Program Studi Teknik Informatika
Institut Teknologi Sumatera

Nama	:	1. **A Edwin Krisandika Putra (122140003)** 2. **Fathan Andi Kartagama(122140055)** 3. **Rizki Alfariz Ramadhan (122140061)**
Mata Kuliah	:	**Sistem Teknologi Multimedia (IF25-40305)**
Tugas	:	**Final Project: Gestune Musik dari Gerakan Tangan**
Dosen Pengampu	:	**Martin Clinton Tosima Manullang, Ph.D.**
Tanggal	:	December 1, 2025

FINAL PROJECT

Gestune Musik dari Gerakan Tangan

1 Tentang Aplikasi

1.1 Deskripsi Proyek

Proyek ini mengembangkan sistem multimedia yang memungkinkan pengguna menghasilkan musik hanya dengan gerakan tangan secara real-time. Sistem ini memanfaatkan MediaPipe Hand Tracking untuk mendeteksi posisi dan gerakan tangan, yang kemudian digunakan untuk mengontrol dua elemen utama: Arpeggiator dan Drum Machine.

Pada Arpeggiator, tinggi posisi tangan mengatur pitch nada, sementara gerakan pinch gesture mengatur volume suara. Sedangkan Drum Machine menyediakan lima pola ritme drum berbeda yang dapat diaktifkan melalui kombinasi gerakan jari. Selain itu, sistem dilengkapi dengan Audio Reactive Visualizer yang menampilkan efek partikel sesuai intensitas dan ritme musik.

1.2 Anggota Tim

Proyek **Gestune: Musik dari Gerakan Tangan** ini dikembangkan oleh tim yang terdiri dari:

Tabel 1: Anggota Tim Pengembang

Nama Lengkap	NIM	ID GitHub
A Edwin Krisandika Putra	122140003	aloisiusedwin
Fathan Andi Kartagama	122140055	pataanggs
Rizki Alfariz Ramadhan	122140061	Alfariz11

1.3 Fitur Utama

Aplikasi Gestune memiliki beberapa fitur utama yang memungkinkan pengguna untuk membuat musik dengan gerakan tangan:

1.3.1 Arpeggiator (Tangan Kiri)

- **Kontrol Pitch**: Tinggi posisi tangan kiri mengatur pitch nada yang dihasilkan. Semakin tinggi posisi tangan, semakin tinggi pitch nada yang dihasilkan.
- **Kontrol Volume**: Gesture pinch (mendekatkan ibu jari dan telunjuk) mengatur volume suara. Semakin kecil jarak antara ibu jari dan telunjuk, semakin kecil volume suara, begitu juga sebaliknya.

1.3.2 Drum Machine (Tangan Kanan)

Drum Machine menyediakan lima pola ritme drum yang berbeda, yang dapat diaktifkan dengan kombinasi jari yang terangkat:

- **Pattern 1 (Basic 4/4)**: Tidak ada jari yang terangkat atau hanya telunjuk
- **Pattern 2 (With clap)**: Telunjuk + tengah terangkat
- **Pattern 3 (Syncopated)**: Telunjuk + tengah + manis terangkat
- **Pattern 4 (Break beat)**: Semua jari kecuali ibu jari terangkat
- **Pattern 5 (Minimal)**: Semua jari terangkat

1.3.3 Audio Reactive Visualizer

Sistem dilengkapi dengan visualizer yang menampilkan efek partikel secara real-time yang bereaksi terhadap intensitas dan ritme musik yang dihasilkan. Visualizer ini membuat pengalaman pengguna lebih interaktif dan menarik.

2 Instalasi

2.1 Prasyarat

Sebelum melakukan instalasi, pastikan sistem Anda memenuhi prasyarat berikut:

- **Python**: Versi 3.8 atau lebih tinggi
- **Webcam**: Diperlukan untuk hand tracking
- **Sistem Operasi**: Windows, macOS, atau Linux

2.2 Langkah Instalasi

Ikuti langkah-langkah berikut untuk menginstall aplikasi Gestune:

2.2.1 Clone Repository

Langkah pertama adalah meng-clone repository dari GitHub:

```
git clone https://github.com/Alfariz11/Gestune-Musik-dari-gerakan-tangan.git
cd Gestune-Musik-dari-gerakan-tangan
```

2.2.2 Membuat Virtual Environment

Sangat disarankan menggunakan UV - sebuah virtual environment manager yang lebih cepat dan modern.

