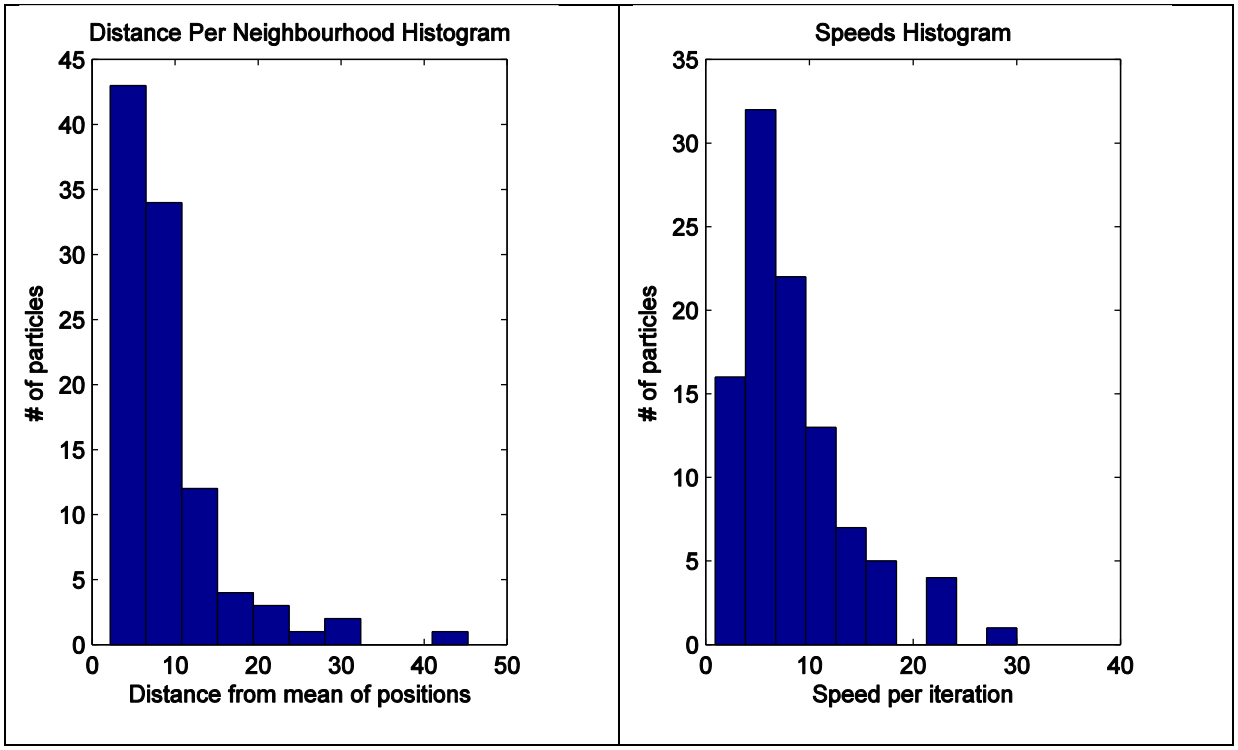


Figure 26: Distance and Speed for 10 Particle Swarms, Iteration #21

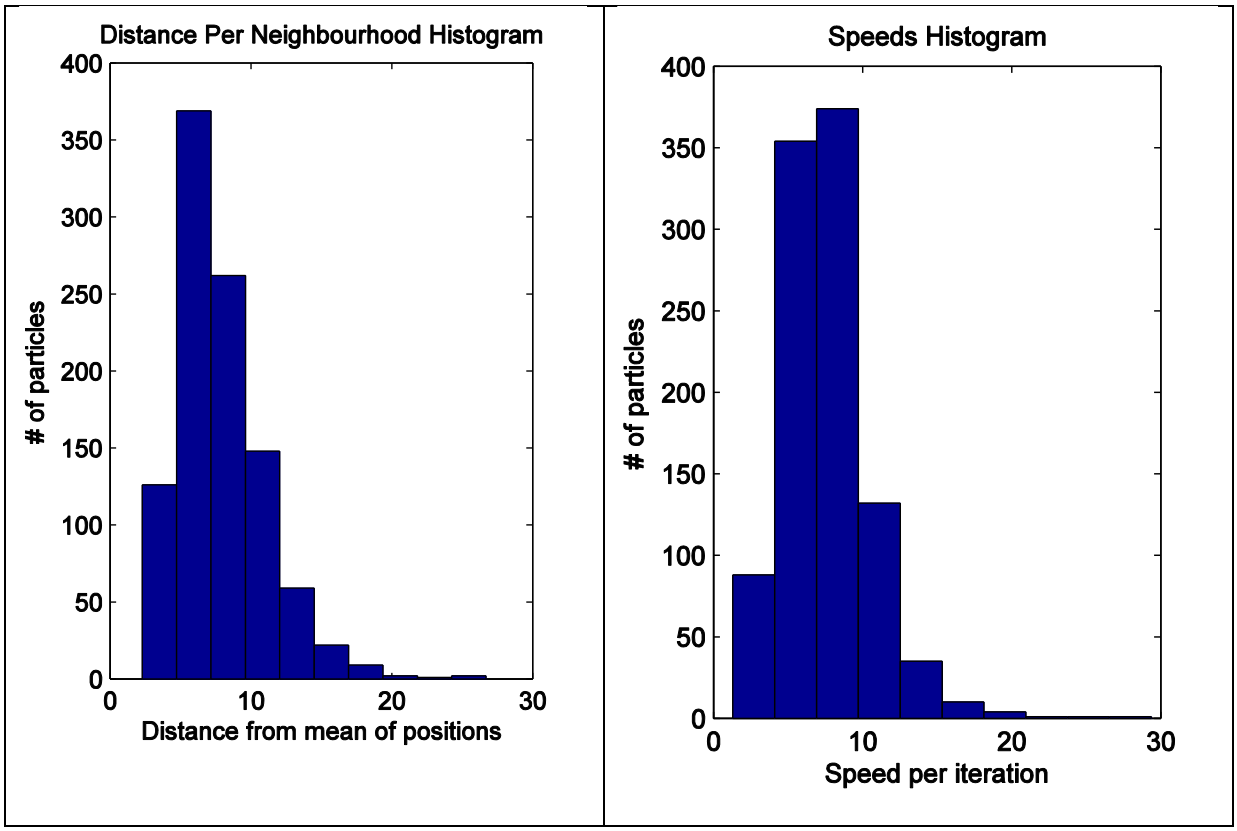


Figure 27: Distance and Speed for 100 Particle Swarms, Iteration #18

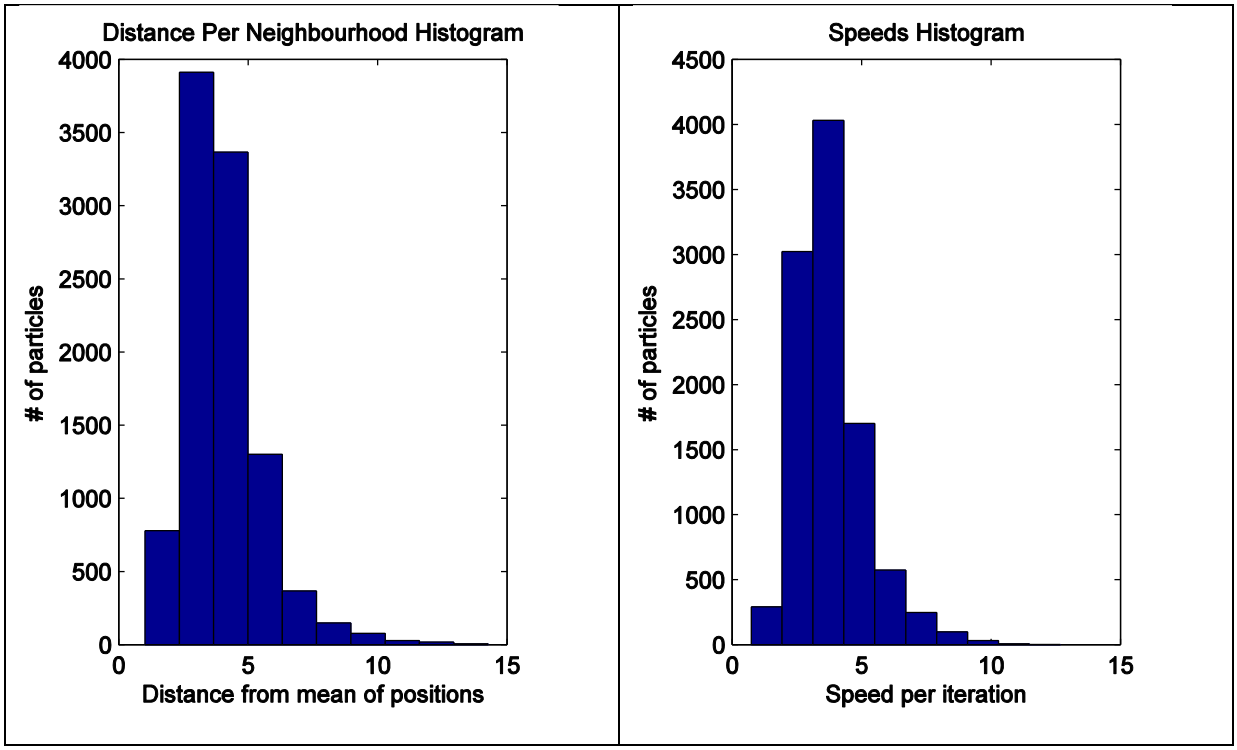


Figure 28: Distance and Speed for 1000 Particle Swarms, Iteration #10

In the high particle count tests, the histograms look relatively similar. In both cases, the particles appear to move at speeds of around 3 units per iteration, and the particles are spaced out at approximately 10 units apart. The lower particle count test, despite being at a higher iteration count, has outliers around 40 units away from the average position. Since the particles were initialized with a position between [0,1] for each dimension, particles travelled outside that region. The maximum distance would have otherwise been $\sqrt{32}$ (due to 32-dimensional particles) or around 5.66, which is smaller the distance between many of the particles. The histogram for the speeds shows that the particles are not settled when they reach the result for work, since most particles still have a speed of over 1 unit per iteration.

