Name: Amar Yele

Computer Engineering 2024

Name of Task: Real Time Video Stream using OpenCV with Java

Requirements:

- Real-Time Video Stream Analysis
- Task: Create a system to analyze and process real-time video streams.
- Requirements: Use Java along with libraries like OpenCV for video processing and analysis
- Implement features such as object detection, motion tracking, or facial recognition.
- Develop a frontend to visualize processed video streams and analysis results
- Deliverables
- Source code for video processing and analysis.
- Integration with video sources (e.g., cameras or video feeds).

Source Code:

- **VideoStreamMain.java File**

```
package com.openCV;

import org.opencv.core.Core;

import org.opencv.core.Mat;
import org.opencv.core.MatOfRect;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;
import org.opencv.videoio.VideoCapture;

// JAVA FX
import javafx.application.Application;
import javafx.embed.swing.SwingFXUtils;
import javafx.scene.Scene;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

import javax.swing.SwingUtilities;
import java.awt.image.BufferedImage;

//Motion Tracking
import org.opencv.video.BackgroundSubtractorMOG2;
import org.opencv.video.Video;

//Saving vdo to File
import org.opencv.videoio.VideoWriter;
//import org.opencv.videoio.VideoWriter_fourcc;
```

```
//Adding buttons
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;

public class VideoStreamMain extends Application{

    static {
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
    }

    private VideoCapture capture;
    private CascadeClassifier faceDetector;
    private CascadeClassifier pedestrianDetector;
    private ImageView imageView;

  //Motion Tracking
    private BackgroundSubtractorMOG2 bgSubtractor;

    //Saving vdo to file
    private VideoWriter videoWriter;


    @Override
    public void start(Stage primaryStage) {
            //Load the Face Detection Model
        faceDetector = new
CascadeClassifier("D:\\Fullstack\\VDO_Streamer\\opencv\\build\\etc\\haarcascades\\haarcascade_frontalface_def
ault.xml");
        pedestrianDetector = new
CascadeClassifier("D:\\Fullstack\\VDO_Streamer\\opencv\\build\\etc\\haarcascades\\haarcascade_fullbody.xml");

      //Motion Tracking
        // Initialize the Background Subtractor
//        MOG2 stands for Mixture of Gaussians
        bgSubtractor = Video.createBackgroundSubtractorMOG2();

        //Saving vdo to file
//        videoWriter = new VideoWriter("output.avi", VideoWriter.fourcc('M', 'J', 'P', 'G'), 10, new Size(640, 480));

        if (faceDetector.empty()|| pedestrianDetector.empty()) {
            System.err.println("Error loading cascade file for face detection.");
            System.exit(0);
        }

        //Initialize the VDO capture from the Default Camera
        capture = new VideoCapture(0);

        if (!capture.isOpened()) {
            System.err.println("Error opening video capture.");
            System.exit(0);
        }

        imageView = new ImageView();

        //adding the Buttons
        Button startButton = new Button("Start");
        Button stopButton = new Button("Stop");
```

```
        startButton.setOnAction(e -> new Thread(this::processVideo).start());
        stopButton.setOnAction(e -> {
            capture.release();
            System.exit(0);
        });

        HBox controls = new HBox(10, startButton, stopButton);
        BorderPane root = new BorderPane();
//        BorderPane root = new BorderPane(imageView);
        root.setCenter(imageView);
        root.setBottom(controls);

        Scene scene = new Scene(root, 800, 600);
        primaryStage.setScene(scene);
        primaryStage.setTitle("Real Time Video Stream With Face Detection");
        primaryStage.show();



        // Define the codec and create a VideoWriter object
        int codec = VideoWriter.fourcc('M', 'J', 'P', 'G');  // MJPG codec
        videoWriter = new VideoWriter("D:/Fullstack/VDO_Streamer/output.avi", codec, 30, new Size(640, 480));


       // Start the video processing thread
        new Thread(this::processVideo).start();
    }

    private void processVideo() {
        Mat frame = new Mat();
        Mat grayFrame = new Mat();
        MatOfRect faces = new MatOfRect();
        MatOfRect pedestrians = new MatOfRect();
        Mat foregroundMask = new Mat(); // This will store the foreground mask

        while (capture.read(frame)) {
            // Convert the frame to grayscale
            Imgproc.cvtColor(frame, grayFrame, Imgproc.COLOR_BGR2GRAY);
            Imgproc.equalizeHist(grayFrame, grayFrame);
            Imgproc.GaussianBlur(grayFrame, grayFrame, new Size(5, 5), 0);

            // Apply the background subtractor to get the foreground mask
            bgSubtractor.apply(frame, foregroundMask);

           // Detect faces in the frame
            faceDetector.detectMultiScale(grayFrame, faces, 1.1, 3, 0, new Size(30, 30), new Size(300, 300));
            pedestrianDetector.detectMultiScale(grayFrame, pedestrians, 1.1, 3, 0, new Size(30, 30), new Size(300,
300));

            //counting the head in frame
            System.out.println("Number of faces detected: " + faces.toArray().length);

            // Find contours in the foreground mask
            java.util.List<org.opencv.core.MatOfPoint> contours = new java.util.ArrayList<>();
            Imgproc.findContours(foregroundMask, contours, new Mat(), Imgproc.RETR_EXTERNAL,
Imgproc.CHAIN_APPROX_SIMPLE);
```

```
            // Count the number of contours
            int movingObjectCount = contours.size();
            System.out.println("Number of moving objects detected: " + movingObjectCount);

            // Threshold the foreground mask
            Imgproc.threshold(foregroundMask, foregroundMask, 25, 255, Imgproc.THRESH_BINARY);

            // Filter contours based on area
            java.util.List<org.opencv.core.MatOfPoint> filteredContours = new java.util.ArrayList<>();
            for (org.opencv.core.MatOfPoint contour : contours) {
                if (Imgproc.contourArea(contour) > 500) { // Minimum area threshold
                    filteredContours.add(contour);
                }
            }

           // Draw rectangles around detected faces
            for (Rect rect : faces.toArray()) {
                Imgproc.rectangle(frame, rect.tl(), rect.br(), new Scalar(0, 255, 0), 3);
            }

            for (Rect rect : pedestrians.toArray()) {
                Imgproc.rectangle(frame, rect.tl(), rect.br(), new Scalar(255, 0, 0), 3);
            }

            // Save frame to video file
            videoWriter.write(frame);

           // Convert the processed frame to a BufferedImage
            BufferedImage image = matToBufferedImage(frame);

           // Update the JavaFX ImageView with the new frame
            javafx.scene.image.Image fxImage = SwingFXUtils.toFXImage(image, null);
            SwingUtilities.invokeLater(() -> imageView.setImage(fxImage));
        }

     // Release the video capture when done
        capture.release();
        videoWriter.release();
    }

    private BufferedImage matToBufferedImage(Mat mat) {
        int type = BufferedImage.TYPE_BYTE_GRAY;
        if (mat.channels() > 1) {
            type = BufferedImage.TYPE_3BYTE_BGR;
        }
        BufferedImage image = new BufferedImage(mat.width(), mat.height(), type);
        mat.get(0, 0, ((java.awt.image.DataBufferByte) image.getRaster().getDataBuffer()).getData());
        return image;
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

- **Pom.xml file**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.openCV</groupId>
  <artifactId>Video-Stream</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Video Stream</name>
  <description>Create a system to analyze and process real-time video streams</description>

</project>
```