Program Studi Teknik Informatika
Institut Teknologi Sumatera

Nama:
1. **Daffa Abdurahman Jatmiko (121140181)**
2. **Andreas Gumarang Sihotang (121140168)**
3. **Ihsan Triyadi (121140163)**

Tugas : **Tugas Besar**

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

Tanggal:24/12/2024

1 Pendahuluan

1.1 Latar Belakang

Sebagai bagian dari tugas akhir mata kuliah Pengolahan Sinyal Digital IF(3024), dikembangkan sistem real-time untuk mendeteksi sinyal respirasi dan remote-photopletysmography (rPPG) melalui webcam. Sistem ini mengintegrasikan dua pendekatan: deteksi pernapasan melalui analisis gerakan bahu menggunakan pose-landmarker MediaPipe, serta pengukuran detak jantung non-invasif dengan algoritma Plane Orthogonal-to Skin (POS) yang menganalisis variasi warna pada region wajah yang terdeteksi melalui face-detector MediaPipe.

2 Alat dan Bahan

2.1 Lingkungan Pengembangan

Implementasi sistem pengukuran sinyal vital non-kontak ini memerlukan spesifikasi perangkat dan perangkat lunak yang memadai untuk pemrosesan video real-time dan analisis sinyal digital.

2.2 Bahasa Pemrograman

Sistem dikembangkan menggunakan Python karena keunggulannya dalam ekosistem computer vision dan pengolahan sinyal digital, serta dukungan ekstensif dari komunitas open source.

2.3 Komponen Perangkat Lunak

2.3.1 Library Pemrosesan Visual

- **OpenCV**: Framework utama untuk akuisisi dan pemrosesan video real-time
- **MediaPipe**: Menyediakan model AI untuk deteksi wajah dan landmark pose, khususnya untuk tracking area wajah dan pergerakan bahu

2.3.2 Library Analisis dan Komputasi

- **NumPy**: Pemrosesan array dan operasi matematika untuk analisis sinyal
- **SciPy**: Implementasi algoritma pemrosesan sinyal dan ekstraksi fitur

2.3.3 Library Pendukung Sistem

- **os & sys**: Manajemen fungsi dan variabel sistem
- **requests**: Komunikasi HTTP
- **tqdm**: Visualisasi progress komputasi

3 Metodologi

3.1 Ekstraksi Sinyal Vital

3.1.1 Sinyal rPPG

Implementasi algoritma Plane Orthogonal-to-Skin (POS) untuk deteksi detak jantung menggunakan analisis perubahan krominasi pada region wajah. Metode ini dipilih karena ketahanannya terhadap variasi pencahayaan dan gerakan subjek.

3.1.2 Sinyal Respirasi

Pengukuran pola respirasi menggunakan metode Zone-based Weighted Optical Flow yang membagi ROI menjadi tiga zona vertikal dengan pembobotan berbeda (kiri: 0.5, tengah: 1.0, kanan: 0.5). Metode ini dipilih karena ketahanannya terhadap noise gerakan lokal melalui sistem pembobotan zona, dimana zona tengah memiliki bobot lebih tinggi karena umumnya memiliki pergerakan respirasi yang lebih stabil, sementara zona kiri dan kanan dengan bobot lebih rendah tetap berkontribusi namun dengan pengaruh yang lebih kecil untuk mengkompensasi gerakan lokal.

Tahapan detail metode ini meliputi:

1. Penentuan Region of Interest (ROI):
   - Deteksi landmark bahu menggunakan MediaPipe Pose Detection
   - Perhitungan titik tengah ROI dari koordinat kedua bahu
   - ROI dibentuk dengan ukuran tetap dari titik tengah
2. Pre-processing dan Enhancement ROI:
   - Konversi ke grayscale
   - Penerapan CLAHE (Contrast Limited Adaptive Histogram Equalization)
   - Aplikasi Gaussian blur (3x3) untuk reduksi noise
   - Peningkatan edge menggunakan teknik edge enhancement
3. Zone-based Feature Detection:
   - ROI dibagi menjadi 3 zona vertikal dengan bobot:
     - Zona kiri (0.5): Kompensasi gerakan tubuh bagian kiri
     - Zona tengah (1.0): Area utama pergerakan respirasi
     - Zona kanan (0.5): Kompensasi gerakan tubuh bagian kanan
   - Deteksi 50 titik fitur menggunakan Shi-Tomasi corner detection
   - Refresh titik fitur ketika jumlah < 25 untuk menjaga stabilitas tracking
4. Multi-zone Tracking:

- Implementasi pyramidal Lucas-Kanade optical flow
- Tracking independen untuk setiap zona
- Perhitungan median posisi Y titik-titik dalam setiap zona
- Penerapan weighted average berdasarkan bobot zona

5. Temporal Processing:
   - Sliding window 8 sampel dengan weighted moving average
   - Pembobotan linear dari 0.5 hingga 1.0 untuk sampel terbaru
   - Normalisasi sinyal ke mean = 0

4 Penjelasan

Berikut adalah penjelasan mengenai alur kerja program yang dikembangkan pada proyek ini:

4.1 Remote Photoplethysmography (RPPG

Pertama mengimport library/pustaka yang digunakan, library/pustaka yang digunakan adalah sebagai berikut.

```
import numpy as np
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt
import scipy.signal as signal
import time
```

Kode 1: Import Library/Pustaka

4.1.1 Mengunduh Model face-detector dari MediaPipe

```
def download_model_face_detection():
    """
    Fungsi ini bertanggung jawab untuk mengunduh model face detector dari MediaPipe.
    Model ini digunakan untuk mendeteksi wajah dalam video.
    """
    # Membuat direktori 'model' jika belum ada
    model_dir = "model"
    os.makedirs(model_dir, exist_ok=True)

    # URL untuk mengunduh model face detector dari MediaPipe
    url = "https://storage.googleapis.com/mediapipe-models/face_detector/blaze_face_short_range/
    float16/latest/blaze_face_short_range.tflite"
    filename = os.path.join(model_dir, "face_detector.task")

    # Memeriksa apakah file model sudah ada dan tidak kosong
    if os.path.exists(filename) and os.path.getsize(filename) > 0:
        print(f"File model {filename} sudah terunduh. Melewati proses pengunduhan...")
        return filename

    # Proses pengunduhan dengan progress bar menggunakan tqdm
    try:
        print(f"Mengunduh model face detector ke direktori: {filename}...")
        response = requests.get(url, stream=True)
        response.raise_for_status()  # Memastikan request berhasil
        total_size = int(response.headers.get('content-length', 0))
        block_size = 1024
```

```
        # Membuka file dan menampilkan progress bar
        with open(filename, 'wb') as f, tqdm(
            total=total_size,
            unit='iB',
            unit_scale=True,
            unit_divisor=1024,
        ) as pbar:
            # Menulis data ke file secara bertahap
            for data in response.iter_content(block_size):
                size = f.write(data)
                pbar.update(size)

        # Verifikasi file hasil unduhan
        if os.path.getsize(filename) == 0:
            raise ValueError("File yang terunduh kosong")

        print("Proses pengunduhan model face detector berhasil!")
        return filename

    except Exception as e:
        print(f"Terdapat error dalam proses pengunduhan model face detector: {e}")
        if os.path.exists(filename):
            os.remove(filename)  # Membersihkan file yang tidak lengkap
        raise
```

