Program Studi Teknik Informatika
Institut Teknologi Sumatera

Mata Kuliah: **Digital Signal Processing (IF3024)**
Nama Anggota:

1. **Attar Akram Abdillah (121140013)**
2. **Natasya Ate Malem Bangun (121140052)**

1 Pendahuluan

Proyek ini bertujuan untuk mengembangkan sebuah sistem yang menggabungkan pengukuran sinyal respirasi dan remote-photopletysmography (rPPG) menggunakan video webcam. Sistem ini memproses video secara real-time, menghasilkan dan menampilkan sinyal respirasi dan rPPG untuk analisis lebih lanjut. Pengukuran sinyal respirasi dan rPPG penting untuk aplikasi medis dan kesehatan seperti pemantauan kondisi pasien secara terus-menerus.

2 Deskripsi Sistem dan Algoritma

Sistem yang dikembangkan dalam proyek ini menggunakan video yang direkam oleh webcam untuk memproses data sinyal. Algoritma yang digunakan mencakup pemrosesan video untuk ekstraksi sinyal respirasi dan rPPG, menggunakan teknik-teknik pemfilteran sinyal untuk mengidentifikasi dan memisahkan komponen-komponen yang relevan dari data video yang didapat. Sistem akan terdiri dari tiga bagian, yaitu `respiration_module.py`, `rppg_module.py`, dan `main.py`.

2.1 respiration_module.py

Kode ini akan melakukan pembacaan sinyal pernapasan melalui deteksi dari video yang diinputkan ke dalam kode ini. Rincian singkat dari fungsi kode ini adalah sebagai berikut:

- Melakukan pemrosesan sinyal pernapasan dengan menggunakan mediapipe. Mediapipe digunakan dalam deteksi pose yang akan digunakan sebagai region of interest (ROI) untuk identifikasi pernapasan.
- Melakukan filterisasi dengan menggunakan median filter dan low-pass filter terhadap sinyal yang didapatkan.
- Menyediakan fungsional untuk melakukan inisialisasi model mediapipe yang akan digunakan untuk deteksi pose.

2.2 rppg_module.py

Kode ini akan melakukan pembacaan sinyal rPPG melalui deteksi dari video yang diinputkan, yang pada kasus ini dengan menggunakan feed video dari webcam secara real-time. Rincian singkat fungsi dari modul ini adalah sebagai berikut:

- Fokus pada melakukan ekstraksi sinyal rPPG dengan menggunakan face detection yang disediakan oleh mediapipe.
- Melakukan filterisasi sinyal (band-pass dan low-pass) untuk memproses sinyal hijau yang dominan pada analisis rPPG.
- Menghitung detak jantung berdasarkan puncak sinyal yang terdeteksi oleh kode.

2.3 main.py

Kode ini akan menjalankan kedua module yang sudah dibuat secara bersamaan. Rincian singkat dari kode ini adalah sebagai berikut:

- Mengelola input video menggunakan OpenCV dan memprosesnya untuk mengekstraksi sinyal respirasi dan rPPG (remote photoplethysmography).
- Mengimplementasikan threading untuk menangkap dan memproses frame secara real-time.
- Memvisualisasikan intensitas respirasi dan detak jantung yang dihasilkan dari sinyal rPPG.

3 Desain Filter dan Pemrosesan Sinyal

Dalam sistem ini, sinyal respirasi dan rPPG diekstraksi menggunakan berbagai jenis filter. Parameter filter yang digunakan dipilih berdasarkan analisis sinyal yang diperoleh, dan dipilih untuk meminimalkan noise serta memastikan sinyal yang dihasilkan mempunyai kualitas yang baik. Filter ini meliputi low-pass, median filter, dan band-pass filter yang menyesuaikan dengan rentang frekuensi yang relevan.

Filter yang Diterapkan pada Deteksi Pernapasan

Filter Median

```
def median_filter(data, kernel_size=5):
    """
    Applies a median filter to the input signal.

    Args:
        data (np.ndarray): Input signal to be filtered.
        kernel_size (int): Size of the median filter window (default is 5).

    Returns:
        np.ndarray: Filtered signal.
    """
    return medfilt(data, kernel_size)  # Apply median filter to smooth the signal
```

Kode 1: Implementasi Filter Median

Filter median adalah salah satu metode filtering nonlinear yang digunakan untuk mengurangi noise dalam sinyal, khususnya noise yang berupa lonjakan atau nilai ekstrim (outlier). Filter ini bekerja dengan mengganti setiap nilai dalam sinyal dengan nilai median dari sejumlah titik tetangga di sekitar nilai tersebut.

Filter Low-Pass

```
def low_pass_filter(data, cutoff, fs, order=5):
    """
    Applies a low-pass Butterworth filter to the input signal.

