**1.- ¿Qué es OpenCV?**

OpenCV (Open Source Computer Vision) es una librería software open-source de visión artificial y machine learning. OpenCV provee una infraestructura para aplicaciones de visión artificial.

OpenCV tiene una licencia BSD, lo que permite utilizar y modificar el código, tiene una comunidad de más de 47000 personas y más de 7 millones de descargadas. Es una librería muy usada a nivel comercial, desde Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota, Applied Minds, VideoSurf, Zeitera, etc.

La librería tiene más de 2500 algoritmos, que incluye algoritmos de machine learning y de visión artificial para usar. Estos algoritmos permiten identificar objetos, caras, clasificar acciones humanas en vídeo, hacer tracking de movimientos de objetos, extraer modelos 3D, encontrar imágenes similares, eliminar ojos rojos, seguir el movimiento de los ojos, reconocer escenarios, etc.

Se usa en aplicaciones como la detección de intrusos en vídeos, monitorización de equipamientos, ayuda a navegación de robots, inspeccionar etiquetas en productos, entre otros.

OpenCV está escrito en C++, tiene interfaces en C++, C, Python, Java y MATLAB interfaces y funciona en Windows, Linux, Android y Mac OS.

**2.- Instalación de OpenCV en MacOS**

1. Instalar XCode

Instale XCode desde la App Store.

Si XCode disponible en App Store no es compatible con su sistema operativo:

* Encuentre la versión XCode compatible con su sistema operativo en esta tabla https://en.wikipedia.org/w/index.php?title=Xcode#Version\_comparison\_table
* Vaya a esta página web <https://developer.apple.com/download/more/>
* Inicia sesión si tienes una cuenta de desarrollador de Apple o crea tu cuenta e ingresa.
* Busque xcode y descargue la versión compatible con su sistema operativo.
* Instalar XCode.
* Después de la instalación, abra XCode y acepte la licencia xcode-build cuando la solicite.

2. Instalar OpenCV

Ahora que XCode se ha instalado, pasaremos a la instalación de OpenCV.

Primero, instalemos Homebrew .

ruby -e "$(curl –fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install>)"

brew update

También añadiremos Homebrew a PATH.

# Add Homebrew path in PATH

echo "# Homebrew" >> ~/.bash\_profile

echo "export PATH=/usr/local/bin:$PATH" >> ~/.bash\_profile

A continuación, instalaremos los requisitos: Python 3 , CMake y Qt 5 .

brew install python3

brew install cmake

brew install qt5

QT5PATH=/usr/local/Cellar/qt/5.11.2\_1

También guardaremos el directorio de trabajo actual en la variable cwd y la versión OpenCV (master) en cvVersion .

cwd=$(pwd)

cvVersion="master"

# Clean build directories

rm -rf opencv/build

rm -rf opencv\_contrib/build

# Create directory for installation

mkdir installation

mkdir installation/OpenCV-"$cvVersion"

Ahora, instalemos las bibliotecas de Python y creamos el entorno de Python.

sudo -H pip3 install -U pip numpy

# Install virtual environment

sudo -H python3 -m pip install virtualenv virtualenvwrapper

VIRTUALENVWRAPPER\_PYTHON=/usr/local/bin/python3

echo "VIRTUALENVWRAPPER\_PYTHON=/usr/local/bin/python3" >> ~/.bash\_profile

echo "# Virtual Environment Wrapper" >> ~/.bash\_profile

echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.bash\_profile

cd $cwd

source /usr/local/bin/virtualenvwrapper.sh

############ For Python 3 ############

# create virtual environment

mkvirtualenv OpenCV-"$cvVersion"-py3 -p python3

workon OpenCV-"$cvVersion"-py3

# now install python libraries within this virtual environment

pip install cmake numpy scipy matplotlib scikit-image scikit-learn ipython dlib

# quit virtual environment

deactivate

######################################

A continuación, vamos a clonar los repositorios de github de OpenCV.

|  |
| --- |
| git clone https://github.com/opencv/opencv.git  cd opencv  git checkout master  cd ..    git clone https://github.com/opencv/opencv\_contrib.git  cd opencv\_contrib  git checkout master  cd ..    cd opencv  mkdir build  cd build |

Finalmente, usaremos CMake para construir OpenCV.

cmake -D CMAKE\_BUILD\_TYPE=RELEASE \

            -D CMAKE\_INSTALL\_PREFIX=$cwd/installation/OpenCV-"$cvVersion" \

            -D INSTALL\_C\_EXAMPLES=ON \

            -D INSTALL\_PYTHON\_EXAMPLES=ON \

            -D WITH\_TBB=ON \

            -D WITH\_V4L=ON \

            -D OPENCV\_SKIP\_PYTHON\_LOADER=ON \

            -D CMAKE\_PREFIX\_PATH=$QT5PATH \

            -D CMAKE\_MODULE\_PATH="$QT5PATH"/lib/cmake \

            -D OPENCV\_PYTHON3\_INSTALL\_PATH=~/.virtualenvs/OpenCV-"$cvVersion"-py3/lib/python3.7/site-packages \

        -D WITH\_QT=ON \

        -D WITH\_OPENGL=ON \

        -D OPENCV\_EXTRA\_MODULES\_PATH=../../opencv\_contrib/modules \

        -D BUILD\_EXAMPLES=ON ..

make -j$(sysctl -n hw.physicalcpu)

make install

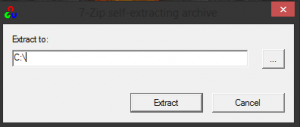
cd $cwd

¡Y eso es! Por ahora debería tener OpenCV instalado con éxito en su sistema.

