Deteksi Tangan untuk Menghitung Jumlah Jari menggunakan Python dengan Library OpenCV dan Mediapipe

Annisa Raihan Delana[1], Stephen Prasetya Chrismawan[2], Alvin Aryanta Suwardono[3], Maulana Zhahran[4]

[1,2,3,4]Informatika, Fakultas Teknik, Universitas Jenderal Soedirman, Indonesia
NIM : [1]H1D021021, [2]H1D021025, [3]H1D02039, [4]H1D02170,
Email: [1]annisa.delana@mhs.unsoed.ac.id, [2]stephen.chrismawan@mhs.unsoed.ac.id ,
[3]alvin.suwardono@mhs.unsoed.ac.id , [4]maulana.zhahran@mhs.unsoed.ac.id

(Artikel dikirimkan tanggal : 05 Desember 2023)

Abstrak

Pemanfaatan OpenCV dan MediaPipe dalam pengembangan aplikasi pengolahan citra digital pengenalan gestur berbasis tangan, dengan fokus pada deteksi dan penghitungan jari tangan manusia. OpenCV, sebuah API sumber terbuka untuk pemrosesan gambar, digunakan bersama MediaPipe, sebuah framework ditulis dalam bahasa C++, untuk mencapai deteksi tangan yang akurat. MediaPipe Hands, dengan dukungan machine learning real-time, memberikan solusi dengan 21 landmark pada tangan melalui regresi, memungkinkan pelokalan titik kunci yang tepat. Implementasi dilakukan melalui dua modul utama, yaitu HandTrackingModule untuk deteksi tangan dan FingerCounter untuk menghitung jumlah jari yang diangkat pada citra webcam secara real-time. Proses deteksi tangan menggunakan MediaPipe Hands, dan hasilnya digunakan untuk mendapatkan posisi landmark tangan. Penggunaan OpenCV memfasilitasi visualisasi landmark dan garis penghubung tangan pada citra yang menggunakan konsep pembuatan skeleton pada citra digital. Metode penelitian menciptakan modul tersebut dengan tujuan mendukung pengembangan aplikasi pengenalan gestur atau kontrol berbasis tangan. Hasil percobaan melibatkan pembuatan module HandTrackingModule dan FingerCounter, yang mampu mendeteksi tangan, menghitung jumlah jari yang diangkat, serta memberikan visualisasi pada citra webcam. Pengujian dilakukan dengan menggunakan dataset berisi gambar dunia nyata dengan 21 koordinat 3D, memastikan keakuratan dan keandalan model yang dikembangkan. Dengan demikian, pemanfaatan OpenCV dan MediaPipe dalam penelitian ini memberikan kontribusi pada pengembangan aplikasi yang dapat mengenali pola gerakan jari tangan manusia secara real-time, membuka peluang untuk berbagai aplikasi dalam pengenalan gestur dan interaksi berbasis tangan.

Kata kunci: *landmark, mediapipe, opencv*

Hand Detection for Counting Fingers using Python with OpenCV and Mediapipe

Abstract

The utilization of OpenCV and MediaPipe in the development of a digital image processing application for hand gesture recognition is the main focus of this research, particularly emphasizing the detection and counting of human fingers. OpenCV, an open-source API for image processing, is employed alongside MediaPipe, a framework written in C++, to achieve accurate hand detection. MediaPipe Hands, supported by real-time machine learning, provides a solution with 21 landmarks on the hand through regression, enabling precise localization of key points. The implementation involves two main modules: HandTrackingModule for hand detection and FingerCounter for real-time counting of raised fingers in webcam images. The hand detection process utilizes MediaPipe Hands, and the results are used to obtain the position of hand landmarks. The use of OpenCV facilitates the visualization of landmarks and connecting lines on images, employing the concept of creating a skeleton on digital images. The research methodology creates these modules with the aim of supporting the development of hand gesture recognition or control applications. Experimental results include the creation of the HandTrackingModule and FingerCounter modules, capable of detecting hands, counting raised fingers, and providing visualization in webcam images. Testing is conducted using a dataset containing real-world images with 21 3D coordinates, ensuring the accuracy and reliability of the developed model. Thus, the utilization of OpenCV and MediaPipe in this study contributes to the development of applications capable of real-time recognition of human hand movement patterns, opening possibilities for various applications in gesture recognition and hand-based interactions.

