

# Aplikasi Algoritma Divide and Conquer Pada Hand Tracking Sederhana

Althaaf Khasyi Atisomya - 13521130

Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung  
E-mail: althaafka004@gmail.com

**Abstract**—*Hand Tracking* atau pelacakan pergerakan tangan adalah metode yang digunakan untuk menafsirkan gerakan tangan manusia melalui perangkat lunak maupun perangkat keras secara *real-time*. Dalam penerapannya *hand tracking* melibatkan pemrosesan citra serta algoritma untuk menafsirkan gerakan tangan. Salah satu cara untuk menafsirkan pergerakan tangan dapat dilakukan dengan pencarian *convex hull*. *Convex hull* adalah himpunan titik-titik terluar sebuah objek yang dapat menutup semua bagian objek. Pencarian *convex hull* dapat memanfaatkan algoritma *divide and conquer*. Algoritma *divide and conquer* adalah strategi pemecahan masalah dengan membagi masalah menjadi bagian-bagian yang lebih kecil kemudian menggabungkan solusi dari bagian bagian tersebut untuk mendapat solusi dari masalah yang utuh. Dalam penelitian ini, algoritma *divide and conquer* akan diterapkan untuk melakukan pelacakan pada pergerakan tangan berupa deteksi jumlah jari.

**Keywords**—*hand tracking; divide and conquer; convex hull;*

## I. PENDAHULUAN

Dalam perkembangan teknologi yang semakin pesat, interaksi antara pengguna dan komputer semakin menarik perhatian. Salah satu aplikasi penting dalam interaksi antara pengguna dan komputer adalah pengenalan gerakan manusia khususnya pelacakan pergerakan tangan atau *hand tracking*.

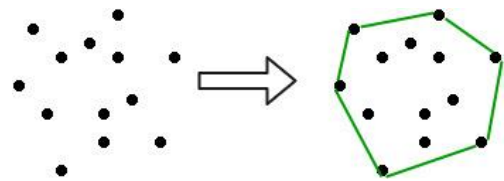
*Hand tracking* memungkinkan sistem secara *real-time* mengidentifikasi arti dari pergerakan tangan manusia. Hal ini membuka peluang sangat luas untuk membuat aplikasi yang lebih intuitif. Contoh aplikasi penerapan *hand tracking* yang sudah ada saat ini adalah pengendalian permainan *game* yang dapat dilakukan dengan gerakan tangan, pada bidang robotika untuk mengendalikan robot berbasis tangan, dan pada *virtual and augmented reality* yang memungkinkan pengguna untuk berinteraksi dengan lingkungan virtual.



Gambar 1. Aplikasi Penerapan Hand Tracking

(Sumber: <https://dailysocial.id/post/microsoft-kembangkan-teknologi-hand-tracking-canggih-untuk-berinteraksi-dengan-objek-virtual> 21 Mei 2022

Untuk dapat mendapatkan informasi dari pergerakan tangan, *hand tracking* memerlukan algoritma yang efektif dan akurat dalam menafsirkan data. Salah satu cara untuk melacak pergerakan tangan dapat dilakukan dengan memanfaatkan *convex hull*. *Convex hull* adalah himpunan terkecil titik-titik pada objek yang dapat membungkus seluruh objek.



Gambar 2. Contoh Convex Hull

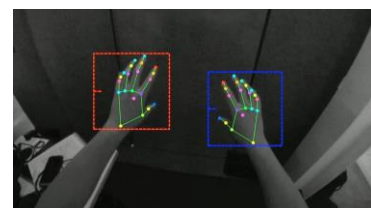
(Sumber: <https://www.geeksforgeeks.org/convex-hull-using-divide-and-conquer-algorithm/> 21 Mei 2022 pukul 20.32)

Pada penelitian ini, akan dibahas aplikasi *hand tracking* sederhana untuk menghitung jari pada gestur tangan. Dengan menggunakan algoritma *divide and conquer* untuk mencari *convex hull* akan didapatkan bentuk umum dari tangan yang kemudian dapat diolah lebih lanjut untuk mendeteksi jumlah jari dari gestur tangan.

