Program Studi Teknik Informatika
Institut Teknologi Sumatera

Anggota 1: **Handayani(122140166)**
Anggota 2: **Francois Novalentino Sinurat (121140007)**
Anggota 3: **Dimas Azi Rajab Aizar (121140135)**
Mata Kuliah: **Sistem/Teknologi Multimedia (IF4021)**

Tugas Ke: **Tugas Besar**
Tanggal: **31-05-2025**

SuperPower Menggunakan MediaPipe dan OpenCV

1 Pendahuluan

Teknologi computer vision dan gesture recognition memainkan peran penting dalam menciptakan interaksi manusia-komputer yang lebih alami dan intuitif, dengan deteksi gerakan tangan real-time sebagai elemen kunci untuk interaksi yang dinamis [1]. Pustaka MediaPipe Python yang terintegrasi dengan OpenCV memungkinkan pelacakan hingga 21 titik penting (landmarks) pada tangan secara efisien, dan telah digunakan dalam berbagai aplikasi seperti kontrol permainan rehabilitasi (exergame), interaksi virtual, serta analisis pose untuk pengenalan emosi [2]. Proyek ini kami bangun dengan tema kekuatan super, lebih tepatnya “SuperPower”, memanfaatkan kombinasi MediaPipe dan OpenCV untuk menciptakan interaksi visual berbasis gestur tangan. Sistem ini mendeteksi gerakan jari pengguna dan memberikan efek animasi seperti api atau partikel yang mengikuti gerakan tangan secara dinamis. Efek-efek ini tidak hanya meningkatkan daya tarik visual, tetapi juga memungkinkan pengalaman pengguna yang lebih immersif dan menyenangkan, menyerupai penggunaan "kekuatan super". Penggunaan pendekatan ini sejalan berbagai konteks interaktif, termasuk game edukatif, pelatihan virtual, hingga terapi rehabilitatif yang mendukung pengembangan antarmuka digital yang lebih menarik, responsif, dan meminimalkan beban pengguna [3].

2 Alat dan Cara Kerja

Filter ini dibuat menggunakan bahasa pemrograman Python dengan memanfaatkan beberapa library utama yaitu OpenCV untuk pengolahan dan penampilan citra video, MediaPipe untuk deteksi dan pelacakan tangan secara real-time, NumPy untuk manipulasi data numerik, serta ImageIO dan os untuk membaca file animasi dari sistem. Perangkat keras yang digunakan adalah kamera (webcam) yang berfungsi sebagai input untuk menangkap gerakan tangan pengguna. Sedangkan perangkat lunak yang diperlukan mencakup sistem operasi komputer, Python interpreter, dan editor atau IDE seperti Visual Studio Code atau Jupyter Notebook. Program ini bekerja dengan menangkap video dari kamera, kemudian menggunakan MediaPipe untuk mendeteksi posisi tangan dan jari. Efek animasi dari file .gif atau .mp4 kemudian ditambahkan secara dinamis ke posisi tangan atau ujung jari yang terangkat. Hasil akhirnya ditampilkan secara langsung dalam jendela video, menciptakan pengalaman interaktif berbasis augmented reality.

Proses dimulai dengan pengambilan gambar secara real-time menggunakan kamera, kemudian gambar diproses dan dianalisis menggunakan model deteksi telapak tangan untuk menemukan posisi tangan dalam frame. Setelah posisi telapak tangan teridentifikasi, sistem melanjutkan dengan mende-

teksi 21 landmark tangan yang mencakup sendi dan ujung jari. Deteksi ini bersifat 3D dan sangat presisi, sehingga memungkinkan pelacakan gerakan tangan secara detail dan real-time.

Dalam implementasinya pada program, hasil dari deteksi landmark tangan digunakan untuk menentukan posisi yang tepat untuk menempatkan efek visual (GIF atau MP4) ke dalam video. Misalnya, jika hanya satu jari yang diangkat, efek akan diarahkan ke ujung jari tersebut. Proses ini didukung oleh pemrosesan gambar dengan OpenCV dan penggunaan MediaPipe untuk pelacakan tangan. Integrasi ini menciptakan pengalaman augmented reality yang responsif dan interaktif, di mana pengguna dapat mengendalikan posisi efek hanya dengan gerakan tangan di depan kamera tanpa perlu perangkat tambahan.

3 Penjelasan Kode Program

3.1 Import Library

Library yang diperlukan dalam program ini diantaranya cv2, mediapipe, numpy, imageio, dan os.

```
1
2   import cv2
3   import mediapipe as mp
4   import numpy as np
5   import imageio
6   import os
7   import math
8   import collections
9   from typing import Dict, List, Optional, Tuple
10
```

Kode 3.1: Library yang Digunakan

Penjelasan:

- cv2 (OpenCV): Untuk menangkap video dari kamera, memproses gambar, menampilkan hasil akhir, serta melakukan operasi seperti flip, resize, dan overlay efek.
- mediapipe: Digunakan untuk deteksi dan pelacakan tangan secara real-time. Program memanfaatkan solusi mp.solutions.hands dari MediaPipe untuk mendapatkan 21 titik landmark tangan.
- numpy: Untuk manipulasi array dan perhitungan posisi titik tangan serta blending efek visual dengan frame kamera.
- imageio: Digunakan untuk membaca file animasi berformat .gif sebagai rangkaian frame yang akan di-overlay ke video.
- os: Untuk mengakses dan membaca file efek dari direktori lokal, serta mengelola path file.
- math: Untuk perhitungan matematika, terutama operasi akar kuadrat dan jarak Euclidean.
- collections :Untuk struktur data deque (double-ended queue) yang efisien untuk menyimpan history data dengan batas maksimum.
- from typing import Dict, List, Optional, Tuple : Untuk type hints yang membuat kode lebih mudah dibaca dan di-debug.

3.2 Class HandEffectTracker

```
class HandEffectTracker
```

Kode 3.2: HandEffectTracker

Penjelasan: Class ini adalah komponen utama untuk melacak gerakan tangan dan menambahkan efek visual secara real-time menggunakan webcam.

3.2.1 Fungsi Inisialisasi

```
def __init__(self, effects_folder: str = "effects"):
```

Kode 3.3: Inisialisasi init

Penjelasan: constructor yang otomatis dipanggil ketika membuat instance baru dari class HandEffect-Tracker.

- Menginisialisasi pelacak efek tangan
- Parameter: effects-folder: Direktori yang berisi file efek GIF
- Mengatur pelacakan tangan MediaPipe dengan pengaturan optimal
- Menginisialisasi parameter konfigurasi untuk efek dan pelacakan
- Membuat cache untuk landmark telapak tangan

3.2.1.1 Konfigurasi MediaPipe Hands

```
    self.mp_hands = mp.solutions.hands
       self.hands = self.mp_hands.Hands(
           static_image_mode=False,
           max_num_hands=2,
           min_detection_confidence=0.7,
           min_tracking_confidence=0.5,  # Lower tracking confidence for better performance
           model_complexity=1  # Use simpler model for better FPS
       )

```

Kode 3.4: Konfigurasi MediaPipe Hands

Penjelasan:

- Membuat instance dari kelas Hands dengan parameter khusus:
  - False = Menggunakan tracking antar frame, lebih efisien untuk video
  - True = Setiap frame diproses independen, lebih akurat tapi lambat
- max-num-hands=2: Maksimal jumlah tangan yang dapat dideteksi
- min-detection-confidence=0.7 (Range: 0.0 - 1.0): Batas minimum confidence untuk mendeteksi tangan baru
- min-tracking-confidence=0.5: Batas minimum confidence untuk melanjutkan tracking tangan yang sudah terdeteksi
- model-complexity=1: Kompleksitas model neural network

