Práctica 3. Segmentación y sistemas de iluminación.
(revisado septiembre 2019)

Leslie Trejo Zamudio, Ana María Hernández Moreno

Resumen— Se hace uso de los conocimientos adquiridos en clase, uso de las librerías de openCV y el lenguaje de programación Python para proceder a realizar técnicas de segmentación de imágenes (a color y a escalas de grises), ejecutar métodos de umbralización y observar cómo se ejecutan algunas de estas técnicas en tiempo real.

Abstract— The acquired class knowledge is used, openCV libraries and Python programming language are used to make image segmentation techniques (color and grayscale), execute threshold methods and observe these techniques being executed in real time.

Palabras claves— Iluminación, umbralización, segmentación, colores, contornos, técnicas.

Index Terms— Illumination, threshold, segmentation, colors, contours, techniques.

1 Introducción.

El objetivo principal de esta práctica, es realizar ejemplos con aspectos fundamentales que afectan a las imágenes como lo es la iluminación, además, se muestra como este parámetro puede afectarlas, así mismo, se trabajará con la obtención de contornos, por último, se indagara en algunos de los espacios de color como lo son HSV y YCrCb.

2. Procedimiento

2.1 Sistemas de iluminación en visión artificial

El tener la iluminación adecuada desde la parte de visión artificial al momento de realizar las capturas fotográficas a objetos es muy importante ya que permite realizar un buen análisis posteriormente, por esta razón, es necesario realizar un estudio personalizado de la iluminación que requiere el objeto, con el fin de obtener la más óptima para cada aplicación.

2.1.1. Luz frontal:

Definición: La cámara se posiciona mirando en la misma dirección que la luz.

Ventajas: Elimina sombras, se puede utilizar a grandes distancias cámara/objeto.

Desventajas: Intensos reflejos sobre superficies reflectantes

Aplicación: Para superficies con pocos reflejos: papel, tela, detección de marcas de diferentes colores, cambio de color en cualquier superficie.

2.1.2. Luz lateral

Definición: La cámara se posiciona mirando el objeto mientras que la dirección de luz es lateral al objeto. La inclinación viene determinada por el grado de resalte de relieves.

Ventajas: Resalta los relieves pequeños que sean de los objetos.

Desventajas: Con ángulos pequeños respecto a la horizontal, la luz producirá sombras en todos los relieves.

Aplicación: Para resaltar bordes, rayas y fisuras en dirección determinada.

2.1.3. Iluminación por campo oscuro

Definición: La luz es emitida lateralmente con un ángulo muy pequeño mediante un anillo en todas las direcciones, rebotando en los defectos del objeto a analizar e incidiendo en la cámara.

Ventajas: Destaca los detalles en superficies con muy poco contraste.

- *Leslie Trejo Zamudio estudia ingeniería mecatrónica en el Instituto Politecnico Nacional E-mail: leslietrejo293802@correo.itm.edu.co.*
- *Ana maría Hernández Moreno estudia ingeniería mecatrónica en el Instituto tecnológico Metropolitano ITM (e-mail: anahernandez205388@correo.itm.edu.co)*

Desventajas: No es recomendable en superficies que absorban la luz.

Aplicación: Utilizada para resaltar incrustaciones y códigos alfanuméricos con poco contraste, verificación de grabados tipo laser.

2.1.4. Iluminación por contraste
Definición: La luz es emitida desde la parte posterior del objeto quedando este entre la fuente de iluminación y la cámara.

Ventajas: Permite inspecciones de siluetas con mediciones muy precisas y de impurezas en los objetos transparentes o translúcidos.

Desventajas: No permite reconocer los detalles superficiales del objeto, código, inscripciones.

Aplicación: Utilizada para la inspección de la silueta del objeto, materiales translúcidos o trasparentes.

2.1.5. Iluminación axial difusa

Definición: La luz emitida siendo reflejada 90° por un espejo semitransparente que desvía los haces de luz en la misma dirección que el eje de la cámara.

Ventajas: Permite inspecciones de códigos en materiales altamente reflectantes.

