Seguimiento de Color en Imágenes mediante Cámara Web

Calle, Diego Arturo
dnetix@gmail.com
Universidad de Antioquia

Resumen—Este artículo se espera explicar el proceso mediante el cual se consigue que un programa rastree un color el cual puede ser dado o seleccionado por el usuario en una imagen, brindando la posición y el área cubierta del mismo, pero sin limitarse a los 3 colores primarios Rojo, Verde y Azul.

Índice de Términos—Procesamiento Digital de Imágenes, Seguimiento de Color, RGB, MATLAB

I. INTRODUCCIÓN

Dado que las computadoras se han hecho completamente necesarias en nuestra vida cotidiana se han buscado diferentes formas de interactuar con ellas, en especial de una forma más "humana" que un teclado y un mouse, usando un dispositivo común como la cámara web se puede realizar interacciones tan diversas como el entretenimiento humano, la medicina, la automatización de tareas repetitivas, mejorar la accesibilidad para personas con discapacidades.

En este artículo se introduce el algoritmo para el seguimiento de color mediante que se usará en un simple juego en el cual a través de la imagen obtenida por la cámara web se obtiene la posición del color buscado y se pueden estallar burbujas que descienden por la pantalla cuando son intersectadas.

II. ESTADO DEL ARTE

Actualmente en el mercado hay tecnologías que explotan de una forma más eficiente, aunque también más costosa y no de tanto alcance, el procesamiento digital de imágenes para que un usuario pueda interactuar con un equipo, tales como el Wii Remote de Nintendo® o el Kinect[1] de Microsoft® [2].

También se han explorado tecnologías más económicas para el modelado 3d usando por ejemplos guantes de color[3] los cuales mediante la identificación del color y la ubicación del mismo es capaz de calcular la posición de la mano y con esto permitir una interacción del usuario con una máquina.

Cabe notar también que se han hecho esfuerzos en el análisis de rasgos faciales como el movimiento de los labios cuando una persona está hablando [3] con el fin de conocer lo que está hablando sin necesidad de un micrófono.

III. TEORIA DEL COLOR EN COMPUTADORES

Para el desarrollo del proyecto es importante conocer como un computador procesa una imagen, de manera que se pueda descomponer en los componentes necesarios para este articulo nos basaremos en el modelo de color RGB (Rojo, Verde, Azul) mediante el cual un monitor nos hace percibir los colores.

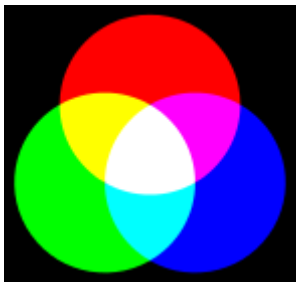

Figura 1. Modelo de color RGB

Al tratarse de un sistema aditivo de color, cuando no es enviado ningún color percibimos el negro, todos los colores son expresados en combinaciones de valores que oscilan entre 0 y 255 para cada componente, cuando se combinan todos percibimos el color blanco.

Cada imagen que se encuentra en un computador puede descomponerse en unas coordenadas (x) y (y) que nos permiten saber la posición del pixel en cuestión y de cada pixel, otras 3 capas de color identificadas como 1 para rojo, 2 para verde y 3 para azul, con los valores de cada componente de color. Por ejemplo si tomáramos el pixel en la posición (1, 1) de la figura 1, y obtuviéramos sus componentes de color encontraríamos un vector [0, 0, 0] ya que al ser de color negro carece de cualquier valor en cada componente de color.

IV. EXPERIMENTACIÓN

El desarrollo del algoritmo puede descomponerse en 6 etapas principales que se describen a continuación, este experimento es desarrollado en MathWorks MATLAB® que nos permite de una manera rápida la prueba del concepto.

1. Adquisición de la imagen ***I(x, y, c)*** y descomposición en una matriz de componentes de color. ***R(x, y) = A(x, y, 1), G(x, y) = I(x, y, 2), B(x, y) = I(x, y, 3)***

En realidad el origen de la imagen es irrelevante lo que es realmente importante es que se obtenga la información de la imagen como una matriz de color con sus componentes RGB el cual nos brinda MATLAB mediante la función "**imread**" para el fin de la prueba del concepto usado cargaremos una imagen que paso a paso se descompondrá hasta que obtengamos la información de la ubicación del color buscado.

