Facial Recognition

Object Oriented Programming in Java

Project Report for 2nd Semester

Raunak Singh | Akhil Sharma | Ishan Bansal | Antas | Vipul Wasnik

Cluster Innovation Centre, Univeristy of Delhi
BTech. (Information Technology and Mathematical Innovations)

Index

Certificate of Completion

This is to certify that the following students: Raunak Singh, Akhil Sharma, Ishan Bansal, Antas and Vipul Wasnik have completed this project under my guidance and supervision as per the contentment of the requirements of the second semester in the course BTech (Information Technology and Mathematical Innovations) at the Cluster Innovation Centre, University of Delhi.

Prof. Anjani Kumar Verma
Dept. of Computer Science
Cluster Innovation Centre
Delhi University

Acknowledgment

It is a great pleasure for us to present our project for the subject Object Oriented Programming in Java, a part of the second semester's curriculum in BTech (Information Technology and Mathematical Innovations) at the Cluster Innovation Centre, University of Delhi.

We would like to thank our teacher and mentor Prof. Anjani Kumar Verma who helped render this project an adequate success.

Introduction

Face detection -- also called facial detection -- is an artificial intelligence (AI) based computer technology used to find and identify human faces in digital images. Face detection technology can be applied to various fields -- including security, biometrics, law enforcement, entertainment and personal safety -- to provide surveillance and tracking of people in real time.

Face detection has progressed from rudimentary computer vision techniques to advances in machine learning (ML) to increasingly sophisticated artificial neural networks (ANN) and related technologies; the result has been continuous performance improvements. It now plays an important role as the first step in many key applications -- including face tracking, face analysis and facial recognition. Face detection has a significant effect on how sequential operations will perform in the application.

In face analysis, face detection helps identify which parts of an image or video should be focused on to determine age, gender and emotions using facial expressions. In a facial recognition system -- which maps an individual's facial features mathematically and stores the data as a faceprint -- face detection data is required for the algorithms that discern which parts of an image or video are needed to generate a faceprint. Once identified, the new faceprint can be compared with stored faceprints to determine if there is a match.

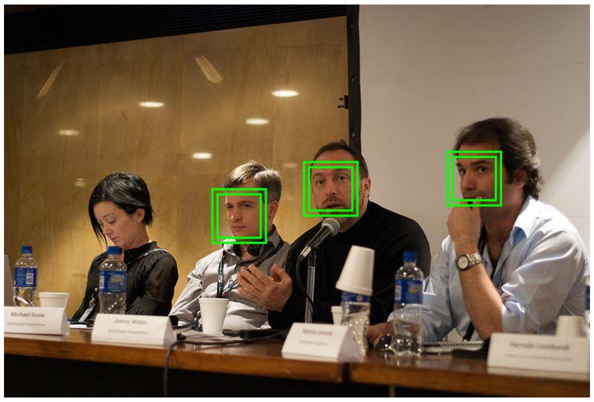
OpenCV

OpenCV is a software toolkit for processing real-time image and video, as well as providing analytics, and machine learning capabilities.

Using OpenCV, developers can access many advanced computer vision algorithms used for image and video processing in 2D and 3D as part of their programs. The algorithms are otherwise only found in high-end image and video processing software.

Video analytics is much simpler to implement with OpenCV API's for basic building blocks such as background removal, filters, pattern matching and classification. Real-time video analytics capabilities include classifying, recognizing, and tracking: objects, animals, people, specific features such as vehicle number plates, animal species, and facial features such as faces, eyes, lips, chin, etc.

OpenCV is written in Optimized C/C++, is cross-platform by design and works on a wide variety of hardware platforms. OpenCV has a wide range of applications in traditional computer vision applications such as optical character recognition or medical imaging. For example, OpenCV can detect Bone fractures. OpenCV can also help classify skin lesions and help in the early detection of skin melanomas2.



Automatic Face Detection using OpenCV.

Face Detection Methods

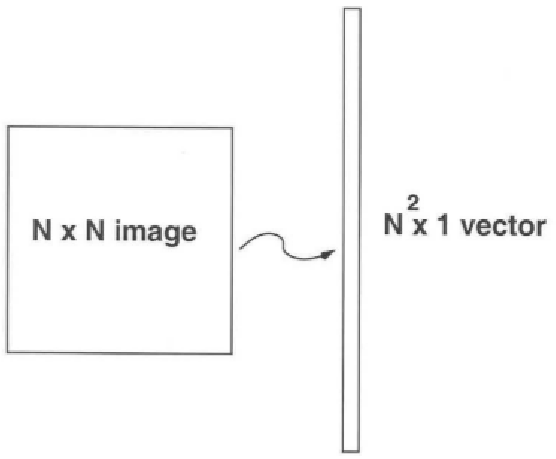
Among the many Face Detection Methods, Appearance-Based Face Detection Methods are the most popular as they provide better accuracy and performance. In general, the Appearance-Based Method relies on techniques from statistical analysis and machine learning to find the relevant characteristics of face images.
The Appearance-Based model further divided into sub-methods for the use of face detection which are as follows:

1. EigenFaces Method

PCA (Principal Component Analysis) is a dimensionality reduction technique that was proposed by Pearson in 1901. It uses Eigenvalues and EigenVectors to reduce dimensionality and project a training sample/data on small feature space. Let's look at the algorithm in more detail (in a face recognition perspective).

