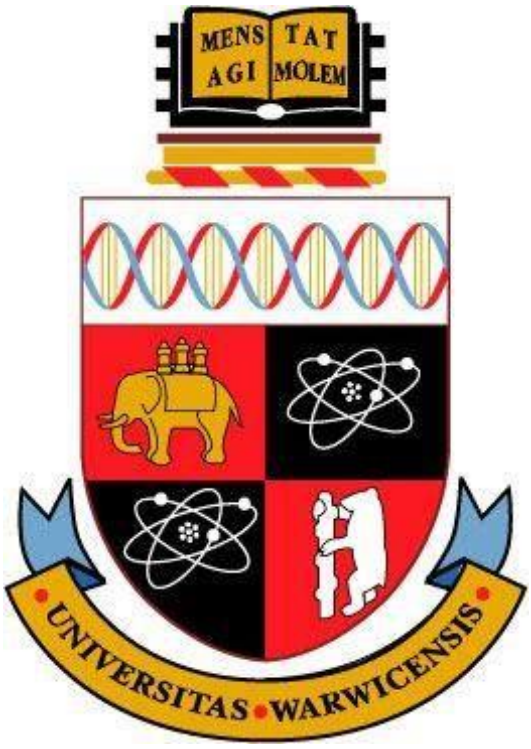
Emotion Recognition from a Sequence of 3D Facial Point Clouds



University of Warwick

School of Engineering

ES327 Technical report

Project by Benjamin Hoskings

Abstract

Affective computing and machine vision is increasingly becoming an integral part of our daily lives, with the development of systems that can accurately gauge human emotions or affective states having countless potential implementations in a wide array of human-machine interactions. Key applications include the early diagnosis of multiple medical conditions, real time accurate product evaluation, remote hospital patient monitoring and safety indicators in cars.

In the following report, a novel classification framework is presented to identify a person's emotions from an input depth scan of their face. The framework utilises three-dimensional (3D) data to increase its ability to operate under challenging conditions such as low illumination levels and large head pose variations. The classification task is defined as recognising an emotion as one of Ekmans' six fundamental emotions (Happy, Sad, Angry, Fear, Surprise or Disgust) and the input data is restricted to facial data alone. The proposed framework is composed of three fundamental modules: the pre-processing module, the feature extraction module, and finally the classification module. In the pre-processing module, the input data is refined via the application of pose-correction, face crop and surface smoothing algorithms. The feature extraction module defines two novel feature vectors that represent the movement of multiple facial landmarks over time, as well as integrating an Action Unit detection system based on the FACS framework. The classification module presents a novel adaptive classification system, based on the strategic selection of the most applicable classifier for the extracted features. The module utilises multi-class ensemble and support vector machine (SVM) classifiers.

The framework has been validated on the most widely used BU-3DFE and BU4DFE to enable performance evaluations against other SOTA frameworks. The proposed system produces higher recognition accuracies than average within the field, however direct comparison between systems is usually problematic due to lack of accepted standards for producing validation figures.

The project has produced extensive, well documented, and easily interpretable code for the implementation of the entire framework in MATLAB. This has been identified as a key output that previous studies have neglected to present and is a vitally important contribution of this work.

Table of Contents

List of Figures

Glossary and abbreviations

1. Glossary

Term	Abbreviation	Term	Abbreviation
Two-dimensional	2D	Three-dimensional	3D
Four-dimensional	4D	Facial emotion recognition	FER

Term	Explanation	abbreviation
10-fold Cross Validation	A statistical method to applied to ensure the validity of classification results	10-CV
Action unit	A movement of facial muscle pairs, creating a distinctive deformation (i.e., lowering an eyebrow)	AU
Neural Network	A machine learning technique	NN
State-of-the-art	Representing the very best within the current relevant research field	SOTA
Support vector machine	Read further at [15]	SVM
Voxel image	An image which captures depth information, as well as texture information.	-
RGB image	An image with three colour channels, representing the pixel's red, green and blue values respectively.	-
Principal component analysis	A way of dimensionally reducing the…	PCA

Acknowledgements

I am extremely grateful to my project supervisor, Dr. Tardi Tjahjadi for his invaluable advice, support, and guidance during this project, and would like to express my sincere thanks for his diligence in providing me with anything that I might ask for. I would also like to thank Dr. Thomas Popham for dealing with my wide array of questions, and always taking time to support me in my work. Finally, to my friends and family who have provided me with unconditional support throughout the project, I will always be grateful for your calming words of advice.

Introduction

1. Project overview

The project aims to produce a framework capable of recognising a person's emotional state, using data based on 3-dimensional (3D) facial expressions. The recognition task is commonly termed Facial Emotion Recognition (FER).

The recognition of human emotions by machines is something that can bring huge benefits to a wide variety of applications. Healthcare is a key field in which the technology can bring significant advances; its use in rehab programs and pain detection systems (for remote monitoring of patients), as well as aiding the diagnoses of various conditions such as Alzheimer's or Autism has been suggested [1]. Driver tiredness warning indicators [2] are another application that could have a very large (and immediate) impact on the safety of driving around the world; 29,804 people were recorded as killed or seriously injured in road collisions in the UK alone in 2020 [3]. Integrating such a system into vehicles that could reduce this figure significantly would undoubtably be a major technological success.

The ability for a system to automatically (and non-invasively) provide accurate, unbiased, real-time feedback about the engagement of a consumer whilst utilising a product is invaluable for future developments and enhancement [4]. This ability coupled with the general growth of the smart devices provides an extremely broad variety of applications and justification for undertaking the project.

Multiple systems have implemented solutions that that are based on two-dimensional data (2D) (i.e., a standard RBG image). However, more recently the literature has been turning to three-dimensional (3D) data to overcome many of the issues that arise when using 2D data alone [5]. Hence this project aims to expand the current understanding within the newly developing field, as well as building on previously presented 2-dimentional (2D) FER frameworks.

1.1. System expectations

Fundamentally, the FER framework to be developed must produce accurate recognition, taking 3D data as its input. Due to the greater availability of well-labelled databases that are linked to Ekman's six basic emotions [6], the framework output will be a prediction of one of those emotions: Happy, Sad, Anger, Fear, Surprise and Disgust. Future implementations of the system in practical situations will require the framework to produce outputs in real-time, and as such the code must be fast to run, without compromising significantly on recognition accuracy.

Producing well documented code for the implementation of the project is also a key objective of this work, and as such the expectation is that will be easily accessible to anyone with a basic knowledge of the MATLAB coding language.

1.2. System requirements

The following represents a more formal set of requirements of the project deliverables:

1. To develop a framework that classifies a person's emotion based on a 3D input data.
2. The framework must produce emotion recognition accuracies of at least 75%.
3. The framework must classify an input into one of 6 pre-defined classes.
4. The framework must take no more than 1 second to classify the input.
5. The framework must have readily readable and accessible code to enable further development and collaboration.

1.3. Structure of report

The technical report consists of ten chapters, including this introductory chapter. The remaining chapters are structured as follows:

Chapter 2 presents literature review…

Chapter 3 presents the pre-processing module of the FER framework. This normalises, refines and distils the input data, improving classification performances and ensuring that the data is in the correct form for the subsequent feature extraction.

Chapter 4 presents the feature extraction module of the FER framework. The formation of the proposed action unit features and fusion features is detailed, along with investigations into other feature vectors.

Chapter 5 presents the classification module of the FER framework. The chosen classifiers within the system are presented, and the proposed Adaptive Classifier module is described.

Chapter 6 presents the optimisation of the feature extraction and classification modules. Each of the primary parameters that define both modules have been tested over a large range of values to obtain the optimal classification performance of the framework.

Chapter 7 presents the FER results when the proposed framework is applied to the BU3DFE and BU4DFE datasets [7][8].

Chapter 8 presents a detailed documentation of the software created to implement the classification system.

Chapter 9 presents the analysis of the devised FER framework in terms of the fulfilment of the system requirements.

Chapters 10 and 11 presents the conclusions of the project, as well as suggestions for further works to expand on all areas addressed in the report.

Literature Review

2. State of the Art

This section presents a review of the current state-of-the-art methods within the scope of three 3D FER tasks. The review first outlines the primary reasons for undertaking the project, the existing challenges, and the potential applications of the technology. Most FER frameworks can be deconstructed into three fundamental modules: pre-processing, feature extraction and classification. Thus, the review focuses on these modules discussed in the relevant published papers.

2.1. 3D data – A more robust solution

Traditional FER frameworks have focused on automated emotion recognition from two-dimensional (2D) RGB facial images. While these systems can achieve high accuracies with minimal data, they often lack the real-world robustness that their more sophisticated 3D counterparts can integrate. 3D frameworks utilising 3D data (the inclusion of depth values for pixels), e.g., [1], [9], [10], can employ a range of pre-processing algorithms to mitigate the effects of variations in head pose and scene illumination.

2.2. Current challenges within the field of 3D FER

Many papers within the literature neglect to report the methods used to perform FER in such detail that would allow replication of their results in most (if not all) aspects of their proposed framework [5]. This is a key challenge in the community as peer review is a fundamental and indispensable process in furthering scientific progress. The publicly available datasets [6] also facilitate the development of FER systems that are comparable or better than the state-of-the-art frameworks. However, there is the lack of consistency between the data provided by different datasets.

Many of the papers surveyed deal with only 3D images (or analogous depth scans) [5][6]. This is despite the literature presenting clear evidence of the increased effectiveness when considering multimodal data methods (e.g., processing 3D texture images or 3D videos). This tendency is likely due to the required complexity of the frameworks needed to deal with these types of data; with an increased density of data comes an increase in computational cost. The same arguments apply to the use of local feature extraction and pre-processing methods, where many of the papers defaulted to simplicity in their solutions, sacrificing robustness and accuracy of their proposed method [5].

2.3. Pre-processing

Many of the solutions to FER reported the use of pose correction of the data prior to classification [5]. The most common algorithm used is Iterative Closest Point (ICP) [11]. However, the Möbius transform, principal component analysis (PCA) and Procrustes superimposition have also been presented as methods for achieving such a correction. Pose correction is able to be applied to any of the points in the input image, e.g., correction of facial key points only, key points and their neighbouring points, or the key points of the entire face. The parameters utilised with the ICP algorithm are generally not reported and as such it is difficult to obtain performance comparisons against other pose correction techniques.

To remove the noisy and redundant data around the facial region, some studies reported the use of a spherical crop of the data [10]. Whilst again the process is generally under-reported, most of the processes crop a spherical region centred on the nose tip, of between 70mm-90mm radius [5].

To reduce the noise within the collected data, various filtering techniques have been applied [12], e.g., median filter, Gaussian filter, average filter and Laplacian operator. For some frameworks to be robust against data with holes, such as those expressing an emotion with an open mouth (e.g., shock, pain, etc.), holes in the data are occasionally filled using cubic interpolation [13]. However, none of the studies employing this technique detail the process.

2.4. Feature extraction

The FER framework proposed in [1] aims to utilise the descriptive power contained in the evolution of an expression over time. The framework maps the input depth scans into a set of elements on a Grassmann Manifold. The trajectories of the elements are subsequently analysed and the distances between the elements are calculated. These values are combined to produce the descriptive feature, Geometric Motion History (GMH).

The paper [9] proposes a method based on shape spectral analysis. For each facial landmark, a surface patch of the local geometry is extracted, and the mesh coordinates for each patch are subsequently projected, via the Graph Laplacian operator, onto a common eigenspace. The relevant projections constitute the feature vectors. The dimensionality of the output features is restricted to the first 50 components of the eigenspace based upon a previous work's findings.

The use of a 3D Morphable Model (3DMM) to extract features is presented in [14]. A model is initialised that enables the separation of the personal identity of the subject from their emotional expressions. To obtain the subject's temporal evolution of emotional expressions a 3DMM is computed for two successive time frames. The expression parameters of the models are compared to represent the scene

flow between the two frames. To utilise the temporal information, the first and second derivatives of the scene flow are computed. The two computed features are concatenated to form the 3D Facial Spatio-temporal Features (3DSF) of the frames. Finally, window-based mean pooling and standard deviation pooling are applied to the 3DSF, to form the final feature vector.

2.5. Classification

Support Vector Machine (SVM) [15] is the most common means for the classification of emotions. This method consistently outperforms, in terms of classification accuracy, those that use Hidden Markov Models [16] or Fishers Linear Discriminant Analysis [17].

The paper [1] presents a Structured Output SVM (SO-SVM) for classification. Each depth video is tagged with the start and end times of an emotion expression. Due to the SVM output being a binary value, sub-sequences obtained between the start and end of the emotion expression are given a positive label, whilst the remaining data is given a negative label.

The increased use of deep learning techniques to perform FER [6][14][18] is noted, as these models have the potential to be more robust to various inconsistencies within the data, and to reduce the reliance on computationally expensive pre-processing techniques. Despite the promise of high recognition rates, there are important limitations to deep learning methods. Firstly, an extremely large volume of data is required to train a highly accurate network, and this is further problematic given the density of data in each emotion sample (which is further amplified when using 4-dimensional (4D) data). This volume of data is also simply not available in many of the publicly available databases, thus providing a significant barrier for these classification systems.

The paper [14] presents a Long Short-Term Memory (LSTM) Neural Network (NN) for FER. The network consists of two stacked hidden layers, a LSTM layer, an output unit, and an activation function (tanh). The LSTM is trained using the concordance correlation coefficient as the loss function.

Framework methodology

3. Pre-processing module

Due to the variations in the quality of data capture and acquisition techniques utilised in different publicly available datasets, a pre-processing module is applied to improve the generality and robustness of the emotion recognition framework. The pre-processing module also enhances the recognition accuracy under non-ideal conditions, such as those encountered in real world applications. These conditions include varying illumination levels, head pose variations of the subject, and facial occlusions produced by pose variations or other body gestures.

The pre-processing module applies pose-correction, face crop and surface smoothing (noise reduction) algorithms to the raw data, due to the consensus among researchers in the field that these significantly aid recognition frameworks. The module is structured in such a way that more computationally intensive operations such as the fine noise filter and the fine pose normalisation are computed after the face crop, avoiding the processing of redundant data. The overview of the module is shown in Figure 1. The results are shown in Figure 10.

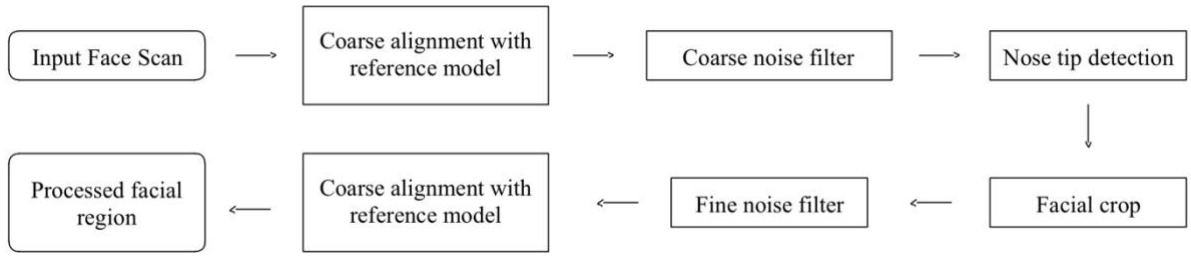


Figure 1. The pre-processing module.

3.1. Pose normalisation

A key advantage of utilising 3D face data over the more common 2D face data approach is that it enables accurate facial pose normalisation. The approach used within the module is to register the input to a reference model with the desired normalised pose (i.e., facing front and centre). There are multiple registration algorithms available to achieve this, however the Iterative Closest Point (ICP) algorithm [11] was selected because of its ability to simultaneously compute the optimal registration, and the transformation matrix that accompanies it.