Desventajas: No permite reconocer relieves en el objeto.

Aplicación: Utilizada para la inspección de superficies planas reflectantes como PCB, etiquetas reflectantes, inspecciones de impresión sobre aluminio.

2.1.6. Iluminación difusa tipo Domo

Definición: La luz es emitida dentro de una cúpula esférica resultando una luz difusa desde todas direcciones, eliminando sombras y reflejos.

Ventajas: Eliminación de sombras y minimización de arrugas, polvo y relieves.

Desventajas: Coste elevado.

Aplicación: Utilizada para la inspección de instrumental médico, espejos, compact disk, latas.

2.1.7. Iluminación por láser

Definición: La iluminación mediante láser o luz estructurada se utiliza normalmente para resaltar o determinar una tercera dimensión de un objeto.

Ventajas: No influye la iluminación externa

Desventajas: Coste elevado.

Aplicación: Ajuste de procesos de corte, control de profundidad de objetos.

2.2 Umbralización código de barras

En este parágrafo, se desea obtener 3 fotos diferentes de un código de barras (de un producto real) usando diferentes sistemas de iluminación, para luego aplicar una segmentación a partir de la umbralización de las imágenes que tenemos. Procedemos con llevar estas imágenes que se encuentran a color a una escala de grises, luego las segmentaremos y analizaremos la que nos brinde mejor información.

La imagen 1, imagen 2 e imagen 3 son del mismo producto (código de barras), se evidencia que tiene muy poca iluminación y al aplicarle el método de segmentación gran parte de la información se pierde.

Imagen 1. Imagen con poca iluminación.

Imagen 2. Imagen a escala de grises

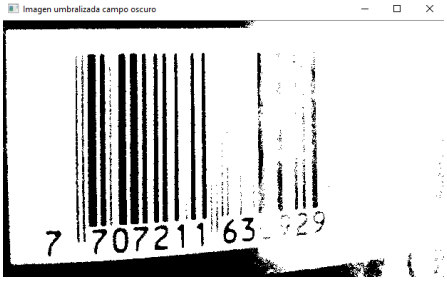

Imagen 3. Imagen segmentada

En la imagen 4, imagen 5 e imagen 6 se puede evidenciar que la imagen presenta una iluminación diferente, la iluminación que se aplico es una iluminación frontal. Luego la imagen 4 pasa a ser trabajada en escala de grises, al realizar la segmentación (imagen 6) se evidencia gran detalle y no hay un exceso de brillo que nos de mucha perdida de información.

Imagen 4. Imagen de código de barras. Iluminación frontal

Imagen 5. Imagen en escala de gris

Imagen 6. Imagen segmentada

En la imagen 7, imagen 8 e imagen 9 se aplica de nuevo una nueva forma de iluminación, para este caso se utiliza la iluminación lateral, al igual que en los casos anteriores se realiza un cambio a la imagen a escala de gris y luego se aplica la segmentación, en la imagen 9 se puede observar que al igual que en la imagen 3 hay perdida de información, evidenciando los problemas que se originan por la forma de iluminación al tomar la fotografía.

Imagen 7. Imagen código de barras. Iluminación lateral

Imagen 8. Imagen en escala de grises

Imagen 9. Imagen segmentada

Con la información evidenciada en los procesos de adquisición de las imágenes (fotografías de ellas con diferentes formas de iluminación), pasarlas a una imagen en escala de gris y luego segmentarlas, se decide que es mejor trabajar con la imagen 6, ya que presenta buena definición y a simple vista no se logra observar pérdidas significativas de información que nos origine inconveniente cuando se realice el proceso de determinar el ancho de cada barra y cuantas barras hay en ella. Al aplicar un código especifico en esta imagen 6, se encuentra que hay 31 barras.

2.3 Detección de contornos

Un contorno es un conjunto de puntos que conectados unos con otros de manera consecutiva forman una figura que rodea un objeto determinado. Los contornos son una herramienta útil para el análisis de formas, detección y reconocimiento de objetos.

