Reconocimiento de dedos con Matlab

Práctica 2 de Procesado de Imagen y Vídeo

Alejandro Amat
Pablo Díaz
Óscar Pina
2022-2023 Q1
PIV

Planteamiento del Problema y posibles aproximaciones

Partiendo desde la implementación realizada en la primera parte de la práctica donde a partir de imágenes entrantes se establecía una máscara binaria con los píxeles de piel, se procederá intentando establecer el número de dedos que aparecen en esta. Para ello, primero es conveniente tener claro cuál es el problema, dividirlo en partes y barajar distintas aproximaciones que se pueden llevar a cabo.

Nótese que nuestro Input será una máscara binaria donde tendremos '0' en los píxeles detectados como piel y '1' en los complementarios. A partir de aquí se debe ser capaz de entender qué algoritmo usar de cara a poder contabilizar de forma correcta los dedos que aparecen.

El primer planteamiento consiste en computar el centro de gravedad de la mano y de esa forma establecer qué punto es el origen de coordenadas. Teniendo este, a partir de un umbral de distancia se podría calcular cuántos dedos hay. Para esto, lo primero que se debe hacer sería calcular las distancias respectivas de cada píxel de piel al píxel con valor '1' (fondo) más cercano. Matlab proporciona la función 'bwdist(image)' que realiza precisamente la acción mencionada.

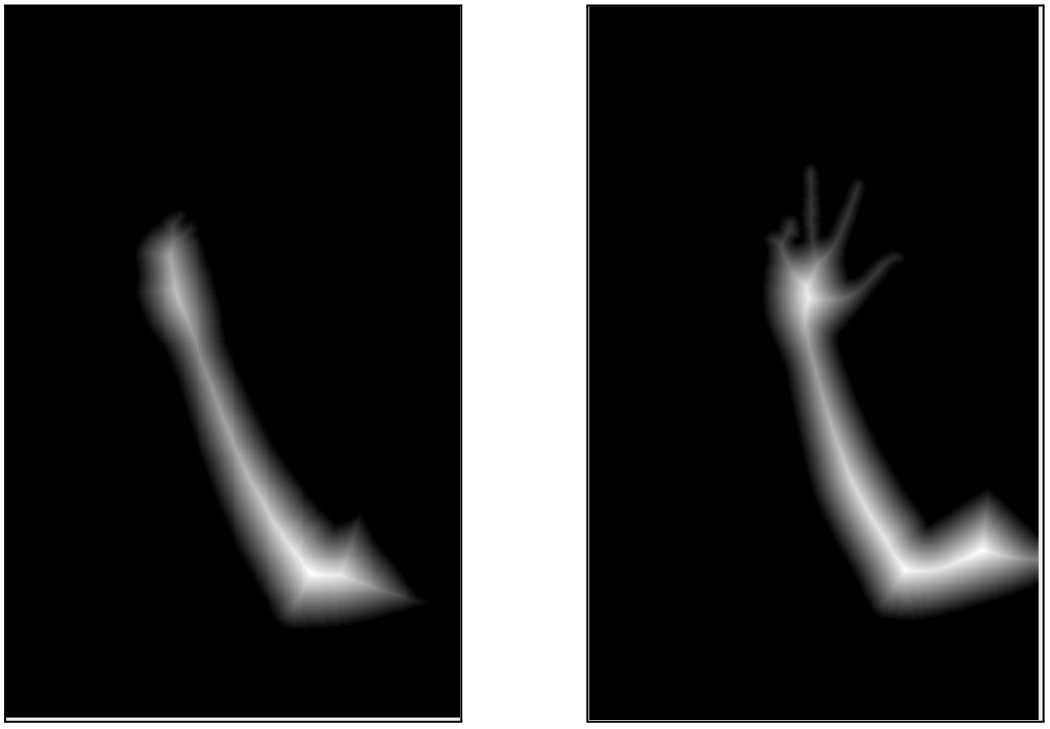


Imagen 1: Cálculo de distancias relativas al fondo de dos imágenes

Tras este paso, se procede a eliminar todas aquellos píxeles que estén a una distancia menor a cierto umbral. De esta forma se puede conseguir eliminar los dedos de la imagen dado que al ser finos estos se encuentran muy cercanos al fondo. Seguidamente, se procedería a calcular el centro de gravedad mediante algún algoritmo no muy complejo. Por ejemplo, la media de la posición de los píxeles de las zonas de piel.

Es aquí donde surge el primer problema de cara a la realización de la práctica: ¿Cómo eliminar el brazo y antebrazo? Esto es crucial ya que afecta directamente al cálculo del centro de gravedad. Si este no se calcula solo teniendo en cuenta la palma de la mano, aparecerá mucho más cerca del brazo y será confuso a la hora de establecer distancias. Por no decir ineficiente.

Tras valorar distintas formas de eliminar el antebrazo, la que más se acercó al resultado fue intentar rotar la imagen de forma que siempre estuviera en vertical. Una vez se obtenía una imagen donde el brazo aparece en vertical se continúa contando fila por fila la cantidad de píxeles blancos en cada una de estas. La muñeca, por lo general, es un punto de inflexión en donde la cantidad de píxeles blancos de esa fila será menor que los de la fila superior e inferior. Por ello, una vez detectemos esta fila, se elimina todo lo que se encuentre abajo.