## II. LANDASAN TEORI

### A. Hand Tracking

*Hand tracking* atau pelacakan pergerakan tangan adalah metode yang digunakan untuk pemantauan dan pengenalan gerakan tangan manusia menggunakan perangkat keras maupun perangkat lunak. *Hand tracking* bertujuan untuk memperoleh informasi tentang posisi, orientasi, dan arti isyarat tangan secara *real-time*.



Gambar 3. Contoh Hasil Penerapan Hand Tracking

(Sumber: <https://www.uploadvr.com/quest-hand-tracking-2-1-fast-movements/> 21 Mei 2022 pukul 19.50)

Proses pelacakan pergerakan tangan melibatkan penerapan teknik seperti perekaman data, pemrosesan citra, deteksi tangan, dan pemodelan matematika untuk mengartikan data yang diterima. Algoritma yang digunakan dalam pelacakan pergerakan tangan juga dapat berbeda-beda bergantung dari informasi apa saja yang ingin didapat dari gerakan tangan.

Proses pelacakan pergerakan tangan umumnya terdiri dari beberapa langkah penting meliputi

1. Perekaman data

Perekaman data merupakan langkah awal dalam proses pelacakan tangan. Perekaman data dilakukan dengan memanfaatkan perangkat keras yang sesuai untuk mendapatkan citra digital gerakan tangan. Dalam hal ini, perangkat keras yang digunakan adalah kamera untuk menghasilkan data berupa rangkaian *frame* video yang selanjutnya akan diolah pada tahap berikutnya.

2. Deteksi tangan dan segmentasi kulit

Deteksi tangan dapat dilakukan dengan segmentasi warna kulit. Segmentasi kulit ini bertujuan untuk mendeteksi *pixel* yang memiliki warna kulit dari setiap citra digital yang ditangkap kamera. Pada umumnya citra yang ditangkap kamera memiliki format RGB. Untuk mempermudah pendeteksian warna kulit, format RGB akan diubah ke format YcrCb.

Format YcrCb memiliki keunggulan dapat memisahkan informasi kecerahan dengan komponen warna, di mana komponen Y (*luminance*) memberikan informasi tentang kecerahan *pixel*, sementara komponen Cr dan Cb (*chrominance*) memberikan informasi tentang tingkat perbedaan warna. Hal ini memudahkan memisahkan area kulit dengan latar belakang yang memiliki perbedaan signifikan.

3. Penafsiran Pergerakan Tangan

Setelah tidak terdapat objek-objek yang tidak dibutuhkan maka akan dilakukan pencarian *convex hull* pada citra tangan. *Convex hull* digunakan untuk mengelompokkan titik-titik yang mewakili jari-jari tangan dan membentuk area terluar yang mencakup seluruh tangan. Untuk mengidentifikasi celah jari akan dilakukan perhitungan dengan memanfaatkan *convexity defect* dari *convex hull* yang ada.

B. Algoritma Divide and Conquer

Algoritma *divide and conquer* adalah strategi pemecahan masalah dengan membagi masalah menjadi bagian-bagian yang lebih kecil kemudian menggabungkan solusi dari bagian-bagian tadi untuk mendapatkan solusi masalah secara utuh. Pada dasarnya algoritma ini kan memecah masalah yang kompleks menjadi lebih sederhana sehingga lebih mudah dan cepat untuk diselesaikan.

Algoritma *divide and conquer* terdiri dari tiga tahap, yaitu

1. *Divide*, yaitu dengan membagi masalah menjadi beberapa sub masalah yang masih memiliki

karakteristik yang sama dengan masalah utama namun lebih sederhana.

