Rastreo de Movimiento de Objetos Geométricos en Video
Informe Trabajo Interciclo

Facultad De Ingeniería, Universidad De Cuenca
GRÁFICOS POR COMPUTADORA
Freddy L. Abad L., Edisson S. Reinozo T.
{freddy.abadl, edisson.reinozo}@ucuenca.edu.ec

***Resumen*- El mundo actual requiere cada vez más tener ambientes controlados. Estos ambientes controlados permiten a un gobierno garantizar la seguridad, por ejemplo, de sectores estratégicos (hospitales, escuelas, supermercados). El campo de la visión por computadora provee una solución sencilla mediante el Object tracking o rastreo de movimiento. Esta técnica de procesamiento de video se implementa en distintas fases: segmentación de la imagen, identificación de contornos, aproximación de objetos, rastreo a través de los cuadros de video y el dibujo de las secciones en movimiento. Cada fase, contiene diversos problemas que este informe pretende explorar y proponer varias metodologías que aborden estos problemas. Sin duda el campo de la visión por computadora, presenta muchas ventajas antes diversas situaciones en las cuales explorar y explotar. El uso de este recurso, abre una multitud de posibilidades, entre ellas, por ejemplo, el reconocimiento de personas en movimiento en un video, la identificación de armas en sectores sensibles, el uso de implementos de seguridad por el personal, entre otros. Todos estos problemas y conceptos, se puede abordar utilizado varias opciones provistas por la API OpenCV para Python en 2D.**

***Palabras Clave*- OpenCV, Contorno, Figura, Segmentación.**

I. INTRODUCCIÓN

El reconocimiento de elementos en un video es parte fundamental del campo de la visión por computadora y sus derivados usos en la Inteligencia Artificial. El reconocimiento de elementos de un video, realiza mediante, varios algoritmos, el filtrado, trazo e identificación del elemento en búsqueda [1]. Este uso se propone en esta práctica reconocer figuras geométricas regulares. El proceso inicial de filtrado permite disminuir la cantidad de datos a procesar, segregando las imágenes o segmento de imágenes [1] que no aportan a la finalidad de los objetivos de esta práctica. Este proceso pertenece a un concepto mayor, el cual es la Segmentación de Imágenes. Elsevier (2016), define a la segmentación de imágenes como "el proceso de dividir una imagen digital en varias partes o grupos de píxeles u objetos". Esto con el fin de simplificar la representación de una imagen o el procesamiento como "imágenes independientes". Este concepto se usa para localizar objetos y para encontrar los límites de estos dentro de una imagen. [1].

II. OBJETIVOS

A. Objetivo General

1. Rastrear objetos geométricos (Object Track) en un video.

B. Objetivo Específico

1. Utilizar metodologías la segmentación de imágenes y reconocimiento de contornos apropiados para un entorno supervisado por capturas de video ya sea con una calidad alta o baja.
2. Identificar figuras geométricas estáticas y en movimiento mediante funciones provistas por a API de OpenCV 2D para Python.
3. Grabar los resultados de reconocimiento de figuras en un video nuevo, basado en el video original, agregando el contorneado y etiquetas.
4. Optimizar el código fuente, utilizando prácticas de programación que faciliten el mantenimiento en el tiempo de este.
5. Medir la eficacia del algoritmo de rastreo ante uno o varios videos de prueba.

III. MARCO TEORICO

Los conceptos teóricos, necesarios para entender la metodología y finalidad de este proyecto, se explica a través de fases. Estas fases obedecen a las etapas abordadas por el objetivo principal, las cuales son explicadas en la sección IV.

A continuación, se definen los conceptos de cada algoritmo y metodología abordado.

A. Tratamiento y Segmentación de Imágenes.

Esta fase del Object tracking, trata a las imágenes o frames de un video por separado. Se aplica técnicas de procesamiento de

imágenes, como la transformación y filtración de imágenes [2].