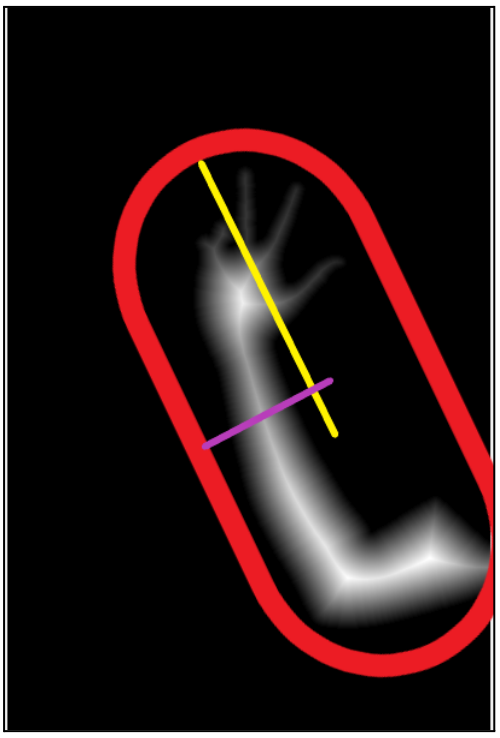


Imagen 2: RegionProps (Orientation)

De nuevo, Matlab proporciona herramientas para esto. Mediante el uso de 'RegionProps' y del atributo 'Orientation', se calcula la elipse que envuelve la imagen dando así forma al eje mayor y menor. De esta forma se puede conseguir la rotación necesaria a aplicar a las imágenes para poder proceder con el cálculo de la posición de la muñeca.

Sin embargo, a pesar del correcto funcionamiento de este algoritmo, alguna de las imágenes nos impedían realizar la medición por el número de píxeles. Esto sucede ya que en la misma fila se solapan píxeles de la muñeca y del brazo, como se muestra en la imagen 3. Esto además hará que los máximos y los mínimos de píxeles vayan creciendo y disminuyendo, haciendo más complicado saber cuál es el correcto.

Se podrían haber contemplado soluciones a esto pero consideramos que todo el proceso estaba siendo demasiado complicado y quizás era mejor cambiar de estrategia. Es por eso que finalmente, tras probar también con técnicas de watershed, se optó por hacer un procesado meramente con elementos morfológicos.

Imagen 3: Imagen donde el brazo se solapará con píxeles de muñeca

Para poder aprovechar el cálculo de distancias relativas al fondo de la imagen, decidimos aplicar un nuevo algoritmo. En este caso, tras eliminar los píxeles más cercanos estos se restan a la imagen original. Obteniendo así los contornos de la imagen deseada. Es desde este punto donde se pueden establecer una serie de elementos para dar forma al algoritmo. La gracia se encuentra en que la imagen restante cuenta (dependiendo del grosor escogido) con los dedos al completo y con un borde de tamaño X en los contornos. Si a este resultado se le aplica un cierre morfológico que elimine el borde pero no los dedos, podremos quedarnos únicamente con estos últimos. En el siguiente apartado se muestra detalladamente el algoritmo utilizado enfatizando en la programación y características principales.

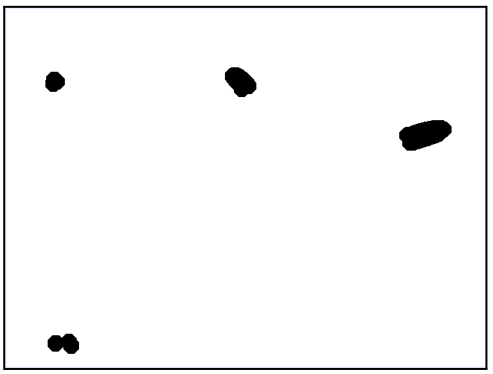


Imagen 4: Resultado inicial después de aplicar cierre morfológico

Programación del algoritmo

Programa principal

El programa principal ha sido dividido en tres bloques; un programa main que prepara el escenario(preprocesado), realiza las inicializaciones pertinentes y llama a las otras funciones, el programa algo6, que aplica diferentes operaciones morfológicas para procesar las imágenes y llegar al escenario de interés y el bloque algo7, donde se aplica un postprocesado para eliminar errores y se decide el número de dedos de la imagen.

El programa main empieza cargando todas las imágenes de prueba en una celda para su posterior manipulación y se inician diferentes parámetros.

Después, se inicializa el bucle principal para procesar todas las imágenes de prueba. Como ya se ha comentado en la introducción, tenemos que trabajar con dos tipos de imágenes muy diferentes a nivel morfológico, imágenes con grosor de brazo finas y gruesas,
lo que nos obliga a diferenciarlas y procesarlas de manera diferente. El elemento principal que diferencia estos dos tipos de imágenes es el grosor, número de píxeles que conforman el brazo y la mano, en relación con el tamaño total de la imágen. Por eso, en el programa main se calcula este ratio para la diferenciación de las imágenes. También se aplica la función distancia a las imágenes y el tamaño de la salida ya que posteriormente se necesitará.

```
image_file = imread(list_images(k).name);
actual_image=1-image_file;
[l,w] = size(actual_image);
grosor_ratio = sum(actual_image(:))/(l*w);
D = bwdist(image_file);
[A,B] = size(D);
```

Imagen 5: Comparativa imágenes con ratio de píxeles pequeño vs grande (0.11 vs 0.38)

Esta diferencia en los ratios se mantiene para todas las imágenes de estos dos grupos.
Una vez hemos caracterizado el grosor de la imagen tratamos las diferentes imágenes de manera no homogénea, se utilizarán elementos estructurantes diferentes y de diferente tamaño para procesar las imágenes. La función main llama a la función algo6 pasándole como parámetro las dimensiones y el resultado de la función distancia aplicada a la imagen, el threshold de la distancia que se utilizará para quedarnos con los bordes de la imagen, el tamaño del elemento estructurante para las operaciones morfológicas y un parámetro que nos indica si la imagen a procesar es "fina"(parámetro vale 1) o si es gruesa (valor 0).

```
if(grosor_ratio>= 0.17)
   if(grosor_ratio >=0.38)
      [CC, im_result] = algo6(A,B,D,image_file, 27.5, 17, 0);
   elseif(grosor_ratio >=0.28)
      [CC, im_result] = algo6(A,B,D,image_file, 17.5, 11, 0);
   else
      [CC, im_result] = algo6(A,B,D,image_file, 19.5, 12, 0);
   end
   [num] = algo7(CC, im_result);
else
   [CC, im_result] = algo6(A,B,D,image_file, 10.5, 6, 1);
   num = if_zero(CC);
end
```

Podemos observar cómo, además de llamar a la función algo6 también se aplica el postprocesado con la función algo7 y con if_zero(), función que se aplica para eliminar pequeñas marcas negras que pueden ser consideradas dedos. Además, también se percibe como sub separamos las imágenes gruesas en dos, gruesas o muy gruesas. Esto se hace porque nos permite obtener mejores resultados.

