Head Movement and Facial Expression Transfer from 2D Video to a 3D Model

Mairead Grogan

A dissertation submitted to the University of Dublin, Trinity College,

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Interactive Entertainment Technology)

University of Dublin, Trinity College

2013

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university.

Mairead Grogan

August 30, 2013

Permission to Lend and/or Copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Mairead Grogan

August 30, 2013

Acknowledgments

Thanks to Dr. Rozenn Dahyot for the encouragement to work on this particular topic and for her help throughout the project as my Project Supervisor.

Thanks to the excellent lecturers we had teaching us throughout the M.Sc. in Interactive Entertainment Technology, for their insightful and encouraging courses and for many helpful discussions over the year.

Mairead Grogan

Head Movement and Facial Expression Transfer from 2D Video to a 3D Model

Mairead Grogan
University of Dublin, Trinity College, 2013

Supervisor: Dr. Rozenn Dahyot

The realistic transfer of facial animation from a face in a 2D video to a 3D model has proved to be a persistent challenge in recent years. In this thesis we propose a method which transfers both the head movement and facial expressions of a person in a 2D video to a 3D model. Our method uses corresponding feature points on both the 3D model and 2D face to calculate the rigid head pose of the person in the video. This position is given by a rotation and a linear transformation. Once these parameters are calculated, they are used to transform the 3D model so that the head is in the correct position. The next step is to transfer the expression of the 2D face onto the rotated 3D model. The facial expression is represented by the displacement of the current set of feature points in the 2D video relative to a set of reference feature points. We assume that these facial expression feature points only move in 2 dimensions. We

then use the displacement vectors to transform the corresponding feature points on the 3D model. These newly transformed feature points are used to drive a Laplacian deformation which computes the positions of the remaining non-feature points on the 3D model. The final results constructed using this algorithm are both accurate and realistic. This method improves upon previous research in the area which assumed that the head remained static throughout the video. The simple acquisition system of the input video data ensures that this technique could be implemented in applications where motion capture or 3D scanning systems are unavailable. Our method can be used in a wide range of applications including character animation for 3D films, online avatars, expression exaggeration and reconstructing a video sequence from multiple viewpoints.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Non rigid pattern detection, recognition and synthesis are major topics of interest in computer vision and graphics applications. In particular, the face is a part of the body which is of great importance in scientific fields such as biometry, psychology, human-computer interaction, animation, biomechanics, and film and game industries. The computer games industry alone, worth $ 67 billion in 2012, dedicates a huge amount of resources to implement state of the art facial capture and reconstruction techniques to enhance the realism of the characters they create. In the film industry, many actors performances have been captured and used to animate different characters.

Facial dynamics are also significantly important in verbal communication, and many researchers have spent time processing lip movements and poses in order to decipher speech content and perform lip reading. Emotions and gestural messages can also be interpreted and recreated using facial data. One such application is the use of computer generated facial expressions to help teach autistic people how to read emotions. Recent advances in technology have seen engineers create a virtual talking head which is capable of replicating human emotions with unprecedented realism. Thus the construction of a highly accurate facial animation system is hugely important in both science and technology, although significant challenges still exist as no perfect facial animation system has yet been created. This has ensured that the face remains a hugely popular research topic, with new techniques constantly being developed to improve how it is modelled.

Facial animation has proved to be very challenging for two main reasons. The first

is that the face is not a rigid structure, but a flexible surface. This means that it is very difficult to specify how the face moves. Secondly, we are extremely familiar with the human face and what motions and expressions are natural for a face. This means that we notice small deviations from what we think are natural face dynamics and this has resulted in the 'uncanny valley' hypothesis. This holds when the replications of a human or human face look and move almost exactly like humans, but not quite. This results in feelings of revulsion among human observers. This is a state which researchers have been trying to overcome in the last few years.

A huge challenge with expression transfer is the difficulty associated with recreating realistic expressions on the target 3D model. The aim is to ensure that the 3D model maintains its own facial characteristics while still reflecting the expression and speech animation seen in the 2D video. The accuracy of the expressions which are transferred is another challenge. A simple smile can appear very different depending on the individual characteristics of the person. This means that a generic smile alone should not be transferred to the 3D model, but instead the unique smile of the person in the video. In addition, the recreated animation should be smooth.

However, the transfer of the facial expression alone is not sufficient. The movement of the head is also key when trying to convey the emotions of the person in the video. A simple nod or tilt of the head can communicate as much as the expression itself. Because of this we present a method which reconstructs both the rigid head movement and facial expression of the person in the video using a 3D model. We implement a feature point based technique in order to calculate the movement of the head and present a Laplacian deformation approach to recreate the expression on the model.

The computation is carried out using Matlab R2012b. The 3D model used during this project is the Basel Face model. This is a Morphable Model which was constructed from registered 3D scans of 100 male and 100 female faces. Details about the model and it's construction are available in [27]. The model mesh consists of 53,490 vertices which are connected by 160,470 triangles. Two lists of feature point positions are also provided with the model. The first is a subset of the Farkas points and the second is a subset of the MPEG4 FDP points. We use these lists to find the feature point positions needed when calculating the rigid head motion and facial expression of the person in the video.

The video used in this project was the Talking Face Video by Timothy Cootes [10].

This video consists of 5000 frames which display a person engaged in a conversation. The camera was static and trained on the individual. Throughout the video, although the person is moving their head, the movement still remains almost entirely within the image. A set of 68 feature points were chosen and tracked throughout the video. The tracking was performed semi-automatically and was checked visually in order to ensure that the positions were generally sufficiently accurate. Figure 1.1 displays a frame from the video with the 68 feature points labelled. The positions of these feature points are provided with the video.

Figure 1.1: This image shows the 68 feature points which were tracked throughout the Talking Face Video.

The rest of this thesis is organised as follows. Chapter 2 discusses the state of the art research in facial animation and expression transfer. Chapter 3 describes and compares three methods which were implemented to recreate the head pose of the person in the video. Chapter 4 discusses how the movement of the feature points on the face were calculated and used to transfer the facial expression to the 3D model. Laplacian deformation is then described and the final results are presented. Chapter 5 discusses the limitations of our algorithm and future work that could be carried out.

Chapter 2

Literature review

2.1 Facial Animation

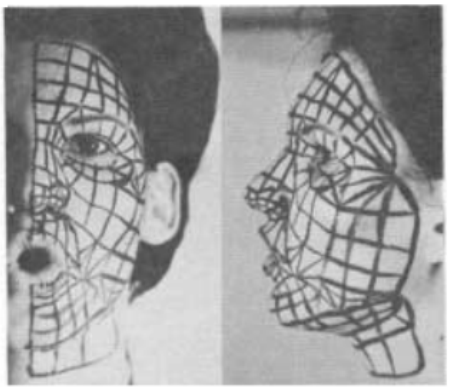

Figure 2.1: A pair of data photographs taken of an actor's face in [26].

Facial animation has been an important topic in computer science since the 1970's, when Frederick Parke first published his theses entitled "Computer Generated Animation of Faces"[26], and "A Parametric Model for Human Faces"[25]. In [26], Parke outlined a simple way to represent and animate a face with a sequence of expressions. The surface of the face was approximated using a skin of polygons, with a higher polygon density in areas of higher curvature such as the nose and chin. In order to create the polygonal skin, polygons were drawn onto one half of an assistant's face. They then completed a set of expressions in order to see how well the set of polygons approximated these expressions. The skin was then determined by recording the vertices

of each of the polygons. In order to animate the face, two photographs were taken of the assistant's face doing different expressions, one from the side and one from the front. In order to animate changing expressions, a computer program was developed which interpolates between two expressions.

Since this paper was published, the area of facial animation has become an important research topic across a range of disciplines. The topic can be categorised into the following areas:

1. Creating facial models.
2. Modelling static facial expressions.
3. Creating dynamic facial expressions.
4. Rendering.

We will review the methods being used to model static facial expressions and create dynamic ones, focusing on those methods which are closely related to our project.

2.2 Modelling static facial expressions

Many methods have been developed in order to model static facial expressions in a realistic manner. These involve mapping a set of parameters to the appearance of the face model. There are two tasks associated with this problem:

1. Developing a suitable parameter set, and
2. Mapping the numerical values of the parameters to the observable changes in the appearance of the facial model.

There are many parameters which can be taken into consideration when trying to describe the appearance of the face. These can range from the underlying structure of the face to it's physical and biomechanical properties. Using physically based methods to model the face is an active research area, and aims to animate a human head by mimicking it's natural movements. In [36] Waters uses physically based methods to simulate facial muscle contractions and create several different facial expressions.

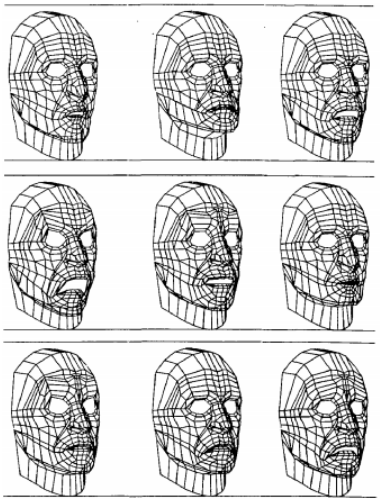

Figure 2.2: These are 9 different expressions which were created in [36] by moving 7 different muscles in the face.

Waters presents a muscle model which simulates two types of muscles - linear muscles which pull and sphincter muscles which squeeze (Figure 2.2). Ten muscles were added to the facial topology and used to create the expressions. However, as very few muscles were used, only very basic expressions could be reproduced (Figure 2.3).

Figure 2.3: This image shows the emotion of surprise recreated on the face model by Waters in [36]. Here the brows are curved and high, and the eyes are wide.

An extension of muscle-based parametrisation is physics based approaches. These models include separate layers for muscle and skin. The skin is deformed with virtual muscles that are attached to the mesh which represents the skin. In [22] Lee et al. develop an anatomically accurate physics-based model which simulates tissue, skull and synthetic muscles. A person's face is first scanned and used to create a polygonal

mesh which forms the epidermal layer and an algorithm then constructs the multi-layered skin and skull structure. Muscles are also inserted which create forces which deform the tissue and create realistic expressions. Constraints are included which allow the tissue to slide over the skull without penetrating it. All of these features allow for realistic animation and ensure that the facial model will be similar to any specific individual (Figure 2.4).

Figure 2.4: These images show a variety of emotions reconstructed by Lee et al. in [22]. The final image also has the epidermal mesh superimposed on the model.

In [31], Sifakis et al. also create an anatomically correct model using laser and MRI scans of a male subject. They use a motion capture system and a set of surface landmarks on the face to track facial movements. They create a system which automatically determines the muscle activation which tracks the surface landmarks. A three dimensional non linear finite element method is used to deform the tissue around the skull and creates a visually plausible animation.

However, simulating facial animations using biomechanical methods is still an active research area as our limited knowledge of the human skin, muscle and bone structure makes it difficult to build these models correctly.

2.3 Modelling dynamic facial expressions - Performance Capture

Another method of facial animation is performance driven animation, which typically consists of a non-rigid tracking stage followed by an expression retargeting procedure. In all of these systems there is a fundamental tradeoff between the quality of the acquired data and the complexity of the acquisition setup. Some systems have been designed to capture with great accuracy which can create stunning animations, typi-

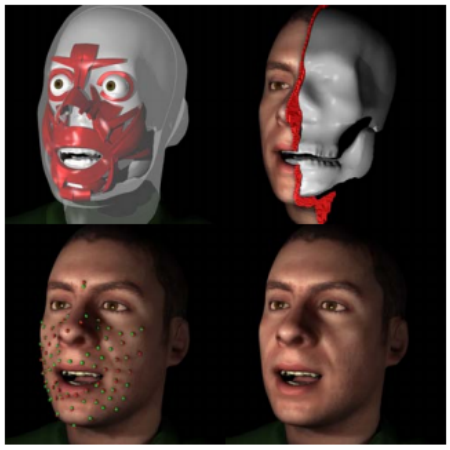

Figure 2.5: A facial expression created by Sifakis et al. using 32 facial muscles (top left) and simulated on a finite element tetrahedral mesh (top right). The muscle activations and bone kinematics are estimated to match motion capture data (bottom left). The final result expression can be seen on the bottom right.

cally used in the film industry. Marker based techniques are usually used for realtime facial animation because of their robustness. The motion parameters produced create convincing animations, even when retargeted onto non-human creatures. The more advanced acquisition systems make use of special equipment such as facial markers [38, 9, 19], camera arrays [6, 4], and structured light projectors [42, 37].

2.3.1 Marker based approaches

In [38] Williams et al. demonstrated one of the first marker based performance driven systems. Markers were placed on the performer's face and were tracked in 2D. The resulting movements were then projected onto the facial model which was animated in the cylindrical coordinates of the range data and then converted into 3D meshes with normals. The result was quite realistic, although it was still very basic and the model could not have an open mouth or eyes (Figure 2.6).

In [16], Guenter et al. implemented a similar technique. However, unlike Williams, who tracked in 2D, Guenter et al. tracked in 3D. Again, unlike Williams, who used a single static texture image of the person's face, Guenter et al. merged the video streams from multiple cameras in order to create a single texture map. This was then applied to the 3D facial model. The texture map sequence captured simultaneously with the 3D deformation data captured many details of expression that would be difficult to

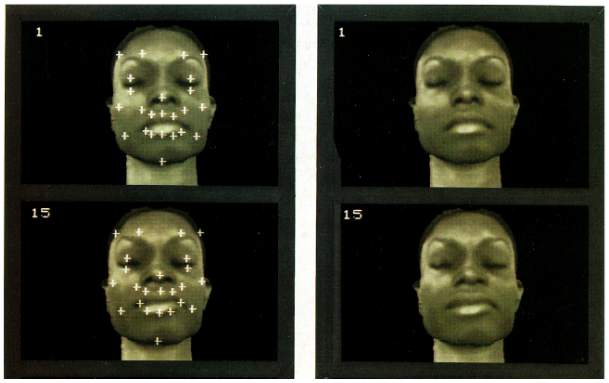

Figure 2.6: Results from [38]. These images show the tracked expression on the actors face to the left, and the reconstructed expression on the model to the right.

capture any other way.