Training Algorithm

Let's Consider a set of m images of dimension N*N (training images)
We first convert these images into vectors of size N2 such that:



Now we calculate the average of all these face vectors and subtract it from each vector.

Now we take all face vectors so that we get a matrix of size of N2 * M.

Now, we find the Covariance matrix by multiplying A with AT. A has dimensions N2 * M, thus AT has dimensions M * N2. When we multiplied this gives us matrix of N2 * N2, which gives us N2 eigenvectors of N2 size which is not computationally efficient to calculate. So we calculate our covariance matrix by multiplying AT and A. This gives us M * M matrix which has M (assuming M << N2) eigenvectors of size M.

In this step we calculate eigenvalues and eigenvectors of the above covariance matrix using the formula below.

$$A^T A \nu_i = \lambda_i \nu_i$$

$$A A^T A \nu_i = \lambda_i A \nu_i$$

$$C' u_i = \lambda_i u_i$$

where, $C' = AA^T$ and $u_i = A\nu_i$.

From the above statement It can be concluded that C' and C have the same eigenvalues and their eigenvectors are related by the equation $u_i = A\nu_i$. Thus, the M eigenvalues (and eigenvectors) of the covariance matrix give the M largest eigenvalues(and eigenvectors) of C'.

Now we calculate Eigenvector and Eigenvalues of this reduced covariance matrix and map them into the C' by using the formula $u_i = A\nu_i$.

Now we select the K eigenvectors of C' corresponding to the K largest eigenvalues (where K < M). These eigenvectors have size N2.

In this step we used the eigenvectors that we got in the previous step. We take the normalized training faces (face – average face) x_i and represent each face vector in the linear combination of the best K eigenvectors (as shown in the diagram below).

$$x_i - \psi = \sum_{j=1}^{K} w_j u_j$$

These u_j are called EigenFaces.

Now, we take the coefficient of eigenfaces and represent the training faces in the form of a vector of those coefficients.

Testing/Detection Algorithm

Given an unknown face y, we need to first preprocess the face to make it centered in the image and have the same dimensions as the training face.

Now, we subtract the face from the average face .

Now, we project the normalized vector into eigenspace to obtain the linear combination of eigenfaces.

From the above projection, we generate the vector of the coefficient
We take the vector generated in the above step and subtract it from the training image to get the minimum distance between the training vectors and testing vectors

If this distance is below tolerance level Tr, then it is recognised with l face from the training image, else the face is not matched from any faces in the training set.