The 10-particle swarm was continued over more iterations to see if the particles ever come to rest. Figure 29 shows the histograms for the 10-particle test after 400 iterations.

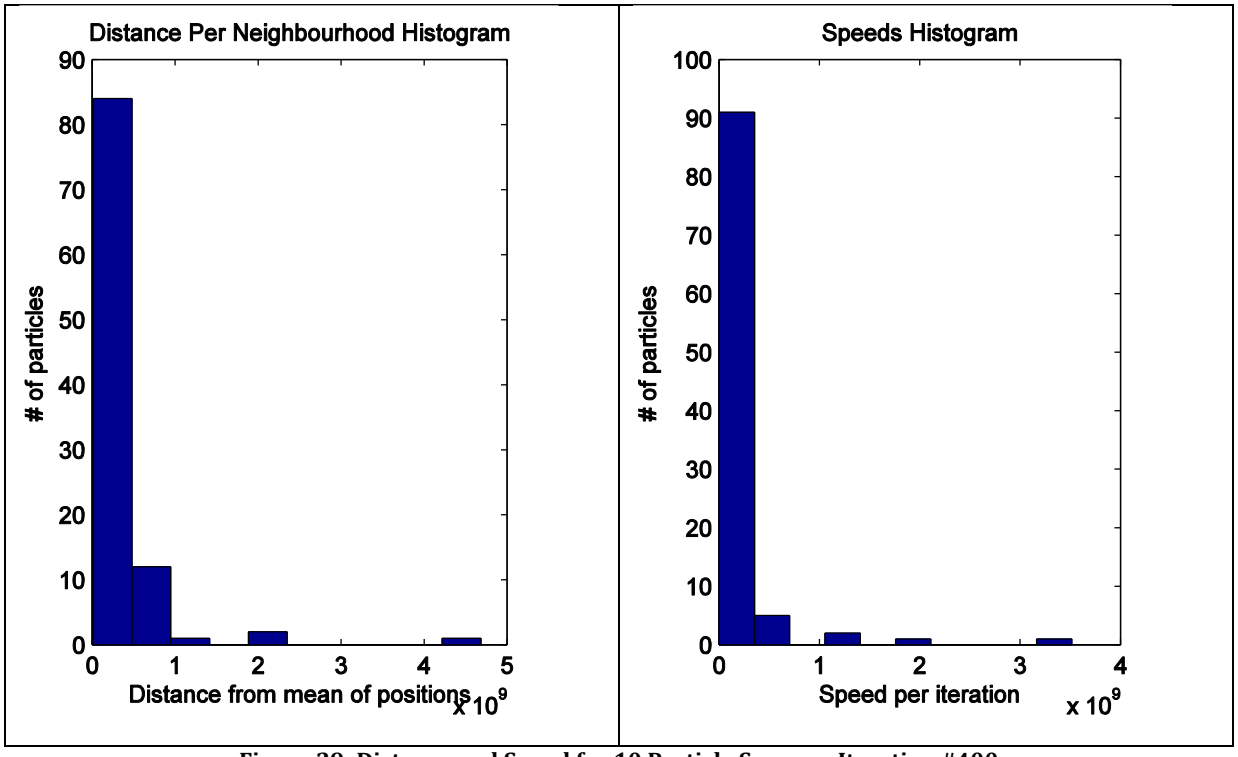


Figure 29: Distance and Speed for 10 Particle Swarms, Iteration #400

Note that the speed and distance in the histograms is counted in the billions of units, so a noteworthy amount of particles have left the original initialization space and have diverged. A reason for this is the problem space has repeating points, for example

due to the normalization in during the static analysis phase (in (6), explained in Section 3.7), having very large weights is the same as having a one weight of 1 and many smaller weights.

Given these results, it could be said that small particle swarms are better than larger ones for this problem (in terms of computation time to reach a reasonable result). Additionally, some particles diverge or continue at relatively high velocities because the PSO does not know that there are constraints to this problem. For these reasons, the same tests were performed with a constrained PSO to see if the results changed with the constraints.

4.2. Testing Using Constrained PSO

The test in Section 4.1 was repeated with the 10, 50, and 100 particle swarms with all parameters the same except for the use of the constrained PSO. Unlike the conventional PSO, the constrained PSO initialized and kept the particles within a set of boundaries for the entire run. The boundaries were set as the following:

- Weights: [-1, 1]
- Static membership analysis clustering radius: $[0, \sqrt{2^2 * 17}]$
- Dynamic membership analysis clustering radius: $[0, \sqrt{2^2 * 11 * N_v}]$
- *Where N_v is the number of vertices used for the target face in this region*

The weights were constrained to [-1, 1] since the static membership analysis scales these values with equation (6). The radii were constrained to be between 0 and the radius that would be required such that the hypersphere defining the space that is being clustered would be twice the size needed to encompass all vectors. This constraint allows the PSO to fit all of the results in the static or dynamic membership in one cluster. The other extreme is at a radius of 0, where each element would belong to its own cluster. For the static step's clustering, there are 17 clusters which give 17 dimensions, and due to the constraint between [-1 and 1], the dimensions for the particles can be up to 2 units apart. For the dynamic step's clustering, there are vertices up to 2 units apart along 11 dimensions, again due to the number of membership functions. The value is also multiplied by the number of vertices in the target face because of the summing across the vertices performed by the step immediately before

clustering, described in Section 3.8.1. These potential maximum distances are how the square root values above were selected. The initialization velocities were also fitted to these boundaries by allowing the velocity to be set between [-*b/2*, +*b/2*], where *b* is the boundary for that dimension. Note this means the speeds for the particles in the dimensions controlling the weights were the same as the original PSO at [-0.5, 0.5], but the speeds in the dimensions were larger.

The use of the new initialization and the larger velocities for the dimensions controlling the radii allowed for the PSO to settle on solutions faster. However, after many iterations, the results for both type of PSO were similar. Figure 30 compares the execution of the original and the constrained PSO for 10, 50, and 100 particles per swarm.