2.3.2 Markerless approaches

For a more realistic representation of the face, markerless approaches such as realtime 3D scanners can capture fine-scale dynamics and are therefore more advantageous. However these require controlled studio environments and highly specialised sensors. In [42], Zhang et al. proposed a technique to reconstruct a high resolution face in real time using structured light techniques (Figure 2.7). The system projects a colour

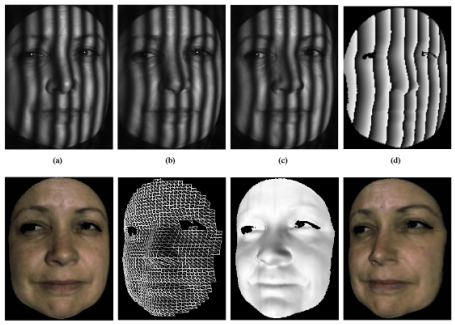

Figure 2.7: Result of the sinusoidal phase-shifting method for shape acquisition presented by Zhang et al.[42] (a)-(c) Phase-shifted fringe images. (d) Wrapped phase map. (e) 2D color image for texture mapping. (f) 3D wire frame model. (g) 3D shaded model. (h) 3D model with color texture mapping.

pattern whose RGB channels are coded with either sinusoidal or trapezoidal fringe patterns. This results in three grayscale patterns being projected sequentially onto the

object. A high speed camera is used to capture these three images and from them the face can be reconstructed.

Beeler et al. [4] present a method to construct the geometry of a face in a single shot under normal lighting conditions. Pore-scale geometry can be captured and the results are similar to those acquired using state of the art equipment. In order to demonstrate the robustness of their technique, models were reconstructed from captures of faces of varying gender, age and ethnicity with complex facial expressions. However, this research does not capture the dynamics of the face, and is solely concerned with creating a high resolution facial model.

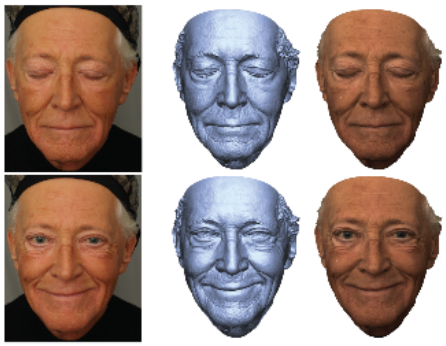

Figure 2.8: This shows a high resolution face model created using the technique described by Beeler et al. in [4].

In order to capture high resolution facial motion, non rigid registration and tracking algorithms are used across sequences of input geometry, texture or both. However, most of these systems are designed to focus on precision and not to achieve interactive performance in general environments. An example of this type of system is presented by Zhang et al. in [41]. The acquisition system used in this paper is made up of synchronised video cameras and structured light projectors which are used to capture images from several viewpoints. They then implement a technique which generates high resolution, dynamically controllable face models from these input videos. They also present tools which enable a user to create new animations and model the movements in the input video sequence (Figure 2.9).

On the other end of the scale are systems which use only one camera to capture and model facial dynamics. These systems commonly use 2D parametric shape models for non-rigid tracking. These systems also require that facial landmarks such as the corners of the eyes and mouth are accurately tracked. Optical flow is typically applied

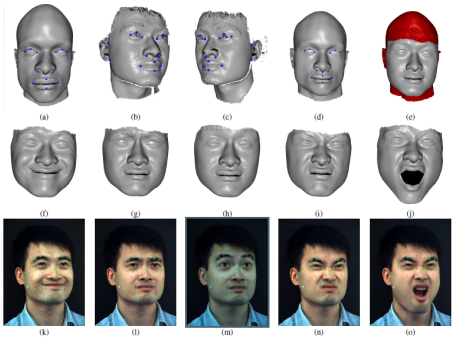

Figure 2.9: This figure demonstrates the template fitting process used in [41]. (a) A face template. (b)-(c) Depth maps from different viewpoints. Corresponding points are manually identified. (d) The template after it has been warped using the feature correspondences. (e) Initial mesh after the warped template has been fitted to the depth maps. (f)-(j) Selected meshes after the initial mesh has been tracked through the whole sequence using optical flow and depth maps.

for this purpose. However, noise in the input data makes this type of tracking very unreliable, especially for less prominent landmarks. In order to make tracking more robust, geometric constraints among the features are typically incorporated in the flow computation. This means that the tracking of each feature will be influenced by the position of the others. Different types of geometric constraints have been proposed [7], including restriction on feature displacements in expression change [8], adherence to physically based deformable mesh models [14, 13], and correspondence to face models constructed from measured examples [30, 5].

In [12], DeCarlo et al. use a simple face model with a small number of parameters to describe a variety of facial expressions, and extract the motion of the face from a single video sequence. This research computes forces based on the edges in the input image and integrates optical flow into the deformable model formulation. Optical flow provides information that can be used to constrain the motion of the face model. The shape of the face is also modelled separately from the motion of the face.

In [30] Pighin et al. fit a 3D face model to each frame in the input video in an analysis-by synthesis approach. The face model uses a linear combination of 3D face models, each corresponding to a facial expression. This allowed them to match realistic renderings of faces to the target images and recover parameters that can be used in animation. When fitting the model, the error function is minimised over the set of facial expressions and face positions spanned by the model. The parameters extracted from the image data can then be used to animate a synthetic face.

Figure 2.10: The top row shows the target images and the bottom row shows the results achieved by Pighin et al. [30] of fitting the model to these images.

Chai et al.[8] create a system which allows the user to control the animations of a 3D model by acting out the desired motion in front of a video camera. They developed a real time facial tracking system which extracts a number of feature points from the video. A preprocessing motion capture database is then used to translate these low-quality 2D feature points into high quality facial movements. They also developed an expression retargeting technique which allowed the synthesized motion to be applied to a new model.

In [34] Vlasic et al. extract speech-related mouth movements, expressions and three dimensional pose from a video sequence and use the extracted parameters to drive a 3D model. The animations can then be seemlessly rendered back into the target footage. Vlasic et al. use multilinear algebra from a set of 3D face models containing different identities, expressions and speech movements. These three parameters are learned and can be controlled independently on the 3D model. The expression and facial movement in the video are analysed and a set of optimal parameters are calculated which fit those extracted from the video. These are then used to animate the 3D model.

In [29] Pei et al. propose a method which transfers the speech movements of a person in a video onto a 3D model. They implement an unsupervised learning process which extracts intrinsic geometry from the video in order to create 3D key viseme shapes. These are then uses to form a set of 2D visemes which are then mapped to 3D face space and used to animate the 3D model.

Recently, deformation transfer between triangle meshes has become an important research topic in geometric modeling and high resolution 3D face modeling. Research has also been carried out into the use of Laplacian coordinates or differential coordinates when deforming a mesh. Laplacian coordinates are calculated for each vertex in a mesh and they represent the local shape of the mesh around that vertex. In [33]

Sorkine et al. proposed a method which represents a surface in terms of its Laplacian coordinates. They developed an interactive tool which allows the user to deform the mesh by transforming a number of handle positions. Laplacian deformation is used to calculate the deformed vertex positions. The local shape of the mesh is maintained as much as possible given the constraints posed by the user. The main computation involved is solving a sparse linear system which can be done at interactive rates. They also demonstrate that this method can be used to transfer details from one mesh to another, or transplant partial surface meshes.

Much of the research in Laplacian deformation has been based around constructing a method which can transform the differential coordinates in order to ensure that the optimal deformed mesh is reconstructed. Yu et al. [39] implement an editing system which is based on a gradient field. In this case the transformation of the handles is propagated to all vertices using a weighting scheme based on geodesic distances. Zayer et al. [40] propose a similar technique but instead propagate along harmonic fields. In [20], Au et al. present a method which tackles some of the limitations of Sorkine et al. implementation. They first propose implementing an iterative technique which updates the Laplacian coordinates at each time step in order to better preserve the local shape of the mesh. However, they state that if the mesh has irregular connectivity or complex geometry, the results of the iterative technique may not converge. They then propose a second method which involves converting the original mesh to its dual mesh and calculating dual Laplacian coordinates for each of the dual vertices. An iterative technique is again implemented in order to update the dual Laplacian coordinates. As the degree of each of the dual vertices is three, the connectivity of the dual mesh is regular. This removes the instability of the iterative process and gives improved deformation results.

In [43], Zhao et al. use Laplacian deformation to transfer the expression of a person in a 2D video onto a 3D model. In order to do this they use a number of corresponding feature points on the face in the video and on the 3D model. One assumption made in this publication is that the head remains static throughout the video and does not change position. In order to capture the facial expression seen in the video, they calculate the movement vectors of each of the facial features in frame i. This involves taking a reference image that displays the person in the video with a neutral expression. They then find the difference between the positions of the feature

points in the reference image and those in frame i. These movement vectors are then used to deform the positions of the corresponding feature points on the 3D model. Once the feature points have been deformed the final step is to deform the remaining

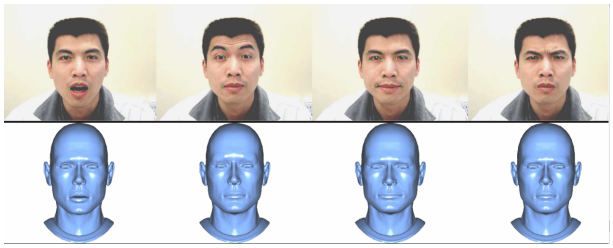

Figure 2.11: These images show the results achieved by Zhao et al. in [43].

vertices of the model. This is done using Laplacian deformation. The positions of the deformed feature points are taken to be the handle constraints. The results from this method can be seen in Figure 2.11. The expression on the model appears to accurately reflect the expression of the person in the video. However, the assumption that the head remains static throughout the video limits the data which can be processed using this technique. In our project we aim to implement a method similar to that of Zhao et al. but which can be applied to any video. This means that both the rigid head motion and facial expression of the person in the video need to be recreated.

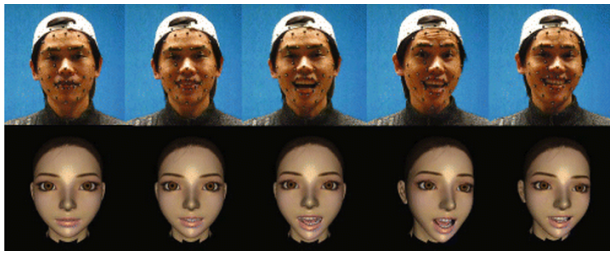

Figure 2.12: These images show the results achieved by Wan et al. in [35].

Zhao et al. also point out that as their method only considers the 2D facial movements of the face and not the 3D movements, this may cause artifacts in their results. In [35] Wan et al. overcome this problem by using motion capture data instead of 2D video data. They capture the facial expression and head movement of the performer and decompose the data into rigid head motion and change in facial expression. In order to do this they first calculate the rigid transformation of the head, given by a rotation and translation. They then apply the inverse transformation to the 3D data

in order to remove the rigid head motion from the data. At this stage the data only represents the change in facial expression. They use 36 markers to capture the facial expression of the performer and for each marker i they calculate the difference between the marker's position in two consecutive frames. These movement vectors are then used to deform the corresponding feature points on the 3D model. Laplacian deformation is used to calculate the positions of the remaining vertices. The results of this algorithm can be seen in Figure 2.12 and show that both rigid head rotation and facial expression have been successfully transferred to the 3D model. We aim to reconstruct similar results using a video sequence as input data.

Chapter 3

Head Pose Estimation

The first step in this project is to estimate the rigid pose of the head for each frame of a video. We do this by constructing an image of the Basel Face Model using a virtual camera. There are two assumptions that could be made at this stage of the project. We could assume that the camera position remains fixed throughout the video and that the position of the head is changing. In this case, the transformation of the head would need to be calculated. This is equivalent to calculating the rotation $\mathbf{R}$ and translation t which transforms the world coordinate system to the camera coordinate system. This can be seen in Figure 3.1. On the other hand, we could assume that the position of the head remains fixed throughout the video and that the camera position is changing. This involves calculating the orientation $\mathbf{R}$ and centre $\tilde{C}$ of the camera in the world coordinate system. The (3×3) orientation matrix $\mathbf{R}$ is equivalent to the rotation matrix which transforms the world coordinate frame to the camera coordinate frame. When we calculate the centre $\tilde{C}$ we can also use it to find the translation t using the relation

$$t = -(\mathbf{R}\tilde{C}). \tag{3.1}$$

Therefore both assumptions are essentially equivalent. For this step in the project, we implement three algorithms which calculate the position and orientation of the camera in the world coordinate system. We therefore assume that the position of the head is fixed and calculate the camera centre $\tilde{C}$ and camera orientation $\mathbf{R}$. This will allow us to recreate the pose of the head with the model for each video frame.