2. Geometric Face Recognition

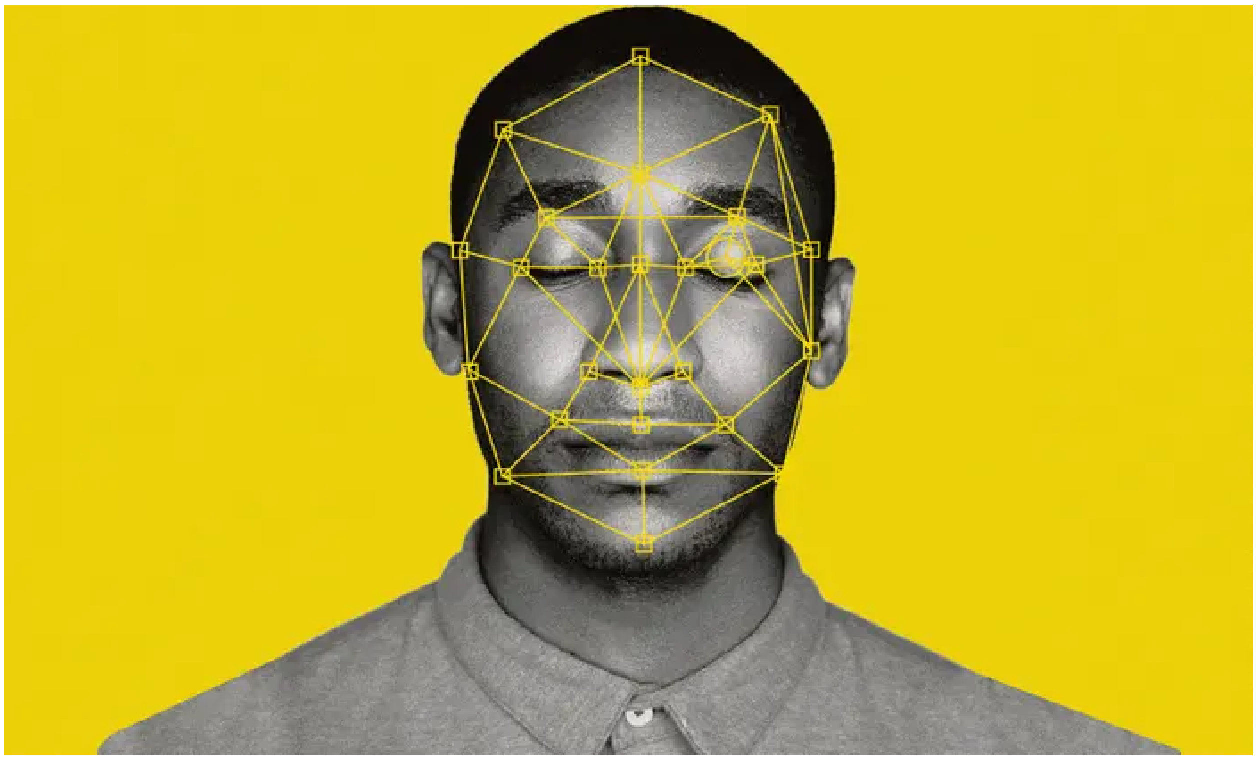
Geometric face recognition is perhaps one of the most intuitive approaches to facial recognition.

This approach works via constructing marker points on the face which include; position of eyes, position of the nose, and the position of ears.

This approach works by creating the following parameters: Distance between the (marker) points and angles between them.

The recognition is then performed by calculating the Euclidean distance between the feature vectors of a probe and the reference image.

The advantage of this approach is that it is robust against changes in illumination, but constructing the feature vectors can often be a challenging endeavor.



3. Cascades Method

This method works by scanning a picture from the top left corner and moving down across small blocks of data while scanning them.

The OpenCV cascade works by breaking the problem of detecting faces into multiple stages. For each block it encounters, it performs a rough and quick test.

If the test is successful, that is, if the block passes the test, a more sophisticated test is performed.

This Haar Cascade Classifier algorithm needs a lot of positive images, that is, images with faces, and also a lot of negative images, that is, images without faces. The algorithm then extracts features from these images. After the calculation of relevant features, the algorithm applies each and every feature on all the training images. For each feature, the algorithm finds the best threshold which will classify the faces to positive and negative. Features with minimum error rates are chosen.

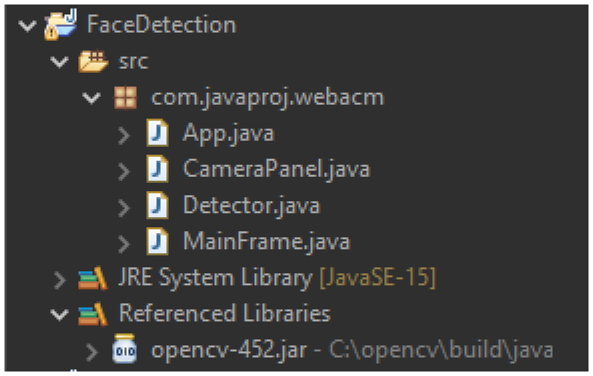
In a nutshell, the algorithm takes an image, applies features to it, and checks whether it is a face or not.

An advantage of this method is that most of the images will return negative during the first few stages, in essence, the algorithm will not waste a lot of time testing all features on it.

Program for Face Detection

Program Description

Our program has 4 files in it:



```
FaceDetection
  src
    com.javaproj.webacm
      App.java
      CameraPanel.java
      Detector.java
      MainFrame.java
  JRE System Library [JavaSE-15]
  Referenced Libraries
    opencv-452.jar - C:\opencv\build\java
```