Finalmente, las funciones de postprocesado nos devuelven el número de dedos que se han detectado. De esta manera podemos calcular el F-score y evaluar la eficiencia del sistema, se explicará este proceso más detalladamente en apartados posteriores. También utilizamos el programa principal para calcular el tiempo de ejecución.

Generación de bordes y primer operador morfológico

Una vez hecho el preprocesado y las debidas inicializaciones a través de la función algo6 realizamos el procesado principal. A esta función le pasamos como argumento las dimensiones de la imagen distancia, la imagen de la distancia, la imagen a procesar, un threshold de la distancia de los pixeles(para calcular el contorno), la dimensión del elemento estructurante que se utilizará posteriormente y un atributo que nos indica si la imagen a tratar es "estrecha", basándonos en la definición que ya hemos explicado.
Para quedarnos con el contorno del brazo y la mano, primero aplicamos un bucle en el que establecemos que los píxeles que tengan una distancia inferior al threshold de distancia pasado como parámetro tengan valor 1 y los pixeles con mayor valor de distancia del threshold tengan valor 0. Esto hará que obtengamos una imagen igual en formas que la imagen original pero de tamaño reducido. Posteriormente, se resta la imagen original con la imagen reducida obtenida y ,de esta forma, obtenemos una aproximación del contorno y borde de la imagen.

```
function [CC, im_result] = algo6(A,B,D, image_file, border, SE_size, thin)
   %Nos quedamos solo con los elementos más cercanos al borde
   for i = 1 : A
      for j = 1 : B
         if(D(i,j)<border)
           D(i,j) =1;
         else
            D(i,j) =0;
         end
      end
   end

   %Restamos a la imagen original.
   im = image_file -D;
```

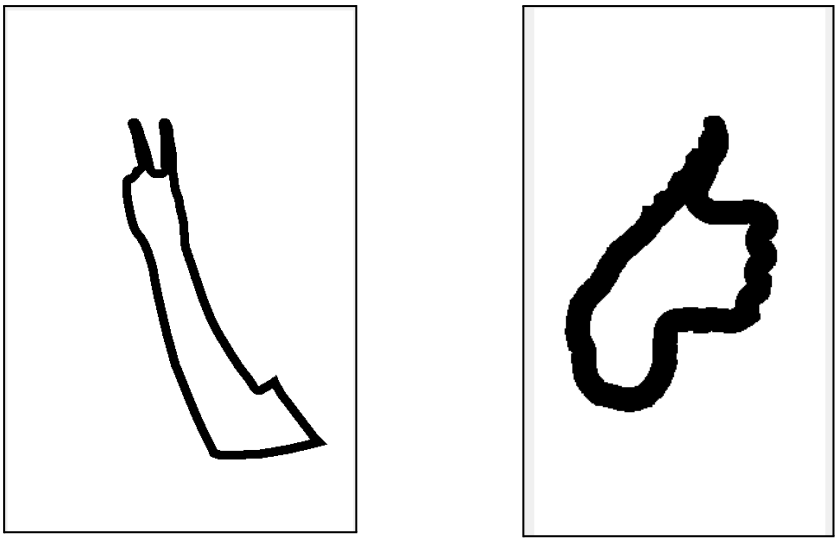


Imagen 6: Comparativa de la Estimación del contorno y bordes de imágenes de tipo I y II.

Una vez obtenemos esta imagen del contorno de la imagen, se puede percibir como en todas ellas, los dedos presentan un mayor grosor(tiene mayor numero de pixeles negros) que el resto del contorno del brazo y mano. Aprovechando esto, decidimos hacer un cierre morfológico con un disco como elemento estructurante y cuyo tamaño se pasa como parámetro en la función. En las imágenes más gruesas este elemento será de mayor tamaño que en las imágenes finas.

```
SE = strel('disk', SE_size);
im_result = imclose (im, SE);
```

Al hacer esta operación percibimos que en las imágenes gruesas funciona bastante bien, separado de manera muy clara visualmente los dedos. Con las imágenes finas, si no tienen muchos dedos funciona pero percibimos que no es capaz de separar bien los dedos contiguos, cuando la mano enseña dos o más dedos.

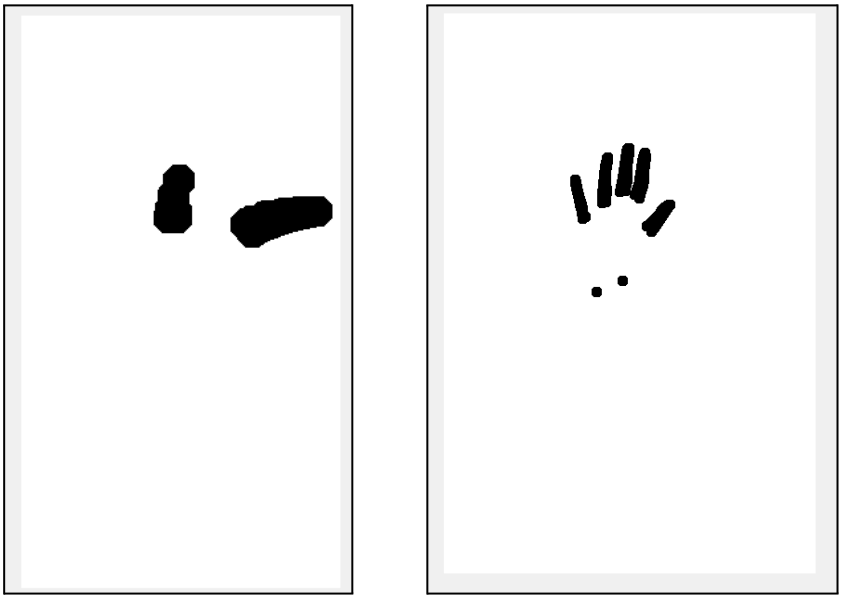


Imagen 7: Comparativa de Segmentación de los dedos de imágenes tipo I y II

Podemos ver como en la imagen 9, aparte de cierto ruido que será descartado en el postprocesado, hay dedos contiguos que se juntan en una única región y no conseguimos separarlos bien. Por este motivo, se ha optado por aplicar a las imágenes finas otro cierre morfológico utilizando como elemento estructural una recta de tamaño 22 píxeles y con inclinación de 88º con respecto a la horizontal. Como los dedos suelen tener la forma de una recta con grosor y cierta inclinación, este elemento estructural consigue solventar de manera muy positiva el problema de la unión de dedos contiguos.

```
 if(thin ==1)
     SE_th = strel('line', 21, 88);
     im_result = imclose(im_result, SE_th);
  end
CC = bwconncomp(im_result);
```