```

```
    Args:
        data (np.ndarray): Input signal to be filtered.
        cutoff (float): Cutoff frequency in Hz.
        fs (float): Sampling frequency in Hz.
        order (int): Order of the filter (default is 5).

    Returns:
        np.ndarray: Filtered signal.
    """
    nyquist = 0.5 * fs  # Nyquist frequency (half of the sampling frequency)
    normal_cutoff = cutoff / nyquist  # Normalize the cutoff frequency
    b, a = butter(order, normal_cutoff, btype='low', analog=False)  # Design the Butterworth filter
    return filtfilt(b, a, data)  # Apply the filter to the input data
```

Kode 2: Implementasi Filter Low-Pass

Filter low-pass adalah filter berbasis frekuensi yang dirancang untuk menyaring komponen frekuensi tinggi dalam sinyal, sehingga hanya komponen frekuensi rendah (yang relevan dengan respirasi) yang dipertahankan.

Filter yang Diterapkan pada Deteksi rPPG

Filter Band-Pass

```
def band_pass_filter(data, lowcut, highcut, fs, order=5):
    """
    Applies a band-pass Butterworth filter to the input signal.

    This function implements a band-pass filter that allows frequencies within the specified range
    (between `lowcut` and `highcut`) to pass through while attenuating frequencies outside this
    range.

    Args:
        data (np.ndarray): Input signal.
        lowcut (float): Lower cutoff frequency in Hz. Frequencies below this value are attenuated.
        highcut (float): Upper cutoff frequency in Hz. Frequencies above this value are attenuated.
        fs (float): Sampling frequency in Hz. Used to calculate the Nyquist frequency.
        order (int): Order of the filter (default is 5). A higher order results in a sharper cutoff
    .

    Returns:
        np.ndarray: Filtered signal.
    """
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    if len(data) <= max(len(a), len(b)):
        return np.array(data)  # Return raw data if too short for filtering
    return filtfilt(b, a, data)
```

Kode 3: Implementasi Filter Band-Pass

Filter band-pass digunakan untuk menyaring sinyal agar hanya frekuensi tertentu yang berkaitan dengan detak jantung (biasanya 0.7–4 Hz, yang setara dengan 42–240 detak per menit) dapat dilewatkan.

4 Penjelasan Kode

4.1 `respiration_module.py`

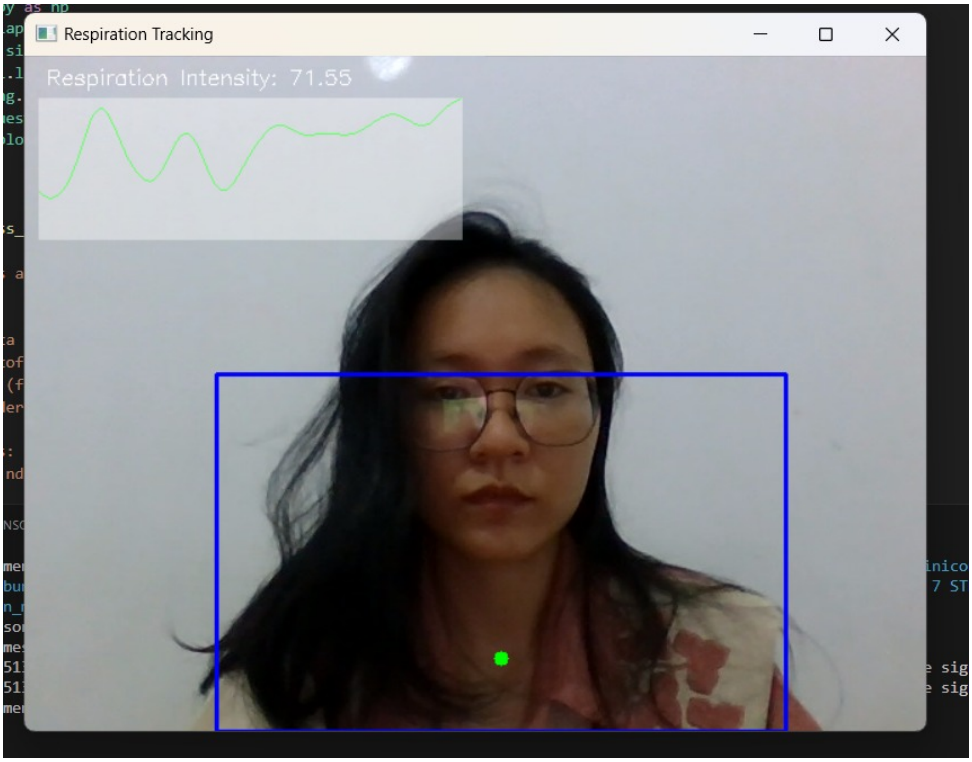

Figure 1: Tampilan Respiration

Kode ini berjalan dengan alur sebagai berikut:

1. **Pengambilan Webcam:** `process_respiration_from_webcam` mulai mengambil frame dari webcam.
2. **Deteksi Wajah:** MediaPipe mendeteksi wajah dan mengekstrak kotak pembatas wajah.
3. **Ekstraksi RGB:** `get_respiration_roi` menghitung nilai rata-rata kanal RGB (terutama kanal hijau) dari area wajah yang terdeteksi.
4. **Penyaringan Sinyal:** Sinyal pernapasan (intensitas hijau) disaring menggunakan Median Filter dan Low-Pass Filter untuk menghilangkan noise dan mempertahankan frekuensi relevan.
5. **Perhitungan Sinyal Pernapasan:** Menghitung intensitas pernapasan dari sinyal yang telah difilter.
6. **Visualisasi Real-Time:** Tampilan video dilapisi dengan grafik sinyal pernapasan yang telah difilter dan intensitas pernapasan yang ditampilkan di layar.
7. **Plotting Sinyal:** Setelah loop selesai, sinyal pernapasan yang telah difilter dipetakan dalam grafik akhir.

Hal ini dimungkinkan karena beberapa fungsi yang terdapat pada file ini, yaitu `initialize_pose_landmarker`, `get_respiration_roi`, dan `process_respiration_from_webcam` yang dirincikan berikut ini:

4.1.1 `initialize_pose_landmarker`

Fungsi ini menginisialisasi model atau objek yang digunakan untuk mendeteksi titik-titik pose tubuh.

```python
def initialize_pose_landmarker():
    """
    Initializes and returns the MediaPipe Pose Landmarker.

    Returns:
        PoseLandmarker: The MediaPipe PoseLandmarker object.
    """
    PoseLandmarker = mp.tasks.vision.PoseLandmarker
    BaseOptions = mp.tasks.BaseOptions
    PoseLandmarkerOptions = mp.tasks.vision.PoseLandmarkerOptions
    VisionRunningMode = mp.tasks.vision.RunningMode

    model_path = download_pose_model()

    options = PoseLandmarkerOptions(
        base_options=BaseOptions(model_asset_path=model_path),
        running_mode=VisionRunningMode.IMAGE,
        num_poses=1,
        min_pose_detection_confidence=0.5,
        min_pose_presence_confidence=0.5,
        min_tracking_confidence=0.5
    )
    return PoseLandmarker.create_from_options(options)
```

4.1.2 `get_respiration_roi`

Fungsi ini bertugas untuk menentukan Region of Interest (ROI) pada gambar atau frame untuk menganalisis sinyal pernapasan.

```python
def get_respiration_roi(image, pose_landmarker, scale_factor=0.7):
    """
    Detects the shoulders and returns the region of interest (ROI) for respiration signal
    extraction.

    Args:
        image (np.ndarray): Input video frame (image).
        pose_landmarker: MediaPipe pose detector object.
        scale_factor (float): Scaling factor for ROI size (default is 0.7).

    Returns:
        tuple: Coordinates of the ROI (left_x, top_y, right_x, bottom_y, center).
    """
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=image_rgb)
    detection_result = pose_landmarker.detect(mp_image)

    if not detection_result.pose_landmarks:
        raise ValueError("No pose detected in the frame!")

    landmarks = detection_result.pose_landmarks[0]
    height, width = image.shape[:2]
    left_shoulder = landmarks[11]
    right_shoulder = landmarks[12]

    center_x = int((left_shoulder.x + right_shoulder.x) * width / 2)
    center_y = int((left_shoulder.y + right_shoulder.y) * height / 2)

    x_size = int(abs(left_shoulder.x - right_shoulder.x) * width * scale_factor)
    y_size = x_size

    left_x = max(0, center_x - x_size)
    right_x = min(width, center_x + x_size)
```

```
    top_y = max(0, center_y - y_size)
    bottom_y = min(height, center_y + y_size)

    return (left_x, top_y, right_x, bottom_y, (center_x, center_y))
```

4.1.3 `process_respiration_from_webcam`

Fungsi ini adalah fungsi utama yang memproses sinyal pernapasan secara real-time menggunakan umpan video dari webcam.

```
def process_respiration_from_webcam(video):
    """
    Processes respiration signal in real-time from a webcam feed.

    Args:
        video: Video capture object for the webcam feed.
    """
    cap = video
    pose_landmarker = initialize_pose_landmarker()
    respiration_signal = []

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        try:
            left_x, top_y, right_x, bottom_y, center = get_respiration_roi(frame, pose_landmarker)
            roi = frame[top_y:bottom_y, left_x:right_x]
            avg_intensity = np.mean(roi[:, :, 1])
            respiration_signal.append(avg_intensity)

            if len(respiration_signal) > 30:
                filtered_signal = median_filter(respiration_signal, kernel_size=5)
                filtered_signal = low_pass_filter(filtered_signal, cutoff=0.8, fs=30)

                if len(filtered_signal) > 33:
                    ...
        except ValueError:
            continue

        cv2.imshow('Respiration Tracking', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
    plotting_respiration_signals(respiration_signal)
```

4.2 `rppg_module.py` Kode ini berjalan dengan alur sebagai berikut

Kode ini bekerja dengan alur sebagai berikut:

1. **Pengambilan Video Webcam**: `process_rppg_from_webcam` mulai mengambil frame dari webcam.
2. **Deteksi Wajah**: MediaPipe mendeteksi wajah dan mengekstrak kotak pembatas.
3. **Ekstraksi RGB**: `extract_rgb_signals` menghitung nilai rata-rata RGB dari wilayah wajah yang terdeteksi.