The ICP algorithm aligns an input point cloud to a reference point cloud. The process iteratively finds registrations between points in the two clouds and then computes the transformation that minimizes the

distance between them. This process is repeated until convergence is reached, or the maximum number of iterations has been reached. Due to the high computational cost of this algorithm, an optional sample rate is provided to the function to down-sample the two clouds. This significantly reduces the computational time required to perform the function and is highly recommended for point clouds with more than 10,000 points. Another key factor in the section of the ICP algorithm is that the output is in the form of a rigid transformation, rather than non-rigid transformation, ensuring that the process is linear.

3.2. Point cloud surface smoothing

Many 3D data acquisition systems, particularly computationally low-cost systems, often contain high level of noise in their capture. To mitigate the noise, and smooth the face surface, the density of neighbouring points is observed. Anomalous data will often (and by definition) have few neighbours surrounding it within a given distance. The average distance between each of the anomalous data point and the closest K Nearest Neighbours (KNN) [ref] is thus computed, and if this is significantly larger than that of the average, the point will be treated as noise.

This is achieved by checking the local point density around each point. The mean distance between each point and its closest specified number of points (default is 20 in software implementation) is calculated for every point in the cloud. If this average distance is greater than 2.5 times the average for all points, the point is considered noise, and is thus removed. These values have been determined empirically to be effective whilst maintaining relative computational efficiency.

3.3. Nose tip detection

Given a point cloud of facial coordinates, many studies within the literatures simply take the point furthest away from the face x-y frontal plane (i.e., the highest z value), as the nose tip [14]. Whilst it is generally true that the nose tip is the highest z value, this assumption is susceptible to noisy/anomalous data. The pre-processing module thus employs multiple verification techniques to ensure that the correct point is identified.

The construction of a mirror face and registering algorithms to obtain the plane of face symmetry is presented in [10]. This technique forms the basis of the pose normalisation process and is utilised in the identification of the nose tip. Each step of the process is shown and enumerated in Figure 2, showing the full identification workflow.

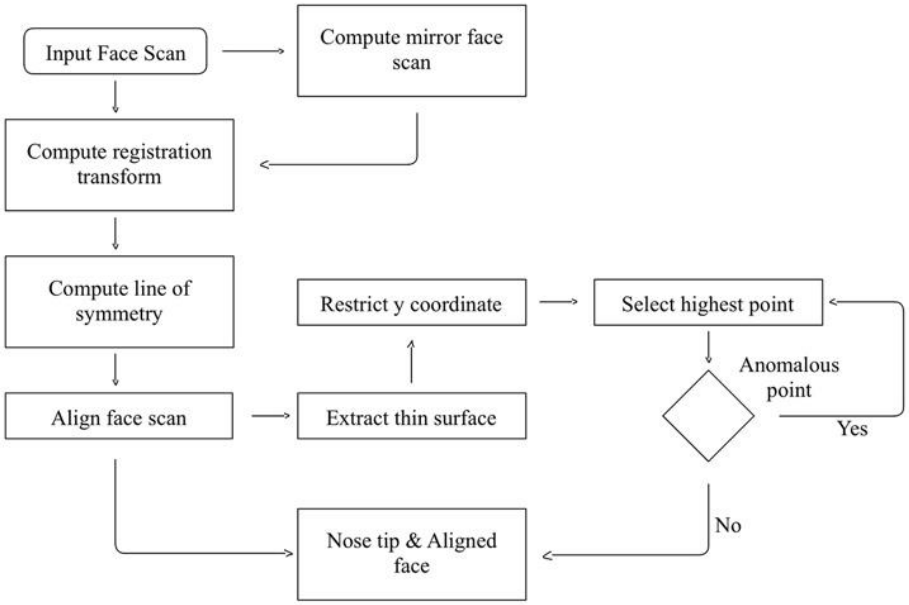


Figure 2. The proposed nose tip detection algorithm

The nose tip detection starts by generating the mirror face of the input face scan. This is achieved by inverting the horizontal frontal x-coordinate of every point in the point cloud. The mirror face is then registered with the original scan, using the ICP algorithm. The registration outputs the optimal mapping transformation matrix of corresponding points in the two scans. From the transformation matrix, the x, y and z rotations are extracted, and from these values the plane of face symmetry is computed, as denoted by the red plane in Figure 3. The points of the symmetry plane are calculated by applying the inverse of the transformation output by the ICP algorithm to the y-z plane.

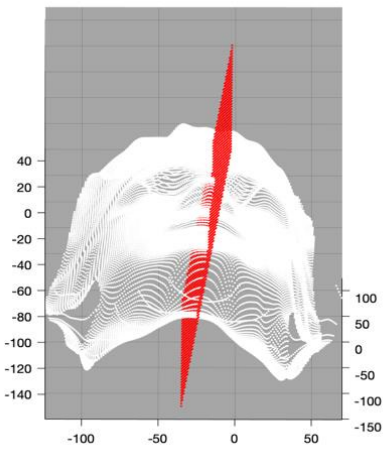


Figure 3. Face plane of symmetry

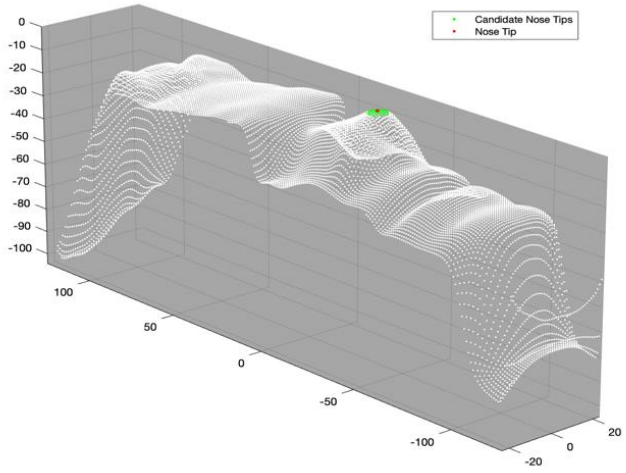


Figure 4. Thin surface with candidate points.

The face is then aligned such that the plane of face symmetry is parallel to the vertical frontal y axis. This process produces a normalised version of each face scan. Next, a thin surface is taken as the set of

points whose distance from this line of symmetry is less than 10mm, as denoted by the white surface in Figure 4.

The final point set of candidate nose tips is taken as all points on this surface that are within a given distance from the centre of the scan. This is due to the occurrence of noise/misleading data around the edges of the scans, which could protrude out further than the nose tip. From this point set, the highest point (i.e., furthest away from the face z-x frontal plane) is selected to undergo the final verification step. This step checks that the point has at least 5 other points within a distance of 5mm around it and acts to prevent the selection of an anomalous data point. a point fails this check, the next highest point is selected until a candidate point passes. A sample rate less than 1 can be provided to speed up the computation due to the usage of the ICP registration algorithm. However, it should be noted that this comes at the expense of accuracy and might generate anomalous results should the rate be reduced significantly.

3.4. Face crop

Due to the inherent density of the 3D data, the pre-processing module aims to remove any redundant data. When analysing facial movements for recognising an emotion, the most descriptive parts of the face are often the eyes, eyebrows and the mouth. All of these regions are typically within a small distance from the nose tip, and as such the amount of data to be processed can be dramatically reduced to improve operational efficiency (whilst maintaining the important emotional descriptive data) by utilising only the points which are close to the nose tip.

To achieve this, spherical crop of the data is extracted, centred on nose tip as illustrated in Figure 5. The crop is achieved by computing the Euclidian distances from the nose tip for each point in the face scan. The radius of the crop ($r = 100mm$) has been empirically determined from the dataset. Any point at a distance greater than r is removed from the scan. Formally, the spherical crop is achieved using an exhaustive nearest neighbour searcher [19], where every distance between the nose tip and a point in the face scan is computed.

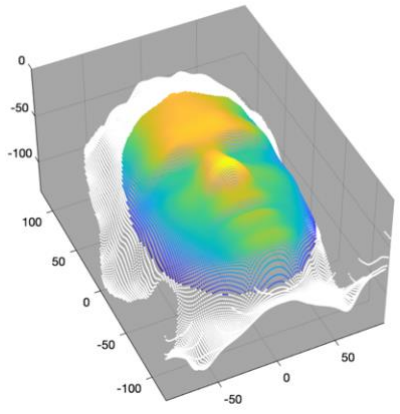


Figure 5. Spherical face crop.

3.5. Point cloud re-sampling

To improve the regularity of the pre-processed data, each cropped facial region is resampled to contain a set number of points. This has been set to be 20,000 for a reasonable density of points. The resampling process involves first up-sampling the point cloud along with a subsequent down-sampling to the desired resolution.

3.5.1. Up-sampling

To up-sample the facial region, first a Delaunay triangulation mesh is created using the x and y coordinates of each of the facial vertices, as shown in Figure 6. Next the centres of the triangles, referred as incentres, are calculated as illustrated in Figure 7, and a new facial point is placed at every centre. The z value is taken as the mean of the z values of the three vertices of a triangle. This creates an evenly up-sampled face, with similar local point-densities to that of the input cloud, illustrated in Figure 8.

3.5.2. Down-sampling

For the subsequent down-sampling of the point cloud, the number of desired points is randomly selected from the point cloud. This process is justified so long as the number of desired points remains large, and this is ensured in all applications within the module (the number of desired points is never less than 5,000).

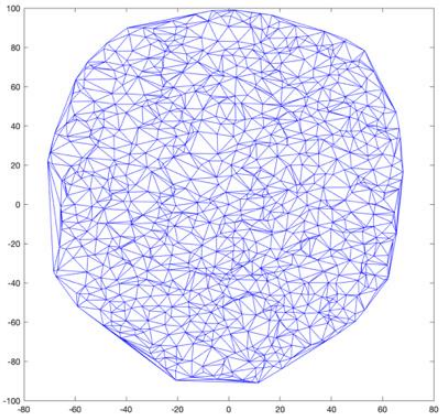


Figure 6. 2D Delaunay Triangulation mesh.

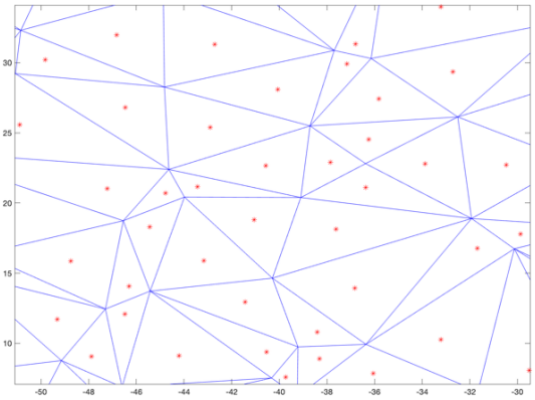


Figure 7. Incentres of triangulation mesh.

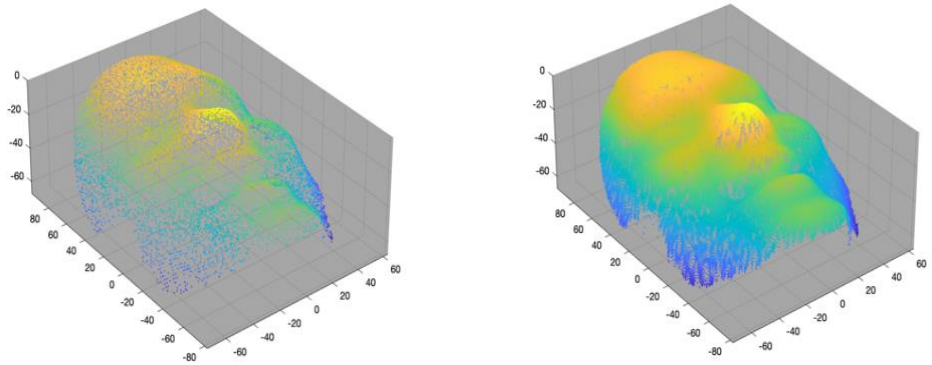


Figure 8. Up-sampling of (left plot) low-resolution input point cloud to (right plot) point cloud with similar local-point densities.

3.6. Voxelization

The point cloud locations are in continuous 3D space and need to be translated into discrete 3D space for further use in key point detection, feature extraction and classification processes. The voxelization algorithm creates a uniform 3D grid and places it over the limits of the point cloud. This is achieved by placing each point within the cloud in a bin based on its location, performed by the MATLAB function *pcbin* [20]. The number of bins in each of the x, y and z dimensions is specified by the optional argument *GridSize*, given as $[xbins, ybins, zbins]$. Any grid section (bin) which contains at least one point is given a value of 1, whereas all others are given a value of zero. This creates a face mask as shown on the left of Figure 9. For certain applications it is preferable to have a solid volume rather than a mask and as such the values behind the facial surface can be filled in, as illustrated on the right of Figure 9.

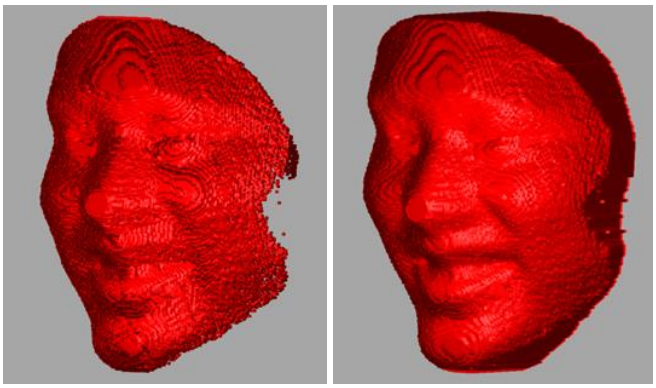


Figure 9. Voxel mask (left) and Voxel volume (right).

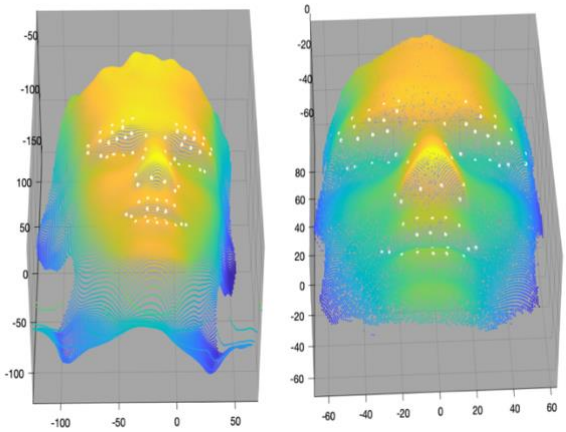


Figure 10. Raw data (left) and processed data (right).

3.7. Neutral face selection

To enable comparable classification on the BU4DFE dataset, an additional module is required to determine the subject's neutral appearance, as this is not explicitly provided. For each of the six expressed emotions, there is theoretically a neutral face at the start and end frames of each of the sequences. Therefore, there are 12 candidate neutral faces (2 for each of the 6 expressed emotions). To assess which frame is most likely to be a true neutral face, each is registered to the reference model using the ICP algorithm, and the Mean Squared Error (MSE) between the two point clouds is calculated. Frames with larger MSE values are more different to the neutral face, and thus are more likely to be emotionally expressive frames. The neutral is thus taken as the frame with the lowest MSE value from the set of input candidates.

4. Feature Extraction

4.1. Action unit detection

Action unit (AU) are facial features that are investigated due to the widely used Facial Action Coding System (FACS) [21]. The FACS defines multiple AUs, each corresponding to the movement of a specific muscle pair in the facial region. For instance, AU12 describes the movement of the Zygomatic Major muscles, whose activation leads to the upward and backward pulling of the lips' corners, producing a smile. Consequently, AU12 is associated with the expression of joyful emotions. In total, 30 AUs can be identified to capture a wide range of facial movements and expressions.

Tracking the movements of facial landmarks over time enables the detection of multiple relevant AU activations. The FACS framework typically operates on 2D images, and therefore despite the landmarks being given in 3D space, the movements are checked in just the horizontal x and vertical y directions. To identify specific activations, movements of the relevant landmarks are monitored as detailed in Table 1.