Keywords: *landmark, mediapipe, opencv*

1. PENDAHULUAN

OpenCV (Open Computer Vision) merupakan API yang bersifat open source yang bisa digunakan untuk image processing. Pada tahun 1999 OpenCV dikembangkan oleh Garry Bradsky dan baru dirilis tahun 2020. Beberapa algoritma didukung oleh OpenCV untuk jenis penelitian yang berhubungan dengan computer vision dan machine learning. Selain itu OpenCV juga bisa dikembangkan dengan bahasa pemrograman C++, Python, Java dan lain sebagainya[1]. Tidak hanya itu, OpenCV juga tersedia dalam berbagai platform, seperti Windows, Linux, OSX, Android, IOS, dan lain sebagainya. OpenCV juga dapat melakukan operasi high-speed GPU dengan menggunakan interface berbasis CUDA, CuDnn serta OpenCL. Kombinasi terbaik untuk dapat melakukan operasi berkecepatan tinggi tersebut adalah dengan perpaduan antara OpenCV, C++ API dan bahasa pemrograman Python[2].

MediaPipe juga merupakan framework yang ditulis menggunakan bahasa C++ dan bisa digunakan untuk mengembangkan aplikasi cross-platform, seperti windows, linux, macintosh, android bahkan website. Dengan menggunakan multithreading dan GPU acceleration, membuat MediaPipe sangat cepat dalam memproses banyak data yang masuk. MediaPipe mempunyai calculator yang akan berjalan ketika program dimulai dan akan berhenti ketika program selesai. Calculator tersebut digunakan MediaPipe untuk konsep multithreading[3]. MediaPipe menggunakan machine learning secara real-time (ML solution for live and streaming media). Machine learning yang ditawarkan oleh MediaPipe antara lain Face Detection, Face Mesh, Iris, Hands, Pose, Holistic, Hair Segmentation, Object Detection, Box Tracking, Instant Motion Tracking, Objectron, KNIFT[4]. Dari banyaknya solusi machine learning yang ditawarkan MediaPipe, maka penelitian ini memfokuskan penggunaan OpenCV dan MediaPipe untuk mengenali tangan kanan manusia dan melakukan segmentasi khususnya untuk jari jempol dan jari telunjuk saja. Hal yang dilakukan pertama kali setelah tangan terdeteksi oleh kamera/webcam, maka komputer akan memberikan landmark pada tangan. Proses pembentukan landmark ini didasarkan dari model yang dibuat oleh machine learning MediaPipe dan OpenCV.

Mediapipe Hands adalah solusi dari pelacakan tangan dan jari dengan sangat akurat. Fitur ini menggunakan machine learning untuk menyimpulkan 21 titik pada 3D landmark dari sebuah tangan dari satu frame secara real time. metode ini mencapai performa untuk menjalankan tracking secara langsung pada perangkat mobile, dan bahkan bisa mendeteksi lebih dari satu tangan. Setelah deteksi telapak tangan di seluruh gambar, model telapak tangan kami kemudian dilakukan pelokalan titik kunci yang tepat dari 21 koordinat buku jari 3D di dalam wilayah tangan yang terdeteksi melalui regresi, yaitu prediksi koordinat langsung. Model ini mempelajari representasi pose tangan internal yang konsisten dan kuat bahkan untuk tangan yang terlihat sebagian dan oklusi diri[5].