Instalasi UV (Disarankan) Langkah pertama adalah menginstall UV:

Untuk Windows:

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Untuk macOS/Linux:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Setelah UV terinstall, buat dan aktifkan virtual environment:

```
# Buat virtual environment
uv venv

# Aktifkan virtual environment
# Windows:
.venv\Scripts\activate

# macOS/Linux:
source .venv/bin/activate
```

Alternatif: Python venv (Traditional) Jika tidak ingin menggunakan UV, Anda dapat menggunakan venv bawaan Python:

Untuk Windows:

```
python -m venv venv
venv\Scripts\activate
```

Untuk macOS/Linux:

```
python3 -m venv venv
source venv/bin/activate
```

Catatan: UV jauh lebih cepat dalam instalasi dependencies dan manajemen environment. Dokumentasi lengkap UV tersedia di https://docs.astral.sh/uv/

2.2.3 Install Dependencies

Setelah virtual environment aktif, install semua dependencies yang diperlukan:

```
pip install -r requirements.txt
```

2.3 Verifikasi Instalasi

Untuk memverifikasi bahwa audio berfungsi dengan baik, Anda dapat menjalankan script testing:

```
python test_drums.py
```

Jika Anda mendengar suara drum, maka instalasi telah berhasil.

3 Cara Penggunaan

3.1 Menjalankan Aplikasi

Untuk menjalankan aplikasi Gestune, ikuti langkah-langkah berikut:

3.1.1 Jalankan Aplikasi Utama

Jalankan file `main.py` dengan perintah:

```
python main.py
```

3.1.2 Pengaturan Webcam

- Pastikan webcam Anda aktif dan terhubung dengan baik
- Posisikan diri Anda di depan webcam dengan pencahayaan yang cukup
- Pastikan kedua tangan terlihat jelas dalam frame kamera

3.2 Kontrol Aplikasi

3.2.1 Tangan Kiri - Arpeggiator

Tangan kiri digunakan untuk mengontrol Arpeggiator dengan dua cara:

1. **Mengatur Pitch Nada**
   - Naikkan tangan untuk mendapatkan nada yang lebih tinggi
   - Turunkan tangan untuk mendapatkan nada yang lebih rendah

- Pitch nada akan berubah secara gradual mengikuti posisi vertikal tangan

2. **Mengatur Volume**
   - Lakukan gesture pinch dengan mendekatkan ibu jari dan telunjuk
   - Semakin kecil jarak antara ibu jari dan telunjuk, semakin kecil volume
   - Semakin besar jarak antara ibu jari dan telunjuk, semakin besar volume

3.2.2 Tangan Kanan - Drum Machine

Tangan kanan digunakan untuk mengontrol Drum Machine dengan lima pola ritme berbeda:

Tabel 2: Pola Drum Machine

Gesture	Pattern	Deskripsi
Tidak ada jari / Telunjuk	Pattern 1	Basic 4/4
Telunjuk + Tengah	Pattern 2	With clap
Telunjuk + Tengah + Manis	Pattern 3	Syncopated
Semua kecuali ibu jari	Pattern 4	Break beat
Semua jari	Pattern 5	Minimal

3.2.3 Kontrol Keyboard

Selain kontrol dengan tangan, terdapat juga kontrol keyboard:

- **Q atau ESC**: Keluar dari aplikasi

3.3 Tips Penggunaan

Untuk mendapatkan pengalaman terbaik saat menggunakan aplikasi Gestune, perhatikan tips berikut:

1. **Pencahayaan**: Pastikan ruangan memiliki pencahayaan yang cukup agar hand tracking dapat mendeteksi tangan dengan optimal
2. **Jarak dengan Webcam**: Jaga jarak yang nyaman dengan webcam (sekitar 50-100 cm)
3. **Posisi Tangan**: Pastikan kedua tangan terlihat jelas dalam frame kamera dan tidak tertutup oleh objek lain
4. **Gerakan**: Lakukan gerakan dengan smooth dan tidak terlalu cepat agar sistem dapat mendeteksi dengan akurat
5. **Visualizer**: Perhatikan Audio Reactive Visualizer yang menampilkan efek partikel sesuai intensitas dan ritme musik yang sedang dimainkan