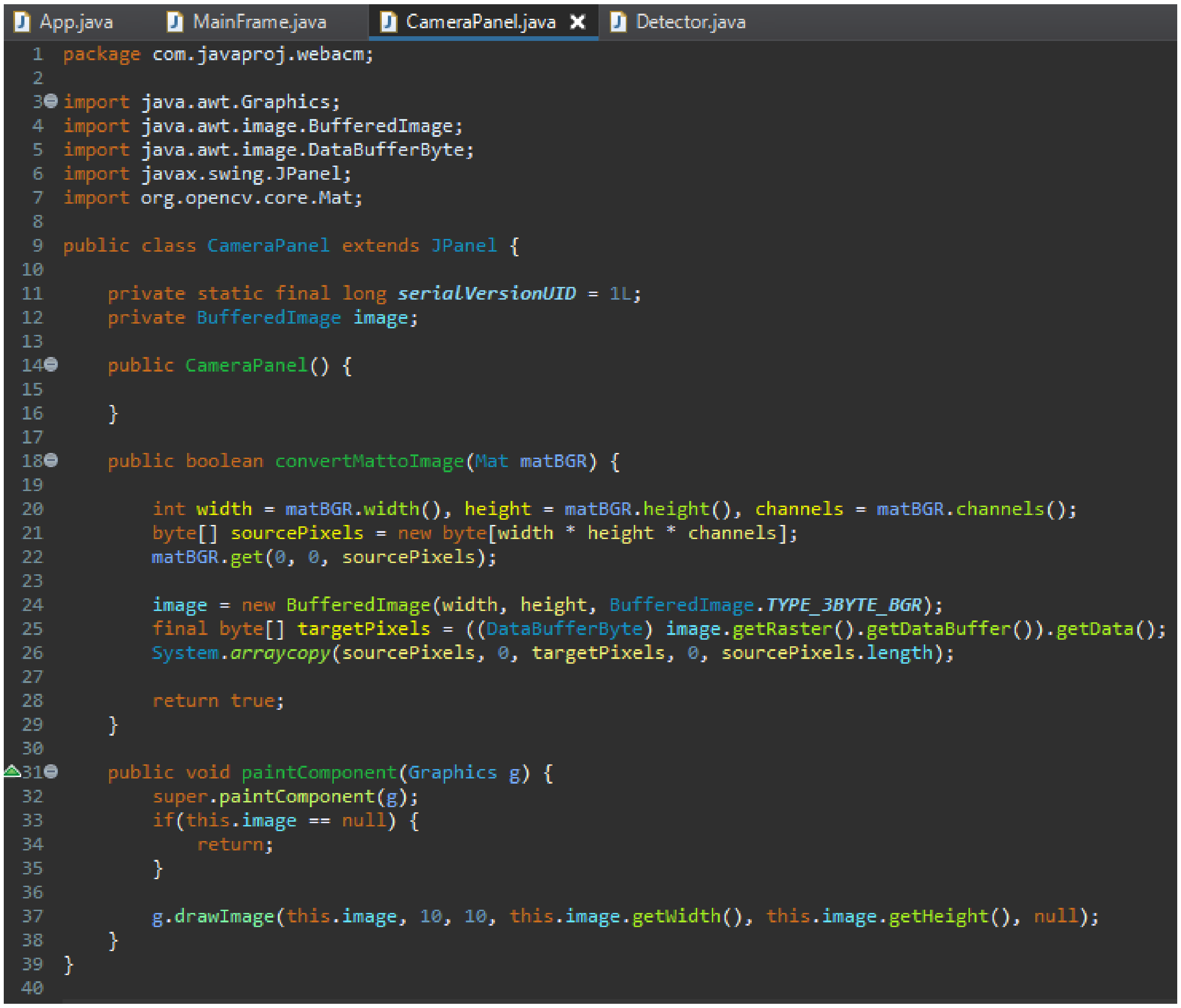
App.java
It is the main application which should be run on loading the Java Project. It opens the Face Detection Window according to the device using SystemLookAndFeel.

Mainframe.java
It is the core application which is connected to the other 2 processing applications. It defines the Face Detection Window width and height, opens the WebCam using CameraPanel and tries to detect a face using the Detector. If it successfully detects a face, it repaints the camera frame, drawing a red box around the face.