Kode 2: Mengunduh Model face-detector dari MediaPipe

4.1.2 Mengunduh Model pose-landmarker dari MediaPipe

```
def download_model_pose_detection():
    """
    Fungsi ini mengunduh model pose landmarker dari MediaPipe.
    Model digunakan untuk mendeteksi pose tubuh pengguna dalam video (khususnya bagian bahu/torso atas).
    """
    # Membuat direktori model jika belum ada
    model_dir = "model"
    os.makedirs(model_dir, exist_ok=True)

    # URL untuk mengunduh model pose landmarker
    url = "https://storage.googleapis.com/mediapipe-models/pose_landmarker/pose_landmarker_heavy/float16/latest/pose_landmarker_heavy.task"
    filename = os.path.join(model_dir, "pose_landmarker.task")

    # Memeriksa apakah file sudah ada dan tidak kosong
    if os.path.exists(filename) and os.path.getsize(filename) > 0:
        print(f"File model {filename} sudah terunduh. Melewati proses pengunduhan...")
        return filename

    try:
        print(f"Mengunduh model pose landmarker ke direktori: {filename}...")
        response = requests.get(url, stream=True)
        response.raise_for_status()

        total_size = int(response.headers.get('content-length', 0))
        block_size = 1024

        # Proses pengunduhan dengan progress bar
        with open(filename, 'wb') as f, tqdm(
            total=total_size,
```

```
            unit='iB',
            unit_scale=True,
            unit_divisor=1024,
        ) as pbar:
            for data in response.iter_content(block_size):
                size = f.write(data)
                pbar.update(size)

        # Verifikasi hasil unduhan
        if os.path.getsize(filename) == 0:
            raise ValueError("File yang terunduh kosong")

        print("Proses pengunduhan model pose landmarker berhasil!")
        return filename

    except Exception as e:
        print(f"Terdapat error dalam proses pengunduhan model face detector: {e}")
        if os.path.exists(filename):
            os.remove(filename)
        raise
```

Kode 3: Mengunduh Model pose-landmarker dari MediaPipe

4.1.3 Memeriksa ketersediaan GPU

```
def check_gpu():
    """
    Fungsi ini memeriksa ketersediaan GPU pada sistem.

    Returns:
        str: "NVIDIA" untuk GPU NVIDIA, "MLX" untuk Apple Silicon, atau "CPU" jika tidak ada GPU
    """
    system = platform.system()
    print(f"System: {system}")

    # Memeriksa apakah sistem adalah Windows atau Linux
    if system == "Linux" or system == "Windows":
        try:
            # Mencoba menjalankan perintah nvidia-smi untuk deteksi GPU NVIDIA
            nvidia_output = subprocess.check_output(['nvidia-smi']).decode('utf-8')
            return "NVIDIA"
        except (subprocess.CalledProcessError, FileNotFoundError):
            return "CPU"

    # Memeriksa apakah sistem adalah macOS dengan arsitektur Apple Silicon
    elif system == "Darwin":  # macOS
        try:
            # Mendapatkan informasi CPU
            cpu_info = subprocess.check_output(['sysctl', '-n', 'machdep.cpu.brand_string']).decode
    ('utf-8').strip()
            print(f"CPU: {cpu_info}")
            if "Apple" in cpu_info:  # Mencakup semua chip buatan Apple (M1/M2/M3)
                return "MLX"
        except subprocess.CalledProcessError:
            pass
        return "CPU"
```

Kode 4: Memeriksa ketersediaan GPU

4.1.4 Mengambil Region of Interest (ROI) dari Webcam

```python
def get_initial_roi(image, landmarker, x_size=100, y_size=30, shift_x=0, shift_y=-30):
    """
    Fungsi untuk mengambil ROI dari webcam berdasarkan pergerakan posisi bahu pasien.

    Args:
        image (np.ndarray): Frame dari webcam
        landmarker (task): MediaPipe pose detector yang sudah didownload
        x_size (int): Ukuran ROI pada sumbu x
        y_size (int): Ukuran ROI pada sumbu y
        shift_x (int): Pergeseran ROI pada sumbu x
        shift_y (int): Pergeseran ROI pada sumbu y

    Returns:
        tuple: Koordinat ROI (left_x, top_y, right_x, bottom_y)
    """
    # Mengubah warna BGR ke RGB untuk MediaPipe
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width = image.shape[:2]  # Mengambil dimensi frame webcam

    # Membuat gambar MediaPipe dari frame webcam
    mp_image = mp.Image(
        image_format=mp.ImageFormat.SRGB,
        data=image_rgb
    )

    # Mendeteksi pose dari frame webcam
    detection_result = landmarker.detect(mp_image)

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

    # Mendeteksi tubuh pengguna dari landmark pertama
    landmarks = detection_result.pose_landmarks[0]

    # Mengambil landmark bahu kiri dan kanan
    left_shoulder = landmarks[11]
    right_shoulder = landmarks[12]

    # Menghitung posisi tengah dari bahu kiri dan kanan
    center_x = int((left_shoulder.x + right_shoulder.x) * width/2)
    center_y = int((left_shoulder.y + right_shoulder.y) * height/2)

    # Mengaplikasikan shift terhadap titik tengah
    center_x += shift_x
    center_y += shift_y

    # Menghitung batasan ROI berdasarkan posisi tengah dan ukuran ROI
    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)

    # Memvalidasi ukuran ROI
    if (right_x - left_x) <= 0 or (bottom_y - top_y) <= 0:
        raise ValueError("Invalid ROI dimensions")

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

Kode 5: Mengambil Region of Interest (ROI) dari Webcam

4.2 Respiration

lakukan hal yang sama dengan respirasi mengimport Library/pustaka

```
import time
import cv2
import numpy as np
import mediapipe as mp
```

Kode 6: Import Library/Pustaka

4.2.1 Fungsi get$_{i}nitial_{r}oi$

```
def get_initial_roi(image, model_path, x_size, y_size, shift_x, shift_y):
    """
    Fungsi ini bertanggung jawab untuk mendeteksi Region of Interest (ROI) awal untuk
    pelacakan respirasi menggunakan MediaPipe.