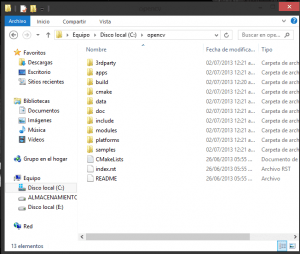
**3.- Instalación de OpenCV en MS Windows**

Para Comenzar descargamos OpenCV desde la pagina de descargas de Opencv(link anterior) y seleccionamos la version mas reciente (a la fecha es la 2.4.6) y comenzamos a instalarlo.

Unicamente nos pide seleccionar la carpeta donde quieres que se instale OpenCV.



Nosotros elegimos C:/ ahora tendremos instalado OpenCV en C:/opencv



Ahora registraremos en las variables del sistema los directorios de OpenCV

Vamos a WINDOWS XP

* Inicio -> Panel de control -> Sistema -> Opciones avanzadas
* Haga clic en Variables de entorno, en Variables del sistema, busque PATH y haga clic en él.
* En las ventanas Editar, modifique PATH agregando la ubicación de la clase al valor de PATH. Si no dispone del elemento PATH, puede optar por agregar una nueva variable y agregar PATH como el nombre y la ubicación de la clase como valor.

WINDOWS 7

* Seleccione Equipo en el menú Inicio
* Seleccione Propiedades del sistema en el menú contextual
* Haga clic en Configuración avanzada del sistema > ficha Opciones avanzadas
* Haga clic en Variables de entorno, en Variables del sistema, busque PATH y haga clic en él.
* En las ventanas Editar, modifique PATH agregando la ubicación de la clase al valor de PATH. Si no dispone del elemento PATH, puede optar por agregar una nueva variable y agregar PATH como el nombre y la ubicación de la clase como valor.

WINDOWS 8

* Presiona la tecla de Windows
* Escriba «variables de entorno del sistema» y cambiese a «Configuracion»
* Haga clic en Variables de entorno, en Variables del sistema, busque PATH y haga clic en él.
* En las ventanas Editar, modifique PATH agregando la ubicación de la clase al valor de PATH. Si no dispone del elemento PATH, puede optar por agregar una nueva variable y agregar PATH como el nombre y la ubicación de la clase como valor.

Ahora en el campo «Valor de la Variable» agregaremos estas rutas

;C:opencv;C:opencvbuild;

en caso de que no hayas cambiado la ruta de instalacion de OpenCV y listo estara Instalado OpenCV en Windows

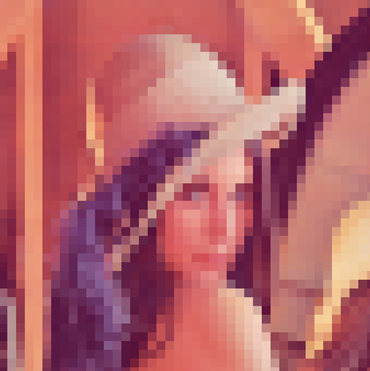
**4.- Trabajando con imágenes**

Con OpenCV contamos con diferentes funciones que nos permiten procesar las imágenes de entrada, ya sea para resaltar o descartar zonas de la misma para su posterior análisis o simplemente si deseamos transformar la imagen original para mejorar el color, añadir un efecto, mesclar la imagen, etc., en este tutorial veremos algunas de estas funciones y aprenderemos a utilizarlas.

En la primera parte veremos algunos filtros creados por nosotros en distintos tutoriales, luego veremos algunas funciones incluidas en la biblioteca.

**Pixelar una imagen**

Para lograr el efecto pixelado de una imagen analizaremos cada uno de los píxeles que componen la imagen y calculamos el valor promedio de los píxeles vecinos, para esto no utilizaremos una función OpenCV, creamos nuestra propia implementación, si deseas más detalles del algoritmo visita: pixelar una imagen con OpenCV.