La detección de contornos en OpenCV se aplica sobre imágenes binarizadas con el objetivo de tener una mayor precisión, por lo tanto, se recomienda antes de encontrar los contornos aplicar un algoritmo de detección de bordes canny o threshold; la figura obtenida puede ser analizada posteriormente para determinar cuál es el objeto que se ha detectado. En OpenCV, encontrar contornos es como encontrar un objeto blanco a partir de un fondo negro.

Para obtener los contornos se debe utilizar la función findContours (), mientras que drawContours () ayuda a dibujarlos.

2.3.1. Métodos de aproximación al contorno

- **CV_CHAIN_APPROX_NONE.** Guarda todos los puntos límites de los contornos.

- **CV_CHAIN_APPROX_SIMPLE**. Elimina todos los puntos redundantes y comprime el contorno, ahorrando así memoria.

- **CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS.** Es un algoritmo especial, aplica uno de los matices del algoritmo de aproximación de la cadena Teh-Chin.

- **Offset**. Es un parámetro opcional, esta opción permite desplazar cada punto del contorno encontrado, esto es útil si los contornos se extraen del ROI de la imagen y luego ellos podrán ser analizados en todo el contexto de la imagen.

2.3.2. Modo de recuperación de contorno.

- **CV_RETR_EXTERNAL.** Recupera sólo los contornos exteriores extremos. Establece la jerarquía: hierarchy [i][2] = hierarchy [i][3] = -1 para todos los contornos.

- **CV_RETR_LIST**: Recupera todos los contornos sin establecer ninguna relación jerárquica.

- **CV_RETR_CCOMP:** Recupera todos los contornos y los organiza en una jerarquía de dos niveles. En el nivel superior, hay límites externos de los componentes. En el segundo nivel, hay límites de los orificios. Si hay otro contorno dentro de un orificio de un componente conectado, se coloca en el nivel superior.

- **CV_RETR_TREE.** Recupera todos los contornos y reconstruye una jerarquía completa de contornos anidados. Esta jerarquía completa se construye y se muestra en la demo de OpenCV contours.c.

2.3.3. Características de los contornos:

Momento: Los momentos de imagen le ayudan a calcular algunas características como el centro de masa del objeto, el área del objeto, etc. La función **cv2.moments ()** da un diccionario de todos los valores de momento calculados.

Contorno de área: El área de contorno viene dada por la función **cv2.contourArea()** o por momentos, M['m00'].

Perímetro del contorno: También se le llama longitud del arco. Puede ser encontrado usando la función **cv2.arcLength()**. El segundo argumento especifica si la forma es un contorno cerrado (si pasa True), o sólo una curva.

Aproximación de contorno: Aproxima una forma de contorno a otra forma con menos vértices dependiendo de la precisión que especifiquemos. Es una implementación del algoritmo Douglas-Peucker.

Para comprender esto, suponga que se está tratando de encontrar un cuadrado en una imagen, pero debido a algunos problemas en la imagen, no se logra obtener un cuadrado, si no, una "mala forma" (Ver Imagen 10. Imagen de la izquierda). Se puede utilizar esta función para aproximar la forma. En este caso, el segundo argumento se llama épsilon, que es la distancia máxima entre el contorno y el contorno aproximado. Es un parámetro de precisión. Una sabia selección de épsilon es necesaria para obtener el rendimiento correcto. En la segunda imagen (Imagen 10 imagen del centro), la línea verde muestra la curva aproximada para épsilon = 10% de la longitud del arco. La tercera imagen (Imagen 10 imagen de la derecha) muestra lo mismo para épsilon = 1% de la longitud del arco. El tercer argumento especifica si la curva está cerrada o no.

```
epsilon = 0.1*cv2.arcLength(cnt, True)
approx = cv2.approxPolyDP(cnt, epsilon, True)
```

Imagen 10. Aproximación de contorno.

Convex Hull: Se verá similar a la aproximación del contorno, pero no lo es (ambos pueden proporcionar los mismos resultados en algunos casos). Aquí, la función **cv2.convexHull()** comprueba una curva para detectar defectos de convexidad y la corrige.