3.4 Troubleshooting

Jika mengalami masalah saat menggunakan aplikasi, coba solusi berikut:

- **Tangan tidak terdeteksi**: Perbaiki pencahayaan atau sesuaikan jarak dengan webcam
- **Audio tidak keluar**: Periksa pengaturan audio sistem Anda atau jalankan `test_drums.py` untuk verifikasi
- **Aplikasi lambat**: Tutup aplikasi lain yang berjalan untuk mengalokasikan lebih banyak resource

4 Penjelasan Code

Bagian ini menjelaskan breakdown dari code-code penting dalam aplikasi Gestune beserta fungsinya.

4.1 Arsitektur Aplikasi

Aplikasi Gestune terdiri dari beberapa modul utama:

- **main.py**: Entry point aplikasi dan manajemen lifecycle
- **hand_tracker.py**: Deteksi dan tracking gerakan tangan menggunakan MediaPipe
- **arpeggiator.py**: Sintesis suara dan kontrol pitch untuk tangan kiri
- **drum_machine.py**: Pattern sequencer dan drum sounds untuk tangan kanan
- **gesture_processor.py**: Bridge antara hand tracking dan music generation
- **gestune_ui.py**: User interface menggunakan PyQt6

4.2 Main Application (main.py)

File `main.py` berfungsi sebagai entry point aplikasi dan mengelola lifecycle keseluruhan sistem.

4.2.1 Inisialisasi Aplikasi

```
class GestuneApplication:
    def initialize(self) -> bool:
        # Create Qt application
        self.app = QApplication(sys.argv)
        self.app.setApplicationName("Gestune")

        # Create main window
        self.window = GestuneUI()

        # Initialize gesture processor
        self.processor = GestureProcessor()

        # Connect signals
        self._connect_signals()

        return True
```

Kode 1: Inisialisasi komponen utama

Penjelasan:

- Membuat Qt application dengan `QApplication` untuk GUI
- Inisialisasi `GestuneUI` sebagai window utama
- Membuat `GestureProcessor` yang akan running di background thread
- Menghubungkan signals antara processor dan UI

4.2.2 Signal Connections

```
def _connect_signals(self):
    # Processor -> UI signals
    self.processor.frame_processed.connect(
        self.window.update_camera_feed
    )

    self.processor.drum_hit.connect(
        self._on_drum_hit
    )

    # UI -> Processor signals
    self.window.bpm_changed.connect(
        self.processor.set_bpm
    )
```

Kode 2: Koneksi signals untuk komunikasi antar komponen

Penjelasan:

- Menggunakan PyQt6 signals untuk komunikasi thread-safe
- Processor mengirim frame yang sudah diproses ke UI
- UI mengirim perubahan BPM ke processor
- Pattern observer untuk decoupling komponen

4.3 Hand Tracking (hand_tracker.py)

Modul ini bertanggung jawab untuk mendeteksi dan melacak gerakan tangan menggunakan MediaPipe.

4.3.1 Inisialisasi MediaPipe

```
class HandTracker:
    def __init__(self, enable_roi: bool = True):
        self.mp_hands = mp.solutions.hands

        self.hands = self.mp_hands.Hands(
            model_complexity=1,
            min_detection_confidence=0.7,
            min_tracking_confidence=0.7,
            max_num_hands=2
        )

        # Define ROI zones
        self.roi_zones = {
            'Left': ROIZone(
                x_min=0.0, x_max=0.5,
                color=(100, 200, 255),  # Blue
                name='ARPEGGIATOR ZONE'
            ),
            'Right': ROIZone(
                x_min=0.5, x_max=1.0,
                color=(255, 100, 150),  # Pink
                name='DRUM ZONE'
            )
        }
```

Kode 3: Setup MediaPipe Hands

Penjelasan:

- `model_complexity=1`: Menggunakan full model untuk akurasi maksimal
- `min_detection_confidence=0.7`: Hanya hand dengan confidence ŏ03e 70% yang dideteksi
- `max_num_hands=2`: Tracking maksimal 2 tangan (kiri dan kanan)
- ROI zones membagi layar menjadi dua: kiri untuk arpeggiator, kanan untuk drums