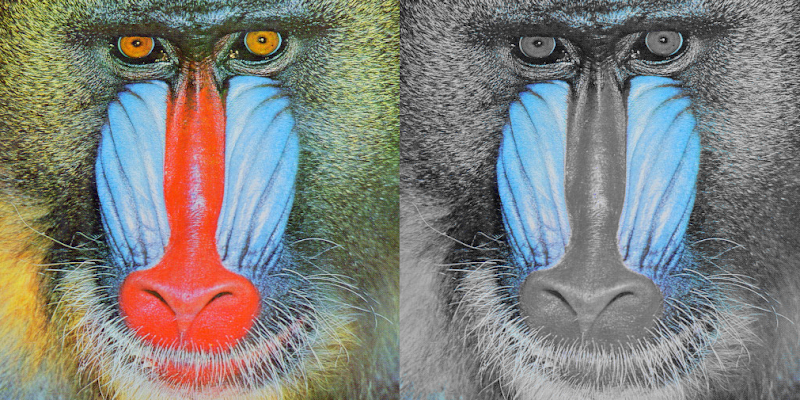
**Filtro Twirl**

Este es un filtro que aplica una transformación 2D a la posición de cada uno de los píxeles de la imagen, este es otro de los filtros creados en nuestros tutoriales, puedes verlo en: OpenCV filtro twirl, nuestro algoritmo nos permite variar el ángulo para obtener resultados diferentes, la salida sería:

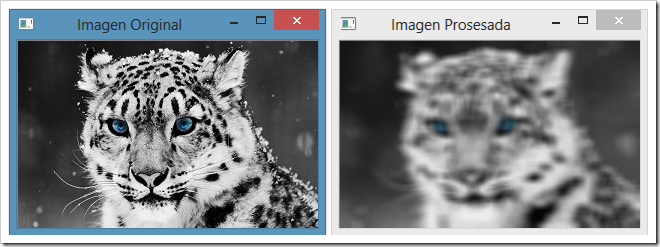


**Acentuar color**

Usaremos este filtro cuando deseemos resaltar un área de un color en específico, las áreas de la imagen que nos estén en el rango de color indicado se convertirán a escala de grises, de esta manera logramos resaltar solo aquellas áreas que estén pintadas con el color indicado, más información en: resaltar color con OpenCV.



Ahora veremos un grupo de funciones proporcionadas por OpenCV que nos facilitarán el procesamiento y análisis de imágenes, mencionaremos la función, los parámetros que requiere la misma y daremos un ejemplo del resultado que obtendremos al aplicar el correspondiente algoritmo.



**GaussianBlur**

GaussianBlur(img, dst, Size(13,7), 8);

Esta función la usamos para aplicar un blur a una imagen utilizando un filtro gaussiano, los parámetros requeridos son:

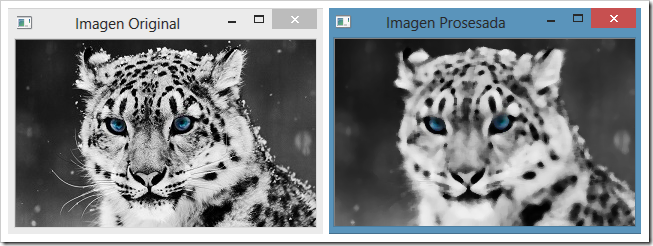
1. img: imagen original.
2. dst: matriz donde se almacenara la nueva imagen.
3. Size(width, height): tamaño del kernel, deben ser positivos e impares.
4. 8: desviación estándar en el eje X.

**MedianBlur**

cv::medianBlur(img, dst, 5);

Aplica un blur a la imagen indicada usando el filtro de mediana, los parámetros para esta función son:

1. img: imagen original o de entrada.
2. dst: objeto Mat donde guardaremos la salida.
3. 5: tamaño del kernel (ksize) debe ser mayor que 1 e impar.

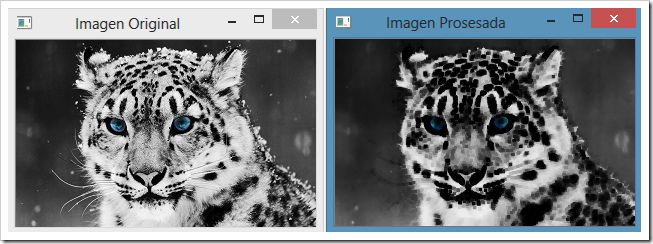


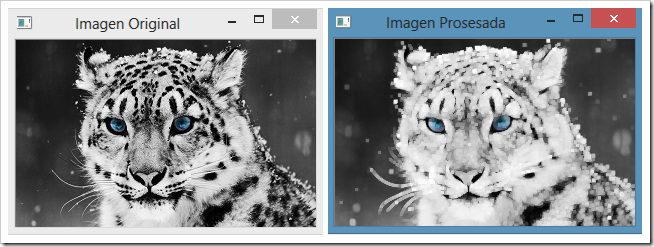
**Erode y Dilate**

cv::erode(img, dst, krl);

cv::dilate(img, dst, krl);

Estos son funciones que aplican las operaciones morfológicas de erosión y dilatación respectivamente, normalmente son usadas sobre imágenes binarias, pero también se pueden utilizar en imágenes a color como en el siguiente ejemplo:



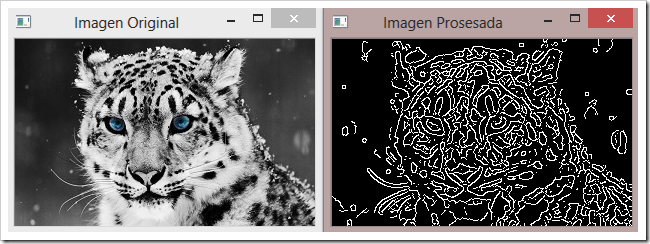


**Canny (Detección de bordes)**

cv::Canny(src, dst, t1, t2);

Esta y otras funciones proporcionadas por la biblioteca nos permiten detectar los bordes o contornos de una imagen, el resultado se almacena en una imagen binaria.