    Cara kerja:
    1. Mengkonversi gambar dari BGR ke RGB untuk MediaPipe
    2. Menginisialisasi detektor pose dengan model yang diberikan
    3. Mendeteksi landmark pose, khususnya posisi bahu
    4. Menghitung ROI berdasarkan posisi bahu
    """
    # Konversi warna dan ambil dimensi
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    height, width = image.shape[:2]

    # Konfigurasi dan inisialisasi pose detector
    options = mp.tasks.vision.PoseLandmarkerOptions(
        base_options=mp.tasks.BaseOptions(model_asset_path=model_path),
        running_mode=mp.tasks.vision.RunningMode.IMAGE
    )

    # Deteksi pose dan hitung ROI
    with mp.tasks.vision.PoseLandmarker.create_from_options(options) as pose_detector:
        mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=image_rgb)
        detection_result = pose_detector.detect(mp_image)

        # Validasi hasil deteksi
        if not detection_result.pose_landmarks or len(detection_result.pose_landmarks) == 0:
            raise ValueError("No pose detected!")

        # Ekstrak landmark bahu dan hitung posisi tengah
        landmarks = detection_result.pose_landmarks[0]
        left_shoulder = landmarks[11]
        right_shoulder = landmarks[12]

        # Hitung koordinat ROI
        center_x = int((left_shoulder.x + right_shoulder.x) * width / 2) + shift_x
        center_y = int((left_shoulder.y + right_shoulder.y) * height / 2) + shift_y

        # Tetapkan batas ROI dengan mempertimbangkan batas frame
        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
```

Kode 7: Fungsi get$_{i}nitial_{r}oi$

4.2.2 Kelas FlowTracker

```
class FlowTracker:
    """
    Kelas ini menangani pelacakan aliran (flow) berdasarkan titik-titik yang terdeteksi dalam ROI.

    Fitur utama:
    - Pelacakan posisi historis
    - Smoothing menggunakan weighted average
    - Pembagian ROI menjadi zona dengan bobot berbeda
    """
    def __init__(self, history_size=8):
        # Inisialisasi parameter pelacakan
        self.history_size = history_size
        self.last_positions = None
        self.last_y_pos = None
        self.smoothing_window = []

    def get_flow_position(self, points, roi_coords):
        """
        Menghitung posisi aliran berdasarkan titik-titik yang dilacak dengan:
        1. Pembagian ROI menjadi 3 zona (kiri, tengah, kanan)
        2. Pembobotan berbeda untuk setiap zona
        3. Smoothing menggunakan weighted average
        """
        if len(points) == 0:
            return self.last_y_pos

        left_x, top_y, right_x, bottom_y = roi_coords
        width = right_x - left_x

        # Kembalikan bobot yang lebih seimbang
        zone_weights = {'left': 0.5, 'center': 1.0, 'right': 0.5}

        zones = {
            'left': (left_x, left_x + width/3),
            'center': (left_x + width/3, left_x + 2*width/3),
            'right': (left_x + 2*width/3, right_x)
        }

        zone_positions = {'left': [], 'center': [], 'right': []}

        for point in points:
            x, y = point.ravel()
            for zone_name, (zone_left, zone_right) in zones.items():
                if zone_left <= x <= zone_right:
                    zone_positions[zone_name].append(y)
                    break

        weighted_sum = 0
        total_weight = 0

        for zone_name, positions in zone_positions.items():
            if positions:
                zone_y = np.median(positions)
                weighted_sum += zone_y * zone_weights[zone_name]
                total_weight += zone_weights[zone_name]

        if total_weight > 0:
            y_pos = weighted_sum / total_weight

            self.smoothing_window.append(y_pos)
```

```
            if len(self.smoothing_window) > self.history_size:
                self.smoothing_window.pop(0)

            # Gunakan weighted average yang lebih ringan
            weights = np.linspace(0.5, 1.0, len(self.smoothing_window))
            weights /= weights.sum()
            smoothed_y = np.average(self.smoothing_window, weights=weights)

            self.last_y_pos = smoothed_y
            return smoothed_y

        return self.last_y_pos
```

Kode 8: Kelas FlowTracker

4.2.3 Fungsi process$_{f}rame$

```
def process_frame(frame, tracking_data):
    """
    Fungsi ini memproses setiap frame video untuk pelacakan respirasi.

    Tahapan proses:
    1. Konversi frame ke grayscale
    2. Deteksi fitur jika diperlukan
    3. Pelacakan optical flow
    4. Pembaruan posisi dan visualisasi
    """
    # Konversi ke grayscale untuk pemrosesan
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    y_pos = None

    # Inisialisasi tracker jika belum ada
    if 'flow_tracker' not in tracking_data:
        tracking_data['flow_tracker'] = FlowTracker()

    # Deteksi fitur baru jika jumlahnya kurang
    if len(tracking_data.get('features', [])) < 25:
        # Ekstrak dan tingkatkan ROI
        roi = frame_gray[tracking_data['roi_coords'][1]:tracking_data['roi_coords'][3],
                         tracking_data['roi_coords'][0]:tracking_data['roi_coords'][2]]

        # Pre-processing ROI
        roi_enhanced = cv2.equalizeHist(roi)
        roi_enhanced = cv2.GaussianBlur(roi_enhanced, (5,5), 0)

        # Deteksi fitur baru
        new_features = cv2.goodFeaturesToTrack(
            roi_enhanced,
            maxCorners=50,
            qualityLevel=0.15,
            minDistance=7,
            blockSize=7
        )

        # Update posisi fitur jika ditemukan
        if new_features is not None:
            new_features = new_features + np.array(
                [[tracking_data['roi_coords'][0], tracking_data['roi_coords'][1]]],
                dtype=np.float32
            )
            tracking_data['features'] = new_features
```

```
45
46      # Pelacakan optical flow
47      if len(tracking_data.get('features', [])) > 0:
48          # Hitung optical flow
49          new_features, status, error = cv2.calcOpticalFlowPyrLK(
50              tracking_data['old_gray'],
51              frame_gray,
52              tracking_data['features'],
53              None,
54              winSize=(21,21),
55              maxLevel=2,
56              criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)
57          )
58
59          # Update dan visualisasi jika ada fitur yang valid
60          if new_features is not None:
61              good_new = new_features[status == 1]
62              y_pos = tracking_data['flow_tracker'].get_flow_position(
63                  good_new, tracking_data['roi_coords']
64              )
65              frame = draw_flow_zones(frame, tracking_data['roi_coords'], good_new)
66              tracking_data['features'] = good_new.reshape(-1, 1, 2)
67
68      # Simpan frame grayscale untuk iterasi berikutnya
69      tracking_data['old_gray'] = frame_gray.copy()
70      return frame, y_pos
```

Kode 9: Fungsi process$_f ramer$

4.2.4 Fungsi draw$_f low_z ones$