2. *Conquer (Solve)*, yaitu tahapan dimana masing-masing submasalah diselesaikan secara langsung apabila telah menjadi sangat sederhana atau secara rekursif apabila masih kompleks.
3. *Combine*, yaitu menggabungkan solusi dari masing-masing submasalah sehingga membantu solusi untuk masalah utama.

```

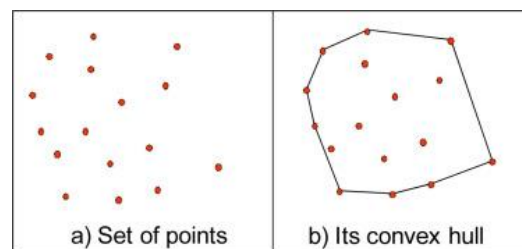
procedure DIVIDEandCONQUER(input P : problem, n : integer)
  { Menyelesaikan persoalan P dengan algoritma divide and conquer
  Masukan: masukan persoalan P berukuran n
  Luaran: solusi dari persoalan semula }
  Deklarasi
  r : integer

  Algoritma
  if n ≤ n0 then {ukuran persoalan P sudah cukup kecil }
    SOLVE persoalan P yang berukuran n ini
  else
    DIVIDE menjadi r upa-persoalan, P1, P2, ..., Pr, yang masing-masing berukuran n1, n2, ..., nr
    for masing-masing P1, P2, ..., Pr, do
      DIVIDEandCONQUER(Pi, ni)
    endfor
    COMBINE solusi dari P1, P2, ..., Pr menjadi solusi persoalan semula
  endif
  
```

Gambar 4. Skema Umum Algoritma Divide and Conquer  
 (Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf) 22 Mei 2022 pukul 00.49)

C. Convex Hull

*Convex hull* adalah konsep dalam geometri yang mengacu pada himpunan terkecil dari himpunan titik-titik yang dapat membungkus seluruh himpunan titik tersebut. Himpunan titik-titik ini kemudian akan membentuk poligon yang mengelilingi seluruh titik.



Gambar 5. Himpunan Titik dan Convex Hull-nya  
 (Sumber: <https://www.sciencedirect.com/topics/earth-and-planetary-sciences/convex-hull> 22 Mei 2022 pukul 01.03)

Terdapatat berbagai macam algoritma untuk mencari *convex hull* dari suatu himpunan titik. Namun, yang akan digunakan pada penelitian ini adalah *convex hull* dengan memanfaatkan algoritma *divide and conquer*.

Secara umum proses pencarian *convex hull* pada himpunan titik S yang memiliki titik sebanyak n dengan algoritma *divide and conquer* adalah

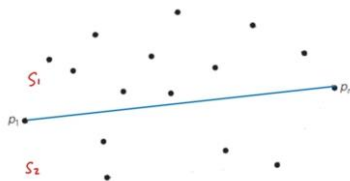
1. Mengurutkan himpunan titik  $S = \{p_1, p_2, p_3, \dots, p_n\}$  berdasarkan nilai absis yang menaik, jika ada nilai absis yang sama diurutkan berdasarkan nilai ordinat menaik.
2. Membuat garis yang menghubungkan  $p_1$  dan  $p_n$ . Kedua titik ini merupakan titik yang memiliki jarak horizontal

terjauh sehingga kedua titik ini pasti merupakan bagian dari *convex hull*. Masukkan  $p_1$  dan  $p_n$  ke himpunan penyelesaian.

- Membagi titik-titik menjadi dua bagian yaitu  $S_1$  yaitu semua titik yang berada pada sebelah kiri atas dari garis  $p_1p_n$  dan  $S_2$  yaitu semua titik yang berada pada sebelah kanan bawah garis  $p_1p_n$ . Untuk menentukan letak titik relatif terhadap garis akan menggunakan rumus determinan.

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_2y_3 + x_3y_1 - x_3y_2 - x_2y_1 - x_1y_3$$

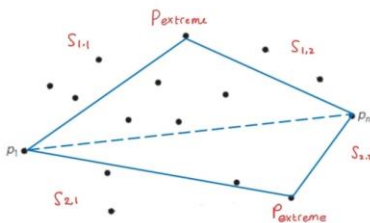
Titik  $(x_3, y_3)$  berada disebelah kiri atas garis  $((x_1, y_1), (x_2, y_2))$  jika hasil positif, sebelah kanan bawah jika negatif. Semua titik yang berada pada garis  $p_1p_n$  tidak mungkin merupakan *convex hull* sehingga diabaikan.



