Sistema de Extracción de Puntos Claves de la Mano y Conteo de Dedos

Diego F. Díaz[1], Jhon H. Fernández[2],
[1]Dept. Ingeniería Eletrónica, Pontificia Universidad Javeriana, Bogotá D.C., Colombia, di-diego@javeriana.edu.co
[2]Dept. Ingeniería Eletrónica, Pontificia Universidad Javeriana, Bogotá D.C., Colombia, jhon_fernandez@javeriana.edu.co

***Resumen*—En el siguiente documento se presenta un sistema de extracción de puntos clave de mano y conteo de dedos mediante herramientas de procesamiento de imágenes y modelos de deep learning. Los resultados obtenidos para los tiempos de procesamiento fueron los siguientes: 5-10 segundos para CPU en Colab y 0.09-0.012 segundos en la GPU de Colab. Para los resultados gráficos se obtuvo un rendimiento óptimo, con espacio para mejoras. Link del repositorio: https://github.com/JhonHader/HAND-KEYPOINTS-DETECTION.git.**

***Index Terms*—deep learning, puntos clave.**

I. Introducción

El problema consiste en extraer y detectar los puntos clave en la mano y contar los dedos extendidos de esta. Las alternativas existentes consisten principalmente en entrenar modelos de deep learning o utilizar modelos pre-entrenados. Las ventajas que tienen estos modelos frente a técnicas de procesamiento de imágenes es su mejora a la sensibilidad de la iluminación ambiente e incluso en cuanto a tiempos de procesamiento. La solución que se propone es implementar métodos de procesamiento de imágenes y modelos pre-entrenados para la detección de puntos claves y conteo de dedos.

El artículo se divide en una sección de Herramientas Teóricas donde se presentan las principales herramientas de procesamiento de imágenes implementadas en el proyecto, posterior a esto se encuentra la sección de Solución Propuesta donde se presentará en detalle el sistema implementado, en la sección de Resultados se presenta el rendimiento del sistema junto con resultados gráficos, por último se presentan conclusiones y posibles mejoras para un trabajo futuro.

II. Herramientas Teóricas

II-A. Conteo de Dedos

II-A1. Operaciones Morfológicas: Las operaciones morfológicas son operaciones no lineales que se realizan sobre una imagen mediante ventanas, con el fin de disminuir el ruido en una imagen binarizada.

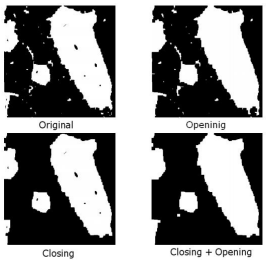


Figura 1: Operaciones morfológicas.

Las principales operaciones son: erosión, dilatación, opening y closing. La **erosión** consiste en realizar una **AND** entre todos los pixeles de la ventana, obteniendo así la expansión del fondo, la frontero del objeto suavizada, y reducción de los objetos pequeños en la ventana. En la **dilatación** se realiza una **OR** entre todos los pixeles de la ventana. En el caso de la dilatación el objeto de interés se expande en tamaño, de nuevo la frontera del objeto se ve suavizada y los huecos pequeños se ven reducidos. Con estas dos operaciones se definen **opening** y **closing**. Opening consiste en realizar una erosión y luego una dilatación, permitiendo eliminar objetos pequeños sin distorsionar objetos grandes. Realizar closing es el proceso contrario, dilatar primero y luego erosionar, de esta manera se rellenan los huecos de los objetos sin distorsionarlos.

II-A2. Sustracción de Fondo: Para la sustracción de fondo se parte del primer fotograma del video, definiendo así nuestro fondo. A partir de esto se procede a restar cada fotograma siguiente con el fondo original, así podemos discriminar cualquier objeto nuevo que este en la imagen.

Figura 2: Sustracción de fondo.

II-A3. Umbralización: La umbralización es el proceso en el cual una imagen en grises se convierte en una imagen binaria, es decir en blanco y negro. La umbralización puede ser global o local. En la umbralización global un único umbral se le asigna a toda la imagen. Para este método existen varias maneras de encontrar el umbral, la gran mayoria basándose

en el histograma del nivel de grises. En particular el método de **Otsu** calcula el umbral minimizando la varianza intraclase entre los pixeles blancos y negros.

II-A4. Extracción de Contornos: Para la extracción de contornos se utilza el método de OpenCv **findCountours**, basado en el algoritmo de 1985 por Satoshi Suzuki.

II-B. Puntos Claves

Se hace uso de una implementación basada en el artículo ***"Hand Keypoints Detection in Single Images using Multiview Bootstrapping"*** [1] el cual basa su método de detección por medio de triangulaciones de una misma escena vista desde diferentes perspectivas, si hay un error en la triangulación de alguna imágen se realiza una reproyección de la misma y se vuelve a triangular hasta obtener la triangulación de todas las imagenes [3].

