Facial Feature Detection and Reenactment Onto Computer Generated Imagery

Abhishek Pande and Arpit Gupta

Real-time emotion extraction and transfer are essential and open problems in Computer Vision and Computer Graphics. With studies ranging from analysis and creation of facial reenactment, deep-fakes to more commercial domains like creating real-life Computer Generated Imagery (CGI) in movies, with exact human-like facial emotions, the above-stated problems find their applications in various diverse fields.
This paper presents a novel method of facial feature detection and the reenactment of a monocular video target sequence in real-time. The source is a video sequence captured from a commercial commodity webcam. Our goal is to transfer the extracted facial features and animate them upon a Computer Generated Imagery (CGI). To this end, we address facial feature extraction from a monocular video using dlib and solve the head-pose estimation problem. Secondly, we propose a method to capture and transfer fine wrinkle level details displayed in various facial expressions.

1. *Introduction*

Real-time facial feature extraction and transfer have a long history in computer vision. Some fruitful approaches have relied on methods and techniques such as area localization, gaze detection, Convolution Neural Networks based emotion detection, etc. In this project, we present a way of capturing facial features and transferring them onto a CGI.
We do the proposed by first tracking the facial features using the facial landmark data and dlib python library in our method. The algorithm is based upon a model trained with multiple images in which the facial landmark features are marked and then passed through the ensemble of regression trees. After training, these regression trees are capable of estimating the facial landmark. The facial landmarks are obtained in the form of points on the face, and each point gives us the location of a major facial feature/landmark.
Our work focuses on improving the current methods of capturing and transferring facial features and emotions and reenacting them onto computer-generated imagery. Computer Generated Imagery, better known as CGI, is applying computer-based graphics to create images in art, video games, media, films, and television. At the most basic CGI level is creating still or animated visual contents using computer software.

The proposed methodology finds its use case in:

• **Animating Movie CGI, Cartoons, Video Books**: Allowing real-time facial feature tracking and transfer on-to the CGI character with fine wrinkle level details.

2. *Methodology*

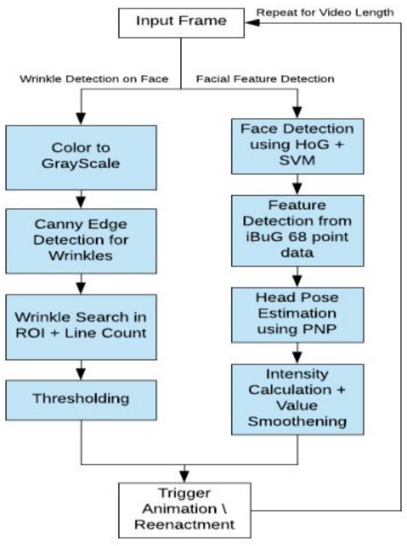
Input Frame
Repeat for Video Length
Wrinkle Detection on Face
Facial Feature Detection
Color to GrayScale
Canny Edge Detection for Wrinkles
Wrinkle Search in ROI + Line Count
Thresholding
Face Detection using HoG + SVM
Feature Detection from iBuG 68 point data
Head Pose Estimation using PNP
Intensity Calculation + Value Smoothening
Trigger Animation \ Reenactment

Fig. 1: Flow Chart for Feature Reenactment

Fig. 2: Facial CGI in Movies

A. *Facial Feature Detection*

We use dlib to detect facial features in the incoming target video stream. For this, we use the 68 point shape predictor data set. This maps the image of a human face on-to a 128-dimensional vector space. The face recognition is then performed by mapping faces to the 128 dimensional space and then checking their euclidean distance. The accuracy in facial feature detection depends upon the distance threshold. Our method has used the default threshold of 0.6, which provides an accuracy of 99.38% on the LFW facial database benchmark.

```
p = "shape_predictor_68_face_landmarks.dat"
predictor = dlib.shape_predictor(p)
detector = dlib.get_frontal_face_detector()
```

The 68 points obtained are part of the iBUG 300-W data set used to train the dlib facial landmark predictor. Other data sets are also available in which the number of identified points varies—for example, 194 point model trained on HELEN data set. Changing the number of points affects the detailing in capturing v/s the predictor's time. The predictor used identifies points on the face such as the corners of the mouth, along with the eyebrows, on eyes, and so forth. The frontal face detector is made using the classical Histogram of Oriented Gradients (HOG) combined with a linear classifier (Support Vector Machines), image pyramid, and sliding window scheme. The predictor provides us with the method to obtain a bounding box of the face (i.e., the x, y-coordinates of the face) in the video stream frames. Given the face region, the algorithm detects the key facial structures in the face region.
To estimate the landmark locations, the shape predictor uses the following algorithm:

• Examines the sparse set of input pixel intensities (features to the input model).