The orientation $\mathbf{R}$ and camera centre $\tilde{C}$ are known as the extrinsic camera param-

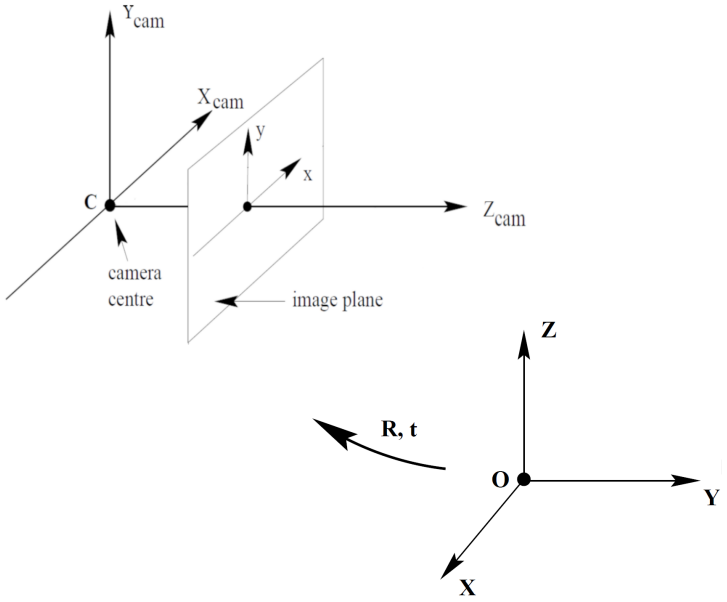

Figure 3.1: This image is taken from [17] and shows that the world coordinate frame and camera coordinate frame are related via a (3 × 3) rotation matrix $\boldsymbol{R}$ *and a translation* t*.*

eters. We let the (3×3) matrix $\mathbf{K}$,

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.2}$$

represent the intrinsic camera parameters. The values f_x and f_y are the horizontal and vertical focal lengths and (u_0, v_0) are the coordinates of the principal point. Increasing the focal length will make the camera zoom in more while decreasing the focal length will decrease the zoom and make the image appear as if it were taken further away. $\mathbf{R}$, $\tilde{C}$ and $\mathbf{K}$ must be known in order to accurately reconstruct the position of the head in the video.

The first method that we implement was described in [17] and is known as the Direct Linear Transformation (DLT) method. Before implementing this algorithm, we choose a set of feature points on the 3D model and a corresponding set of feature points on the 2D face. The DLT algorithm then uses these corresponding feature points to calculate the (3×4) camera projection matrix $\mathbf{P}$. The camera projection matrix is a

transformation matrix which projects the 3D feature points onto the 2D feature points. $\mathbf{P}$ can be decomposed to find the camera position $\tilde{C}$, camera orientation $\mathbf{R}$ and intrinsic matrix $\mathbf{K}$. $\mathbf{P}$ has the following decomposed form

$$\mathbf{P} = \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} & & \\ & \mathbf{R} & \\ & & \end{pmatrix} \left(\begin{array}{ccc|c} 1 & 0 & 0 & \tilde{C}_1 \\ 0 & 1 & 0 & \tilde{C}_2 \\ 0 & 0 & 1 & \tilde{C}_3 \end{array} \right). \tag{3.3}$$

where $\tilde{C} = (\tilde{C}_1, \tilde{C}_2, \tilde{C}_3)^T$. Therefore the DLT algorithm estimates both the extrinsic and intrinsic camera parameters. All of these are then used to reconstruct the correct pose of the head.

The second algorithm that we implement is the Normalised DLT algorithm which is a variation of the DLT algorithm. In [17], Hartley and Zisserman show that the results of the DLT algorithm depend on the coordinate system in which the feature points are expressed. This means that the choice of coordinate system is important when computing an accurate projection matrix $\mathbf{P}$. They describe a method of normalizing the data before the DLT algorithm is applied. This involves translating the coordinates of each data set so that the centroids are at the origin. Then the coordinates are scaled so that on average, for a point $\mathbf{x} = (x, y, w)^T$ each of x, y and w will have the same magnitude. The DLT algorithm is then applied to these new normalised data points.

Both the DLT and normalised DLT algorithms estimate the extrinsic and intrinsic camera parameters. For our final implementation we decided to implement a method which only estimates the extrinsic camera parameters and assumes that the intrinsic camera parameters are known. This is equivalent to estimating the camera pose or 'exterior orientation' of the camera. In this case, there are 6 parameters that must be estimated, three for the camera rotation $\mathbf{R}$ and three for the position $\tilde{C}$ of the camera [17].

As we had already implemented the DLT algorithm, we investigated whether the method could be modified in order to estimate the camera pose given that the intrinsic camera parameters are known. We found that the DLT algorithm can be used to estimate an initial camera matrix $\mathbf{P}$ and the intrinsic parameters of this matrix can then be clamped to the desired values. This modified matrix $\mathbf{P}$ can then be used to initialize an iterative technique which estimates suitable values for $\mathbf{R}$ and $\tilde{C}$, given the

new intrinsic camera parameters. Ideally the fixed intrinsic parameters will be close to the values originally obtained by the DLT algorithm, however this is not always the case. If there is a significant difference between these values, an incorrect initial matrix will be used which may lead to large residuals and hinder the converge of the iterative technique. Because of this the accuracy of the results obtained may be low [17].

In order to obtain more stable results, we implemented the EPnP algorithm, which was proposed by Lepetit et al. in [23]. This is a non-iterative method which uses a number of feature point correspondences to find an accurate estimate of $\mathbf{R}$ and $\tilde{C}$. The central idea associated with this method is to express the 3D feature points in terms of four control points. This reduces the problem to estimating the coordinates of these four control points in the camera coordinate system instead of the position of each of the feature points. As a result, this algorithm proved to be faster than other state-of-the-art methods at its time of publication.

In the next section we will explain these three algorithms in more detail. We will also present results from several experiments which were implemented in order to determine which algorithm is more suitable for our application.

3.1 The Direct Linear Transformation (DLT) Algorithm

The first method implemented is the Direct Linear Transformation method as described by Hartley and Zisserman in [17]. First, we assume that there are a number of point correspondences $\mathrm{X}_i \longleftrightarrow \mathrm{x}_i$ between the 3D model coordinates X_i and the 2D image points x_i. Both X_i and x_i are homogeneous coordinates and therefore are of the form

$$\mathrm{X}_i = \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ W_i \end{pmatrix}, \ \mathrm{x}_i = \begin{pmatrix} x_i \\ y_i \\ w_i \end{pmatrix}. \tag{3.4}$$

We let $\mathbf{P}$ be the (3×4) projection matrix

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \tag{3.5}$$

such that $\mathrm{x}_i = \mathbf{P}\mathrm{X}_i$. Using this system of three equations, we find that $\mathrm{x}_i \times \mathbf{P}\mathrm{X}_i = \mathbf{0}$ ($\times$ represents the cross product in this case). This gives a set of equations for the i-th correspondence as follows:

$$\mathbf{A}_i.p = \mathbf{0} \tag{3.6}$$

where

$$\mathbf{A}_i = \begin{pmatrix} w_iX_i & w_iY_i & w_iZ_i & w_iW_i & 0 & 0 & 0 & 0 & -x_iX_i & -x_iY_i & -x_iZ_i & -x_iW_i \\ 0 & 0 & 0 & 0 & -w_iX_i & -w_iY_i & -w_iZ_i & -w_iW_i & -y_iX_i & -y_iY_i & -y_iZ_i & -y_iW_i \end{pmatrix}, \tag{3.7}$$

$p = (p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}, p_{31}, p_{32}, p_{33}, p_{34})^T$ is a 12 element column vector of unknowns and $\mathbf{0} = (0, 0)^T$.

For a set of n correspondences, the equations for each correspondence are stacked to create a $(2n \times 12)$ matrix $\mathbf{A}$:

$$\begin{pmatrix} w_1X_1 & w_1Y_1 & \cdots & -x_1W_1 \\ 0 & 0 & \cdots & -y_1W_1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{12}X_{12} & w_{12}Y_{12} & \cdots & -x_{12}W_{12} \\ 0 & 0 & \cdots & -y_{12}W_{12} \end{pmatrix}. \tag{3.8}$$

In order to find the projection matrix $\mathbf{P}$, the equation $\mathbf{A}p = \mathbf{0}$ is solved for p. A minimum number of 6 correspondences are needed to obtain a solution to $\mathbf{A}p = \mathbf{0}$. To ensure the best results are achieved, $n > 6$ is chosen. In this case, the system is overdetermined and therefore there is no exact solution apart from the zero solution. However, one can find an approximate solution, namely a vector p which minimizes a suitable cost function. In order to avoid the solution $p = \mathbf{0}$ an additional constraint is needed. We add a condition on the norm of p, namely that $\|p\| = 1$. Since $\mathbf{P}$ is only defined up to scale, the value of the norm of p is unimportant. Therefore, instead of solving $\mathbf{A}p = \mathbf{0}$, we minimize $\|\mathbf{A}p\|$ subject to $\|p\| = 1$. In order to compute a suitable

value for p we let $\mathbf{A} = \mathbf{UDV}^T$, the singular value decomposition (SVD) of $\mathbf{A}$. p is then given by the last column of $\mathbf{V}$. In order to find $\mathbf{P}$, the entries of the vector p are rearranged as follows:

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \tag{3.9}$$

To find the camera centre $\tilde{C}$, camera orientation $\mathbf{R}$ and camera intrinsics $\mathbf{K}$, the camera matrix $\mathbf{P}$ must be decomposed. If $C = \left(\tilde{C}^T, 1\right)^T$ is a 4 element column vector representing the camera centre in homogeneous coordinates, the first thing to note is that $\mathbf{P}C = 0$. Because of this the SVD of $\mathbf{P}$ can be used to find C. Namely, if $\mathbf{P} = \mathbf{UDV}^T$, then the last column of $\mathbf{V}$ will give the value of C. If we let $C = (C_1, C_2, C_3, C_4)$, then to find the Euclidean coordinates of the camera centre $\tilde{C}$, each coordinate of C is divided by C_4 as follows:

$$\tilde{C} = \left(\frac{C_1}{C_4}, \frac{C_2}{C_4}, \frac{C_3}{C_4}\right). \tag{3.10}$$

$\mathbf{P}$ can be written in terms of $\mathbf{K}$, $\mathbf{R}$ and $\tilde{C}$ as follows:

$$\mathbf{P} = \mathbf{KR}[I| - \tilde{C}], \tag{3.11}$$

where $[I| - \tilde{C}]$ represents the identity matrix concatenated with the vector $-\tilde{C}$. Therefore we know that the first 3 rows and columns of $\mathbf{P}$ make up the matrix $\mathbf{KR}$. We can then use the RQ decomposition of $\mathbf{KR}$ to find both $\mathbf{K}$ and $\mathbf{R}$. These can then be used to capture the correct pose of the model in 3D space. This algorithm is described in detail in [17].

3.2 Normalised DLT Algorithm

The second method that we implemented was the normalised DLT algorithm. In [17], Hartley and Zisserman show that the camera projection matrix computed using the DLT algorithm depends on the coordinate frame in which the points are expressed. They state that normalising the data improves the accuracy of the results and also ensures that the algorithm is invariant to arbitrary choices of scale and coordinate

origin. The first step is to compute a similarity transformation S which transforms the 2D points so that their centroid is at the origin. Each 2D point is of the form $\mathrm{x} = (x, y, w)$ and we want to scale each coordinate so that each of x, y and w have the same average magnitude. Instead of choosing a different scale factor for each coordinate direction, an isotropic scaling factor is chosen which scales the x and y-coordinates equally. To achieve this, we ensure that S also scales the coordinates so that their root-mean-squared (RMS) distance from the origin is $\sqrt{2}$. This means that the 'average' point has coordinates of magnitude $(1, 1, 1)^T$. A second similarity transform S' is computed which transforms the 3D points so that their centroid is at the origin and scales them to ensure that their RMS distance from the origin is $\sqrt{3}$. This ensures that the 'average' point has coordinates of magnitude $(1, 1, 1, 1)^T$. The normalised data sets are then given by x' and X' where $\mathrm{x}'_i = S\mathrm{x}_i$ and $\mathrm{X}'_i = S'\mathrm{X}_i$. A projection matrix $\tilde{\mathbf{P}}$ is then found by applying the DLT algorithm to the correspondences $\mathrm{X}'_i \longleftrightarrow \mathrm{x}'_i$. The projection matrix $\mathbf{P}$ for the unnormalised data can be found from $\tilde{\mathbf{P}}$ as follows:

$$\mathbf{P} = S^{-1}\tilde{\mathbf{P}}S'. \tag{3.12}$$

The projection matrix can then be decomposed using the same method described for the DLT algorithm. This algorithm is described in detail in [17].

3.3 EPnP Method

The final method that we implement is the EPnP method which was described by Lepetit et al. in [23]. When implementing this algorithm we let p_i^w, $i = 1, ..., n$ and c_j^w, $j = 1, ..., 4$ represent the world coordinates of the 3D model and four control points respectively. We then let p_i^c, $i = 1, ...n$ and c_j^c, $j = 1...4$ represent their camera coordinates. The first step in this algorithm is to express the n 3D world coordinates as a weighted sum of the four control points:

$$p_i^w = \sum_{j=1}^{4} \alpha_{ij} c_j^w, \text{ with} \sum_{j=1}^{4} \alpha_{ij} = 1. \tag{3.13}$$

Here the α_{ij} are the homogeneous barycentric coordinates and are easily found. The control points can be chosen arbitrarily, however the stability of the algorithm is in-

creased if the centroid of the world coordinates is chosen to be one, and the other three form a basis aligned with the principal directions of the data[23]. Because of this, $\{0,0,0\}, \{1,0,0\}, \{0,1,0\}$, and $\{0,0,1\}$ were chosen to be the control points in our implementation.