Filtro de Imágenes: Inicialmente se usa técnicas de filtración de imágenes, para este fin, existen varios métodos de desenfoque de imágenes: gaussiano, median, Laplaciano, etc. La finalidad de esta técnica de transformación de imágenes, es el eliminar los detalles de una imagen, que, al procesar posteriormente, puede provocar errores por ruido. En el caso de esta práctica se optó por un filtro gaussiano.

a. Filtro Gaussiano

El filtro gaussiano es un filtro de paso bajo que elimina los componentes de alta frecuencia se reducen [3]. Este método matemático, actúa en cada píxel de la capa o Frame, estableciendo su valor en el promedio de todos los valores de píxeles presentes en un radio definido en el diálogo [4]. Así, a valores más alto producirá mayor desenfoque. Así, el efecto visual de esta es un desenfoque suave que asemeja al de ver la imagen a través de una pantalla translúcida [5]. Ver Imagen 1.

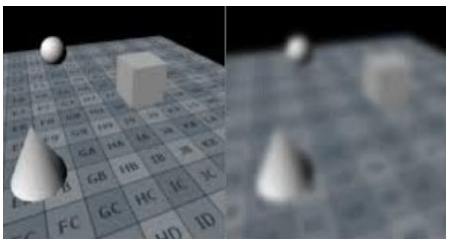

Image 1: Imagen Original vs Imagen con Filtro Gaussiano [6]

Segmentación de Imágenes:

I. Detección de Bordes

Para obtener el objetivo principal de esta práctica, el cual es reconocer figuras geométricas, se debe segmentar las imágenes mediante detección de bordes. La detección de bordes puede realizarse con una variedad de métodos matemáticos para identificar puntos en una imagen digital donde el brillo de la imagen cambia bruscamente formando discontinuidades. Así, se obtiene zonas importantes de la imagen [7].

A. Método de Canny

El algoritmo de Canny, forma parte de las técnicas de detección de bordes o Edge Detection. Este algoritmo se crea en base a la variación o gradiente de los píxeles, así a mayor gradiente, mayor probabilidad de que los píxeles analizados son parte de un borde. Este método busca tener una buena detección, marcando todos los bordes de la imagen, además de una buena localización, marcando los bordes reales de la imagen original y finalmente tener una respuesta mínima, minimizando los bordes falsos [8]. Ver imagen 2.

Imagen 2: Detección de bordes mediante el algoritmo de Canny, imagen real vs imagen con bordes [9]

B. Aproximación de Objetos

Determinados los bordes, continua la aproximación de objetos algoritmos según el tipo de figura a encontrar, ya sean figuras con esquinas (cuadrado, rectángulo, triángulo) o figuras sin esquinas (círculos). Esto lo realiza mediante encontrar contornos y posteriormente mediante algoritmos Hough o Aproximación de Polígonos. Los contornos son curvas que une todos los puntos continuos (a lo largo del límite), que tienen el mismo color o intensidad, así se garantiza que son figuras cerradas o convexas [10].

a. Estimación según número de Corner o esquinas

La aproximación según las esquinas permite determinar figuras geométricas con vértices, como los cuadrados o triángulos. Encontrado el contorno, se estima el tipo de polígono mediante el número de Corner o esquinas.

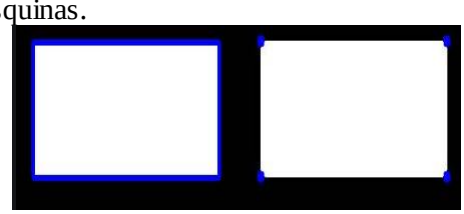

Imagen 3: Aproximación de Figuras mediante la estimación de contorno [12]

b. Estimación de figuras circulares

El caso de figuras circulares, es distinto al de polígonos con esquinas, ya que un círculo se conforma una curva cerrada sin Corner. En este caso, existen múltiples técnicas de identificación de círculos, entre los cuales los algoritmos de transformación de Hough (Hough Circles).

La Transformación de Hough Circle (CHT) es una técnica de extracción de características básicas en imágenes imperfectas, es decir, incluso en figuras donde el contorno de la curva está incompleto [11]. Esto mediante una

búsqueda exhaustiva de objetos con un alto grado de simetría radial. Como lo define Smith B, "el método trata cada pixel de la imagen, donde cada vez que un píxel transformado con una intensidad mayor que cero aterriza en una coordenada cartesiana, esa coordenada obtiene un voto. A medida que la imagen continúa transformándose en un círculo de un radio dado, si un círculo en la imagen tiene el mismo radio, los votos se acumularán en el centroide de este círculo. Por lo tanto, al encontrar los máximos en la transformación (puntos con el mayor número de votos) puede encontrar el centroide de círculos dentro de la imagen" [13] Ver imagen 4.