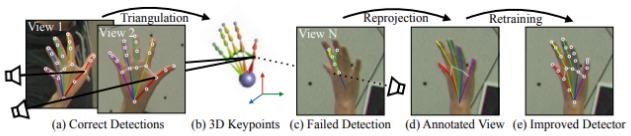


Figure 3: Multiview Bootstrapping. (a) A multiview system provides views of the hand where keypoint detection is easy, which are used to triangulate (b) the 3D position of the keypoints. Difficult views with (c) failed detections can be (d) annotated using the reprojected 3D keypoints, and used to retrain (e) an improved detector that now works on difficult views.

Después de triangular correctamente todas las imagenes se obtiene un mapa espacial de probabilidades con 22 puntos (21 para los puntos claves en la mano y otro más para el fondo) [4]

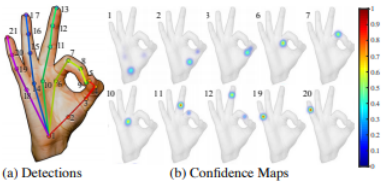
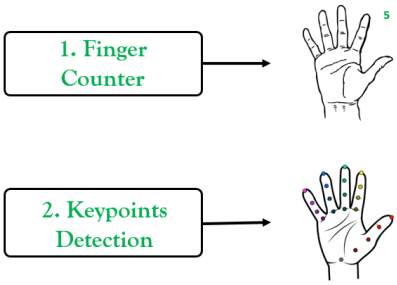


Figure 4: (a) Input image with 21 detected keypoints. (b) Selected confidence maps produced by our detector, visualized as a "jet" colormap overlaid on the input.

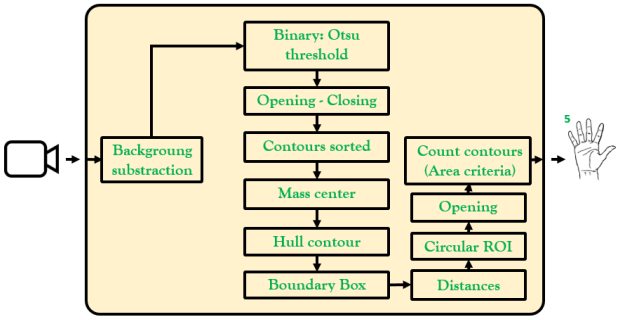
III. Solución Propuesta

La solución propuesta consiste en la implementación de dos subsistemas, uno para conteo de dedos y otro para detección de puntos clave [5]:



III-A. Conteo de dedos [Finger counter]

Este sistema utiliza unicamente técnicas de procesamiento de imágenes. Se define el siguiente algoritmo:



Inicialmente se debe tener fijado un fondo y posterior a esto se realiza el siguiente proceso con imagenes en escala de grises: se realiza una sustracción de fondo para obtener la mano, luego se umbraliza con el criterio de **Otsu**, una vez se obtiene la imagen binaria se realizan operaciones morfológicas sobre esta para filtrar el ruido, a continuación se encuentra el contorno y su respectivo centro de masa, luego se halla el contorno envolvente de la mano y caja delimitadora, posteriormente se calcula la distancia euclidiana desde el centro masa hasta los puntos máximos de la caja delimitadora (arriba $[d_2]$, izquierda $[d_1]$ y derecha $[d_3]$) y se obtiene la distancia máxima $[R]$ de la cual se obtiene una distancia $[r]$ la cual será el radio de la región de interés [ROI], donde dicha región es circular y con un ancho de 6 pixeles. Finalmente se aplican operaciones morfológicas sobre ROI para filtrar ruido y se hace una operación lógica AND con la umbralización de la mano (binaria) para obtener la intercepción entre el circulo y la mano, luego se obtienen los contornos y se considera un contorno o no dependiendo de un criterio de área, en este caso si el área (en pixeles) es mayor a 300 pixeles no se considera como un dedo. Este criterio fue obtenido de manera práctica e iterativa,

De manera gráfica el algoritmo se ilustra en la figura 3.

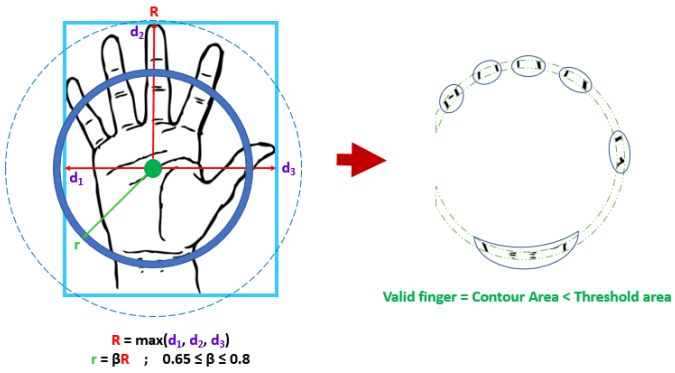
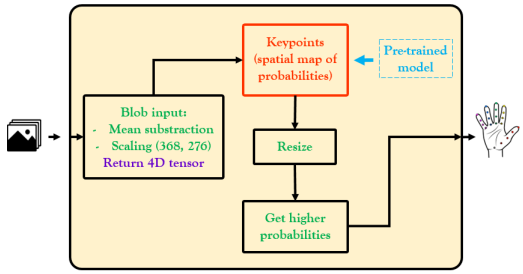