**Gambar 6.** Ilustrasi Langkah 3 Pencarian *Convex Hull*

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian4.pdf) 22 Mei 2022 pukul 07.35)

- Untuk sebuah bagian terdapat dua kemungkinan yaitu apabila tidak ada titik lain, maka  $p_1, p_n$  menjadi pembentuk *convex hull* bagian tersebut. Jika masih terdapat titik, pilihlah titik yang memiliki jarak terjauh dari garis  $p_1p_n$  (misal  $p_{extreme}$ ) dengan menggunakan rumus determinan. Masukkan  $p_{extreme}$  ke himpunan penyelesaian. Semua titik yang berada dalam segitiga  $p_{extreme}p_1p_n$  diabaikan.
- Membagi kumpulan titik pada masing-masing sub-bagian (misal  $S_1$ ) menjadi bagian  $S_{1,1}$  (sebelah kiri atas garis  $p_1p_{extreme}$ ) dan bagian  $S_{1,2}$  (sebelah kiri atas garis  $p_n p_{extreme}$ ).

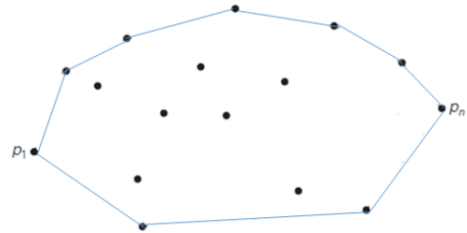


**Gambar 7.** Ilustrasi Langkah 5 Pencarian *Convex Hull*

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian4.pdf) 22 Mei 2022 pukul 07.37)

- Ulangi langkah 4 dan 5 untuk masing-masing bagian yang ada secara rekursif.

Dari proses pencarian *convex hull* diatas akan didapat himpunan titik-titik yang dapat mencangkup semua titik apabila dihubungkan dengan garis lurus. Agar proses menghubungkan titik-titik *convex hull* mudah,  $p_{extreme}$  dimasukkan pada indeks setelah  $p_1$  tiap memasukkan  $p_{extreme}$  pada himpunan solusi. Hal ini dilakukan agar himpunan solusi terurut dan proses menghubungkan titik-titik *convex hull* lebih mudah.



**Gambar 8.** Hasil Pencarian *Convex Hull*

(Sumber: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian4.pdf) 22 Mei 2022 pukul 07.55)

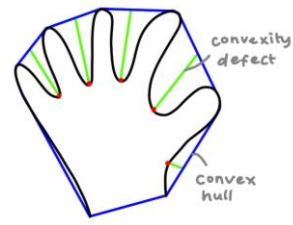
#### D. Penafsiran Hand Tracking

Identifikasi pergerakan tangan dilakukan dengan memanfaatkan *convex hull* untuk mencari *convexity defects* dari poligon hasil *convex hull*. *Convexity defects* adalah konsep dalam geometri yang mengacu pada bagian dari titik objek yang tidak mengikuti pola *convex hull* dari objek tersebut.



**Gambar 9.** *Convexity Defects* yang Ditandai dengan Titik Merah

(Sumber: <https://theailearner.com/2020/11/09/convexity-defects-opencv/> 22 Mei 2022 pukul 02.14)



**Gambar 10.** Ilustrasi Pemanfaatan *Convexity Defects* dalam Mengenali Celah Jari

Ketika tangan membentuk pose tertentu seperti membuka jari-jari tangan, akan terbentuk celah antara jari-jari tersebut. Celah jari tersebut dapat dideteksi menggunakan *convexity defects*. Secara visual, *convexity defects* terlihat sebagai titik pada *contour* tangan yang melengkung ke dalam dan jauh dari *convex hull*. Dengan memanfaatkan *convexity defects* ini kita dapat menghitung jumlah celah jari tangan.

Perlu dilakukan perhitungan untuk membedakan *convexity defects* yang merupakan celah tangan dengan *convexity defects* yang bukan merupakan celah jari tangan. Perbedaan tersebut dapat dilihat dengan menghitung sudut sudut antara *convexity defects* dan *convex hull* yang mengapitnya serta jarak titik *convexity defects* dengan *convex hull*-nya menggunakan rumus sebagai berikut,

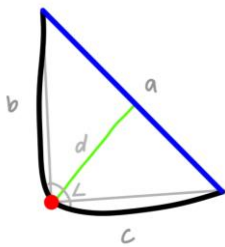
$$s = \frac{a + b + c}{2}$$

$$L = \sqrt{s \times (s - a) \times (s - b) \times (s - c)}$$

$$d = \frac{2a}{L}$$

$$\angle = \cos^{-1}\left(\frac{b^2 + c^2 - a^2}{2bc}\right)$$

dengan L adalah luas yang terbentuk oleh *convex hull* dan *convexity defect*, d adalah jarak garis *convex hull* dan *convexity defect* dan  $\angle$  adalah sudut *convex hull* dan *convexity defect*.