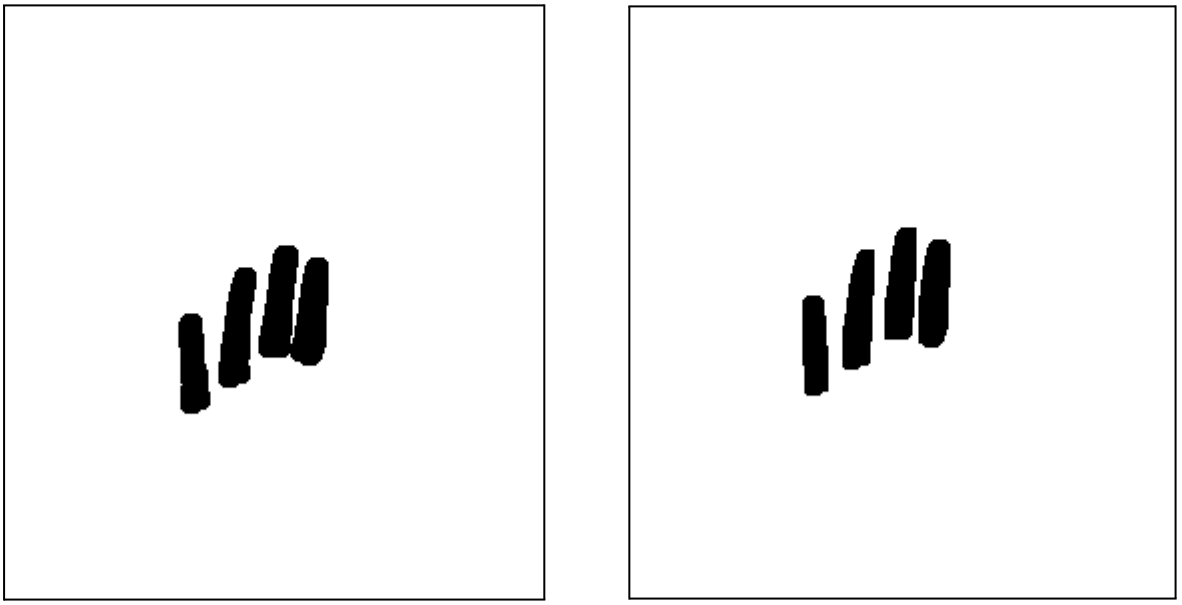


Imagen 8: Segmentación antes del cierre con la recta vs después

Podemos observar con las imágenes 10 y 11 como al realizar dicho cierre con la recta dos dedos que pertenecían a la misma región conexa se han separado en dos regiones y todos los dedos se han separado más. Con este resultado podemos percibir la efectividad de esta operación morfológica en las imágenes para segmentar los dedos.

Finalmente, usando la función de Matlab *bwconncomp()* extraemos las regiones conexas de la imagen así como el número total de estas. El algoritmo devuelve la imagen con la segmentación de los dedos(la imagen original tratada con el procesado) así como las regiones conexas. Posteriormente se realizará un postprocesado para limpiar y optimizar los resultados.

Cálculo de Distancias y tamaños relativos

Como se observa en la imagen 4 y se ha comentado en los apartados previos, después de aplicar el cierre morfológico inicial aparecen muchos elementos conexos y disjuntos que claramente no pertenecen a dedos de la mano.

Esto suele pasar en las imágenes muy cercanas a la cámara donde los dedos son de mayor tamaño y, por ende, el borde de detección debe ser mayor para incluirlos enteros. Esto lleva a que los bordes de los contornos del brazo también se vuelven mayores. Tanto que, al eliminarlos, también eliminamos los elementos que sí son dedos. Esto es un error muy recurrente en nuestro algoritmo pero se puede intentar solventar teniendo en cuenta el tamaño y las distancias de estos conjuntos conexos y caracterizarlos.

El algoritmo solo se aplicará a las imágenes con un ratio de píxeles mayor a 0.17. Ya que, como se ha mencionado previamente, en las otras ya existe otro postprocesado y ya se eliminan los pequeños elementos resultantes.

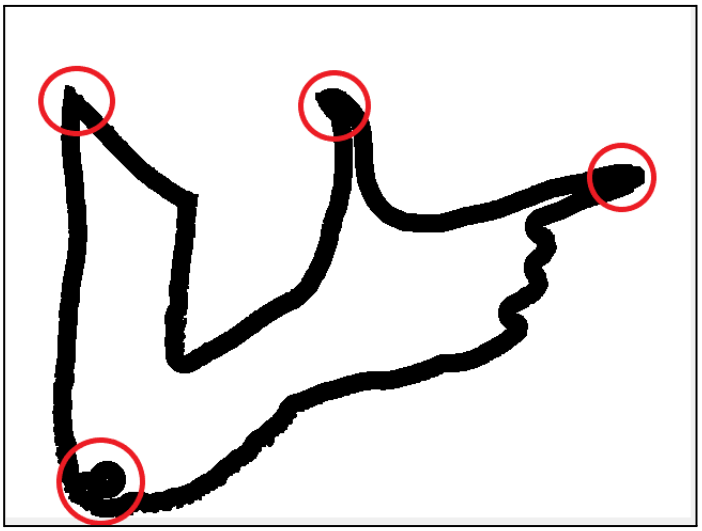


Imagen 9: Elementos detectados como dedos

Para crear el algoritmo que tenga en cuenta las distancias y tamaños es necesario pasar como parámetro el CC (conncomp) que se calcula en el algoritmo previo. Este incluye:

- Conectividad
- Tamaño de la imagen
- Número de objetos conexos
- Píxeles de cada elemento conexo

A partir de este punto, el criterio que se decidió seguir fue usar el elemento de mayor tamaño como referencia. Ya que, a priori, este siempre será un dedo. Los únicos casos donde no sería correcto asumir esto es cuando la imagen es 0. Sin embargo, todos estos casos corresponden con índices de ratio de píxeles inferiores a 0.17 (Primer tipo de imágenes).