1. src: imagen de entrada.
2. dst: imagen de salida , será una imagen de un canal de 8 bits.
3. t1, t2: son los valores de tolerancia, nos ayudan a limitar los bordes detectados.



Guardar una imagen

Una vez hemos procesado la imagen tal vez nos interese guardar en disco una copia de la imagen procesada, OpenCV nos proporciona la función cv::imwrite() para almacenar una imagen en un archivo indicado, disponemos de distintos formatos como JPG, PNG, TIFF.

vector<int> compression\_params;

compression\_params.push\_back(CV\_IMWRITE\_PNG\_COMPRESSION);

compression\_params.push\_back(9);

imwrite("imagen.png", dst, compression\_params);

**5.- Desplegando una imagen**

Aquí tenéis un programa para cargar y mostrar una imagen en OpenCV:

#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <iostream>

using namespace cv;

using namespace std;

int main( int argc, char\*\* argv )

{

  if( argc != 2) {

    cout <<" Usage: mostrar\_imagen <archivo de imagen>" << endl;

    return -1;

  }

  Mat image;

  image = imread(argv[1], CV\_LOAD\_IMAGE\_COLOR);

  if(! image.data ) {

    cout <<  "Could not open or find the image" << std::endl ;

    return -1;

}

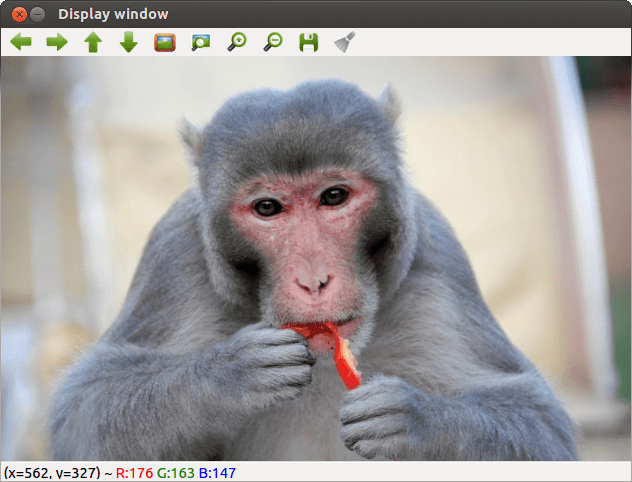
  namedWindow( "Display window", CV\_WINDOW\_AUTOSIZE );

  imshow( "Display window", image );

  waitKey(0);

  return 0;

}



**6.- Capturando una imagen**

OpenCV emplea el comando **CvCaptureFromCAM** para accesar a la cámara web conectada vía USB a la computadora. El parámetro **CV\_CAP\_ANY** , nos facilita que la función tome la cámara web que esté conectada en ese instante. El identificador **pCapture** toma un valor de **TRUE** si la cámara es hallada y un valor de **FALSE** si esta tiene algún problema o simplemente no hay cámara web

conectada. El comando **cvQueryFrame**, nos proporciona la captura de un cuadro de imagen de la cámara web, si la imagen se obtiene adecuadamente el identificador **pVideoFrame** arroja un valor de **TRUE** de otra forma es **FALSE**.

Para guardar la imagen capturada se emplea la función: **cvSaveImage** . Está espera 2 parámetros, el nombre de archivo y el cuadro de imagen capturado. Algunas ocasiones dependiendo de la forma de captura de la imagen, se debe agregar un tercer parámetro que indica el número índice de cámara web que se está empleando, en nuestro caso -0- .

Finalmente, **cvReleaseCapture** se encarga de cerrar el objeto de captura de video, y dejar libres recursos, se le da, de parámetro un objeto del tipo de clase **CvCapture**.

A continuación, se da un ejemplo, de captura de cuadro de imagen, el cual está escrito de tal forma que pueda capturar dos cuadros de imagen.

#include <stdio.h>  
#include "cv.h"  
#include "highgui.h"  
#include <string.h>  
int main(int argc, char \*\* argv)  
{  
CvCapture \* pCapture = 0;  
IplImage \* pVideoFrame = 0;  
int i;  
char filename[50];  
// Inicializar video de captura  
pCapture = cvCaptureFromCAM( CV\_CAP\_ANY );  
if( !pCapture )  
{  
fprintf(stderr, "Ocurrio una falla al momento de iniciar la captura de video.\n");  
return -1;  
}  
// Captura dos cuadros de imagen y escribelos como archivos  
for(i=0; i<2; i++)  
{  
pVideoFrame = cvQueryFrame( pCapture );  
if( !pVideoFrame )  
{  
fprintf(stderr, "Ocurrio una falla al obtener un cuadro de imagen\n");  
}  
// Guarda el cuadro de imagen del video capturado como un archivo imagen  
sprintf(filename, "CuadroImagen%d.jpg", i+1);  
if( !cvSaveImage(filename, pVideoFrame,0) )  
{  
fprintf(stderr, "Ocurrio un error al momento de guardar el archivo %s\n", filename);  
}  
// IMPORTANTE: NO quites o modifiques la imagen que se regresa.  
// de cvQueryFrame() !  
}  
// Termina la captura de video y libera recursos  
cvReleaseCapture( &pCapture );  
return 0;  
}

**7.- Abriendo un archivo de imagen**

Leer y guardar archivos de imagen

La tarea más simple y común que realizaremos será cargar imagen a partir de una archivo en disco, para hacerlo disponemos de la función imread(), mientras que imwrite() nos permitirá guardar una imagen.