Untuk mendapatkan data kebenaran dasar, dilakukan pelabelan keterangan secara manual ~30K gambar dunia nyata dengan 21 koordinat 3D, seperti yang ditunjukkan di bawah ini (diambil nilai Z dari peta kedalaman gambar, jika ada per koordinat yang sesuai). Untuk menutupi kemungkinan pose tangan dengan lebih baik dan memberikan pengawasan tambahan pada sifat geometri tangan, juga dibuat model tangan sintetis berkualitas tinggi di berbagai latar belakang dan memetakannya ke koordinat 3D yang sesuai[6].

Berdasarkan latar belakang kecerdasan buatan pada computer vision, hand recognition, OpenCV dan MediaPipe maka penulis bertujuan untuk mengenali pola gesture gerakan jari tangan manusia, khususnya jari jempol dan jari telunjuk. Gerakan gesture yang dimaksudkan disini digunakan menghitung jumlah jari tangan yang diangkat secara real-time. OpenCV dan MediaPipe dengan bahasa python digunakan sebagai library pengenalan pola landmark model jari-jari tangan manusia yang terdeteksi pada kamera/webcam secara real-time.

2. METODE

Untuk membuat aplikasi deteksi jumlah jari tangan, tahapan yang perlu dilakukan adalah sebagai berikut :

2.1 Pembuatan Module HandTrackingModule

Pembuatan *HandTrackingModule* dimulai dengan inisialisasi kelas *handDetector* yang memungkinkan penggunaan parameter opsional untuk konfigurasi deteksi tangan. Dalam prosesnya, modul menggunakan library Mediapipe untuk mendeteksi dan melacak tangan dalam setiap frame kamera. Hasil deteksi ini kemudian diproses untuk mendapatkan posisi landmark tangan, yang nantinya dapat digunakan untuk berbagai aplikasi, seperti pengenalan gestur atau interaksi berbasis tangan. Pada tahap terakhir, modul memberikan opsi untuk menggambar landmark dan garis penghubung tangan pada citra, meningkatkan visualisasi hasil deteksi.

2.2 Pembuatan Module FingerCounter

Sementara itu, FingerCounter menggunakan modul HandTrackingModule yang telah dibuat sebelumnya. Tahap pertama melibatkan inisialisasi parameter, termasuk resolusi kamera dan objek detektor tangan. Selanjutnya, dalam loop pemrosesan video, program membaca frame dari

webcam dan menghitung jumlah jari yang diangkat dengan memanfaatkan posisi landmark tangan. Informasi ini ditampilkan secara langsung pada frame, termasuk jumlah jari yang diangkat dan jenis tangan (kanan atau kiri). Metode penelitian dalam jurnal ini menciptakan modul tersebut dengan tujuan utama mendukung pengembangan aplikasi pengenalan gestur atau kontrol berbasis tangan dengan menggunakan pendekatan hand tracking melalui library Mediapipe.

3. HASIL DAN PEMBAHASAN

Berikut ini adalah snapshot source code beserta penjelasan fungsi dari tahapan-tahapan yang ada pada metode penelitian :

3.1 Module HandTrackingModule

Pada module ini terdapat satu kelas yang bernama *handDetector* yang memiliki beberapa fungsi :

3.1.1 __init__

```
def __init__(self, mode=False, maxHands=2, modelComplexity=1, detectionCon=0.5, trackCon=0.5):
    # Inisialisasi kelas handDetector dengan parameter opsional
    self.mode = mode
    self.maxHands = maxHands
    self.modelComplexity = modelComplexity
    self.detectionCon = detectionCon
    self.trackCon = trackCon

    # Inisialisasi objek Mediapipe hands
    self.mpHands = mp.solutions.hands
    self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.modelComplexity,
                                    self.detectionCon, self.trackCon)
    # Inisialisasi objek Mediapipe drawing_utils
    self.mpDraw = mp.solutions.drawing_utils
    self.mpDrawStyles = mp.solutions.drawing_styles
```

Gambar 1. Snapshot Fungsi __init__

Constructor __init__ pada kelas handDetector seperti yang ditunjukkan pada Gambar 1 memiliki beberapa parameter yang digunakan untuk mengonfigurasi dan menginisialisasi objek detektor tangan. Beberapa parameter yang ada pada fungsi ini yaitu yang pertama adalah Parameter-mode.