3.2.1.2 Konfigurasi Hand Effect Tracker

```
    self.config = {
            'alpha': 0.8,                # Increased smoothing
            'base_effect_size': 180,     # Base size for reference
            'max_effect_size': 400,
            'min_effect_size': 60,
            'effect_y_offset': -120,
            'size_smoothing': 0.3,       # Smoothing for size changes
            'hand_size_scale': 1.5,      # Multiplier for hand size to effect size
            'reference_hand_size': 120,  # Reference hand size for scaling
            'fist_threshold': 0.08,      # More reliable fist detection
            'stability_threshold': 15,   # Minimum movement to update position
            'finger_effect_size': 80,    # Size for single finger effects
            'finger_size_scale': 0.8,    # Scale factor for finger effects
            # Trail configuration
            'trail_max_length': 30,      # Maximum number of trail points
            'trail_min_distance': 8,     # Minimum distance between trail points
            'trail_thickness': 8,        # Trail line thickness
            'trail_fade_steps': 15,      # Number of fade steps for trail
        }

```

Kode 3.5: Konfigurasi Hand Effect Tracker

Penjelasan:

- Parameter Smoothing and Stabilitas
  - alpha:
    - Mengontrol smoothing pergerakan
    - Nilai lebih tinggi = gerakan lebih halus tapi lebih lag.
    - Range 0-1

    Mengontrol smoothing pergerakan
    - Threshold minimal pergerakan untuk update posisi
    - Mencegah efek bergetar saat tangan diam.
  - Nilai lebih tinggi = gerakan lebih halus tapi lebih lag.
  - Range 0-1
- Parameter Ukuran Efek
  - base-effect-size: Ukuran dasar efek visual (dalam pixel) dan digunakan sebagai referensi.
  - max-effect-size: Ukuran maksimum efek yang diizinkan dan mencegah efek terlalu besar.
  - min-effect-size: Ukuran minimum efek yang diizinkan dan menjaga efek tetap visible.
  - effect-y-offset: Offset vertikal efek dari posisi tangan dan nilai negatif = efek di atas tangan.
- Parameter Scaling
  - size-smoothing: Mengontrol smoothing perubahan ukuran dan mencegah efek ukuran berubah terlalu cepat.
  - hand-size-scale: Multiplier untuk mengatur ukuran efek relatif terhadap ukuran tangan. 1 = efek lebih besar dari tangan.
  - reference-hand-size: Ukuran tangan referensi dalam pixel dan digunakan untuk scaling relatif.

- Parameter Gesture
  - fist-threshold: Threshold untuk deteksi gesture kepalan dan nilai lebih tinggi = deteksi lebih ketat.
  - finger-effect-size: Ukuran efek khusus untuk gesture jari tunggal dan biasanya lebih kecil dari efek tangan penuh.
  - finger-size-scale: Faktor skala untuk efek jari dan <1 = efek jari lebih kecil dari efek tangan
- Konfigurasi Trail (Jejak)
  - trail-max-length: Jumlah maksimum titik dalam jejak dan mengontrol panjang jejak.
  - trail-min-distance: Jarak minimum antar titik jejak dan mencegah jejak terlalu detail/padat.
  - trail-thickness: Ketebalan garis jejak dalam pixel.
  - trail-fade-steps: Jumlah langkah untuk fade out jejak dan mengontrol efek memudar.

3.2.1.3 Palm Landmarks

```
    self.palm_landmarks = [0, 5, 9, 13, 17]  # More stable landmarks
```

Kode 3.6: Palm Landmarks

Penjelasan: Merupakan array yang menyimpan indeks landmark telapak tangan yang paling stabil untuk pelacakan. Landmarks ini digunakan terutama dalam fungsi *-calculate-palm-center()* untuk menentukan posisi tengah telapak tangan yang stabil sebagai referensi penempatan efek.

3.2.2 Fungsi Loading Efek

```
    def _load_effects(self, folder_path: str, flip_horizontal: bool = True) -> Dict:
```

Kode 3.7: Loading Efek

Penjelasan: Fungsi ini bertujuan untuk memuat efek-efek GIF dari folder yang ditentukan ke dalam program.

3.2.2.1 Inisialisasi

```
    effects = {}
```

Kode 3.8: Inisialisasi

Penjelasan: Membuat dictionary kosong untuk menyimpan efek.

3.2.2.2 Validasi Folder

```
    if not os.path.exists(folder_path):
    print(f"Warning: Effects directory not found: {folder_path}")
    return effects
```

Kode 3.9: Validasi Folder

Penjelasan:Mengecek apakah folder efek ada dan jika tidak ada, mengembalikan dictionary kosong.

3.2.2.3 Iterasi File GIF

```
    for filename in os.listdir(folder_path):
    if not filename.lower().endswith('.gif'):
        continue
```

Kode 3.10: Validasi Folder

Penjelasan:Loop setiap file dalam folder dan skip file non-GIF

3.2.2.4 Persiapan Nama dan Path

```
    name = os.path.splitext(filename)[0]  # Nama file tanpa ekstensi
    full_path = os.path.join(folder_path, filename)  # Path lengkap
```

Kode 3.11: Persiapan Nama dan Path

3.2.2.5 Loading dan Konversi GIF

```
    try:
    gif_frames = imageio.mimread(full_path)  # Baca GIF
    if flip_horizontal:
        frames = [cv2.flip(np.array(f), 1) for f in gif_frames]  # Flip horizontal
    else:
        frames = [np.array(f) for f in gif_frames]  # Konversi ke numpy array
```

Kode 3.12: Validasi Folder

Penjelasan:

- Memuat frame-frame GIF menggunakan imageio
- Konversi ke numpy array

3.2.2.6 Penyimpanan ke Dictionary

```
    effects[name] = {
    "type": "gif",
    "frames": frames,
    "frame_count": len(frames)
    }
```

Kode 3.13: Penyimpanan ke Dictionary

Penjelasan: Menyimpan informasi efek dalam format:

- type: Tipe efek (gif)
- frames: Array frame-frame
- frame-count: Jumlah frame

3.2.2.7 Error Handling

```
    try:
    # ...loading code...
    except Exception as e:
    print(f"Error loading {filename}: {e}")
```

Kode 3.14: Error Handling

Penjelasan: Menangani error saat loading file dan mencegah crash program jika ada file rusak.

3.2.2.8 Mengembalikan Dictionary Effect

```
    return effects
```

Kode 3.15: Mengembalikan Dictionary Efek

Penjelasan: Untuk mengembalikan dictionary yang berisi seluruh efek GIF yang telah dimuat.

3.2.3 Deteksi Gesture

```
    def _detect_gesture(self, landmarks) -> Tuple[str, Optional[Tuple[float, float]]]:
```

Kode 3.16: Deteksi Gesture

Penjelasan: Fungsi ini mendeteksi gestur tangan berdasarkan posisi jari-jari dan mengembalikan tipe gestur beserta posisinya.

3.2.3.1 Definisi Landmark Jari

```
    fingers = {
    'thumb': [4, 3, 2],      # Jempol: ujung, tengah, pangkal
    'index': [8, 6, 5],      # Telunjuk
    'middle': [12, 10, 9],   # Tengah
    'ring': [16, 14, 13],    # Manis
    'pinky': [20, 18, 17]    # Kelingking
    }
```

Kode 3.17: Definisi Landmark Jari

3.2.3.2 Array Penyimpanan

```
    extended_fingers = []      # Menyimpan jari yang terangkat
    finger_positions = {}      # Menyimpan posisi ujung jari
```

Kode 3.18: Array Penyimpanan

3.2.3.3 Deteksi Jari

```
    for finger_name, (tip, pip, mcp) in fingers.items():
    # Deteksi khusus untuk ibu jari
    if finger_name == 'thumb':
        is_extended = abs(landmarks[tip].x - landmarks[mcp].x) > 0.04
    # Deteksi untuk jari lainnya
    else:
        is_extended = (landmarks[tip].y < landmarks[pip].y and
                       landmarks[tip].y < landmarks[mcp].y)
```

Kode 3.19: Deteksi Jari

3.2.3.4 Penentuan Gestur

```
    # Kepalan Tangan
    if len(extended_fingers) == 0:
        return "fist", None
```

```
    # Satu Jari
    elif len(extended_fingers) == 1:
        finger_name = extended_fingers[0]
        return f"single_{finger_name}", finger_positions[finger_name]

    # Tangan Terbuka
    else:
        return "open_hand", None

```

Kode 3.20: Penentuan Gestur

Penjelasan:

- Kepalan Tangan (Fist)
  - Tidak ada jari terangkat
  - Return: ("fist", None)
- Satu Jari (Single Finger)
  - Satu jari terangkat
  - Return: ("single-[nama-jari]", (x, y))
  - Contoh: ("single-index", (0.6, 0.4))
- Tangan Terbuka (Open Hand)
  - Lebih dari satu jari terangkat
  - Return: ("open-hand", None)
- Unknown
  - Saat terjadi error
  - Return: ("unknown", None)

3.2.3.5 Error Handling

```

    try:
        # kode deteksi
    except Exception as e:
        print(f"Gesture detection error: {e}")
        return "unknown", None

```

Kode 3.21: Error Handling

Penjelasan:

- Try Block
  - Berisi kode utama deteksi gesture
  - Mencakup operasi yang mungkin gagal: Akses landmark coordinates, Kalkulasi jarak, Pengecekan kondisi jari
- Exception Handling: Menangkap semua jenis exception dan menyimpan error dalam variabel e
- Error Response: Mencetak pesan error dengan detail dan format: "Gesture detection error: [pesan error spesifik]"
- Return Value: Gesture type: "unknown", Position: None, dan menandakan gesture tidak dapat dideteksi

3.2.4 Fungsi Menghitung Pusat Telapak Tangan

```

    def _calculate_palm_center(self, landmarks) -> Tuple[float, float, float]:


```

Kode 3.22: Menghitung Pusat Telapak Tangan

Penjelasan: Untuk menghitung posisi pusat telapak tangan dengan mengambil rata-rata koordinat dari landmark-landmark telapak tangan yang stabil.

3.2.4.1 Ekstraksi Koordinat

```

    x_coords = [landmarks[i].x for i in self.palm_landmarks]
    y_coords = [landmarks[i].y for i in self.palm_landmarks]
    z_coords = [landmarks[i].z for i in self.palm_landmarks if landmarks[i].z is not None]

```

Kode 3.23: Ekstraksi Koordinat

Penjelasan:

- Mengambil koordinat x, y, z dari landmark telapak tangan

- self.palm$_landmarks = [0, 5, 9, 13, 17](wristdantitikMCP)Filterzkoordinatyangtidak None$

3.2.4.2 Kalkulasi Rata-rata

```

    cx = np.mean(x_coords)  # Rata-rata koordinat X
    cy = np.mean(y_coords)  # Rata-rata koordinat Y
    cz = np.mean(z_coords) if z_coords else None  # Rata-rata koordinat Z jika ada

```

Kode 3.24: Kalkulasi Rata-rata

Penjelasan:

- Menghitung ukuran tangan berdasarkan:
  - Panjang tangan (pergelangan ke ujung jari tengah)
  - Lebar telapak (jarak antara MCP jari kelingking dan telunjuk)
  - Rentang jempol

3.2.4.3 Return Value

```

    return cx, cy, cz

```

Kode 3.25: Return Value

Penjelasan:

- cx: Posisi x pusat telapak (0-1)
- cy: Posisi y pusat telapak (0-1)
- cz: Kedalaman pusat telapak

3.2.5 Fungsi Menghitung Ukuran Tangan

```
1
2   def _calculate_hand_size(self, landmarks, frame_dims: Tuple[int, int]) -> float:
3
```

Kode 3.26: Menghitung Ukuran Tangan

Penjelasan: Fungsi ini menghitung ukuran tangan menggunakan tiga metode pengukuran berbeda untuk mendapatkan estimasi yang stabil.

3.2.5.1 Panjang Tangan (Hand Length)

```
1
2   wrist = landmarks[0]         # Pergelangan tangan
3   middle_tip = landmarks[12]   # Ujung jari tengah
4   hand_length = math.sqrt(
5       (middle_tip.x - wrist.x)**2 + (middle_tip.y - wrist.y)**2
6   ) * min_dim
7
```

Kode 3.27: Panjang Tangan (Hand Length)

Penjelasan:

- Mengukur jarak dari pergelangan ke ujung jari tengah
- Menggunakan rumus Euclidean distance
- Bobot: 0.6 (60

3.2.5.2 Lebar Telapak (Palm Width)

```
1
2   index_mcp = landmarks[5]     # Pangkal jari telunjuk
3   pinky_mcp = landmarks[17]    # Pangkal jari kelingking
4   palm_width = math.sqrt(
5       (pinky_mcp.x - index_mcp.x)**2 + (pinky_mcp.y - index_mcp.y)**2
6   ) * min_dim
7
```

Kode 3.28: Lebar Telapak (Palm Width)

Penjelasan:

- Mengukur lebar telapak tangan
- Jarak antara pangkal telunjuk dan kelingking
- Bobot: 1.8 (180

3.2.5.3 Rentang Ibu Jari (Thumb Span)

```
1
2   thumb_tip = landmarks[4]     # Ujung ibu jari
3   thumb_span = math.sqrt(
4       (thumb_tip.x - wrist.x)**2 + (thumb_tip.y - wrist.y)**2
5   ) * min_dim
6
```

Kode 3.29: Ekstraksi Koordinat

Penjelasan:

- Mengukur jarak dari pergelangan ke ujung ibu jari

- Bobot: 0.4 (40

3.2.5.4 Kalkulasi Akhir

```
hand_size = (hand_length * 0.6 + palm_width * 1.8 + thumb_span * 0.4)
```

Kode 3.30: Kalkulasi Akhir

Penjelasan:

- Palm Width: 1.8 (paling besar) karena:
  - Lebih stabil saat tangan bergerak
  - Kurang terpengaruh gerakan jari
- Hand Length: 0.6 (menengah) karena:
  - Memberikan indikasi ukuran keseluruhan
  - Bisa berubah saat jari bergerak
- Thumb Span: 0.4 (paling kecil) karena:
  - Paling tidak stabil
  - Sangat terpengaruh gerakan ibu jari

3.2.5.5 Mengembalikan Nilai

```
    return hand_size
```

Kode 3.31: Mengembalikan Nilai

Penjelasan: Mengembalikan nilai ukuran tangan yang telah dikalkulasi melalui weighted average dari tiga pengukuran.

3.2.6 Fungsi *overlay-effect-optimized*

```
    def _overlay_effect_optimized(self, base_frame: np.ndarray, effect_frame: np.ndarray,
                                  x: int, y: int, size: int) -> None:
```

Kode 3.32: Fungsi *overlay-effect-optimized*

Penjelasan: Untuk menggabungkan efek visual api ke frame video dengan cara yang optimal dan efisien.

3.2.6.1 Inisialisasi dan Bounds Checking

```
    h, w = base_frame.shape[:2]
    half_size = size // 2
    x1, y1 = max(0, x - half_size), max(0, y - half_size)
    x2, y2 = min(w, x + half_size), min(h, y + half_size)
```

Kode 3.33: Inisialisasi dan Bounds Checking

Penjelasan:

- Mendapatkan dimensi frame
- Menghitung area efek
- Memastikan koordinat dalam batas frame

3.2.6.2 Early Exit

```

    if x1 >= x2 or y1 >= y2:
    return

```

Kode 3.34: Early Exit

Penjelasan: Mencegah pemrosesan jika efek di luar frame

3.2.6.3 Kalkulasi Region Efek

```

    effect_x1 = half_size - (x - x1)
    effect_y1 = half_size - (y - y1)
    effect_x2 = effect_x1 + (x2 - x1)
    effect_y2 = effect_y1 + (y2 - y1)

```

Kode 3.35: Ekstraksi Koordinat

Penjelasan: Menghitung area yang akan dioverlay

3.2.6.4 Pemrosesan Efek

```

    effect_resized = cv2.resize(effect_frame, (size, size),
                                interpolation=cv2.INTER_LINEAR)
    effect_region = effect_resized[effect_y1:effect_y2, effect_x1:effect_x2]

```

Kode 3.36: Pemrosesan Efek

Penjelasan: Resize efek ke ukuran yang diinginkan dan ekstrak region yang diperlukan