```
1  def draw_flow_zones(frame, roi_coords, points):
2      """
3      Fungsi untuk visualisasi zona dan indikator aliran pada frame.
4
5      Fitur visualisasi:
6      1. Kotak ROI utama
7      2. Pembagian zona vertikal
8      3. Titik-titik pelacakan dengan warna berbeda per zona
9      """
10     # Mengekstrak koordinat ROI (Region of Interest) dari parameter roi_coords
11 left_x, top_y, right_x, bottom_y = roi_coords
12 # Menghitung lebar ROI dengan mengurangi koordinat x kanan dengan x kiri
13 width = right_x - left_x
14
15 # Menggambar kotak ROI utama pada frame
16 # Parameter: frame, titik awal (left_x, top_y), titik akhir (right_x, bottom_y)
17 # Warna hijau (0, 255, 0), ketebalan garis 2 pixel
18 cv2.rectangle(frame, (left_x, top_y), (right_x, bottom_y), (0, 255, 0), 2)
19
20 # Menggambar garis vertikal untuk membagi ROI menjadi 3 zona
21 for i in range(1, 3):  # Loop 2 kali untuk membuat 2 garis pembagi
22     # Menghitung posisi x untuk setiap garis pembagi (1/3 dan 2/3 dari lebar)
23     x = left_x + (width * i // 3)
24     # Menggambar garis vertikal dari atas ke bawah ROI
25     # Warna kuning (0, 255, 255), ketebalan garis 1 pixel
26     cv2.line(frame, (x, top_y), (x, bottom_y), (0, 255, 255), 1)
27
28 # Menggambar titik-titik yang terdeteksi dengan warna berbeda berdasarkan zona
29 if points is not None:  # Jika ada titik yang terdeteksi
30     for point in points:
```

```
        # Mengubah format titik menjadi koordinat x, y
        x, y = point.ravel()
        # Mengkonversi koordinat ke integer
        x, y = int(x), int(y)

        # Menentukan zona berdasarkan posisi relatif terhadap ROI
        rel_x = x - left_x  # Posisi relatif dari titik kiri ROI
        if rel_x < width/3:  # Zona kiri
            color = (0, 165, 255)  # Warna oranye untuk zona kiri
        elif rel_x < 2*width/3:  # Zona tengah
            color = (0, 0, 255)   # Warna merah untuk zona tengah
        else:  # Zona kanan
            color = (0, 165, 255)  # Warna oranye untuk zona kanan

        # Menggambar titik dengan radius 3 pixel dan warna sesuai zona
        # Parameter -1 berarti titik diisi penuh (filled)
        cv2.circle(frame, (x, y), 3, color, -1)

# Mengembalikan frame yang sudah digambar
return frame
```

Kode 10: Fungsi draw$_flow_zones$

4.2.5 Fungsi enhance$_roi$

```
def enhance_roi(roi):
    """
    Fungsi untuk meningkatkan kualitas ROI.

    Tahapan peningkatan:
    1. Konversi ke grayscale
    2. Aplikasi CLAHE untuk peningkatan kontras
    3. Edge enhancement dengan Gaussian blur
    """
    if roi is None or roi.size == 0:
        return None

    # Konversi dan peningkatan kualitas ROI
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    enhanced = clahe.apply(gray)
    enhanced = cv2.GaussianBlur(enhanced, (3,3), 0)
    return enhanced
```

Kode 11: Fungsi enhance$_roi$

4.2.6 Main Webcam Processing Function

```
def process_webcam(model_path, max_seconds, x_size, y_size, shift_x, shift_y):
    """
    Fungsi utama untuk memproses webcam dan melacak pergerakan bahu untuk analisis respirasi.

    Args:
        model_path: Path ke model pose detection
        max_seconds: Durasi maksimum perekaman dalam detik
        x_size: Ukuran window tracking pada sumbu x
        y_size: Ukuran window tracking pada sumbu y
        shift_x: Pergeseran tracking window pada sumbu x
        shift_y: Pergeseran tracking window pada sumbu y
    """
```

```
    # Inisialisasi webcam dan parameter video
    cap = cv2.VideoCapture(0)  # Membuka webcam default
    fps = cap.get(cv2.CAP_PROP_FPS)  # Mendapatkan FPS webcam
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))  # Mendapatkan lebar frame
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))  # Mendapatkan tinggi frame
    max_frames = int(fps * max_seconds)  # Menghitung jumlah frame maksimum

    # Setup pose detector
    pose_landmarker = tracking.setup_pose_landmarker(model_path)

    # Inisialisasi array untuk menyimpan data
    timestamps = []  # Array untuk menyimpan waktu
    y_positions = []  # Array untuk menyimpan posisi y

    try:
        # Memproses frame pertama
        ret, first_frame = cap.read()
        if not ret:
            raise ValueError("Could not read first frame!")

        # Inisialisasi tracking pada frame pertama
        tracking_data = tracking.initialize_tracking(
            first_frame,
            pose_landmarker,
            x_size, y_size,
            shift_x, shift_y
        )

        # Inisialisasi counter dan timer
        frame_count = 0
        start_time = time.time()

        # Loop utama pemrosesan video
        while True:
            # Membaca frame dari webcam
            ret, frame = cap.read()
            if not ret or frame_count >= max_frames:
                break

            # Memproses frame untuk analisis respirasi
            frame, y_pos = respiration.process_frame(frame, tracking_data)

            # Menyimpan data jika posisi terdeteksi
            if y_pos is not None:
                current_time = time.time() - start_time
                timestamps.append(current_time)
                y_positions.append(y_pos)

                # Menambahkan plot real-time ke frame
                frame = plotter.overlay_plot_on_frame(frame, timestamps, y_positions)

            # Menampilkan frame yang telah diproses
            cv2.imshow("Shoulder Tracking", frame)
            # Mengecek input 'q' untuk keluar
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

            frame_count += 1
```

Kode 12: Main Webcam Processing Function

4.2.7 Cleanup dan Visualisasi Akhir

```
finally:
        # Membersihkan resources
        cap.release()
        cv2.destroyAllWindows()

    # Membuat plot hasil akhir
    plotter.plot_shoulder_movement(timestamps, y_positions)

    # Mengembalikan data yang telah dikumpulkan
    return timestamps, y_positions
```

Kode 13: Cleanup dan Visualisasi Akhir

4.2.8 Pose Landmarker Setup

```
def setup_pose_landmarker(model_path):
    """
    Setup MediaPipe pose landmarker untuk deteksi pose.