Next a matrix $\mathbf{M}$ is derived in whose kernel the solution will lie. We let $\mathbf{K}$ be the known intrinsic camera matrix and $\{\mathbf{u}_i\}_{i=1,\ldots,n}$ be the 2D image coordinates. Then we have

$$\forall i, \omega_i \begin{bmatrix} \mathbf{u}_i \\ 1 \end{bmatrix} = \mathbf{K} p_i^c = \mathbf{K} \sum_{j=1}^{4} \alpha_{ij} c_j^c. \tag{3.14}$$

Here the ω_i are scaler projective parameters. This expression can be expanded by considering the specific 2D coordinates $[u_i, v_i]^T$ of each image point $\mathbf{u}_i$, the 3D coordinates $\left[x_j^c, y_j^c, z_j^c\right]^T$ of each control point c_j^c and the intrinsic matrix $\mathbf{K}$,

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.15}$$

When we substitute these parameters into Equation 3.14 we get the following,

$$\forall i, \omega_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix}. \tag{3.16}$$

The 12 control point coordinates $\{(x_j^c, y_j^c, z_j^c)\}_{j=1,\ldots,4}$ and n projective parameters $\{\omega_i\}_{i=1,\ldots,n}$ are the unknown parameters of this linear system. From the last row of Equation 3.16 we know that $\omega_i = \sum_{j=1}^{4} \alpha_{ij} z_j^c$. When this expression is substituted into the first two rows, we get the following two equations for each reference point:

$$\sum_{j=1}^{4} \alpha_{ij} f_u x_j^c + \alpha_{ij}(u_c - u_i) z_j^c = 0, \tag{3.17}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_v y_j^c + \alpha_{ij}(v_c - v_i) z_j^c = 0. \tag{3.18}$$

As these equations no longer contain ω_i, we can concatenate the equations for each

of the n reference points and generate a linear system of the form $\mathbf{Mx} = 0$. $\mathbf{M}$ is a $(2n \times 12)$ matrix and x $= \left[c_1^{cT}, c_2^{cT}, c_3^{cT}, c_4^{cT}\right]^T$ is a 12-vector made up of the unknowns. Equation 3.17 and 3.18 are rearranged and the coefficients of each reference point make up the entries of the matrix $\mathbf{M}$. As Equations 3.17 and 3.18 do not involve the image referential system there is no need to normalize the 2D projections as in the DLT algorithm [23].

The solution $\mathbf{x}$ is then found in the null space of $\mathbf{M}$ and we can express it as

$$\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i \tag{3.19}$$

where $\mathbf{v}_i$ is the set of columns of the right-singular vectors of $\mathbf{M}$. They are also the null eigenvectors of the matrix $\mathbf{M}^T\mathbf{M}$ which is a (12×12) matrix.

Once the null eigenvectors of $\mathbf{M}^T\mathbf{M}$ are found, all that remains to be computed are appropriate values for $\{\beta_i\}_{i=1,\ldots,N}$ which are the coefficients given in Equation 3.19. In theory, if at least six reference points have been imaged by a perspective camera and the data is perfect the null-space of $\mathbf{M}^T\mathbf{M}$ should have an exact dimension of $N = 1$. However the dimension may vary from 1 to 4 due to the focal length of the camera, the amount of noise present and the structure of the reference points. In order to overcome this problem, instead of picking a value for N, solutions are computed for all four values of N and the value that gives the smallest reprojection error is chosen. The formula for calculating the reprojection error is given by

$$res = \sum_i \text{dist}^2 \left(\mathbf{K} \left[\mathbf{R}|t\right] \begin{bmatrix} p_i^w \\ 1 \end{bmatrix}, \mathbf{u}_i \right). \tag{3.20}$$

where $\text{dist}(\tilde{m}, n)$ is the 2D distance between the point n and the point m expressed in homogeneous coordinates.

Case $N = 1$: In this case, the equation $\mathbf{x} = \beta\mathbf{v}$ holds. In order to solve for β we let the distance between the control points in the camera coordinate system equal the distance between them in the world coordinate system. We let $\mathbf{v}^{[i]}$ represent a sub-vector of $\mathbf{v}$ corresponding to the coordinates of the control point c_i^c. For example, the first three elements of $\mathbf{v}$ make up the 3-vector $\mathbf{v}^{[1]}$. Given that the distance between

pairs of control points must be maintained we get the following equation:

$$\|\beta \mathbf{v}^{[i]} - \beta \mathbf{v}^{[j]}\|^2 = \|\mathbf{c}_i^w - \mathbf{c}_j^w\|^2. \tag{3.21}$$

Since we know the distances $\|\mathbf{c}_i^w - \mathbf{c}_j^w\|$ we can compute β using the following formula:

$$\beta = \frac{\sum_{\{i,j\}\in[1;4]} \|\mathbf{v}^{[i]} - \mathbf{v}^{[j]}\| \cdot \|\mathrm{c}_i^w - \mathrm{c}_j^w\|}{\sum_{\{i,j\}\in[1;4]} \|\mathbf{v}^{[i]} - \mathbf{v}^{[j]}\|^2}. \tag{3.22}$$

Case $N = 2$: In this case we have $\mathbf{x} = \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2$ and the distance constraints are given by

$$\|(\beta_1 \mathbf{v}_1^{[i]} + \beta_2 \mathbf{v}_2^{[i]}) - (\beta_1 \mathbf{v}_1^{[j]} + \beta_2 \mathbf{v}_2^{[j]})\|^2 = \|\mathrm{c}_i^w - \mathrm{c}_j^w\|^2. \tag{3.23}$$

In order to solve for β_1 and β_2, a technique known as linearisation is implemented. It was previously implemented by Ansar and Daniilidis in [2] in order to estimate point depths. It involves letting $\beta_{11} = {\beta_1}^2$, $\beta_{12} = \beta_1\beta_2$ and $\beta_{22} = {\beta_2}^2$ and then solving a linear system in $[\beta_{11}, \beta_{12}, \beta_{22}]^T$. The linear system is made of six equations and can be written as $\mathbf{L}\beta = \rho$. Here $\beta = [\beta_{11}, \beta_{12}, \beta_{22}]^T$, $\mathbf{L}$ is a (6×3) matrix created from elements of $\mathbf{v}_1$ and $\mathbf{v}_2$ and the elements of the 6-vector ρ are the squared distances $\|\mathrm{c}_i^w - \mathrm{c}_j^w\|^2$. The pseudoinverse of $\mathbf{L}$ is used to solve the system. The values of β_1 and β_2 can be refined further using Equation 3.22.

Case $N = 3$: Again, the six distance constraints from Equation 3.23 are used and a linear system $\mathbf{L}\beta = \rho$ is obtained. In this case $\mathbf{L}$ is a (6×6) matrix formed with elements of $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ and $\beta = [\beta_{11}, \beta_{12}, \beta_{13}, \beta_{22}, \beta_{23}, \beta_{33}]^T$. It is solved in a similar way to the $N = 2$ case.

Case $N = 4$: In this case there are four unknowns, β_1, β_2, β_3 and β_4. Although the distance constraints used in the previous cases still suffice, there are no longer enough constraints as the linearisation procedure treats all 10 products $\beta_{ab} = \beta_a\beta_b$ as unknowns. Therefore a technique known as relinearization, which was implemented by Kipnis and Shamir in [21] was used instead. Further details about this algorithm can be found in [23].

Figure 3.2: The feature points used to calculate the rigid head movement of the face in each of the experiments.

3.4 Experimental Results

In this section we present the results of several experiments which were used to test the accuracy of the three methods outlined above. For the DLT and normalised DLT algorithms we implemented code based on the information in [17]. We also used the following resources in order to implement the technique in Matlab [3, 11]. When implementing the EPnP method in Matlab we used code made available by Lepetit et al. at [24]. We implemented this code and changed certain elements so that it would be suitable for our application.

3.4.1 Synthesised Data with No Noise

The first implementation involved creating synthetic data from the 3D model and applying the three algorithms to this data in order to compare their accuracy. First, the coordinates of 18 feature points on the model face were filled into the (3×18) matrix $\mathbf{X}$. These feature points correspond to those seen in Figure 3.2. Then a (3×4) projection matrix $\mathbf{P}$, where $\mathbf{P} = \mathbf{KR}[I| - \tilde{C}]$, was generated with a given rotation matrix $\mathbf{R}$, intrinsic matrix $\mathbf{K}$ and camera centre $\tilde{C}$. The projection matrix $\mathbf{P}$ was then applied to $\mathbf{X}$ to generate the (2×18) matrix $\mathbf{x}$. In this case there is no noise present and $\mathbf{x} = \mathbf{PX}$ exactly.

The DLT algorithm, normalised DLT algorithm and EPnP methods were then applied to the feature point correspondences $\mathbf{x}_i \longleftrightarrow \mathbf{X}_i$. Both the DLT and normalised DLT algorithms estimated $\mathbf{R}$, $\tilde{C}$ and $\mathbf{K}$. In the case of the EPnP method, we assumed that the intrinsic matrix $\mathbf{K}$ was known and only $\mathbf{R}$ and $\tilde{C}$ were estimated. For each method, an estimated camera projection matrix $\hat{\mathbf{P}}$ was constructed from the estimated extrinsic and intrinsic parameters.

The error was then calculated for each technique based on the average distance in pixels between the points in the matrix $\mathbf{x}$ representing the original image points, and the image points $\hat{\mathbf{x}}$ which are created using the estimated projection matrix $\hat{\mathbf{P}}$. The image points $\hat{\mathbf{x}}$ are created using the equation $\hat{\mathbf{x}} = \hat{\mathbf{P}}\mathbf{X}$. The formula for the error is given by

$$\text{error} = \frac{\sum_i \left(\text{dist}(\hat{\mathbf{x}}, \mathbf{x})\right)^2}{n} \tag{3.24}$$

where $\text{dist}(a, b)$ represents the 2D distance between the homogeneous coordinates a and b. n represents the number of feature points on the face. In this case, $n = 18$.

In order to calculate the mean error, 50 different projection matrices are used to create 50 sets of synthesised image points. Each algorithm is then applied to these data sets and the errors are calculated. If we let error_j represent the error that was calculated when an algorithm was applied to the j-th data set then the mean error of this algorithm is given by

$$\text{mean error} = \frac{\sum_j \text{error}_j}{m}, \tag{3.25}$$

where m is the total number of data sets. In this case $m = 50$. The mean error and standard deviation were calculated for each algorithm and the results are presented in the first row of Table 3.1. The second row of Figure 3.3 and 3.4 show the results of the normalised DLT and EPnP methods.

The error results in Table 3.1 and the image results in Figures 3.3 and 3.4 show that the accuracy of both algorithms is good when there is no noise present in the data.

3.4.2 Synthesised Data with Added Noise

For the second implementation the same process was repeated but Gaussian noise with mean zero and unit variance was added to the synthesised image points. Again the DLT, normalised DLT and EPnP algorithms were applied and the errors were

Table 3.1: Error Results for Synthesised Data

	DLT		***Norm DLT***		***EPnP***	
Noise	**Mean**	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std**
$\sigma = 0$	0.0020	0.0014	0.0000	0.0000	0.0006	0.0004
$\sigma = 1$	1.4860	0.2101	1.4837	0.2108	1.3880	0.1297
$\sigma = 3$	4.6083	0.6616	4.5547	0.6410	4.3788	4.6655
$\sigma = 10$	16.4695	2.5147	14.7654	2.2785	14.3641	1.8746

The results above show the mean error and standard deviation for each method when noise of mean zero and standard deviation σ is added to the synthesised data.

computed. The process was repeated for 50 different projection matrices and the mean error and standard deviation were calculated as before, shown in Table 3.1. These figures demonstrate that although the added noise reduces the accuracy of the results, the errors are low and the estimated projection matrices are accurate. In Figure 3.3 and Figure 3.4 we reconstruct the head pose using the camera parameters estimated by the normalised DLT and EPnP methods respectively.

The process was repeated with added Gaussian noise of mean zero and variance 3 and again with mean zero and variance 10. Table 3.1 shows that the mean error for each algorithm increases as the noise increases. In Figure 3.3 we can see that when the normalised DLT algorithm is implemented, the reconstructed head pose is quite accurate when the added noise is low. However, inaccuracies begin to arise when the noise is increased. These inaccuracies are caused by the variance in focal length. Although the errors calculated demonstrate that the normalised DLT algorithm accurately estimates a projection matrix $\mathbf{P}$, we found that the variance in the focal length and camera position caused difficulties when plotting the results with the 3D model. The result images show that the camera appears to zoom in and out incorrectly, and when an increased amount of noise is present the orientation of the head in the recreated image is also incorrect. Although the numerical errors appear to be low, they do not take these plotting inaccuracies into account.

In comparison, Figure 3.4 displays the head pose reconstructed when the EPnP method was implemented. These images show very little deviation from the original pose. Even as the noise increases, the position of the head in the images is almost identical to the originals. This method appears to create more stable results and

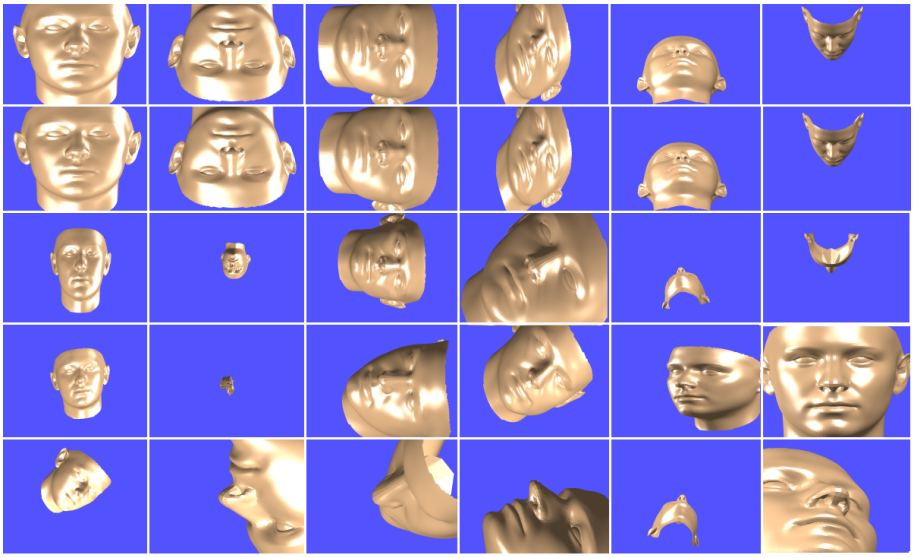

Figure 3.3: Results from the first and second experiments with synthesised data. The top row displays the original images created from the synthesised data. The second row displays the head position reconstructed when there was no noise added to the data. The third, fourth and fifth rows display the results when noise of mean 0 and variance $\sigma = 1$, 3 and 10 is added to the image data.

the fixed intrinsic camera parameters ensure that the camera does not zoom in and out incorrectly. As the DLT algorithm is similar to the normalised DLT algorithm and allows the focal length to vary, it suffers from the same plotting inaccuracies as the normalised DLT algorithm. The errors are also higher and in [17] Hartley and Zisserman state that the results of the DLT algorithm will be less accurate than the normalised DLT algorithm. Therefore we assume that the images created using the DLT algorithm will be less accurate and suffer from similar instabilities as the DLT algorithm and do not plot the DLT results.