3.2.6.5 Alpha Blending

```

    if effect_region.shape[2] == 4:  # RGBA
    effect_rgb = effect_region[:, :, :3]
    alpha = effect_region[:, :, 3:4] / 255.0

    roi = base_frame[y1:y2, x1:x2]
    blended = (alpha * effect_rgb + (1 - alpha) * roi).astype(np.uint8)
    base_frame[y1:y2, x1:x2] = blended

```

Kode 3.37: Ekstraksi Koordinat

Penjelasan:

- Untuk efek dengan transparansi (RGBA)
- Memisahkan channel RGB dan alpha
- Melakukan blending dengan formula: result = alpha * effect + (1 - alpha) * background

3.2.6.6 RGB Handling

```
    else:  # RGB
    base_frame[y1:y2, x1:x2] = effect_region
```

Kode 3.38: RGB Handling

Penjelasan: Untuk efek tanpa transparansi dan langsung overlay ke frame

3.2.6.7 Error Handling

```
    try:
    # processing code
    except Exception as e:
        print(f"Overlay error: {e}")
```

Kode 3.39: Ekstraksi Koordinat

Penjelasan: Menangani error saat pemrosesan dan mencegah crash program

3.2.7 Fungsi *-draw-finger-trail*

```
    def _draw_finger_trail(self, frame: np.ndarray, trail_points: List[Tuple[int, int]]) -> None:
```

Kode 3.40: Fungsi *-draw-finger-trail*

Penjelasan: Fungsi ini membuat efek trail/jejak visual di belakang gerakan jari dengan efek bayangan dan cahaya.

3.2.7.1 Early Return

```
    if len(trail_points) < 2:
    return
```

Kode 3.41: Early Return

Penjelasan: Cek minimal ada 2 titik untuk membuat garis dan mencegah error saat trail belum terbentuk

3.2.7.2 Konfigurasi Efek

```
    shadow_offsets = [(2, 2), (1, 1), (0, 0)]  # Offset bayangan
    shadow_colors = [(0, 0, 100), (0, 0, 150), (0, 0, 255)]  # Gradasi warna merah
```

Kode 3.42: Konfigurasi Efek

Penjelasan: 3 layer bayangan dengan offset berbeda dan warna dari merah gelap ke terang

3.2.7.3 Pemrosesan Trail

```
    for offset_idx, (dx, dy) in enumerate(shadow_offsets):
    for i in range(1, len(trail_points)):
```

Kode 3.43: Untuk setiap layer bayangan

Penjelasan: Cek minimal ada 2 titik untuk membuat garis dan mencegah error saat trail belum terbentuk

```
    fade_factor = (i / len(trail_points)) * 0.8 + 0.2  # Range 0.2-1.0

```

Kode 3.44: Kalkulasi Fade

Penjelasan: Mengatur transparansi berdasarkan posisi di trail dan semakin ke belakang semakin transparan

```
    pt1 = (trail_points[i-1][0] + dx, trail_points[i-1][1] + dy)
    pt2 = (trail_points[i][0] + dx, trail_points[i][1] + dy)

```

Kode 3.45: Posisi Trail dengan Offset

Penjelasan: Menambah offset untuk efek bayangan dan membuat kedalaman visual

```
    thickness = int(self.config['trail_thickness'] * fade_factor)
    thickness = max(1, thickness)  # Minimal 1 pixel

```

Kode 3.46: Ketebalan Trail

Penjelasan: Trail menipis di bagian belakang dan menggunakan konfigurasi dari self.config

```
    base_color = shadow_colors[offset_idx]
    color = tuple(int(c * fade_factor) for c in base_color)

```

Kode 3.47: Warna dengan Fade

Penjelasan: Mengaplikasikan fade ke warna dasar dan membuat gradasi transparansi

```
    cv2.line(frame, pt1, pt2, color, thickness)

```

Kode 3.48: Main Loop

Penjelasan: Menggambar segmen garis pada frame video menggunakan OpenCV

```
    if offset_idx == 2:  # Main trail
    glow_thickness = max(1, thickness // 2)
    glow_color = (50, 50, 255)  # Merah terang
    cv2.line(frame, pt1, pt2, glow_color, glow_thickness)

```

Kode 3.49: Efek Cahaya (Glow)

Penjelasan: Menambah garis tipis terang di atas trail utama dan memberi efek cahaya/glow

3.2.8 Fungsi *-update-finger-trail*

```
    def _update_finger_trail(self, state: Dict, current_pos: Tuple[int, int], current_finger: str)
    -> None:

```

Kode 3.50: Fungsi *-update-finger-trail*

Penjelasan: Fungsi ini mengupdate titik-titik trail (jejak) yang mengikuti gerakan jari.

3.2.8.1 Inisialisasi Trail

```
    if 'trail_points' not in state:
    state['trail_points'] = collections.deque(maxlen=self.config['trail_max_length'])
    state['last_trail_finger'] = None

```

Kode 3.51: Inisialisasi Trail

Penjelasan:

- Membuat deque baru jika belum ada
- maxlen: Membatasi panjang trail
- Mencatat jari terakhir yang ditrack

3.2.8.2 Reset Trail

```
    if 'trail_points' not in state:
    state['trail_points'] = collections.deque(maxlen=self.config['trail_max_length'])
    state['last_trail_finger'] = None

```

Kode 3.52: Reset Trail

Penjelasan: Reset trail jika:

- Jari yang ditrack berubah
- Gesture bukan single finger

3.2.8.3 Penambahan Titik Trail

```
    if (len(state['trail_points']) == 0 or
    math.sqrt((current_pos[0] - state['trail_points'][-1][0])**2 +
              (current_pos[1] - state['trail_points'][-1][1])**2) >= self.config['trail_min_distance
    ']):
    state['trail_points'].append(current_pos)

```

Kode 3.53: Penambahan Titik Trail

Penjelasan: Menambah titik baru jika:

- Trail kosong
- Jarak dari titik terakhir >= *trail-min-distance*

3.2.9 Fungsi *-update-hand-state*

```
    def _update_finger_trail(self, state: Dict, current_pos: Tuple[int, int], current_finger: str)
    -> None:

```

Kode 3.54: Fungsi *-update-hand-state*

Penjelasan: Fungsi ini mengupdate status tracking tangan untuk setiap frame, termasuk posisi, gesture, dan efek trail.

3.2.9.1 Inisialisasi State

```
    if hand_idx not in self.hand_states:
    self.hand_states[hand_idx] = {
        # Smoothing Coordinates
        'smooth_x': None,        # Koordinat X yang dihaluskan
        'smooth_y': None,        # Koordinat Y yang dihaluskan
        'smooth_size': None,     # Ukuran efek yang dihaluskan

        # Animation Control
        'frame_index': 0,        # Index frame animasi GIF

        # Position Tracking
        'position_history': collections.deque(maxlen=5),  # Riwayat 5 posisi terakhir
        'is_stable': False,      # Status kestabilan tangan

        # Gesture Info
        'current_gesture': 'unknown',  # Gesture yang terdeteksi
        'finger_position': None,       # Posisi jari (x,y)

        # Trail System
        'trail_points': collections.deque(maxlen=self.config['trail_max_length']),  # Titik-titik
    trail
        'last_trail_finger': None      # Jari terakhir yang membuat trail
    }

```

Kode 3.55: Inisialisasi State

Penjelasan: Mengecek apakah tangan dengan index tertentu sudah memiliki state dan jika belum, membuat state baru

- Menghasilkan gerakan yang lebih halus
- Kontrol animasi efek
- Stabilitas dan prediksi gerakan
- Kontrol efek berdasarkan gesture
- Membuat efek trail/jejak

```
    state = self.hand_states[hand_idx]

```

Kode 3.56: mengambil state (kondisi) tangan

Penjelasan: Menyederhanakan akses ke state tangan yang sedang diproses dan membuat referensi lokal untuk efisiensi kode

3.2.9.2 Deteksi Gesture

```
    gesture, finger_pos = self._detect_gesture(landmarks)
    previous_gesture = state['current_gesture']
    state['current_gesture'] = gesture

```

Kode 3.57: Deteksi Gesture

Penjelasan:

- Memanggil fungsi *-detect-gesture* untuk menganalisis landmark tangan

- Menyimpan gesture yang terdeteksi sebelumnya
- Memperbarui state dengan gesture yang baru terdeteksi