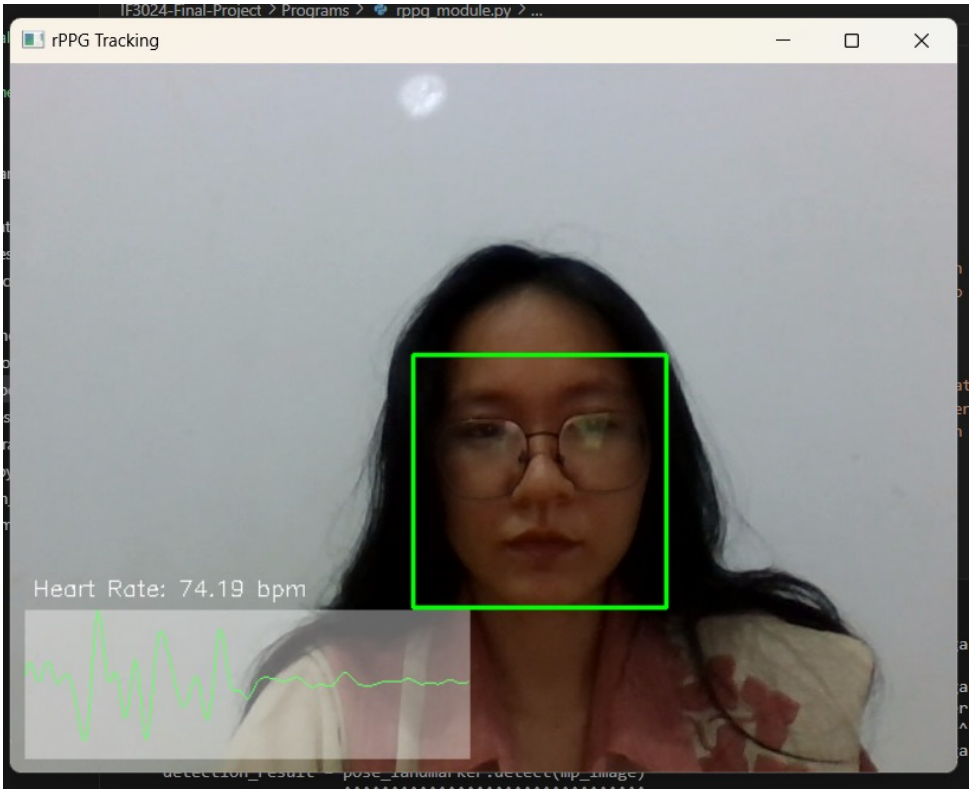

Figure 2: Tampilan rPPG

4. **Penyaringan Sinyal**: Kanal hijau (`g_signals`) disaring dengan `band_pass_filter` untuk mempertahankan frekuensi yang terkait dengan detak jantung.

5. **Perhitungan Detak Jantung**: Puncak-puncak dalam sinyal yang disaring diidentifikasi, dan detak jantung dihitung.

6. **Visualisasi Real-Time**: Umpan video ditumpangkan dengan grafik sinyal yang disaring dan detak jantung ditampilkan dalam bpm.

7. **Pemplotan Sinyal**: Setelah loop berakhir, sinyal yang disaring terakhir dipetakan dengan puncaknya yang ditandai.

Hal ini dimungkinkan karena beberapa fungsi yang terdapat pada file ini, yaitu `extract_rgb_signals` dan `process_rppg_from_webcam` yang dirincikan di bawah ini.

4.2.1 extract_rgb_signals

Fungsi ini melakukan komputasi nilai rerata RGB yang didapatkan dari ROI yang ditentukan pada frame, sesuai dengan wajah yang terdeteksi.

```
def extract_rgb_signals(frame, bbox):
    """
    Extracts the average RGB signals from the bounding box region of the frame.

    This function calculates the average signal intensity for each of the Red, Green, and Blue
    channels
    in the specified bounding box region of the input frame.

    Args:
        frame (np.ndarray): Input video frame.
```

```
        bbox (tuple): Bounding box coordinates (x, y, width, height), where (x, y) is the top-left
                      corner and (width, height) defines the dimensions of the bounding box.

    Returns:
        tuple: Average RGB values as (R, G, B), representing the average signal intensities for the
               respective channels.
    """
    x, y, width, height = bbox
    roi = frame[y:y + height, x:x + width]
    r_signal = np.mean(roi[:, :, 2])  # Red channel
    g_signal = np.mean(roi[:, :, 1])  # Green channel
    b_signal = np.mean(roi[:, :, 0])  # Blue channel
    return r_signal, g_signal, b_signal
```

