Heart Rate Monitoring System Documentation

1. Overview

1.1 Purpose

The Heart Rate Monitoring System is a Python-based application that estimates a user's heart rate in real-time using remote photoplethysmography (rPPG). It processes video input from a webcam to detect a face, extract color channel signals from facial regions (forehead and cheeks), and compute heart rate through signal processing techniques. The system is intended for research, educational, or personal use in controlled environments and is not designed for medical-grade accuracy.

1.2 Features

- **Face Detection**: Uses dlib's frontal face detector and 68-point landmark predictor to identify facial regions.
- **Region of Interest (ROI) Selection**: Defines forehead and combined (forehead + cheeks) ROIs for signal extraction.
- **Signal Processing**: Applies bandpass filtering and Eulerian Video Magnification to enhance subtle color changes, followed by FFT for heart rate estimation.
- **Real-Time Visualization**:
    - Displays ROIs with colored rectangles.
    - Shows green and red channel images.
    - Overlays heart rate in beats per minute (bpm).
    - Plots heart rate history and filtered signals.
- **User Interaction**: Runs continuously until the user presses 'q' to quit.

1.3 Scope

The system supports a single user at a time, operates in stable lighting conditions with minimal motion, and requires a webcam and the dlib shape predictor model. It is not intended for clinical use or environments with significant lighting variations or movement.

2. Installation

2.1 System Requirements

- Hardware:
  - Computer with a 2.5 GHz dual-core CPU (minimum) and 4 GB RAM.
  - USB webcam with at least 640x480 resolution at 30 FPS.
  - 1 GB free disk space for dependencies and model file.
- Software:
  - Python 3.8 or higher.
  - Operating System: Windows 10/11, macOS 10.15+, or Ubuntu 20.04+.
  - CMake and a C++ compiler (for dlib compilation).

2.2 Dependencies

The following Python libraries are required (specified in requirements.txt):

```
opencv-python>=4.5.5.64

dlib>=19.22.0

numpy>=1.19.5

scipy>=1.7.3
```

Python's built-in collections module is used (no installation needed).

2.3 Installation Steps

1. Set Up Python Environment:
   - Verify Python 3.8+ is installed: python --version.
   - Create and activate a virtual environment:

- python -m venv venv
- source venv/bin/activate # On Windows: venv\Scripts\activate

2. **Install Dependencies:**
   - Create a requirements.txt file with the content above.
   - Install packages:
   - pip install -r requirements.txt
   - **Note for dlib:**
     - Ensure CMake and a C++ compiler are installed:
       - **Windows:** Install Visual Studio with C++ development tools.
       - **Linux:** sudo apt-get install build-essential cmake.
       - **macOS:** Install Xcode and brew install cmake.
     - For GPU support (optional), install CUDA/cuDNN and use pip install dlib --verbose.
3. **Download dlib Shape Predictor:**
   - Download shape_predictor_68_face_landmarks.dat.bz2 from http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2.
   - Extract to obtain shape_predictor_68_face_landmarks.dat.
   - Place it in the same directory as the Python script.
4. **Verify Installation:**
   - Run a test script to ensure libraries load correctly:
   - import cv2, dlib, numpy, scipy.signal

    - print("All libraries imported successfully!")

2.4 Example Directory Structure

heart_rate_monitor/

├── heart_rate_monitor.py # Main Python script

├── shape_predictor_68_face_landmarks.dat # dlib model file

├── requirements.txt # Dependency list

├── venv/ # Virtual environment

3. Usage

3.1 Running the Application

1. Ensure the webcam is connected and the shape predictor file is in the working directory.
2. Open a terminal in the project directory and activate the virtual environment:
3. source venv/bin/activate # On Windows: venv\Scripts\activate
4. Run the script:
5. python heart_rate_monitor.py
6. Position your face in front of the webcam, ensuring stable lighting and minimal movement.
7. Press 'q' to quit the application.

3.2 User Interface

- Main Video Feed:
    - Green rectangle: Forehead ROI.
    - Blue rectangle: Combined ROI (forehead + cheeks).
    - Top-left: Resized green and red channel images.

    - Top–right: Current heart rate (e.g., ”Heart Rate: 72 bpm”).
    - Message ”No face detected” if no face is found.
- **Bottom Strip**:
    - Left: Heart rate history graph (last 60 seconds, 40–200 bpm range).
    - Right: Filtered green and red signal plots (last 200 frames).
- **Window**: Titled ”Heart Rate Monitor”.

3.3 Configuration

- **Camera Index**: Adjust cv2.VideoCapture(1) in the code (try 0 or 1) if the default webcam index fails.
- **Amplification Factor**: Modify alpha = 10 in the code to adjust signal amplification.
- **Frame Rate**: The code assumes 30 FPS (fs = 30). Adjust if your camera differs significantly.