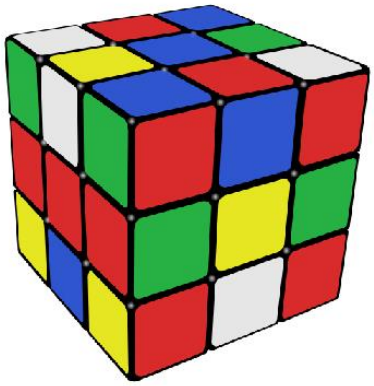

Figura 2. Imagen de ejemplo para la prueba del concepto

2. Selección del color que se desea filtrar. C(r, g, b)

La selección del pixel en MATLAB se realiza con la función "**impixel**" para efectos del ejemplo se seleccionó un pixel del color verde del cubo de Rubik que en sus componentes RGB es [39, 177, 68]

3. Filtrado del color para obtener una matriz binaria

Hay muchas formas de realizar este efecto, el objetivo es obtener una nueva matriz binaria la cual deberá tener valores 0 en donde no aplica el color que se busca e unos donde si se presenta este color, definiendo como color aquel pixel en el cual las componentes RGB corresponden a las que buscamos.

Con base en la matriz de color RGB que nos proporciona MATLAB y el vector con los componentes RGB que se han proporcionado creamos una matriz binaria vacía (todos los elementos en 0) de la misma dimensión que la imagen que vamos a procesar.

El algoritmo que se empleó funciona de la siguiente manera; mediante una variable de varianza (***var)*** que se podría denominar también tolerancia se da cierta libertad al cambio de color, en una imagen de buena calidad como la empleada en el ejemplo se puede tener una tolerancia baja, pero cuando se usa una cámara web de baja resolución o en una situación de baja luminosidad, un mismo color varia en tonos.

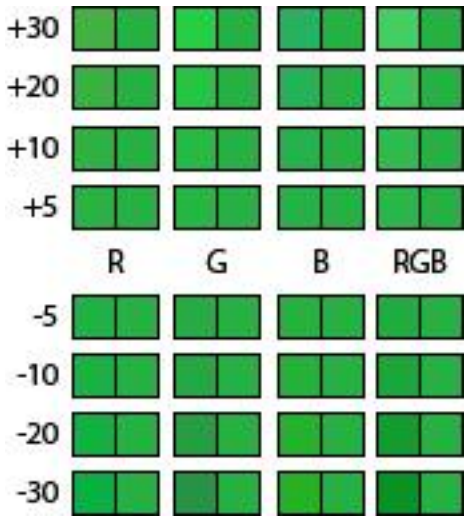

Figura 3. Varianzas de color con ***var*** = 30

Como se observa en la figura 3, un mismo color [39, 177, 68] siempre a la derecha, y sus varianzas en cada una de sus componentes y en todas combinadas a la izquierda, es una tolerancia de color amplia pero aun así poco perceptible al ojo, pero para el algoritmo que se emplea es crucial.

Una vez establecida una tolerancia comienzan los filtros, operando en cada una de las capas de color. Se toma el valor de la componente a trabajar y se le resta la tolerancia que se determina como mínimo, si esta resta es menor que 0 se establece como 0. Mediante la función "**find**" se obtiene para la capa de color actual todas las posiciones de los valores que sean superiores a este valor mínimo obtenido, luego intersecto estas posiciones con las posiciones de los valores que sean menores que el valor máximo, determinado por el valor de la componente a trabajar más la tolerancia. Obtenidas estas posiciones se intersectan repitiendo el proceso con cada capa de color RGB.

Ejemplo. Se tiene una imagen de 4x4 pixeles con las siguientes componentes de color.

R	
39	41
0	130

G	
177	167
10	65

B	
68	71
70	66

Se crea la matriz binaria vacía y una tolerancia de 10

Binaria	
0	0
0	0

Obtenemos la componente Roja del color seleccionado que

es 39 en este caso, determinamos entonces el valor mínimo como 39 – 10 = 29

MATLAB es un lenguaje vectorial por definición podemos obtener el valor en una matriz haciendo un llamado a su posición en X y Y o mediante un entero que toma la matriz como si fuera un gran vector. Por lo mismo cuando se ejecuta

posiciones = find(R >= 29)

Nos retornará un arreglo con los valores [1, 3, 4] que son las posiciones de los valores que contienen un valor mayor que 29 ahora lo intersectamos con las posiciones que cumplan la condición menores al máximo, en este caso 39 + 10 = 49

posiciones = intersect(posiciones, find(R <= 49))

El find interno retornará [1, 3] y la intersección entre esta respuesta y [1, 3, 4] es [1, 3]

Ahora con la capa verde, hallamos el mínimo en esta componente que está determinado por 177 - 10 = 167

posiciones = intersect(posiciones, find(G >= 167))

Se intersecta con los que sean menores al valor máximo aceptado 177 + 10 = 187

posiciones = intersect(posiciones, find(G <= 187))

El mismo proceso continua con la capa Azul, luego de terminar con las intersecciones solo se tendrán las posiciones [1, 3] entonces para cada posición ya filtrada entonces se establece el valor 1 en la matriz binaria creada anteriormente que al final tendrá los siguientes valores.