Parameter-mode adalah parameter boolean yang menentukan apakah detektor tangan akan menggunakan model pelacakan tangan dalam mode statis (True) atau dinamis (False). Jika bernilai True, detektor akan beroperasi dalam mode statis, sedangkan jika False, detektor akan beroperasi dalam mode dinamis.

Parameter maxHands adalah parameter integer yang menentukan jumlah maksimum tangan yang akan dideteksi oleh detektor. Dengan mengatur nilai parameter ini, pengguna dapat mengontrol seberapa banyak tangan yang dapat dideteksi dalam satu frame.

Parameter modelComplexity adalah parameter integer yang menentukan kompleksitas model deteksi tangan. Nilainya harus berada dalam rentang [0, 2]. Semakin tinggi nilai kompleksitas, semakin kompleks modelnya. Pemilihan nilai kompleksitas ini dapat memengaruhi tingkat keakuratan dan kecepatan deteksi tangan.

Parameter detectionCon adalah parameter float yang merupakan ambang batas kepercayaan minimum untuk mendeteksi tangan. Hanya tangan dengan kepercayaan di atas nilai ini yang akan dianggap terdeteksi oleh detektor. Pengaturan ambang batas ini memungkinkan kontrol terhadap tingkat kepercayaan deteksi.

Parameter trackCon adalah parameter float yang merupakan ambang batas kepercayaan minimum untuk melacak tangan. Hanya tangan dengan kepercayaan di atas nilai ini yang akan dianggap berhasil dilacak. Dengan mengatur ambang batas pelacakan, pengguna dapat mempengaruhi ketepatan pelacakan tangan oleh detektor.

3.1.2 findHands

```
def findHands(self, img, draw=True):
    # Mengubah format citra ke RGB
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Proses deteksi tangan dengan Mediapipe hands
    self.results = self.hands.process(imgRGB)
    handedness = self.results.multi_handedness
    if handedness != None :
        for idx, hand_handedness in enumerate(handedness):
            handedness = hand_handedness.classification[0].label

    # Menggambar landmark dan garis penghubung tangan jika draw=True
    if self.results.multi_hand_landmarks:
        for handLms in self.results.multi_hand_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(img, handLms,
                                           self.mpHands.HAND_CONNECTIONS,
                                           self.mpDrawStyles.get_default_hand_landmarks_style(),
                                           self.mpDrawStyles.get_default_hand_connections_style()
                                           )
    return img, handedness
```

Gambar 2. Snapshot Fungsi findHands

Fungsi findHands dalam kelas handDetector seperi yang ditampilkan pada Gambar 2 bertujuan untuk mendeteksi dan, jika diinginkan, menggambar landmark tangan dan garis penghubungnya pada suatu citra. Berikut adalah penjelasan masing-masing statement dalam fungsi tersebut:

Pertama, citra masukan diubah formatnya menjadi RGB menggunakan cv2.cvtColor(img, cv2.COLOR_BGR2RGB). Ini diperlukan karena model deteksi tangan Mediapipe beroperasi pada citra dalam format RGB. Dan ini merupakan salah satu penerapan pengolahan citra digital yaitu mengubah format gambar.

Selanjutnya, proses deteksi tangan dilakukan dengan menggunakan objek self.hands (yang merupakan instance dari kelas mp.solutions.hands.Hands). Hasil deteksi disimpan dalam variabel self.results.

Setelah deteksi tangan dilakukan, dilakukan pengecekan terhadap multi_handedness (kedudukan tangan) dari hasil deteksi. Jika ada informasi kedudukan tangan, maka dilakukan iterasi untuk setiap tangan yang terdeteksi. Pada setiap iterasi, nilai dari handedness (kedudukan tangan) diubah sesuai dengan label klasifikasi tangan yang didapatkan dari hasil deteksi.