En términos generales, las curvas convexas son las curvas que siempre están abultadas, o al menos planas. Y si está abultada por dentro, se llama defectos de convexidad. En la imagen 11 la línea roja muestra el casco convexo de la mano. Las marcas de flecha de doble cara muestran los defectos de convexidad, que son las desviaciones máximas locales del casco con respecto a los contornos.

Imagen 11. Imagen convexa de la mano

Comprobación de la convexidad: La función para comprobar si una curva es convexa o no, **cv2.isContourConvex()**. Simplemente devuelve si es verdadero o falso.

Delimitación de rectángulo: Se encuentran dos tipos, ellos son:

- **Rectángulo de limite recto:** Es un rectángulo recto, no considera la rotación del objeto. Así

que el área del rectángulo delimitador no será mínima. Se encuentra por la función **cv2.boundingRect().**

- **Rotación de rectángulo:** El rectángulo delimitador se dibuja con un área mínima, por lo que también considera la rotación. La función utilizada es **cv2.minAreaRect()**. Devuelve una estructura Box2D que contiene los siguientes detalles ((esquina superior izquierda (x, y), (ancho, alto), ángulo de rotación)). Para dibujar este rectángulo, necesitamos 4 esquinas del rectángulo. Se obtiene mediante la función **cv2.boxPoints()**.

En la figura 12, ambos rectángulos se muestran en una sola imagen. El rectángulo verde muestra el rectángulo delimitador normal. El rectángulo rojo es el rectángulo girado.

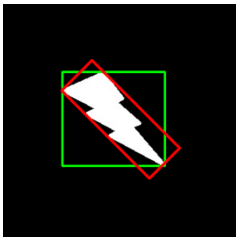

Imagen 12. Delimitación de rectángulo

Círculo de cierre mínimo: Se encuentra la circunferencia de un objeto utilizando la función cv2.minEnclosingCircle(). Es un círculo que cubre completamente el objeto con un área mínima. Ver imagen 13

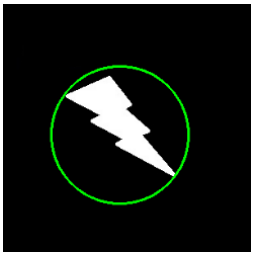

Imagen 13. Circulo de cierre

Colocación de una elipse: Se ajusta una elipse a un objeto. Devuelve el rectángulo rotatorio en el que está inscrita la elipse. Ver imagen 14

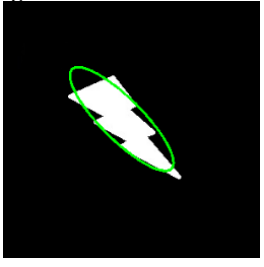

Imagen 14. Elipse

Ajuste de una línea: Se puede ajustar una línea a un conjunto de puntos. La imagen de abajo contiene un conjunto de puntos blancos. Podemos aproximar una línea recta a ella. (Imagen 15)

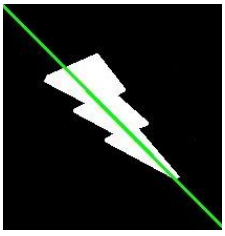

Imagen 15 Ajuste de una línea.

2.4.4. Desarrollo de ejemplo

En la imagen 7 se desea identificar los contornos que rodean a cada una de las herramientas presentes, para esto, primero se debe obtener una imagen binarizada.

Para obtener la imagen binarizada, se procede a utilizar un procedimiento de umbralización, pero antes, se aplicará un filtro gaussiano para reducir los ruidos que tiene la imagen de entrada. En la imagen 8, se logra identificar el ruido presente que contiene la imagen original previo a la umbralizacion. En la imagen 9, se observa el resultado al aplicar el filtro gaussiano previo a la umbralizacion

Imagen 16. Imagen original.