    Args:
        model_path: Path ke file model pose detection

    Returns:
        PoseLandmarker: Instance MediaPipe pose landmarker yang telah dikonfigurasi
    """
    # Import kelas-kelas yang diperlukan dari MediaPipe
    PoseLandmarker = mp.tasks.vision.PoseLandmarker
    BaseOptions = mp.tasks.BaseOptions
    PoseLandmarkerOptions = mp.tasks.vision.PoseLandmarkerOptions
    VisionRunningMode = mp.tasks.vision.RunningMode

    # Konfigurasi opsi untuk pose landmarker
    options = PoseLandmarkerOptions(
        base_options=BaseOptions(model_asset_path=model_path),
        running_mode=VisionRunningMode.IMAGE,  # Mode pemrosesan per frame
        num_poses=1,  # Hanya deteksi 1 pose
        min_pose_detection_confidence=0.5,  # Threshold confidence deteksi
        min_pose_presence_confidence=0.5,  # Threshold confidence keberadaan pose
        min_tracking_confidence=0.5  # Threshold confidence tracking
    )

    return PoseLandmarker.create_from_options(options)
```

Kode 14: Pose Landmarker Setupr

4.2.9 ROI Detection

```
def get_initial_roi(frame, pose_landmarker, x_size, y_size, shift_x, shift_y):
    """
    Mendapatkan Region of Interest (ROI) berdasarkan posisi bahu.

    Args:
        frame: Frame video dari webcam
        pose_landmarker: Instance pose landmarker yang sudah disetup
        x_size: Lebar ROI
        y_size: Tinggi ROI
        shift_x: Pergeseran horizontal ROI
        shift_y: Pergeseran vertikal ROI
```

```
    Returns:
        tuple: Koordinat ROI (left_x, top_y, right_x, bottom_y)
    """
    # Konversi frame ke RGB untuk MediaPipe
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    height, width = frame.shape[:2]

    # Membuat image MediaPipe
    mp_image = mp.Image(
        image_format=mp.ImageFormat.SRGB,
        data=frame_rgb
    )

    # Deteksi landmarks pose
    detection_result = pose_landmarker.detect(mp_image)

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

    landmarks = detection_result.pose_landmarks[0]

    # Mendapatkan posisi bahu (landmarks 11 dan 12)
    left_shoulder = landmarks[11]
    right_shoulder = landmarks[12]

    # Menghitung titik tengah antara bahu
    center_x = int((left_shoulder.x + right_shoulder.x) * width / 2)
    center_y = int((left_shoulder.y + right_shoulder.y) * height / 2)

    # Mengaplikasikan pergeseran
    center_x += shift_x
    center_y += shift_y

    # Menghitung batas ROI
    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)
```

Kode 15: ROI Detectionr

4.2.10 Feature Detection

```
def initialize_features(roi, left_x, top_y):
    """
    Mendeteksi fitur-fitur yang akan ditracking dalam ROI.

    Args:
        roi: Region of Interest dari frame
        left_x: Koordinat x kiri ROI
        top_y: Koordinat y atas ROI

    Returns:
        np.ndarray: Array titik-titik fitur yang terdeteksi
    """
    features = cv2.goodFeaturesToTrack(
        roi,
        maxCorners=60,  # Maksimal 60 titik fitur
```

```
        qualityLevel=0.15,  # Threshold kualitas fitur
        minDistance=3,  # Jarak minimal antar fitur
        blockSize=7  # Ukuran block untuk deteksi
    )

    if features is not None:
        features = np.float32(features)
        # Menyesuaikan koordinat ke frame penuh
        features[:,:,0] += left_x
        features[:,:,1] += top_y

    return features
```

Kode 16: Feature Detection

4.2.11 Optical Flow Parameters

```
def get_lk_params():
    """
    Mendapatkan parameter untuk Lucas-Kanade optical flow.

    Returns:
        dict: Parameter untuk optical flow
    """
    return dict(
        winSize=(15, 15),  # Ukuran window pencarian
        maxLevel=2,  # Level pyramid maksimum
        criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03)  # Kriteria terminasi
    )
```

Kode 17: Optical Flow Parameters

4.2.12 Tracking Initialization

```
def initialize_tracking(frame, pose_landmarker, x_size, y_size, shift_x, shift_y):
    """
    Menginisialisasi sistem tracking dengan menggabungkan semua komponen.

    Args:
        frame: Frame video dari webcam
        pose_landmarker: Instance pose landmarker
        x_size: Lebar ROI
        y_size: Tinggi ROI
        shift_x: Pergeseran horizontal ROI
        shift_y: Pergeseran vertikal ROI

    Returns:
        dict: Data tracking yang dibutuhkan untuk pemrosesan selanjutnya
    """
    # Mendapatkan ROI awal
    roi_coords = get_initial_roi(frame, pose_landmarker,
                                 x_size, y_size, shift_x, shift_y)
    left_x, top_y, right_x, bottom_y = roi_coords

    # Inisialisasi frame untuk optical flow
    old_frame = frame.copy()
    old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)

    # Mendapatkan ROI awal dan deteksi fitur
    roi = old_gray[top_y:bottom_y, left_x:right_x]
```

```
    features = initialize_features(roi, left_x, top_y)

    # Mengembalikan semua data tracking dalam dictionary
    return {
        'roi_coords': roi_coords,
        'old_gray': old_gray,
        'features': features,
        'lk_params': get_lk_params()
    }
```

Kode 18: Tracking Initialization

4.2.13 Real-time Plot Overlay

```
def overlay_plot_on_frame(frame, timestamps, y_positions):
    """
    Menambahkan plot real-time ke frame video.

    Args:
        frame: Frame video dari webcam
        timestamps: Array waktu pengukuran
        y_positions: Array posisi y yang terukur

    Returns:
        frame: Frame dengan plot yang sudah ditambahkan
    """
    # Skip jika belum ada data
    if not timestamps or not y_positions:
        return frame

    # Menggunakan backend Agg untuk plotting tanpa GUI
    plt.switch_backend('Agg')

    # Membuat figure dan axes untuk plot
    fig, ax = plt.subplots(figsize=(4, 3))
    ax.plot(timestamps, y_positions, "g-", linewidth=2)
    ax.set_xlabel("Time (seconds)")
    ax.set_ylabel("Y Position (pixels)")
    ax.set_title("Shoulder Movement")

    # Mengkonversi plot menjadi gambar
    fig.canvas.draw()
    plot_img = np.frombuffer(fig.canvas.tostring_rgb(), dtype=np.uint8)
    plot_img = plot_img.reshape(fig.canvas.get_width_height()[::-1] + (3,))

    # Membersihkan resources matplotlib
    plt.close(fig)

    # Mengubah ukuran plot agar muat di pojok frame
    h, w = frame.shape[:2]
    plot_h, plot_w = plot_img.shape[:2]
    scale = min(h/3/plot_h, w/3/plot_w)  # Plot ukuran 1/3 dari frame
    new_h, new_w = int(plot_h*scale), int(plot_w*scale)
    plot_img = cv2.resize(plot_img, (new_w, new_h))

    # Menempatkan plot di pojok kanan atas
    frame[10:10+new_h, w-new_w-10:w-10] = plot_img

    return frame
```

Kode 19: Real-time Plot Overlay

4.2.14 Final Analysis Plot

```
def plot_shoulder_movement(timestamps, y_positions):
    """
    Membuat plot analisis akhir dari gerakan bahu dengan filtering dan estimasi
    respiratory rate.