Selanjutnya, dilakukan pengecekan terhadap self.results.multi_hand_landmarks untuk memastikan bahwa terdapat landmark tangan yang terdeteksi. Jika ya, maka dilakukan iterasi untuk setiap tangan dan, jika draw=True, landmark tangan

dan garis penghubungnya digambar pada citra menggunakan self.mpDraw.draw_landmarks.

Terakhir, fungsi mengembalikan citra yang mungkin telah dimodifikasi (jika draw=True) dan nilai dari handedness. Hasil ini dapat digunakan untuk mengetahui kedudukan (left/right) dari tangan yang terdeteksi. Menampilkan landmark pada tangan menggunakan prinsip pengolahan citra digital. Salah satu prinsip yang juga digunakan pada proses ini yaitu menemukan skeleton dari citra.

3.1.3 findPosition

```python
def findPosition(self, img, handNo=0, draw=True):
    # Mendapatkan posisi landmark tangan
    lmList = []
    if self.results.multi_hand_landmarks:
        myHand = self.results.multi_hand_landmarks[handNo]
        # handedness = self.results.multi_handedness
        for id, lm in enumerate(myHand.landmark):
            # Mendapatkan koordinat piksel (cx, cy) dari landmark
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            # Menambahkan informasi posisi landmark ke dalam lmList
            lmList.append([id, cx, cy])
            # Menggambar lingkaran di atas setiap landmark jika draw=True
            if draw:
                cv2.circle(img, (cx, cy), 15, (255, 0, 255), cv2.FILLED)
                # Menambahkan teks ID landmark di sampingnya
                cv2.putText(img, str(id), (cx, cy), cv2.FONT_HERSHEY_PLAIN,
                            1, (255, 255, 255), 2)
    return lmList
```

Gambar 3. Snapshot Fungsi findPosition

Seperti yang ditampilkan dalam Gambar 3, fungsi findPosition pada kelas handDetector bertujuan untuk mendapatkan dan, jika diinginkan, menggambar posisi landmark tangan pada suatu citra. Berikut adalah penjelasan masing-masing statement dalam fungsi tersebut.

Fungsi dimulai dengan inisialisasi list lmList yang akan digunakan untuk menyimpan informasi posisi landmark tangan. Selanjutnya, dilakukan pengecekan terhadap self.results.multi_hand_landmarks untuk memastikan bahwa terdapat landmark tangan yang terdeteksi.

Jika ada landmark tangan yang terdeteksi, program memilih tangan tertentu (berdasarkan indeks handNo) dari daftar tangan yang terdeteksi. Selanjutnya, dilakukan iterasi untuk setiap landmark tangan pada tangan yang dipilih.

Dalam setiap iterasi, koordinat piksel (cx dan cy) dari landmark dihitung berdasarkan lebar (w) dan tinggi (h) citra. Informasi tersebut kemudian ditambahkan ke dalam list lmList dalam bentuk [id, cx, cy], yang merepresentasikan ID landmark dan koordinat piksel x dan y.

Selanjutnya, terdapat blok kode yang berfungsi menggambar lingkaran di atas setiap landmark jika parameter draw bernilai True. Lingkaran ini dihasilkan menggunakan fungsi cv2.circle dengan warna (255, 0, 255), yang merupakan warna magenta, dan diisi penuh (cv2.FILLED). Selain itu, teks ID landmark juga ditambahkan di sampingnya menggunakan fungsi cv2.putText.

Terakhir, fungsi mengembalikan list lmList yang berisi informasi posisi landmark tangan. Hasil ini dapat digunakan untuk analisis lanjutan atau penggunaan lainnya sesuai dengan kebutuhan aplikasi.