Gambar 11. Ilustrasi Pemanfaatan Perhitungan Luas dan Jarak

Pada penelitian ini *convexity defect* dapat disebut sebagai celah jari apabila memiliki sudut kurang dari  $90^\circ$  dan jarak dengan garis *convex hull*-nya lebih dari 30 pixel pada ukuran kamera  $350 \times 350$  pixel. Setelah celah jari didapatkan maka jumlah jari dapat dihitung sebagai jumlah *convexity defects* yang merupakan celah jari ditambah dengan satu.

### III. IMPLEMENTASI

#### A. Perekaman Data dengan Library OpenCV

Penelitian ini akan menggunakan library OpenCV dalam bahasa Python untuk mendapatkan data berupa video secara real-time. Objek kamera dapat diinisiasi dengan menggunakan fungsi 'cv2.VideoCapture(0)'. Ukuran layar kamera juga akan disesuaikan menurut kebutuhan.

Setelah kamera berhasil diinisiasi, akan digunakan *while loop* untuk terus membaca *frame* video menggunakan fungsi 'camera.read()'. Metode ini akan mengembalikan *frame* yang berupa gambar.

Apabila pembacaan *frame* berhasil, akan dilakukan hand tracking pada *frame* menggunakan fungsi *hand\_tracking*(*frame*) yang akan dijelaskan di tahap selanjutnya. Fungsi ini akan mengembalikan *frame* yang telah diolah untuk mendeteksi pergerakan tangan.

Hasil dari *frame* yang telah diolah akan ditampilkan menggunakan *cv2.imshow()*. Fungsi ini akan membuat *window* yang menampilkan *hand tracker* secara *real-time*. Untuk menyimpan *frame* dapat dilakukan dengan menekan tombol 's' dan untuk keluar dari program dapat dilakukan dengan menekan tombol 'q'.

```
# setup camera
camera = cv2.VideoCapture(0)
camera.set(cv2.CAP_PROP_FRAME_WIDTH, 350)
camera.set(cv2.CAP_PROP_FRAME_HEIGHT, 350)

while True:
    _, frame = camera.read()
    frame = cv2.flip(frame, 1)

    # hand tracking
    try:
        frame = hand_tracking(frame)
    except:
        pass

    cv2.imshow('WebCam', frame)

    if cv2.waitKey(1) == ord('q'):
        break
    if cv2.waitKey(1) == ord('s'):
        cv2.imwrite("image.jpg", frame)
        break

camera.release()
cv2.destroyAllWindows()
```

Gambar 12. Implementasi Setup Kamera

#### B. Deteksi Tangan dan Segmentasi Kulit

Tahap selanjutnya adalah tahap deteksi tangan dan segmentasi kulit. Tujuan dari tahap ini adalah untuk memisahkan latar belakang dan objek-objek yang tidak diperlukan. Segmentasi kulit ini bertujuan untuk mendeteksi *pixel* yang memiliki warna kulit dari setiap citra digital yang ditangkap kamera.

```
def skin_segmentation(image):
    # change RGB to YCrCb
    image = cv2.blur(image, (3, 3))
    image_YCrCb = cv2.cvtColor(image, cv2.COLOR_BGR2YCR_CB)

    # range warna kulit
    min_YCrCb = np.array([0,133,77], np.uint8)
    max_YCrCb = np.array([235,173,127], np.uint8)

    # skin segmentation
    skin_region = cv2.inRange(image_YCrCb, min_YCrCb, max_YCrCb)
    skin_image = cv2.bitwise_and(image, image, mask = skin_region)
    skin_image = cv2.erode(skin_image, np.ones((5, 5), np.uint8))

    # convert to black and white
    skin_image = cv2.cvtColor(skin_image, cv2.COLOR_BGR2GRAY)
    skin_image = cv2.threshold(skin_image, 2, 255, cv2.THRESH_BINARY)[1]

    return skin_image
```