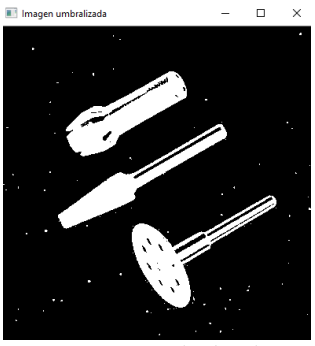

Imagen 17. Imagen umbralizada con ruido

Imagen 18. Imagen umbralizada después de aplicar filtro

El siguiente paso es obtener los contornos y luego dibujarlos sobre la imagen original, para detectar los contornos utilizamos la función findContours (), donde se debe indicar la imagen binaria en donde se ubicarán los contornos, se procede a utilizar RETR_TREE para establecer la jerarquía de contornos y al final indicamos el método usado para simplificar los contornos, utilizando el método CHAIN_APPROX_SIMPLE. Se dibuja los contornos utilizando la función drawContours (), dibujando todos los contornos encontrados como se puede apreciar en la imagen 19.

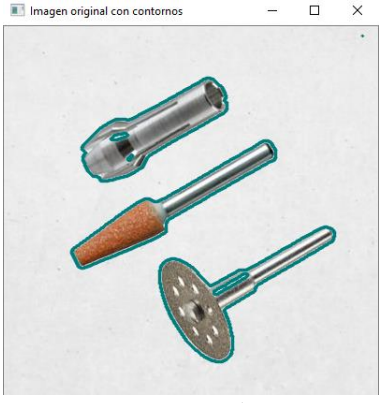

Imagen 19. Imagen original con contornos.

En la imagen 10 se puede observar que se han dibujado varios contornos, dependiendo de lo que se desee hacer, se utilizara los diferentes características para definir los contornos, es decir, suponga que los contornos internos que se encuentran en la herramienta uno y tres, no son de interés en la aplicación que se está desarrollando, una manera sencilla de filtrar estos contornos no deseados es calcular el área de los mismos y establecer un rango mínimo y máximo aceptable.

2.4 Segmentación de color a partir de un video en tiempo real

En el desarrollo de este punto se segmentarán 5 colores en tiempo real, primero se transformará la imagen de entrada al espacio de color y posteriormente se segmentará cada color. Los espacios de color que utilizaremos serán HSV y YCrCb.
Primero se segmentará en el espacio HSV.