3.4.3 Real Image Data

In the third implementation, instead of synthesising the image data using given intrinsic and extrinsic camera parameters, five images of the model were taken and the pixel coordinate positions of the 18 facial landmarks were manually calculated(Figure 3.5). For each image a matrix $\mathbf{x}$ made up of these landmark positions was created. The DLT, normalised DLT and EPnP methods were then applied to the correspondences $\mathbf{X}_i \longleftrightarrow \mathbf{x}_i$ for each image.

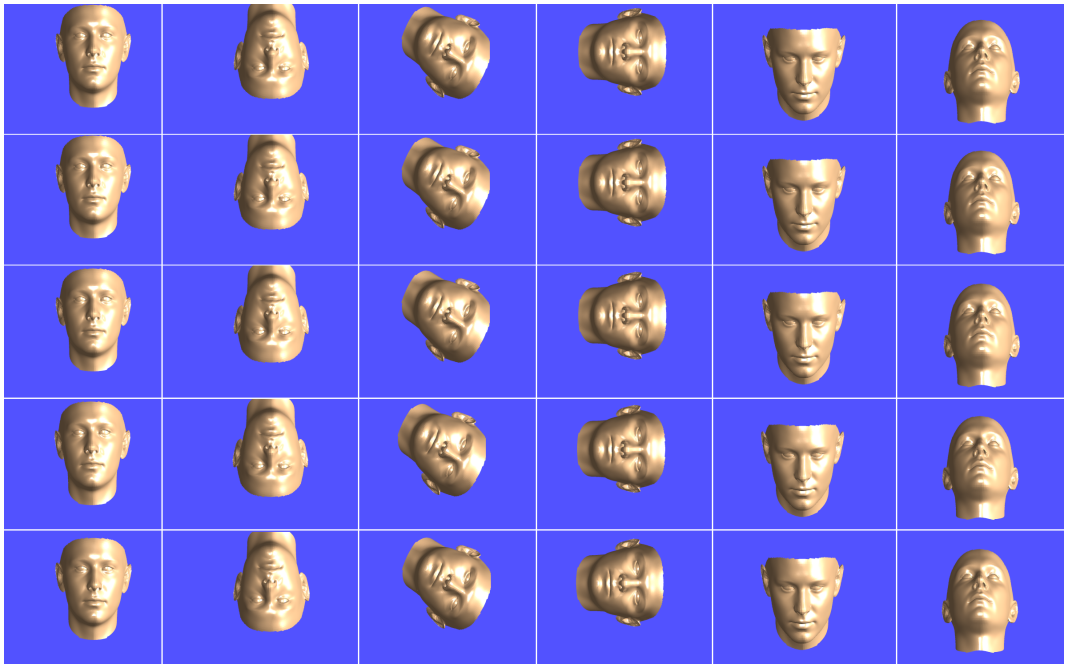

Figure 3.4: The EPnP results from the first and second experiments with synthesised data. The top row shows the original images. The second row shows the results when there was no noise added to the image data. As there was no error, these images represent an exact reconstruction of the original images. The third, fourth and fifth rows represent the results when Gaussian noise of mean zero and variance $\sigma = 1$, 3 and 10 were added.

Unlike in the first two experiments, in this case we did not know the intrinsic camera parameters. This means that a suitable intrinsic matrix $\mathbf{K}$ needed to be estimated before an accurate solution could be found using the EPnP method. In order to do this, we applied the EPnP method several times with different intrinsic parameters in $\mathbf{K}$ until the reconstructed head pose resembled that in the original images. When we achieved an accurate reconstruction we then implemented the DLT and normalised DLT algorithms, calculated the errors and compared the results. These can be seen in Table 3.2. The reconstructed head poses for the normalised DLT and EPnP algorithms can be seen in Figure 3.5.

Again the errors in Table 3.2 indicate that the DLT and normalised DLT algorithms gave a good approximation to $\mathbf{P}$. The high errors seen in the EPnP method are caused by the inaccuracy of the intrinsic matrix $\mathbf{K}$. However, Figure 3.5 shows that even though the intrinsic matrix may have been unknown and the errors appear to be higher for the EPnP method, the reconstructed results are more accurate. Again this is due

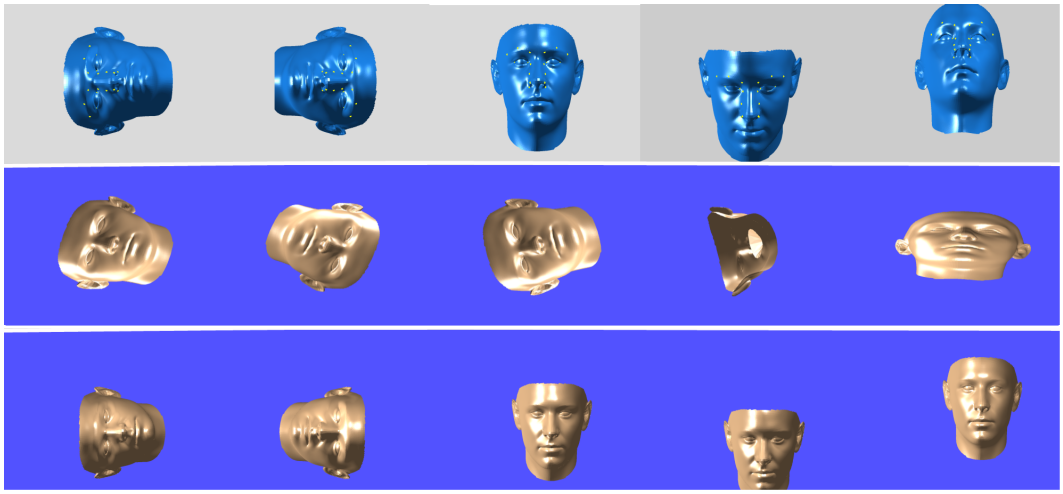

Figure 3.5: Results from the experiment using real image data of the model. The top row displays the original images taken of the model. The feature point positions were manually calculated from these. The second and third row display the results from the normalised DLT and EPnP methods respectively.

to difficulties the varying focal length posed when we were plotting the results. The errors calculated do not take these inaccuracies into account.

Table 3.2: Error Results for Real Image Data

Algorithm	**Mean Error**	**Std Deviation**
DLT	3.9581	0.8481
Normalised DLT	3.9357	0.8401
Normalised EPnP	81.7413	7.5831

3.4.4 Talking Face Video

In the final implementation the positions of 18 facial landmarks were calculated in a set of images taken from the Talking Face Video, a video database by Timothy Cootes (Figure 3.2). For each image the landmark positions were stored in the matrix $\mathbf{x}$ and the DLT, normalised DLT and EPnP algorithms were applied to the correspondences $\mathbf{x}_i \longleftrightarrow \mathbf{X}_i$. Again, an intrinsic matrix $\mathbf{K}$ was estimated and remained constant throughout. The process was repeated for 3 sets of 100 images in the database. The mean error and standard deviation of each algorithm for each set are given in Table 3.3. Again the error results for the EPnP algorithm are higher than those for the DLT and

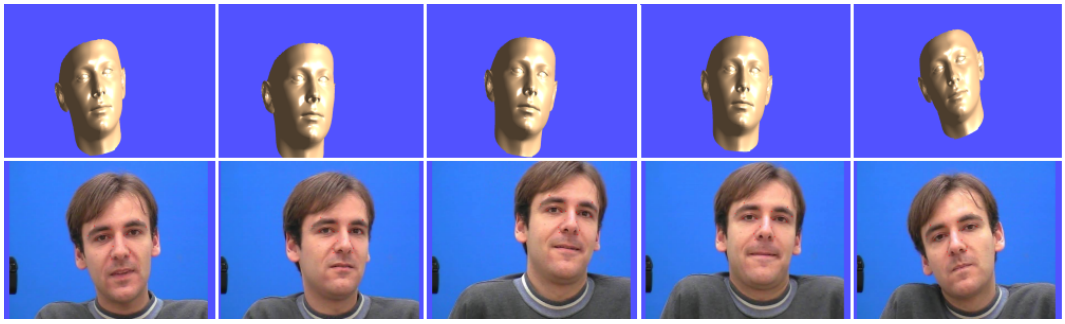

Figure 3.6: The results of the normalised DLT algorithm when it was implemented on Timothy Cootes' video. The bottom row displays the images taken from Timothy Cootes' video. The top row displays results obtained when the algorithm was applied to the data.

Figure 3.7: The results of the EPnP algorithm when it was implemented on Timothy Cootes' video. The top row displays the images taken from Tim Cootes' video. The bottom row displays the results obtained when the algorithm was applied to the data.

normalised DLT algorithms. Like the previous experiment, this is because we used a trial and error method to estimate the intrinsic matrix $\mathbf{K}$ when implementing the EPnP algorithm. The reconstructed images can be seen in Figure 3.6 and 3.7. These images show that both the normalised DLT and EPnP methods reconstruct the head pose quite accurately.

We also noted that for each experiment, the figures in Table 3.3 are higher than those calculated during the previous experiments. This may be due to several factors, including

- Noise or outliers in the image data.
- Changes in facial expression causing some landmarks to change independently of the rigid head motion.

- The landmarks given in the image may not correspond precisely with those landmarks on the model. This is due to the individual facial characteristics of the person in the video. For example, the landmark in the image which lies along the length of the persons nose may be further down than it's corresponding landmark on the model.

Table 3.3: DLT Error Results

	DLT		***Norm DLT***		***EPnP***	
	Mean	**Std Dev**	**Mean**	**Std Dev**	**Mean**	**Std**
Set 1	40.2878	1.0621	31.4040	0.1691	51.6790	2.2643
Set 2	41.9517	3.0134	31.6413	1.1375	51.8112	0.4242
Set 3	41.3970	1.1024	31.1484	0.4217	52.6578	1.3499

3.5 Comparison of Algorithms

From these experiments we can see that when the data is synthesised and the intrinsic matrix **K** is known, the errors calculated for the DLT, normalised DLT and EPnP methods are very similar. However, when the estimated camera parameters were then used to reconstruct the head pose, the EPnP method produced results that were a lot more stable than the normalised DLT algorithm. This was caused by the varying focal lengths calculated by the DLT algorithm. We found it difficult to reconstruct the head pose accurately when the focal length was allowed to vary for each image. The camera zoomed in and out and the reconstructed images were therefore inaccurate when a large amount of noise was added.

When real image data is used and the intrinsic parameters are not known, the errors for the EPnP method are much larger than those for the normalised DLT algorithm. This is because the intrinsic matrix must first be estimated and may not be extremely accurate. However, the reconstructed results are still more stable when the EPnP method is implemented. For these reasons we decided to implement the EPnP method when calculating the rigid head movement in our application. We found that the plotted results were more stable and gave a more accurate reconstruction of the head position.

Chapter 4

Expression Transfer

We have now calculated the rigid head pose of the person in each video frame. The next step is to estimate their facial expression and transfer it to the 3D model. In order to do this, we must first calculate the movement of each feature point in every frame of the 2D video. This method was also implemented by Zhao et al. in [43]. In this paper 16 feature point correspondences were used to reconstruct the facial expression of a person in a 2D video. However, as they assumed that the head position was fixed throughout the video their methods differ from the method we implemented for this project.

In order to calculate the movement of each of the 16 feature points in frame i, Zhao et al. took a reference image which showed the person in the video with a neutral expression on their face. This reference image can be seen in Figure 4.1. They then computed the pixel coordinates of each of the feature points in the reference image. They stored these values in a vector $\mathbf{F}_0$. They then calculated the position of the feature points in frame i of the video and stored the values in a vector $\mathbf{F}_i$. The set of relative movement vectors $\mathbf{E}_i$ is the difference between these two vectors,

$$\mathbf{E}_i = \mathbf{F}_i - \mathbf{F}_0. \tag{4.1}$$

As they assumed there was no head motion in their video, the motion vectors given by $\mathbf{E}_i$ represent how the facial expression of the person in the video changed. However, in our video both the head position and facial expression of the person are changing simultaneously. This means that if we implemented the same method as Zhao et al.,

Figure 4.1: The reference image used by Zhao et al. in [43] is shown on the left. A frame from their video is shown on the right with the 16 feature points marked in green.

the vectors $\mathbf{E}_i$ would represent both the change in facial expression and head position. In order to calculate the change in facial expression alone we implement and compare two methods which are describe below.

4.1 Projection Method

Figure 4.2: This image shows the 36 feature points chosen to calculate the facial expression in the video. A corresponding set of feature points were chosen on the 3D model. This frame is also the reference image used when the Transformation Method is implemented.

The first step in the Projection Method is to choose a set of feature points on the face which will enable us to accurately capture the facial expression of the person. The

36 feature points chosen on the 2D face can be seen in Figure 4.2. A corresponding set of 36 feature points are also calculated on the model. Feature points around the eyes, eyebrows, nose and mouth are chosen in order to capture when the person smiles, raises their eyebrows, blinks their eyes and a wide range of other expressions. In order to calculate how these feature points move we use the 3D model to calculate a set of 2D feature point positions which represent a face that has a neutral expression and has the same head pose as the person in the video.

