



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 2, 2025



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 memungkinkan pengguna untuk memainkan instrumen drum secara individual dengan mengangkat jari yang sesuai. 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 Fungsi Gesture Tangan Kanan

Tangan kanan berperan sebagai antarmuka utama untuk mengendalikan modul drum pada sistem ini. Gesture yang dibaca dari posisi jari digunakan untuk memicu bunyi instrumen perkusi, mengaktifkan pola ritmis, serta melakukan pergantian pola secara real-time. Secara umum, fitur-fitur tangan kanan yang digunakan dalam sistem adalah sebagai berikut:

- Pemicu bunyi instrumen drum:** Setiap jari tangan kanan merepresentasikan satu instrumen drum sehingga perubahan posisi jari dapat langsung menghasilkan bunyi *kick*, *snare*, *hihat*, *hightom*, atau *crash cymbal*.
- Aktivasi instrumen dalam sequencer:** Ketika jari berada pada posisi terbuka, instrumen terkait akan mengikuti pola ritmis yang telah diprogram pada *drum sequencer*.
- Pemicu manual (manual trigger):** Perubahan gerakan dari jari tertutup menjadi terbuka digunakan sebagai pemicu pukulan drum secara langsung untuk memberikan variasi ritmis yang lebih ekspresif.

- **Pergantian pola ritme:** Gesture menggenggam (*fist gesture*) digunakan untuk mengganti pola ritme drum secara otomatis tanpa menggunakan tombol fisik atau antarmuka tambahan.

1.3.3 Drum Machine (Tangan Kanan)

Drum Machine menyediakan lima pola ritme drum yang berbeda, yang dapat diganti dengan membuat gestur menggenggam tangan kanan. :

- **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

1.3.4 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:

```
1 git clone https://github.com/Alfariz11/Gestune-Musik-dari-gerakan-tangan.git
2 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:

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

Untuk macOS/Linux:

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

Setelah UV terinstall, buat dan aktifkan virtual environment:

```
1 # Buat virtual environment
2 uv venv
3
4 # Aktifkan virtual environment
5 # Windows:
6 .venv\Scripts\activate
7
8 # macOS/Linux:
9 source .venv/bin/activate
```

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

Untuk Windows:

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

Untuk macOS/Linux:

```
1 python3 -m venv venv
2 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:

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

2.3 Verifikasi Instalasi

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

```
1 python test_audio_engine.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:

```
1 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 mengaktifkan instrumen drum berdasarkan jari yang diangkat:

Tabel 2: Mapping Jari Drum Machine

Jari	Instrumen	Deskripsi
Ibu Jari (Thumb)	Kick	Bass drum
Telunjuk (Index)	Snare	Snare drum
Tengah (Middle)	Hi-hat	Closed hi-hat
Manis (Ring)	Clap	Hand clap
Kelingking (Pinky)	Clap	Hand clap (alternatif)

Setiap jari yang diangkat akan mengaktifkan track instrumen tersebut dalam loop sequencer yang sedang berjalan.

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
- **audio_engine.py:** Engine audio terpadu untuk sintesis suara dan drum sequencing
- **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

```
1 class GestuneApplication:
2     def initialize(self) -> bool:
3         # Create Qt application
4         self.app = QApplication(sys.argv)
5         self.app.setApplicationName("Gestune")
6
7         # Create main window
8         self.window = GestuneUI()
9
10        # Initialize gesture processor
11        self.processor = GestureProcessor()
12
13        # Connect signals
14        self._connect_signals()
15
16    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

```
1 def _connect_signals(self):
2     # Processor -> UI signals
3     self.processor.frame_processed.connect(
4         self.window.update_camera_feed
5     )
6
7     self.processor.drum_hit.connect(
8         self._on_drum_hit
9     )
10
11    # UI -> Processor signals
12    self.window.bpm_changed.connect(
13        self.processor.set_bpm
14    )
```

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

```
1 class HandTracker:
2     def __init__(self, enable_roi: bool = True):
3         self.mp_hands = mp.solutions.hands
4
5         self.hands = self.mp_hands.Hands(
6             model_complexity=1,
7             min_detection_confidence=0.7,
8             min_tracking_confidence=0.7,
9             max_num_hands=2
10        )
11
12     # Define ROI zones
13     self.roi_zones = {
14         'Left': ROIZone(
15             x_min=0.0, x_max=0.5,
16             color=(100, 200, 255), # Blue
17             name='ARPEGGIATOR ZONE'
18         ),
19         'Right': ROIZone(
20             x_min=0.5, x_max=1.0,
21             color=(255, 100, 150), # Pink
22             name='DRUM ZONE'
23         )
24     }
```

Kode 3: Setup MediaPipe Hands

Penjelasan:

- `model_complexity=1`: Menggunakan full model untuk akurasi maksimal
- `min_detection_confidence=0.7`: Hanya hand dengan confidence 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