Gambar 13. Implementasi Segmentasi Kulit

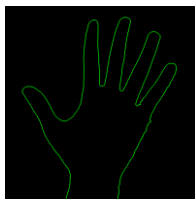
Untuk mendeteksi warna kulit terlebih dahulu kita ubah format RGB menjadi format YCrCb dengan fungsi *cv2.cvtColor*( *image*, *cv2.COLOR\_BGR2YCR\_CB*). Kemudian, kita definisikan *range* warna kulit yaitu *min\_YCrCb* dan *max\_YCrCb* dan menghapus *pixel* yang tidak berada pada *range min\_YCrCb* dan *max\_YCrCb*.

Pada tahap ini latar belakang telah hilang menyisakan *pixel* dengan warna kulit saja yaitu tangan. Kita ubah image menjadi warna hitam putih dengan warna putih merupakan tangan dan warna hitam merupakan *background*.



Gambar 14. Hasil Segmentasi Kulit

Setelah gambar hanya menyisakan tangan, perlu didefinisikan seluruh titik yang membentuk objek tangan dengan menggunakan fungsi `cv2.findContours(...)` yang mengembalikan array of *Contour*.



Gambar 15. Titik-Titik Pembentuk Tangan

### C. Penerapan Convex Hull

Dengan memanfaatkan titik-titik pembentuk objek tangan, akan dicari *convex hull* dengan objek tangan dengan algoritma *divide and conquer*, adapun implementasi kode pencarian *convex hull* sebagai berikut

```
def det(x1,x2,x3):
    # fungsi determinan untuk menentukan posisi titik x3
    # relatif terhadap garis yang dibentuk x1 dan x2
    return (x1[0]*x2[1] + x3[0]*x1[1] + x2[0]*x3[1]
            - x3[0]*x2[1] - x2[0]*x1[1] - x1[0]*x3[1])

def convex_hull(points):
    n = len(points)
    points.sort(key=lambda x: (x[0],x[1]))
    p1 = points[0]
    pn = points[n-1]

    # himpunan solusi hull
    global hull
    hull = [p1,pn]

    # membagi points menjadi dua bagian
    points_1 = list(filter(lambda x: (det(p1,pn,x) > 0), points))
    points_2 = list(filter(lambda x: (det(p1,pn,x) < 0), points))

    # rekursif untuk kedua bagian
    convex_hull_recursion(points_1,p1,pn,True)
    convex_hull_recursion(points_2,p1,pn,False)

    return hull
```

Gambar 16. Implementasi Pencarian Convex Hull

```
def convex_hull_recursion(points,p1,pn,isUpper):
    global hull

    if len(points) == 0: # tidak ada titik yang memenuhi
        return

    elif len(points) ==1: # hanya ada satu titik yang memenuhi
        p_idx = hull.index(p1 if isUpper else pn)
        hull.insert(p_idx+1,points[0])
        return

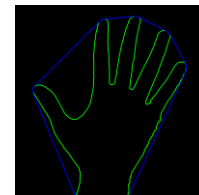
    else:
        p_idx = hull.index(p1 if isUpper else pn)
        points_det = [det(p1,pn,x) for x in points]

        # cari p extreme dan masukkan pada himpunan solusi
        idx_extreme = max(points_det) if isUpper else min(points_det)
        p_extreme = points[points_det.index(idx_extreme)]
        hull.insert(p_idx+1,p_extreme)

        # membagi lagi menjadi dua bagian
        if (isUpper):
            points_2 = list(filter(
                lambda x: (det(p_extreme,pn,x) > 0), points))
            points_1 = list(filter(
                lambda x: (det(p1,p_extreme,x) > 0), points))
        else:
            points_2 = list(filter(
                lambda x: (det(p_extreme,pn,x) < 0), points))
            points_1 = list(filter(
                lambda x: (det(p1,p_extreme,x) < 0), points))

        # rekursif untuk kedua bagian
        convex_hull_recursion(points_1,p1,p_extreme,isUpper)
        convex_hull_recursion(points_2,p_extreme,pn,isUpper)
```