Por tanto, para encontrar el máximo:

```
for i = 1 : CC.NumObjects
      [sizes, w] = size(CC.PixelIdxList{i});
      if(sizes>max_size)
         max_size = sizes;
         index_max = i;
      end
   end
```

De esta forma ya obtenemos el máximo tamaño de los elementos y almacenamos el índice del que se trata. Una vez en este punto, es necesario conocer la ubicación del elemento de tamaño máximo, para poder compararlo. Para hacer esto nos quedamos con el número de píxel que se encuentra en el primer valor del vector de píxeles. Sin embargo, el número de píxel que se otorga es el total y no en 'x' e 'y'. Por ello hay que pasarlo:

```
if(num~=0)
      reference_position_max = CC.PixelIdxList{index_max}(1);
      ref_max_x= mod(reference_position_max, l);
      if(ref_max_x==0)
         ref_max_x = l;
      end
      ref_max_y =  (reference_position_max - ref_max_x)/ l ;

   end
```

Para ello obtenemos el valor del píxel del elemento mayor, como comentábamos. Este valor indica la posición teniendo en cuenta que la esquina izquierda superior es 0 y va sumando uno progresivamente. Por ello, 'x' será este valor módulo 'ancho de la foto'. La 'y' es este valor dividido por el ancho de la foto. Antes de eso, le restamos 'x' para que nos dé natural, ya que sino es decimal.

Finalmente, simplemente iteramos por todos los elementos y calculamos la distancia con respecto al elemento referente. Si la distancia es mayor a 400, se agregará esté elemento a una lista que se tendrá en cuenta más tarde. La distancia se ha calculado como abs(1.x -2.x) + abs(1.y - 2-y); Sin tener en cuenta los cuadrados y raíz de la norma. Se almacenan todas las distancias.

```
%Define las distancias entre el dedo más grande y los demás elementos
%(que pueden ser, o no, dedos)

   for i = 1 : CC.NumObjects
      if(i ~= index_max)
         reference_position = CC.PixelIdxList{i}(1);
         ref_x= mod(reference_position, l);
         if(ref_x==0)
            ref_x = l;
         end
         ref_y =  (reference_position - ref_x)/ l ;

         dist = abs(ref_y - ref_max_y) + abs(ref_x - ref_max_x);
         if(dist>400)
            indices_not_counted(i) = i;
         end
         distances(i)= dist;
      end
    end
```

En caso de que el elemento esté a menos de 400 píxeles de distancia se aplicará una condición laxa y en el otro caso será mucho más respectivo

```
[sizes, w] = size(CC.PixelIdxList{i});
      if(sum(indices_not_counted==i)==0)
         if((sizes < max_size * 0.30 && distances(i)>250) || (distances (i) <250 && sizes <
max_size * 0.12))
            num = num -1;
```

Como se observa, si el elemento está más cerca de 400 píxeles, se caracterizará en función de dos parejas de restricciones. Se considerará que no es dedo si:

- Está a más de 250 píxeles y es menor a 0.3 veces el tamaño máximo
- Está a menos de 250 píxeles y es menor a 0.12 veces el tamaño máximo

Para los que estaban tan alejados se considera lo siguiente:

- Es menor a 0.47 el tamaño máximo

Es por ello que es muy fácil que no acepte los elementos que se encuentren tan distantes. Esa es la intención, ya que por lo general no son dedos.

Es cierto que al calcular solamente la distancia respecto a un solo dedo pueden salir cosas con poco sentido, pero calcular la distancia entre todos los elementos sin saber siquiera cuáles son los dedos era más arriesgado.

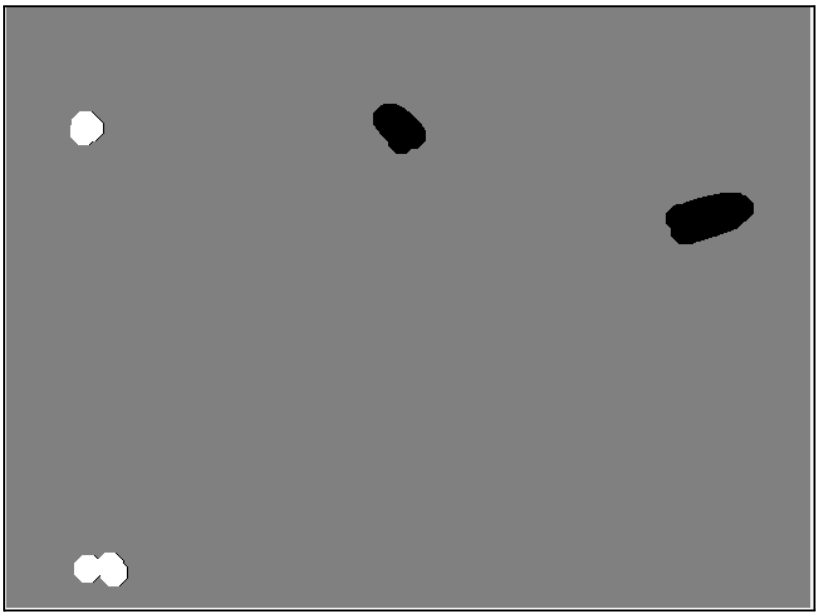


Imagen 10: Resultado después de aplicar algoritmo

Resultados y Análisis

Resultados Analíticos

Para poder comprobar nuestros resultados hemos establecido dos criterios en cuanto a rendimiento de precisión. En primera instancia hemos comprobado qué porcentaje de dedos hemos acertado respecto al número de dedos totales. Es decir, si el número a obtener es 4 y nuestro algoritmo detecta 3, nuestra precisión es del 75%. En los casos donde se detectan más de los que deberían, se hace de forma inversa. Por otro lado, también hemos establecido el criterio de paridad. Si el número es el indicado, el resultado será positivo. En caso contrario, falso.