Binaria	
1	1
0	0

Y el proceso de filtrado es finalizado proporcionándonos una matriz con las posiciones de los pixeles que cumplen las condiciones dadas. En este caso (1, 1) y (1, 2)

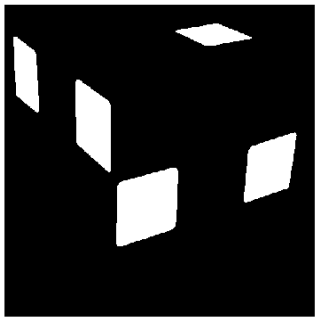

Figura 4. Resultado binario del proceso con el cubo

4. Determinar los centros y las cajas del color obtenido

Con MATLAB realizamos este proceso mediante la función: **regionprops(bw_area, 'BoundingBox', 'Centroid')**

La cual nos retorna una estructura con cada una de las cajas que se forman mediante la unión de los 1 que se encuentren agrupados, para nuestro ejemplo de la matriz 4x4 el Centroid tendría las coordenadas (1.5, 1) y la BoundingBox sería [1 1 1 0] determinando que la caja empieza en la posición (1,1) y se extiende 1 en x y 0 en y.

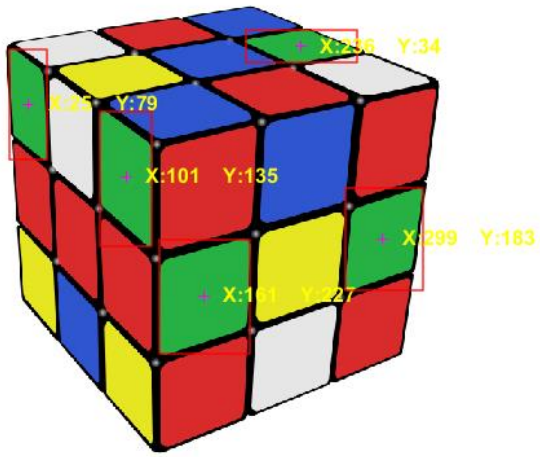

Figura 5. Cubo de Rubik con centros y cajas de unión

En la figura 5 se muestra el cubo de Rubik, al dibujar estas cajas (BoundingBox) y el Centroid. Hasta este punto ya se rastrea el color en una imagen. Si se itera el proceso sobre las capturas (snapshots) de una webcam esta mostrará los centros y cajas de unión del color seccionado.

5. Dibujar círculos de circunferencias variables y moverlos por la pantalla

Para este proceso se utiliza una función de dibujo que se

detalla en los comentarios del código, no cabe en el alcance de este articulo dado que es un proceso trivial y para un resultado final aún se puede optimizar mucho, es importante de cada circulo que se vaya a replicar con este experimento que se conozca la posición en la cual se encuentra el circulo y su radio.

6. Detectar las colisiones entre las regiones de color y los círculos.

Para la detección de colisiones se usa un algoritmo que hace una comparación de cada circulo con cada caja de contenido, comprobando 3 condiciones y al fallar cualquiera de las 3 se dicta que no hay colisión.

Estas son:

- Que se intersecten en el rango de X

Para que esta condición se cumpla, debe haber al menos un pixel que se intersecte en el rango de X del circulo dado por

RXCirculo = (centro – radio : centro + radio)

En notación de MATLAB si el circulo tiene un centro de la coordenada X de 20 y su radio de 10 entonces

RXCirculo = [10, 11, 12, 13, …, 27, 28, 29, 30]

Para la BoundingBox este valor ya no es dado con el valor en X y su extensión.

- Que se intersecten en el rango de Y

Para que esta condición se cumpla, debe haber al menos un pixel que se intersecte en el rango de Y del circulo dado por

RYCirculo = (centro – radio : centro + radio)

En notación de MATLAB si el circulo tiene un centro de la coordenada Y de 30 y su radio de 10 entonces

RYCirculo = [20, 21, 22, 23, …, 37, 38, 39, 40]

Para la BoundingBox este valor ya no es dado con el valor en Y y su extensión.

- Que realmente se intersecten en Y

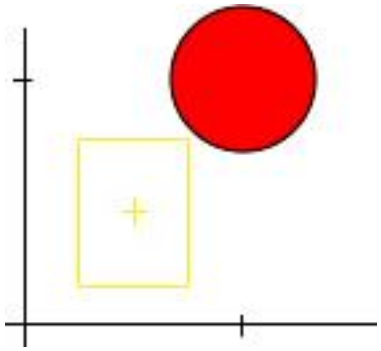

Figura 6. Falsa colisión

Si se llega a este punto aun no es suficiente para determinar una colisión puesto que se puede tratar de una falsa colisión como se muestra en la figura 6, para determinar si realmente se generó una colisión se usa el valor más cercano al centro del circulo de la intersección en el eje X y con ese valor aplicamos el teorema de Pitágoras.