3.2 Module FingerCounter

```python
import cv2
import time
import HandTrackingModule as htm

# Tentukan resolusi kamera
wCam, hCam = 640, 480

# Ambil video dari webcam
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)

# Inisialisasi waktu frame sebelumnya
pTime = 0

# Buat objek detektor tangan
detector = htm.handDetector(detectionCon=0.75)

# Tentukan ID ujung jari
tipIds = [4, 8, 12, 16, 20]
```

Gambar 4. Snapshot Source Code Module FingerCounter

Bagian dari source code yang ditampilkan pada Gambar 4 bertujuan untuk menentukan resolusi kamera, mengambil video dari webcam, menginisialisasi waktu frame sebelumnya, membuat objek detektor tangan menggunakan modul HandTrackingModule, dan menentukan ID ujung jari yang akan digunakan dalam proses penghitungan jari.

Pertama, program menentukan resolusi kamera yang diinginkan dengan variabel wCam dan hCam yang masing-masing menunjukkan lebar dan tinggi citra kamera. Nilai-nilai ini kemudian akan digunakan untuk mengatur resolusi kamera saat membuka video dari webcam.

Selanjutnya, program menggunakan OpenCV untuk mengakses video dari webcam dengan menggunakan cv2.VideoCapture(0). Kode cap.set(3, wCam) dan cap.set(4, hCam) digunakan untuk mengatur lebar dan tinggi citra kamera sesuai dengan nilai yang telah ditentukan sebelumnya.

Selanjutnya, variabel pTime diinisialisasi dengan nilai waktu awal. Ini akan digunakan untuk mengukur waktu yang diperlukan untuk memproses setiap frame sehingga dapat dihitung frames per detik (FPS).

Selanjutnya, program membuat objek detektor tangan dari kelas handDetector yang diberi nama detector. Objek ini diinisialisasi dengan parameter detectionCon sebesar 0.75, yang merupakan ambang batas kepercayaan untuk mendeteksi tangan. Detektor ini menggunakan modul HandTrackingModule yang disamakan dengan htm.

Program juga mendefinisikan list tipIds yang berisi indeks landmark dari ujung jari-jari tangan yang akan digunakan dalam proses penghitungan jari. Setiap elemen dalam list ini merepresentasikan indeks dari ujung jari ibu jari, telunjuk, jari tengah, jari manis, dan jari kelingking sesuai dengan urutan yang diberikan. List ini akan digunakan dalam langkah-langkah selanjutnya untuk mengidentifikasi posisi dan gerakan masing-masing jari pada tangan yang terdeteksi.

```
# Mulai loop pemrosesan video
while True:
    # Baca frame dari webcam
    success, img = cap.read()
    # Deteksi tangan dalam frame
    img, hand_type = detector.findHands(img)
    # Dapatkan landmark (titik kunci) tangan
    lmList = detector.findPosition(img, draw=False)
    img = cv2.flip(img, 1)


    # Periksa apakah ada tangan yang terdeteksi
    if len(lmList) != 0:
        # Inisialisasi list jari kosong
        fingers = []

        # Periksa posisi ibu jari
        if (lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1]) and hand_type == "Left":
            fingers.append(1)
        elif (lmList[tipIds[0]][1] > lmList[tipIds[0] - 1][1]) and hand_type == "Right" :
            fingers.append(0)
        elif (lmList[tipIds[0]][1] < lmList[tipIds[0] - 1][1]) and hand_type == "Right" :
            fingers.append(1)
        elif (lmList[tipIds[0]][1] < lmList[tipIds[0] - 1][1]) and hand_type == "Left" :
            fingers.append(0)
```

Gambar 5. Snapshot Source Code Module FingerCounter

Seperti yang ditampilkan pada Gambar 5, dalam loop tak terbatas (while True), program membaca frame terbaru dari webcam menggunakan cap.read(), dan menyimpannya dalam variabel img. Setelah itu, dilakukan deteksi tangan pada frame menggunakan metode findHands dari objek detektor (detector). Hasil deteksi disimpan kembali dalam variabel img, dan tipe tangan (kanan atau kiri) diperoleh dari nilai yang dikembalikan oleh metode.