Each AU is associated with an increase or decrease in its position value for each direction. For example, AU12, which corresponds to a smile, has an increase in the y direction and an increase in value from the vertical plane of face for the x direction. To evaluate changes in the vertical direction, the y-coordinate of each landmark is compared. An increase in the y-coordinate value indicates movement in an upward direction, while a decrease indicates movement in a downward direction. To evaluate the changes in the x-direction, an additional consideration is necessary. This is because landmark pairs tend to move in opposite directions, relative to the origin of the face region. For example, the corners of the mouth both move away from the mouth centre (moving in opposite directions) when smiling.

To ensure that only descriptive movements are detected and to mitigate against noise, a threshold value is used. This threshold ensures that the movements detected are sufficiently large to be meaningful. to account for the scale in landmark movements in different AUs. Three threshold values are used: small, medium and large. The threshold choice reflects that the magnitude of the AU movement (i.e., AU12 will produce a large landmark movement when activated compared to the relatively small movements within AU41 which squints the eyes). The magnitude designations are shown in Table 1.

Figure 11 shows the process of detecting movements in the vertical and horizontal directions on the left and right, respectively. Firstly, the change in landmark position, Δ, is calculated. Next the change in the value of the relevant component (dx or dy) of the movement is checked, and if the movement is larger

than the specified value, the AU is considered present within the data and the activation flag is set to true.

Using the 68 landmarks provided in the dataset, the identification of the AUs in Table 1 is performed. For classification, a Boolean feature vector of length 16 is formed from the activation of each AU in Table 1 (i.e., each element is 1 if the AU is detected, 0 otherwise), termed α_F.

$$\alpha_F = \left[\begin{array}{cccccccccc} \overleftrightarrow{AU1} & \overleftrightarrow{AU2} & \overleftrightarrow{AU4} & \overleftrightarrow{AU5} & \overleftrightarrow{AU10} & \overleftrightarrow{AU12} & \overleftrightarrow{AU15} & \overleftrightarrow{AU16} & \cdots \\ \overleftrightarrow{AU17} & \overleftrightarrow{AU18} & \overleftrightarrow{AU20} & \overleftrightarrow{AU24} & \overleftrightarrow{AU25} & \overleftrightarrow{AU27} & \overleftrightarrow{AU41} & \overleftrightarrow{AU42} & \end{array} \right] \quad (4.1)$$

Where $\overleftrightarrow{AUx}$ represents the activation of the relevant action unit as a Boolean value.

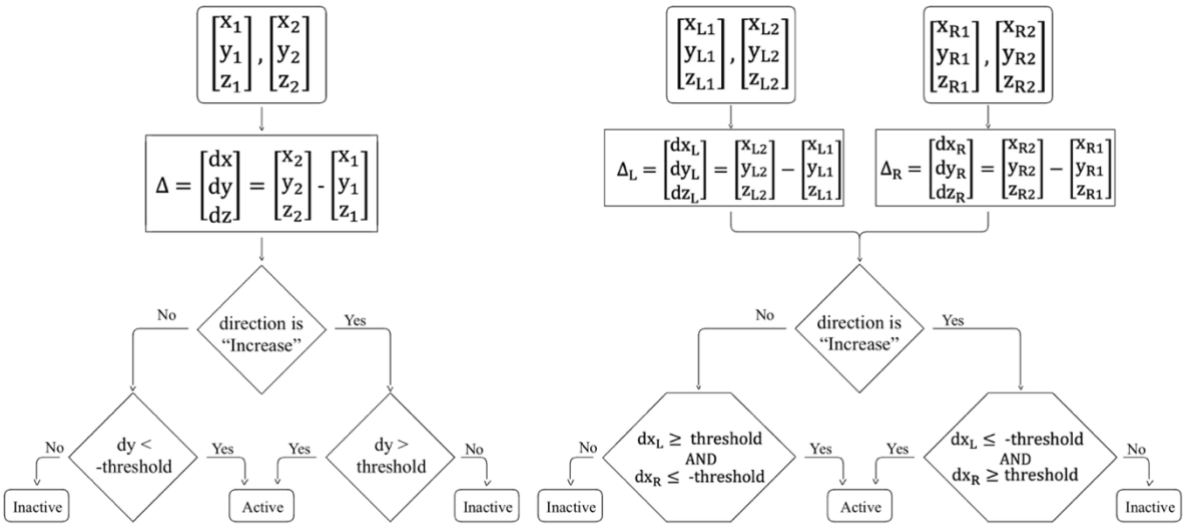


Figure 11. Horizontal AU detection (left) and Vertical AU (right)

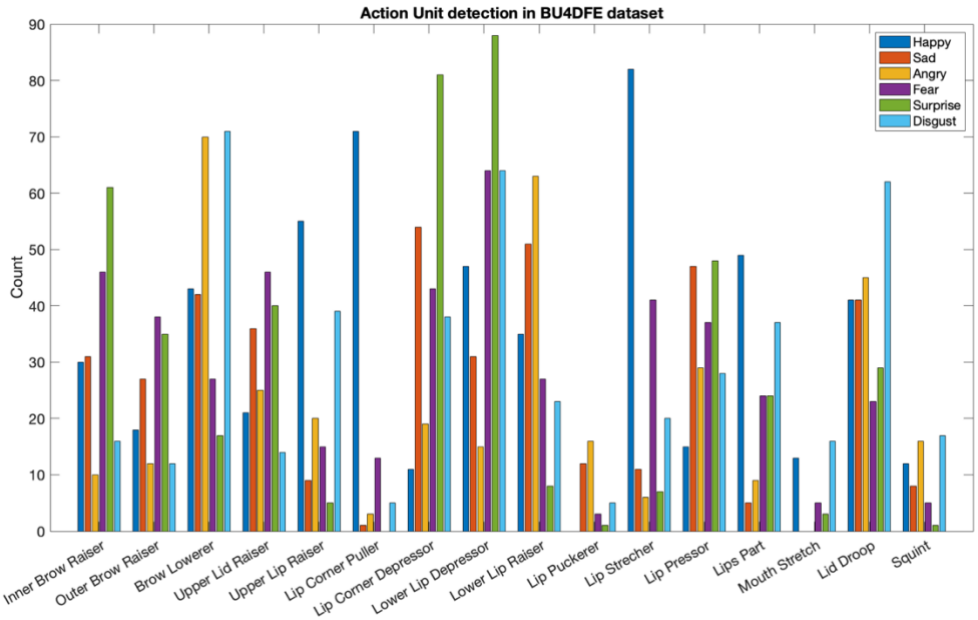


Figure 12. Action Units detected on BU4DFE dataset.

Action Unit	Description	Threshold Size	Relevant Landmarks (See Figure 4)	x-direction	y-direction
AU1	Inner brow raiser	Medium	17, 27	-	Increase
AU2	Outer brow raiser	Small	21, 31	-	Increase
AU4	Brow lowerer	Medium	17, 27, 26, 36	-	Decrease
AU5	Upper lid raiser	Small	3, 11	-	Increase
AU10	Upper lip raiser	Large	51, 52, 53	-	Increase
AU12	Lip corner puller	Large	49, 55	Increase	Increase
AU15	Lip corner depressor	Medium	49, 55	-	Decrease
AU16	Lower lip depressor	Medium	57, 58, 59	-	Decrease
AU17	Chin raiser	Medium	57, 58, 59	-	Increase
AU18	Lip puckerer	Small	51, 53, 57, 59	Decrease	-
AU20	Lip stretcher	Medium	49, 55	Increase	-
AU24	Lip pressor	Medium	52	-	Decrease
AU25	Lips part	Small	52, 58	-	Increase, Decrease
AU27	Mouth stretch	Large	49, 52, 55, 58	Decrease	Increase, Decrease
AU41	Lid droop	Medium	3, 11	-	Decrease
AU42	Squint	Small	3, 7, 11, 15	-	Increase, Decrease

Table 1. Activation Unit identification information.

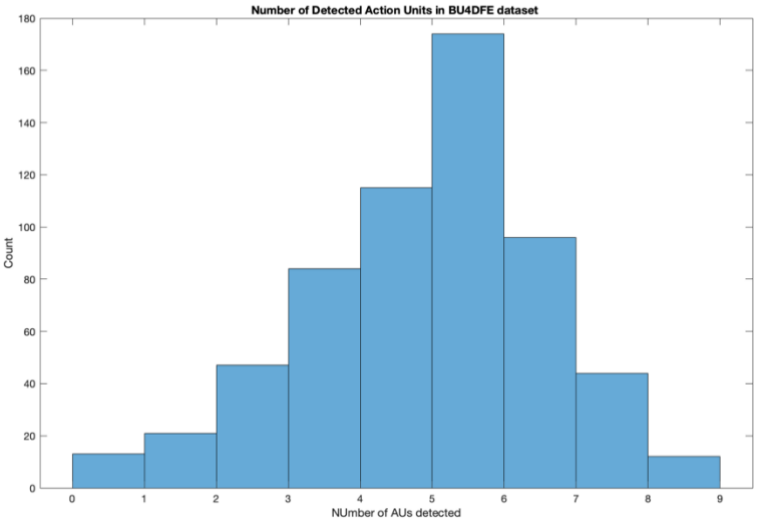


Figure 13. Distribution of the amount of Action Units detected within an expression.

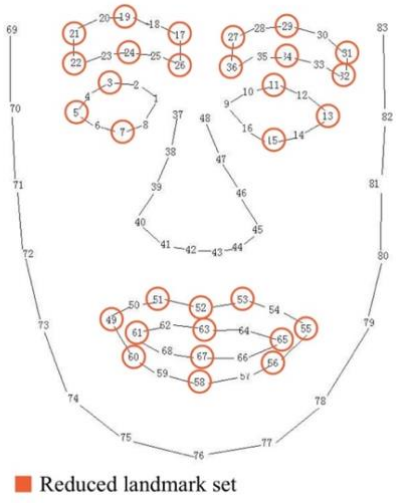


Figure 14. Map of 3D landmarks provided by the dataset.

Figure 12 shows the detected AUs by emotion for the BU4DFE dataset. The emotions Happy and Surprise are shown to be the two most expressive emotions, with the others containing more subtle deformations. Notably, limited AU activations are detected for sad and fearful expressions, and therefore accurate classification with this limited explanatory data is not possible. To overcome this, the classification module has been developed with this in mind.

Figure 13 shows the distribution of the number of detected action units for the data within the BU4DFE dataset. The graph shows that the system is capable of detecting a significant number (> 4) of activations for many samples, however there are still multiple samples in which the AU detection does not yield a significant amount of data, and thus the system cannot rely on this feature extraction method alone.

The FACS system also describes the movements of the cheeks as indictors of emotion expression. Due to the set of landmarks provided, the tracking of their movements is not possible and is thus considered as a future area of development.

4.2. Delta feature extraction

The next set of features investigated are extracted from the movement of a number of key points on the face over time. To track their movements, the landmarks are grouped into local regions: left eye, right eye, left eyebrow, right eyebrow and the mouth. Evaluating the movement of a landmark within this specific group is beneficial because the movements of the landmarks are more descriptive with respect to their specific region, rather than the face as a whole. To track any given landmark, the initial position, L1, the subsequent position, L2, and the (fixed) centre of the local region (i.e., mouth, eye, or eyebrow), C, are used, i.e.,

$$\mathrm{L1} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}, \quad \mathrm{L2} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}, \quad \mathrm{C} = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}. \tag{4.2}$$

Firstly, the Euclidean distance between the two landmark locations in each of the three coordinate directions are calculated, denoted as Δ_R, i.e.,

$$\Delta_1 = \mathrm{L1} - \mathrm{C}\,, \qquad \Delta_2 = \mathrm{L2} - \mathrm{C}\,, \qquad \Delta_R = \mathrm{L1} - \mathrm{L2}\,, \tag{4.3}$$

Since the size of this movement is also descriptive of the expression, the magnitude of the landmark position change is thus calculated, denoted as δ_R. Next the change in distance between the landmark and the region centre is calculated. This value describes the direction of the movement as towards or away from the centre of the region and is useful for differentiating landmarks that act in one direction. For

example, an eyebrow raise will be given a value >1 while an eyebrow droop will be given a value < 1 for landmark 17 (see Figure 4)). This value is denoted as δ_C, i.e.,

$$\delta_1 = |\Delta_1|\,, \qquad \delta_2 = |\Delta_2|\,, \qquad \delta_R = |\Delta_R|\,, \qquad \delta_C = \delta_2 - \delta_1\,. \tag{4.4}$$

Descriptive information is also be extracted from the angles (in radians) relating the two locations, and the region centre. The three angles representing the movement between the landmark's position in two consecutive frames are calculated, Θ_R, as well as the change in angle between the landmark and the centre for each of the frames, Θ_C, i.e.,

$$\Theta_R = \tan^{-1}\begin{bmatrix}\Delta_{Ry}/\,\Delta_{Rx}\\ \Delta_{Rz}/\,\Delta_{Rx}\\ \Delta_{Rz}/\,\Delta_{Ry}\end{bmatrix}, \quad \Theta_C = \tan^{-1}\begin{bmatrix}\Delta_{2y}/\,\Delta_{2x}\\ \Delta_{2z}/\,\Delta_{2x}\\ \Delta_{2z}/\,\Delta_{2y}\end{bmatrix} - \tan^{-1}\begin{bmatrix}\Delta_{1y}/\,\Delta_{1x}\\ \Delta_{1z}/\,\Delta_{1x}\\ \Delta_{1z}/\,\Delta_{1y}\end{bmatrix} \tag{4.5}$$

The total descriptor, Δ_F, thus potentially comprises of three movement values, two distance values, and six angle values for a total of 11 descriptive values for each landmark, i.e.,

$$\Delta_F = [\Delta_R \quad \delta_R \quad \delta_C \quad \Theta_R \quad \Theta_C]^T\,. \tag{4.6}$$

The features for each landmark are concatenated together to form the total descriptor, termed the delta features. When using the full 56 facial landmarks provided, this results in a maximum delta feature length of 616. A subset of the provided landmarks (30 of 56, circled orange in Figure 4) and parameters is proposed to reduce this significantly. A summary of the parameter combinations is shown in Table 2, with an X indicating the sub-feature's inclusion in the feature vector.

Feature ID	Landmark count	Δ_R	δ_R	δ_C	Θ_R	Θ_C	Feature length
1	56	X	X	X	X	X	616
2	30	X	X	X	X	X	330
3	56	X	X	X	X	-	448
4	30	X	X	X	X	-	240
5	56	X	X	X	-	-	280
6	30	X	X	X	-	-	150
7	56	X	-	-	-	-	168
8	30	X	-	-	-	-	90

Table 2. A selection of delta feature subset possibilities, with different resulting vector lengths.

To inform the system about possible anomalies within the data, a warning flag is also integrated within the function. The flag is raised whenever a landmark has moved significantly more than expected (set to a value of 15 in any coordinate direction).

4.2.1. Frame sequences

The delta feature set is derived from the difference between two frames, and the accuracy of subsequent classifications vary significantly depending on the chosen frames. When selecting a set of frames, the features can be calculated in two ways: either between each of the frames and the neutral face, or between consecutive frames. For instance, if feature extraction is performed over frames 1, 25, and 50,

the feature pairs can either be [1, 25] and [25, 50], or [1, 25] and [1, 50]. Both sets of features capture the changes in facial expressions over time, but they observe different deformation dynamics. During classification investigations on both the BU3DFE and BU4DFE datasets, the use of non-adjacent features was found to produce higher recognition accuracies.