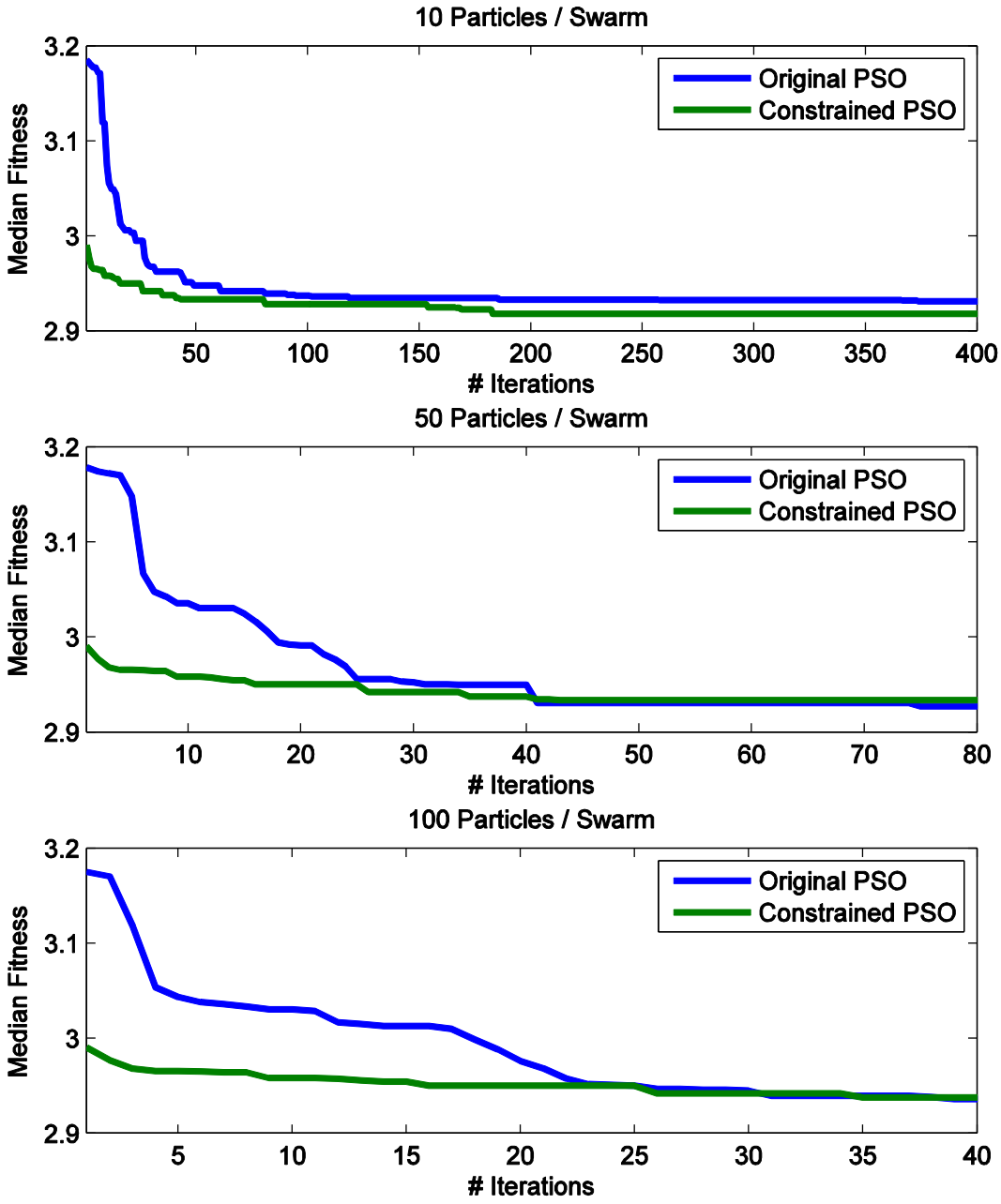


Figure 30: Median Fitness for Original vs Constrained PSO

The test of iterating until the result of manually setting weights was not useable because all of the constrained PSO runs initialized to be better than this result. Therefore, the stopping condition was set to be number of iterations. From these results, it can be seen that the constrained PSO reaches a good result faster for all sizes. To find out which swarm size would be best, each of the constrained PSOs were run for 8000 mapping runs. This is 800 iterations for 10 particles, 160 iterations for 50

particles, and 80 iterations for 100 particles per swarm. As in the other tests, 10 independent swarms were run and the median of the result was taken to keep note of a "typical" use of the PSO. The result is shown in Figure 31.

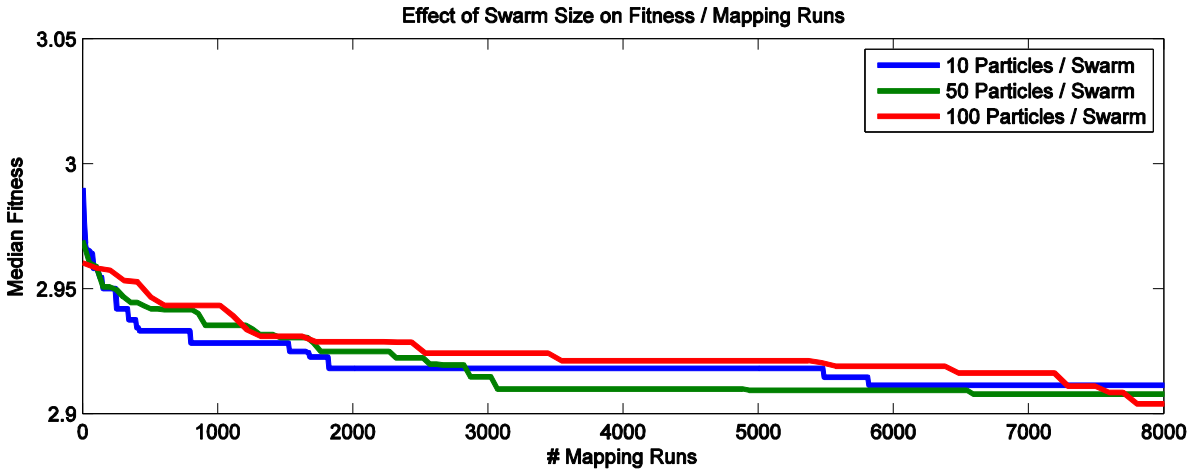


Figure 31: Comparison of Swarm Sizes for Constrained PSO

These results are similar to the tests on the original PSO, in that the 10 particle swarm settles faster than other sizes. This is because for the same number of mapping runs, the 10 particle swarm has had more iterations (and therefore more velocity updates) to find the solution. The larger swarms however over a longer number of mapping runs are able to settle on even better solutions. For practical reasons however, very long runs with very large swarms would result in diminishing returns.

As an example of the amount of time these long runs take, 8000 mapping runs on test computer took 7 hours, 40 minutes to complete. Additional tests were run with 250 iterations for the 100 and 250 particle swarms (taking 2.5 days to compute), but neither resulted in better fitness than what is shown in Figure 31. This time was recorded on a test computer with the following configuration:

- CPU: Intel Core i5 2400 with Turbo Boost enabled
- Memory: 8 GiB PC3-1333
- MATLAB r2010b set to perform single threaded execution

These results show that a constrained PSO can arrive at better fitness results than the original PSO over the short term, and has similar performance over the long term. Additionally, the results settle for many mapping runs over certain results until

they are able to "break into" a better fitness value due to finding more mappings or more restored mappings. The PSO in general can function as a quick optimization to find a meaningful mapping result, but there is room for improvement by looking at different particle swarm optimization variants or attempting to use other unsupervised learning techniques. This can be seen because the larger particle swarm over a very long run can find some even better results, so it is possible that additional variations in the unsupervised learning in this system can settle on these results with less computational time.

Given this exploration of different configurations of PSO for the mouth region, the mapping for other regions was performed using the 50 particle swarm over 100 iterations (5000 mapping runs), as this number of mapping runs was near the point where longer runs have diminishing returns.

4.3. Mapping Results for All Regions

The first tests to tune the PSO only used the mouth region, since the mouth is the most complex with the largest vertex count and the most animation parameters, for both faces. In addition, there were previous results with the mouth that can be compared. After the PSO configuration was set in the testing described in the previous section, that configuration was used for all of the following regions:

- Mouth
- Jaw
- Left eyebrow
- Right eyebrow

The eyes were not used because the muscle-based face cannot move its eyes, and therefore no result could be generated for those regions. These tests were performed using a 50 particle swarm over 100 iterations. As with the other tests, 10 independent swarms were run and the charts show the median fitness of the 10 swarms. The fitness vs. number of iterations curve for each face region is shown in Figure 32, below.

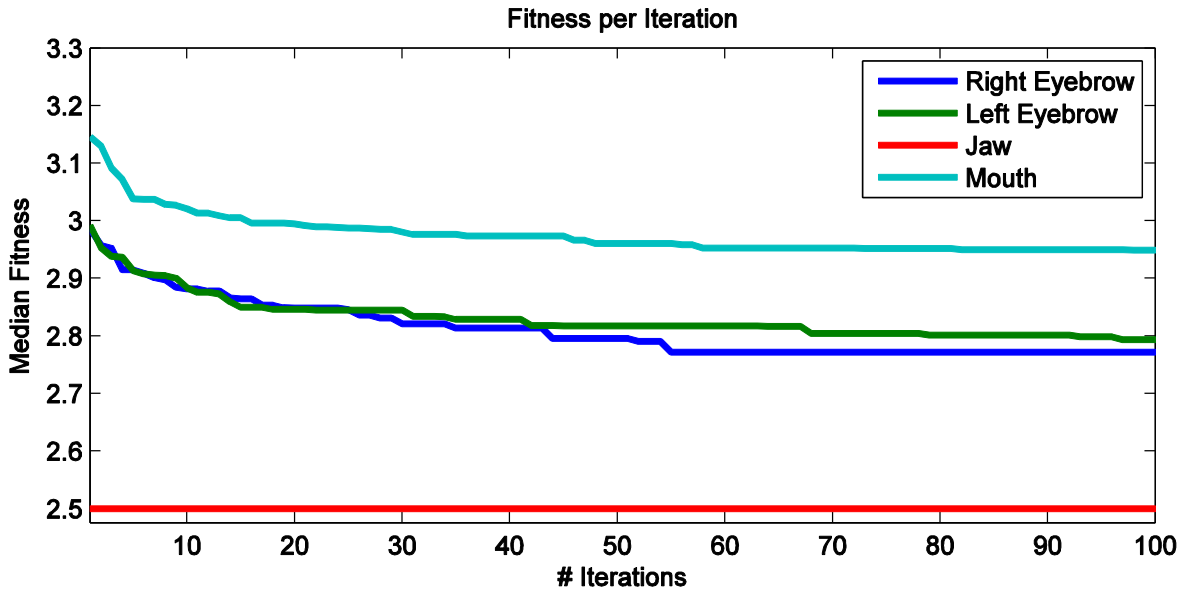


Figure 32: Fitness per Iteration for Face Regions

The fitness per iteration shows that the left and right eyebrows are still complex enough that the particle swarm takes time settling on good results, while the jaw is so simple that the particle swarm cannot find a result better than the initial iteration. The jaw is simple because the Candide3 and muscle-based faces both have only one parameter that affects the jaw region, and the movement is a "jaw lowerer", which is a simple downwards movement. The eyebrows and jaw used the first rejection criterion (angle error) discussed in Section 3.8.1 to find mapping pairs, while the mouth used the second criterion (worse than neutral). The choice of the second criterion was because the angle error criterion generated more visible artifacts for the mouth region than the first criterion. The change of criterion for this part of the test is also why the fitness result looks different than in Figure 31.