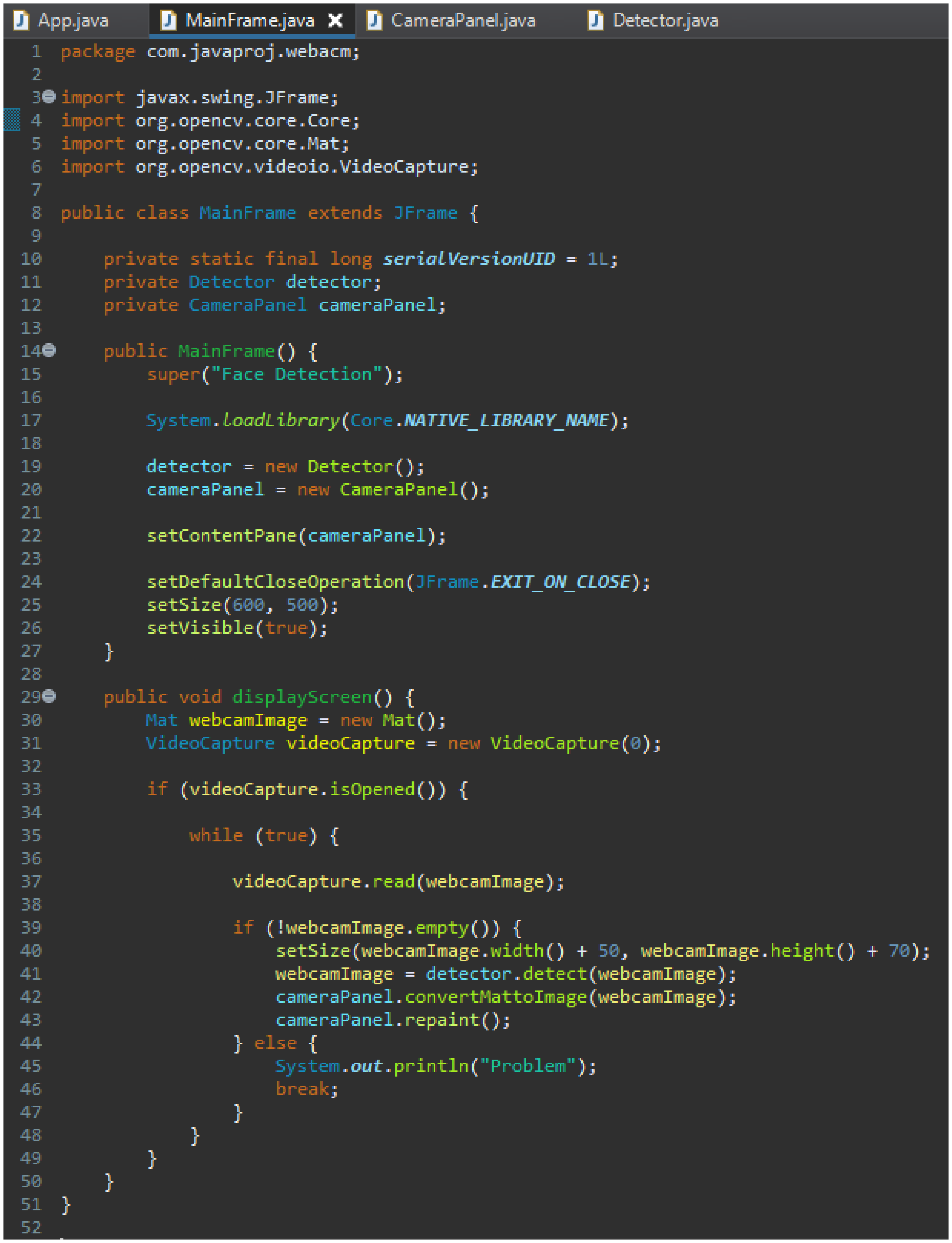
Detector.java
It converts the input feed images from the WebCam from colored to grayscale, and then equalises it. It looks for a face-like structure using the file ‘haarcascade_frontalface_alt.xml’. If it is successful in finding a face, it creates a rectangle around it and returns the finished product back as a colored image.

CameraPanel.java
It converts the input feed from the WebCam to a matrix. It starts reading the image from the top-left corner and gradually moves from left to right, top to bottom to finish (Cascade Style Face Detection Method) and finishes at the bottom-right corner. It also paints around the detected faces.