```
1 def process_frame(self, frame: np.ndarray) -> Dict:
2     # Convert BGR to RGB
3     rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
4
5     # Process with MediaPipe
6     self.results = self.hands.process(rgb_frame)
7
8     # Process detected hands
9     if self.results.multi_hand_landmarks:
10         for hand_landmarks, handedness in zip(
11             self.results.multi_hand_landmarks,
12             self.results.multi_handedness
13         ):
14             hand_label = handedness.classification[0].label
15
16             # Check if hand is in ROI
17             in_roi = self._is_hand_in_roi(
18                 hand_landmarks, hand_label
19             )
20
21             # Store hand data
```

```
22     self.hand_data[hand_label] = {  
23         'landmarks': hand_landmarks,  
24         'in_roi': in_roi,  
25         'center_x': center_x,  
26         'center_y': center_y  
27     }  
28  
29     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

```
1 def get_fingers_extended(self, hand_label: str) -> List[bool]:  
2     landmarks = self.hand_data[hand_label]['landmarks'].landmark  
3  
4     # Thumb (horizontal comparison)  
5     if hand_label == 'Right':  
6         thumb_extended = landmarks[4].x < landmarks[3].x  
7     else:  
8         thumb_extended = landmarks[4].x > landmarks[3].x  
9  
10    # Other fingers (vertical comparison)  
11    index_extended = landmarks[8].y < landmarks[6].y  
12    middle_extended = landmarks[12].y < landmarks[10].y  
13    ring_extended = landmarks[16].y < landmarks[14].y  
14    pinky_extended = landmarks[20].y < landmarks[18].y  
15  
16    return [thumb_extended, index_extended,  
17             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

```
1 def get_pinch_distance(self, hand_label: str) -> float:  
2     thumb = self.hand_data[hand_label]['thumb_tip']  
3     index = self.hand_data[hand_label]['index_tip']  
4
```

```
5 # Calculate 3D Euclidean distance
6 distance = np.sqrt(
7     (thumb.x - index.x)**2 +
8     (thumb.y - index.y)**2 +
9     (thumb.z - index.z)**2
10 )
11
12 # Apply smoothing
13 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 Audio Engine (audio_engine.py)

Modul ini menggantikan `arpeggiator.py` dan `drum_machine.py` menjadi satu engine audio terpadu yang menangani sintesis suara dan drum sequencing secara real-time.

4.4.1 Initialization & Scheduler

```
1 class AudioEngine:
2     def __init__(self, assets_path=None):
3         # Initialize mixer
4         pygame.mixer.init(frequency=44100, size=-16, channels=2, buffer=512)
5
6         # Load Drums
7         self.drums = {}
8         # ... load wav files ...
9
10        # Synth State
11        self.scale = self._generate_scale_frequencies()
12        self.synth_sounds = self._precompute_synth_sounds()
13
14        # Scheduler State
15        self.bpm = 100
16        self.step_duration = (60 / self.bpm) / 4 # 16th notes
17
18        # Start Scheduler Thread
19        self.scheduler_thread = threading.Thread(
20            target=self._scheduler_loop,
21            daemon=True
22        )
23        self.scheduler_thread.start()
```

Kode 7: Inisialisasi Audio Engine dan Scheduler

Penjelasan:

- Menggunakan satu thread terpisah (`scheduler_loop`) untuk timing yang presisi
- Menggabungkan loading asset drum dan pre-computation suara synth
- Mengatur BPM dan durasi step untuk sinkronisasi

4.4.2 Synthesis & Precomputation

```
1 def _precompute_synth_sounds(self):
2     sounds = []
3     for freq in self.scale:
4         # Generate Sine Wave
5         t = np.linspace(0, duration, int(sample_rate * duration), False)
6         wave = np.sin(2 * np.pi * freq * t)
7
8         # Simple Envelope (Attack/Decay)
9         envelope = np.concatenate([
10             np.linspace(0, 1, int(sample_rate * 0.01)), # Attack
11             np.linspace(1, 0, int(sample_rate * (duration - 0.01))) # Decay
12         ])
13         wave = wave * envelope
14
15         # Convert to Sound object
16         sound = pygame.sndarray.make_sound(stereo)
17         sounds.append(sound)
18
19 return sounds
```