Select images of the mappings from the Candide3 face to the muscle-based face are shown below, for motions that both faces are capable of performing. These images are generated from the swarm with the best fitness result. The mapping errors as well as the mappings created for expressions the muscle-based face could not perform are described in the next section. The screenshots in Figure 33 show mapping results for

the mouth and jaw regions combined, while the screenshots in Figure 34 show mapping results for the left and right eyebrows. All other mappings are in Appendix 2.

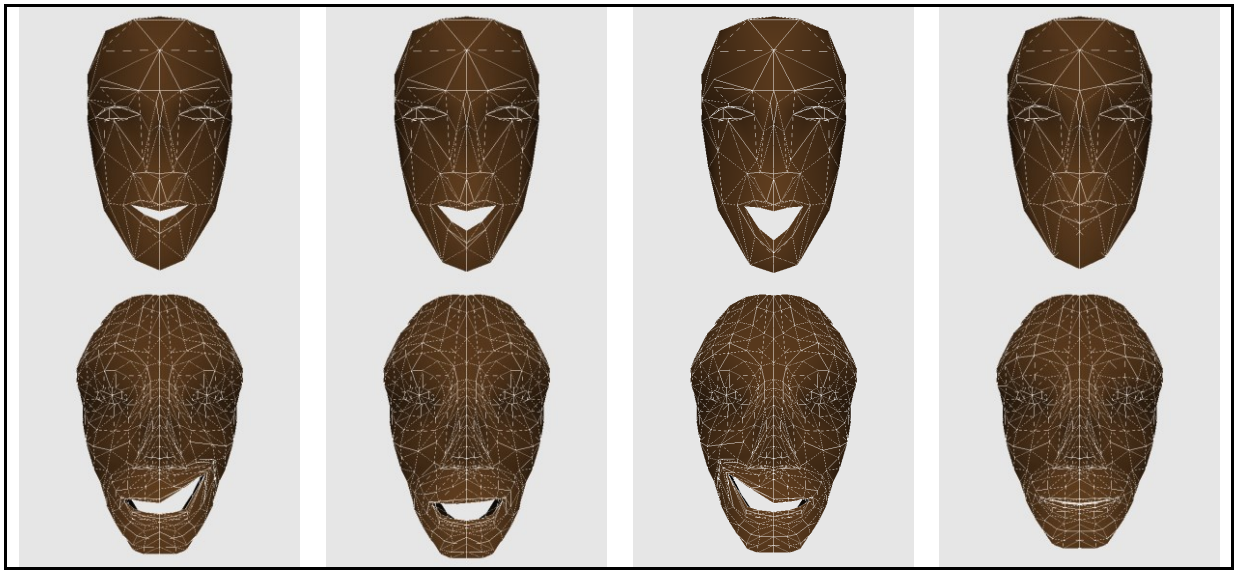


Figure 33: Mappings for Mouth and Jaw Regions

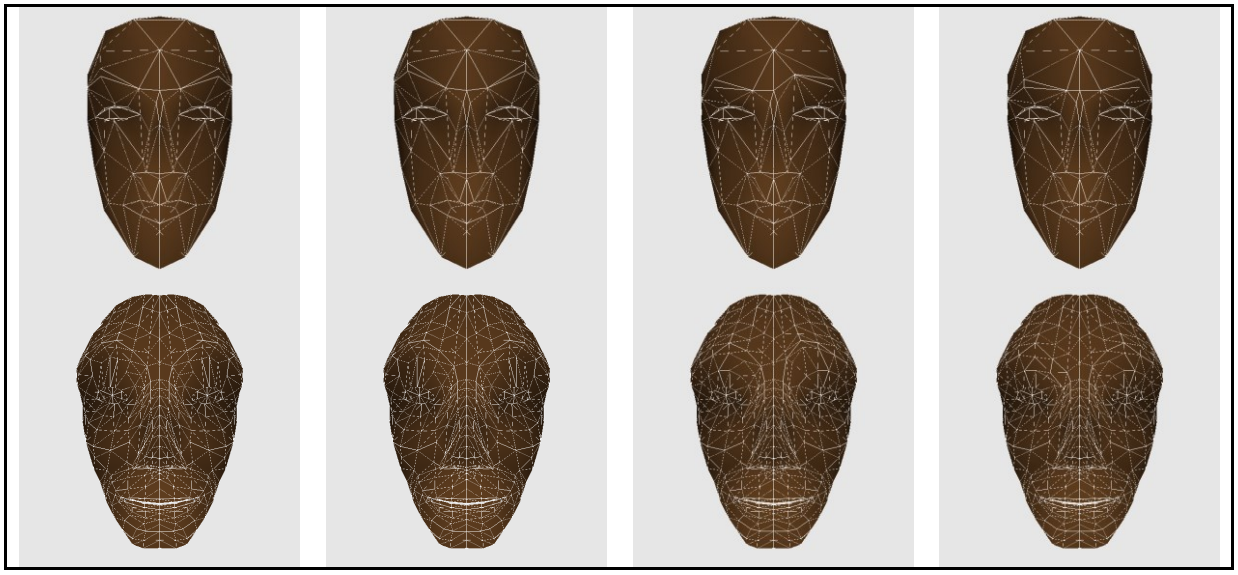


Figure 34: Mappings for Eyebrow Regions (Combined left and right eyebrow)

The fitness scores for each region in terms of the criteria defined in Section 3.9.2 are shown in Table 6. The standard deviation of error per vertex, scaled to the largest movement of each vertex, is also shown. The total number of target mappings inputted into the system is also shown in the column for the unique mappings, to evaluate how many of the target motions are being used in the mappings.

Table 6: Errors and Number of Mappings per Region

Region	Mean Error Per Vertex	Standard Deviation of Error	Perfectly Restored Mappings	Unique Mappings
Mouth	0.095	0.029	4	12 of 20
Jaw	0.625	N/A (all same)	3	3 of 4
Left Eyebrow	0.358	0.102	4	5 of 8
Right Eyebrow	0.423	0.088	4	5 of 8

These results show that the algorithm uses over half of the expressions of the target face to attempt to clone expressions, and the error is larger or smaller depending on the region of the face used. The sources of the error are discussed in the next section.

4.4. Discussion

The proposed system was able to work with a muscle-based face and with a small amount of manual steps compared to systems in literature. It was able to generate meaningful mappings using unordered feature points and with different numbers of feature points between the source and target faces. In addition, this system used a relatively small number of feature points, with 49 points on the Candide3 face. Although there were implementations that used a smaller number of points [65] [67], they required the points in a specific order or manually-matched pairs of facial expressions for reference. The algorithm was able to reproduce simple motions for each of the regions, but it had produced errors which show visible artifacts. These errors come from weaknesses of the algorithm and the differences between the faces.

One of the largest sources of error is from the way the dynamic membership analysis can only assign one target expression to one source expression for any given face region. It cannot blend target expressions to create new expressions using combinations of expressions within the same region. In the mouth region, artifacts are

generated because the Candide3 face moves both lip corners while the muscle-based face moves only one lip corner, resulting in mappings such as in Figure 35. For the eyebrow regions, similar artifacts occur because the Candide3 face moves both the inner and outer eyebrow while the muscle-based face only moves one part of the eyebrow at a time. This artifact is shown in Figure 36.