3.2.9.3 Penanganan Reset

```
    # Reset trail if gesture changed from single finger to something else
        if (previous_gesture.startswith('single_') and
            not gesture.startswith('single_')):
            state['trail_points'].clear()
            state['last_trail_finger'] = None

        # Reset if fist detected
        if gesture == "fist":
            state.update({
                'smooth_x': None, 'smooth_y': None, 'smooth_size': None,
                'position_history': collections.deque(maxlen=5),
                'is_stable': False, 'finger_position': None
            })
            state['trail_points'].clear()
            state['last_trail_finger'] = None
            return

```

Kode 3.58: Penanganan Reset

Penjelasan:

- Kondisi 1: *previous-gesture.startswith('single-')*> Mengecek apakah gesture sebelumnya adalah single finger
- Kondisi 2: *not gesture.startswith'single-'*> Mengecek apakah gesture saat ini BUKAN single finger
- if gesture == "fist": Mengecek apakah gesture yang terdeteksi adalah kepalan tangan
- *'smooth-x': None*: Reset posisi X yang dihaluskan
- *'smooth-y': None*: Reset posisi Y yang dihaluskan
- *'smooth-size'*: None: Reset ukuran efek
- *'position-history'*: collections.deque(maxlen=5): Reset history posisi
- *'is-stable': False*: Reset status stabilitas
- *'finger-position'*: None: Reset posisi jari
- *state['trail-points'].clear()*: Hapus semua titik trail
- *state['last-trail-finger']* = None: Reset jari yang ditrack

3.2.9.4 Pemrosesan Gesture

```
    # Bagian 1: Penanganan Single Finger Gesture
if gesture.startswith("single_"):
    # Cek apakah posisi jari terdeteksi
    if finger_pos:
        # Konversi koordinat normalized (0-1) ke pixel
        cx, cy = finger_pos                          # Ambil koordinat normalized
        cx_px, cy_px = int(cx * min_dim), int(cy * min_dim)  # Konversi ke pixel
```

```
        state['finger_position'] = (cx_px, cy_px)    # Simpan posisi jari

        # Update trail effect untuk jari
        self._update_finger_trail(state, (cx_px, cy_px), gesture)

        # Kalkulasi ukuran efek untuk single finger
        # 1. Hitung ukuran tangan saat ini
        current_hand_size = self._calculate_hand_size(landmarks, frame_dims)
        # 2. Hitung rasio terhadap ukuran referensi
        size_ratio = current_hand_size / self.config['reference_hand_size']
        # 3. Kalkulasi ukuran efek final dengan scaling
        target_effect_size = int(self.config['finger_effect_size'] * size_ratio * self.config['
    finger_size_scale'])
        # 4. Terapkan batas minimum 40px dan maximum 150px
        target_effect_size = max(40, min(150, target_effect_size))
    else:
        return  # Keluar jika posisi jari tidak terdeteksi

```

Kode 3.59: Single Finger Gesture

Penjelasan:

- Menangani gesture jari tunggal
- Menggunakan posisi ujung jari untuk efek
- Membuat trail effect
- Menggunakan ukuran efek yang lebih kecil

```

# Bagian 2: Penanganan Open Hand Gesture
 else:
        # Kalkulasi pusat telapak tangan
        cx, cy, cz = self._calculate_palm_center(landmarks)  # Dapatkan koordinat center
        cx_px, cy_px = int(cx * min_dim), int(cy * min_dim) # Konversi ke pixel
        state['finger_position'] = None  # Reset posisi jari

        # Bersihkan trail jika sebelumnya single finger
        if previous_gesture.startswith('single_'):
            state['trail_points'].clear()       # Hapus semua titik trail
            state['last_trail_finger'] = None   # Reset jari terakhir

        # Kalkulasi ukuran efek untuk open hand
        # 1. Hitung ukuran tangan saat ini
        current_hand_size = self._calculate_hand_size(landmarks, frame_dims)
        # 2. Hitung rasio terhadap ukuran referensi
        size_ratio = current_hand_size / self.config['reference_hand_size']
        # 3. Kalkulasi ukuran efek dengan scaling
        target_effect_size = int(self.config['base_effect_size'] * size_ratio * self.config['
    hand_size_scale'])
        # 4. Terapkan batas min/max dari config
        target_effect_size = max(self.config['min_effect_size'],
                                 min(self.config['max_effect_size'], target_effect_size))

```

Kode 3.60: Open Hand Gesture

Penjelasan:

- Menangani gesture tangan terbuka

- Menggunakan pusat telapak tangan
- Menghapus trail dari gesture sebelumnya
- Menggunakan ukuran efek normal

3.2.9.5 Smoothing

```
    # Smooth the effect size menggunakan exponential smoothing
    if state['smooth_size'] is None:
        # Jika belum ada ukuran sebelumnya, gunakan ukuran target langsung
        state['smooth_size'] = target_effect_size
    else:
        # Ambil nilai alpha untuk smoothing dari config (0-1)
        size_alpha = self.config['size_smoothing']  # Default: 0.3

        # Aplikasikan formula exponential smoothing:
        # - Hasil dikonversi ke integer untuk ukuran pixel
        state['smooth_size'] = int(size_alpha * target_effect_size +
                                   (1 - size_alpha) * state['smooth_size'])

```

Kode 3.61: Smoothing Ukuran

Penjelasan:

- *smooth-size* adalah teknik untuk membuat perubahan ukuran efek lebih halus dengan menggunakan *exponential smoothing*. Formula:

$$new - size = \alpha \cdot target + (1 - \alpha) \cdot previous$$

  Di mana α (alpha) mengontrol seberapa cepat perubahan terjadi. Nilai α yang kecil membuat perubahan lebih halus tetapi terasa lebih lambat (lag).

- *size-alpha* $= 0.3$ artinya: 30% dari ukuran target dan 70% dari ukuran sebelumnya.
- Hasil akhir dikonversi ke integer karena ukuran piksel harus berupa bilangan bulat.

```
    # Smooth position updates - Menghaluskan pergerakan posisi
    if state['smooth_x'] is None:
        # Inisialisasi awal posisi smooth jika belum ada
        state['smooth_x'], state['smooth_y'] = cx_px, cy_px  # Set posisi awal
        state['is_stable'] = True  # Set status stabil
    else:
        # Hitung jarak perpindahan dari posisi sebelumnya
        distance = math.sqrt((cx_px - state['smooth_x'])**2 +
                             (cy_px - state['smooth_y'])**2)

        # Cek apakah perpindahan cukup signifikan
        if distance > self.config['stability_threshold']:  # Default: 15px
        # Ambil nilai alpha untuk smoothing dari config (0-1)
        alpha = self.config['alpha']  # Default: 0.8

        # Aplikasikan exponential smoothing:
        state['smooth_x'] = int(alpha * cx_px + (1 - alpha) * state['smooth_x'])
        state['smooth_y'] = int(alpha * cy_px + (1 - alpha) * state['smooth_y'])
        state['is_stable'] = True  # Update status stabil


```

Kode 3.62: Smoothing Posisi

Penjelasan:

- **Inisialisasi:** Jika belum ada posisi smooth, gunakan posisi saat ini. Set `status_stabil = True`.
- **Deteksi Pergerakan:** Hitung jarak Euclidean antara posisi baru dan posisi smooth sebelumnya. Cek apakah jarak > threshold (default: 15px).
- **Smoothing Process:** Menggunakan exponential smoothing. Formula:

$$new - pos = \alpha \cdot current + (1 - \alpha) \cdot previous$$

  dengan $\alpha = 0.8$ untuk memberikan respons cepat tapi tetap smooth. Konversi ke integer untuk koordinat piksel.
- **Stability Control:** Hanya update posisi jika pergerakan signifikan. Mencegah jitter pada pergerakan kecil. Menghasilkan tracking yang lebih stabil.