4.3.2 Frame Processing

```python
def process_frame(self, frame: np.ndarray) -> Dict:
    # Convert BGR to RGB
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process with MediaPipe
    self.results = self.hands.process(rgb_frame)

    # Process detected hands
    if self.results.multi_hand_landmarks:
        for hand_landmarks, handedness in zip(
            self.results.multi_hand_landmarks,
            self.results.multi_handedness
        ):
            hand_label = handedness.classification[0].label

            # Check if hand is in ROI
            in_roi = self._is_hand_in_roi(
                hand_landmarks, hand_label
            )

            # Store hand data
            self.hand_data[hand_label] = {
                'landmarks': hand_landmarks,
                'in_roi': in_roi,
                'center_x': center_x,
                'center_y': center_y
            }

    return self.hand_data
```

Kode 4: Memproses setiap frame untuk deteksi tangan

Penjelasan:

- Convert frame dari BGR (OpenCV) ke RGB (MediaPipe)
- MediaPipe mendeteksi landmarks (21 titik per tangan)
- Classify hand menjadi Left atau Right
- Check apakah tangan berada di ROI zone yang sesuai
- Return dictionary berisi data semua tangan yang terdeteksi

4.3.3 Gesture Recognition

```python
def get_fingers_extended(self, hand_label: str) -> List[bool]:
    landmarks = self.hand_data[hand_label]['landmarks'].landmark

    # Thumb (horizontal comparison)
    if hand_label == 'Right':
        thumb_extended = landmarks[4].x < landmarks[3].x
    else:
        thumb_extended = landmarks[4].x > landmarks[3].x

    # Other fingers (vertical comparison)
    index_extended = landmarks[8].y < landmarks[6].y
    middle_extended = landmarks[12].y < landmarks[10].y
    ring_extended = landmarks[16].y < landmarks[14].y
    pinky_extended = landmarks[20].y < landmarks[18].y

    return [thumb_extended, index_extended,
            middle_extended, ring_extended, pinky_extended]
```

Kode 5: Deteksi jari yang terangkat

Penjelasan:

- Untuk **ibu jari**: membandingkan posisi x (horizontal) karena ibu jari membuka ke samping
- Untuk **jari lainnya**: membandingkan posisi y (vertical) - tip harus di atas PIP joint
- Return list of booleans: [thumb, index, middle, ring, pinky]
- Digunakan untuk kontrol drum machine (setiap jari = drum berbeda)

4.3.4 Pinch Gesture Detection

```python
def get_pinch_distance(self, hand_label: str) -> float:
    thumb = self.hand_data[hand_label]['thumb_tip']
    index = self.hand_data[hand_label]['index_tip']

    # Calculate 3D Euclidean distance
    distance = np.sqrt(
        (thumb.x - index.x)**2 +
        (thumb.y - index.y)**2 +
        (thumb.z - index.z)**2
    )

    # Apply smoothing
    return self._smooth_value(f'{hand_label}_pinch', distance)
```

Kode 6: Mendeteksi jarak pinch untuk volume control

Penjelasan:

- Menghitung jarak 3D antara ujung ibu jari dan telunjuk
- Menggunakan formula Euclidean distance dengan komponen x, y, dan z
- Smoothing diterapkan untuk mengurangi jitter/noise
- Jarak ini dimap ke volume (pinch rapat = volume kecil)

4.4 Arpeggiator (arpeggiator.py)

Modul untuk sintesis suara dan kontrol pitch menggunakan tangan kiri.

4.4.1 Waveform Generation

```python
def generate_waveform(self, frequency: float, t: np.ndarray):
    if self.waveform == 'rich_sine':
        # Rich sine with harmonics
        base = np.sin(2 * np.pi * frequency * t)
        harmonic1 = 0.3 * np.sin(4 * np.pi * frequency * t)
        harmonic2 = 0.15 * np.sin(6 * np.pi * frequency * t)
        wave = base + harmonic1 + harmonic2

    elif self.waveform == 'square':
        wave = np.sign(np.sin(2 * np.pi * frequency * t))

    elif self.waveform == 'saw':
        wave = 2 * (t * frequency - np.floor(t * frequency + 0.5))

    return wave / np.max(np.abs(wave))
```