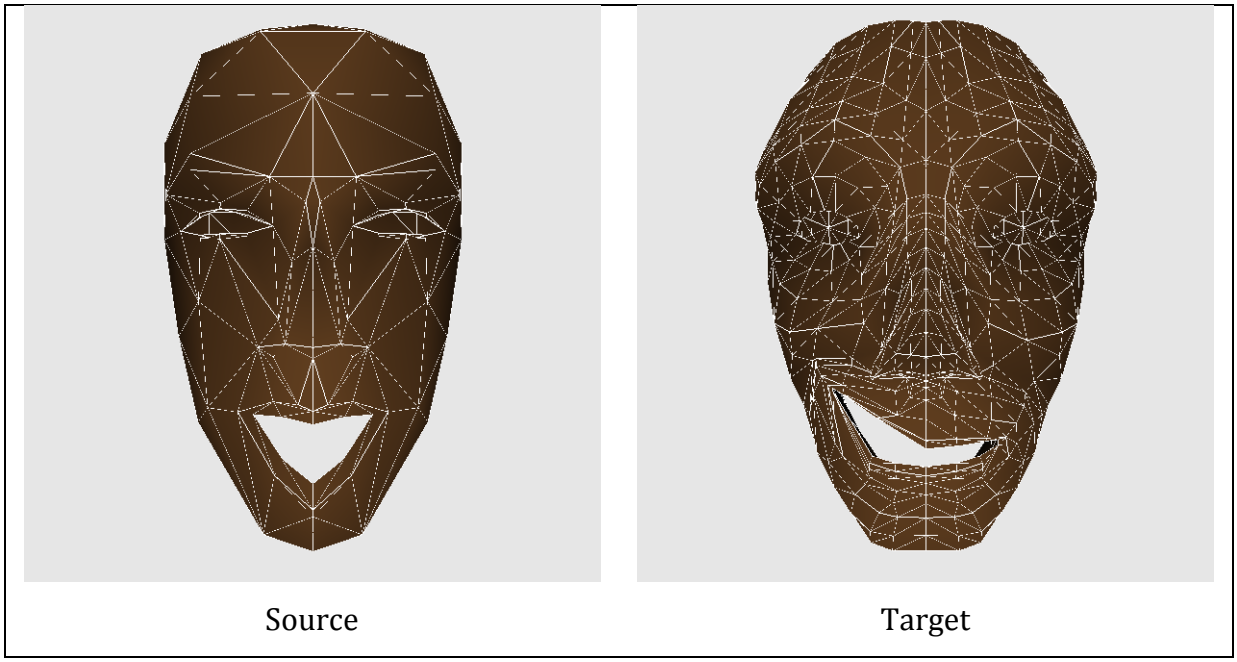


Figure 35: Uneven Lip Corners Artifact

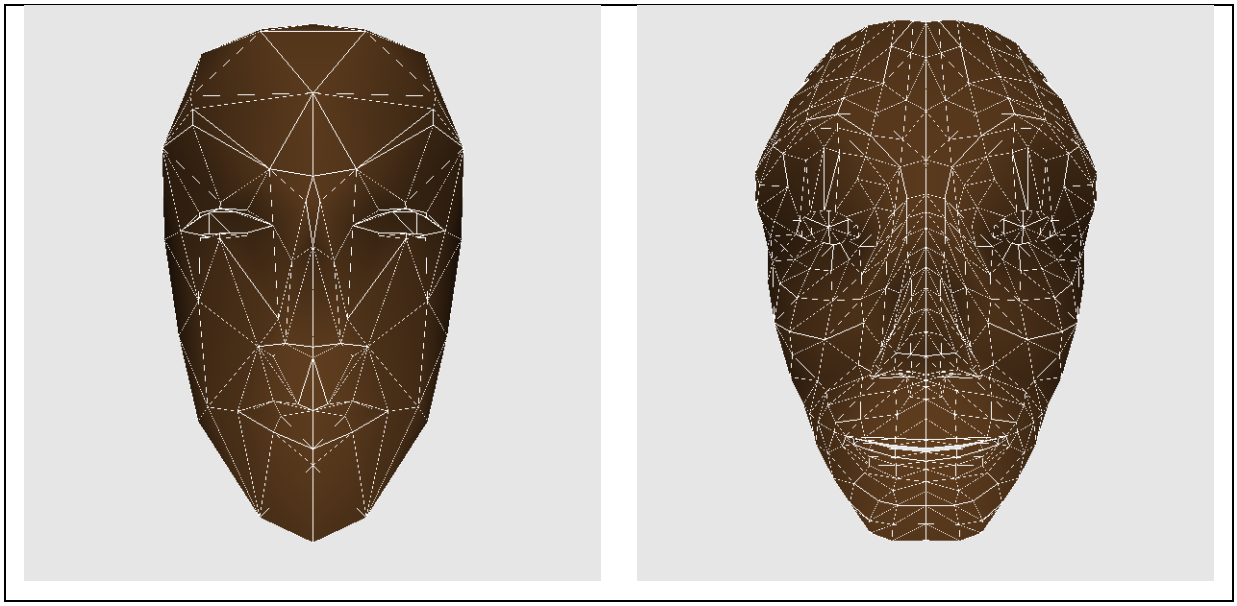


Figure 36: Uneven Eyebrow Raiser Artifact

This limitation of one-to-one mappings is there because the fitness criteria of perfectly restored mappings and unique mappings could not be implemented if the system always generated new expressions. New fitness criteria would be required because vertex error cannot be used on its own in this problem. In testing with vertex error alone as the sole criterion or as the dominant criterion, the PSO would settle in solutions where there were very few mapping pairs, because the error is zero for any solution with one or more perfectly reproduced expressions. In addition, one cannot force the system to create a mapping pair for every source expression, because some faces simply cannot reproduce expressions of other faces. For instance, the muscle-based face in this study cannot move its eyes at all. Another potential solution to this issue would be to divide the face up into more mouth regions in order that expressions for different regions do not conflict. However, this would require more steps for the user and partially defeat the ease-of-use requirement of this system.

Another source of error is how the algorithm may map completely different target expressions to different intensities of the same source expression. This is because for certain cases, the clustering performed in the dynamic analysis stage is insufficient to be able to group the same class of expressions together. The artifact produced is most easily seen on the mouth region, shown in Figure 37.

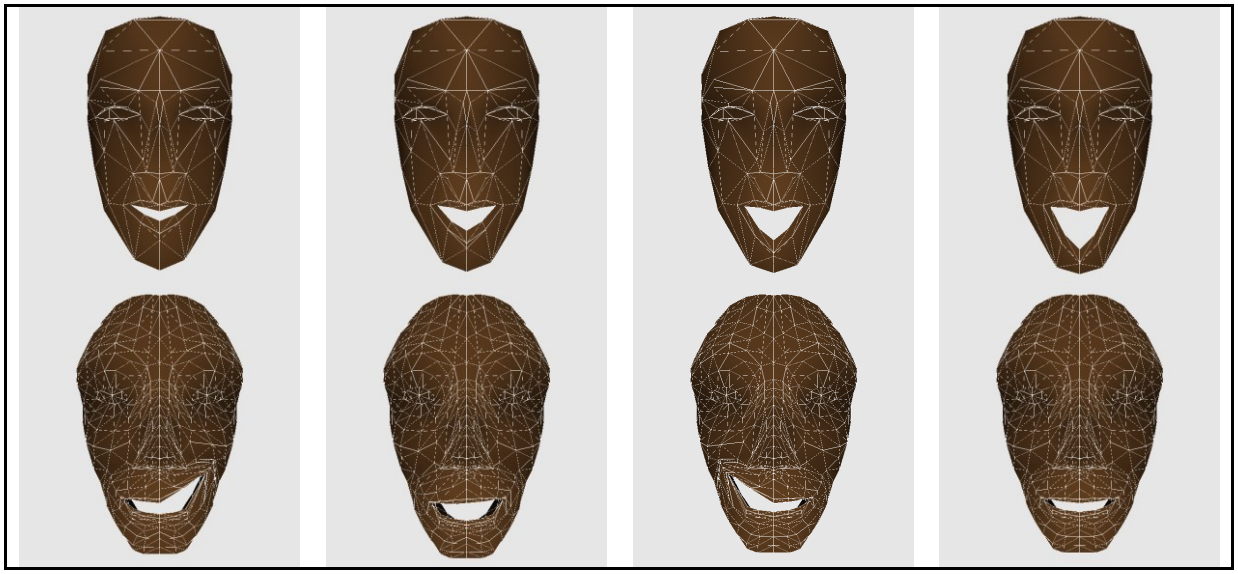


Figure 37: Artifact for Jaw Drop Lip Corners

In the above figure, four different intensities of the Candide3 jaw drop are mapped to a combination of jaw drop and lip corner puller on the muscle-based face. In addition, the lip corner puller is of a relatively high intensity (between 0.75 and 1), causing very visible artifacts. These artifacts are present because the Candide3's jaw drop shares some motions of the muscle-based face's lip corner puller. In addition, high intensity artifacts occur many times because of how the algorithm favours angle error over magnitude error in the clustering of expressions. Certain candidates for mapping have high magnitude error but a lower angle error. With the current implementation, despite creating a more visible artifact, these candidates are chosen due to that low angle error. There are several possibilities to lessen this error:

1. Experiment with different shapes of membership functions, allowing larger differences in angle error to be considered acceptable
2. Make a criterion for "velocity errors", where the error is considered over a sequence of source expression to target expression mappings, similar to [66]
3. Penalize solutions that introduce completely different parameters that what was already used for lower intensities of the source expressions

Of these possibilities, the first one would be the simplest, since it would not require changes to the algorithm. It has a smaller chance of working however, since it does not directly attempt to address the errors of added motions. The second and third possibilities require that expressions be grouped into sequences. Of these two, the second has been shown to help reduce this type of error in literature. The third possibility would affect the flexibility of the proposed system, because it assumes that an expression requires the same parameters to be activated no matter what intensity. However, that assumption cannot be guaranteed for models such as the AAM, because the relationship between parameters and expressions can be arbitrarily formed in training.

Chapter 5. Conclusion

This thesis presented an overview of the state of the art in facial expression cloning, which found a need for FEC systems that work with multiple types of face models. As an attempt to fill this need, it proposed components of a facial expression cloning method to work with parameterized faces.

Chapter 1 discussed problems that could be alleviated with FEC that operate on different face implementations. Different standards, cultures, and views on psychology of facial expressions result in many variations for facial expression animation systems to work with. The design of faces are influenced by their creators, and the degree to which a face imitates human proportions and shapes depends on the creators' stance on the Uncanny Valley problem. For humanlike faces, the influence may be the choice of subjects the appearance of the face is inspired or scanned from. For nonhuman or cartoon-like faces, there are very wide variations because of the artistic freedom the creators have for such a face. Facial expressions also have many variations because of the different notations that can be used to describe a facial expression, and because facial expressions are context and culture-dependent. Because of the many possibilities for faces and facial expressions, FEC systems can reduce the manual labour required to create expressions for multiple faces, and they can also be part of a larger system that learns human facial expressions.