4.3. Fusion feature construction

Due to the descriptive nature of both the detected AUs and the delta features, a final concatenation of the respective feature vectors has been utilised to maximise the potential information contained in the final module output. In this case, the delta features (with length 150) are fused together with the AU detection feature (with length 16) to create a final feature vector of length 166, given as the fusion feature, or Σ_F, i.e.,

$$\Sigma_F = \begin{bmatrix} \Delta_F \\ \alpha_F \end{bmatrix}. \tag{4.7}$$

4.4. Expression feature extraction

The work in [22] presents a face model in which the expression deformations of the participant are separated from that of their neutral face. The rationale is that a person's face at any point is a combination a "Mean Face Shape" $\bar{S}$, the person's unique Facial identity S_{ID}, and the deformations S_{exp} as a result of expressing an emotion, i.e.,

$$S = \bar{S} + S_{ID} + S_{exp}\,. \tag{4.8}$$

5. Classification Module

The machine learning models investigated in this project are binary classifiers: they aim to find an optimal hyperplane that separates two classes of data samples in the input data. As such, multiple binary classifiers, called learners, are needed to perform multi-class classification tasks such as FER. This can be implemented in several ways via the use of different coding styles. The styles supported by MATLAB are shown in Table 3. More detailed information can be found in [23].

Coding style	No. of learners' equation	Learners for 6 class FER
One vs. One	$K(K-1)/2$	15
One vs. All	K	6
All binary combinations	$2^{(K-1)}-1$	31
All ternary combinations	$(3^K - 2^{(K+1)} + 1)/2$	301
Dense random	$\sim 10 \log_2 K$	~26
Sparse random	$\sim 15 \log_2 K$	~39

Table 3. MATLAB supported multi-class classifier coding styles, and the number of binary learners associated with each type.

The One vs. One coding style is selected in this project to minimise the complexity of the models whilst maintaining sufficient descriptive power. Further experimentation could be undertaken to further optimise the classification.

5.1. Classification algorithm selection

With the use of MATLAB's integrated Classification Learner Application, a wide range of machine learning algorithms was investigated to determine the most applicable classifier for the input data used in the project. Due to the high accuracies previously obtained with the use of SVMs [5], this class of classifiers was selected for the initial development of the classification module.

5.2. Action unit feature classifier

An SVM with a linear kernel function (i.e., a Linear SVM) was selected as it produces the highest recognition accuracies for the proposed delta features. A minimum activation limit is required during training because any descriptor with less than three activations provides an ambiguous basis for classification (i.e., predicting an emotion based upon an eye squint alone will produce inaccurate results). The score function of the Linear SVM is given by,

$$f(x) = x'\beta + b\,, \tag{5.1}$$

where x is the extracted AU feature vector (observation), β contains the coefficients that define an orthogonal vector to the separating hyperplane and b is the bias term.

5.3. Fusion feature classifier

To classify emotions based on fusion features, an ensemble classifier is used to produce the highest recognition accuracies. The dimensions of the subspace classifier have been set to 30, i.e., using 30 binary learners.

5.4. Adaptive classification module

To enhance the accuracies of the classification module, an adaptive model as shown in Figure 15 is introduced. This module relies on the logic that when a sufficiently large amount of AUs is detected, the AU Classifier produces higher accuracies than the fusion classifier. If, however, the framework fails to detect a significant amount of AUs, the fusion classifier produces better results. To integrate this logic into the module, the AU feature is produced first. If the number of activations is above a specified value (given as 7 in Figure X), then the module will classify the expression on this data alone using the AU Classifier (Linear SVM). In the case that an insufficient number of activations is detected, the module will create the delta feature associated with the input data, and fuse that with the previously constructed activation unit feature. The classification is then performed by the Fusion Classifier.

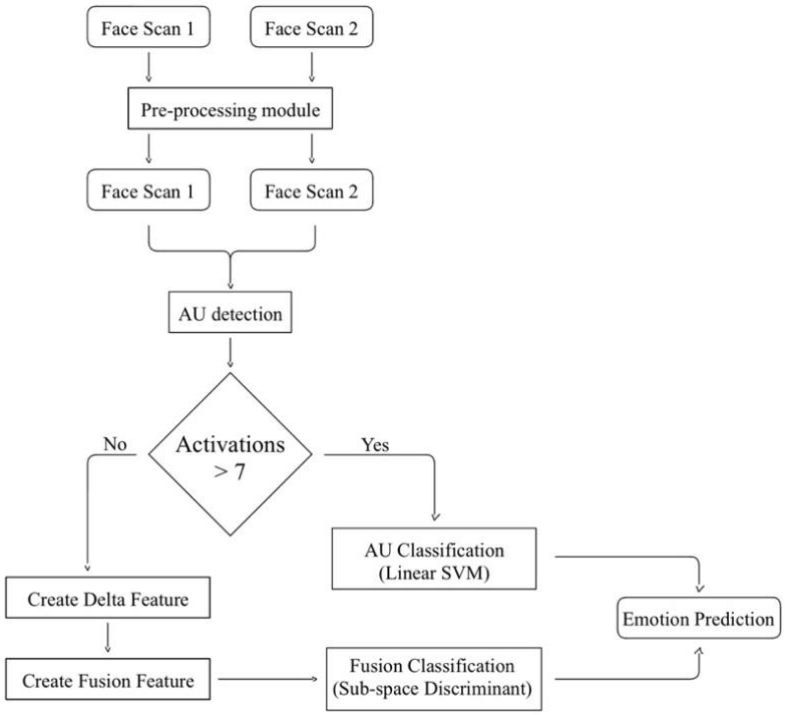


Figure 15. Logic of the adaptive classification.

5.5. Alternative classification methods

As an alternative to the classical feature extraction and classification method, deep learning methods have also been investigated as a novel classification framework. The framework classifies point clouds directly, identifying the spatial relationships within the point clouds.

Due to the novelty of the framework, MATLAB does not support the use of any pre-trained voxel based neural networks, and therefore transfer learning could not be applied to reduce training times and increase the accuracies of the network. Due to the small set of data (~100 instances of each emotion) and the lack of pre-trained models, the networks struggle to obtain classification accuracies greater than

50%, and therefore development was discontinued in favour of the use of more traditional machine learning methods.

6. Adaptive Module Optimisation

To enable the adaptive module to obtain the highest classification accuracies, each of its defining parameters has been investigated across a set of multiple values. The optimisation was carried out in two phases: optimising the amount and clarity of the information contained within the extracted features; and tuning the classification module itself. All optimisation tests were performed with 10-fold cross validation (10-CV) to ensure the validity of the results.

To obtain the training features used to classify the data, it is noted that the threshold values associated with the AU feature have significant impact on the module accuracy. If the thresholds are too high then few AUs will be detected, and classification on such sparse data tends to be highly inaccurate. However, if the thresholds are too low, the activations are vulnerable to noise and the classifier might overfit to the data. The choice of delta feature subset also has a significant influence on the module accuracy.

6.1. Threshold value optimisation

To evaluate the effect of varying threshold values, several values have been evaluated to empirically find the most optimal values. To reduce the computational requirements, the small AU threshold has been fixed at 0.05. The classification accuracies have been tested for 8 values over a range of 0.1-7.1 for both the medium (T_M) and large (T_L) threshold values. These values have been applied to both the creation of the AU features and the fusion features, resulting in an investigation evaluating the accuracies for 4096 permutations of thresholds of the form:

$$T_\alpha = [0.05 \quad T_M \quad T_L], \qquad T_\Sigma = [0.05 \quad T_M \quad T_L], \tag{6.1}$$

where T_α and T_Σ are the threshold values used to compute the AU and fusion feature vectors, respectively. The optimal values are shown in Table 4.

Dataset	Optimal $\boldsymbol{T_\alpha}$	Optimal $\boldsymbol{T_\Sigma}$
BU3DFE	$[0.05 \quad 4.1 \quad 0.1]$	$[0.05 \quad 6.1 \quad 7.1]$
BU4DFE	$[0.05 \quad 2.1 \quad 1.1]$	$[0.05 \quad 4.1 \quad 5.1]$

Table 4. Optimal AU threshold values for BU3DFE and BU4DFE.

6.1.1. Delta subset optimisation

To evaluate the most descriptive delta feature subset, the accuracies of the module have also been determined over a range of values (as detailed in Table 2.) Four primary feature sets have been investigated, which are those detailed in Table 2, with the reduced landmark vector for each subset investigated too. Figures 16 and 17 show the results of the optimisation trials on both the delta and landmark subsets, on both the BU3dFE and BU4DFE datasets. Table 5 summarises the results.

Dataset	Optimal delta subset	Landmark count
BU3DFE	$[\Delta_R \quad \delta_R \quad \delta_C \quad \Theta_R \quad \Theta_C]$	30
BU4DFE	$[\Delta_R \quad \delta_R \quad \delta_C]$	30

Table 5. Optimal delta parameters for BU3DFE and BU4DFE datasets.

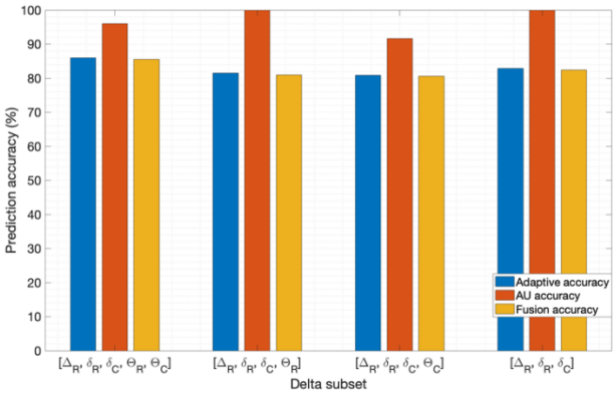


Figure 16. Classifier accuracies with different delta feature subsets on BU3DFE dataset.

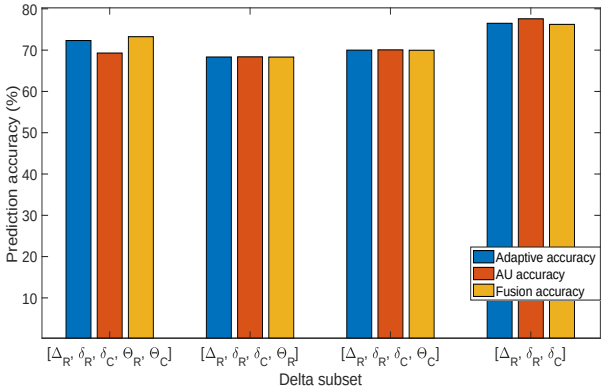


Figure 17. Classifier accuracies with different delta feature subsets on BU4DFE dataset.

The use of PCA [24] is suggested as an area of further development due to the fusion feature being of length 616 under all delta sub-feature inclusions. This would potentially reduce the time taken to classify the frame set. However, it comes at the expense of accuracy due to the nature of PCA (that the data is reduced in complexity).

6.2. Adaptive module parameter optimisation

The classification module also has a parameter to optimise. The AU limit at which the module performs the classification on the AU feature alone has an effect the accuracy. Figures 18 and 19 display the accuracies of the adaptive classifier module when tested with different activation limit values. For the BU3DFE dataset, the accuracy of the AU classifier is seen to be less effective than that for the BU4DFE dataset, and thus the best module performance is seen when the data is classified on the fusion features alone (i.e., the activation limit is set to 10). For the BU4DFE dataset, the Adaptive

classifier performs the best with an AU limit of 6.

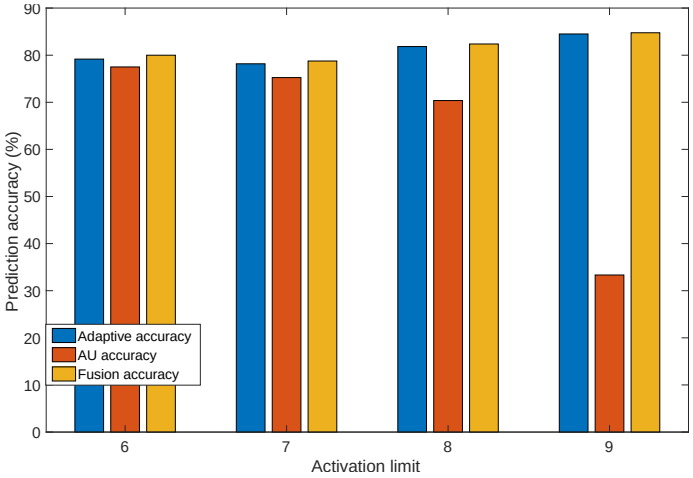


Figure 18. Classification accuracies under varying activation limits on BU3DFE dataset.

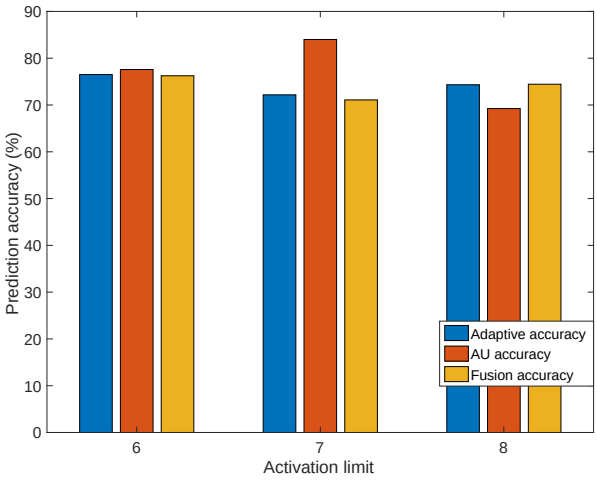


Figure 19. Classification accuracies under varying activation limits on BU4DFE dataset.

Facial Emotion Recognition Results

7. Results

The BU3DFE and BU4DFE datasets contain data on 100 and 101 participants, respectively. All results have been tested on all available participants under a 10-fold cross validation (10-CV) scheme to ensure the validity of the results.

7.1. BU3DFE dataset

The BU3DFE dataset provides emotions on a scale of 4 intensities from a minimally (1) to a maximally (4) expressive face. Unsurprisingly, Figure 20 shows that the optimal recognition accuracies obtained when classifying emotions of intensity 4, and as the intensity decreases so does the accuracy of the classifier. The best classification results, and corresponding module parameters, are detailed in Table 6. The data split is the percentage of samples classified on the AU classifier. The training data was created from 90 of the 100 participants, utilising only the most intense expressions. The classifier was subsequently validated on the remaining 10 unseen participants' expressions under a 10-CV scheme.

Adaptive accuracy	Fusion accuracy	AU accuracy	Data split
84.5%	84.76%	33.3%	5.0%
AU Limit	Expression intensity	Landmark count	Feature subset
9	4	30	$[\Delta_R \quad \delta_R \quad \delta_C \quad \Theta_C]$

Table 6. Adaptive module validation accuracies on BU3DFE dataset.

7.1.1. Intensities

As the intensity of the expression increases, it is no surprise that that accuracy of the framework also increases. This is a combination of the AUs becoming more prevalent, and thus more likely to be correctly identified, as well as a larger variation in the delta features, with the distinctions between different classes becoming more defined. The recognition accuracies for each intensity are shown in Figure 21.

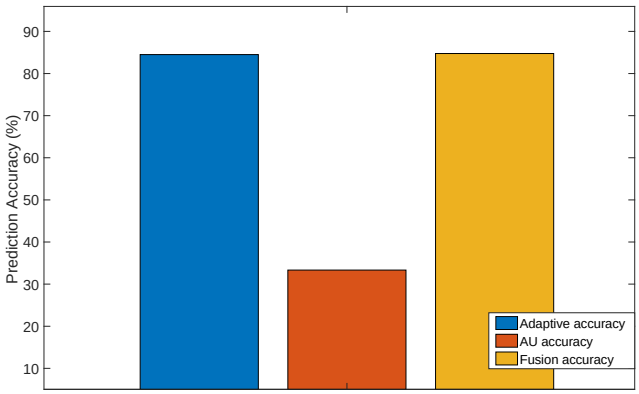


Figure 20. Optimal Adaptive module accuracies on BU3DFE dataset.