For each frame i of the video we implement the EPnP method in order to estimate the camera orientation $\mathbf{R}$ and camera centre $\tilde{C}$. These are then used to calculate the camera projection matrix $\mathbf{P}$. $\mathbf{P}$ is then applied to the coordinates of the 36 3D feature points on the model. This is equivalent to rotating the model by $\mathbf{R}$, translating it by the linear transformation t (where $t = -\mathbf{R}\tilde{C}$) and then projecting it into 2D. This produces a set of 36 2D image coordinates which represent an image of the model which has the same head pose as the person in the video. Figure 4.3 shows this set of 2D image coordinates as well as the image coordinates of the 36 feature points in the video frame i. We store the coordinates of the projected 3D feature points in a vector $\mathbf{Q}_0$. We also store the pixel coordinates of the feature points from frame i in a vector $\mathbf{Q}_i$. We then subtract these two vectors in order to calculate the motion vectors $\mathbf{E}_i$ of the feature points in frame i,

$$\mathbf{E}_i = \mathbf{Q}_i - \mathbf{Q}_0. \tag{4.2}$$

This technique removes the rigid head motion information from the motion vector. However, upon inspection of the motion vectors it becomes apparent that the difference between Q_i and Q_0 does not only estimate the difference in expression. As the individual characteristics of the person in the video are different from those of the 3D model, this information is also present in the motion vectors. For example, in Figure 4.3 we can see that the distance between the model's nose and mouth is longer than that of the person in video. The shape of the model's nose is also different to the shape of the person's nose in the video. Because of this, the motion vectors calculated also take into account the individual characteristics of the person in the video. This causes errors to arise when these motion vectors are used to deform the 3D model later in the project. Even if the person in the video has a neutral expression, when the model is deformed

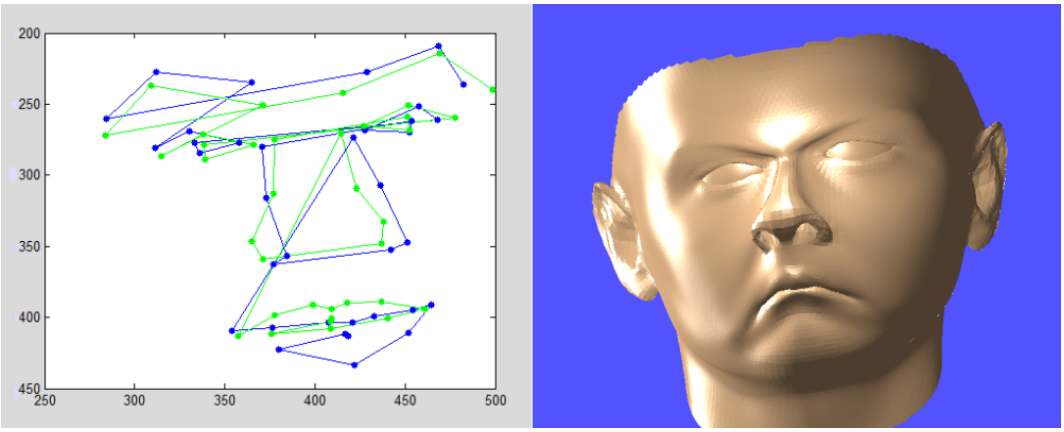

Figure 4.3: In the image on the left the connected green points represent the positions of the feature points in a single frame of the video. The connected blue points represent the positions of the feature points which have been projected from the 3D model. The difference in facial characteristics can be seen in this image. The image on the right shows the results of the mesh deformation when the motion vectors from the Projection Method are used as constraints. This face should represent a neutral expression. Stretching and distortion are evident around the lips and nose.

using these motion vectors the deformation will reflect the difference in facial characteristics. This causes undesirable results such as those shown in Figure 4.3. Because of these issues we decide to investigate a second method of calculating the movement vectors.

4.2 Transformation Method

The second method implemented involves using a rigid transformation between 2 sets of 2D feature points in order to remove the rigid head movement from the data. The first step is to chose a reference image frame from the video which displays the person with a neutral expression on their face. The video frame displayed in Figure 4.2 was chosen as the reference image. For each frame i of the 2D video, the set of feature points in the reference image (A) is then transformed so that they were aligned with the set of feature points in frame i (B). In order to calculate the transformation between sets A and B the centroids of both sets of feature points are translated to the origin. This transformation was made up of a rotation R and a translation T. The covariance matrix $\mathbf{H}$ of the feature point sets is calculated and the Singular Value Decomposition of $\mathbf{H}$ is used to find the optimal rotation matrix R between sets A and

B. The translation T is then given by

$$T = -R \times \text{centroid}_A + \text{centroid}_B.(\text{cross product}) \tag{4.3}$$

The information at [18] was used in order to implement this technique.

After this transformation has been applied to set A, the feature points in the reference image and those in frame i will be aligned. The new transformed positions of the feature points in the reference image are stored in a vector $\mathbf{Q}_0$. The position of the feature points in frame i are stored in the vector $\mathbf{Q}_i$. The movement vectors for frame i are stored in the vector $\mathbf{E}_i$ and are calculated using the following equation:

$$\mathbf{E}_i = \mathbf{Q}_i - \mathbf{Q}_0. \tag{4.4}$$

Although this technique generates accurate movement vectors for most frames of the video, when the head in the video is not facing straight ahead errors begin to emerge. This is because the reference image displays the face from a front facing viewpoint. For example, if the person in the video has a neutral expression but their face is turned sideways, the facial features will move due to the difference in perspective, not because of a difference in expression. Although this error is present, we found that the estimated movement vectors are still reasonably accurate. Because of this we decided to use the Transformation Method rather than the Projection Method when calculating the movement vectors.

4.3 Transforming 3D feature points using Movement Vectors

The movement vectors calculated using the Transformation Method are then used to deform the corresponding feature points on the 3D model. As the model is in 3D, the feature points should move in three dimensional space. As we calculated the movement vectors from the 2D video, they only describe the movement of the feature points in the X and Y directions. However, in [43] Zhao et al. state that for most front facing facial expressions, the movement of the 3D feature points is a lot greater in the X and Y directions than in the Z direction. Therefore, like Zhao et al. we assume that the

3D feature points only move in the XY plane and do not move in the Z direction. Under this assumption, we mapped the 2D movement vectors from frame i onto the 3D model.

Generally, the scale of the face in the video and the scale of the 3D model are not the same. In order to ensure that we move the feature points on the 3D model by the correct amount we normalise the movement vectors before mapping them onto the model. Specifically, we calculate the distance between the corners of the mouth in the reference image, denoted $\mathbf{f}_2$, and on the 3D model, denoted $\mathbf{f}_3$. Then the set of normalised movement vectors for frame i is

$$\mathbf{M}_i = \frac{f_3}{f_2}\mathbf{E}_i. \tag{4.5}$$

Once the normalised movement vectors are calculated we then rotate the 3D model using the rotation matrix $\mathbf{R}$ and the translation t so that the model will be in the correct position. Therefore we are now assuming the the camera is fixed and that the head is moving. If we let $d_j = (x_j, y_j, z_j)$ denote the position of the j-th feature point on the rotated 3D model and $m_j = (a_j, b_j)$ denote the normalised movement vector of the j-th feature point in a frame of the video, the transformed position of the 3D feature point is $\tilde{d}_j = (x_j + a_j, y_j + b_j, z_j)$. Now that we have the deformed position of the 36 feature points on the rotated 3D model, the next step is to calculate the deformation of the remaining vertices. This will be carried out using Laplacian Deformation and will be explained in the next section.

4.4 Laplacian Deformation

In this section we explain how to calculate the positions of the non-feature points on the 3D model. We have already computed the deformed positions of the feature points and will use these as constraints in order to deform the rest of the model mesh. We assume that the model has a triangular mesh and implement Laplacian deformation to deform it. A lot of research has been carried out in this area and this technique has been used extensively for deforming 3D models [1] [33] [20]. The main idea is to calculate a Laplacian coordinate for each vertex in the model mesh. These Laplacian coordinates capture the local differential information around the vertex. During deformation the

Laplacian coordinates are manipulated with respect to handle constraints and are used to reconstruct the deformed vertex positions. The local shape descriptors are preserved as much as possible which ensures that distortion of the local shape of the mesh is minimized [20].

A lot of research in this area focuses on the best method to manipulate the Laplacian coordinates when they are constrained by handles. They mainly focus on how to reorient the Laplacian coordinates in the correct way. In [39] Yu et al. propose using a geodesic distance method in order to propagate the transformation of the handles to other points on the mesh. The propagated transformations are then used to modify the Laplacian coordinates of these points. In [40] Zayer et al. propose using harmonic functions in order to calculate the transformations. The technique that we implement in this project was proposed by Au et al. in [20]. They aim to retain both parametrization information(shapes of the triangles) and geometry information (sizes of local features).

4.5 Laplacian Editing

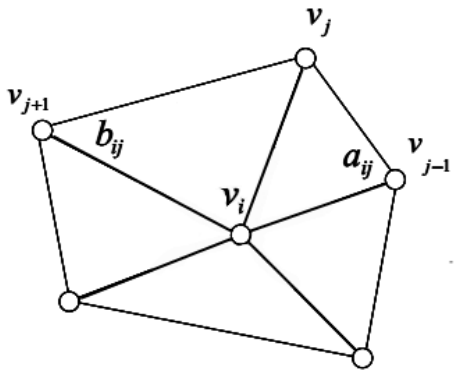

Figure 4.4: This diagram shows the position of a_{ij} and b_{ij}, the angles opposite the edge (i, j) [20].

Let $\mathbf{V} = (v_1, v_2, v_3, ..., v_n)$ represent the Euclidean coordinates and $\mathbf{L} = (l_1, l_2, l_3, ..., l_n)$ represent the Laplacian coordinates of the mesh. Both $\mathbf{V}$ and $\mathbf{L}$ are $(n \times 3)$ matrices. Let i^* represent the index set of vertices which are adjacent to v_i. These vertices are also known as the one-ring neighbourhood of v_i. The vertex v_i has a corresponding

Laplacian coordinate l_i which represents the local shape of vertex v_i. l_i is defined as:

$$l_i = \sum_{j \in i^*} w_{ij}(v_j - v_i). \tag{4.6}$$

w_{ij} is the weight of the edge (i, j) which connects vertex v_i to vertex v_j. In this case the cotangent weights are used and are defined as

$$w_{ij} = (\cot(a_{ij}) + \cot(b_{ij})). \tag{4.7}$$

Here a_{ij} and b_{ij} represent the angles opposite the edge (i, j). These can be seen in Figure 4.4. The cotangent weights approximate the curvature normal at the vertex v_i [20].

There is a linear relationship between the Laplacian coordinates $\mathbf{L}$ and the Euclidean coordinates $\mathbf{V}$. This linear relationship is given by a matrix $\mathbf{B}$ and the Laplacian coordinates can be written as $\mathbf{L} = \mathbf{BV}$. $\mathbf{B}$ is an $(n \times n)$ matrix and it's elements are derived from w_{ij}. In order to calculate the matrix $\mathbf{B}$ we must first calculate an $(n \times n)$ matrix $\mathbf{W}$ as follows:

$$\mathbf{W}_{ij} = \begin{cases} w_{ij} & \text{if } (i, j) \text{ is an edge} \\ 0 & \text{otherwise.} \end{cases}$$

We then create an $(n \times 1)$ column vector $\mathbf{D}$. $\mathbf{D}$ is simply a column vector made up of ones. We then let $\mathbf{w} = \mathbf{WD}$. Therefore the i-th element in the column vector $\mathbf{w}$ represents the sum of the i-th row of the matrix $\mathbf{W}$. We then create a matrix $\tilde{\mathbf{W}}$ as follows:

$$\tilde{\mathbf{W}}_{ij} = \begin{cases} \mathbf{w}_i & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

Then the matrix $\mathbf{B}$ is

$$\mathbf{W} - \tilde{\mathbf{W}}. \tag{4.8}$$

$\mathbf{B}$ is referred to as the Laplace operator and its elements are known as Laplacian coefficients. In order to calculate the deformed vertex coordinates $\mathbf{V}'$, we minimize $\|\mathbf{BV}' - \mathbf{L}\|^2$ using the deformed feature point positions calculated in the previous section as constraints. The idea is to minimize the sum of the squared differences between the

Laplacian coordinates before and after deformation. This problem is equivalent to solving a sparse linear system

$$\mathbf{AV}' = \mathbf{b} \tag{4.9}$$

in a least squares sense. The first 36 rows of the $((n + 36) \times 3)$ matrix $\mathbf{b}$ are made up of the positions of the deformed feature points on the model. The remaining rows are derived from $\mathbf{L}$. The matrix $\mathbf{A}$ is derived from $\mathbf{B}$ and a second $(36 \times n)$ matrix $\mathbf{C}$ where

$$\mathbf{C}_{ij} = \begin{cases} 1 & \text{if the } i\text{-th feature point on the 3D model is given by the } j\text{-th row of } \mathbf{V}. \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$\mathbf{A} \ = [\mathbf{C}^T | \mathbf{B}^T]^T. \tag{4.10}$$

The built in least squares solver in Matlab was used to solve this linear system in our implementation.

4.6 Iterative Laplacian Editing

Reconstructing the mesh using the previous method can cause some undesirable results such as shearing and stretching distortion. This is because the original Laplacian coordinates do not reflect the deformation of the local features during editing. They only represent the shape of the model before any deformation has taken place [20]. Therefore constructing the deformed mesh from the original Laplacian coordinates and the handle constraints would lead to some distortion. In order to overcome this Au et al. [20] propose implementing an iterative technique which aims to retain two types of information from the original mesh:

- parameterization information (shape of triangles);
- geometry information(size of local features).

The Laplacian coefficients which make up the matrix $\mathbf{B}$ are considered to represent the local parameterization information. The Laplacian coordinates are considered as the local geometry information of the mesh since they represent the local difference of

vertex positions. Au et al. also state that when deformed surfaces are visually pleasing, the Laplacian coordinate of each vertex is normal to the local surface. Therefore they propose a method which aims to ensure that the magnitude of the Laplacian coordinates and Laplacian coefficients remain similar before and after editing. They also aim to keep the Laplacian coordinates in the normal direction of the local surface. They propose an iterative technique which minimizes the parameterization and geometry distortion by updating both the vertex positions $\mathbf{V}$ and the Laplacian coordinates $\mathbf{L}$.