Al cargar una imagen esta es almacenada en memoria usando el formato de color BGR, por defecto, este es igual al espacio de color RGB pero con los canales invertidos, aquí se almacena la información para cada uno de los canales azul, verde y rojo.

Opcionalmente se puede indicar el modo, por defecto, IMREAD\_COLOR, si deseamos cargar la imagen a escala de grises usaremos, IMREAD\_GRAYSCALE, mientras que el modo IMREAD\_UNCHANGED cargará la imagen en el formato de color definido por el archivo.

**import** **cv2**

*# cargar el archivo PNG indicado*

img = cv2.imread('image.png', cv2.IMREAD\_GRAYSCALE)

*# guardar la imagen en formato JPG*

cv2.imwrite('save.jpg', img)

Al guardar la imagen se utilizará el formato definido por la extensión que le demos al nombre de la imagen, siempre que el mismo este soportado, algunos de estos formatos son: PNG, BMP, TIFF, JPG, otros.

Mostrar una imagen

Cuando hayamos cargado una imagen en memoria es posible mostrarla en una ventana, para este propósito tenemos la función imshow(), con ella creamos una ventana y desplegamos la imagen deseada, debemos indicar el nombre y el objeto que deseamos mostrar, ejemplo:

**import** **cv2**

*# cargar el archivo PNG indicado*

img = cv2.imread('image.png', cv2.IMREAD\_GRAYSCALE)

*# mostrar la imagen en una ventana*

cv2.imshow('titulo', img)

*# esperar hasta que se presiona una tecla*

cv2.waitKey(0)

Al final vemos una llamada a waitKey(0), con esto impedimos que la ventana se cierre inmediatamente, el valor cero indica que se debe esperar indefinidamente hasta que se presione una tecla, si establecemos un valor mayor que cero este será la cantidad de milisegundos que se debe esperar.

Esta función devuelve el código ASCII de la tecla presionada, por ejemplo, 27 para la tecla ESC.

Cuando deseemos destruir una ventana usaremos destroyWindow('ventana') indicando el nombre de la ventana que deseamos eliminar, si tenemos varias ventanas y deseamos destruirlas todas disponemos de destroyAllWindows()

**8.- Capturando video**

Capturar video de la webcam

No solo podemos trabajar con imágenes cargadas desde archivos, también podemos leer imágenes desde la cámara web de nuestro PC, si capturamos una serie de imágenes sucesivas tendremos un video en tiempo real, veamos cómo se hace.

Para capturar imágenes provenientes de la cámara usaremos un objeto VideoCapture(0) debemos indicar el índice de la cámara que deseamos utilizar, por ejemplo, si tenemos dos habilitadas, indicamos cero para la primera, uno para la siguiente y así sucesivamente.

En lugar de indicar el índice de cámara, también es posible indicar el nombre de un archivo de video, si el formato esta soportado este se visualizará.

**import** **cv2**

cap = cv2.VideoCapture(0)

**while**(True):

ret, frame = cap.read()

**if** ret:

cv2.imshow('video', frame)

**if** cv2.waitKey(1) & 0xFF == 27:

**break**

cap.release()

cv2.destroyAllWindows()

Para capturar un cuadro de video usamos cap.read() este devuelve un booleano que indica si la captura es válida y el objeto imagen que representa la captura, de aquí en adelanta tratamos la captura como si fuese cualquier otra imagen, al terminar usaremos cap.release() para liberar recursos.

Guardar un archivo de video

Para escribir un archivo de video necesitaremos un objeto VideoWriter indicaremos el nombre del archivo, el codec a utilizar, el fps (cuadros por segundo) y el tamaño del video.

Los codec que podemos utilizar dependen de cada sistema, es decir si no tienes los codec para FLV no podrás ver ni escribir en este formato, utilizaremos un código de 4 byte para identificar cada uno de ellos, puedes encontrar la lista en, fourcc.org, para indicarlo tenemos la función: cv2.VideoWriter\_fourcc('M','J','P','G') para MJPG.