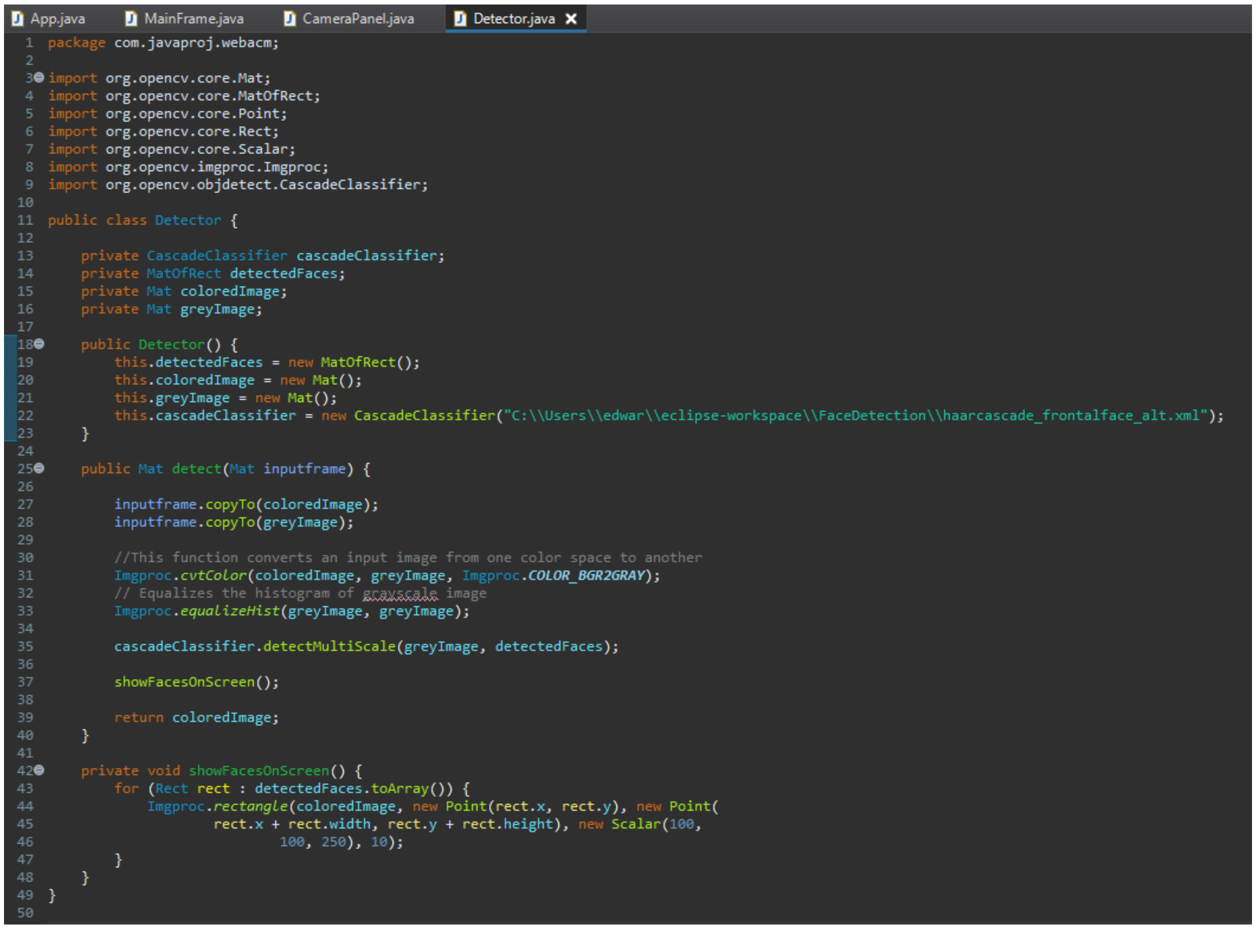
Code Snippets



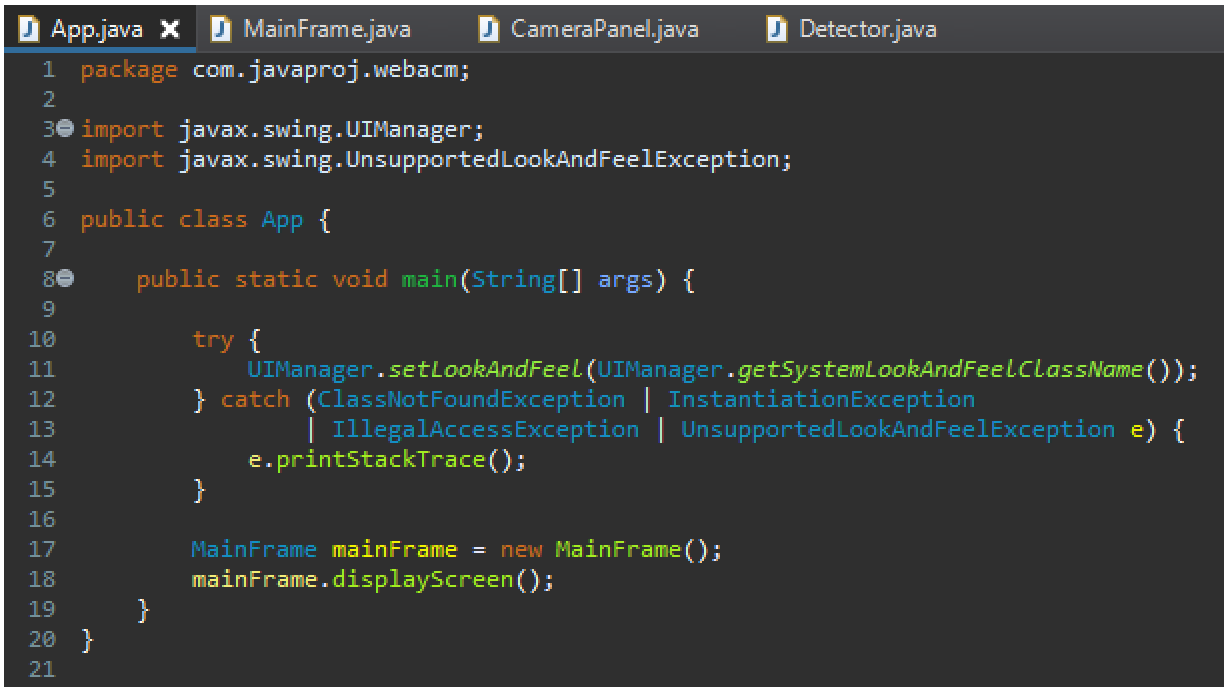
```java
package com.javaproj.webacm;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import javax.swing.JPanel;
import org.opencv.core.Mat;

public class CameraPanel extends JPanel {

    private static final long serialVersionUID = 1L;
    private BufferedImage image;

    public CameraPanel() {

    }

    public boolean convertMattoImage(Mat matBGR) {

        int width = matBGR.width(), height = matBGR.height(), channels = matBGR.channels();
        byte[] sourcePixels = new byte[width * height * channels];
        matBGR.get(0, 0, sourcePixels);

        image = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
        final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
        System.arraycopy(sourcePixels, 0, targetPixels, 0, sourcePixels.length);

        return true;
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        if(this.image == null) {
            return;
        }

        g.drawImage(this.image, 10, 10, this.image.getWidth(), this.image.getHeight(), null);
    }
}