$$y = \sqrt{radio^2 + x^2}$$

Donde con este valor de y construiremos un nuevo rango de Y como en la condición 2 para detectar si se trata realmente de una colisión de haber puntos de y intersectados entonces borramos el circulo y se apunta la colisión

V. RESULTADOS

Para la revisión del resultado obtenido hemos realizado una métrica del tiempo que toma procesar en segundos, una imagen de diferentes tamaños dados en pixeles, siendo la mayor de 2000x2000px y la menor de 100x100, teniendo en cuenta que la resolución que brindan las cámaras web comerciales en el momento con muy pocas excepciones superan los 1920x1080px considerado FullHD.

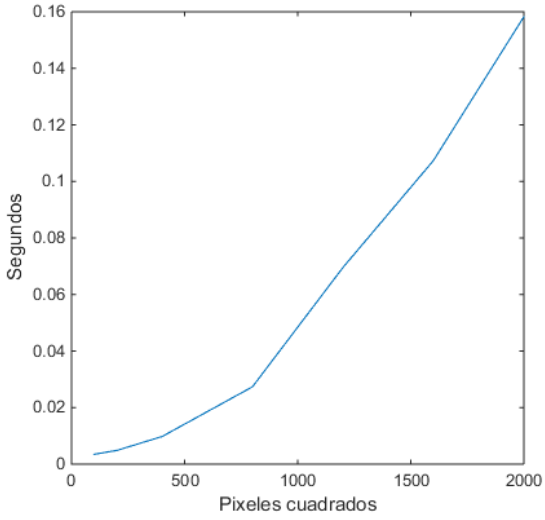

Figura 7. Tiempo que toma procesar con el algoritmo

imágenes de diferentes tamaños.

Como se observa en la figura 7, el rendimiento para una imagen relativamente grande de 2000x2000px es aceptable.

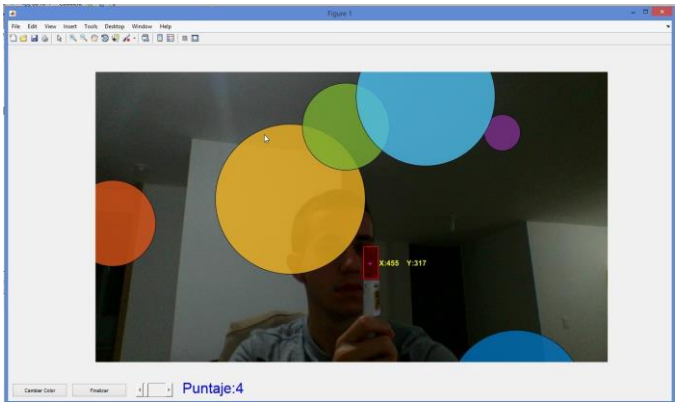

Figura 8. Interfaz final del juego

VI. CONCLUSIONES Y LINEA FUTURA

El algoritmo presentado permite la selección y posterior seguimiento de un color determinado y no se limita únicamente a uno de los 3 colores primarios esto nos brinda una mayor jugabilidad pero tratándose de un concepto aún, es posible hacerlo más eficiente.

Las colisiones han probado que se detectan en el momento en cual realmente se presenta una intersección entre las cajas de color identificadas y el círculo en pantalla, este fue uno de los retos ya que inicialmente cuando no se apreciaba una colisión el sistema la determinaba y eliminaba el círculo.

Como trabajo futuro se podría mejorar el algoritmo para la detección de los colores de la piel tomando diferentes muestras de la piel del jugador para que no se tengan que emplear objetos de un color destacable sino que sea posible jugar con las manos, sin necesidad de pintar los dedos de algún color.

REFERENCIAS

[1] Kinect. http://www.xbox.com/es-es/xbox360/accessories/kinect/home, 2011
[2] Franco. Oscar, Perez-Sala. Xavier, et al (2011). Identificación y seguimiento de personas usando kinect por parte de un robot seguidor. Universitat Politècnica de Catalunya
[3] Robert Y. Wang and Jovan Popovic, Real-Time Hand-Tracking with a Color Glove, ACM Transaction on Graphics (SIGGRAPH 2009), 28(3), August 2009.
[4] Ceballos. Alexander, Gómez Mendoza. Juan Bernardo, Prieto. Flavio. (2009). Seguimiento del contorno externo de la boca en imágenes de vídeo. Revista Ingenierías Vol. 8, 129-155

Autor
Diego Calle
Estudiante de Ingeniería de Sistemas de la Universidad de Antioquia
2015

El código fuente del juego se puede descargar directamente de GitHub en el siguiente enlace.
https://github.com/dnetix/tracking