Kode 8: Pre-compute suara synthesizer

Penjelasan:

- Suara disintesis di awal (pre-compute) untuk mengurangi latency saat runtime
- Menggunakan gelombang sinus sederhana dengan envelope attack/decay
- Disimpan dalam list `synth_sounds` yang dipetakan ke index nada

4.4.3 Sequencer Loop

```
1 def _play_step(self, step):
2     with self.lock:
3         # Play Drums
4         for drum_name in self.active_drums:
5             if self.drum_pattern[drum_name][step]:
6                 if drum_name in self.drums:
7                     self.drums[drum_name].play()
8
9         # Play Arpeggios
10        for hand_idx, arp_data in self.active_arpeggios.items():
11            root_idx = arp_data['note_index']
12            # Simple Arpeggio Pattern
13            offsets = [0, 2, 4, 2]
14            offset = offsets[step % 4]
15
16            note_idx = root_idx + offset
17            if 0 <= note_idx < len(self.synth_sounds):
18                sound = self.synth_sounds[note_idx]
19                sound.play()
```

Kode 9: Loop utama sequencer

Penjelasan:

- `_play_step` dipanggil setiap 16th note
- Mengecek `active_drums` (drum yang sedang diaktifkan user)
- Memainkan arpeggio berdasarkan root note yang dipilih user

4.5 Gesture Processor (gesture_processor.py)

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

4.5.1 Main Processing Loop

```
1 def run(self):
2     while self.running:
3         ret, frame = self.cap.read()
4         if not ret:
5             continue
6
7         # Mirror frame
8         frame = cv2.flip(frame, 1)
9
10        # Process frame
11        self._process_frame(frame)
12
13        # Calculate FPS
14        self._update_fps()
15
16        # Emit processed frame
17        self.frame_processed.emit(frame)
```

Kode 10: 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

4.5.2 Arpeggiator Control

```
1 def _process_arpeggiator(self, hand_info: Dict, frame_shape: Tuple):
2     # Get hand height (for pitch)
3     hand_height = 1.0 - hand_info['wrist_y']
4
5     # Get pinch distance (for volume)
6     pinch_distance = self.tracker.get_pinch_distance(HandSide.LEFT.value)
7
8     # Map height to note index
9     if self.audio:
10         num_notes = len(self.audio.scale)
11         note_index = int(hand_height * num_notes)
12         note_index = max(0, min(note_index, num_notes - 1))
13
14         # Map pinch to volume
15         volume = max(0.0, min(1.0, (1.0 - pinch_distance)))
16
17         # Update arpeggiator in AudioEngine
18         self.audio.update_arpeggio("left_hand", note_index, volume)
```

Kode 11: Kontrol arpeggiator dengan tangan kiri

Penjelasan:

- **Hand height** dimap ke index nada dalam skala pentatonik

- **Pinch distance** dimap ke volume
- **AudioEngine** menangani playing dan timing dari arpeggio

4.5.3 Drum Control

```
1 def _process_drums(self, hand_info: Dict, frame_shape: Tuple):
2     # Get finger extension status
3     fingers = self.tracker.get_fingers_extended(HandSide.RIGHT.value)
4
5     # Map fingers to drums
6     active_drums = set()
7     drum_map = {0: 'kick', 1: 'snare', 2: 'hihat', 3: 'clap', 4: 'clap'}
8
9     for i, is_extended in enumerate(fingers):
10         if is_extended and i in drum_map:
11             active_drums.add(drum_map[i])
12
13     # Update active drums in AudioEngine
14     if self.audio:
15         self.audio.update_drums(active_drums)
```

Kode 12: Kontrol drum machine dengan tangan kanan

Penjelasan:

- Setiap jari dipetakan ke instrumen drum tertentu
- Jika jari terangkat, instrumen tersebut dimasukkan ke **active_drums**
- **AudioEngine** akan memainkan instrumen yang aktif sesuai pattern sequencer

Lampiran

A. Logbook Mingguan

Tabel 3: Logbook Perkembangan Proyek

Tanggal	Kegiatan	Hasil / Progress
10/28/2025	Pembuatan Repository GitHub Tugas Besar	Repository 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 Projek (Visualizer) dari CV2 ke PyGame, Penambahan pattern beat baru dan asset 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
12/2/2025	Refactoring Audio Engine & Integrasi Sistem	Mengganti Arpeggiator dan Drum Machine terpisah dengan AudioEngine terpadu. Implementasi scheduler loop untuk timing yang lebih presisi. Mengubah kontrol drum menjadi finger mapping

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+