Figura 3: Algoritmo de conteo de dedos

III-B. Detección de puntos claves [Keypoints detection]

Para este sistema de utiliza una implementación de ***OpenCV*** desarrollada con ***dnn (deep neural network)*** [1]

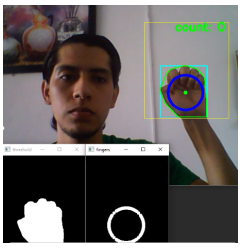


Para el sistema lo primero que se hace es acondicionar las imagenes de entradas a las especificaciones dadas por la implementación, la entrada debe ser pasada por una función ***blob*** que consta en reajustar la imagen a un tamaño de 368x276 y se hace una substracción con la media para cada canal, de esta función se obtiene un tensor en 4D $[1, 3, 368, 276]$, posteriormente se pasa por el modelo de detección de puntos claves del cual se obtiene un mapa espacial de probabilidades para los puntos claves, esto se reajusta al tamaño original de la imágen y se obtienen los puntos de probabilidad más altos.

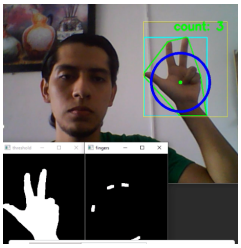
Este modelo se ejecuta en Google Colab donde se puede hacer uso de hardware como GPU de manera gratuita, esto se realiza con el fin de obtener tiempos de procesamiento muy pequeños (para una futura implementación en tiempo real).

IV. Resultados

A continuación se presentan los resultados gráficos para el conteo de dedos y detección de puntos clave.
En las figuras 4a, 4b y 5 se muestra, por una lado, la imagen binaria de la mano, esto es después de realizada la sustracción de fondo, por otro se muestra la región de interés circular y por último el resultado del algoritmo.



(a) Conteo de dedos (0 dedos).



(b) Conteo de dedos (3 dedos).

Figura 4: Conteo de dedos.

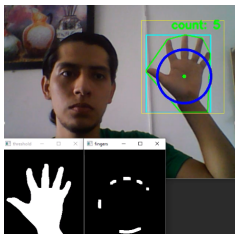


Figura 5: Conteo de dedos (5 dedos).

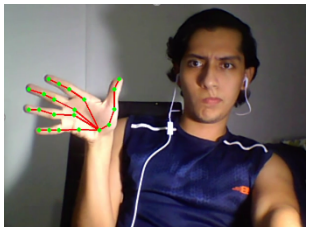
En la cuadro I se observa el registro de los tiempos de procesamiento para un fotograma en la CPU de Colab y con la GPU para la detección y extracción de puntos claves en la mano. A partir de estos resultados se puede llegar a estimar el fluídez (fps) de un video para su procesamiento en tiempo real.

Procesador	Tiempo por fotograma [s]
CPU Colab	5-10
GPU Colab	0.09-0.12

Cuadro I: Comparación del tiempo de procesamiento.

Para la detección de puntos claves de la mano se obtuvieron los siguientes resultados:

(a) Fotograma con mano a procesar.



(b) Fotograma con *keypoints* obtenidos.

Figura 6: Puntos claves de la mano.

V. Conclusiones

- Con el tiempo de procesamiento de la GPU de Google Colab se puede llegar a procesar 9 frames por segundo.
- Una posible mejora para el algoritmo de conteo de dedos es entrenar un modelo de deep learning e implementarlo en tiempo real, ya que las herramientas de procesamiento de imágenes consumen más tiempo de procesamiento.
- Implementar filtros o métodos para suavizar el movimiento, por ejemplo implementando un filtro tipo Savitzky-Golay.
- Para la implementación en tiempo real se propone migrar C++ o implementar en JavaScript, esto con el fin de disminuir el tiempo de procesamiento sin GPU para C++ y tener compatibilidad con la nube para el caso de JavaScript.

Referencias

[1] Hand Keypoints Detection in Single Images using Multiview Bootstrapping. Simon, Joo, Matthews, Sheikh. Carnegie Mellon University. Took from: https://arxiv.org/pdf/1704.07809.pdf

[2] Deep Learning + Estimación de postura de la Mano. Carlos Julio Pardo. Vision por computador. Took from: https://carlosjuliopardoblog.wordpress.com/2019/05/24/deep-learning-estimacion-de-postura-de-la-mano/
[3] Finding extreme points in contours with OpenCV. Adrian Rosebrock. Pyimagesearch. Took from: https://www.pyimagesearch.com/2016/04/11/finding-extreme-points-in-contours-with-opencv/