4.2.2 process_rppg_from_webcam

Fungsi ini melakukan pemrosesan video yang diinputkan, contohnya saja dengan video real-time dari webcam. Fungsi ini melakukan pemrosesan dari nilai sinyal RGB yang ditangkap dan melakukan pemrosesan sinyal dengan melakukan kalkulasi dan menampilkan detak jantung.

```
def process_rppg_from_webcam(video):
    """
    Processes rPPG signals in real-time from a webcam feed.

    This function captures frames from a webcam feed, detects faces using MediaPipe's Face
    Detection
    module, and extracts RGB signals from the face region. It visualizes the heart rate in real-
    time
    by filtering the extracted green channel signal and detecting peaks.

    Args:
        video (cv2.VideoCapture): The video capture object for the webcam feed.

    Returns:
        None
    """
    cap = video
    mp_face_detection = mp.solutions.face_detection
    face_detection = mp_face_detection.FaceDetection(model_selection=1, min_detection_confidence
    =0.5)

    r_signals, g_signals, b_signals = [], [], []

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        results = face_detection.process(frame_rgb)

        if results.detections:
            for detection in results.detections:
                bboxC = detection.location_data.relative_bounding_box
                h, w, _ = frame.shape
                x = int(bboxC.xmin * w)
                y = int(bboxC.ymin * h)
                width = int(bboxC.width * w)
                height = int(bboxC.height * h)

                # Extract RGB signals from the face bounding box
```

```
                r, g, b = extract_rgb_signals(frame, (x, y, width, height))
                r_signals.append(r)
                g_signals.append(g)
                b_signals.append(b)

                # Draw bounding box on the frame
                cv2.rectangle(frame, (x, y), (x + width, y + height), (0, 255, 0), 2)

        # Real-time visualization of signal intensity
        if len(g_signals) > 33:  # Ensure enough data for filtering
            graph_height, graph_width = 100, 300
            graph_top_left = (10, frame.shape[0] - graph_height - 10)
            overlay = frame.copy()

            # Apply band-pass filter
            filtered_signal = band_pass_filter(g_signals, lowcut=0.7, highcut=2.5, fs=30)

            # Normalize filtered signal to scale it properly for visualization
            min_signal = np.min(filtered_signal)
            max_signal = np.max(filtered_signal)
            range_signal = max_signal - min_signal

            if range_signal > 0:
                # Normalize the signal to the range [0, graph_height]
                scaled_signal = (filtered_signal - min_signal) / range_signal * graph_height
            else:
                scaled_signal = np.zeros_like(filtered_signal)

            # Only visualize the last 300 frames
            if len(scaled_signal) > 300:
                scaled_signal = scaled_signal[-300:]

            for i in range(1, len(scaled_signal)):
                start_point = (graph_top_left[0] + i - 1, graph_top_left[1] + graph_height - int(
    scaled_signal[i - 1]))
                end_point = (graph_top_left[0] + i, graph_top_left[1] + graph_height - int(
    scaled_signal[i]))
                cv2.line(overlay, start_point, end_point, (0, 255, 0), 1)

            # Draw visualization box
            cv2.rectangle(frame, graph_top_left, (graph_top_left[0] + graph_width, graph_top_left
    [1] + graph_height), (255, 255, 255), -1)
            frame = cv2.addWeighted(overlay, 0.6, frame, 0.4, 0)

            # Add text for live heart rate calculation
            peaks, _ = find_peaks(filtered_signal, prominence=0.5)
            heart_rate = 60 * len(peaks) / (len(filtered_signal) / 30) if len(filtered_signal) > 0
    else 0
            cv2.putText(frame, f"Heart Rate: {heart_rate:.2f} bpm", (15, graph_top_left[1] - 10),
    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

        # Display the frame
        cv2.imshow('rPPG Tracking', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

    plotting_rppg_signals(r_signals, g_signals, b_signals)
```