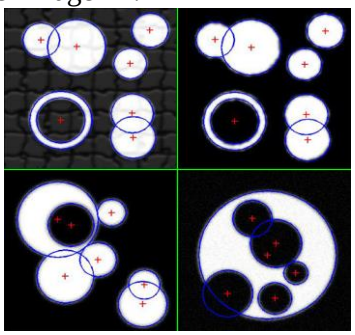

Imagen 4: Búsqueda de círculos en una imagen digital mediante la Transformación de Hough Circles (CHT) [14]

C. Rastreo de objetos a través del cuadro de video.

Todas las etapas definidas se ejecutan en todos los frames de un video. Así, se puede estimar la continuidad de movimiento de determinado objeto. Para realizar el tracking de un objeto, es importante encontrar el centro de cada figura, mediante la estimación según el área y dimensiones de la figura. El seguimiento del objeto a rastrear, almacena las coordenadas de la estimación de figura en el último Frame analizado. Estas coordenadas se almacenan en formato de cola, permitiendo realizar su seguimiento correctamente. Todos estos procesos, son algoritmos implementados manualmente, sin ninguna técnica como el caso de las transformaciones de imágenes.

Imagen 5: Rastreo de un objeto [15]

D. Dibujo de las áreas estáticas y en movimiento.

La rotulación o dibujo de las figuras geométricas encontradas y de su movimiento, se realiza en cada Frame. Para tal se utiliza funciones provistas por la API OpenCV, las cuales son explicadas en la sección IV.D (Metodología).

IV. METODOLOGÍA

El proyecto se implementó en distintas fases subyacentes, las cuales se definieron en la sección III (Marco Teórico), a continuación, se detalla su implementación.

A. Captura de Video

Inicialmente, se debe implementar funciones que alimenten los consecuentes algoritmos con video o secuencias de frames. OpenCV provee la función VideoCapture() para la captura de video, sus parámetros permiten reproducir un video existente, o conexión a una webcam o a una cámara externa [16]. Además, se implementó la función grab() que permite capturar el escritorio como fuente de datos para el video (Screen Record) [17].

B. Tratamiento y Segmentación de Imágenes

a. Filtro Gaussiano para el filtrado de imágenes digitales

La API de OpenCV provee la función GaussianBlur, la cual recibe de parámetro la imagen digital en 2 canales (blanco y negro o escala de grises) y varios parámetros, entre los cuales el núcleo de desenfoque y tamaño de grano gaussiano además de sus desviaciones estándar de kernel [18]. Estos parámetros deben ser variantes, ya que será diferente el caso donde la cámara utilizada para la captura de video tenga poca resolución a una con alta resolución.

b. Método Canny para la estimación de bordes

Para la estimación de bordes, OpenCV provee la función Canny() que recibe como parámetros la imagen filtrada anteriormente (Gaussian Blur) y los parámetros de tamaño de apertura (valor mínimo y valor máximo) [19] [20].

C. Aproximación de Objetos

Para la aproximación de objetos, en el caso de polígonos con esquinas, OpenCV provee la función findContours. Esta recibe diversos parámetros entre los cuales está la imagen previa con detección de bordes imagen, el modo de recuperación de contorno, y el método de aproximación de contorno, es decir cuántos píxeles se almacenan [21]. Finalmente, provee la función aproxPolygon() el cual determinara según el número de esquinas encontradas, saber qué tipo de figura geométrica forma.

Para la aproximación de objetos circulares, mediante la transformación de Hough Circle (CHT), OpenCV provee la función HoughCircles(). Esta recibe varios parámetros, entre los cuales la imagen identificada los contornos, el método de detección (gradientes), la distancia mínima entre círculos y los rangos máximo y mínimo de radios de círculos a identificar [22].