Chapter 2 summarized the implementations of faces and FEC systems. It defined facial expressions as temporary deformations created in human faces by deformation of tissues on the face. They convey much of the information in a conversation between humans. There have been many face models created. One of the main works that led to parameterized faces was the FACS system, a tool created by psychologists to quantify facial expressions, and concepts from FACS were adapted to animation to produce facial expressions on virtual and robot faces. The virtual faces that describe their movements with parameters can be classified as AAMs, physically-based models, muscle-based models, and shape-based models. AAMs are used in many FEC systems because it is a system that can generate parameters from labelled images and video of humans, so

training of AAM-based systems actually creates a face model that can be used to transfer expressions to other similar models. Physically-based models are computationally intensive but they provide insight into the forces involved in moving parts of the face, while muscle-based models are a simplification which only emulates the final deformation for a physically-based model. Shape-based models are the simplest models for virtual faces, and are very common in animation due to this simplicity. They define a base mesh and deformations. There are many FEC systems in literature, with applications in animation and privacy. The prior systems used all of the models except for the physically-based face. FEC implementations for robots currently require more manual steps than most FEC that maps to virtual avatar faces.

Compared to prior arts in literature, the proposed system was shown to require a relatively small number of manual steps, and it was able to function with no knowledge of the underlying implementation of the faces. It was tested by mapping between a shape-based face and a muscle-based face that have few similarities. The algorithm was able to map basic movements between these faces, although it did produce visible anomalies in the mapping when dealing with expressions that one of the two faces did not support with a single action. This project also demonstrated that particle swarm optimization is effective for optimizing facial expression cloning, as the automatic optimization surpassed results from manual experimentation on the parameters of this system in [76].

5.1. Future Work

To be able to better satisfy the objective of mapping expressions between any face model including robot faces, future works include testing the algorithm on more faces and modifications to the algorithm to increase its accuracy and allow it to work on a wider set of facial expressions.

One of the future works is to validate this FEC system on multiple robots and be able to quantitatively state the mapping error to the robot. To add a robot to this system, a subsystem must be created to record the robot's expressions. To measure the error of cloning to robots, a virtual model and a robot with the same movements will be

created, and mapping between these two models will be performed to measure the error of the system that records expressions. The sources of this error are variations that come from the robot's actuators and the system used to measure the movements. This system will eventually be extended with another training stage to allow the robot to learn new expressions from its human users.

There are also future works possible for the mapping algorithm to make it more accurate and handle more expressions. The biggest improvement would be to allow the dynamic membership analysis to combine target expressions together in the same region. To work with a many-to-many mapping, new fitness criteria must be created for the PSO, since there could be infinitely many possibilities for mappings. The current criteria assumes a discrete space for mapping, which would no longer be the case if the system is allowed to generate new target expressions. In addition, many of the other steps of this system can be varied in different situations, like using different shapes for the membership functions. To prevent large visible artifacts where unrelated target expressions appear to be added for different source expressions, a "velocity" or first derivative of the error across multiple intensities of the same expressions can be attempted. Modifications on the selection criteria of the dynamic analysis stage would also be useful to attempt to eliminate cases where angle error is small but magnitude error is large, producing visible artifacts. More experiments with the unsupervised learning stage could also lead to better results, so good results can be found with less runs of the algorithm. Future works would be to attempt to use other variations of constrained PSO or other optimization techniques such as neural networks and genetic algorithms. Another improvement to the flexibility of the system would be to add another machine learning algorithm that would allow new expression pairs to be added while the system is operating. Having such a system adjust itself with no downtime would make it both more efficient and more natural as a human-computer interface.

References

[1] J. Tao and T. Tan, "Affective Computing: A Review," *Affective Computing and Intelligent Interaction,* vol. 3784, no. 1, pp. 981-995 , 2005.

[2] C. Breazeal, "Role of expressive behaviour for robots that learn from people," *Royal Society of London. Series B, Biological sciences, Philosophical transactions of,* vol. 364, no. 1535, pp. 3527-3538, 2009.

[3] M. Mori, "The Uncanny Valley," *Energy,* vol. 7, no. 4, pp. 33-35, 1970.

[4] K. F. MacDorman, R. D. Green, C.-C. Ho and C. T. Koch, "Too real for comfort? Uncanny responses to computer generated faces," *Including the Special Issue: Enabling elderly users to create and share self authored multimedia content,* vol. 25, no. 3, pp. 695-710, 2009.

[5] F. Hegel, M. Lohse and B. Wrede, "Effects of visual appearance on the attribution of applications in social robotics," in *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, Toyama, 2009.

[6] P. Ekman, "Universals and cultural differences in facial expressions of emotion," *Nebraska Symposium on Motivation,* vol. 19, pp. 207-282, 1971.

[7] P. Ekman and E. L. Rosenberg, What the face reveals : basic and applied studies of spontaneous expression using the facial action coding system (FACS), New York: Oxford University Press, 2005.

[8] K. Isbister, "The Face," in *Better Game Characters by Design*, San Francisco, Morgan Kaufmann, 2006, pp. 143-159.

[9] Z. Zeng, M. Pantic, G. I. Roisman and T. S. Huang, "A Survey of Affect Recognition Methods: Audio, Visual, and Spontaneous Expressions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 31, no. 1, pp. 39-58, 2009.

[10] A. Khanam and M. Mufti, "Intelligent Expression Blending for Performance Driven Facial Animation," *Consumer Electronics, IEEE Transactions on,* vol. 53, no. 4, pp. 578-584, 2007.

[11] J. Song, B. Choi, Y. Seol and J. Noh, "Characteristic facial retargeting," *Computer Animation and Virtual Worlds,* vol. 22, no. 2-3, pp. 187-194, 2011.

[12] A. A. Root and J. A. Stephens, "Organization of the central control of muscles of facial expression in man," *The Journal of physiology,* vol. 549, no. 1, pp. 289-298, 2003.

[13] D. Terzopoulos and K. Waters, "Physically-based facial modelling, analysis, and animation," *Visualization and Computer Animation, The Journal of,* vol. 1, no. 2, pp. 73-80, 1990.

[14] D. Kim and J. Sung, "Roles of the Face in Human Communication," in *Automated Face Analysis*, New York, IGI Global, 2009, p. 1.

[15] P. Ekman and H. Oster, "Facial Expressions of Emotion," *Annual Review of Psychology,* vol. 30, no. 1, pp. 527-554, 1979.

[16] B. Fasel and J. Luettin, "Automatic facial expression analysis: a survey," *Pattern Recognition,* vol. 36, no. 1, pp. 259-275, 2003.

[17] P. Black, S. Porter, A. Baker and N. Korva, "Uncovering the Secrets of the Human Face: The Role of the Face in Pro-social and Forensic Contexts," in *Facial Expressions: Dynamic Patterns, Impairments, and Social Perceptions*, New York, Nova Publishers, 2012, pp. 41-66.

[18] T. Hashimoto, H. Kobayashi and N. Kato, "Educational system with the android robot SAYA and field trial," in *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, Taipei, 2011.

[19] K. Goris, J. Saldien, B. Vanderborght and D. Lefeber, "Probo, an Intelligent Huggable Robot for HRI Studies with Children," in *Human-Robot Interaction*, Croatia, INTECH, 2010, pp. 33-42.

[20] T. Fukuda, M. Nakashima, F. Arai and Y. Hasegawa, "Facial expressive robotic head system for human-robot communication and its application in home environment," *IEEE, Proceedings of,* vol. 92, no. 11, pp. 1851-1865, 2004.

[21] K. Berns and J. Hirth, "Control of facial expressions of the humanoid robot head ROMAN," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Bejing, 2006.

[22] A. Mehrabian, "Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in Temperament," *Current Psychology,* vol. 14, no. 4, pp. 261-292, 1996.

[23] A. Mehrabian, "Inferring Emotions," in *Silent Messages*, Belmont, Wadsworth Publishing Co., 1981, pp. 43-55.

[24] J. B. Bavelas and N. Chovil, "Faces in dialogue," in *The Psychology of Facial Expression*, Cambridge, Cambridge University Press, 1997, pp. 334-346.

[25] T. Tojo, Y. Matsusaka, T. Ishii and T. Kobayashi, "A conversational robot utilizing facial and body expressions," in *Systems, Man and Cybernetics. 'Cybernetics Evolving*

to Systems, Humans, Organizations, and their Complex Interactions, 2000 IEEE International Conference on, Nashville, 2000.

[26] R. Kaliouby and P. Robinson, "Real-Time Inference of Complex Mental States from Facial Expressions and Head Gestures," in *Real-Time Vision for Human-Computer Interaction*, Heidelberg, Springer, 2005, pp. 181-200.

[27] N. Ersotelos and F. Dong, "Building highly realistic facial modeling and animation: a survey," *The Visual Computer,* vol. 24, no. 1, pp. 13-30, 2008.

[28] Y.-l. Tian, T. Kanade and J. F. Cohn, "Evaluation of Gabor-wavelet-based facial action unit recognition in image sequences of increasing complexity," in *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, Washington DC, 2002.

[29] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar and I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, San Francisco, 2011.

[30] I. S. Pandzic and R. Forchheimer, "The Origins of the MPEG-4 Facial Animation Standard," in *MPEG-4 Facial Animation: The Standard, Implementation and Applications*, Chichester, John Wiley & Sons, Ltd, 2002, pp. 17-56.

[31] J. Ahlberg, "CANDIDE-3 - An Updated Parameterised Face," 2001. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.5603. [Accessed 27 3 2013].