    Args:
        timestamps: Array waktu pengukuran
        y_positions: Array posisi y yang terukur

    Returns:
        tuple: (timestamps, y_positions) data yang digunakan
    """
    if len(timestamps) > 0 and len(y_positions) > 0:
        print(f"Plotting data with {len(timestamps)} points...")

        # Kembali ke backend default untuk display
        plt.switch_backend('TkAgg')

        # Setup plot
        plt.figure(figsize=(12, 6))

        # Normalisasi waktu ke detik
        t = np.array(timestamps) - timestamps[0]

        # Pre-processing sinyal
        y_preprocessed = filters.preprocess_signal(y_positions)

        # Menyesuaikan timestamps untuk sinyal yang telah diproses
        t_preprocessed = t[:len(y_preprocessed)]

        # Normalisasi sinyal ke mean=0
        y_normalized = y_preprocessed - np.mean(y_preprocessed)

        # Aplikasi bandpass filter
        y_filtered = filters.apply_bandpass_filter(y_normalized)

        # Menghitung respiratory rate
        breaths_per_minute, peaks = filters.estimate_respiratory_rate(
            y_filtered,
            t_preprocessed
        )

        # Membuat plot dengan multiple signals
        plt.plot(t_preprocessed, y_normalized,
                label='Preprocessed Signal',
                color='blue',
                alpha=0.3)
        plt.plot(t_preprocessed, y_filtered,
                label='Filtered Signal',
                color='red')
        plt.plot(t_preprocessed[peaks], y_filtered[peaks],
                "x",
                label='Peaks')

        # Menambahkan labels dan informasi
        plt.xlabel('Time (seconds)')
        plt.ylabel('Y Position (pixels)')
        plt.title(f'Shoulder Movement Over Time\n' +
                  f'Estimated Respiratory Rate: {breaths_per_minute:.1f} breaths/min')
        plt.legend()
```

```
        plt.grid(True)
        plt.show()

        print(f"Estimated Respiratory Rate: {breaths_per_minute:.1f} breaths/min")

    return timestamps, y_positions
```

Kode 20: Final Analysis Plot

4.2.15 Bandpass Filter Design

```
def butter_bandpass(lowcut, highcut, fs, order=2):
    """
    Membuat Butterworth bandpass filter dengan karakteristik yang lebih lembut.

    Args:
        lowcut: Frekuensi cut-off bawah
        highcut: Frekuensi cut-off atas
        fs: Frequency sampling
        order: Order filter (default=2 untuk filter yang lebih lembut)

    Returns:
        tuple: Koefisien filter (b, a)
    """
    nyquist = 0.5 * fs  # Frekuensi Nyquist
    low = lowcut / nyquist  # Normalisasi frekuensi
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return b, a
```

Kode 21: Bandpass Filter Design

4.2.16 Bandpass Filter Application

```
def apply_bandpass_filter(signal, fs=30):
    """
    Mengaplikasikan bandpass filter untuk sinyal respirasi.

    Args:
        signal: Sinyal input
        fs: Frequency sampling (default=30 Hz)

    Returns:
        np.ndarray: Sinyal yang telah difilter
    """
    # Range frekuensi untuk respirasi normal
    lowcut = 0.1    # 6 nafas per menit
    highcut = 0.5   # 30 nafas per menit

    # Membuat filter dengan order=2
    b, a = butter_bandpass(lowcut, highcut, fs, order=2)

    # Aplikasi filter dengan filtfilt (zero-phase filtering)
    filtered_signal = filtfilt(b, a, signal)

    return filtered_signal
```

Kode 22: Bandpass Filter Application

4.2.17 Signal Preprocessing

```python
def preprocess_signal(y_positions, window=15):
    """
    Preprocessing sinyal dengan metode yang lebih halus.

    Args:
        y_positions: Array posisi y
        window: Ukuran window untuk smoothing (default=15)

    Returns:
        np.ndarray: Sinyal yang telah dipreprocess
    """
    y_array = np.array(y_positions)

    # Membersihkan outlier dengan percentile
    percentile_5 = np.percentile(y_array, 5)
    percentile_95 = np.percentile(y_array, 95)
    mask = (y_array >= percentile_5) & (y_array <= percentile_95)
    y_cleaned = y_array[mask]

    # Normalisasi sinyal
    y_normalized = y_cleaned - np.mean(y_cleaned)

    # Smoothing dengan Savitzky-Golay filter
    y_smoothed = savgol_filter(
        y_normalized,
        window_length=15,  # Harus ganjil
        polyorder=3        # Order polynomial
    )

    return y_smoothed
```

Kode 23: Signal Preprocessing

4.2.18 Respiratory Rate Estimation

```python
def estimate_respiratory_rate(filtered_signal, timestamps):
    """
    Mengestimasi respiratory rate dari sinyal yang telah difilter.

    Args:
        filtered_signal: Sinyal yang telah difilter
        timestamps: Array waktu pengukuran

    Returns:
        tuple: (respiratory_rate, peak_indices)
    """
    if len(filtered_signal) < 2:
        return 0, []

    # Deteksi puncak sinyal
    peaks, _ = find_peaks(
        filtered_signal,
        distance=15,      # Minimal jarak antar puncak
        prominence=0.05,  # Threshold prominence
        height=None       # Tidak ada batasan tinggi
    )