Gambar 7. Implementasi Pencarian Convex Hull Rekursif

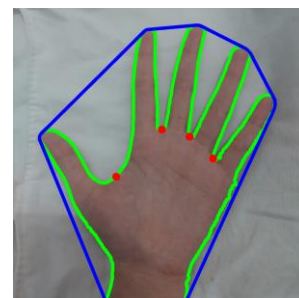


Gambar 17. Hasil Pencarian Convex Hull

### D. Penafsiran Hand Tracking

Tahapan selanjutnya adalah menafsirkan gestur tangan dengan memanfaatkan *convexity defects* dari *convex hull* yang telah kita temukan. Untuk mencari *convexity defects* dapat menggunakan fungsi `cv2.convexityDefects(contour, hull_ids)`.

*Convexity defects* diklasifikasikan sebagai celah jari apabila sudut yang dibentuk kurang dari sama dengan 90° dan memiliki jarak dengan garis *convex hull* lebih dari 30 *pixel*. Setelah mendapatkan *convexity defects* yang merupakan celah jari, kita telah mendapatkan jumlah jari dari gestur tangan.



Gambar 18. Hasil Pencarian Penafsiran Hand Tracking

```

def hand_tracking(image):
    # hand detection
    skin_image = skin_segmentation(image)

    # find contours and change to points
    contours, _ = cv2.findContours(skin_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contours = sorted(contours, key=cv2.contourArea, reverse=True)
    points = cv2.contour_to_points(contours)

    # find convex hull
    hull = cv2.convex_hull.convex_hull(points)
    hull = points_to_cv2_hull(hull)

    # find convexity defects
    contour0 = contours[0]
    hull_indices = points_to_indices(hull[0], contour0)
    defects = cv2.convexityDefects(contour0, hull_indices)

    # draw contours, convex hull, and convexity defects
    green_color = (0, 255, 0) # green_color for contours
    blue_color = (255, 0, 0) # blue_color for convex hull
    red_color = (0, 0, 255) # red_color for convexity defects

    cv2.drawContours(image, contours, 0, green_color, 5, 8)
    cv2.drawContours(image, hull, 0, blue_color, 5, 8)

    finger_count = 1
    for i in range(defects.shape[0]):
        start_id, end_id, far_id, d = defects[i, 0]

        # length of triangle sides
        a = math.sqrt((end[0] - start[0])**2 + (end[1] - start[1])**2)
        b = math.sqrt((far[0] - start[0])**2 + (far[1] - start[1])**2)
        c = math.sqrt((end[0] - far[0])**2 + (end[1] - far[1])**2)

        # calculate distance and angle
        s = (a+b+c)/2
        ar = math.sqrt(s*(s-a)*(s-b)*(s-c))
        d = (2*ar)/a
        angle = math.acos((b**2 + c**2 - a**2)/(2*b*c)) * 180 / math.pi



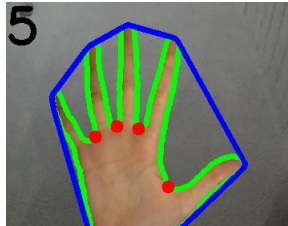
        # ignore defects that are not between two fingers
        if (angle <= 90 and d > 30):
            cv2.circle(image, far, 8, red_color, -1)
            finger_count += 1

    # put text
    cv2.putText(image, str(finger_count), (0, 50), cv2.FONT_HERSHEY_SIMPLEX, 2, 2, 5)

    return image

```

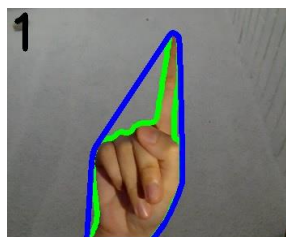
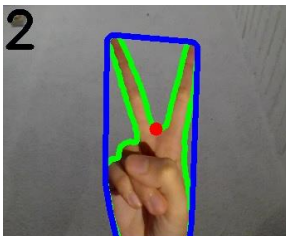
Gambar 19. Implementasi Kode Fungsi Hand Tracking

	3
	4
	5

Tabel 1. Hasil Uji Coba Program

E. Hasil Penelitian

Uji coba dilakukan dengan menggunakan kode yang telah diimplementasikan. Uji coba ini bertujuan untuk mengetahui jumlah jari dari gestur tangan secara *real-time* menggunakan kamera. Uji coba dilakukan pada tempat yang terang dan *background* yang berbeda dengan warna kulit.