[32] Valve Corporation, "Source," 2007. [Online]. Available: http://source.valvesoftware.com/. [Accessed 1 July 2013].

[33] T. F. Cootes, C. J. Taylor, D. H. Cooper and J. Graham, "Active Shape Models-Their Training and Application," *Computer Vision and Image Understanding,* vol. 61, no. 1, pp. 38-59, 1995.

[34] D. Kim and J. Sung, "Face Modeling," in *Automated Face Analysis*, New York, IGI Global, 2009, pp. 45-88.

[35] J. Sung, T. Kanade and D. Kim, "A Unified Gradient-Based Approach for Combining ASM into AAM," *Computer Vision, International Journal of,* vol. 75, no. 2, pp. 297-309, 2007.

[36] T. Cootes, G. Edwards and C. Taylor, "Active appearance models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 23, no. 6, pp. 681-685, 2001.

[37] M. D. Cordea, "A 3D anthropometric muscle-based active appearance model for model-based video coding," Ottawa-Carleton Institute for Electrical and Computer Engineering, Ottawa, 2008.

[38] N. Stoiber, R. Seguier and G. Breton, "Facial animation retargeting and control based on a human appearance space," *Computer Animation and Virtual Worlds,* vol. 21, no. 1, pp. 39-54, 2010.

[39] D. Terzopoulos and K. Waters, "Analysis and synthesis of facial image sequences using physical and anatomical models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on ,* vol. 15, no. 6, pp. 569-579, 1993.

[40] S. Morishima, S. Hajime and D. Terzopoulos, "A Physics-based Talking Head for Interface Agent," 2007. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7186. [Accessed 15 February 2013].

[41] L. Li, Y. Liu and H. Zhang, "A Survey of Computer Facial Animation Techniques," in *Computer Science and Electronics Engineering (ICCSEE), 2012 International Conference on*, Hangzhou, China, 2012.

[42] A. Tekalp and J. Ostermann, "The Origins of the MPEG-4 Facial Animation Standard," in *MPEG-4 Facial Animation: The Standard, Implementation and Applications*, Hoboken, Wiley Blackwell, 2002, pp. 17-56.

[43] J. Ostermann, "Face Animation in MPEG-4," in *MPEG-4 Facial Animation: The Standard, Implementation and Applications*, Hoboken, Wiley Blackwell, 2002, pp. 17-56.

[44] H. Song, Y.-M. Kim, J.-C. Park, C. H. Kim and D.-S. Kwon, "Design of a robot head for emotional expression: EEEX," in *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, Munich, 2008.

[45] B. Bickel, M. Gross, P. Kaufmann, M. Skouras, B. Thomaszewski, D. Bradley, T. Beeler, P. Jackson, S. Marschner and W. Matusik, "Physical face cloning," *Graphics, ACM Transactions on,* vol. 31, no. 4, pp. 1-10, 2012.

[46] H. Zhang and D. Liu, "Fuzzy Sets," in *Fuzzy Modeling and Fuzzy Control*, Boston, Birkhauser, 2006, pp. 4-10.

[47] O. Castillo and P. Melin, "Type-2 Fuzzy Logic," in *Type-2 Fuzzy Logic: Theory and Applications*, Berlin, Springer-Verlag, 2008, pp. 5-28.

[48] A. Halder, P. Rakshit, S. Chakraborty, A. Konar, A. Chakraborty, E. Kim and A. K. Nagar, "Reducing uncertainty in interval type-2 fuzzy sets for qualitative improvement in emotion recognition from facial expressions," in *Fuzzy Systems*

(FUZZ-IEEE), 2012 IEEE International Conference on, Brisbane, 2012.

[49] Y. Zhao, X. Wang, M. Goubran, T. Whaylen and E. M. Petriu, "Human emotion and cognition recognition from body language of the head using soft computing techniques," *Ambient Intelligence and Humanized Computing, Journal of* , vol. 4, no. 1, pp. 121-140, 2013.

[50] J.-y. Noh and U. Neumann, "Expression cloning," in *Conference on Computer graphics and interactive techniques, 28th Annual Proceedings of*, New York, 2001.

[51] L. Williams, "Performance-driven facial animation," *ACM SIGGRAPH Computer Graphics,* vol. 24, no. 4, pp. 235-242, 1990.

[52] J. Noh and U. Neumann, "Facial Animation by Expression Cloning," in *Data-Driven 3D Facial Animation*, London, Springer London, 2007, pp. 187-216.

[53] M. de la Hunty, A. Asthana and R. Goecke, "Linear Facial Expression Transfer with Active Appearance Models," in *Pattern Recognition (ICPR), 20th International Conference on*, Istanbul, 2010.

[54] Microsoft Corporation, "Face Tracking," 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj130970.aspx. [Accessed 11 February 2012].

[55] I. Matthews and S. Baker, "Active appearance models revisited," *Computer Vision, International Journal of,* vol. 60, no. 2, pp. 135-164, 2004.

[56] J. Jia, S. Zhang and L. Cai, "Facial expression synthesis based on motion patterns learned from face database," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, Hong Kong, 2010.

[57] L. Lei and Z. Miao, "Individual facial expression transferring using Active Appearance Model and FAP in MPEG-4," in *Image Analysis and Signal Processing (IASP), 2010 International Conference on*, Xiamen, 2010.

[58] N. Kholgade, I. Matthews and Y. Sheikh, "Content retargeting using parameter-parallel facial layers," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 2011 Proceedings of*, New York, 2011.

[59] J.-B. Kim, Y. Hwang, W.-C. Bang, H. Lee, J. D. K. Kim and C. Kim, "Real-time realistic 3D facial expression cloning for smart TV," in *Consumer Electronics (ICCE), 2013 IEEE International Conference on*, Las Vegas, 2013.

[60] T. Sucontphunt, Z. Mo, U. Neumann and Z. Deng, "Interactive 3D facial expression posing through 2D portrait manipulation," in *Graphics interface, 2008 Proceedings of* , Toronto, 2008.

[61] T. Weise, H. Li, L. Van Gool and M. Pauly, "Face/Off," in *ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '09, Proceedings of*, New York, 2009.

[62] Y. Lin, M. Song, D. T. P. Quynh, Y. He and C. Chen, "Sparse Coding for Flexible, Robust 3D Facial-Expression Synthesis," *Computer Graphics and Applications, IEEE,* vol. 32, no. 2, pp. 76-88, 2012.

[63] Microsoft Corporation, "Face Tracking," 2013. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj130970.aspx. [Accessed 5 June 2013].

[64] D. Huang and F. Torre, "Facial Action Transfer with Personalized Bilinear Regression," in *Computer Vision – ECCV 2012 SE - 11*, Florence, 2012.

[65] L. Dutreve, A. Meyer and S. Bouakaz, "Feature points based facial animation retargeting," in *2008 ACM symposium on Virtual reality software and technology, Proceedings of*, New York, 2008.

[66] Y. Seol, J. P. Lewis, J. Seo, B. Choi, K. Anjyo and J. Noh, "Spacetime expression cloning for blendshapes," *ACM Trans. Graph.,* vol. 31, no. 2, pp. 14:1-14:12, 2012.

[67] M. Gotoh, M. Kanoh, S. Kato and H. Itoh, "A Neural-Based Approach to Facial Expression Mapping Between Human and Robot," in *Knowledge-Based Intelligent Information and Engineering Systems*, Berlin, Springer Berlin Heidelberg, 2007, pp. 194-201.

[68] P. Jaeckel, N. Campbell and C. Melhuish, "Facial behaviour mapping—From video footage to a robot head," *Towards Autonomous Robotic Systems 2008: Mobile Robotics in the UK 10th British Conference on Mobile Robotics - Towards Autonomous Robotic Systems (TAROS 2007),* vol. 56, no. 12, pp. 1042-1049, 2008.

[69] M. Tscherepanow, M. Hillebrand, F. Hegel, B. Wrede and F. Kummert, "Direct imitation of human facial expressions by a user-interface robot," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, Paris, 2009.

[70] R. B. Queiroz, A. Braun, J. L. Moreira, M. Cohen, S. R. Musse, M. R. Thielo and R. Samadani, "Reflecting user faces in avatars," in *10th international conference on Intelligent virtual agents, Proceedings of*, Berlin, 2010.

[71] Z. Zhang, "Microsoft Kinect Sensor and Its Effect," *IEEE Multimedia,* vol. 19, no. 2, pp. 4-10, 2012.

[72] The MathWorks, Inc., "subclust," 2013. [Online]. Available: http://www.mathworks.com/help/fuzzy/subclust.html. [Accessed 3 July 2013].

[73] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman and A. Wu, "An efficient k-means clustering algorithm: analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 24, no. 7, pp. 881-892, 2002.

[74] R. Kruse, C. Borgelt, F. Klawonn, C. Moewes, M. Steinbrecher and P. Held, "Introduction to Evolutionary Algorithms," in *Computational Intelligence*, London, Springer-Verlag, 2013, pp. 167-195.

[75] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, Perth, 1995.

[76] P. Santos and E. Petriu, "Facial Expression Cloning with Fuzzy Set Clustering," in *Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA 2013), 2013 IEEE International Conference on*, Milan, 2013.