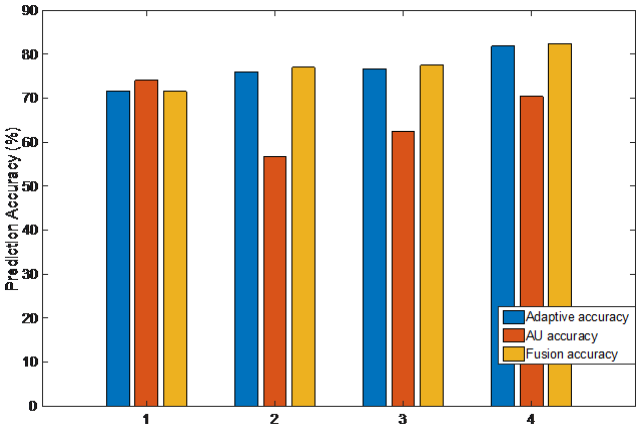


Figure 21. Recognition accuracies for varying expression intensities on BU3DFE dataset.

The ROC curve and confusion matrix shown in Figure 22 further outline the performance of the adaptive framework. The classifier has a high recognition performance across all emotions, and performs particularly well for Happy, Sad and Surprise emotions.

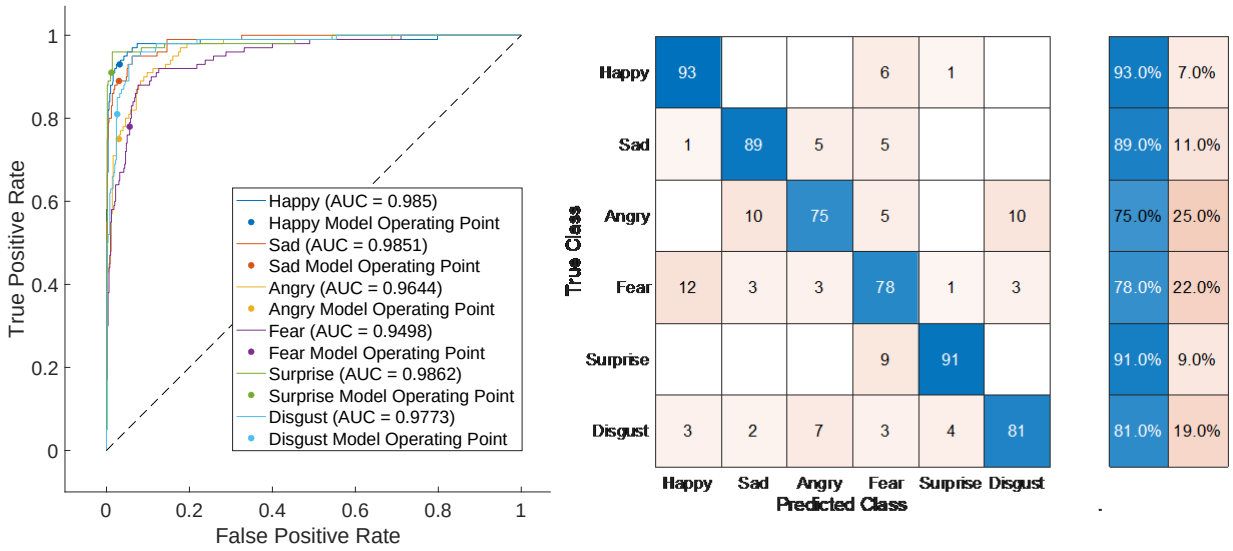


Figure 22. ROC performance curves (left) and confusion matrix (right) for the classification module on the BU3DFE dataset.

7.2. BU4DFE dataset

To perform a similar evaluation of the framework on a 4D dataset, the landmark change between two of the frames within the provided sequences are used. While this does not utilise the additional descriptive information contained within temporal evolution of the emotion, it does act as further validation towards the system efficacy as a whole. To select the first of the two frames, a selection algorithm is applied to the entire sequence to find the "most neutral frame" within the sequence. This enables the framework to operate in the same fashion as used in the 3D classification, where the expression is compared to a neutral face. The second frame is selected as the 50th frame within the sequence, as this is theoretically the time at which the expression is the most intense.

The results, detailed in Table X and shown visually Figure 25, show that the classifier has obtains slightly lower recognition accuracies than obtained on the 3D dataset. However, the adaptive classifier performs far better on this dataset, and suggests that the activation unit detection module provides far more descriptive data when detecting over time.

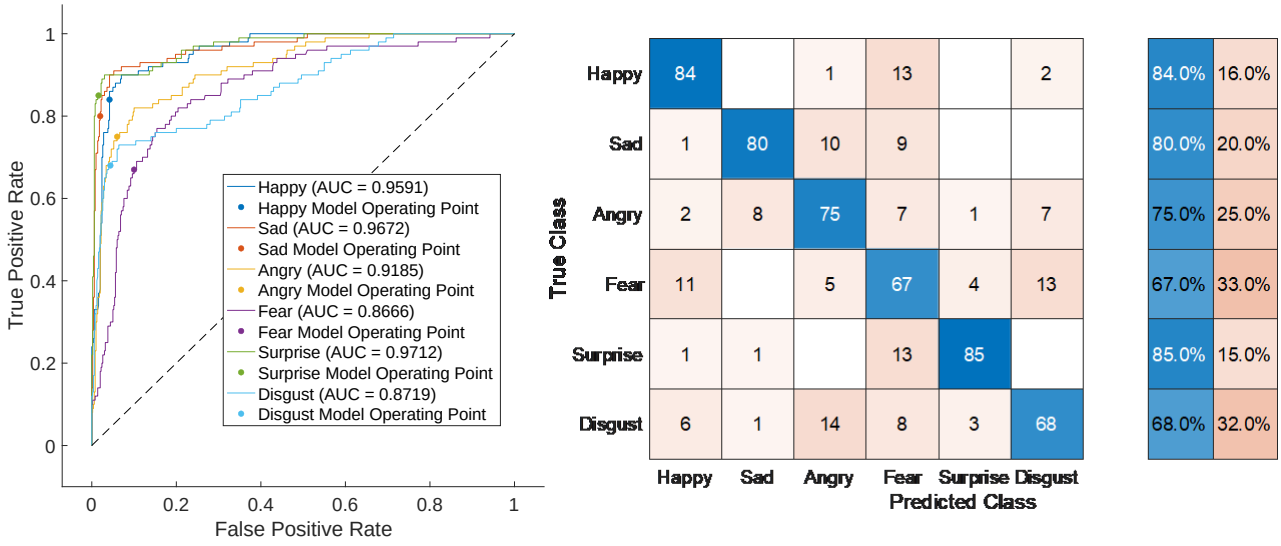


Figure 23. ROC performance curves (left) and confusion matrix (right) for the classification module on BU4DFE dataset.

Adaptive accuracy	Fusion accuracy	AU accuracy	Data split
76.5%	76.24 %	77.6%	19.3%
AU Limit	Landmark count		Feature subset
6	30		$[\Delta_R \quad \delta_R \quad \delta_C]$

Table 7. Adaptive module validation accuracies on BU3DFE dataset.

7.3. Prediction times

To understand the computational efficiency of the program, the time taken to perform the primary functions of each module of the framework has been computed and presented in Table 7. Due to the

frequent use of the computationally expensive ICP algorithm, the pre-processing module is over 3 times slower than any other. This being said, both the AU detection and delta feature construction operates very fast and are therefore readily applicable to a real-world detection framework.

Pre-processing (s)	AU detection (s)	Delta feature construction (s)	Prediction time (s)
0.4887	1.4517e-04	1.3500e-04	0.1589
Total Classification time (s)			
0.6479			

Table 7. Computation time for the framework's primary functions.

Software Documentation

8. Software

The software is broken down into the three modules of the classification framework. This section details each function within the framework in terms of its purpose, and inputs and outputs and their respective datatypes. Many of the functions in the framework also have an optional "Demo" flag that can be set to return intermediary values to provide more detail on what is being performed within the function. Three DEMO MATLAB live scripts have also been produced to allow a fast comprehension of the framework logic and processes.

8.1. Processing functions

The functions in Section 10.1 apply to the raw data to enable accurate subsequent classification. Most of the functions take a point cloud, *ptCloudIn*, and perform an augmentation action upon it, producing the output, *ptCloudOut*. In any function that the input point cloud undergoes any geometric transformation (i.e., translations or rotations), the landmarks are also input, and undergo identical transformations to ensure that the registration between the pointsets is maintained. ?Table 1? provides an overview of the processing functions, which are each individually detailed below.

8.1.1. downsampleFace.m

This function reduces the number of points within the input point cloud. The default number of point in the output cloud is set to 20,000. However, this can be specified by naming the key-value pair "pointCount".

Variable Name	Input/Output (default value)	Variable Details
ptCloudIn	Input	pointCloud
pointCount	Optional input (default = 20,000)	pointCloud
ptCloudOut	Output	pointCloud

Table 8. Downsampling function details.

8.1.2. upsampleFace.m

This function increases the number of points within the point cloud, by interpolating new points between the existing points. The DEMO flag outputs a plot of the triangulation mesh, with the centre of each triangle marked in red, shown in Figure 6.

Variable Name	Input/Output (default value)	Variable Details
ptCloudIn	Input	pointCloud
pointCount	Optional input (20,000)	pointCloud
ptCloudOut	Output	pointCloud

Table 9. Upsampling function details.

8.1.3. noiseFilter.m

The noise filter function removes any outliers from the surface of the input point cloud. The function returns the filtered cloud, and optionally both the indices of the inlier points and the outlier points. The DEMO flag outputs a figure with the inlier points shown in white, whilst the outlier points are shown in red.

Variable Name	Input/Output (default value)	Variable Details
ptCloudIn	Input	pointCloud
NumNeighbours	Optional input (10)	integer
Mag	Optional input (2)	number
ptCloudOut	Output	pointCloud
inIdx	Optional output	Nx1 column vector
outIdx	Optional output	Nx1 column vector

Table 10. Noise filter function details.

8.1.4. normalisePose.m

This function aligns an input point cloud to a reference point cloud, using the ICP algorithm. The output is the optimal registration of the input to the reference, along with the optional 3D transformation matrix (of size 4x4) that maps the input to the output.

Variable Name	Input/Output (default value)	Variable Details
ptCloudIn	Input	pointCloud
ptCloudRef	Input	pointCloud
SampleRate	Optional input (1)	Range (0, 1]
ptCloudOut	Output	pointCloud
Transform	Optional output	rigidtfom3d

Table 11. Pose normalisation function details.

8.1.5. voxelizeCloud.m

This function takes a point cloud as an input and outputs its representation in discrete space (3D image). if the optional “Solid” flag is set, then all points within the mask are given a value of 1, and the output becomes a face volume.

Variable Name	Input/Output (default value)	Variable Details
ptCloudIn	Input	pointCloud
GridSize	Optional input ([50, 50, 32])	3x1 row vector
Solid	Optional input (false)	Boolean value
voxelImage	Output	logical array

Table 12. Voxelization function details.

8.1.6. findNoseTip.m

This function finds the location of the nose tip within the input point cloud, the process is detailed is Section 3.3. The DEMO flag returns multiple figures. Firstly, the registration of the central line to that of the reference is shown. Secondly the input face and its plane of symmetry is shown. Next the input, along with the selection of candidate nose tips are displayed. Finally, the input and the selected nose tip are shown together.

Variable Name	Input/Output (default value)	Variable Details
ptCloudIn	Input	pointCloud
landmarksIn	Input	pointCloud
noseTip	Input	1x3 row vector
ptCloudOut	Output	pointCloud
landmarksOut	Output	pointCloud

Table 13. Nose tip identification function details.

8.1.7. centreFace.m.m

This function translates all points within *ptCloudIn* and *landmarksIn* such that the nose tip, and thus the face and landmarks are all centred on the origin.

Variable Name	Input/Output (default value)	Variable Details

ptCloudIn	Input	pointCloud
landmarksIn	Input	pointCloud
centralRef	Optional input	Boolean value
SampleRate	Optional input (1)	logical array

Table 14. Face centring identification function details.

8.2. Feature extraction functions

These functions create descriptive feature vectors given two sets of landmarks. They have multiple parameters that can be specified to modify the outputs to favour certain requirements (i.e., use a subset of the landmarks to reduce the length of the feature vector).

8.2.1. detectAU.m

This function detects the activation of 16 different AUs defined in FACS. The output is a 16x1 AU feature vector, α_F (See Equation 4.1). The *thresholds* variable specifies the thresholds to use for the detections, in the form [small, medium, large].

Variable Name	Input/Output (default value)	Variable Details
lm1	Input	pointCloud
lm2	Input	pointCloud
thresholds	Optional input ([1, 1, 1])	1x3 row vector
activations	Output	16x1 feature vector

Table 15. Action unit detection function details.

8.2.2. deltaFeatureV4.m

This function composes the delta feature that represents the two landmark locations. The *Subfeatures* optional input specifies which sub-features to include in the vector. The array acts as a logical index to the full feature set of $[\Delta_R \quad \delta_R \quad \delta_C \quad \Theta_R \quad \Theta_C]$ (i.e., a value of [1, 0, 1, 0, 1] will create the feature vector using the set of sub-features: $[\Delta_R \quad \delta_C \quad \Theta_C]$). The *Subset* optional argument specifies whether to use the subset of 30 out of the total 56 relevant landmarks.

Variable Name	Input/Output (default value)	Variable Details
lm1	Input	pointCloud
lm2	Input	pointCloud
Subfeatures	Optional input ([1, 1, 1, 1, 1])	1x5 row vector
Subset	Optional input (false)	Boolean value
features	Output	Nx1 feature vector
warningFlag	Optional output	Boolean value, check output if true

Table 16. Delta feature extraction function details.

8.3. Classification functions

These functions define the AU Classifier and the Fusion Classifier. Each specifies the parameters for their respective machine learning models to be trained on the predictor and response data.

8.3.1. LinearAU.m

This function trains a multi-class SVM with a linear kernel function to predict the emotion of from the activation unit feature vectors. It thus takes in predictor data matrix of size Mx16 (where M is the number of training samples) and a response vector of size Mx1 containing the true class of each input. The code has been adapted from that produced by the Classification Learner App integrated within MATLAB.

Variable Name	Input/Output (default value)	Variable Details
trainingData	Input	Mx16 matrix
responseData	Input	Mx1 categorical vector
trainedClassifier	Output	1x5 row vector
validationAccuracy	Optional output	Range [0, 1]

Table 17. AU classifier function details.

8.3.2. AdaptiveSubspaceFuse.m

This function trains a multi-class ensemble classifier to predict the emotion from a fusion feature vector. This function accepts feature vector of any size to enable testing of the different combinations of delta sub-features.

Variable Name	Input/Output (default value)	Variable Details
trainingData	Input	MxN matrix
responseData	Input	Mx1 categorical vector
Workers	Optional input (30)	
Dimension	Optional input (30)	
trainedClassifier	Output	1x5 row vector
validationAccuracy	Optional output	Range [0, 1]

Table 18. Fusion classifier function details.