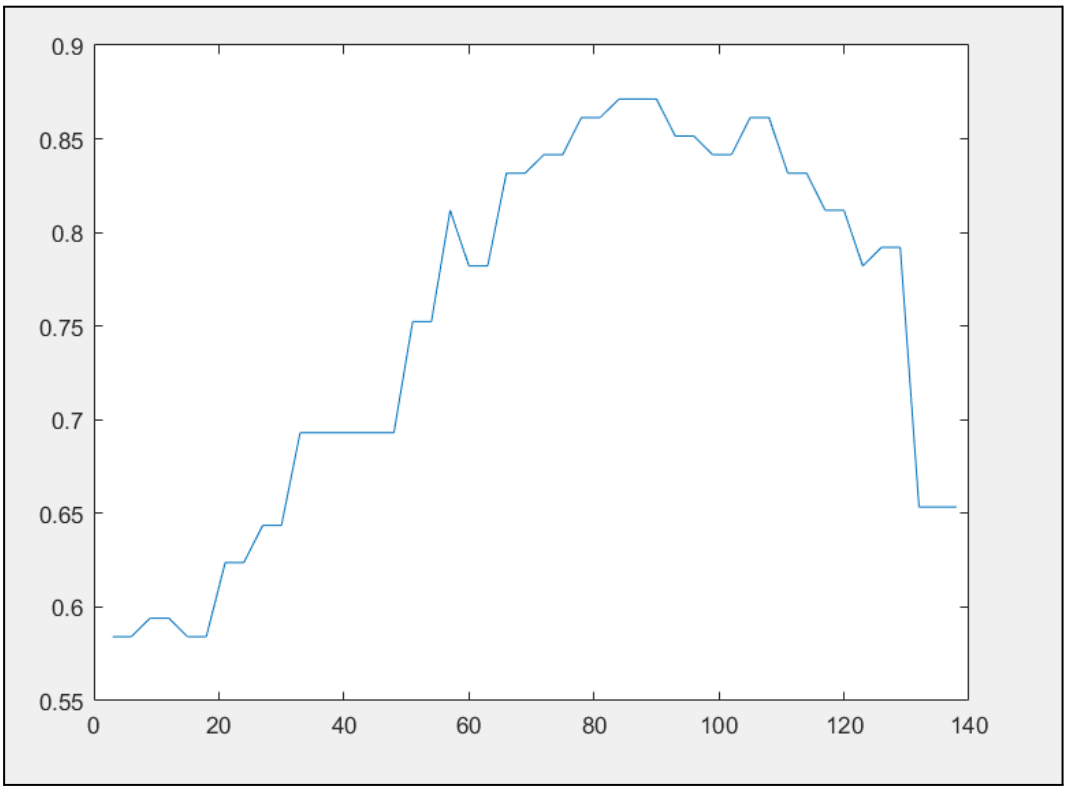
Por tanto, hemos optimizado en función de estos dos porcentajes. Uno nos indicará qué tan cerca calculamos el número de dedos, sin penalizar el no haber acertado. El otro, simplemente suma cuando es estrictamente correcto. Es por esto, que siempre hemos tenido este último en cuenta antes.

Para optimizar el resultado se ha modificado el umbral del grosor y el tamaño del elemento estructurante aplicado en cada caso. Además del cierre morfológico aplicado en las imágenes con ratio de píxel menores a 0.17.

```
if(grosor_ratio>= 0.17)
      if(grosor_ratio >=0.38)
         [CC, im_result] = algo6(A,B,D,image_file, 27.5, 17, 0);
      elseif(grosor_ratio >=0.28)
         [CC, im_result] = algo6(A,B,D,image_file, 17.5, 11, 0);
      else
         [CC, im_result] = algo6(A,B,D,image_file, 19.5, 12, 0);
      end
      [num] = algo7(CC, im_result);
      %disp("Obtained: " + num);
   else
      [CC, im_result] = algo6(A,B,D,image_file, 10.5, 6, 1);
      num = if_zero(CC);
```

Tras diversas iteraciones se ha decidido usar los valores mostrados en el código. Destacar que a pesar de contar con 8 variables a optimizar no es necesario hacer un bucle que incorpore todas (con un coste computacional de $O(n^8)$) sino que se puede hacer de dos en dos ya que las parejas solo se afectan entre ellas y no interfieren con el resto de imágenes.

Tras esto, un valor muy importante a tener en cuenta ha sido el elemento estructurante aplicado a las imágenes de ratio < 0.17. Estas, al ser muy finas y con dedos juntos, presentaban muchos problemas. Es por eso, que como se ha mencionado, se utilizó un elemento con forma de recta y con una orientación de 88 grados (parecida a la que presentan los dedos en este tipo de imágenes).