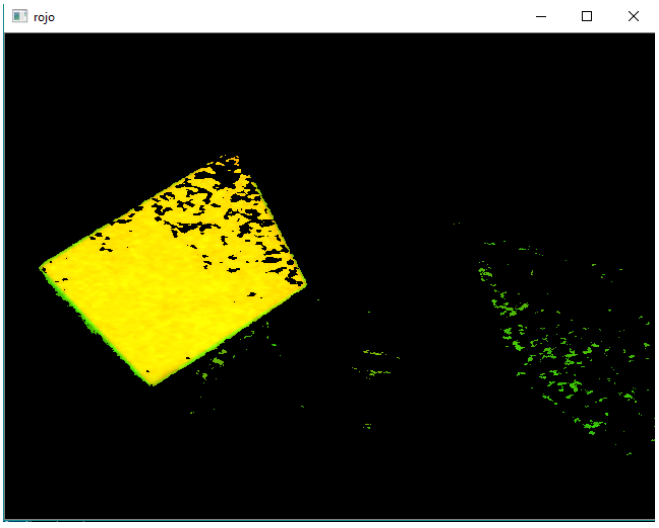

Imagen 20. Imagen color rojo segmentado

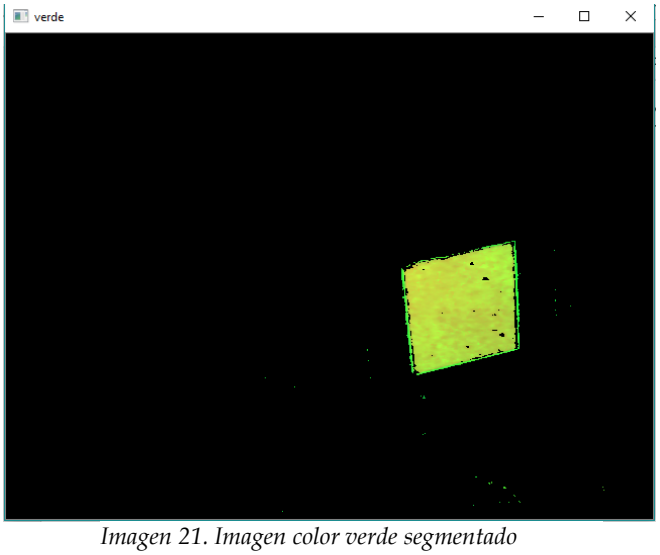

Imagen 21. Imagen color verde segmentado

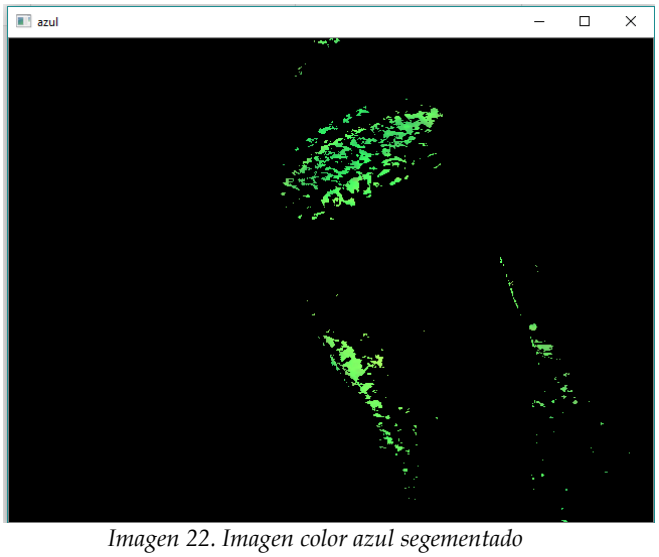

Imagen 22. Imagen color azul segementado

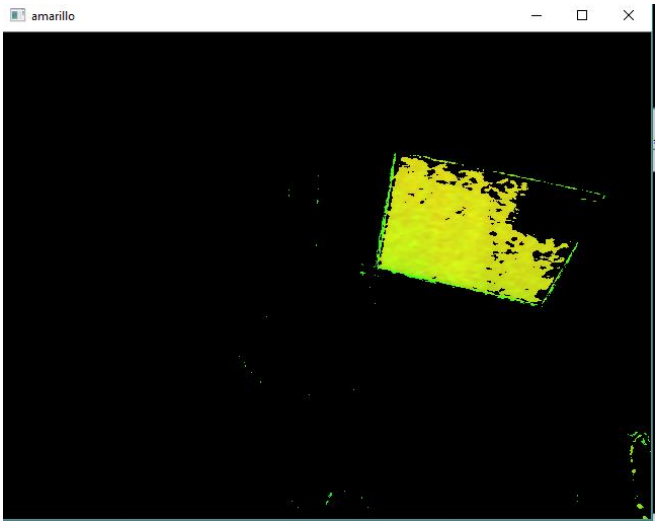

Imagen 23. Imagen color amarillo segmentado

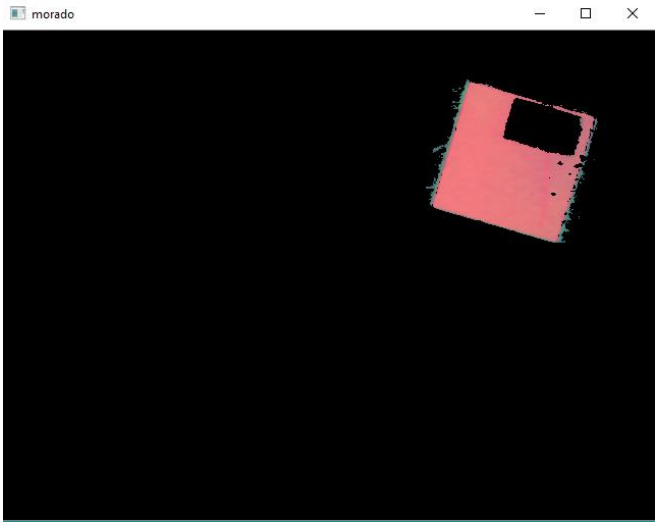

Imagen 24. Imagen color morado segmentado

A continuación, se presentarán los colores segmentados en un sistema de color YCrCb