    if len(peaks) < 2:
        return 0, peaks
```

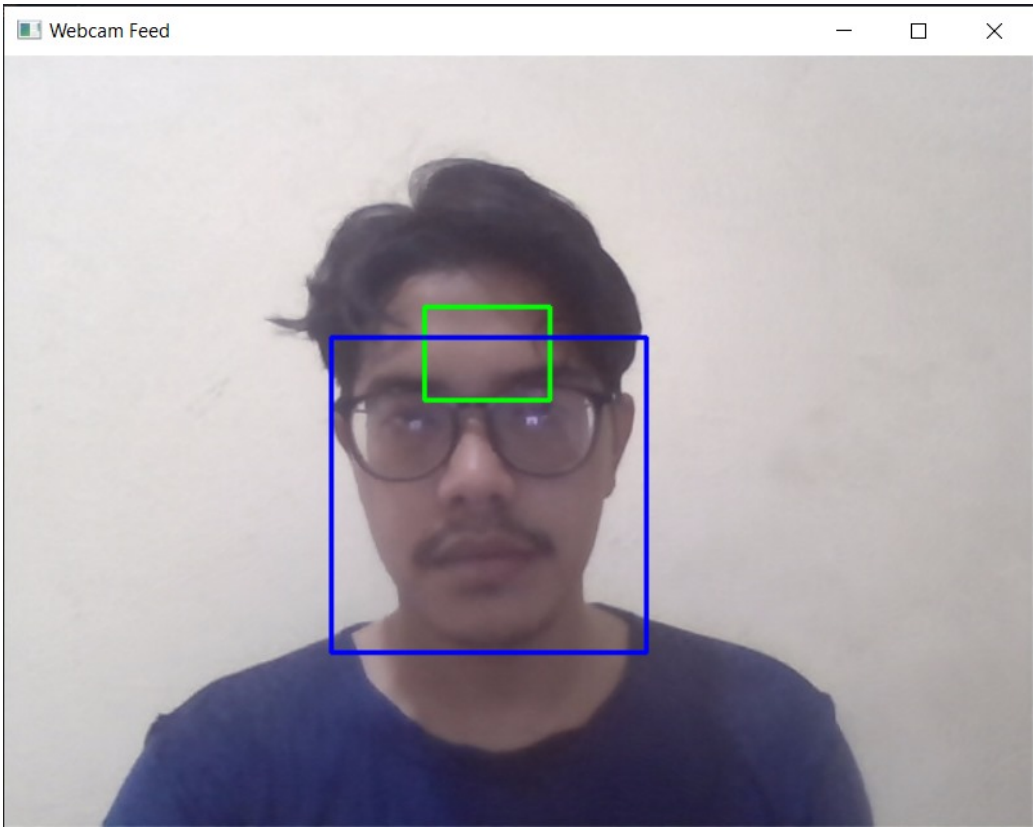
```
25
26      # Menghitung rate pernapasan
27      peak_times = timestamps[peaks]
28      intervals = np.diff(peak_times)
29
30      # Filter interval yang valid (2-8 detik per nafas)
31      valid_intervals = intervals[(intervals >= 2) & (intervals <= 8)]
32
33      if len(valid_intervals) > 0:
34          mean_interval = np.mean(valid_intervals)
35          breaths_per_minute = 60.0 / mean_interval
36      else:
37          breaths_per_minute = 0
38
39      return breaths_per_minute, peaks
```

Kode 24: Respiratory Rate Estimation

5 Hasil Analisis Sinyal

5.1 Analisis Sinyal Remote Photoplethysmography (rPPG)

Berikut menampilkan hasil deteksi wajah dan area fitur tertentu (seperti dahi) dengan implementasi Remote Photoplethysmography (rPPG).



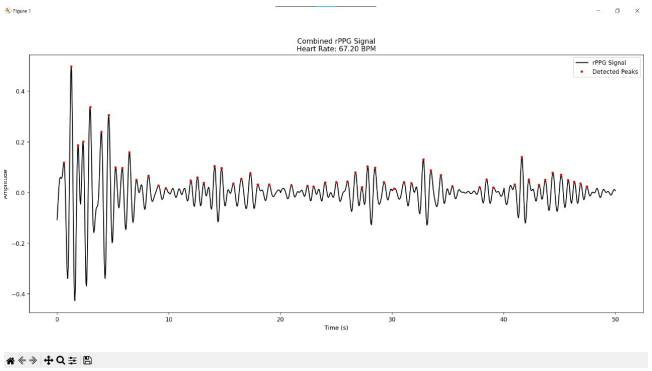
Gambar 1: Gambar Fitur RPPG

Penjelasan singkat: **Kotak Hijau:** Menunjukkan area ROI (Region of Interest), seperti dahi, yang sering digunakan untuk mengestimasi sinyal photoplethysmography dari perubahan warna kulit akibat aliran darah. **Kotak Biru:** Menunjukkan bounding box yang mendeteksi wajah secara keseluruhan. Deteksi wajah ini dilakukan untuk membatasi area analisis ke wajah subjek. **Aplikasi rPPG:** Data visual dari ROI dianalisis untuk mengekstrak informasi kardiovaskular, seperti detak jantung, tanpa kontak langsung dengan sensor fisik.

5.1.1 Hasil Analisis Sinyal rPPG

Analisis Komponen Sinyal rPPG:

1. **Sinyal rPPG:**
   - Grafik menunjukkan sinyal rPPG yang telah diproses. Sinyal ini mencerminkan fluktuasi amplitudo terkait perubahan volume darah di pembuluh darah wajah, yang dapat dihitung untuk menentukan denyut jantung.
   - *Sumbu X*: Menunjukkan waktu (dalam detik).
   - *Sumbu Y*: Menunjukkan amplitudo sinyal rPPG.
2. **Puncak Terdeteksi:**
   - Titik merah di grafik menandai puncak-puncak sinyal yang terdeteksi menggunakan algoritma pengenalan puncak.



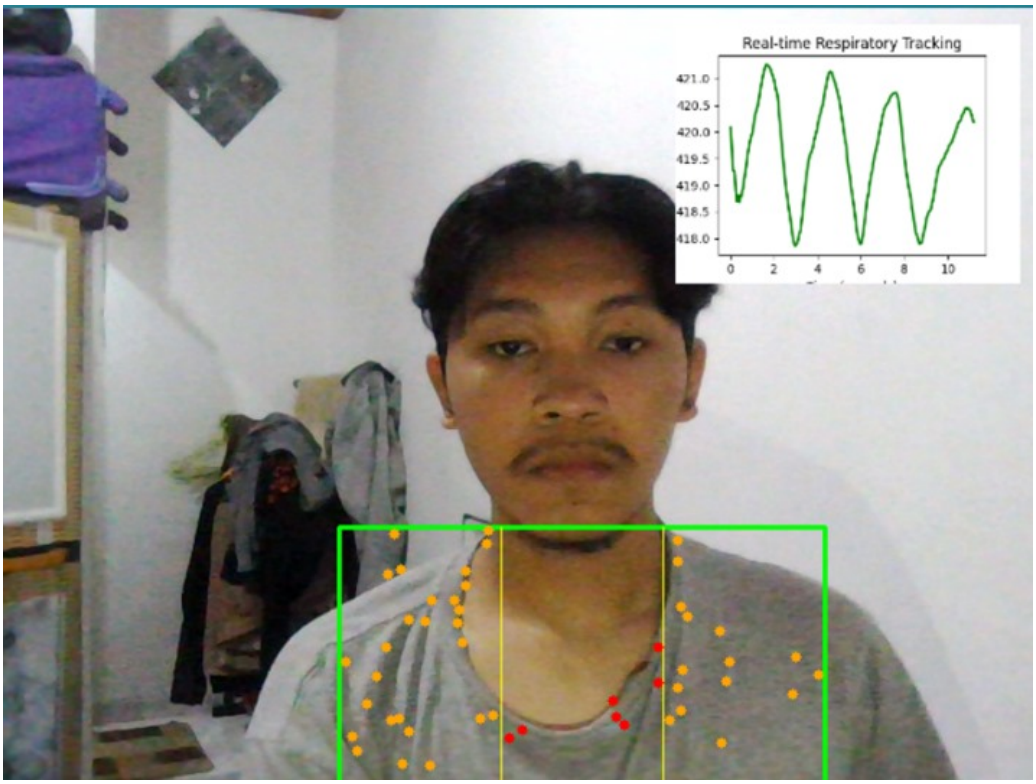
Gambar 2: Gambar Hasil Analisis Sinyal RPPG

- Deteksi puncak ini digunakan untuk menghitung denyut jantung berdasarkan jumlah puncak yang terdeteksi dalam durasi tertentu.

3. **Denyut Jantung (Heart Rate):**
   - Dari grafik, nilai denyut jantung yang dihitung adalah *67,20 BPM (Beats Per Minute)*. Nilai ini relatif rendah, tetapi masih dalam kisaran normal untuk seseorang dalam kondisi istirahat.
4. **Kualitas Sinyal:**
   - Grafik menunjukkan adanya fluktuasi amplitudo yang signifikan pada beberapa titik (misalnya, sekitar detik ke-10 dan ke-20). Hal ini dapat disebabkan oleh gerakan kepala, perubahan cahaya, atau noise lainnya.
   - Sinyal pada bagian lainnya relatif stabil, yang menunjukkan bahwa sistem telah berhasil memproses sebagian besar sinyal meskipun terdapat gangguan.

5.2 Analisis Sinyal Respirasi

Berikut menampilkan hasil tracking pergerakan pernapasan menggunakan implementasi sistem berbasis webcam untuk estimasi respiratory rate. **Penjelasan singkat:**