Gráfica 1: Porcentaje aciertos en función del grado inclinación recta

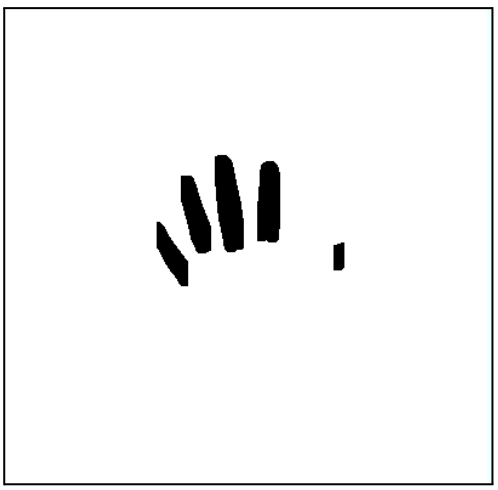


Imagen 11: Imagen con ratio < 0.17 después del cierre con el elemento de recta

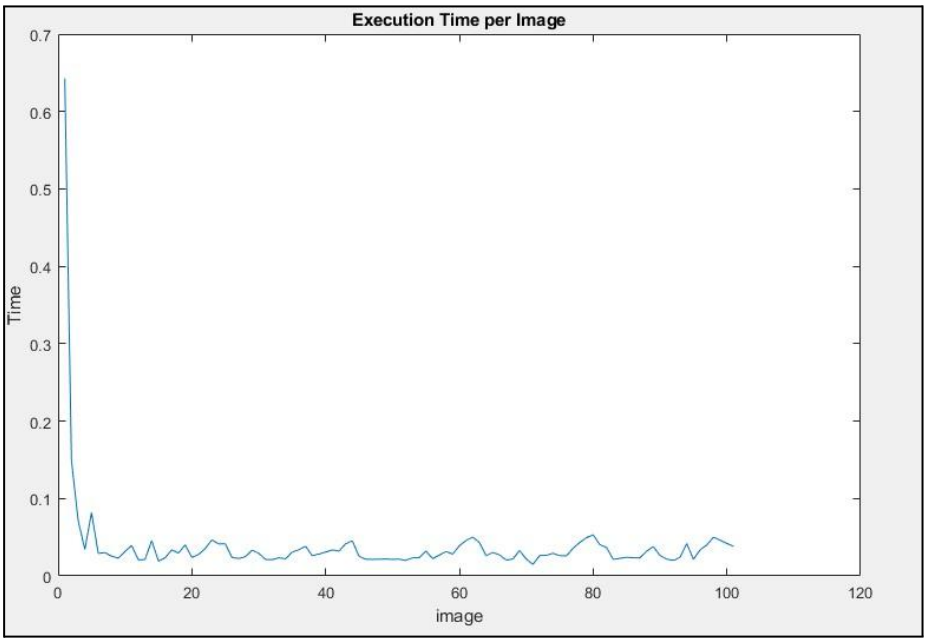
Finalmente, tras todas estas consideraciones y optimizaciones se han obtenido los siguientes dos resultados:

- **Porcentaje de similitud : 94.62 %**
- **Porcentaje acierto : 87.129%**

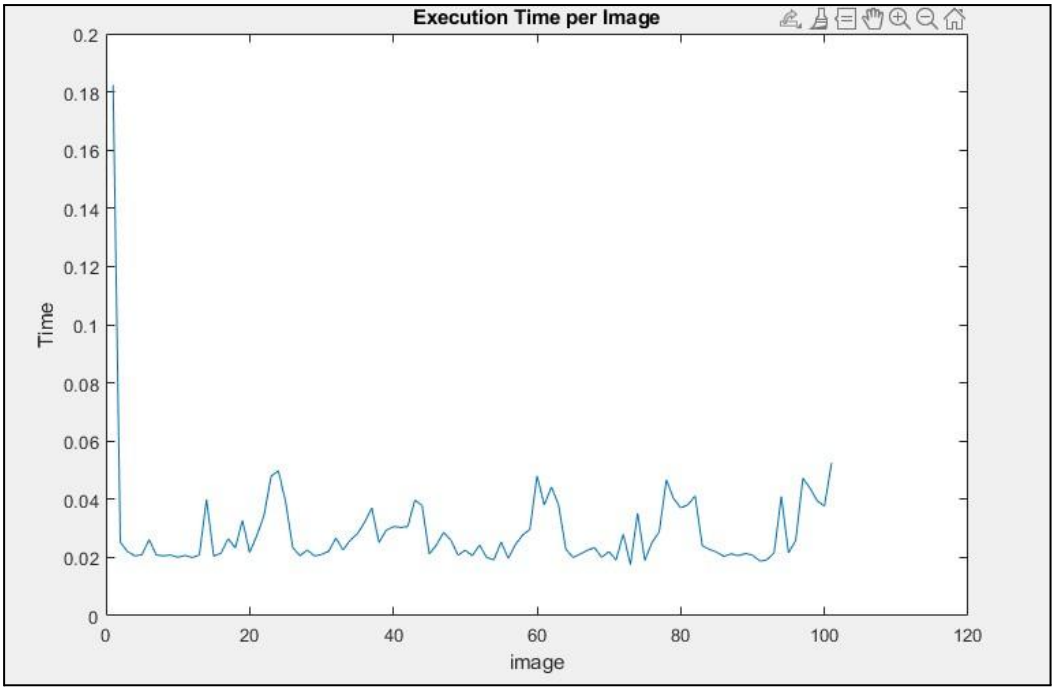
Tiempo de Cálculo

Usamos la función tic y toc de matlab para iniciar un reloj que nos indique el tiempo real de ejecución del algoritmo para cada imagen. Guardamos el resultado en un vector para representar este resultado y calcular la media del tiempo de ejecución.