3.2.9.6 Update History

```
    # Update position history - Menambahkan posisi terbaru ke history tracking
    if state['is_stable']:  # Cek apakah posisi tangan stabil
        # Tambahkan tuple koordinat (x,y) yang sudah dihaluskan ke history
        state['position_history'].append((state['smooth_x'], state['smooth_y']))

```

Kode 3.63: Single Finger Gesture

Penjelasan:

- Kondisi Stabilitas: *if state['is-stable']*: Hanya update history jika posisi tangan stabil. Mencegah tracking noise dan gerakan tidak sengaja
- Data yang Disimpan: *(state['smooth-x'], state['smooth-y'])*: Koordinat yang sudah di-smooth. Format: Tuple berisi (x, y) dalam pixel coordinates
- Struktur Penyimpanan: Menggunakan collections.deque dengan maxlen=5. Otomatis menghapus data terlama saat mencapai batas. Efisien untuk operasi append/pop

3.2.10 Fungsi *process-frame*

```
    def process_frame(self, frame: np.ndarray) -> np.ndarray:
        """Process a single frame with hand tracking and effects"""

```

Kode 3.64: Fungsi *process-frame*

Penjelasan: Fungsi ini memproses setiap frame video untuk mendeteksi dan menambahkan efek visual pada tangan.

3.2.10.1 Persiapan Frame

```
    # Increment frame counter
    self.frame_count += 1  # Menghitung jumlah frame yang telah diproses

    # Prepare frame
    # 1. Flip horizontal (mirror effect)
    frame = cv2.flip(frame, 1)  # Parameter 1 = flip horizontal

```

```
    # 2. Resize frame ke resolusi standar
    frame = cv2.resize(frame, (960, 720))  # Width=960px, Height=720px

    # 3. Dapatkan dimensi frame
    h, w = frame.shape[:2]  # h=height, w=width dari frame
    min_dim = min(h, w)    # Ambil dimensi terkecil untuk crop square

    # 4. Crop frame menjadi persegi (square)
    # Hitung titik awal crop
    x_start = (w - min_dim) // 2  # Posisi X mulai crop
    y_start = (h - min_dim) // 2  # Posisi Y mulai crop

    # Lakukan cropping
    frame_square = frame[y_start:y_start + min_dim,  # Crop vertikal
                         x_start:x_start + min_dim]    # Crop horizontal

```

Kode 3.65: Fungsi *-update-hand-state*

Penjelasan:

- Frame Counter: Melacak jumlah frame yang diproses. Berguna untuk debugging dan analisis. Flip Horizontal
- Membuat efek cermin: Lebih intuitif untuk pengguna. Resize Frame
- Standarisasi ukuran frame: Optimasi performa. Konsistensi tampilan.
- Crop to Square: Memastikan frame berbentuk persegi. Memudahkan pemrosesan efek. Konsistensi aspect ratio

3.2.10.2 Hand Detection

```

    rgb = cv2.cvtColor(frame_square, cv2.COLOR_BGR2RGB)

```

Kode 3.66: Konversi Warna

Penjelasan:

- OpenCV menggunakan format BGR (Blue, Green, Red)
- MediaPipe membutuhkan format RGB (Red, Green, Blue)
- cv2.cvtColor() melakukan konversi warna

```

    results = self.hands.process(rgb)

```

Kode 3.67: Deteksi Tangan

Penjelasan:

- Menggunakan MediaPipe Hands untuk deteksi
- Input: Frame dalam format RGB
- *results.multi-hand-landmarks*: List landmark untuk setiap tangan terdeteksi
- *results.multi-handedness*: List klasifikasi tangan (kiri/kanan)

- Setiap landmark berisi koordinat x, y, z (normalized 0-1)

3.2.10.3 Hand Tracking dan State Management

```
        # Memeriksa apakah ada tangan yang terdeteksi
        if results.multi_hand_landmarks:
            # Membuat set index tangan yang terdeteksi saat ini
            detected_indices = set(range(len(results.multi_hand_landmarks)))

            # Membersihkan state tangan yang tidak terdeteksi lagi
            for idx in list(self.hand_states.keys()):
                if idx not in detected_indices:
                    del self.hand_states[idx]

```

Kode 3.68: Hand Tracking dan State Management

Penjelasan:

- Pengecekan Deteksi Tangan
  - *if results.multi-hand-landmarks*: Mengecek apakah ada tangan yang terdeteksi
  - *results.multi-hand-landmarks* berisi list landmark untuk setiap tangan
- Membuat Set Index Tangan
- Membersihkan State Tidak Terpakai
  - Iterasi melalui semua state tangan yang tersimpan
  - Menghapus state tangan yang tidak lagi terdeteksi
  - Mencegah memory leak dan state yang tidak valid

3.2.10.4 Proses Setiap Tangan

```
        # Iterasi untuk setiap tangan yang terdeteksi dalam frame
        for hand_idx, hand_landmarks in enumerate(results.multi_hand_landmarks):
            # Update state tangan dengan informasi landmark terbaru
            self._update_hand_state(
                hand_idx,                    # Index tangan (0: tangan pertama, 1: tangan kedua)
                hand_landmarks.landmark,     # Array 21 landmark points dari MediaPipe
                (h, w)                      # Dimensi frame (height, width)
            )

            # Mengambil state tangan yang sudah diupdate untuk pemrosesan selanjutnya
            state = self.hand_states[hand_idx]

```

Kode 3.69: Proses Setiap Tangan

Penjelasan:

- Loop Enumerasi: *enumerate(results.multi-hand-landmarks)*
  - Memberikan index dan data landmark untuk setiap tangan
  - *hand-idx*: Nomor urut tangan (0, 1)
  - *hand-landmarks*: Informasi landmark dari MediaPipe (21 titik)
- Update State: *fungsi -update-hand state()*

- Mendeteksi gesture tangan
- Memperbarui posisi tracking
- Menghitung ukuran efek
- Mengupdate trail effect
- Melakukan smoothing pergerakan

- Akses State: *state = self.hand-states[hand-idx]*
  - Mengambil state tangan untuk:
  - Render efek api
  - Menggambar trail
  - Menampilkan debug info
  - Mengatur animasi

3.2.10.5 Drawing Trail Effect

```
        # Cek apakah gesture adalah single finger dan ada minimal 2 titik trail
        if (state['current_gesture'].startswith('single_') and  # Cek gesture single finger
            len(state['trail_points']) > 1):                    # Cek minimal 2 titik untuk trail
            # Gambar trail dengan efek bayangan
            self._draw_finger_trail(
                frame_square,                       # Frame target untuk drawing
                list(state['trail_points'])         # Konversi deque ke list titik-titik trail
            )

```

Kode 3.70: Drawing Trail Effect

Penjelasan:

- Kondisi Drawing
  - $state['current_gesture'].startswith('single_') : Mengecekapakahgestureadalahjaritunggallen(state['trail_point$
- Parameter Drawing
  - *frame-square*: Frame yang akan digambar trail
  - *list(state['trail-points'])*: List koordinat (x,y) untuk trail

3.2.10.6 Render Efek Visual

```
        if (state['is_stable'] and state['smooth_x'] is not None and
            state['smooth_size'] is not None and "efek-api-unscreen" in self.effects):
            # Use the calculated hand size for effect size
            effect_size = state['smooth_size']


```

Kode 3.71: Kondisi Rendering

Penjelasan: Untuk memastikan posisi tangan stabil, koordinat dan ukuran efek telah dikalkulasi, dan efek api tersedia untuk digunakan

```
        # Determine effect position based on gesture
            if state['current_gesture'].startswith("single_") and state['finger_position']:
            # Single finger - effect at fingertip
                    effect_x, effect_y = state['finger_position']
                    effect_y += -30  # Small offset above fingertip
                else:
                    # Open hand - effect at palm center
                    effect_x = state['smooth_x']
                    effect_y = state['smooth_y'] + self.config['effect_y_offset']


```

Kode 3.72: Penentuan Posisi Efek

- Untuk Jari Tunggal:Efek ditampilkan di ujung jari dan posisi sedikit di atas ujung jari (-30 pixel)
- Untuk Tangan Terbuka:Efek ditampilkan di pusat telapak tangan dan posisi diatur dengan offset dari konfigurasi

3.2.10.7 Animasi Efek

```
        effect_frames = self.effects["efek-api-unscreen"]["frames"]

```

Kode 3.73: Mengambil Frame Efek

Penjelasan:

- Mengakses list frame animasi dari efek api
- "efek-api-unscreen" adalah nama efek yang dimuat
- frames berisi sequence gambar untuk animasi

```

        self._overlay_effect_optimized(
            frame_square,        # Frame target
            effect_frames[state['frame_index']],  # Frame efek saat ini
            effect_x,           # Posisi X efek
            effect_y,           # Posisi Y efek
            effect_size         # Ukuran efek
        )

```

Kode 3.74: Overlay Efek

Penjelasan:

- Memanggil fungsi untuk menggabungkan efek dengan frame
- Menggunakan frame animasi sesuai dengan index saat ini
- Posisi dan ukuran ditentukan oleh parameter sebelumnya

```

        state['frame_index'] = (state['frame_index'] + 1) % len(effect_frames)

```

Kode 3.75: Update Animasi

Penjelasan:

- Increment frame index untuk animasi
- **3.2.10.8 Debug Visualization**

```
            # Untuk Single Finger Gesture
            if state['current_gesture'].startswith("single_") and state['finger_position']:
                # Gambar lingkaran magenta di ujung jari
                cv2.circle(frame_square, state['finger_position'],
                           5,               # Radius lingkaran
                           (255, 0, 255),  # Warna Magenta
                           -1)             # Fill lingkaran

            # Untuk Open Hand Gesture
            else:
                # Gambar lingkaran cyan di pusat telapak
                cv2.circle(frame_square, (state['smooth_x'], state['smooth_y']),
                           8,               # Radius lingkaran
                           (0, 255, 255), # Warna Cyan
                           -1)            # Fill lingkaran

```

Kode 3.76: Visualisasi Posisi

Penjelasan: Visualisasi posisi menggunakan dua jenis marker berbeda untuk memudahkan tracking visual. Untuk gesture jari tunggal, sistem menampilkan lingkaran berwarna magenta dengan radius 5 pixel di ujung jari yang aktif. Sementara untuk gesture tangan terbuka, sistem menampilkan lingkaran cyan yang lebih besar (radius 8 pixel) di pusat telapak tangan. Kedua marker ini membantu pengguna memahami titik tracking yang sedang aktif.

```
           # Format nama gesture
            gesture_name = (state['current_gesture'].replace('single_', '').title()
                            if state['current_gesture'].startswith('single_')
                            else 'Open Hand')

            # Info trail jika single finger
            trail_info = (f" | Trail: {len(state['trail_points'])}"
                          if state['current_gesture'].startswith('single_')
                          else "")

            # Gabungkan informasi
            debug_text = f"Hand {hand_idx + 1}: {gesture_name} | Size={effect_size}{
    trail_info}"

            # Tampilkan teks
            cv2.putText(frame_square,          # Frame target
                        debug_text,            # Teks yang ditampilkan
                        (10, 40 + 30 * hand_idx),  # Posisi (x,y)
                        cv2.FONT_HERSHEY_SIMPLEX,  # Font
                        0.6,                   # Ukuran font
                        (0, 255, 0),          # Warna hijau
                        2)                     # Ketebalan

```

Kode 3.77: Teks Informasi Debug

Penjelasan: Sistem menampilkan informasi status tracking dalam bentuk teks di bagian atas frame. Informasi ini mencakup nomor tangan yang terdeteksi (Hand 1/Hand 2), jenis gesture yang terdeteksi (seperti "Index" atau "Open Hand"), ukuran efek yang sedang diterapkan, dan jumlah titik trail untuk gesture jari tunggal. Teks ditampilkan dengan font Helvetica berwarna hijau untuk visibilitas optimal, dengan posisi vertikal berbeda untuk setiap tangan yang terdeteksi.

```
        return frame_square  # Mengembalikan frame yang sudah dimodifikasi
```

Kode 3.78: Return Hasil

Penjelasan: Setelah semua elemen visual (marker posisi dan teks debug) ditambahkan ke frame, fungsi mengembalikan frame yang telah dimodifikasi (frame-square). Frame ini sudah termasuk semua efek visual seperti efek api, trail effect, marker tracking, dan teks informasi. Frame hasil modifikasi ini siap ditampilkan ke layar untuk memberikan feedback visual kepada pengguna.

3.2.11 Fungsi run()

3.2.11.1 Inisialisasi Kamera

```
    cap = cv2.VideoCapture(0)  # Membuka webcam (index 0)

    # Setting optimal untuk kamera
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)   # Lebar frame
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)  # Tinggi frame
    cap.set(cv2.CAP_PROP_FPS, 30)            # Frame rate

```

Kode 3.79: Inisialisasi Kamera

Penjelasan: Optimasi Kamera: Setting resolusi dan FPS optimal

3.2.11.2 Main Loop dengan Error Handling

```
    try:
    while True:
        # Baca frame dari kamera
        ret, frame = cap.read()
        if not ret:  # Jika gagal membaca frame
            break

        # Proses frame untuk tracking dan efek
        processed_frame = self.process_frame(frame)

        # Tampilkan hasil
        cv2.imshow("Hand Effects with Finger Trail", processed_frame)

        # Check keyboard input
        key = cv2.waitKey(1) & 0xFF
        if key == 27 or key == ord('q'):  # ESC atau 'q'
            break

```

Kode 3.80: Main Loop dengan Error Handling

Penjelasan:

- Error Handling: Menggunakan try-finally untuk cleanup
- Interactive Control: ESC atau 'q' untuk keluar

3.2.11.3 Cleanup Resources

```
    finally:
    # Pastikan resource dibebaskan
```

```
    cap.release()             # Tutup kamera
    cv2.destroyAllWindows()  # Tutup semua window

```

Kode 3.81: Cleanup Resources

3.3 Fungsi Main

```

    def main():
        """Main function to run the hand effect tracker"""
        tracker = HandEffectTracker()

        if not tracker.effects:
            print("No effects loaded. Please ensure 'effects' folder exists with GIF files.")
            return

        print("Starting hand effect tracker with finger trail...")
        print("Press ESC or 'q' to quit")
        print("Gestures:")
        print("- Make a fist to reset hand tracking and trail")
        print("- Show single finger for fingertip fire effect with red trail")
        print("- Open hand for palm fire effect (trail will be cleared)")
        print("- Switch between different single fingers to reset trail")

        tracker.run()

    if __name__ == "__main__":
        main()

```

Kode 3.82: Fungsi Main

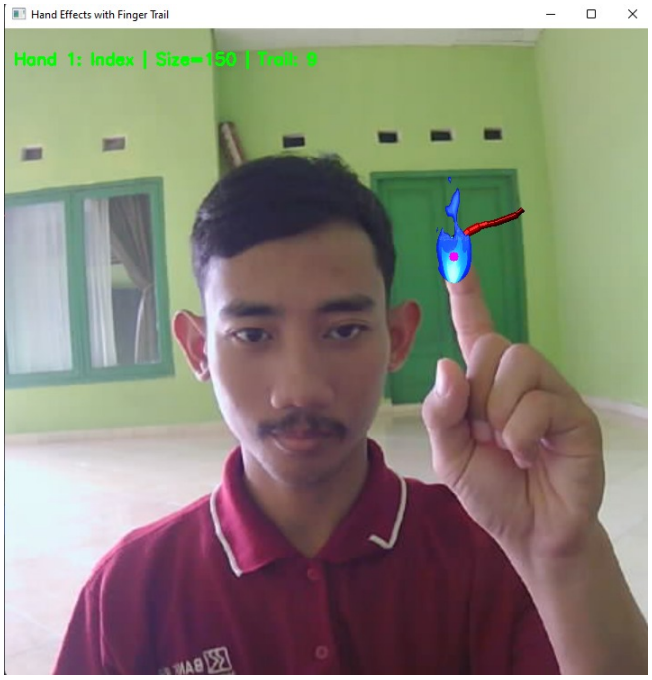
Penjelasan: entry point utama program

- Inisialisasi: Membuat instance HandEffectTracker dan emverifikasi ketersediaan efek GIF
- Instruksi Pengguna:Menampilkan panduan cara penggunaan
- Eksekusi Program: Memanggil tracker.run() untuk memulai loop utama. Menangani kamera dan processing frame. Mendeteksi dan merender efek
- Exit Handler: Program berjalan sampai user menekan ESC/q