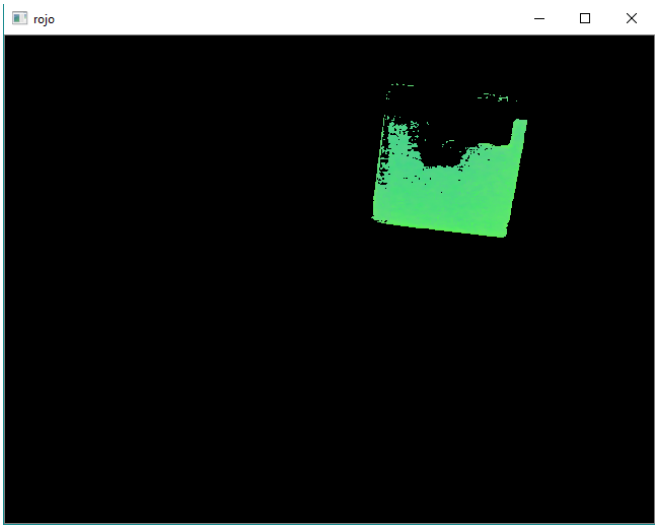

Imagen 25. Imagen color rojo segmentado

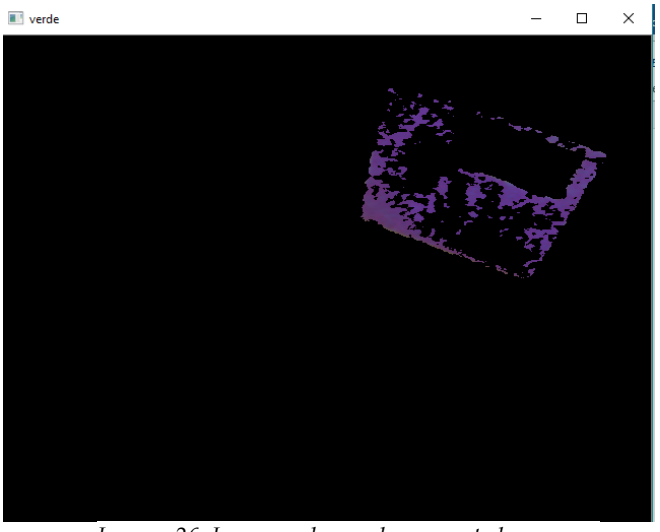

Imagen 26. Imagen color verde segmentado

Imagen 27. Imagen color morado segmentado

Imagen 28. Imagen color azul segmentado

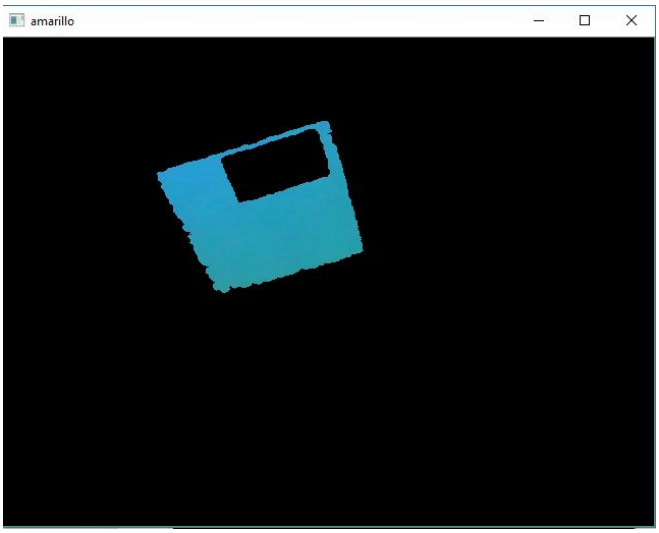

Imagen 29. Imagen color amarillo segmentado

Durante el desarrollo de este punto se puede observar tres espacios de color HSV y YCrCb, de forma directa y BGR de forma indirecta, ya que se usó como base para obtener los parámetros de segmentación de los otros espacios de color. Uno de los problemas encontrados durante el desarrollo fue que al trabajar entre colores secundarios como lo son naranja, amarillo y morado fue conflictivo encontrar los rangos para solo detectar estos colores y no los primarios. Así mismo, si se varia la iluminación en la habitación habrá un error en la detección.

2.5 Método Otsu

El método Otsu es uno de los más utilizados en la determinación automática del umbral de segmentación; además, proporciona el umbral óptimo para la segmentación de la imagen, bajo el criterio de máxima varianza entre fondo y objeto mediante la búsqueda exhaustiva, es decir, este método usa una búsqueda exhaustiva para evaluar el criterio y maximizar la varianza entre clases (el umbral óptimo se logra cuando la varianza entre clases genera un valor mínimo).

Ventajas:

- El método presenta una buena respuesta frente a la mayoría de situaciones del mundo real (imágenes ruidosas, con histogramas planos, mal iluminadas, entre otras.)
- Es automático, no precisa de supervisión humana, preprocesamiento de la imagen y otro tipo de información acerca de la misma.

Desventajas:

- A medida que el número de clases en la imagen aumenta, el método necesita mucho más tiempo para seleccionar un umbral multinivel adecuado.