Hasil Ujicoba	Jumlah Jari
	1
	2

IV. KESIMPULAN DAN SARAN

Aplikasi algoritma *divide and conquer* pada *hand tracking* sederhana terbukti akurat mendeteksi jumlah jari dari gestur tangan. Penerapan algoritma *divide and conquer* untuk mencari *convex hull* melalui pemecahan masalah menjadi langkah-langkah yang lebih kecil menyebabkan kompleksitas pelacakan tangan dapat dikurangi. *Hand tracker* ini terbukti dapat mendeteksi gestur tangan dengan efektif dan akurat.

Untuk penelitian selanjutnya, disarankan untuk meningkatkan akurasi deteksi citra tangan sehingga dapat mendeteksi tangan dengan baik walaupun berada pada keadaan remang-remang. Selain itu penafsiran *hand tracking* dapat dikembangkan menjadi lebih kompleks lagi untuk menghasilkan interaksi manusia dan komputer yang lebih baik lagi.

LINK REPOSITORY GITHUB

<https://github.com/althaafka/Hand-Tracking-Sederhana.git>

LINK YOUTUBE

<https://youtu.be/Z4XxsnMIIz4>

UCAPAN TERIMAKASIH

Dengan penuh rasa syukur, penulis ingin mengucapkan terima kasih kepada Allah Swt. atas berkat dan rahmat-Nya yang memungkinkan penulis menyelesaikan makalah berjudul

“Aplikasi Penerapan Algoritma Divide and Conquer pada Hand Tracking Sederhana”. Makalah ini disusun sebagai bagian dari tugas akhir Semester IV dalam mata kuliah Strategi Algoritma IF2211.

Penyusunan makalah ini tidak akan berhasil tanpa bantuan, arahan, dan dukungan dari berbagai pihak. Oleh karena itu, penulis ingin mengucapkan terima kasih kepada orang tua penulis serta kepada Ibu Dr. Nur Ulfa Maulidevi, S. T, M.Sc. selaku dosen pengajar maka kuliah Strategi Algoritma IF2211, atas bantuan dan bimbingannya.

#### REFERENCES

- [1] Munir, Rinaldi. 2022. *Algoritma Divide and Conquer (Bagian 1)*. Diakses pada 21 Mei pukul 20.34 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian1.pdf)
- [2] Munir, Rinaldi. 2022. *Algoritma Divide and Conquer (Bagian 2)*. Diakses pada 21 Mei pukul 20.40 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf)
- [3] Munir, Rinaldi. 2022. *Algoritma Divide and Conquer (Bagian 3)*. Diakses pada 21 Mei pukul 20.55 dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf)
- [4] Munir, Rinaldi. 2022. *Algoritma Divide and Conquer (Bagian 3)*. Diakses pada 21 Mei pukul 21.35 dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-\(2021\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf)

- [5] Yunita, H., & Setyati E. 2019. *Hand Gesture Recognition Sebagai Pengganti Mouse Komputer Menggunakan Kamera*. Jurnal ELTIKOM: Jurnal Teknik Elektro, Teknologi Informasi dan Komputer. <https://doi.org/10.31961/eltikom.v3i2.114>
- [6] Yudhistiro, Kukuh. 2018. Algoritma Convex Hull dan Freeman Chain Code pada Visual Hand Tracking. Seminar Nasional Sistem Informasi (SENASIF), 2(1), 1005-1010. Retrieved from: <https://jurnalfti.unmer.ac.id/index.php/senasif/article/view/144>
- [7] <https://nalinc.github.io/> diakses pada 21 Mei 2022 pukul 18.00 WIB.

#### PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023



Althaaf Khasyi Atisomya/13521130