• Passes the features into the Ensemble of Regression Trees (ERT).

• The predictor locations are refined to improve the accuracy through a cascade of regressors.

The result is a shape predictor that functions in real-time.
We run the algorithm on the incoming video frames in real-time and obtain the location of the 68 landmark points in the form of 2D coordinates.
The face detector and predictor algorithm provided by the dlib library that we have used for our implementation is created using techniques such as HoG filters, linear classifiers, image pyramid, and sliding window detection schemes. The aim is to extract features into a vector using a Histogram of Oriented Gradients and feeding the vector into a Support Vector Machine (Linear Classifier), which will assess whether the region contains a face or not.

1) *Histogram of Oriented Gradients*: Histogram of oriented gradients (HOG) is a feature descriptor popularly used in computer vision for object detection. As a feature, descriptor HOG helps represent an image in simplified form by extracting useful information and discarding extraneous information. HOG takes an image of the form $n \times m \times d$, where n, m are the height and width of the image, respectively, and d is the number of channels present in the image and returns a processed output vector. The output vector in itself is not useful for viewing purposes. However, it is very useful for image recognition and object detection tasks, which is the main function of recognition and detection of facial features.
In HOG, the histograms (distribution) of oriented gradients are used as

features. Gradients are used as their magnitude is large around edges and corners and pack more information than flat surfaces.
The magnitude and gradient are calculated using (1)

$$g = \sqrt{g_x^2 + g_y^2} \quad = arctan \frac{g_x}{g_y} \tag{1}$$

The processed gradient image highlights the outline in the image. After gradients are calculated, the image is divided into 8 × 8 cells, and histograms for the oriented gradients are calculated. The gradient of each such patch contains 8 × 8 × 2 = 128 numbers, which are represented using a 9 bin histogram. The final step is to normalize the histogram and calculate the feature vector.

2) *Support Vector Machine*: Linear Classifier: Support Vector Machine is a supervised machine learning algorithm mostly used for classification problems. In the SVM algorithm, each data item is plotted as a point in n-dimensional space (where n is the number of features you have). The value of each feature is the value of a particular coordinate. The classification is performed by differentiating the hyper-plane into various classes.
(2) represents our model/classifier as a function.

$$h_{w,b}(x) = g(w^T x + b) \tag{2}$$

3) *Image Pyramid*: Image Pyramids are used when we need to work on the image in different resolutions, generally for searching objects in the image. In these cases, we create images and search for the object in all these images. These sets of images with different resolutions are called Image Pyramids.
There are two types of pyramids possible :

- Up-size the image, i.e., Zoom in
- Downsize the image, i.e., Zoom out

Every layer of the pyramid (image in the set of images) is numbered from bottom to top and is denoted as G_i; layer G_{i+1} is always smaller than layer G_i in an upward pyramid. To create a new layer, we convolve G_i with a Gaussian kernel and remove every even-numbered row and column.
The resulting image is exactly one-quarter the area of its predecessor image.

B. *Head Pose Estimation*

The Head pose is an orientation of the face with respect to the camera. It is the face orientation with respect to the screen or the camera from which the source video is obtained. The orientation/pose can be changed by either moving the camera or moving the head.
The head's movement is an important feature; our head's position affects the important facial features/landmarks and how to capture them. Moreover, as the aim of our project is to improve the CGI facial capture methods, a focus has to be given to capturing the movement of the head and estimate the current pose and transfer it to from the video frame (source: real human face) to the target (CGI face).
An alternate name for the problem head poses estimation is the Perspective-n-Point problem or PNP problem. The problem's target is to detect the orientation of the face with respect to the camera when we have the n-3D points of the face and their corresponding 2D projection.
To solve the head pose estimation problem, we solve the Perspective n Point problem. We have used the PNP function provided by OpenCV, which implements various algorithmic approaches. The type of function to solve the problem is decided by passing a flag. We have used the default flag, which is essentially the Direct Linear Transform, for our method. The flag is called SOLVEPNP_ITERATIVE. This particular algorithm also includes Levenberg-Marquardt optimization.
We first classify the landmark points passed onto the function to estimate the head pose. An implementation of the algorithm is explained in the following pseudo-code:

Algorithm 1 Head Pose Estimation

```
1 : $detector$ ← frontal face detector using dlib
2 : $predictor$ ← 68 landmark file to the shape predictor in dlib
3 : $cap$ ← VideoCapture in OpenCV
4 : $frame\_width$ ← frame width from cap
5 : $frame\_height$ ← frame height from cap
6 : $fps$ ← fps from cap
7 : $model\_points$ ← model point tuple
8 : $image\_points$ ← image point tuple
9 : for n = 1 to VideoLength do
10 :     $image$ ← capture video frame
11 :     $frame$ ← resize captured frame
12 :     $gray$ ← store the image as grayscale converted image
13 :     $rects$ ← detector(gray, 0)
14 :     for rect in rects do
15 :         $(bx, by, bw, bm)$ ← found the bounding box dimension using face_utils
16 :         plot it on the original image
17 :         $shape$ ← predictor(gray, rect)
18 :         Convert the shape obtained to numpy array
19 :         for $(i, (x, y))$ in $enumerate(Shape)$ do
20 :             Check if i is equal to a key landmark point
21 :             map these values on the image
22 :         end for
23 :         $focal\_length$ ← focal length of the webcam
24 :         $center$ ← optical center of the webcam
25 :         $camera\_matrix$ ← generate camera matrix using numpy
26 :         $(rotation\_vector, translation\_vector)$ ← solvePnP(model_points, image_points, camera_matrix, dist_coeffs, flag)
27 :         Find the head pose estimation using slope and Jacobian rotation and translation vector obtained
28 :     end for
29 : end for
```

PNP stands for Perspective-n-Point problem where the target is to detect the face's orientation with respect to the camera.
1) Mathematical Representation of Camera Model:

- **Translation**: The movement of points in 3D from a location (X, Y, Z) to a new location (X‘, Y ‘, Z‘). These movements have three degrees of freedom.
- **Rotation**: Rotation also provides three degrees of freedom and is generally represented in Euler angles. So, translation and rotation are needed to be computed to estimate the 3D pose.

So, translation and rotation are needed to be computed to estimate the 3D pose.

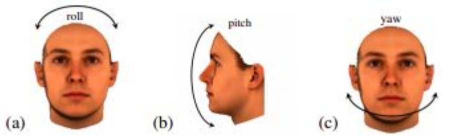

Fig. 3: The three degrees of freedom of the head pose. (a) roll angle denoted as θ_x, (b) pitch angle denoted as θ_y, and (c) yaw angle denoted as θ_z.

2) *Working of PNP Algorithm*: PNP algorithm is one of the oldest algorithms for head detection. In our task, we are constrained to the 3D coordinate system. Our task is to convert the 3D world coordinates into camera coordinates. To achieve this, we use the camera's intrinsic properties to project camera coordinates.

Let us assume (U, V, W) be the location of a 3D point in the World Coordinates; now, if we have prior knowledge of the rotation R, we can use it to map (X, Y, Z) in 3D coordinates to camera coordinates with the help of the translation matrix **t**.

Here, **R** is a 3 × 3 matrix, and **t** is a 3 × 1 matrix.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R \begin{bmatrix} U \\ V \\ W \end{bmatrix} + t$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} R | t \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \tag{3}$$

We can rewrite the above equation as:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{0_0} & r_{0_1} & r_{0_2} & t_x \\ r_{1_0} & r_{1_1} & r_{1_2} & t_y \\ r_{2_0} & r_{2_1} & r_{2_2} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \quad (4)$$

This provides us with multiple linear equations, which can be solved if a sufficient number of correspondence points are known.