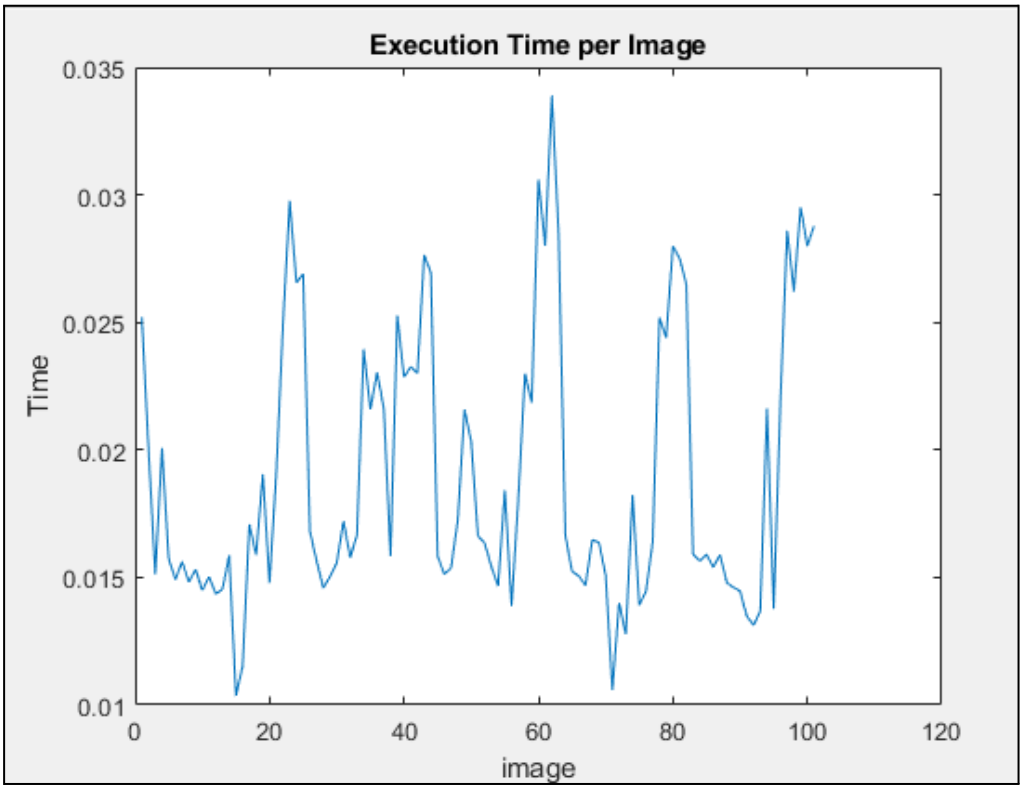
```
T = zeros(1,length(list_images));
for k = 1 : length(list_images)
   image_file = imread(list_images(k).name);
   tic;
.
.
.
 end
 T(k)=toc;
 figure;
 plot(T);
 title('Execution Time per Image');
 xlabel('image');
 ylabel('Time');
```



Gráfica 2 : Tiempo de ejecución del grupo de imágenes procesadas por primera vez



Gráfica 3 : Tiempo de ejecución del grupo de imágenes con inicialización de parámetros



Gráfica 4 : Tiempo de ejecución del grupo de imágenes procesadas una vez inicializado el escenario

Estas tres gráficas, de tres realizaciones diferentes, representan la forma de los tres tipos de realizaciones de la ejecución de todas las imágenes test. La primera tiene un pico muy notorio al principio, ya que las imágenes se tienen que cargar por completo en la aplicación de matlab. Una vez se cargan y están guardadas las imágenes en matlab, las funciones de ejecución siguen una gráfica muy parecida a la gráfica de la tercera imagen. La segunda imagen también presenta un pico al inicio, ya que dependiendo de la realización del procesado del grupo de imágenes, matlab tiene que inicializar algún parámetro. Lo que hace que se tarde más en procesar la primera imagen. Esta gráfica también representa algunas realizaciones que nos hemos encontrado durante el cálculo del tiempo de ejecución.

Calculamos el tiempo medio de ejecución por realización de todas las imágenes de prueba como la la suma de los tiempos de ejecución de cada imagen divididos por el número de imágenes de trabajo. Posteriormente, calculamos la media del tiempo de ejecución sumando los tiempos de ejecución de 50 realizaciones y dividiéndolos, para conseguir la media, por 50. Pensamos que con 50 ejecuciones de todas las imágenes obtenemos un resultado estadísticamente significativo para nuestro problema.

Tiempo medio de ejecución por realización = te(i) = Σ T(i) /length(list_images) = 0.018978 (como ejemplo)

```
media_tiempos = sum(T)/ length(list_images);
disp("Total Tiempo"+ media_tiempos);
```

Tiempo de ejecución medio = Σ te(i)/N = 0,0191352 s

Este resultado se ha obtenido haciendo la media de muchas realizaciones del proceso aleatorio que es el tiempo de ejecución del algoritmo para cada imagen.

Para calcular estos tiempos de ejecución se ha utilizado un PC con procesador Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, 1992 Mhz, 4 procesadores principales, 8 procesadores lógicos. Con 16GB de memoria RAM y corriendo Windows 11.

Presentación de ideas para mejorar el sistema

Diferentes estrategias y operaciones se podrían implementar para mejorar el algoritmo de detección del número de dedos. Podemos destacar opciones que ya fueron tomadas en cuenta durante el desarrollo del proyecto como la utilización de la transformada watershed para mejorar la segmentación y detección de los dedos o intentar implementar un modelo más geométrico para que tuviese en cuenta la geometría de la mano para detectar los dedos. Usando un enfoque geométrico se podría haber calculado el centro de gravedad de la mano y, a partir de ahí, aprovechando que las pintas de los dedos estarían más distanciadas de este centroide, calcular estos puntos de distancia máxima(picos de una función de distancia al centro de masa).

Aún siendo tomadas en consideración, estas opciones no fueron viables de implementar por los diferentes problemas citados con anterioridad.

Al implementar una solución basada principalmente en morfología matemática, esta está muy sesgada al tipo de imágenes que se utilizaron para el entrenamiento. Además no es muy robusta y presenta diferentes errores que no deberían cometerse. Investigando al respecto de estos errores, la falta de robusteza y, sobre todo, la poca adaptabilidad del algoritmo para otro tipo de imágenes nos dimos cuenta que la mejor manera de abordar este problema sería utilizando herramientas del Deep Learning y redes neuronales para la segmentación y clasificación de los dedos.

Utilizando ideas de Machine Learning clásicas como Máquinas de Vectores de Soporte, Bosques Aleatorios y K-Vecinos más Cercanos nos permitiría contar los números de las manos de manera bastante eficiente clasificando eficientemente las imágenes. Se debe añadir que sería necesaria una base de datos etiquetados extensa para el desarrollo de algoritmos basados en estas técnicas.

Por último, se podrían utilizar técnicas de Deep Learning como Perceptrón Multicapa o redes neuronales convolucionales para clasificar de manera eficiente las diferentes imágenes en las clases de nuestro problema, que serían las clases cada una caracterizando la mano enseñando las diferentes combinaciones de dedos.