4.3 Main.py

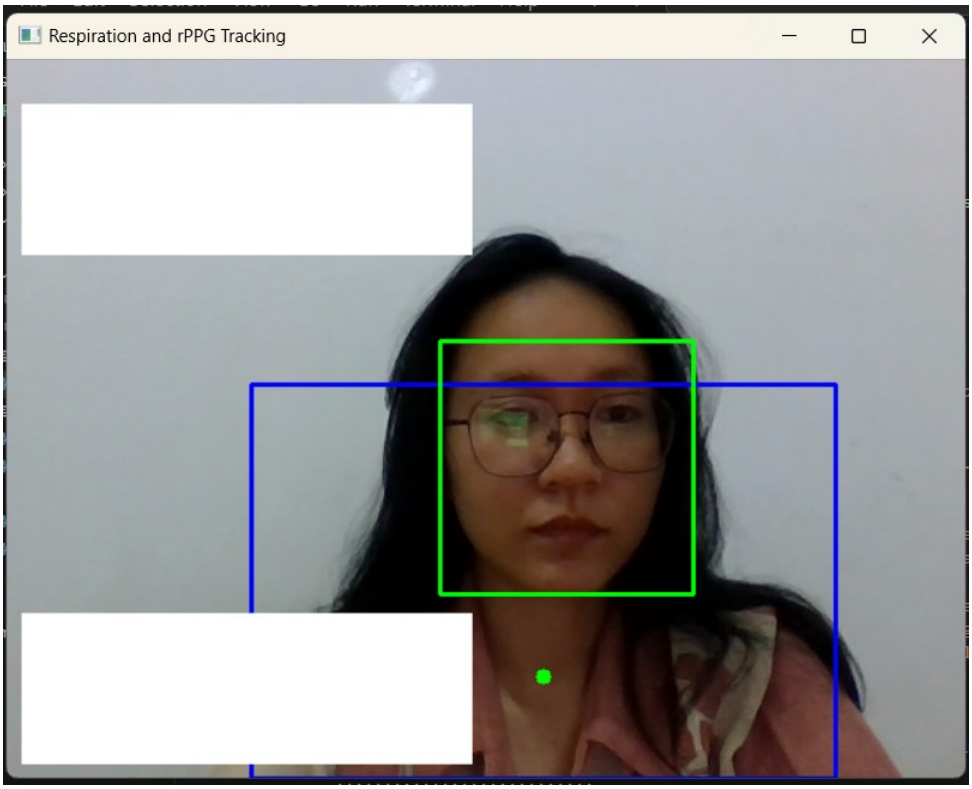

Figure 3: Tampilan Main

Kode ini bekerja dengan alur sebagai berikut ini:

- **Pengambilan Webcam**: OpenCV (cv2.VideoCapture) digunakan untuk membaca frame video secara real-time dari webcam.
- **Deteksi Wajah**: MediaPipe (mp.solutions.face$_detection$)$digunakanuntukmendeteksiwajahdalamsetiapfra$
  $Fungsiextract\_rgb\_signalsdigunakanuntukmenghitungrata-ratanilaiRGB(merah, hijau, biru)dariare$

- **Penyaringan Sinyal**: Sinyal dari kanal hijau (g_signals) difilter menggunakan fungsi band_pass_filter untuk mengisolasi frekuensi detak jantung.
- **Perhitungan Detak Jantung**: Puncak (peaks) dalam sinyal yang sudah difilter dihitung untuk memperkirakan detak jantung dalam satuan beats per minute (bpm).
- **Visualisasi Grafis**: Grafik sinyal dan estimasi bpm ditampilkan secara langsung pada frame video untuk memberikan visualisasi secara real-time.

Hal ini dimungkinkan karena beberapa fungsi yang terdapat pada file ini, diantaranya adalah `__init__` dan `process_frames` yang dirincikan di bawah ini.

4.3.1 `__init__`

Fungsi ini adalah konstruktor untuk kelas SharedVideoProcessor. Fungsi ini akan dipanggil ketika objek dari kelas SharedVideoProcessor dibuat.

```
1 def __init__(self):
2     """
3     Initializes the video capture, pose landmarker, and face detection,
```

```
    and sets up signal storage.
    """
    self.cap = cv2.VideoCapture(0)  # Initialize the video capture with the default webcam
    self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)  # Set the frame width
    self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)  # Set the frame height
    self.cap.set(cv2.CAP_PROP_FPS, 30)  # Set the frames per second

    self.running = True  # Flag to indicate if video processing should continue
    self.frame_lock = threading.Lock()  # Lock for thread-safe access to frames
    self.current_frame = None  # Variable to store the current video frame

    # Initialize pose landmarker for respiration signal extraction
    self.pose_landmarker = initialize_pose_landmarker()

    # Initialize MediaPipe face detection
    self.mp_face_detection = mp.solutions.face_detection
    self.face_detection = self.mp_face_detection.FaceDetection(model_selection=1,
    min_detection_confidence=0.5)

    # Initialize lists to store the signals
    self.respiration_signal = []  # List to store respiration signals
    self.r_signals = []  # List to store red channel values for rPPG
    self.g_signals = []  # List to store green channel values for rPPG
    self.b_signals = []  # List to store blue channel values for rPPG
```