Algorithm

Let $\mathbf{V}^t$ and $\mathbf{L}^t$ represent the vertex positions and Laplacian coordinates of the model mesh at time t respectively. Then $\mathbf{V}^0 = \mathbf{V}$ and $\mathbf{L}^0 = \mathbf{L}$. The vertex coordinates and Laplacian coordinates are both iteratively updated using the following steps until the solutions converge and a termination condition is satisfied:

- **Step 1: Update the Euclidean coordinates.** The Euclidean coordinates $\mathbf{V}^{t+1}$ are calculated by solving the following sparse linear system:

$$\mathbf{A}^T\mathbf{A}\mathbf{V}^{t+1} = \mathbf{A}^T\mathbf{b}^t \tag{4.11}$$

  The matrices $\mathbf{A}$ and $\mathbf{b}$ are derived as described in the previous section.

- **Step 2: Update the Laplacian coordinates.** Once the Euclidean coordinates $\mathbf{V}^{t+1}$ have been calculated the corresponding Laplacian coordinates $\mathbf{L}^{t+1}$ can then be computed. These Laplacian coordinates have two properties that must be preserved at every time step: normal direction and magnitude. In order to maintain the original feature size of the mesh we ensure that the magnitude of the Laplacian coordinates remain unchanged. In order to do this, the Laplacian coordinates are calculated for the vertices $\mathbf{V}^{t+1}$ using the method described in the previous section. Each of the Laplacian coordinates are then normalised so that they have length 1. The new normalised Laplacian coordinates are denoted n_i^{t+1}. The size of the Laplacian coordinates calculated for the original mesh $\|l_i^0\|$ are then used to scale each of the normalised Laplacian coordinates so that they have the same size as the original Laplacian coordinates:

$$l_i^{t+1} = \|l_i^0\| n_i^{t+1} \tag{4.12}$$

The iterations are terminated when the ratio of changes in vertex positions between two consecutive time steps is below a certain threshold [20]. When the iterations converge, the resulting Laplace operator $\mathbf{L}^{t+1}$ is similar to $\mathbf{L}^0$, the Laplace operator calculated over the original mesh. Their Laplacian coefficients also remain similar which means the original parameterization information is retained. The Laplacian coordinates are in the same direction as the curvature normals and their magnitudes remain the same. This means that the local features remain similar to those of the original mesh.

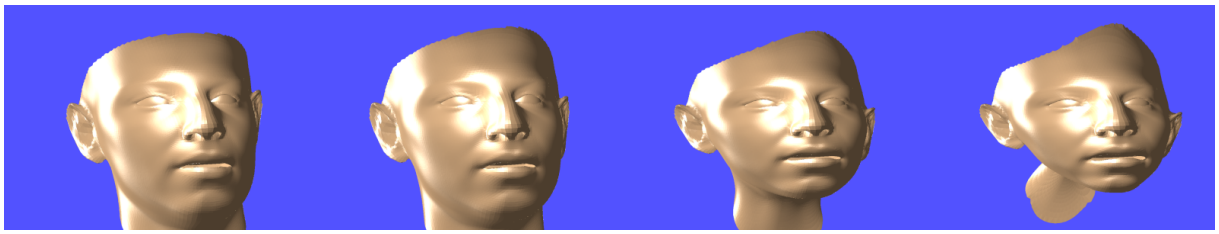

Figure 4.5: These images show the results of the iterative Laplacian deformation. From left to right, the images show the results when $t = 1$, $t = 3$, $t = 5$ and $t = 7$.

When this iterative process was implemented with our model we found that the results did not converge. Instead, the mesh became distorted and stretched. In [20] Au et al. state that the iterative process may fail to converge if the mesh has complex geometry or the one-ring neighbours of a vertex are not coplanar. As there is no common normal direction for all planes formed by the one-ring neighbours of a vertex, the local encoding will contain a tangential component which causes tangential drifts when the iterative process is implemented. The connectivity of the mesh can also affect the effectiveness of the iterative algorithm. If the mesh has irregular connectivity it is more likely that the algorithm will fail to converge. A mesh has irregular connectivity if each of the vertices do not have the same number of neighbours in their one-ring neighbourhood. However, Au et al. propose another method which is a lot more stable than the iterative Laplacian editing and which is more likely to converge. This involves converting the model mesh to its dual mesh and implementing Laplacian deformation on this mesh. Editing in the dual domain eliminates the instability of the iterative technique.

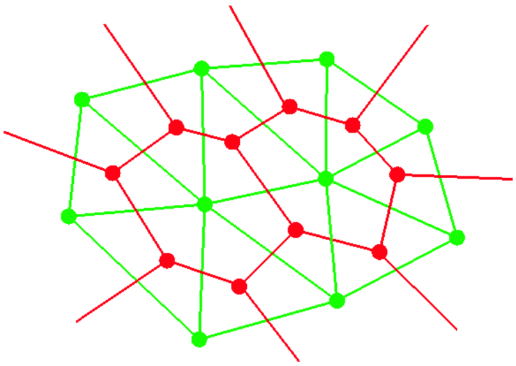

Figure 4.6: This image shows a primal mesh and its corresponding dual mesh. The primal mesh is outlined in green. The corresponding dual mesh is outlined in red.

4.7 Dual Laplacian Editing

The dual mesh consists of vertices which are positioned at the centroids of each face in the primal (original) mesh. This can be seen in Figure 4.6. There is a one to one mapping between the vertices in the dual mesh and the faces in the original mesh. There is also a linear relationship between the dual vertices $\tilde{\mathbf{V}} = (\tilde{v}_1, \tilde{v}_2, \tilde{v}_3, ..., \tilde{v}_{n_d})$ and the primal vertices $\mathbf{V}$. This linear relationship can be described by a matrix $\mathbf{J}$ and $\tilde{\mathbf{V}} = \mathbf{JV}$. $\mathbf{J}$ is constructed from the vertex-face incident matrix and it is normalised so that the sum of each row is equal to one. It is assumed that the input mesh is a triangular mesh. Under this assumption, each vertex in the dual mesh has a valance of exactly three. This means that each vertex is only connected to three other vertices in the dual mesh. Therefore the connectivity of the dual mesh is regular [20].

We define the dual Laplacian coordinate of a dual vertex to be

$$\tilde{l}_i = \sum_{j \in \{1,2,3\}} \tilde{w}_{i,j}(\tilde{v}_{i,j} - \tilde{v}_i), \tag{4.13}$$

where $\tilde{v}_{i,j}, j = \{1, 2, 3\}$ are the one-ring neighbours of the dual vertex $\tilde{v}_i$. In matrix form this is equivalent to

$$\tilde{\mathbf{l}} = \tilde{\mathbf{L}}\tilde{\mathbf{V}} = \tilde{\mathbf{L}}\mathbf{DV}. \tag{4.14}$$

The dual Laplacian coordinate of $\tilde{v}_i$ is in the normal direction of the triangle formed by the neighbours of $\tilde{v}_i$. Because of this, there are no tangential drifts and updating the dual Laplacian coordinates using an iterative technique is a lot more stable than using

the primal mesh. Again, in order to calculate the deformed vertices $\mathbf{V}'$ we minimize $\|\tilde{\mathbf{L}}\mathbf{D}\mathbf{V}' - \tilde{\mathbf{l}}\|^2$. This is equivalent to solving $\tilde{\mathbf{A}}\mathbf{V}' = \tilde{\mathbf{b}}$ in a least squares sense. $\tilde{\mathbf{b}}$ and $\tilde{\mathbf{A}}$ are created in the same way as $\mathbf{A}$ and $\mathbf{b}$, which was described in the previous section. An iterative technique can also be implemented in the dual domain in a similar way as that described for the primal domain. Again, the iterations terminate when the maximum ratio of changes in vertex positions between two consecutive time steps is below a given threshold [20].

As editing in the dual domain could improve the final results, we began implementing it for this project. However, due to time constraints and the complexity of the algorithm we did not get it completed.

4.8 Results

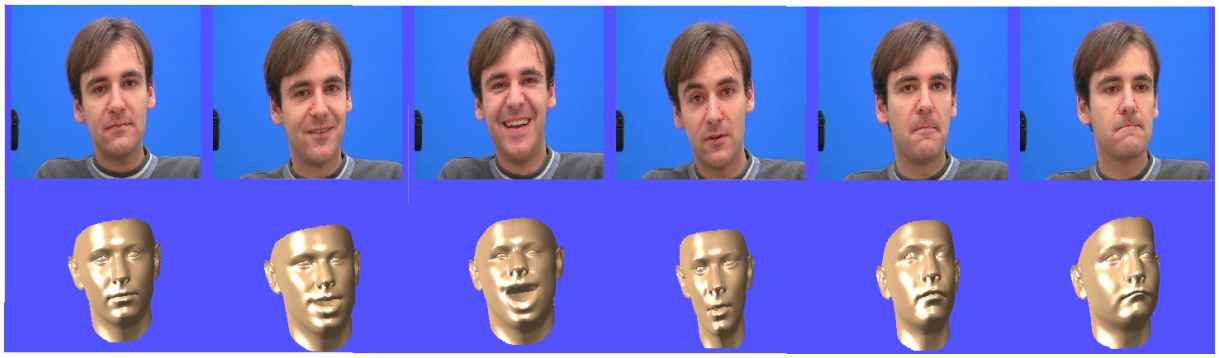

Figure 4.7: These images show the results of the non-iterative Laplacian deformation technique. The images on the top row show the image from the video which was processed. The images below show the model which has been deformed using Laplacian deformation.

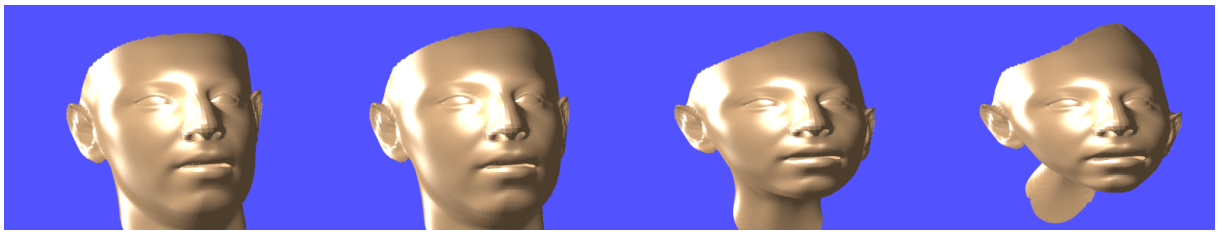

Figure 4.8: These images show the results of the iterative Laplacian deformation. From left to right, the images show the results when $t = 1$, $t = 3$, $t = 5$ and $t = 7$.

The results from the non-iterative Laplacian deformation technique can be seen

in Figure 4.7. These images show that the expression of the person in the video was successfully transferred to the 3D model. The 3D model appears to be laughing, smiling and frowning and these expressions correspond to those seen in the video frames. However, the accompanying video also shows some stretching and shrinking distortions when the expression on the model changes. The aim of the iterative Laplacian editing was to remove these distortions. However, we found that the results of the iterative technique did not converge. The resulting images can be seen in Figure 4.8. We can see that the face appears to become even more distorted as the iterations increase. Because of this, we decided to implement the non-iterative Laplacian editing in order to transfer the expression of the person in the video to the 3D model.

Figure 4.7 therefore displays the final results of our algorithm. Both the head pose and the facial expression of the person in the video have been successfully transferred onto the 3D model. For a single frame of the video, our entire algorithm took 8.818 seconds to process the image and plot the results. The most time consuming part of the algorithm was solving the sparse linear system in order to calculate the deformed vertex positions of the mesh. This took 3.773 seconds when a single frame was processed. Calculating the head pose of the person in the video took 1.209 seconds. The rest of the time was spent setting up the matrices in order to implement the EPnP methods and Laplacian deformation algorithm and plotting the results.

Chapter 5

Conclusions and Future work

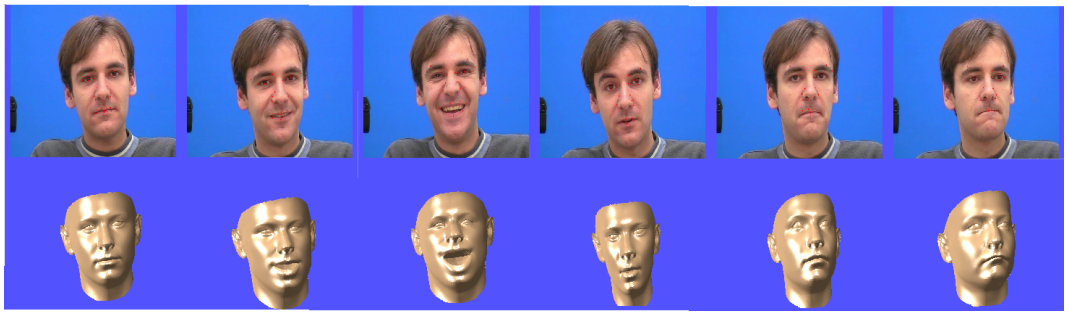

Figure 5.1: Final results.

Figure 5.1 shows the results of our algorithm. Both the head rotation and facial expression have been realistically transferred to the 3D model. The accompanying videos demonstrate the life-like transfer of facial expressions such as frowning, laughing, smiling and eye and eyebrow movements. However, there are still some limitations evident in our algorithm. We will outline some of these limitations and how they can be resolved in future.