D. Rastreo de objetos a través del cuadro de video.

Para el rastreo de objetos, OpenCV no provee ninguna función, por lo cual se implementa manualmente cálculos de variaciones de los centros de las figuras identificadas. Estos cálculos se realizan con una cola de coordenadas almacenadas en cada Frame, que provee las variaciones de movimiento.

E. Dibujo de las áreas estáticas y en movimiento.

Para el dibujo de las figuras, que se realizan en cada Frame o imagen, OpenCV provee la función drawContourn [23] para el caso de polígonos con esquinas y Circle [24] en caso de las figuras curvas. Estas dos comparten parámetros, entre ellos: Frame donde dibujar, coordenadas y tipo de relleno, su única diferencia radica en el parámetro radio para figuras circulares.

F. Grabación de Video

Finalmente, la funcionalidad de grabar video con las figuras identificadas y el tracking de su movimiento, se realiza mediante la funcionalidad VideoWriter() que define la ubicación donde se guardará el archivo resultante, su nombre, su formato de video, sus frames por segundo y sus dimensiones [25].

V. RESULTADOS

Video Demostrativo: https://youtu.be/2vl1zH3-BvM

El resultado de la etapa "Tratamiento y Segmentación de Imágenes" se identifica en las figuras 7 y 8, donde se identifica el Filtro Gaussiano para el filtrado de imágenes digitales y el Método Canny para la estimación de bordes respectivamente. Estas aplicadas al Frame original (Imagen 6). Al realizar estos filtros y detección de borde, se debe configurar para cada caso de detección, según el tipo de cámara disponible, como se explicó en la sección IV.B (Metodología). Esta configuración se realiza mediante los trackbar de las imágenes 9 para la variación del tamaño de apertura para el método Canny para detección de bordes y la imagen 10 el trackbar para la variación de núcleo de desenfoque y tamaño de grano gaussiano para el algoritmo de filtrado gaussiano.

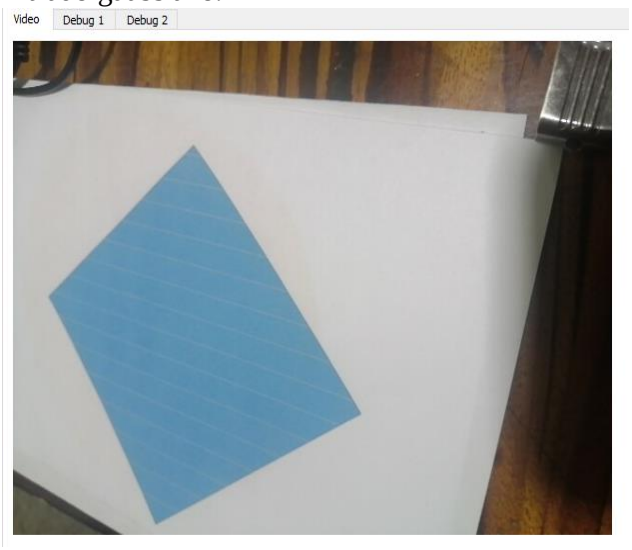

Imagen 6: Frame original

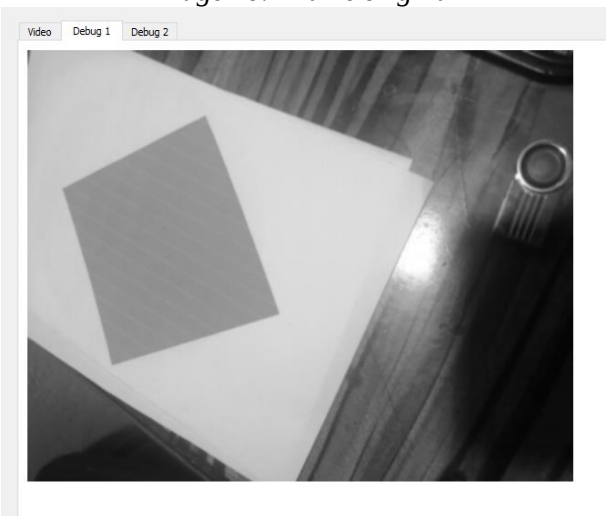

Imagen 7: Filtro Gaussiano para el filtrado de imágenes digitales

Imagen 8: Método Canny para la estimación de bordes

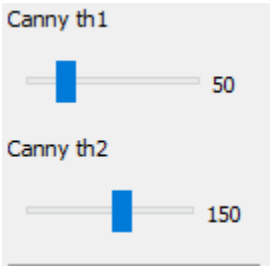

Imagen 9: Trackbar para la variación del tamaño de apertura para el método Canny para detección de bordes