Kode 7: Menghasilkan waveform audio

Penjelasan:

- **Rich Sine**: Sine wave utama + harmonics untuk suara yang lebih "warm"
- **Square**: Waveform kotak untuk suara yang lebih "bright" dan "harsh"
- **Sawtooth**: Waveform gigi gergaji untuk suara yang kaya harmonik
- Normalisasi di akhir untuk mencegah clipping

4.4.2 MIDI to Frequency Conversion

```python
def midi_to_freq(self, midi_note: int) -> float:
    # A4 (MIDI 69) = 440 Hz
    # Formula: f = 440 * 2^((n-69)/12)
    return 440.0 * (2.0 ** ((midi_note - 69) / 12.0))
```

Kode 8: Konversi MIDI note ke frequency

Penjelasan:

- Menggunakan equal temperament tuning system
- A4 (MIDI note 69) adalah reference frequency 440 Hz
- Setiap semitone adalah faktor $2^{1/12}$ dari frequency sebelumnya
- Contoh: C4 (MIDI 60) = 261.63 Hz

4.5 Drum Machine (drum_machine.py)

Modul untuk drum pattern sequencer yang dikontrol oleh tangan kanan.

4.5.1 Pattern Initialization

```python
def _initialize_patterns(self):
    self.patterns = {
        1: {  # Basic 4/4
            'kick':  {0: 1.0, 4: 0.8, 8: 1.0, 12: 0.8},
            'snare': {4: 1.0, 12: 1.0},
            'hihat': {i: 0.6 for i in range(16) if i % 2 == 0}
```

```
        },
        2: {  # With clap
            'kick':  {0: 1.0, 6: 0.7, 8: 1.0, 14: 0.7},
            'snare': {4: 1.0, 12: 1.0},
            'hihat': {i: 0.5 if i % 2 == 0 else 0.3
                      for i in range(16)}
        }
        # ... more patterns
    }
```

Kode 9: Definisi drum patterns

Penjelasan:

- Setiap pattern adalah dictionary dengan key = nama drum
- Value adalah dictionary: step_number: velocity
- Step 0-15 = 16th notes dalam 1 bar (4/4 time)
- Velocity 0.0-1.0 mengontrol loudness dari hit

4.5.2 Update Loop with Swing

```
def update(self, fingers_extended: List[bool], current_time: float):
    # Determine pattern from finger count
    extended_count = sum(fingers_extended)
    pattern_index = min(extended_count + 1, 5)

    # Calculate time since last step
    step_duration = self._get_step_duration(self.current_step)
    time_since_step = current_time - self.last_step_time

    # Advance to next step if enough time passed
    if time_since_step >= step_duration:
        self.current_step = (self.current_step + 1) % 16
        self.last_step_time = current_time

        # Play drums for this step
        drums_hit = []
        pattern = self.patterns[pattern_index]
        for drum_name, steps in pattern.items():
            if self.current_step in steps:
                velocity = steps[self.current_step]
                self._play_drum(drum_name, velocity, current_time)
                drums_hit.append(drum_name)

        return {'step': self.current_step, 'drums': drums_hit}
```

Kode 10: Sequencer update dengan swing timing

Penjelasan:

- Pattern dipilih berdasarkan jumlah jari yang terangkat (0-5 jari = pattern 1-5)
- Swing diterapkan dengan mengubah durasi step genap/ganjil
- Sequencer advance ke step berikutnya setelah durasi terlewati
- Semua drum yang ada di step saat ini dimainkan dengan velocity-nya

4.6 Gesture Processor (gesture_processor.py)

Modul yang menghubungkan hand tracking dengan music generation, berjalan di background thread.