```


App.java | MainFrame.java | CameraPanel.java | Detector.java

```java
package com.javaproj.webacm;

import javax.swing.JFrame;
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.videoio.VideoCapture;

public class MainFrame extends JFrame {

    private static final long serialVersionUID = 1L;
    private Detector detector;
    private CameraPanel cameraPanel;

    public MainFrame() {
        super("Face Detection");

        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

        detector = new Detector();
        cameraPanel = new CameraPanel();

        setContentPane(cameraPanel);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(600, 500);
        setVisible(true);
    }

    public void displayScreen() {
        Mat webcamImage = new Mat();
        VideoCapture videoCapture = new VideoCapture(0);

        if (videoCapture.isOpened()) {

            while (true) {

                videoCapture.read(webcamImage);

                if (!webcamImage.empty()) {
                    setSize(webcamImage.width() + 50, webcamImage.height() + 70);
                    webcamImage = detector.detect(webcamImage);
                    cameraPanel.convertMattoImage(webcamImage);
                    cameraPanel.repaint();
                } else {
                    System.out.println("Problem");
                    break;
                }
            }
        }
    }
}

```



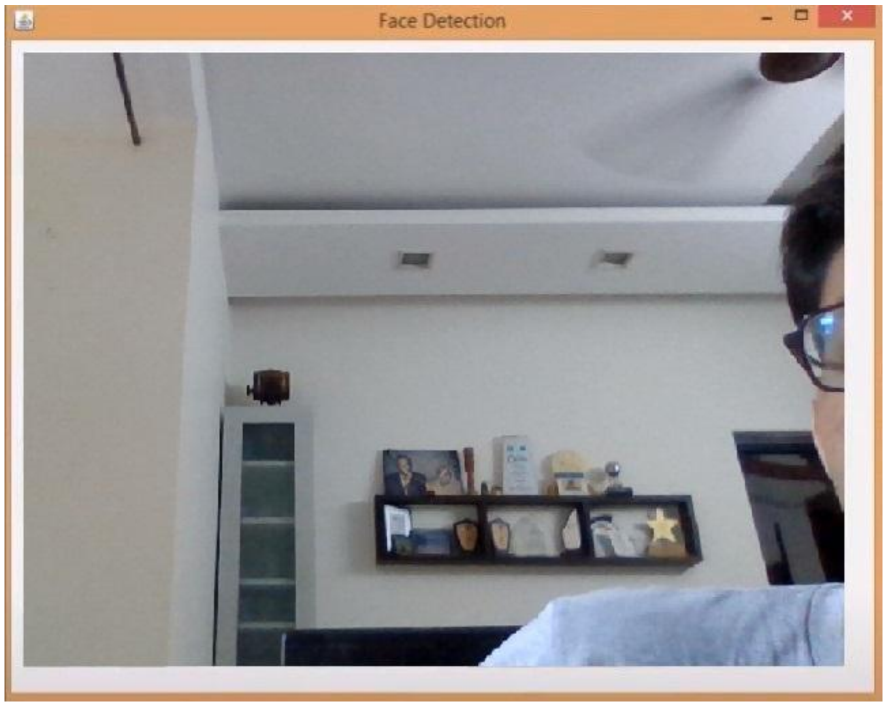
```java
package com.javaproj.webacm;

import org.opencv.core.Mat;
import org.opencv.core.MatOfRect;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;

public class Detector {

    private CascadeClassifier cascadeClassifier;
    private MatOfRect detectedFaces;
    private Mat coloredImage;
    private Mat greyImage;

    public Detector() {
        this.detectedFaces = new MatOfRect();
        this.coloredImage = new Mat();
        this.greyImage = new Mat();
        this.cascadeClassifier = new CascadeClassifier("C:\\Users\\edwar\\eclipse-workspace\\FaceDetection\\haarcascade_frontalface_alt.xml");
    }

    public Mat detect(Mat inputframe) {

        inputframe.copyTo(coloredImage);
        inputframe.copyTo(greyImage);

        //This function converts an input image from one color space to another
        Imgproc.cvtColor(coloredImage, greyImage, Imgproc.COLOR_BGR2GRAY);
        // Equalizes the histogram of grayscale image
        Imgproc.equalizeHist(greyImage, greyImage);

        cascadeClassifier.detectMultiScale(greyImage, detectedFaces);

        showFacesOnScreen();

        return coloredImage;
    }

    private void showFacesOnScreen() {
        for (Rect rect : detectedFaces.toArray()) {
            Imgproc.rectangle(coloredImage, new Point(rect.x, rect.y), new Point(
                    rect.x + rect.width, rect.y + rect.height), new Scalar(100,
                            100, 250), 10);
        }
    }
}
```