Imagen 10: Trackbar para la variación de núcleo de desenfoque y tamaño de grano gaussiano para el algoritmo de filtrado gaussiano.

El resultado de las etapas de Aproximación de Objetos se visualiza en la imagen 11, Rastreo de objetos a través del cuadro de video en la imagen 12 y de dibujo de las áreas estáticas y en movimiento en la imagen 13.

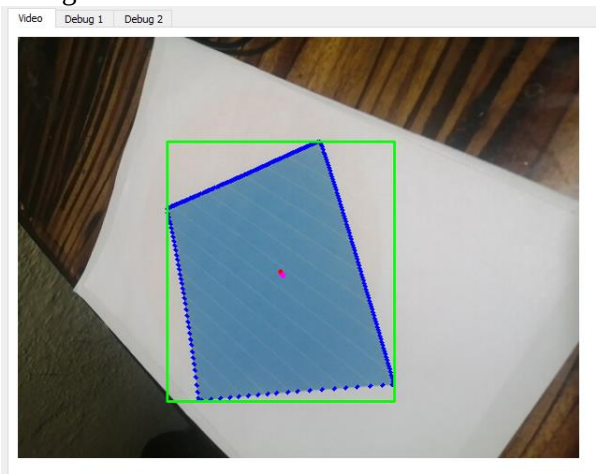

Imagen 11: Aproximación de Objetos

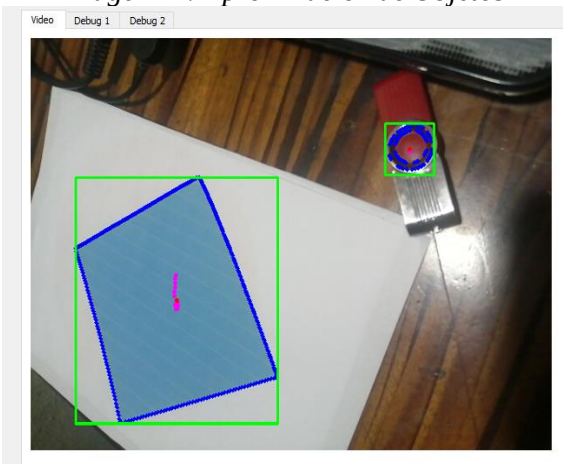

Imagen 12: Rastreo de objetos a través del cuadro de video.

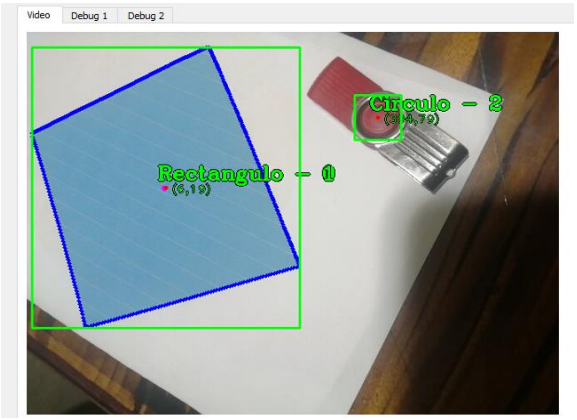

Imagen 13: Dibujo de las áreas estáticas y en movimiento.