FileName	Inputs					Description	Outputs				
	Required			Optional			Primary			Secondary	
exmple.m	input1			optional input1		Example function	output1			extra	
centreFace.m	inCloud	landmarks	noseTip			translates the input cloud to place the nose tip on the origin in 3D space (0, 0, 0)	outputCloud	landmarks			
downsampleFace.m	inCloud			pointCount		Removes points within the cloud to output a point cloyd containing the specified number of points	outputCloud				
noiseFilter.m	inCloud			NumNeighbours	Mag	Smooths the surface of the input point cloud by checking the density of neighborinfg points	outputCloud			InIdx	outIdx
normaizePose.m	inCloud	landmarks	refCloud	SampleRate		Alligns the input point cloud to the reference	outputCloud	transform		error	
upsampleFace.m	inCloud			pointCount		increases the amount of points within the input cloud via linear interpolation of	outputCloud				
voxelizeCloud.m	inCloud			gridSize	solid	Transforms the input point cloud (contiuous space) into a 3D image represenation (discrete space)	outputCloud	landmarks			
findNoseTip.m	inCloud	landmarks (pointCloud)		CentralRef	SampleRate	Finds the nose tip of the face from a face scan.	allignedFace	landmarks	noseTip		
extractFacialRegion.m	inCloud	noseTip		SampleRate		Extracts all points which are within the specified radius fron the nose tip					
getNeutralIdx.m	candidateFaces	refFace				Finds the "most neutral" face amoung the input candidates	neutralFace			idx (double)	

Table 19. Point cloud processing function overview.

Analysis and Conclusion

9. Analysis

The framework has satisfied all of the system requirements set out at in the initial phase of the project planning. On both tested datasets, the accuracies of the optimised adaptive classification module have achieved recognition accuracies above the initially specified 75%, with the performance on BU3DFE being significantly better than this value.

As highlighted by [5], the direct comparison of recognition accuracies between similar SOTA frameworks presents a significant challenge, and a large degree of care must be taken to ensure that any such comparison fully details any limitations that may apply. Firstly, even when framework performance is evaluated on any particular dataset, there are multiple differing methods of selecting the data to perform the analysis on. For example, the BU3DFE dataset provides the expressions of each Eckmans' 6 emotions, at 4 levels of intensity (low = 1, high = 4), as well as a neutral face, for 100 subjects (44 male and 56 female). No clear or accepted set of data to test and train the classification systems on; nearly all studies simply neglect the lowest two intensity expressions, while multiple chose to randomly select a subset of 60 participants from the data provided. In the proposed system, all 100 participants data was used, however at only the most intense level of expression (4). The accuracy validation schemes employed by the systems within the literature also makes the direct comparison of figures more difficult. 10-fold cross validation (10-CV) is the most common method reported to ensure that the classifier performance is not inflated by chance. Despite 10-CV appearing to reach some level of an accepted standard, some papers neglect to mention any cross-validation schemes, and others employ variations of the 10-CV scheme (e.g., 5-fold CV, 20-fold CV, 100x10-CV, etc.). As a result of these factors, **Table X** is presented to verify that the proposed system has produced classification accuracies of comparable to the top 5 SOTA frameworks as identified in [5], however direct comparison should be avoided because of the complications outlined above. The Subjects column indicates how many of the 100 participants were used.

FER study	Subjects	Expression intensities	Classifier	Validation strategy	Accuracy (%)
Hariri et al. [25]	100	3, 4	RBF-SVM	100 × 90%/10%	92.62
Sheng et al. [13]	60	3, 4	SVM	100 × 10-CV	92.1
Li et al. [26]	60	4	DF-CNN (svm)	100 × 10-CV	91.3
Yurtkan and Demirel [27]	100	4	RBF-SVM	100 × 90%/10%	90.8
An and Ruan [28]	100	4	SVM with EPE	100 × 10-CV	90.1
Proposed method	100	4	Adaptive Classifier	10-CV	84.5

Table 20. BU3DFE SOTA classification accuracies.

The accuracies in the classification of each individual emotion are also worth analysing. Happy and Surprised emotions consistently achieve the highest recognition accuracies within the literature. This is likely due to multiple factors, including the relative expressiveness of the emotions (i.e., large facial deformations compared to other emotions such as Sadness) and the less ambiguous nature of the emotions compared to those such as Fear.

The testing done on the BU4DFE dataset (and the development of the neutral face identification system) provides a clear indication that the framework can be applied to less rigorously constrained data, and thus validates the potential of the system to be applied to real world FER tasks. Further work could enable the system to capture more of the descriptive information that can be extracted from monitoring expressions over time.

The delivery of the target computational performance (emotion classification in under 1 second) has also been substantially exceeded, taking 0.65 seconds to output the predicted emotion. A review in terms of computational optimisation could be undertaken to further reduce this, however the current performance is considered satisfactory for most applications. Within the produced code, there are demo options for multiple functions within each module, as well as DEMO_ files that present fast and clear visual representations of the processes being executed within the function/module. The entire source code has also been provided in both Appendix 1 and as a downloadable directory, enabling open access for further development or external collaboration.

Conclusion

10. Conclusion

The project has succeeded in producing multiple significant contributions to the current field of 3D (and 4D) emotion recognition. A novel feature representation of 3D face sequences has been produced, termed Fusion Features, along with an innovative adaptive classification module that is capable of dynamically selecting the optimal classification method for the input data. A method for detecting the presence of Action Units described by the FACS is also presented. A full implementation of the presented framework, including a customisable pre-processing module, has also been implemented in MATLAB and will be publicly released as a further contribution to the field. The augmented version of the BU4DFE dataset used for testing of the framework is also published too, due to the large amount of benefits that it has over the original; this dataset has extracted, pre-processed, and stored the data in a far more convenient and efficient manner for MATLAB based systems. One key advantage is that this

dataset is far smaller in size (under 10GB) compared with the original which is around 560GB. This enables its storage on local hard drives which are typically far faster when running multiple performance tests.

The project has produced an emotion classification framework that has been applied to the task of FER and performs comparably to the standards set within the literature. The potential deployment of the system into edge devices for real world applications has also been validated, however this provided in the capacity as a base system for further development, with the aim of producing higher recognition accuracies.

11. Further Works

Due to the primary motivation of the project being the frameworks' application in real-world systems, the key next step in the development of this framework is its deployment as a real time emotion recognition application. This was a significant challenge due to the framework utilising 3D depth scans as its input, and no access to similar hardware for testing was available. To overcome this issue, a transition to 2D data as the initial input, followed by a projection into 3D space is suggested. The integration of the MediaPipe Face Mesh solution [29] is thought of as the best way to implement this. This transition would also enhance the ability of the system to be deployed to a far greater number of devices, due to the limited availability and high cost of 3D imaging cameras, compared with the ubiquity of 2D imaging cameras in modern devices.

When testing, the BU3DFE and BU4DFE datasets use different coordinate scales, and has different depth scan resolutions. During the optimisation process, the values for each dataset were empirically optimised, however this would cause a significant problem in deployment on multiple different hardware platforms. An expansion of the pre-processing module to further normalise any input data in terms of axis scaling and resolutions (point densities) is suggested.

A more sophisticated set of emotions is also key to suggested to enhance the ability of the framework to a to be applied to a greater selection of engineering problems.

References

[1] Taleb Alashkar, Boulbaba Ben Amor, Mohamed Daoudi, Stefano Berretti. Spontaneous Expression Detection from 3D Dynamic Sequences by Analyzing Trajectories on Grassmann Manifolds. IEEE Transactions on Affective Computing, Institute of Electrical and Electronics Engineers, 2018, 9 (1), pp.271-284. 10.1109/TAFFC.2016.2623718 . hal-01520751

[2] Miao X, Xue C, Li X, Yang L. A Real-Time Fatigue Sensing and Enhanced Feedback System. Information. 2022; 13(5):230. https://doi.org/10.3390/info13050230

[3] https://www.gov.uk/government/statistics/reported-road-casualties-in-great-britain-provisional-estimates-year-ending-june-2022

[4] Yang C, Wu L, Tan K, Yu C, Zhou Y, Tao Y, Song Y. Online User Review Analysis for Product Evaluation and Improvement. Journal of Theoretical and Applied Electronic Commerce Research. 2021; 16(5):1598-1611. https://doi.org/10.3390/jtaer16050090

[5] Gilderlane Ribeiro Alexandre, José Marques Soares, George André Pereira Thé, Systematic review of 3D facial expression recognition methods, Pattern Recognition, Volume 100, 2020, 107108, ISSN 0031-3203, https://doi.org/10.1016/j.patcog.2019.107108.

[6] Muhammad Sajjad et Al, A comprehensive survey on deep facial expression recognition: challenges, applications, and future guidelines, Alexandria Engineering Journal, Volume 68,2023, Pages 817-840, ISSN 1110-0168, https://doi.org/10.1016/j.aej.2023.01.017.

[7] University of Binghamton (2022), Analyzing Facial Expressions and Emotions in Three Dimensional Space with Multimodal Sensing, http://www.cs.binghamton.edu/~lijun/Research/3DFE/3DFE_Analysis.html (Accessed: 19 October 2022)

[8] BU4D ref

[9] Dmytro Derkach, Federico M. Sukno, Automatic local shape spectrum analysis for 3D facial expression recognition, Image and Vision Computing, Volume 79, 2018, Pages 86-98, ISSN 0262-8856, https://doi.org/10.1016/j.imavis.2018.09.007.

[10] Ming, Yue. (2015). Robust regional bounding spherical descriptor for 3D face recognition and emotion analysis. Image and Vision Computing. 35. 10.1016/j.imavis.2014.12.003.

[11] Zhang, Z. (2014). Iterative Closest Point (ICP). In: Ikeuchi, K. (eds) Computer Vision. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-31439-6_179

[12] Q. Zhen, D. Huang, Y. Wang, L. Chen, Muscular movement model-based auto- matic 3D/4D facial expression recognition, IEEE Trans. Multimed. 18 (7) (2016) 1438–1450, doi:10.1109/TMM.2016.2557063.

[13] N. Sheng, Y. Cai, C. Zhan, Changyan Qiu, Yize Cui, Xurong Gao, 3D facial expres- sion recognition using distance features and LBP features based on automat- ically detected keypoints, in: 2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), IEEE, Datong, 2016, pp. 396–401, doi:10.1109/CISP- BMEI.2016.7852743.

[14] E. Pei, M. C. Oveneke, Y. Zhao, D. Jiang and H. Sahli, "Monocular 3D Facial Expression Features for Continuous Affect Recognition," in IEEE Transactions on Multimedia, vol. 23, pp. 3540-3550, 2021, doi: 10.1109/TMM.2020.3026894.

[15] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines," in IEEE Intelligent Systems and their Applications, vol. 13, no. 4, pp. 18-28, July-Aug. 1998, doi: 10.1109/5254.708428.

[16] Rabiner, Lawrence R. and B. H. Juang. “An introduction to hidden Markov models.” IEEE ASSP Magazine 3 (1986): 4-16.

[17] Tharwat, Alaa & Gaber, Tarek & Ibrahim, Abdelhameed & Hassanien, Aboul Ella. (2017). Linear discriminant analysis: A detailed tutorial. Ai Communications. 30. 169-190,. 10.3233/AIC-170729.

[18] Ramya, R., Mala, K. & Selva Nidhyananthan, S. 3D Facial Expression Recognition Using Multi-channel Deep Learning Framework. Circuits Syst Signal Process 39, 789–804 (2020). https://doi.org/10.1007/s00034-019-01144-8

[19] MATLAB. Create exhaustive nearest neighbor searcher. MathWorks Help Centre. Accessed: 10/10/2022. https://uk.mathworks.com/help/stats/exhaustivesearcher.html

[20] MATLAB. Spatially bin point cloud points. MathWorks Help Centre. Accessed: 02/12/2022. https://uk.mathworks.com/help/vision/ref/pcbin.html?s_tid=doc_ta

[21] Prince, Emily B. et al. “Facial Action Coding System.” (2015).

[22] B. Chu, S. Romdhani and L. Chen, "3D-Aided Face Recognition Robust to Expression and Pose Variations," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 1907-1914, doi: 10.1109/CVPR.2014.245.

[23] MATLAB. Fit multiclass models for support vector machines or other classifiers. MathWorks Help Centre. Accessed: 17/01/2023. https://uk.mathworks.com/help/stats/fitcecoc.html?s_tid=doc_ta

[24] I. T Jolliffe and J. Cadima, Principal component analysis: a review and recent developments, (2016), Phil. Trans. R. Soc. A.3742015020220150202

[25] W. Hariri, H. Tabia, N. Farah, A. Benouareth, D. Declercq, 3D facial expression recognition using kernel methods on Riemannian manifold, Eng. Appl. Artif. Intell. 64 (C) (2017) 25–32, doi:10.1016/j.engappai.2017.05.009.

[26] X. Li, Q. Ruan, G. An, Y. Jin, R. Zhao, Multiple strategies to enhance automatic 3D facial expression recognition, Neurocomputing 161 (C) (2015) 89–98, https://doi.org/10.1016/j.neucom.2015.02.063

[27] K. Yurtkan, H. Demirel, Entropy-based feature selection for improved 3D fa- cial expression recognition, Signal Image Video Process. 8 (2) (2014) 267–277, doi:10.1007/s11760- 013- 0543- 1.

[28] S. An, Q. Ruan, 3D facial expression recognition algorithm using local thresh- old binary pattern and histogram of oriented gradient, in: 2016 IEEE 13th In- ternational Conference on Signal Processing (ICSP), IEEE, 2016, pp. 265–270, doi:10.1109/ICSP.2016.7877838.

[29] Lugaresi, Camillo et al. “MediaPipe: A Framework for Building Perception Pipelines.” ArXivabs/1906.08172 (2019): n. pag.

Appendix

The functions below present an overview of the output code files produced. The full directory of models, saved data, demo files and other developed software is provided in the logbook and also available at the following link:

https://livewarwickac.my.sharepoint.com/:f:/g/personal/u2004629_live_warwick_ac_uk/EhJjK-0GV_BPoXufQwg1WBwBaExPMn7zWSS4RuKurrEqjQ?e=KaXh2b

```
function [ptCloudOut, transform, error] = normalizePose(ptCloudIn, ptCloudRef, options)
    % This function finds the mapping tranform to map inCloud to refCloud
    arguments
        ptCloudIn pointCloud
        ptCloudRef pointCloud
        options.SampleRate {mustBeInRange(options.SampleRate, ...
                            0, 1, "exclude-lower")} = 1
        options.Demo = false;
    end

    if options.SampleRate ~= 1
        inCloudDS = pcdownsample(ptCloudIn, "random", options.SampleRate);
        refCloudDS = pcdownsample(ptCloudRef, "random", options.SampleRate);
    else
        inCloudDS = ptCloudIn;
        refCloudDS = ptCloudRef;
    end

    [transform, ~, error] = pcregistericp(inCloudDS,refCloudDS);

    ptCloudOut = pctransform(ptCloudIn, transform);

    if options.Demo
        figure;
        subplot(1, 2, 1)
        pcshow(ptCloudRef.Location, [1, 1, 1])
        subplot(1, 2, 2)
        pcshow(ptCloudRef.Location, [1, 1, 1])
        hold on
        pcshow(ptCloudOut)
        hold off
    end

end
```

```
%% Noise Filtering

function [ptCloudOut, inIdx, outIdx] = noiseFilter(ptCloudIn, options)
    arguments
        ptCloudIn
        options.NumNeighbours = 10
        options.Mag = 2
        options.Demo = false
    end

    [~, dists, valids] = multiQueryKNNSearchImpl(ptCloudIn, ptCloudIn.Location,
options.NumNeighbours+1);
    % This multi-query KNN search uses exact search so every query should
    % return the same number of neighbors if the query is a valid point
    actualNums = double(max(valids(:))-1);
    meanDist = sum(dists(2:actualNums+1, :),1)/actualNums;

    meanD = mean(meanDist);

    inIdx = meanDist <= options.Mag*meanD;
    outIdx = ~inIdx;

    % Select the inliers
    points = ptCloudIn.Location(meanDist <= options.Mag*meanD, :);

    ptCloudOut = pointCloud(points);

    if options.Demo
        figure
        pcshow(ptCloudIn.Location(inIdx, :), [1, 1, 1])
        hold on
        pcshow(ptCloudIn.Location(outIdx, :), [1, 0, 0])
        hold off
    end
end
```

```
function ptCloudOut = extractFacialRegion(ptCloudIn, noseTip, options)
    arguments
        ptCloudIn pointCloud
        noseTip
        options.SampleRate {mustBeInRange(options.SampleRate, ...
                            0, 1, "exclude-lower")} = 1
        options.Demo = false
    end


    if options.SampleRate ~= 1
        ptCloudIn = pcdownsample(ptCloudIn, "random", sampleRate);
    end

    MdlES = ExhaustiveSearcher(ptCloudIn.Location);
    cropIndices = rangesearch(MdlES, noseTip, 100);
    cropIndices = cell2mat(cropIndices);

    ptCloudOut = pointCloud(ptCloudIn.Location(cropIndices, :));

    if options.Demo
        figure
        pcshow(ptCloudIn.Location, [1, 1, 1])
        hold on
        pcshow(ptCloudOut)
    end

end
```

```
function [neutralFace, idx] = getNeutralIndex(candidateFaces, options)
    arguments
        candidateFaces (1, :) pointCloud
        options.Method {mustBeMember(...
                    options.Method,["MinError", "Mean"])} = "MinError"
        options.RefFace = nan;
        options.Demo = false;
    end

    pointCount = length(candidateFaces(1).Location);

    for i = 2:length(candidateFaces)
        if length(candidateFaces(i).Location) > pointCount
            candidateFaces(i) = downsampleFace(candidateFaces(i), pointCount);
        elseif length(candidateFaces(i).Location) < pointCount
            candidateFaces(i) = upsampleFace(candidateFaces(i), pointCount);
        end
    end

    if options.Method == "MinError"
        if isa(options.RefFace, "struct")
            load("MaleReference.mat", "refFaceScanM")
            options.RefFace = refFaceScanM;

        end

        errors = zeros(1, length(candidateFaces));
        for j = 1:length(candidateFaces)
            [~, ~, e] = normalizePose(candidateFaces(j), options.RefFace);
            errors(j) = e;
        end

        idx = find(errors == min(errors), 1);

        if options.Demo

            fprintf("Selecting face %d", idx)
        end

        neutralFace = candidateFaces(idx);
    else
        neutralPoints = zeros(pointCount, 3, length(pointCount));
        for j = 1:length(candidateFaces)
            neutralPoints(:, :, j) = candidateFaces(j).Location;
        end

        neutralPoints = mean(neutralPoints, 3);

        neutralFace = pointCloud(neutralPoints);
    end
end
```

```
function ptCloudOut = downsampleFace(ptCloudIn, pointCount)

    arguments
        ptCloudIn pointCloud
        pointCount double = 20000
    end

    IDs = round(linspace(1, length(ptCloudIn.Location), pointCount));

    ptCloudOut = pointCloud(ptCloudIn.Location(IDs, :));

end
```

```
function [processedFace, landmarks] = processCloud(inCloud, landmarks, refCloud,
refSilentCloud)

    arguments
        inCloud
        landmarks
        refCloud
        refSilentCloud
    end

    tic
    % Coarse Allignment
    [normalScan, tform1, e1] = normalizePose(inCloud, refCloud, "SampleRate", 0.2);
    coarseTime = toc;

    landmarks = pctransform(landmarks, tform1);

    tic
    normalScan = noiseFilter(normalScan, "NumNeighbours", 20, "Mag", 20);
    filterTime = toc;

    % Find Nose Tip
    tic
    [allignedFace, landmarks, noseTip] = findNoseTip(normalScan, landmarks);
    tipTime = toc;

    tipCloud = pointCloud(noseTip, "Color",[1, 0, 0]);

    % Extract Facial Region
    tic
    faceCloud = extractFacialRegion(allignedFace, noseTip);
    extractTime = toc;

    % Reduce Noise
    tic
    silentFaceCloud = noiseFilter(faceCloud,"NumNeighbours", 20, "Mag", 2.25);
    filterTime = toc;

%   Fine Allignment
    tic
    [silentNormFaceCloud, tform2, e2] = normalizePose(silentFaceCloud, refSilentCloud);
    fineTime = toc;

    tipCloud = pctransform(tipCloud, tform2);
    landmarks = pctransform(landmarks, tform2);

    [processedFace, landmarks] = centreFace(silentNormFaceCloud, landmarks, tipCloud);

    display = false;
    if display
        disp("Coarse Allignment Complete completed in " + coarseTime)
        disp("Nose tip extracted in " + tipTime)
        disp("Facial region extracted in " + extractTime)
        disp("Noise Filtered in " + filterTime)
        disp("Fine Allignment Complete completed in " + fineTime)
        disp("Error reduced from " + e1 + " to " + e2)
    end
end
```

```
function [ptCloudOut,landmarksOut] = centreFace(ptCloudIn,landmarksIn, noseTip)

    translation = rigidtform3d([0, 0, 0], -noseTip.Location);

    ptCloudOut = pctransform(ptCloudIn, translation);
    landmarksOut = pctransform(landmarksIn, translation);

end
```

```
function ptCloudOut = upsampleFace(ptCloudIn, pointCount, options)

    arguments
        ptCloudIn pointCloud
        pointCount double = nan;
        options.Demo = false
    end

    dt = delaunayTriangulation(ptCloudIn.Location(:, 1:2));

    centres = incenter(dt);

    if options.Demo
        figure;
        triplot(dt)
        hold on
        plot(centres(:,1),centres(:,2),'*r')
        hold off
    end

    heights = zeros(length(centres), 1);
    for triangle = 1:length(dt.ConnectivityList)
        neighborIDs = dt.ConnectivityList(triangle, :);

        avgHeight = mean(ptCloudIn.Location(neighborIDs, 3));
        heights(triangle) = avgHeight;
    end

    newPoints = [centres, heights];
    newPoints = [ptCloudIn.Location; newPoints];

    if ~isnan(pointCount) && (length(newPoints) > pointCount)
        IDs = round(linspace(1, length(newPoints), pointCount));
    else
        IDs = 1:length(newPoints);
    end

    ptCloudOut = pointCloud(newPoints(IDs, :));

end
```

```
function [ptCloudOut, landmarksOut, noseTip, centralLine] = findNoseTip(ptCloudIn,
landmarksIn, centralRef, options)

    arguments
        ptCloudIn         pointCloud
        landmarksIn       pointCloud
        centralRef      = nan;
        options.SampleRate {mustBeInRange(options.SampleRate, ...
                            0, 1, "exclude-lower")} = 1
        options.Demo logical = false;

    end

    if options.SampleRate  ~= 1
        ptCloudIn = pcdownsample(ptCloudIn, "random", sampleRate * 100);
    end

    mirrorPoints = ptCloudIn.Location;
    mirrorPoints(:, 1) = -mirrorPoints(:, 1);

    mirrorFace = pointCloud(mirrorPoints);

    [~, transform] = normalizePose(ptCloudIn, mirrorFace);
  halfTransform = rigidtform3d(90 * rotm2eul(transform.R) / pi, ...
                               transform.Translation / 2);

    normFace = pctransform(ptCloudIn, halfTransform);
    landmarksIn = pctransform(landmarksIn, halfTransform);

    if options.Demo
        y = (-150:5:50)';
        midplane = pointCloud([zeros(length(y)^2, 1), repmat(y, length(y), 1),
reshape(repmat(y', length(y), 1), [], 1)]);
        invTform = invert(halfTransform);
        symmetryPlane = pctransform(midplane, invTform);

    end

    tform1 = rigidtform3d([0, 0, 0], ...
                                [-sum(normFace.XLimits) / 2, ...
                                 -sum(normFace.YLimits) / 2, ...
                                 -normFace.ZLimits(2)]);

    normFaceCentre = pctransform(normFace, tform1);
    landmarksIn = pctransform(landmarksIn, tform1);

    points = normFaceCentre.Location;
    midpoint = sum(normFaceCentre.XLimits) / 2;
    faceWidth = sum(abs(normFaceCentre.XLimits));
    bandWidth = 0.25;

    centralPoints = points((midpoint + (bandWidth/2) * faceWidth > points(:, 1)) &
...
                           (points(:, 1) > midpoint - (bandWidth/2) * faceWidth), :);

    centralLine = pointCloud(centralPoints, Color=[1, 0, 0]);


    load("CentreReference.mat", "centralLineRef")
    [tform2, centralLine] = pcregistericp(centralLine, centralLineRef, "Tolerance",
[0.005, 0.2]);

    localCloud = pointCloud(centralLine.Location(abs(centralLine.Location(:, 2)) <
30, :));
```

```
    % get the location of the points with the 10 highest z coordinates
    highestZs = maxk(localCloud.Location(:, 3), 50);
    possibleNoseTips = localCloud.Location(localCloud.Location(:, 3) >= min(highestZs),
:);

    possibleNoseTips = sortrows(possibleNoseTips, 3, "descend");

    for tipIdx = 1:length(possibleNoseTips)
        tip = possibleNoseTips(tipIdx, :);
        exhSearcher = ExhaustiveSearcher(localCloud.Location);
        Idx = rangesearch(exhSearcher,tip, 2);
        neighbors = length(Idx{1}) - 1;

%         if neighbors > 5
%             break
%         end
        break
    end

    noseTip = tip;

    ptCloudOut = pctransform(normFaceCentre, tform2);
    landmarksOut = pctransform(landmarksIn, tform2);

    if options.Demo
        figure;
        pcshowpair(centralLine, centralLineRef)

        figure;
        pcshow(ptCloudIn.Location, [1, 1, 1])
        hold on
        pcshow(symmetryPlane.Location, [1, 0, 0])
        hold off

        figure;
        pcshow(pointCloud(localCloud.Location, Color=[1, 1, 1]))
        hold on
        pcshow(pointCloud(possibleNoseTips, Color=[0, 1, 0]))
        pcshow(pointCloud(noseTip, Color=[1, 0, 0]), "MarkerSize", 100)
        hold off
        legend(['', "Candidate Nose Tips", "Nose Tip"], "Color", [1, 1, 1])

        figure;
        pcshow(ptCloudOut)
        hold on
        pcshow(tip, [1, 0, 0], "MarkerSize", 200)
        hold off

    end
end
```

```
function voxelImage = voxelizeCloud(data, options)

    arguments
        data
        options.GridSize = [50, 50, 32]
        options.Solid = false
    end

    if isa(data, "cell")
        ptCloudIn = data{1, 1};
    else
        ptCloudIn = data;
    end

    bins = pcbin(ptCloudIn, options.GridSize);
    faceMask = cellfun(@(c) ~isempty(c),bins);

    if options.Solid
        faceVolume = faceMask;
        for x = 1:size(faceMask, 1)
            for y = 1:size(faceMask, 2)
                for z = 1:size(faceMask, 3)
                    if faceMask(x, y, z) ~= false
                        faceVolume(x, y, 1:z) = true;
                    end
                end
            end
        end

        if isa(data, "cell")
            voxelImage = {faceVolume, data{1, 2}};
        else
            voxelImage = faceVolume;
        end
    else
        if isa(data, "cell")
          voxelImage = {faceMask, data{1, 2}};
        else
            voxelImage = faceMask;
        end
    end
end
```

```
function [features, warningFlag] = deltaFeatureV4(landmarks1, landmarks2, options)

    arguments
        landmarks1 pointCloud
        landmarks2 pointCloud

        options.Subset logical = true;

        options.Subfeatures = [1, 1, 1, 0, 0];


        options.Demo logical = false
    end

    warningFlag = false;

    landmarks1 = landmarks1.Location;
    landmarks2 = landmarks2.Location;

    if ~options.Subset
        leftEyeIDs = 1:8;
        rightEyeIDs = 9:16;

        leftEyebrowIDs = 17:26;
        rightEyebrowIDs = 27:36;

        mouthIDs = 49:68;
    else
        leftEyeIDs = [3, 5, 7];
        rightEyeIDs = [11, 13, 15];

        leftEyebrowIDs = [17, 19, 21, 22, 24, 26];
        rightEyebrowIDs = [27, 29, 31, 32, 34, 36];

        mouthIDs = [49, 51, 52, 53, 55, 56, 58, 60, 61, 63, 65, 67];
    end

    leftEye1 = landmarks1(leftEyeIDs, :); leftEye2 = landmarks2(leftEyeIDs, :);

    rightEye1 = landmarks1(rightEyeIDs, :); rightEye2 = landmarks2(rightEyeIDs, :);



    leftEyebrow1 = landmarks1(leftEyebrowIDs, :);
    leftEyebrow2 = landmarks2(leftEyebrowIDs, :);

    rightEyebrow1 = landmarks1(rightEyebrowIDs, :);
    rightEyebrow2 = landmarks2(rightEyebrowIDs, :);



    mouth1 = landmarks1(mouthIDs, :);
    mouth2 = landmarks2(mouthIDs, :);

    % Left Eye
    [LEF, LED, f1] = createLocalFeatures(leftEye1, leftEye2, options.Subfeatures);

    % Right Eye
    [REF, RED, f2] = createLocalFeatures(rightEye1, rightEye2, options.Subfeatures);

    % Mouth
    [MF, MD, f3] = createLocalFeatures(mouth1, mouth2, options.Subfeatures);
```

```
    [LEBF, LEBD, f4] = createLocalFeatures(leftEyebrow1, leftEyebrow2,
options.Subfeatures);

    % Right Eye
    [REBF, REBD, f5] = createLocalFeatures(rightEyebrow1, rightEyebrow2,
options.Subfeatures);

    features = [LEF; REF; LEBF; REBF; MF];

    if any([f1, f2, f3, f4, f5])
        warningFlag = true;
    end

    if options.Demo

        centres = [mean(leftEye1, 1); mean(rightEye2, 1); ...
                   mean(leftEyebrow1, 1); mean(rightEyebrow1, 1);
                   mean(mouth1, 1)];

        pcshow(landmarks1, [1, 1, 1])
        hold on
        pcshow(centres, [0, 0, 1], "MarkerSize", 100)
        pcshow(leftEye2(LED == 1, :), [0, 1, 0], "MarkerSize", 100)
        pcshow(leftEye2(LED == -1, :), [1, 0, 0], "MarkerSize", 100)

        pcshow(rightEye2(RED == 1, :), [0, 1, 0], "MarkerSize", 100)
        pcshow(rightEye2(RED == -1, :), [1, 0, 0], "MarkerSize", 100)

        pcshow(mouth2(MD == 1, :), [0, 1, 0], "MarkerSize", 100)
        pcshow(mouth2(MD == -1, :), [1, 0, 0], "MarkerSize", 100)

        pcshow(leftEyebrow2(LEBD == 1, :), [0, 1, 0], "MarkerSize", 100)
        pcshow(leftEyebrow2(LEBD == -1, :), [1, 0, 0], "MarkerSize", 100)

        pcshow(rightEyebrow2(REBD == 1, :), [0, 1, 0], "MarkerSize", 100)
        pcshow(rightEyebrow2(REBD == -1, :), [1, 0, 0], "MarkerSize", 100)

        hold off
    end

end

function [f, direction, flag] = createLocalFeatures(landmarks1, landmarks2,
subfeatures)
    arguments
        landmarks1; landmarks2
        subfeatures

    end

    flag = false;

    centre = mean(landmarks1, 1);