**import** **cv2**

cap = cv2.VideoCapture('video.avi')

*# Definir el codec y crear el objeto VideoWriter*

fourcc = cv2.VideoWriter\_fourcc('M','S','V','C')

out = cv2.VideoWriter('output.avi', fourcc, 25.0, (720, 528))

**while**(True):

*# capturar el cuadro*

ret, frame = cap.read()

**if** ret:

*# procesar la captura, convertir a grises*

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

*# escribir el cuadro procesado*

out.write(gray)

*# mostar la captura actual*

cv2.imshow('video', gray)

*# esperar, si se presiona la tecla ESC salir*

**if** cv2.waitKey(1) & 0xFF == 27:

**break**

cap.release()

out.release()

cv2.destroyAllWindows()

**9.- Vista de la cámara**

Usando la biblioteca OpenCV podemos acceder a la cámara web o cualquier otro dispositivo de captura que tengamos instalado en nuestro sistema, cada una de las imágenes capturadas podrán almacenarse para su análisis o procesamiento en tiempo real si así lo deseamos, tenemos disponible una clase que nos servirá para guardar los videos previamente capturados y procesados, el formato de almacenamiento depende de las características habilitadas, pero puede ser MP4, AVI, WMV, etc., y otros si tenemos los códec.

Webcam desde OpenCV

Para acceder a la webcam lo hacemos de manera parecida a como mostramos un video solo que en lugar de especificar un archivo de video a la clase VideoCapture indicaremos el número de dispositivo o índice de cámara que deseamos usar, cero si tenemos solo una cámara.

Código Python

**from** **cv2** **import** \*

namedWindow("webcam")

vc = VideoCapture(0);

**while** True:

next, frame = vc.read()

imshow("webcam", frame)

**if** waitKey(50) >= 0:

**break**;

**10.- Grabando video**

Si deseamos grabar el video proveniente de la webcam, en disco, podemos hacerlo mediante la clase VideoWriter, en el constructor o el método open debemos indicar el nombre con que deseamos guardar el archivo con la extensión, el códec que usaremos, los cuadros por segundo y la resolución del video.

Por ejemplo, para grabar la captura de la webcam en un video AVI:

VideoWriter wtr("webcam.avi", CV\_FOURCC('M','J','P','G'), 30, Size(640,480));

Para indicar el códec debemos indicar las cuatro letras que lo identifican:

* CV\_FOURCC('P','I','M','1') para codec MPEG-1 codec.
* CV\_FOURCC('M','J','P','G') para códec motion-jpeg.

Para finalizar solo nos falta escribir cada cuadro, lo podemos hacer de dos modos:

wtr.write(dest);

También es válido:

wtr << dest;

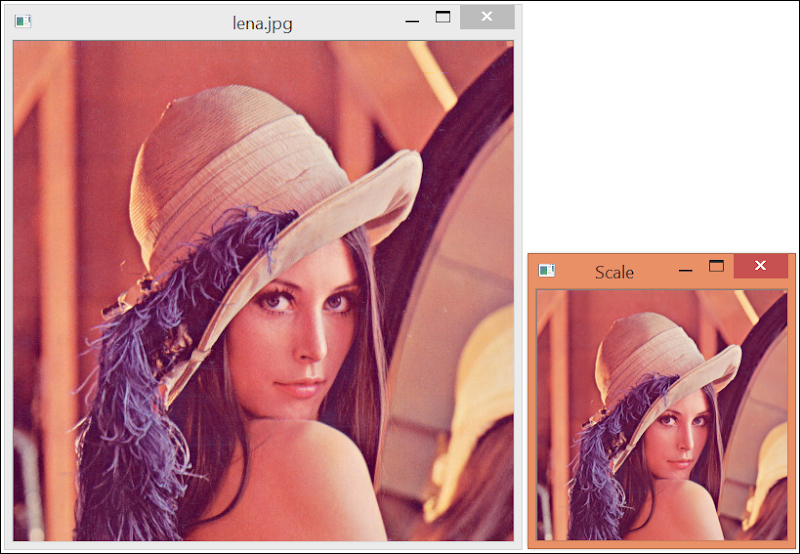
Recordemos que este código debe ir dentro del bucle, después de procesar la imagen, en este caso dest en la imagen ya procesada que deseamos guardar.

**11.- Transformación de imágenes**

Se muestra el estudio de las transformaciones geométricas que podemos aplicar para cambiar la posición, rotación, escala, o inclinación de una imagen, este tipo de transformación no cambia el contenido de la imagen, la deformación es produce por un cambio en la posición de los pixeles que la componen.

**Escalar o cambio de tamaño**

Para realizar esta transformación haremos uso de la función OpenCV llamada cv2.resize() con ella podemos cambiar el tamaño de la imagen de entrada, los parámetros que requiere esta función son: la imagen de entrada, el nuevo tamaño y el método de interpolación utilizado.



**import** **cv2**

**import** **numpy** **as** **np**

src = cv2.imread('lena.jpg')

dst = cv2.resize(src, (256, 256), interpolation=cv2.INTER\_CUBIC)

cv2.imshow('lena.jpg', src)

cv2.imshow('Scale', dst)