VI. CONCLUSIONES

Este proyecto permitió una exploración basta para tener un conocimiento de las herramientas existentes de visión por computadora. Además, se puede concluir que:

- Existen metodologías óptimas para identificar objetos de una imagen digital en 2D. Sin embargo, estas metodologías se ven limitadas según las imágenes que alimenten los algoritmos. La calibración de los algoritmos se considera una parte fundamental del éxito de estos algoritmos de rastreo de movimiento.
- El ruido de las imágenes digitales, es inevitable, una de las técnicas más sencillas pero eficaces para tratar este ruido es el Método Gaussiano. Múltiples herramientas proveen este método, y en este proyecto fue el apropiado frente a otras alternativas.
- El reconocimiento de bordes o contornos puede afrontar por distintas metodologías, sin embargo, el método Canny, elegido y probado en este proyecto presenta ventajas por su eficiencia computacional además de su fácil implementación.
- La aproximación de figuras geométricas, mediante los métodos desarrollados en la API OpenCV, presentan limitantes de estimación de puntos característicos además de un área limitada. Estos componentes deben calibrarse según el dominio del problema.
- Para un mejor rastreo de objetos se debe tener cámaras o fuentes de datos con altos frames por segundos. La práctica contempló cámaras de 7 y 30 fps y el Screen recorder con 100 fps. Obteniendo mejores resultados con el mayor número de fps.
- Se concluye que los conceptos, de Algoritmos de Canny, Algoritmos de Identificación de Bordes, los Algoritmos

empleados para la Histéresis, Algoritmos para el rastreo de movimiento provistos por la librería de OpenCV, además del conocimiento teórico que cada una conlleva son fundamentales, para el empleo de tareas de Visión por Computadora en entornos de la vida cotidiana.

VII. RECOMENDACIONES

Este proyecto logró los objetivos principales y específicos, sin embargo, se identificó varias limitaciones, las cuales se detalla a continuación y se recomienda que hacer en estos casos:

- **Filtrado y detección de bordes y contornos**, para estos casos donde se usaron los métodos gaussianos, Canny y estimación de contorno se identificó que la resolución de la imagen es un limitante. Si bien una imagen a mayor resolución da mayores detalles y mayor ruido, se puede estimar de mejor manera los bordes, debido a los contrastes y brillos que se identifica. Se recomienda usar cámara con una resolución media - alta para una mejor identificación de figuras.
- **Detección de figuras**, este caso se vio marcado en la identificación de círculos. La técnica de Hough Circle, permite identificar cualquier tipo de círculo de una imagen, si bien es una ventaja, también es una desventaja, debido a falsos círculos o círculos fantasmas que se identifica. La recomendación en este caso, es la identificación de contornos no por el método Canny, sino otros algoritmos de estimación según color, que son más precisos.
- **Limitación de FPS**, este limitante se agudiza al buscar trackear el movimiento de una figura. Esto se debe a que a mayor FPS, mejor continuidad de movimiento y mejor tracking. En esta práctica se utilizan 2 tipos de cámara y la grabación de pantalla, estas tienen 7 fps, 30 fps y 100 fps respectivamente. Esto determinó que a mayor fps mejor tracking de un objeto.

VIII. BIBLIOGRAFÍA

[1] Elsevier. (2016, marzo 31). GPU-based Parallel Implementation of Swarm Intelligence Algorithms - 1st Edition. Recuperado 13 de mayo de 2020, de https://www.elsevier.com/books/T/A/9780128093627

[2] Editores Wikipedia. (2020) "Digital Image Processing". Recuperado 23 de mayo de 2020, de https://en.wikipedia.org/wiki/Digital_image_processing

[3] Editores TUTORIALSPOINT. (2020) "OpenCV - Gaussian Blur". Recuperado 23 de mayo de 2020, de https://www.tutorialspoint.com/opencv/opencv_gaussian_blur.htm

[4] Editores. (2020) "". Recuperado 23 de mayo de 2020, de https://docs.gimp.org/2.10/es/gimp-filter-gaussian-blur.html

[5] Editores Wikipedia. (2020) "Gaussian Blur". Recuperado 23 de mayo de 2020, de https://en.wikipedia.org/wiki/Gaussian_blur

[6] Editores QT Designer. (2019) "Gaussian Blur". Recuperado 23 de mayo de 2020, de https://www.google.com/url?sa=i&url=https%3A%2F%2Fdoc.qt.io%2Fqt3dstudio%2Fgaussian-blur-effect.html&psig=AOvVaw1vGJTh_23am5Mn2mN6FRPg&ust=1591171139929000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCPCrgs3U4ukCFQAAAAAdAAAAABAD