4.3.2 process_frames

Fungsi ini dijalankan dalam thread terpisah untuk memproses frame yang diambil.

```
def process_frames(self):
    """
    Processes the captured frames to extract respiration and rPPG signals.

    This method runs in the processing thread and processes the captured frames to extract
    respiration and rPPG signals. It also displays the processed frames with the signals.
    """
    while self.running:
        with self.frame_lock:
            if self.current_frame is None:  # Skip if no frame has been captured yet
                continue
            frame = self.current_frame.copy()  # Copy the current frame for processing

        # Process respiration signal
        try:
            left_x, top_y, right_x, bottom_y, center = get_respiration_roi(frame, self.
    pose_landmarker)
            roi = frame[top_y:bottom_y, left_x:right_x]  # Extract Region of Interest (ROI)
            avg_intensity = np.mean(roi[:, :, 1])  # Calculate average intensity from the green
    channel
            self.respiration_signal.append(avg_intensity)  # Store the respiration signal

            # Draw respiration ROI on the frame
            cv2.rectangle(frame, (left_x, top_y), (right_x, bottom_y), (255, 0, 0), 2)
            cv2.circle(frame, center, 5, (0, 255, 0), -1)
        except ValueError:
            pass  # If respiration ROI cannot be detected, pass

        # Process rPPG signal
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)  # Convert frame to RGB
        results = self.face_detection.process(frame_rgb)  # Detect faces in the frame

        if results.detections:  # If any faces are detected
```

```
            for detection in results.detections:
                bboxC = detection.location_data.relative_bounding_box
                h, w, _ = frame.shape
                x = int(bboxC.xmin * w)
                y = int(bboxC.ymin * h)
                width = int(bboxC.width * w)
                height = int(bboxC.height * h)

                # Extract RGB signals from the detected face region
                r, g, b = extract_rgb_signals(frame, (x, y, width, height))
                self.r_signals.append(r)  # Store red channel signal
                self.g_signals.append(g)  # Store green channel signal
                self.b_signals.append(b)  # Store blue channel signal

                # Draw bounding box around the detected face
                cv2.rectangle(frame, (x, y), (x + width, y + height), (0, 255, 0), 2)

        # Visualize the respiration signal if enough data has been collected
        if len(self.respiration_signal) > 30:
            self.visualize_respiration(frame)

        # Visualize the rPPG signal if enough data has been collected
        if len(self.g_signals) > 33:
            self.visualize_rppg(frame)

        # Display the frame with annotations
        cv2.imshow('Respiration and rPPG Tracking', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):  # Exit the loop if 'q' is pressed
            self.running = False
```

5 Analisis Hasil

Hasil yang diperoleh menunjukkan bahwa sinyal respirasi dan rPPG yang dihasilkan cukup stabil dan jelas. Visualisasi sinyal menunjukkan perkembangan yang mulus, tanpa adanya gangguan yang signifikan, meskipun terdapat beberapa fluktuasi kecil dalam sinyal rPPG yang mungkin disebabkan oleh gangguan lingkungan atau pengaturan kamera. Secara keseluruhan, pemrosesan video dilakukan dengan lancar tanpa lag atau penurunan kualitas. Visualisasi sinyal responsif dan memberikan gambaran yang jelas mengenai frekuensi dan amplitudo sinyal. Berikut beberapa plotting sinyal yang didapatkan dari dijalankannya kode.