Se le aplicara a la imagen 30 (papa con iluminación backligth) una umbralizacion aplicando el método Otsu.

Imagen 30. Imagen de la papa en escala de grises

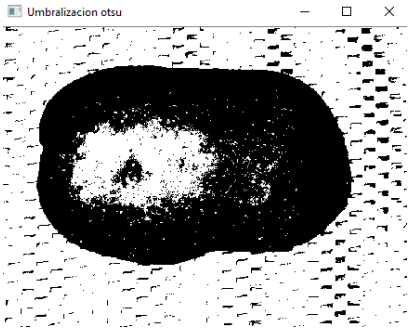

Imagen 31. Imagen binarizada con el método Otsu

En la imagen 31 se puede observar algunos cambios en la textura y algunas imperfecciones en la superficie iluminada, además, se puede observar los contornos de la forma analizada. Este método es muy útil en espacios con iluminación posterior.

2 Conclusiones

Durante la práctica se pudo observar los cambios que se generan en una imagen cuando se aplican diferentes tipos de iluminación y cuál es el indicado para cada caso, se realiza ejemplos en la obtención de contornos y el uso del método otsu para mejorar la obtención de ellos, también se realiza la segmentación de colores en dos espacios de color diferente, de esta forma se observa los rangos y variables de cada espacio.

References

[1] J OpenCV. (s.f.). Open Source Computer Vision. Obtenido de Open Source Computer Vision: https://docs.opencv.org/3.4/d8/dc8/tuto

[2] acodigo. (s.f.). *acodigo.blogspot*. Obtenido de acodigo.blogspot: http://acodigo.blogspot.com/2017/07/umbralizacion-en-opencv.html

[3] docs.opencv.org. (s.f.). *docs.opencv.org*. Obtenido de docs.opencv.org: https://docs.opencv.org/3.4.0/d3/dc0/group__imgproc__shape.html#ga2c759ed9f497d4a618048a2f56dc97f1

[4] programcreek. (s.f.). *programcreek*. Obtenido de programcreek: https://www.programcreek.com/python/example/86843/cv2.contourArea

[5] python's eyes. (s.f.). *python's eyes*. Obtenido de python's eyes: https://pythoneyes.wordpress.com/2017/06/30/filtro-de-la-mediana-con-python-3-eliminacion-de-ruido-sal-y-pimienta-en-imagenes/