5.1 Extreme head rotation

The first limitation can be seen in Figure 5.2. These images show the errors in facial expression which occur when the head in the video frame is turned to the side. This problem is due to the technique used to calculate the movement vectors of each of the

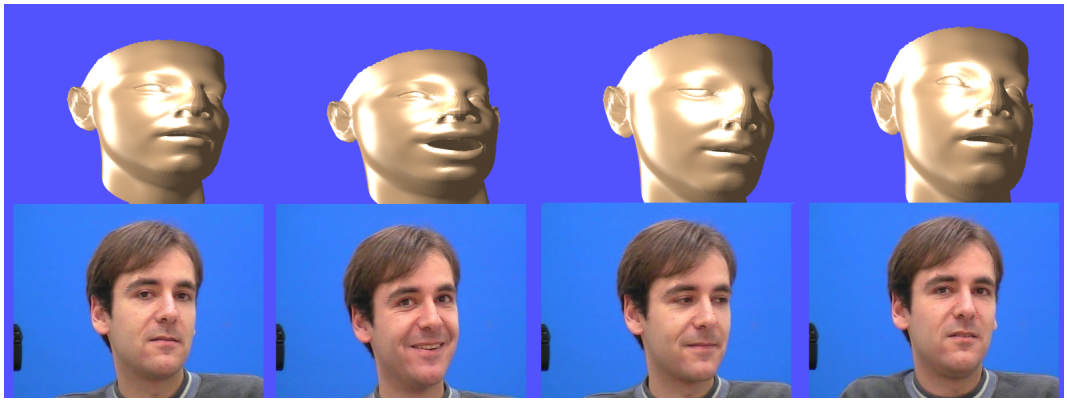

Figure 5.2: In each of these images the nose appears to bend in the wrong direction. The right eye and right hand side of the mouth are also distorted.

feature points in the video. As the reference image displays a face which is looking directly at the camera, errors will arise if the position of the head in frame i is not facing the camera directly. The reference image in this case does not represent a face which has the same head pose as the face in frame i. The movement vectors calculated will therefore contain some errors. The difference between the corresponding feature points will not accurately represent the facial expression, but will also contain information about the position of the head.

In order to overcome this problem, a method similar to the Projection Method described in Chapter 4 could be implemented. The idea behind this method is to first rotate the 3D model so that it was in the correct position. Then, the 3D feature points of the rotated model are projected into 2D. These projected 2D points are then compared to the feature points in frame i in order to calculate the movement vectors. This method was unsuccessful as the individual characteristics of the person in the video created errors when the movement vectors were calculated.

However, as we are using the Basel Face Model, which is a Morphable Model, different linear combinations of 199 principal components can be used to create different identities. When training the model, the training data had gender, height, weight, and age labelled. It is therefore possible to vary face coefficients in order to manipulate the gender, height, weight and age of the model. Figure 5.3 demonstrates some of the variations achievable with this model. The 3D model would then have a similar facial structure to the person in the video. If this new model was rotated and its feature points were projected into 2D, the difference between the projected feature points and

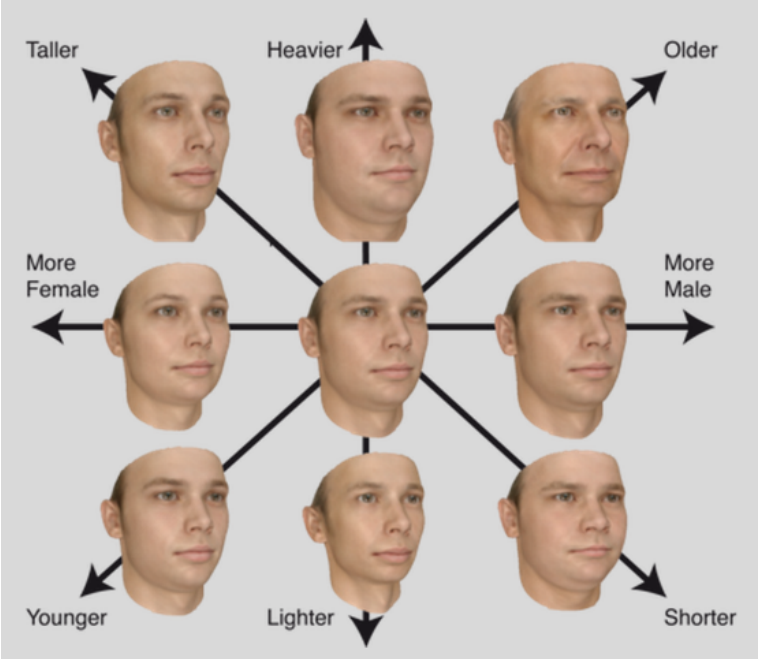

Figure 5.3: This image was taken from [28] and demonstrates some of the variations in gender, height, weight and age achievable with this model.

the feature points in frame i would only be due to changes in facial expression. The errors would be resolved as the facial characteristics of both the model and the face in the video would be the same. This would mean that even if the face was turned to the side, the facial animation would still be realistic and the artifacts visible in Figure 5.2 would be removed.

5.2 Stretching and Shrinking

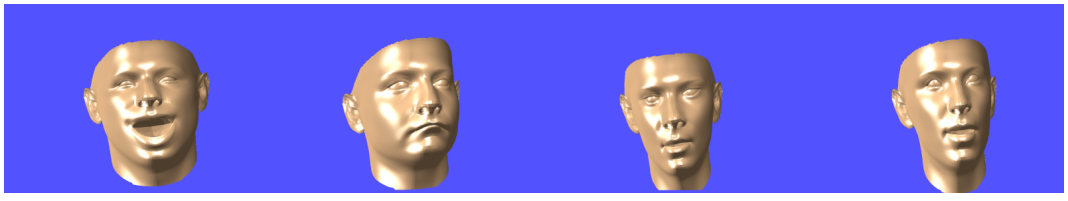

Figure 5.4: These images display the stretching and shrinking artifacts which occur throughout the result videos. The size of the model face appears to change when the expressions change.

Another artifact which is evident in Figure 5.4 and the accompanying videos is the

stretching and shrinking artifacts which occur throughout the video. The face appears to stretch and shrink as the facial expression changes. These artifacts occur because we are using the non-iterative Laplacian deformation technique. As described in the previous chapter, editing could be carried out in the dual domain [20]. This involves converting the model mesh to its dual mesh and implementing Laplacian deformation on this mesh. Editing in the dual domain eliminates the instability of the iterative technique and would eliminate the artifacts seen in Figure 5.4. Another solution would be to add extra constraints to the shape of the deformed model. For example, the distance between the ears on the model could be fixed to a particular value. This would also ensure that the stretching and shrinking was minimized.

5.2.1 Discontinuities in Rigid Head Motion

It can be seen in the accompanying videos that there are some discontinuities in the rigid head transformation calculated using the EPnP method. These discontinuities cause the head to change position rapidly between frames. This problem could be resolved by implementing a smoothing technique which would smooth the resulting video as a post-process. There are many applications available which are designed to remove camera jitter from video sequences[15, 32]. Camera jitter can be seen in videos which are taken with hand held cameras. These applications remove the discontinuities and ensure that the video runs smoothly.

Another solution would be to implement tracking when calculating the head pose. This would mean that the position of the head calculated for the previous frame could be used when calculating the head pose in the current frame. For example, a check could be implemented that would ensure that the transformation calculated in two consecutive frames could not differ by more than a given threshold.

5.3 Conclusion

We propose a method which recreates both the rigid head motion and facial expression of a person in a video using a 3D model. Our experiments show that the transfer results are successful, reconstructing animations which are life like and correspond with the movements seen in the video. As the input data is a 2D video, this method does

not require a complex acquisition system in order to obtain the data. This method could therefore be applied to many applications, especially those for which a complex acquisition system such as motion capture or 3D scanners are unavailable. As the algorithm creates a 3D reconstruction of the face in the video, the camera position in the 3D scene could also be changed in order to present the scene from a different viewpoint. This means that a single video could be used to recreate multiple viewpoints and could be useful when creating 3D films or video clips. This application would also be useful when creating virtual avatars, animating characters for 3D films, online chatting and expression imitation.

Bibliography

[1] Differential coordinates for interactive mesh editing. In *Proceedings of the Shape Modeling International 2004*, SMI '04, pages 181–190, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Adnan Ansar and Konstantinos Daniilidis. Linear pose estimation from points or lines. In *Proceedings of the 7th European Conference on Computer Vision-Part IV*, ECCV '02, pages 282–296, London, UK, UK, 2002. Springer-Verlag.

[3] Daniel Bardsley and Bai Li. 3d reconstruction using the direct linear transform with a gabor wavelet based correspondence measure, 2007.

[4] Thabo Beeler, Fabian Hahn, Derek Bradley, Bernd Bickel, Paul Beardsley, Craig Gotsman, Robert W. Sumner, and Markus Gross. High-quality passive facial performance capture using anchor frames. In *ACM SIGGRAPH 2011 papers*, SIGGRAPH '11, pages 75:1–75:10, New York, NY, USA, 2011. ACM.

[5] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[6] Derek Bradley, Wolfgang Heidrich, Tiberiu Popa, and Alla Sheffer. High resolution passive facial performance capture. *ACM Trans. Graph.*, 29(4):41:1–41:10, July 2010.

[7] Chen Cao, Yanlin Weng, Stephen Lin, and Kun Zhou. 3d shape regression for real-time facial animation. *ACM Trans. Graph.*, 32(4):41:1–41:10, July 2013.

[8] Jin-xiang Chai, Jing Xiao, and Jessica Hodgins. Vision-based control of 3d facial animation. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 193–206, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[9] I chen Lin and Ming Ouhyoung. Mirror mocap: Automatic and efficient capture of dense 3d facial motion parameters from video. the visual computer. *J Zhejiang Univ SCIENCE A*, 21:355–372, 2005.

[10] Timothy Cootes. Talking face video.

[11] Matt Dailey. Dlt demo, 2007.

[12] Douglas DeCarlo and Dimitris Metaxas. The integration of optical flow and deformable models with applications to human face shape and motion estimation. In *Proceedings of the 1996 Conference on Computer Vision and Pattern Recognition (CVPR '96)*, CVPR '96, pages 231–, Washington, DC, USA, 1996. IEEE Computer Society.

[13] Douglas Decarlo and Dimitris Metaxas. Optical flow constraints on deformable models with applications to face tracking. *Int. J. Comput. Vision*, 38(2):99–127, July 2000.

[14] Irfan Essa, Sumit Basu, Trevor Darrell, and Alex Pentland. Modeling, tracking and interactive animation of faces and heads using input from video. In *Proceedings of the Computer Animation*, CA '96, pages 68–, Washington, DC, USA, 1996. IEEE Computer Society.

[15] Amit Goldstein and Raanan Fattal. Video stabilization using epipolar geometry. *ACM Trans. Graph.*, 31(5):126:1–126:10, September 2012.

[16] Brian Guenter, Cindy Grimm, Daniel Wood, Henrique Malvar, and Fredric Pighin. Making faces. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 55–66, New York, NY, USA, 1998. ACM.

[17] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[18] Nghia Ho. Finding optimal rotation and translation between corresponding 3d points, 2013.

[19] Haoda Huang, Jinxiang Chai, Xin Tong, and Hsiang-Tao Wu. Leveraging motion capture and 3d scanning for high-fidelity facial performance acquisition. *ACM Trans. Graph.*, 30(4):74:1–74:10, July 2011.

[20] Oscar Kin-Chung Au, Chiew-Lan Tai, Ligang Liu, and Hongbo Fu. Dual laplacian editing for meshes. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):386–395, May 2006.

[21] Aviad Kipnis and Adi Shamir. Cryptanalysis of the hfe public key cryptosystem by relinearization. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 1999.

[22] Yuencheng Lee, Demetri Terzopoulos, and Keith Waters. Constructing physics-based facial models of individuals. In *In Proc. Graphics Interface 93*, pages 1–8, 1993.

[23] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *Int. J. Comput. Vision*, 81(2):155–166, February 2009.

[24] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: Efficient perspective-n-point camera pose estimation, 2013.

[25] Frederic Ira Parke. *A parametric model for human faces.* PhD thesis, 1974. AAI7508697.

[26] Frederick I. Parke. Computer generated animation of faces. In *Proceedings of the ACM annual conference - Volume 1*, ACM '72, pages 451–457, New York, NY, USA, 1972. ACM.

[27] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. 2009.

[28] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. Morphace, 2009.

[29] Yuru Pei and Hongbin Zha. Transferring of speech movements from video to 3d face space. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):58–69, 2007.

[30] Frdric Pighin, Richard Szeliski, and David H. Salesin. Resynthesizing facial animation through 3d model-based tracking, 1999.

[31] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph.*, 24(3):417–425, July 2005.

[32] Brandon M. Smith, Li Zhang, Hailin Jin, and Aseem Agarwala. Light field video stabilization.

[33] O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, SGP '04, pages 175–184, New York, NY, USA, 2004. ACM.

[34] Daniel Vlasic, Matthew Brand, Hanspeter Pfister, and Jovan Popović. Face transfer with multilinear models. *ACM Trans. Graph.*, 24(3):426–433, July 2005.

[35] Xianmei Wan and Xiaogang Jin. Data-driven facial expression synthesis via laplacian deformation. *Multimedia Tools Appl.*, 58(1):109–123, May 2012.

[36] Keith Waters. A muscle model for animation three-dimensional facial expression. *SIGGRAPH Comput. Graph.*, 21(4):17–24, August 1987.

[37] Thibaut Weise, Hao Li, Luc Van Gool, and Mark Pauly. Face/off: live facial puppetry. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 7–16, New York, NY, USA, 2009. ACM.

[38] Lance Williams. Performance-driven facial animation. *SIGGRAPH Comput. Graph.*, 24(4):235–242, September 1990.

[39] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 644–651, New York, NY, USA, 2004. ACM.

[40] Rhaleb Zayer, Christian Rssl, Zachi Karni, and Hans-Peter Seidel. Harmonic guidance for surface deformation. *Computer Graphics Forum*, 24(3):601–609, 2005.

[41] Li Zhang, Noah Snavely, Brian Curless, and Steven M. Seitz. Spacetime faces: High-resolution capture for modeling and animation. In *ACM Annual Conference on Computer Graphics*, pages 548–558, August 2004.

[42] Song Zhang and Peisen Huang. High-resolution, real-time 3d shape acquisition. In *Proceedings of the 2004 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'04) Volume 3 - Volume 03*, CVPRW '04, pages 28–, Washington, DC, USA, 2004. IEEE Computer Society.

[43] Hui Zhao and Chiew lan Tai. Subtle facial animation transfer from 2d videos to 3d faces with laplacian deformation.