Selanjutnya, posisi landmark (titik kunci) tangan diperoleh dengan memanggil metode findPosition dari objek detektor. Parameter draw diatur sebagai False sehingga landmark tidak digambar pada citra. Posisi landmark ini disimpan dalam variabel lmList.

Setelah itu, citra di-flip secara horizontal menggunakan cv2.flip untuk mendukung pemrosesan tangan yang dapat dikenali oleh detektor tangan, terlepas dari posisi kamera.

Selanjutnya, program melakukan pemeriksaan apakah terdapat tangan yang terdeteksi dengan memeriksa panjang dari list lmList. Jika terdapat tangan, maka dilakukan pengolahan lebih lanjut untuk mengidentifikasi posisi ibu jari dan 4 jari lainnya berdasarkan koordinat landmark.

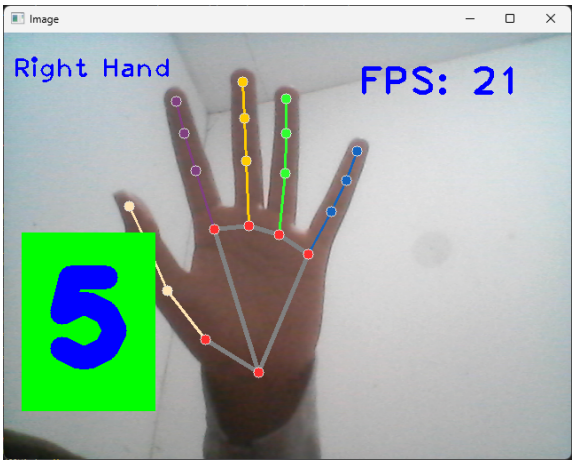
Posisi ibu jari diperiksa dengan membandingkan koordinat y dari ujung ibu jari (lmList[tipIds[0]][1]) dengan koordinat y dari jari sebelumnya (lmList[tipIds[0] - 1][1]). Berdasarkan perbandingan ini dan tipe tangan, nilai 1 atau 0 dimasukkan ke dalam list fingers untuk merepresentasikan apakah ibu jari diangkat atau tidak.

```
        # Periksa posisi 4 jari lainnya
        for id in range(1, 5):
            if lmList[tipIds[id]][2] < lmList[tipIds[id] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)
        hand_type = "Right" if hand_type == "Left" else "Left"
        # Hitung total jumlah jari yang diangkat
        totalFingers = fingers.count(1)
        # Tampilkan total jari yang diangkat
        print(f"{hand_type}: {totalFingers} fingers")
        # Gambar persegi untuk area penghitungan
        cv2.rectangle(img, (20, 225), (170, 425), (0, 255, 0), cv2.FILLED)
        cv2.putText(img, f'{hand_type} Hand', (10, 50), cv2.FONT_HERSHEY_PLAIN, 2, (255, 0, 0), 2)
        # Tampilkan total jari yang diangkat dalam persegi
        cv2.putText(img, str(totalFingers), (45, 375), cv2.FONT_HERSHEY_PLAIN, 10, (255, 0, 0), 25)
    # Hitung FPS
    cTime = time.time()
    fps = 1 / (cTime - pTime)
    pTime = cTime
    # Tampilkan FPS pada frame
    cv2.putText(img, f'FPS: {int(fps)}', (400, 70), cv2.FONT_HERSHEY_PLAIN, 3, (255, 0, 0), 3)
    # Tampilkan frame
    cv2.imshow("Image", img)
    # Periksa penekanan tombol 'q' untuk keluar
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Tutup semua jendela
cv2.destroyAllWindows()
```

Gambar 5. Snapshot Source Code Module FingerCounter

Selanjutnya, seperti pada Gambar 5, program melakukan pemeriksaan untuk 4 jari lainnya dengan iterasi menggunakan loop for. Setiap jari diidentifikasi berdasarkan perbandingan koordinat z dari ujung jari dengan jari sebelumnya. Hasil identifikasi ini juga dimasukkan ke dalam list fingers.