    d1 = landmarks1 - centre;
    d2 = landmarks2 - centre;
    moveR = landmarks2 - landmarks1; % relative movement in x, y, and z  ([x, 3])

    if any(abs(moveR) > 15)
        flag = true;
    end

    magR = sqrt(sum(moveR.^2, 2)); % relative distance between old and new pos;
```

```
    d1Mag = sqrt(sum(d1.^2, 2)); % distance between point 1 and centre
    d2Mag = sqrt(sum(d2.^2, 2)); % distance between point 2 and centre
    magC = d2Mag - d1Mag; % change in distance from centre

    direction = sign(magC);

    angleRXY = atan2(moveR(:, 2), moveR(:, 1));
    angleRXZ = atan2(moveR(:, 3), moveR(:, 1));
    angleRYZ = atan2(moveR(:, 3), moveR(:, 2));

    anglesR = [angleRXY, angleRXZ, angleRYZ];

    angleCXY = atan2(d2(:, 2), d2(:, 1)) - atan2(d1(:, 2), d1(:, 1));
    angleCXZ = atan2(d2(:, 3), d2(:, 1)) - atan2(d1(:, 3), d1(:, 1));
    angleCYZ = atan2(d2(:, 3), d2(:, 2)) - atan2(d1(:, 3), d1(:, 2));

    anglesC = [angleCXY, angleCXZ, angleCYZ];

    features = [];
    if subfeatures(1)
        features = [features, moveR];
    end

    if subfeatures(2)
        features = [features, magR];
    end

    if subfeatures(3)
        features = [features, magC];
    end

    if subfeatures(4)
        features = [features, anglesR];
    end

    if subfeatures(5)
        features = [features, anglesC];
    end

    f = reshape(features', [], 1);
end
```

```
function activations = DetectAUs(lm1, lm2, options)

    arguments
        lm1
        lm2
        options.Thresholds = [1, 1, 1];
        options.Demo = false;
    end

%     xScale = 100 / abs(lm1.XLimits(1) - lm1.XLimits(2));
%     yScale = 150 / abs(lm1.YLimits(1) - lm1.YLimits(2));
%     scaledPoints = [moving.Location(:, 1) * xScale, ...
%                     moving.Location(:, 2) * yScale, ...
%                     moving.Location(:, 3) * zScale];
%
    threshS = options.Thresholds(1); % small treshold
    threshM = options.Thresholds(2); % medium threshold
    threshL = options.Thresholds(3); % large threshold

    % AU 1
    AU1 = checkActivationYDir([17, 27], "increase", threshM); % inner brow raiser
    AU2 = checkActivationYDir([21, 31], "increase", threshS); % outer brow raiser

    AU4 = checkActivationYDir([17, 27, 26, 36], "decrease", threshM); % brow lowerer

    AU5 = checkActivationYDir([3, 11], "increase", threshS); % upper lid raiser

    AU10 = checkActivationYDir([51, 52, 53], "increase", threshL); % upper lip raiser
    AU12 = checkActivationYDir([49, 55], "increase", threshL) &&
checkActivationXDir(49, 55, "increase", threshL);

    AU15 = checkActivationYDir([49, 55], "decrease", threshM); % Lip corner down
    AU16 = checkActivationYDir([57, 58, 59], "decrease", threshM); % Lower lip down

    AU17 = checkActivationYDir([57, 58, 59], "increase", threshM); % Lower lip up

    AU18 = checkActivationXDir([51, 59], [53, 57], "decrease", threshS); % lip pucker
    AU20 = checkActivationXDir(49, 55, "increase", threshM); % lip stretcher

    AU24 = checkActivationYDir(52, "decrease", threshM); % lip pressor

    AU25 = checkActivationYDir(52, "increase", threshS) && checkActivationYDir(58,
"decrease", threshS); % lips part

    AU27 = checkActivationYDir(52, "increase", threshL) && checkActivationYDir(58,
"decrease", threshL); % mouth stretch

    AU41 = checkActivationYDir([3, 11], "decrease", threshM); % lid droop
    AU42 = AU41 && checkActivationYDir([7, 15], "increase", threshS); % squint

    names = ["AU1", "AU2", "AU4", "AU5", "AU10", "AU12", "AU15", "AU16", ...
             "AU17", "AU18", "AU20", "AU24", "AU25", "AU27", "AU41", "AU42"];

    active = [AU1, AU2, AU4, AU5, AU10, AU12, AU15, AU16, ...
              AU17, AU18, AU20, AU24, AU25, AU27, AU41, AU42];

    activations = dictionary(names, active);

    if options.Demo
        AUNames = ...
            ["Inner Brow Raiser", "Outer Brow Raiser", "Brow Lowerer", ...
             "Upper Lid Raiser", "Upper Lip Raiser", "Lip Corner Puller", ...
             "Lip Corner Depressor", "Lower Lip Depressor", "Lower Lip Raiser", ...
             "Lip Puckerer", "Lip Strecher", "Lip Pressor", "Lips Part", ...
             "Mouth Stretch", "Lid Droop", "Squint"];
```

```
        disp(AUNames(active))
    end

    function activated = checkActivationYDir(IDs, change, thresh)

        activated = false;

        loc1 = lm1.Location(IDs, :); % [x1, y1, z1]
        loc2 = lm2.Location(IDs, :); % [x2, y2, z2]

        delta = loc2 - loc1; % in the form of [dx, dy, dz]

        if change == "increase"
            if any(delta(:, 2) >= thresh) && all(delta(:, 2) > 0)
                % if dy > threshold, activate
                activated = true;
            end
        else
            if any(delta(:, 2) <= -thresh) && all(delta(:, 2) < 0)
                % if dy < -threshold, activate
                activated = true;
            end
        end
    end

    function activated = checkActivationXDir(leftIDs, rightIDs, change, thresh)

        activated = false;

        leftLoc1 = lm1.Location(leftIDs, :);
        leftLoc2 = lm2.Location(leftIDs, :);
        rightLoc1 = lm1.Location(rightIDs, :);
        rightLoc2 = lm2.Location(rightIDs, :);

        leftDelta = leftLoc2 - leftLoc1;
        rightDelta = rightLoc2 - rightLoc1;

        if change == "increase"
            leftActive = any(leftDelta(:, 1) <= -thresh) && all(leftDelta(:, 1) < 0);
            rightAcitive = any(rightDelta(:, 1) >= thresh) && all(rightDelta(:, 1) >
0);

            if leftActive && rightAcitive
                activated = true;
            end
        else
            leftActive = any(leftDelta(:, 1) >= thresh) && all(leftDelta(:, 1) > 0);
            rightAcitive = any(rightDelta(:, 1) <= -thresh) && all(rightDelta(:, 1) <
0);

            if leftActive && rightAcitive
                activated = true;
            end
        end
    end
end
```

```
function [trainedClassifier, validationAccuracy] = LinearAU(trainingData, responseData)

    predictorNames = cell(1, size(trainingData, 2));
    for i = 1:size(trainingData, 2)
        predictorNames{i} = sprintf('column_%d', i);
    end

    inputTable = array2table(trainingData, 'VariableNames', predictorNames);

    predictors = inputTable(:, predictorNames);
    response = responseData;

    % Train a classifier
    % This code specifies all the classifier options and trains the classifier.
    template = templateSVM(...
        'KernelFunction', 'linear', 'PolynomialOrder', [], ...
        'KernelScale', 'auto', 'BoxConstraint', 1, ...
        'Standardize', true);
    classificationSVM = fitcecoc(...
        predictors, response, 'Learners', template, ...
        'Coding', 'onevsone', 'ClassNames', categorical(...
        {'Happy'; 'Sad'; 'Angry'; 'Fear'; 'Surprise'; 'Disgust'},...
        {'Happy' 'Sad' 'Angry' 'Fear' 'Surprise' 'Disgust'}));

    % Create the result struct with predict function
    predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
    svmPredictFcn = @(x) predict(classificationSVM, x);
    trainedClassifier.predictFcn = @(x) svmPredictFcn(predictorExtractionFcn(x));

    % Add additional fields to the result struct
    trainedClassifier.ClassificationSVM = classificationSVM;

    % Perform cross-validation
    partitionedModel = crossval(trainedClassifier.ClassificationSVM, 'KFold', 10);

    % Compute validation predictions
    % [validationPredictions, validationScores] = kfoldPredict(partitionedModel);

    % Compute validation accuracy
    validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');

end
```

```
function [trainedClassifier, validationAccuracy] = AdaptiveSubspaceFuse(trainingData,
responseData, options)

    arguments
        trainingData
        responseData
        options.Workers = 30;
        options.Dimension = 30;
    end

    predictorNames = cell(1, size(trainingData, 2));
    for i = 1:size(trainingData, 2)
        predictorNames{i} = sprintf('column_%d', i);
    end

    inputTable = array2table(trainingData, 'VariableNames', predictorNames);
    predictors = inputTable(:, predictorNames);
    response = responseData;

    % Train a classifier
    % This code specifies all the classifier options and trains the classifier.
    subspaceDimension = max(1, min(options.Dimension, width(predictors) - 1));
    classificationEnsemble = fitcensemble(...
        predictors, response, 'Method', 'Subspace', ...
        'NumLearningCycles', options.Workers, ...
        'Learners', 'discriminant', 'NPredToSample', subspaceDimension, ...
        'ClassNames', categorical(...
        {'Happy'; 'Sad'; 'Angry'; 'Fear'; 'Surprise'; 'Disgust'}, ...
        {'Happy' 'Sad' 'Angry' 'Fear' 'Surprise' 'Disgust'}));

    % Create the result struct with predict function
    predictorExtractionFcn = @(x) array2table(x, 'VariableNames', predictorNames);
    ensemblePredictFcn = @(x) predict(classificationEnsemble, x);
    trainedClassifier.predictFcn = @(x) ensemblePredictFcn(predictorExtractionFcn(x));

    % Add additional fields to the result struct
    trainedClassifier.ClassificationEnsemble = classificationEnsemble;

    % Perform cross-validation
    partitionedModel = crossval(trainedClassifier.ClassificationEnsemble, 'KFold', 10);

    % Compute validation predictions
    % [validationPredictions, validationScores] = kfoldPredict(partitionedModel);

    % Compute validation accuracy
    validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
end
```

```
function [facePoints, landmarks, fileNames] = getData3D(sex, ID, emotion, intensities,
options)
    arguments
        sex       {mustBeMember(sex,["M", "F"])}
        ID        uint8
        emotion   {mustBeMember(emotion, ...
                  ["Angry","Disgust", "Fear", "Happy", "Sad", "Surprise", "Neutral"])}
        intensities = 4
        options.Source {mustBeMember(...
                    options.Source,["Raw", "Cropped"])} = "Cropped"
        options.FP  = true
        options.LM  = true
    end

    if emotion == "Neutral"
        intensities = 0;
    end
    if sex == "M" && ID > 44
        error("ID exceeds the number of participants")
    elseif sex == "F" && ID > 56
        error("ID exceeds the number of participants")
    end

    IDString = sprintf("000%i", ID);
    IDString = extractAfter(IDString, strlength(IDString) - 4);

    facePoints = pointCloud.empty(0, length(intensities));
    landmarks = pointCloud.empty(0, length(intensities));
    for intensityIdx = 1:length(intensities)
        intensity = intensities(intensityIdx);
        intString = sprintf("0%i", intensity);
        emString = upper(extractBefore(emotion, 3));

        folderPath = sprintf("/Volumes/UltraTouch/Dataset/BU_3DFE/%s%s", ...
                             sex, IDString);
        fileBase = sprintf("%s%s_%s%s", sex, IDString, emString, intString);
        fileNames = struct2table(dir(folderPath)).name;

        fileNames = fileNames(~contains(fileNames, "._"));
        fileNames = fileNames(contains(fileNames, fileBase));

        if options.Source == "Cropped"
            fileNames = fileNames(contains(fileNames, "F3D"));
        else
            fileNames = fileNames(contains(fileNames, "RAW"));
        end
        faceName = fileNames{contains(fileNames, ".wrl")};
        landmarkName = fileNames{contains(fileNames, ".bnd")};
        facePath = join([folderPath, faceName], "/");
        landmarkPath = join([folderPath, landmarkName], "/");
        if options.FP
            facePoints(intensityIdx) = extractCoordinates(facePath);
        end

        if options.LM
            points = readtable(landmarkPath, "FileType","text");
            try
                if width(points) == 4
                    xyz = table2array([points(:, 2), points(:, 3), points(:, 4)]);
                else
                    xyz = table2array([points(:, 3), points(:, 5), points(:, 7)]);
                end
```

```
            catch ME
                points
                ME.identifier
                rethrow(ME)
            end
            landmarks(intensityIdx) = pointCloud(xyz);
        end
    end

    function cloud = extractCoordinates(path)
        w = vrworld(path);
        open(w)
        faceNodes = nodes(w, "-full");

        for idx = 1:length(faceNodes)
            values = get(faceNodes(idx));
            if values.Name == "Tricorder_object"
                field = getfield(faceNodes(idx));
                coordinates = field.children;
                coordinates =
getfield(getfield(getfield(coordinates).geometry).coord).point;
            end
        end

        uniqueCoords = unique(coordinates, 'rows', 'stable');

        uniqueCoords = uniqueCoords(all(uniqueCoords(:, 1) ~= [0, 0, 0], 2), :);

        cloud = pointCloud(uniqueCoords);

        close(w)
    end
end
```

```
function [facePoints, landmarks] = getData4D(sex, ID, emotion, frames)
    arguments
        sex       {mustBeMember(sex,["M", "F"])}
        ID        uint8
        emotion   {mustBeMember(emotion, ...
                  ["Angry","Disgust", "Fear", "Happy", "Sad", "Surprise"])}
        frames
    end

    if sex == "M" && ID > 43
        error("ID exceeds the number of participants")
    elseif sex == "F" && ID > 58
        error("ID exceeds the number of participants")
    end

    IDString = sprintf("00%i", ID);
    IDString = extractAfter(IDString, strlength(IDString) - 3);

    facePoints = pointCloud.empty(0, length(frames));
    landmarks = pointCloud.empty(0, length(frames));

    for frameIdx = 1:length(frames)
        frame = frames(frameIdx);
        frameString = sprintf("00%i", frame);
        frameString = extractAfter(frameString, strlength(frameString) - 3);

        fFolderSubstrings = ["F001~F020", "F021~F040", "F041~F058"];
        mFolderSubstrings = ["M001~M020", "M021~M043"];

        if sex == "M" && ID <= 20
            folderSubstring = mFolderSubstrings(1);
        elseif sex == "M"
            folderSubstring = mFolderSubstrings(2);
        elseif sex == "F" && ID <= 20
            folderSubstring = fFolderSubstrings(1);
        elseif sex == "F" && ID <= 40
            folderSubstring = fFolderSubstrings(2);
        elseif sex == "F"
            folderSubstring = fFolderSubstrings(3);
        end

        facePath = sprintf("/Volumes/UltraTouch/Dataset/BU4DFE/%s/%s%s/%s/%s.wrl", ...
                           folderSubstring, sex, IDString, emotion, frameString);

        try
            facePoints(frameIdx) = extractCoordinates(facePath);

            landmarkPath =
sprintf("/Volumes/UltraTouch/Dataset/BU4DFE/BU4DFE_BND_V1.1/%s%s/%s/%s.bnd", ...
                           sex, IDString, emotion, frameString);
            points = importdata(landmarkPath);
            xyz = [points(:, 2), points(:, 3), points(:, 4)];

            landmarks(frameIdx) = pointCloud(xyz);
        catch ME
            ME.identifier
            rethrow(ME)
            % 'sl3d:interface:vrmlreaderror'
            break
        end
    end
```

```
    function cloud = extractCoordinates(path)
        w = vrworld(path);
        open(w);
        faceNodes = nodes(w, "-full");

        for idx = 1:length(faceNodes)
            values = get(faceNodes(idx));
            if values.Name == "di3d-COORD"
                field = getfield(faceNodes(idx));
                coordinates = field.point;
            end
        end

        uniqueCoords = unique(coordinates, 'rows', 'stable');

        uniqueCoords = uniqueCoords(all(uniqueCoords(:, 1) ~= [0, 0, 0], 2), :);

        cloud = pointCloud(uniqueCoords);

        close(w)
    end

end
```