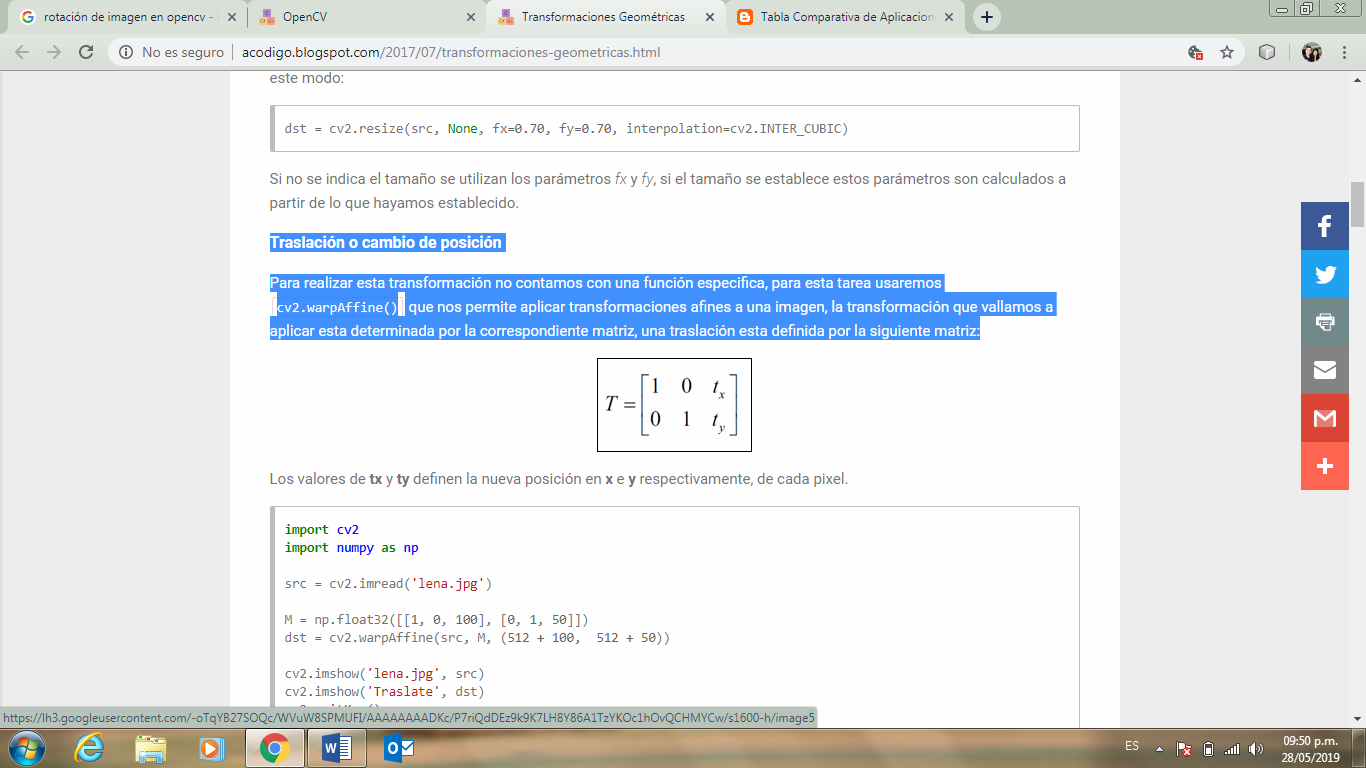
cv2.waitKey()

Si lo deseamos no es necesario indicar el tamaño explícitamente, podemos indicar un factor de escala tanto para el ancho como para la altura, por ejemplo, si deseamos que la imagen tenga el 70% de su tamaño original lo hacemos de este modo:

dst = cv2.resize(src, None, fx=0.70, fy=0.70, interpolation=cv2.INTER\_CUBIC)  
Si no se indica el tamaño se utilizan los parámetros fx y fy, si el tamaño se establece estos parámetros son calculados a partir de lo que hayamos establecido.

**Traslación o cambio de posición**

Para realizar esta transformación no contamos con una función específica, para esta tarea usaremos cv2.warpAffine() que nos permite aplicar transformaciones afines a una imagen, la transformación que vallamos a aplicar está determinada por la correspondiente matriz, una traslación está definida por la siguiente matriz:



Los valores de tx y ty definen la nueva posición en x e y respectivamente, de cada pixel.

**import** **cv2**

**import** **numpy** **as** **np**

src = cv2.imread('lena.jpg')

M = np.float32([[1, 0, 100], [0, 1, 50]])

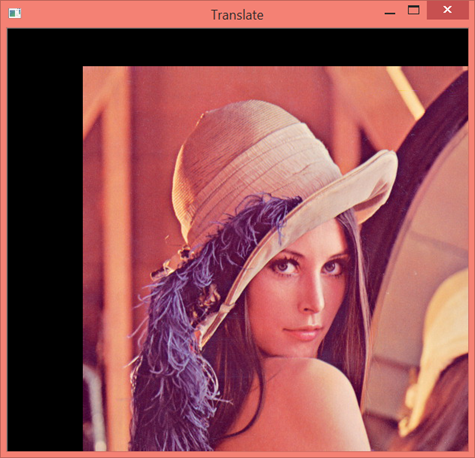
dst = cv2.warpAffine(src, M, (512 + 100, 512 + 50))

cv2.imshow('lena.jpg', src)

cv2.imshow('Traslate', dst)