3.4 Expected Output

- The system detects your face, highlights ROIs, and displays heart rate updates every second.
- Graphs show heart rate trends and filtered signals in real–time.
- Example heart rate range: 60–100 bpm under normal conditions.

4. System Architecture

4.1 Components

The system is structured around the following modules:

1. **Setup and Initialization**:
    - Loads dlib models (face detector, shape predictor).
    - Configures a Butterworth bandpass filter (0.7–3 Hz).

   - Initializes buffers for signals, heart rates, and visualization history.

2. **ROI and Channel Processing**:
   - get_rois: Detects face, defines forehead and combined ROIs, draws rectangles.
   - get_channel_averages: Extracts green/red channel averages, displays channel images.

3. **Signal Processing**:
   - filter_signal: Applies bandpass filter to channel signals.
   - amplify_signal: Enhances signals using Eulerian Video Magnification.
   - get_heart_rate: Computes heart rate via FFT on red-green difference signal.
   - smooth_heart_rate: Averages heart rate estimates.

4. **Visualization**:
   - display_visualizations: Overlays heart rate text and renders graphs.

5. **Main Loop**:
   - Captures video frames, processes them through the pipeline, and updates the display.

4.2 Data Flow

1. **Input**: Webcam video frame (BGR format).
2. **Face Detection**: Convert to grayscale, detect face, extract landmarks.
3. **ROI Extraction**: Define ROIs, extract green/red channel averages.
4. **Signal Processing**:
   - Filter signals (bandpass).
   - Amplify signals (Eulerian magnification).

    - Buffer difference signal (red – green).
    - Compute heart rate via FFT every 30 frames.
    - Smooth heart rate estimates.
5. **Output**: Render frame with ROIs, heart rate text, and graphs.

4.3 Key Algorithms

- **Face Detection**: dlib's HOG-based frontal face detector.
- **Landmark Prediction**: dlib's 68-point shape predictor.
- **Signal Filtering**: 4th-order Butterworth bandpass filter (0.7–3 Hz).
- **Heart Rate Estimation**: FFT with Hamming window, selecting dominant frequency in 0.7–3 Hz.
- **Visualization**: OpenCV for ROI rectangles, text, and line graphs.

5. Technical Details

5.1 Dependencies

- **OpenCV**: Video capture, image processing, visualization.
- **dlib**: Face detection and landmark prediction.
- **NumPy**: Array operations, FFT calculations.
- **SciPy**: Butterworth filter and signal processing.
- **Python collections**: Deque for buffering.

5.2 Assumptions

- Webcam provides ~30 FPS at 640x480 or higher.
- Stable lighting and minimal subject motion.
- Single face in the frame.
- Shape predictor file is present.

5.3 Limitations

- Sensitive to lighting changes and motion.

- Assumes fixed frame rate (30 FPS).
- Computationally intensive due to real-time face detection.
- Not suitable for medical diagnostics.

6. Troubleshooting

6.1 Common Issues

1. "No face detected" persists:
   - Ensure proper lighting and face visibility.
   - Check webcam connection and index in cv2.VideoCapture(1).
2. dlib fails to install:
   - Install CMake and C++ compiler.
   - Use pip install dlib --verbose to diagnose errors.
3. Shape predictor file missing:
   - Download shape_predictor_68_face_landmarks.dat and place it in the working directory.
4. Low frame rate:
   - Reduce face detection frequency (e.g., every 5 frames).
   - Use a more powerful CPU or enable GPU for dlib.
5. Inaccurate heart rate:
   - Ensure stable lighting and minimal movement.
   - Adjust alpha (amplification factor) or filter parameters.

6.2 Debugging Tips

- Print frame rate: Add timing code to measure FPS.
- Log signals: Save avg_green, avg_red, or heart_rate to a file for analysis.
- Test webcam: Use a simple OpenCV script to verify camera access.

7. Future Enhancements

- **Adaptive Frame Rate**: Calculate actual FPS for accurate filtering.
- **Motion Compensation**: Track face movement to stabilize ROIs.
- **Signal Quality Metrics**: Indicate confidence in heart rate estimates.
- **Output Saving**: Save video or log heart rate data.
- **GUI**: Add a graphical interface for configuration.
- **Multi-User Support**: Detect and process multiple faces.

8. References

- dlib Documentation: http://dlib.net/
- OpenCV Documentation: https://docs.opencv.org/
- SciPy Signal Processing: https://docs.scipy.org/doc/scipy/reference/signal.html
- Eulerian Video Magnification: http://people.csail.mit.edu/mrub/vidmag/

9. Contact

For issues or contributions, contact the developer or raise an issue on the project's GitHub repository (if applicable).