4 Hasil dan Pembahasan

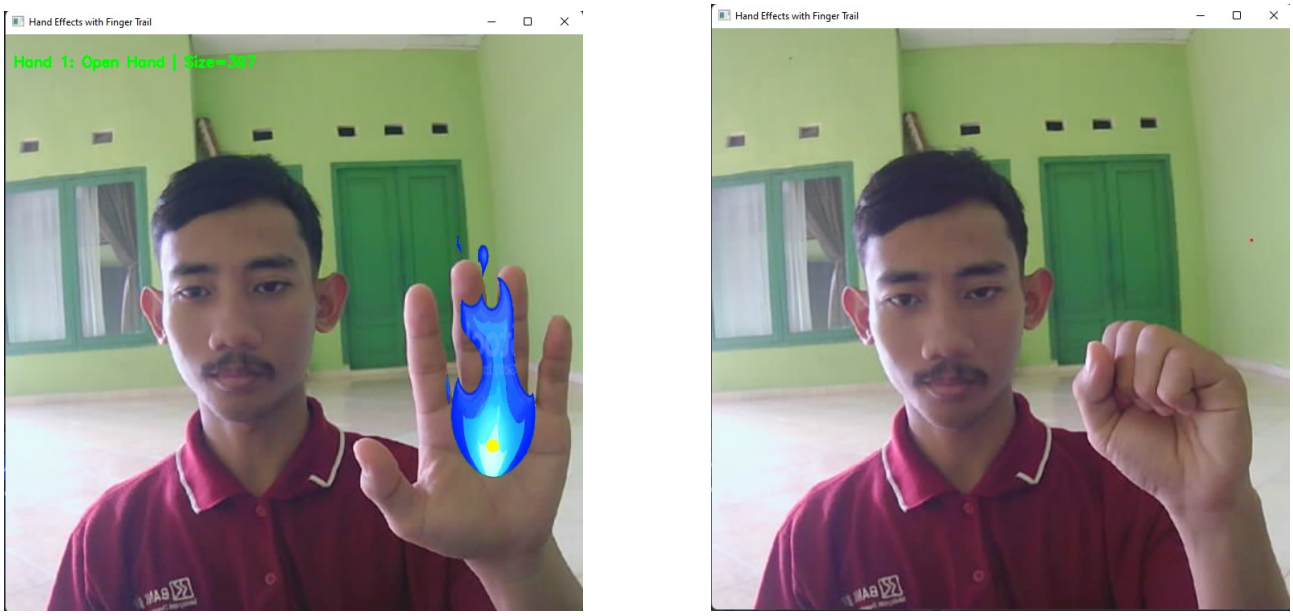
4.1 Hasil

Gambar 4.1 menunjukkan keberhasilan sistem dalam mendeteksi satu jari, yakni jari telunjuk, serta menerapkan efek visual secara presisi pada ujung jari tersebut. Berdasarkan kode program, deteksi ini bekerja dengan menghitung jumlah jari yang terangkat dan mengidentifikasi jenis jari berdasarkan indeks landmark. Efek "Superpower" muncul saat minimal satu jari terdeteksi terbuka, diterapkan tepat pada koordinat jari telunjuk yang terdeteksi aktif. Hal ini menegaskan bahwa sistem mampu mengidentifikasi posisi dan jumlah jari secara akurat serta menampilkan filter sesuai logika yang telah diprogram.



Gambar 4.1: Mendeteksi Satu Jari

Gambar 4.2 menunjukkan keberhasilan sistem dalam mendeteksi dan membedakan gerakan tangan terbuka dan mengepal. Pada kondisi telapak tangan terbuka, sistem berhasil mengenali keberadaan satu tangan dengan lima jari terentang dan menampilkan efek filter "Superpower" secara tepat di area tangan, khususnya mengikuti posisi jari tengah, sesuai logika kode yang mengaktifkan efek saat jumlah jari terdeteksi lebih dari nol. Sebaliknya, saat tangan mengepal, sistem tidak menampilkan efek karena jumlah jari terbuka tidak memenuhi syarat aktivasi, menunjukkan bahwa kode mampu menjalankan klasifikasi gestur secara akurat dan mengeksekusi filter hanya saat kondisi yang sesuai terpenuhi.



(a) Telapak Tangan (b) Mengepal

Gambar 4.2: Deteksi Gerakan Tangan

Gambar 4.3 menunjukkan keberhasilan program dalam mendeteksi dua tangan secara real-time, dengan identifikasi gestur berbeda: tangan kiri dikenali sebagai satu jari menunjuk dan tangan kanan sebagai telapak tangan terbuka. Efek visual berupa nyala api biru ditampilkan secara dinamis di atas jari telunjuk dan telapak tangan, menyesuaikan posisi tangan yang terdeteksi. Informasi tambahan seperti ukuran tangan dan panjang jejak (trail) juga ditampilkan sebagai overlay teks.

Gambar 4.3: Deteksi Satu Jari dan Telapak Tangan

4.2 Pembahasan

Program ini mengimplementasikan sistem deteksi landmark tangan berbasis MediaPipe yang mampu mengidentifikasi 21 titik anatomi kunci pada tangan secara real-time melalui webcam. Sistem menggunakan kombinasi OpenCV untuk pengolahan citra, MediaPipe untuk deteksi landmark, dan imageio untuk memuat efek visual berformat GIF. Arsitektur dibagi menjadi beberapa komponen utama yaitu gesture recognition engine yang menganalisis konfigurasi landmark untuk mengenali gesture (fist, open hand, single finger), position smoothing system untuk stabilisasi koordinat menggunakan exponential smoothing, dan visual effects renderer yang menampilkan animasi GIF pada posisi yang ditentukan berdasarkan gesture yang terdeteksi.

Sistem menyediakan tiga mode interaksi utama berdasarkan gesture yang terdeteksi: mode fist untuk reset tracking, mode open hand yang menempatkan efek visual di tengah telapak tangan, dan mode single finger yang menampilkan efek di ujung jari disertai dengan trail berwarna merah yang mengikuti pergerakan fingertip. Implementasi visual menggunakan teknik alpha blending untuk overlay efek GIF dengan transparansi, dynamic sizing berdasarkan ukuran tangan yang terdeteksi, dan trail rendering dengan shadow effect yang memberikan kesan depth. Project ini juga menerapkan optimisasi performa melalui frame resizing, bounds checking untuk overlay operations, dan caching sistem untuk mencegah recomputation yang tidak perlu, sehingga dapat berjalan smooth pada hardware standar dengan frame rate yang stabil.

5 Cara Instalasi dan Penggunaan

5.1 Cara Instalasi

- Clone Repository ini: git clone https://github.com/han5474ni/Tugas-besar-Multimedia
- Buat dan Aktifkan Virtual Environment: python -m venv venv
  - Windows: ./venv/Scripts/activate
  - macOS/Linux: source venv/bin/activate
- Instal Dependensi: pip install -r requirements.txt

5.2 Cara Menggunakan

- Pastikan webcam terhubung dan berfungsi dengan baik
- Jalankan program: python main.py
- Kontrol Gestur: Telapak Tangan Terbuka, satu Jari
- Kepalan Tangan
- ESC atau Q: untuk keluar

References

[1] E. Martinez-Martin and A. Fernández-Caballero, "Improved human emotion recognition from body and hand pose landmarks on the gemep dataset using machine learning," *Expert Systems with Applications*, vol. 269, Apr. 2025.

[2] R. Husna, K. C. Brata, I. T. Anggraini, N. Funabiki, A. A. Rahmadani, and C. P. Fan, "An investigation of hand gestures for controlling video games in a rehabilitation exergame system," *Computers*, vol. 14, no. 1, Jan. 2025.

[3] M. Nuralin, Y. Daineko, S. Aljawarneh, D. Tsoy, and M. Ipalakova, "The real-time hand and object recognition for virtual interaction," *PeerJ Computer Science*, vol. 10, 2024.