3. Result and Discussion

Changing the number of points from 68 to other values affects the detailing in capturing v/s the time taken by the predictor.
Intrinsic parameters of the camera such as optical center are calculated by the center of the image and the focal length by the width of the image in pixels. Head Pose estimation requires the location of 6 landmarks namely the tip of the nose, chin, left corner of the left eye, right corner of the right eye, left corner of the mouth, right corner of the mouth.
The end result is a shape predictor that functions in real time. We run the algorithm on the incoming video frames in real time and obtain the location of the 68 landmark points in the form of 2D coordinates as well as the head pose estimation.
Using a RGB image and applying the discussed algorithms.

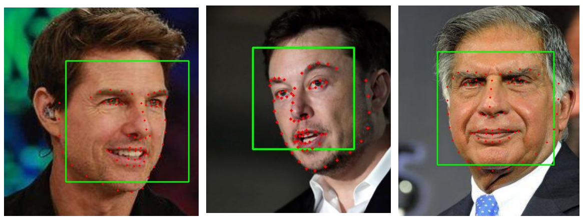

Fig. 4: Detection of 68 facial landmarks

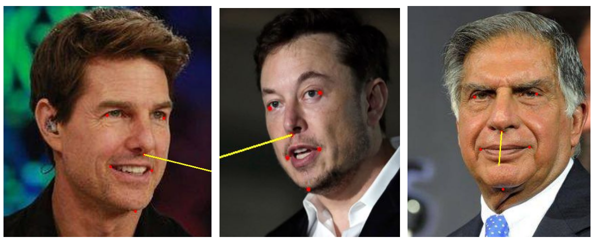

Fig. 5: Head Pose Estimation

4. *Conclusion and Future Work*

The presently developed approaches perform decently in terms of the computational costs as it uses efficient and robust algorithms. The dlib shape predictor does not provide us with wrinkle level details, nor does it help estimate the head pose, which will be included in the model upon completion. The PNP approach can be further optimized by not computing the head pose for every frame. This will lead us to a trade-off between execution speed and accuracy. Reenactment of the complete facial details will be done on the CGI using Blender functionality.

Future Plans: To obtain even finer details of facial expressions like forehead wrinkles, mid-head wrinkles, etc., we propose a canny edge detection based algorithm. We also propose to calculate the number of wrinkle lines obtained and differentiating the obtained lines from noise obtained on running edge detectors over video frames. We use the preexisting blender functionality to transfer and animate these facial features on-to the Computer Generated Imagery (CGI). This functionality allows us to animate over a CGI. We use CGI models that already have major emotion/feature animations present. Based on the real-time data obtained from the webcam video, we trigger these animations. We also consider the moment of landmark feature points between consecutive frames of the video to determine the intensity with which the animation is triggered. We can also include eye gaze detection and reenactment. The algorithm works correctly for fixed movements of the head. The accuracy reduces when head pose changes abruptly. Therefore to detect head pose changes beyond speed and angles, we can use some pretrained data to detect the head pose.

Debashis Ghosh (Department of Electronics and Communication Engineering, IIT Roorkee, Roorkee 247667, India)
✉ E-mail: debashis.ghosh@ece.iitr.ac.in

References

1 Kazemi, V. and J. Sullivan. “One millisecond face alignment with an ensemble of regression trees.” 2014 IEEE Conference on Computer Vision and Pattern Recognition (2014): 1867-1874.

2 B. A. Efraty, M. Papadakis, A. Profitt, S. Shah and I. A. Kakadiaris, "Facial component-landmark detection," 2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG), Santa Barbara, CA, 2011, pp. 278-285, doi: 10.1109/FG.2011.5771411.

3 T. Vatahska, M. Bennewitz and S. Behnke, "Feature-based head pose estimation from images," 2007 7th IEEE-RAS International Conference on Humanoid Robots, Pittsburgh, PA, 2007, pp. 330-335, doi: 10.1109/ICHR.2007.4813889.

4 Q. Wang and L. Shi, "Pose estimation based on PnP algorithm for the racket of table tennis robot," 2013 25th Chinese Control and Decision Conference (CCDC), Guiyang, 2013, pp. 2642-2647, doi: 10.1109/CCDC.2013.6561387.

5 A. Singh, “Feature descriptor,” Analyticsvidhya.com, 04-Sep-2019. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/.

6 N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

7 A. Rosebrock, “Training a custom dlib shape predictor,” Pyimagesearch.com, 16-Dec-2019. [Online]. Available: https://www.pyimagesearch.com/2019/12/16/training-a-custom-dlib-shape-predictor/.