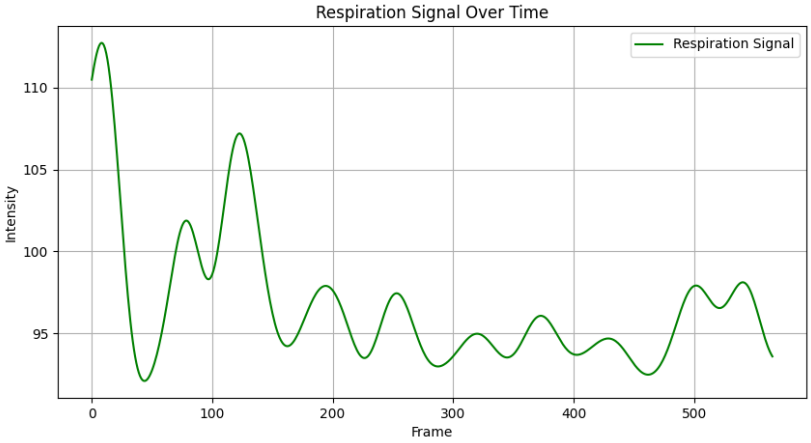

Figure 4: Sinyal Respirasi Relaksasi

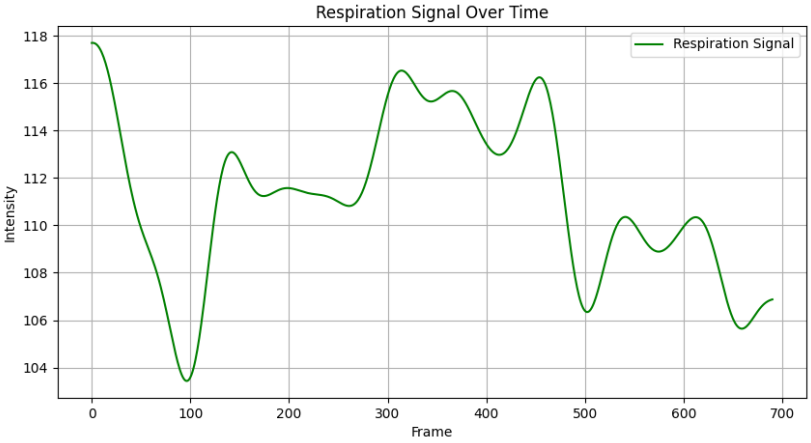

Figure 5: Sinyal Respirasi Setelah Aktivitas

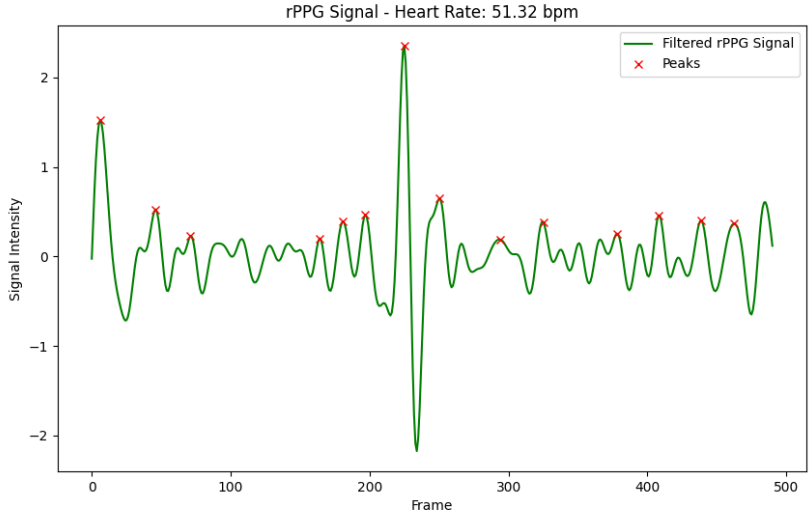

Figure 6: Sinyal rPPG Relaksasi

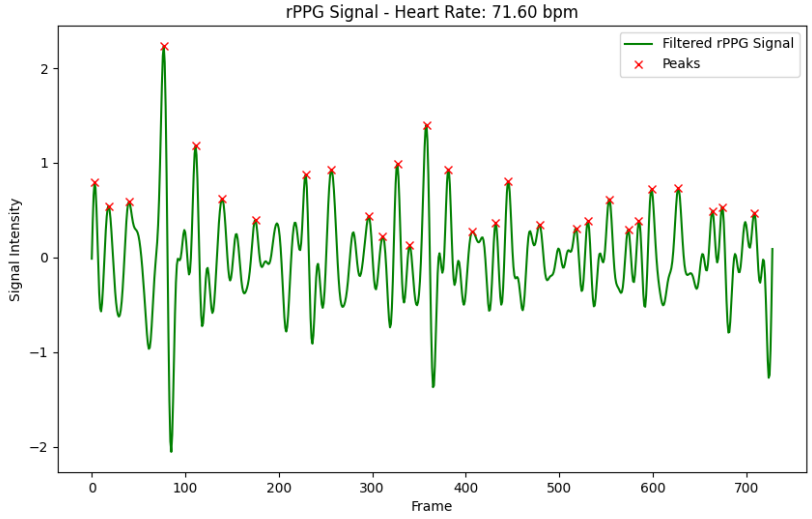

Figure 7: Sinyal rPPG Setelah Aktivitas

6 Kesimpulan

Secara keseluruhan, proyek ini berhasil mengembangkan sistem untuk mengukur sinyal respirasi dan rPPG dengan menggunakan video webcam secara real-time. Kualitas sinyal yang dihasilkan memadai dan dapat digunakan untuk tujuan analisis medis. Penggunaan filter yang tepat sangat penting dalam memastikan keakuratan dan stabilitas sinyal. Namun masih terdapat ketidaksempurnaan implementasi pada main.py yang berjalan sangat berat. Kedepannya, sistem ini dapat diperbaiki dengan meningkatkan akurasi pemrosesan sinyal dan menambah fitur lainnya.

7 Lampiran

Kode sumber untuk program ini disertakan berikut ini: https://github.com/attarakram121140013/IF3024-Final-Project/tree/main

References

[1] ChatGPT: Ketiga Kode, "https://chatgpt.com/share/676a6a67-71b4-800d-8a8b-58076a9ec6b5".

[2] ChatGPT: rPPG, "https://chatgpt.com/share/676a6a99-32f4-800d-a2b5-34c9cde10541".

[3] ChatGPT: Respirasi, "https://chatgpt.com/share/676a6a8b-7e94-800d-9a08-df06a3110919".

[4] ChatGPT: Main, "https://chatgpt.com/share/676a6a76-7ea0-800d-9c06-e65ad5217935".

[5] ChatGPT: Filtering, "https://chatgpt.com/share/676a6a54-c5d4-800d-a49f-6fcacac46534".