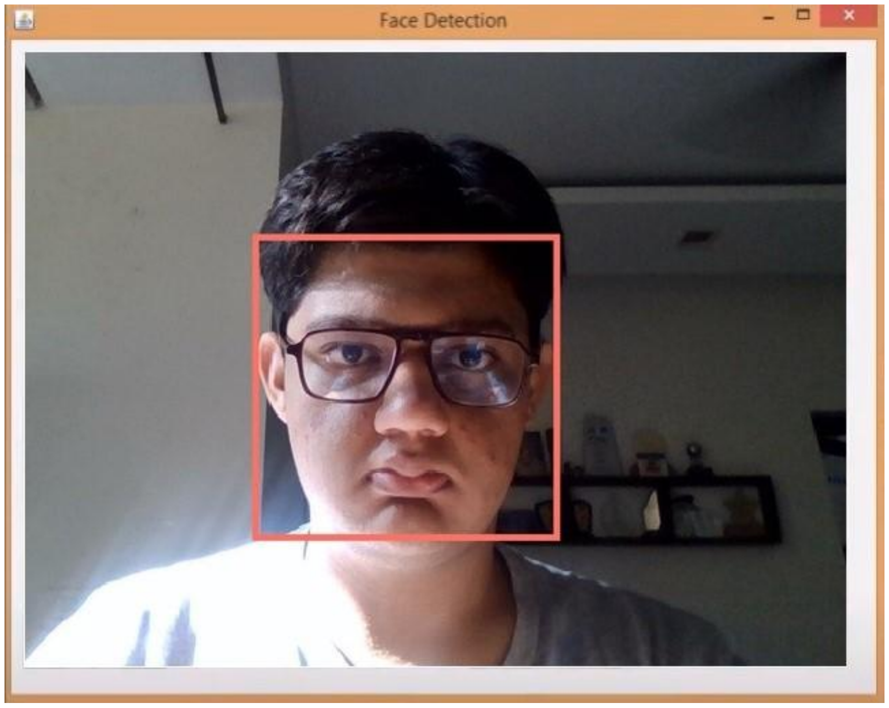


```java
package com.javaproj.webacm;

import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

public class App {

    public static void main(String[] args) {

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (ClassNotFoundException | InstantiationException
                | IllegalAccessException | UnsupportedLookAndFeelException e) {
            e.printStackTrace();
        }

        MainFrame mainFrame = new MainFrame();
        mainFrame.displayScreen();
    }
}
```

Program Output

The program opens the WebCam of the device and tries to detect a face in real time. If it recognises a face, it highlights it in a box.





References

1. https://en.wikipedia.org/wiki/Facial_recognition_system
2. https://opencv-java-tutorials.readthedocs.io/en/latest/06-face-detection-and-tracking.html
3. https://www.tutorialspoint.com/opencv/opencv_face_detection_in_picture.htm
4. https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9#:~:text=The%20primary%20aim%20of%20face,in%20an%20image%20or%20not.&text=It%20is%20widely%20used%20in,the%20images%20and%20recognise%20them.
5. https://en.wikipedia.org/wiki/Cascading_classifiers
6. https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
7. https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d (Python)