4.6.1 Main Processing Loop

```
def run(self):
    while self.running:
        ret, frame = self.cap.read()
        if not ret:
            continue

        # Mirror frame
        frame = cv2.flip(frame, 1)

        # Process frame
        self._process_frame(frame)

        # Calculate FPS
        self._update_fps()

        # Emit processed frame
        self.frame_processed.emit(frame)
```

Kode 11: Loop utama pemrosesan gesture

Penjelasan:

- Loop berjalan di QThread terpisah (tidak blocking UI)
- Frame di-mirror untuk user experience yang lebih natural
- `_process_frame` melakukan hand tracking dan music generation
- Frame yang sudah diproses dikirim ke UI via signal

4.6.2 Arpeggiator Control

```
def _process_arpeg giator(self, hand_info: Dict, frame_shape: Tuple):
    # Get hand height (for pitch)
    hand_height = self.hand_tracker.get_hand_height('Left')

    # Get pinch distance (for volume)
    pinch_distance = self.hand_tracker.get_pinch_distance('Left')

    # Map pinch to volume (0.03-0.15 range -> 0-1)
    volume = 1.0 - np.clip(
        (pinch_distance - 0.03) / 0.12, 0.0, 1.0
    )

    # Update arpeggiator
    result = self.arpeggiator.update(
        hand_height=hand_height,
        pinch_distance=pinch_distance,
        current_time=time.time(),
        bpm=self.current_bpm
    )

    if result:
        self.note_played.emit(result['note'], result['volume'])
```

Kode 12: Kontrol arpeggiator dengan tangan kiri

Penjelasan:

- **Hand height** (0-1) dimap ke MIDI notes untuk pitch control
- **Pinch distance** dimap ke volume (rapat = 0, jauh = 1)
- Arpeggiator menghasilkan notes dengan timing sesuai BPM
- Signal `note_played` dikirim ke UI untuk visualization

4.6.3 Drum Control

```python
def _process_drums(self, hand_info: Dict, frame_shape: Tuple):
    # Get finger extension status
    fingers = self.hand_tracker.get_fingers_extended('Right')

    # Check for fist (pattern change)
    is_fist = self.hand_tracker.is_fist('Right')

    # Update drum machine
    result = self.drum_machine.update(
        fingers_extended=fingers,
        current_time=time.time(),
        is_fist=is_fist
    )

    # Emit drum hits
    if 'drums' in result:
        for drum_name in result['drums']:
            velocity = 1.0  # Could be extracted from pattern
            self.drum_hit.emit(drum_name, velocity)
```

Kode 13: Kontrol drum machine dengan tangan kanan

Penjelasan:

- Deteksi jari mana saja yang terangkat (5 booleans)
- Jumlah jari terangkat menentukan pattern mana yang aktif
- Fist gesture digunakan untuk cycle ke pattern berikutnya
- Signal `drum_hit` dikirim untuk setiap drum yang dimainkan

Lampiran

A. Logbook Mingguan

Tabel 3: Logbook Perkembangan Proyek

Tanggal	Kegiatan	Hasil / Progress
10/28/2025	Pembuatan Repositori GitHub Tugas Besar	Repositori GitHub tugas besar berhasil dibuat dengan struktur awal proyek
11/2/2025	Implementasi Komponen Utama & Integrasi Aplikasi	Hand tracker dengan MediaPipe, Arpeggiator (kontrol pitch & volume), Drum Machine (5 pola ritme), Audio Reactive Visualizer, dan integrasi semua komponen di main application. Perbaikan audio system dengan real audio samples, optimisasi code, Custom BPM feature
11/9/2025	Integrasi PyGame pada proyek dan penambahan fitur	Integrasi Proyek (Visualizer) dari CV2 ke PyGame, Penambahan pattern beat baru dan aset drum terbaru, Perbaikan visualisasi
11/14/2025	UI baru menggunakan PyQT6, fixing bugs	UI baru menggunakan PyQT6, fixing bugs
11/28/2025	Perubahan asset dan pattern pada drum	Perubahan asset dan pattern pada drum machine berhasil
11/30/2025	Remake Pinch BPM Controller	Menerapkan Pinch BPM pada UI baru dan men-sinkronkan perubahan BPM dengan UI & slider

B. Informasi Repository

Repository GitHub

Repository proyek ini dapat diakses di:

- **URL**: https://github.com/Alfariz11/Gestune-Musik-dari-gerakan-tangan
- **Branch Utama**: main

C. Teknologi yang Digunakan

Tabel 4: Teknologi dan Library

Komponen	Teknologi
Hand Tracking	MediaPipe
User Interface	PyQt6
Graphics & Visualizer	PyGame
Audio Processing	PyAudio / Sounddevice
Programming Language	Python 3.8+