Master in Informatics and Multimedia Engineering

Computer Vision and Mixed Reality

1st Project – Feature Based
Face Detection and Recognition for Augmented Reality

1. **Goal**
    a. Develop a computer vision application for face detection and recognition that allows the inclusion of virtual elements aligned with real objects.
    b. Familiarization with the OpenCV (**Open**-Source **C**omputer **V**ision) library for real-time application programming.
2. **Development**
    a. Face detection
        i. Using Haar Cascade Classifier

        OpenCV "CascadeClassifier" method is based on the algorithm of object detection using Haar feature-based cascade classifiers proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.
        ii. Using OpenCV Deep Neural Network module (dnn)

        OpenCV has a deep neural network module and include pre-trained Convolution Neural Network (CNN) for face detection. This network is based on the ResNet architecture with a Single Shot Detector (SSD) for the detection phase.

    Use these algorithms to implement face detection in images collected in real time by a camera (see [1,2] for documentation and [3,4] as examples of implementation).
    b. Face Recognition
        i. Face database

        Perform the acquisition of a face database from the students in the class (7 face images for each student). This set will be used in this project as the training and test sets. It may also be used a face database available on the Internet.
        ii. Face normalization

        Proceed with the normalization of the face images from the training set (rotation, scaling, selection) so that the faces comply with the format defined by MPEG-7 recommendation. That is, each face is represented by a monochrome image (256 levels) with 56 rows and 46 columns, with both eyes, right and left, perfectly aligned horizontally, and located in line 24, columns 16 and 31, respectively.

        For eye detection, Haar Cascade classifier can also be used with proper XML classifiers files.
        iii. Using *EigenFaces*
            1. Compute the mean face vector μ (column vector with size of 2576x1) and the transformation matrix W_{pca} (with size of 2576×*m*), containing the first *m*

eigenfaces (where *m* can be a number between 10 and 20). Use the procedures presented in the slides of the course. View the *m*+1 obtained vectors by interpreting them as face images (*eigenface*) and prove that they form an orthogonal basis.

2. Preform the projection of some of the training set faces on the face subspace and observe their reconstructions. Compute (and view) the error face (difference between the original face and its reconstruction). Show that the error face is orthogonal to the face subspace.

iv. Using *FisherFaces*

Repeat the previous points, but now using the algorithm called *fisherfaces*. Use the procedures presented in the slides of the course.

v. Classification

Develop a Nearest-Neighbor (NN) classifier based on the feature vectors resulting from the *m* projection coefficients. Consider an arbitrary number of classes, c (involving all students in the class or just a part of them).

Test the implemented classifiers using the test set (different from the training set).

c. Combine real and virtual objects.

i. Virtual object normalization

Based on the alignment of section b.ii, normalize the virtual object that is intended to be superimposed on the identified face, for example, a hat, glasses, or other object of your choice. Use different objects for different faces/persons.

ii. Adding with mask

Add to the image, using a mask.

Bibliography

[1] - https://docs.opencv.org/master/d6/d0f/group__dnn.html#ga33d1b39b53a891e98a654fdeabba22eb

[2] - https://github.com/opencv/opencv/wiki/Deep-Learning-in-OpenCV

[3] - https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[4] - https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c

Main *NumPy* functions

Goal	numpy funtion	
	numpy.array	numpy.matrix
turn array into vector	*flatten()*	
construct a matrix from blocks (vectors or matrices)	*vstack(), hstack(), tile(), concatenate()*	*bmat(), concatenate()*
mean or average	*mean()*	
transpose of *a*	*a.transpose() or a.T*	
matrix multiply	*dot()*	*
compute eigenvalues and eigenvectors	*linalg.eig()*	
inverse of square matrix	*linalg.inv()*	
sort the matrix	*sort(), argsort()*	*mat(sort())*
change matrix dimensions	*reshape()*	

Main *OpenCV* function

Goal	OpenCV function(s)
Read an image	*imread()*
Show an image	*imshow(), waitKey()*
Read images from a camera or a video file	*VideoCapture(); VideoCapture.get(); VideoCapture.read();*
Add images	*add(), addWeighted()*
Multiply images	*multiply()*
Absolute difference	*absDiff()*
Apply a blur or a median filter	*blur(), medianBlur()*
Resize an image	*resize()*
Scales, calculates absolute values, and converts the result to 8-bit.	*convertScaleAbs()*
Color conversions	*cvtColor()*
Contour extraction	*Canny()*
Image thresholding	*threshold()*
Calculates an affine matrix of 2D rotation.	*getRotationMatrix2D()*
Applies an affine transformation to an image	*warpAffine()*