[77] M. Setayesh, M. Zhang and M. Johnston, "Detection of continuous, smooth and thin edges in noisy images using constrained particle swarm optimisation," in *13th annual conference on Genetic and evolutionary computation, Proceedings of*, New York, 2011.

[78] M. Daneshyari and G. G. Yen, "Constrained Multiple-Swarm Particle Swarm Optimization Within a Cultural Framework," *Systems, Man, and Cybernetics, IEEE Transactions on,* vol. 42, no. 2, pp. 475-490, 2012.

[79] R. Poli, J. Kennedy and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence,* vol. 1, no. 1, pp. 33-57, 2007.

[80] N. Gebhardt, "Irrlicht Engine - A free open source 3D engine," June 2013. [Online]. Available: http://irrlicht.sourceforge.net/. [Accessed 2 August 2013].

Appendix 1. Software Design

This appendix elaborates on the design of the software implemented to construct and test this algorithm. The function of the software is described in Chapter 3. The proposed system is divided into several components, written in Visual C++ 10 and MATLAB r2010b. These components are separated by their responsibilities, which are representation and processing of face animations, mapping of face animations, and user interface. The software is split into modules according the responsibility of each module. Each module was designed to minimize the changes required if the external libraries that the program depends on are changed. The modules designed and implemented for this project are listed below:

- Main Program and User Interface (femapirr)
- Face Representation and Animation Library (femaplib)
- Mapping Component

The main program module is programmed in C++, and it contains the main executable for the software and various user interface components. It is dubbed "femapirr" because of its use of the Irrlicht engine [80]. The Face Representation and Animation Library, called femaplib, contains all the data structures and algorithms to represent and animate faces. The class diagram for the femaplib module is shown on Figure A. 1. The Mapping Component contains the bulk of the mapping algorithm, which is programmed in MATLAB. This component is run offline, unlike the other modules which run in real-time, and its inputs and outputs are stored in files which are produced or consumed by the other modules. Each of the modules has a single purpose, and there is no duplication of functionality between modules. There was duplication in simple data structures to prevent dependency on external libraries.

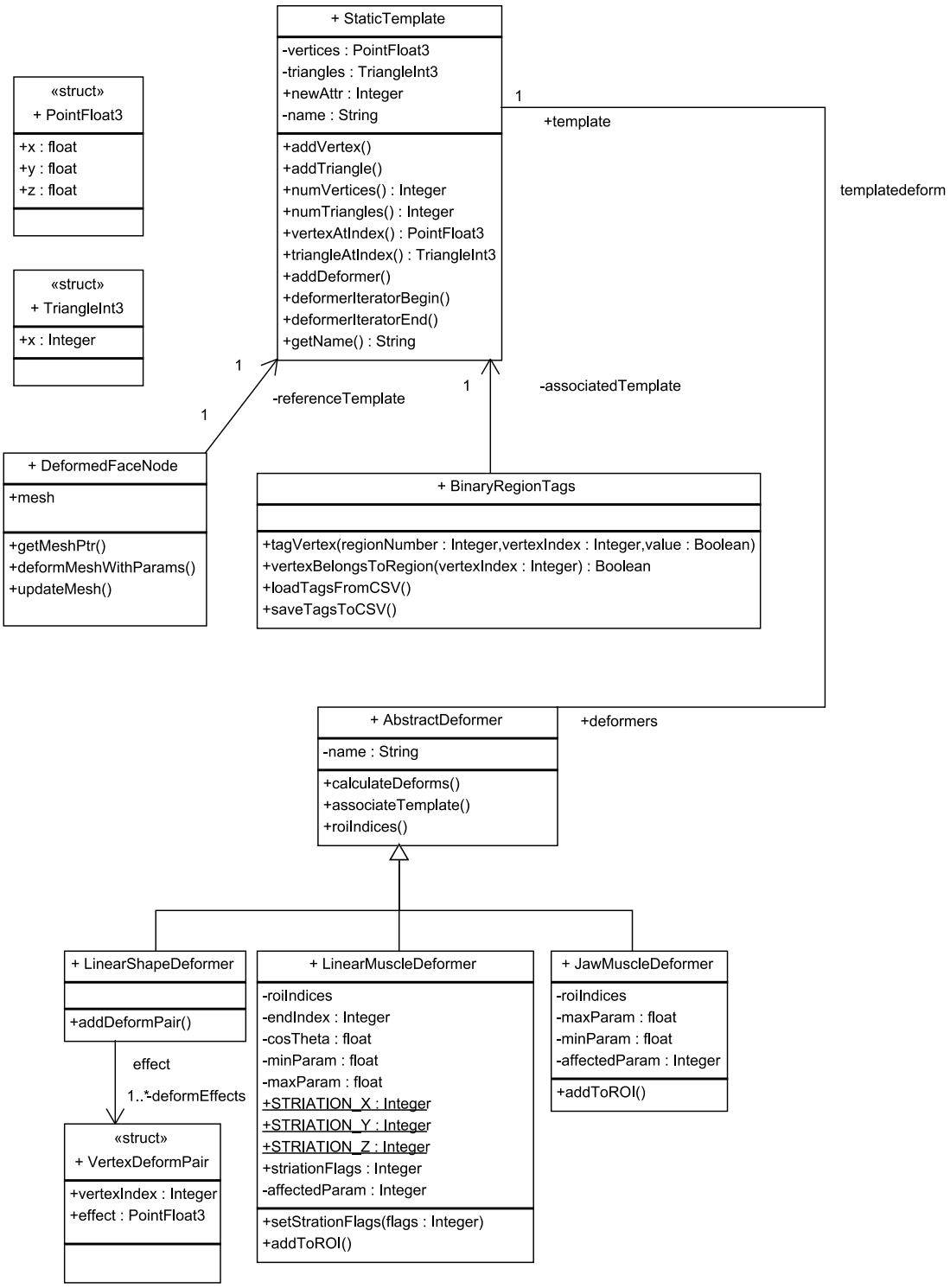


Figure A. 1: Class Diagram for Face Representation Module

The femapirr library imports the Irrlicht library and uses it to display faces and provide a user interface. It is divided into several components: face display, mapping pair display, region labelling, expression extraction, and generation of screenshots. The face display and mapping pair display components show the face and allow the user to browse expression pairs generated by the MATLAB components. The region labelling component provides an interface to experiment with expressions and label face regions. The expression extraction generates the input to the MATLAB components. The screenshot generation automatically takes screenshots of the face regions with expressions, and it is used to generate the screenshots of faces throughout this thesis.

The femaplib library represents and animates the faces in this project. It stores the face meshes and the definitions of any deformations to the face, and it calculates expressions. To work with multiple types of faces, it uses a class hierarchy such that the meshes and deformers are separated. Refer to Figure A. 1 for this structure. The StaticTemplate class stores information about the neutral face and refers to any number of deformers, represented by a collection of *AbstractDeformer* instances. The muscle-based face uses LinearMuscleDeformer and JawMuscleDeformer, while the Candide3 face uses LinearShapeDeformer. Since all of these deformers are subclasses of *AbstractDeformer*, this structure would work for any combination of deformers as long as they can be defined by some set of parameters. The DeformedFaceNode class contains a collection of deformations based on those defined in a StaticTemplate. An application can use many DeformedFaceNode instances per StaticTemplate if more than one expression of the same face needs to be stored or displayed. To reduce coupling, the DeformedFaceNode class is the only class in femaplib that uses the Irrlicht library. Therefore, if the 3D library was replaced, for instance with another 3D library or a control library for a robot, only this class would have to be modified. The BinaryRegionTags class keeps track of the labels generated by the region labelling component, and allows these labels to be saved.

The mapping component is implemented as a set of MATLAB functions. These functions are divided the same way as the workflow of the system, shown in sections 3.6 to 3.9, inclusive. The mapping components are divided into functions that define the

fuzzy memberships, static analysis, and dynamic analysis. The PSO is implemented in two functions: a PSO initialize function that sets the initial positions and velocities, and a PSO run function that runs the swarm until a stopping condition is met. The PSO is divided into two functions which save all their internal variables at each iteration, so the user can resume an interrupted PSO training run or add more iterations.

The implementation of all internal code in this system was made with future expansion in mind. Since this software is an initial proof-of-concept with no older prototype to follow up upon, these rules were used in the development of the software:

1. Multiple external libraries should not be used perform the same task
2. As much as possible, only one internal module should be linked to an external library.
3. Before importing an external library, try to use internal C++ or Visual C++ libraries that have the same functionality.
4. Prioritize predictability and testability over performance.

The first and second rules with external libraries were created to minimize the labour required if an external library must be replaced. The third rule is to minimize the number of external libraries in the system, because adding an external library makes the system dependent on the development of that library. The use of Visual C++ specific functionality was allowed for the project since future works plan to make use of technologies that favour Visual Studio-based applications, such as the Kinect for Windows sensor. As an example, multithreaded algorithms and user interface threads use the Microsoft Asynchronous Agents library. Performance in this system was given the lowest priority because the purpose of the system was a proof-of-concept, and the test faces were simple enough that real-time performance can be achieved without optimizing for performance.

Appendix 2. Candide3 to Muscle-based Face Mappings

This appendix shows all of the face mappings generated by this system. Each column in each figure represents an expression pair. The top row contains the original face mappings and the bottom row shows the mappings for the muscle-based face.