Selanjutnya, program menghitung total jumlah jari yang diangkat dengan menghitung jumlah nilai 1 dalam list fingers. Hasil perhitungan ini dicetak pada konsol sebagai output monitoring.



Gambar 6. Tampilan Webcam

Program juga menampilkan visualisasi, seperti yang terlihat pada Gambar 6, pada citra dengan menggambar persegi dan teks yang menunjukkan jumlah jari yang diangkat. Warna persegi dan teks disesuaikan untuk memberikan informasi visual. Tipe tangan juga ditampilkan di layar.

Terakhir, program memperbarui tipe tangan ke "Right" jika sebelumnya adalah "Left" dan sebaliknya, untuk persiapan pada iterasi berikutnya. Program kemudian menampilkan hasil visual pada citra dan melakukan update FPS pada layar.

Penting untuk dicatat bahwa program ini bersifat tak terbatas dan dapat dihentikan dengan menekan tombol 'q'.

4. KESIMPULAN

Kesimpulan dari hasil dan pembahasan adalah penggunaan OpenCV dan MediaPipe untuk deteksi tangan dan penghitungan jari. MediaPipe Hands memberikan solusi akurat dengan 21 landmark pada tangan melalui regresi. Modul implementasi mencakup HandTrackingModule untuk deteksi dan FingerCounter untuk menghitung jumlah jari yang diangkat pada citra webcam. Dengan demikian, pemanfaatan OpenCV dan MediaPipe menjadi dasar untuk aplikasi pengenalan gestur berbasis tangan yang mengaplikasikan konsep pengolahan citra digital. Pengolahan citra digital yang digunakan yaitu mengubah format citra supaya bisa diolah dan menemukan skeleton pada objek di dalam citra.

DAFTAR PUSTAKA

[1] T. C. A.-S. Zulkhaidi, E. Maria, dan Y. Yulianto, "Pengenalan Pola Bentuk Wajah dengan OpenCV," *J. Rekayasa Teknol. Inf. JURTI*, vol. 3, no. 2, hlm. 181, Jun 2020, doi: 10.30872/jurti.v3i2.4033.

[2] F. Damatraseta, R. Novariany, dan M. A. Ridhani, "Real-time BISINDO Hand Gesture Detection and Recognition with Deep Learning CNN," *J. Inform. Kesatuan*, vol. 1, no. 1, hlm. 71–80, Jul 2021, doi: 10.37641/jikes.v1i1.774.

[3] S. Nur Budiman, S. Lestanti, S. Marselius Evvandri, dan R. Kartika Putri, "PENGENALAN GESTUR GERAKAN JARI UNTUK MENGONTROL VOLUME DI KOMPUTER MENGGUNAKAN LIBRARY OPENCV DAN MEDIAPIPE," *Antivirus J. Ilm. Tek. Inform.*, vol. 16, no. 2, hlm. 223–232, Nov 2022, doi: 10.35457/antivirus.v16i2.2508.

[4] F. Zhang *dkk.*, "MediaPipe Hands: On-device Real-time Hand Tracking," 2020, doi: 10.48550/ARXIV.2006.10214.

[5] A. A. Rizki dan A. U. Zailani, "Implementasi dan Perancangan Virtual Mouse dengan Hand Gesture Recognition menggunakan OpenCV," vol. 1, no. 4, 2023.

[6] A. M. Chalik, B. A. Qowy, F. Hanafi, dan A. Nuraminah, "Mouse Tracking Tangan dengan Klasifikasi Gestur Menggunakan OpenCV dan Mediapipe," vol. 1, no. 2, 2021.