[7] Editores Wikipedia. (2020) "Edge Detection". Recuperado 23 de mayo de 2020, de https://en.wikipedia.org/wiki/Edge_detection

[8] Editores Wikipedia. (2020) "Algoritmo de Canny". Recuperado 23 de mayo de 2020, de https://es.wikipedia.org/wiki/Algoritmo_de_Canny

[9] Sofiane Sahir. (2018) "Canny Edge Detection Step by Step in Python — Computer Vision". Recuperado 23 de mayo de 2020, de https://miro.medium.com/max/566/1*XAgKINgc2c2gNa2nV3zbNQ.png

[10] Documentación OpenCV. (2019) "Py Contourns". Recuperado 23 de mayo de 2020, de https://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

[11] Editores Wikipedia. (2020) "Circle Hough Transform". Recuperado 23 de mayo de 2020, de https://en.wikipedia.org/wiki/Circle_Hough_Transform

[12] Documentación OpenCV. (2019) "Py Contourns". Recuperado 23 de mayo de 2020, de https://www.google.com/url?sa=i&url=https%3A%2F%2Fdocs.opencv.org%2Ftrunk%2Fd4%2Fd73%2Ftutorial_py_contours_begin.html&psig=AOvVaw2Ph4CbEPzv0_sMqjr0MGGQ&ust=1591172498600000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCLDI9dPZ4ukCFQAAAAAdAAAAABAD

[13] Benjamin Smith. (2017) "Hough Circle Transform". Recuperado 23 de mayo de 2020, de https://imagej.net/Hough_Circle_Transform

[14] Tao Peng. (2020) "Detect circles with various radii in grayscale image via Hough Transform". Recuperado 23 de mayo de 2020, de https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.mathworks.com%2Fmatlabcentral%2Ffileexchange%2F9168-detect-circles-with-various-radii-in-grayscale-image-via-hough-transform&psig=AOvVaw3v71s53kFqyapW-ov6XVYP&ust=1591173192387000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCNDbgJ_c4ukCFQAAAAAdAAAAABAD

[15] Adrian Rosebrock. (2015) "Ball Tracking with OpenCV". Recuperado 23 de mayo de 2020, de https://www.pyimagesearch.com/wp-content/uploads/2015/09/ball-tracking-tails-comparison.jpg

[16] Documentación OpenCV. (2020) "Capture Video from Camera". Recuperado 23 de mayo de 2020, de https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

[17] Editores Documentación Python. (2019) "Examples of Python mss". Recuperado 23 de mayo de 2020, de https://python-mss.readthedocs.io/examples.html

[18] Editores Tutoriales Kart. (2020) "Image Smoothing using OpenCV Gaussian Blur". Recuperado 23 de mayo de 2020, de https://www.tutorialkart.com/opencv/python/opencv-python-gaussian-image-smoothing/

[19] Editores GeekforGeeks. (2020) "Real-Time Edge Detection using OpenCV in Python | Canny edge detection method". Recuperado 23 de mayo de 2020, de https://www.geeksforgeeks.org/real-time-edge-detection-using-opencv-python/#:~:text=In%20this%20article%2C%20the%20popular,value%20as%20last%20two%20arguments

[20] Documentation Python. (2018) "Canny Edge Detection". Recuperado 23 de mayo de 2020, de https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

[21] Documentation OpenCV. (2020) "Contours: Getting Started". Recuperado 23 de mayo de 2020, de https://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

[22] Documentación OpenCV. (2018) "Feature Detection". Recuperado 23 de mayo de 2020, de https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles

[23] Documentation OpenCV. (2016) "Structural Analysis and Shape Descriptors". Recuperado 23 de mayo de 2020, de https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=drawcontours

[24] Abhishek Thakur. (2013) "Draw a circle over image OpenCV". Recuperado 23 de mayo de 2020, de https://stackoverflow.com/questions/16484796/draw-a-circle-over-image-opencv

[25] Documentation OpenCV. (2020) "Getting Started with Videos". Recuperado 23 de mayo de 2020, de https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html