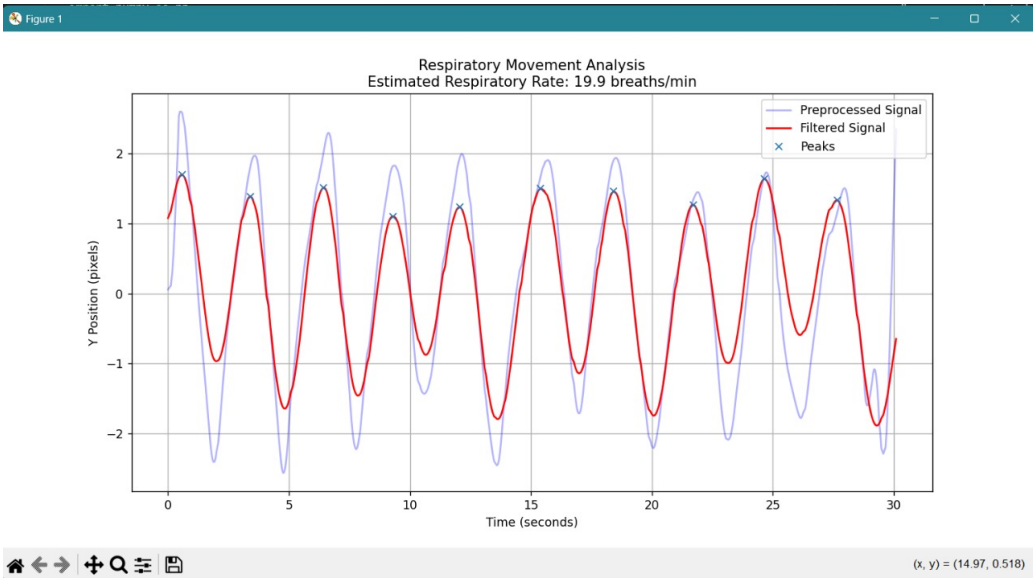
Gambar 3: Implementasi Tracking Pergerakan Respirasi

- **Kotak Hijau:** Region of Interest (ROI) yang mencakup area bahu dan dada atas. ROI ini dibagi menjadi tiga zona vertikal yang ditandai dengan garis kuning untuk optimalisasi tracking.
- **Titik-titik Tracking:**
  - **Titik Merah (Zona Tengah):** Titik-titik tracking utama dengan bobot 1.0, maksimal 20 titik
  - **Titik Oranye (Zona Kiri dan Kanan):** Titik-titik tracking pendukung dengan bobot 0.5, maksimal 15 titik per zona
- **Grafik Real-time:** Plot di pojok kanan atas menampilkan pergerakan vertikal ROI secara real-time dalam rentang waktu 10 detik terakhir, dengan sumbu Y menunjukkan posisi dalam pixel dan sumbu X menunjukkan waktu dalam detik.

5.2.1 Hasil Analisis Sinyal Respirasi

Analisis Komponen Sinyal Respirasi:

1. **Sinyal Preprocessed (Biru):**
   - Menunjukkan data hasil preprocessing awal sebelum filtering
   - *Sumbu X*: Menampilkan waktu pengukuran dalam detik (0-30 detik)
   - *Sumbu Y*: Menunjukkan posisi vertikal dalam satuan pixel
   - Amplitudo sinyal lebih tinggi dengan variasi yang lebih besar, mengindikasikan adanya noise pada data mentah
2. **Sinyal Terfilter (Merah):**
   - Hasil pemrosesan menggunakan bandpass filter Butterworth
   - Menunjukkan pola sinusoidal yang lebih halus dan teratur



Gambar 4: Gambar Hasil Analisis Sinyal Respirasi

- Baseline drift dan noise frekuensi tinggi berhasil dihilangkan
- Amplitudo yang lebih stabil menunjukkan keberhasilan proses filtering

3. **Puncak Terdeteksi (Tanda X):**
   - Menandai titik-titik puncak inspirasi pada sinyal terfilter
   - Jarak antar puncak relatif konsisten, menunjukkan pola pernapasan yang teratur
   - Digunakan untuk perhitungan respiratory rate
4. **Respiratory Rate:**
   - Nilai respiratory rate terukur adalah *19.9 breaths/minute*
   - Nilai ini berada dalam rentang normal untuk orang dewasa (12-20 nafas/menit)
   - Konsistensi periode antar puncak (∼3 detik) menunjukkan irama pernapasan yang stabil
5. **Kualitas Pengukuran:**
   - Signal-to-noise ratio yang baik pada sinyal terfilter, ditunjukkan oleh perbedaan yang jelas antara sinyal biru dan merah
   - Amplitudo puncak yang konsisten (berkisar 1.5-2 pixel) mengindikasikan kedalaman nafas yang stabil
   - Minimal artifak gerakan yang tidak diinginkan, terlihat dari bentuk sinyal yang teratur
   - Deteksi puncak yang akurat dan konsisten sepanjang periode pengukuran

6 Referensi dan Daftar Pustaka

Dalam penelitian ini, kami mengacu pada beberapa referensi terkait pemrosesan sinyal rPPG. Salah satu referensi utama adalah penelitian tentang pemilihan filter digital optimal untuk kondisi sinyal rPPG [1].

References

[1] S. Guler, A. Golparvar, O. Ozturk, H. Dogan, and M. K. Yapici, "Optimal digital filter selection for remote photoplethysmography (rPPG) signal conditioning," 11 Jan. 2023, accessed: 2024-3-24.