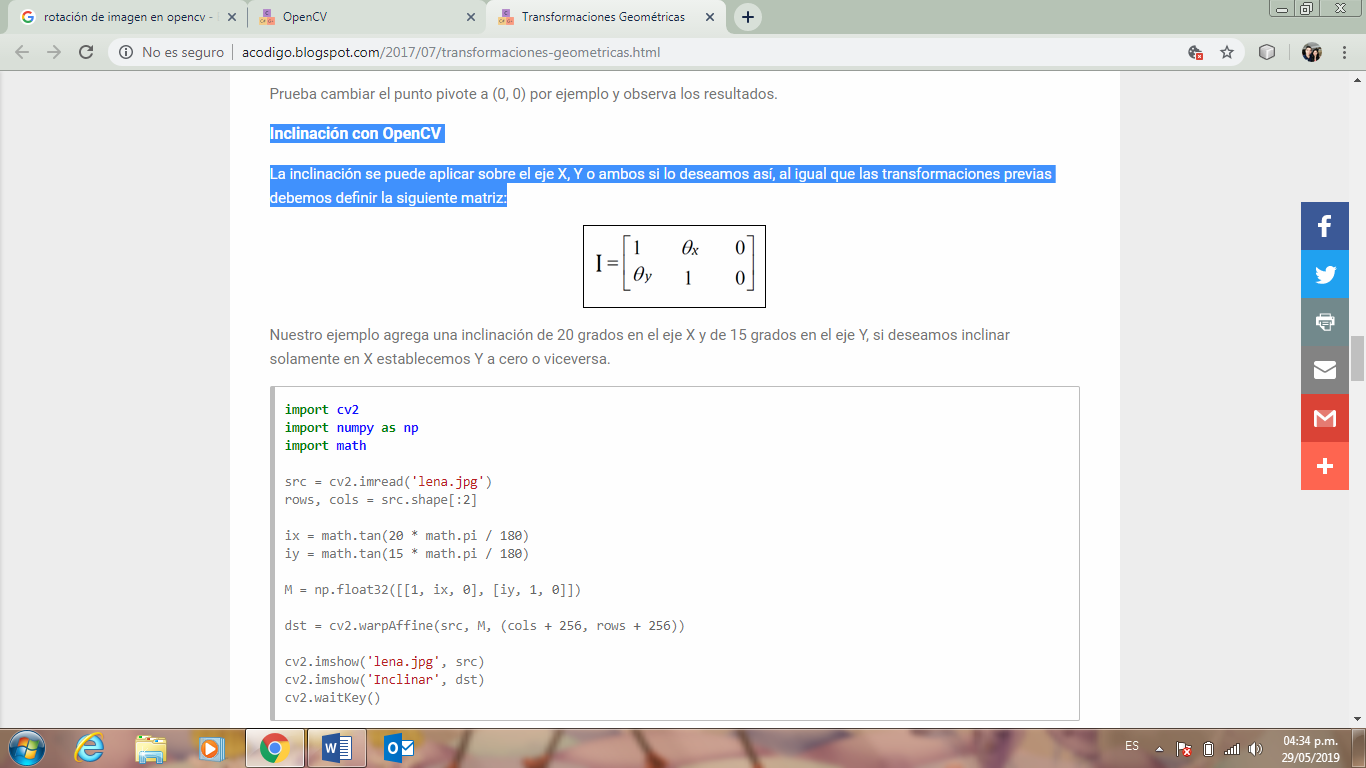
cv2.waitKey()

El último parámetro que usamos en esta función establece el tamaño de la nueva imagen, debemos tener en cuenta que la imagen será desplazada por lo que aumentamos el tamaño en la misma cantidad del desplazamiento para poder tener la imagen complete, un tamaño menor recortaría la imagen.



**Inclinación con OpenCV**

La inclinación se puede aplicar sobre el eje X, Y o ambos si lo deseamos así, al igual que las transformaciones previas debemos definir la siguiente matriz:



Nuestro ejemplo agrega una inclinación de 20 grados en el eje X y de 15 grados en el eje Y, si deseamos inclinar solamente en X establecemos Y a cero o viceversa.

**import** **cv2**

**import** **numpy** **as** **np**

**import** **math**

src = cv2.imread('lena.jpg')

rows, cols = src.shape[:2]

ix = math.tan(20 \* math.pi / 180)

iy = math.tan(15 \* math.pi / 180)

M = np.float32([[1, ix, 0], [iy, 1, 0]])

dst = cv2.warpAffine(src, M, (cols + 256, rows + 256))

cv2.imshow('lena.jpg', src)

cv2.imshow('Inclinar', dst)

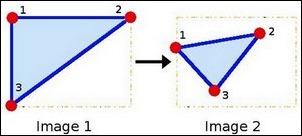
cv2.waitKey()

El resultado de la operación:



Debido la inclinación el tamaño de la imagen de destino debe ser mayor a la original para poder contener la imagen deformada.

Una forma diferente de calcular la matriz de transformación correspondiente es utilizar la función OpenCV llamada cv2.getAffineTransform() para utilizarle debemos indicar dos conjuntos de 3 puntos, el primero define el triangulo que abarca la imagen original, el segundo define la ubicación a donde deben moverse cada uno de estos puntos.



Por ejemplo, para generar una transformación similar a la anterior:

pt1 = np.float32([[0, 0],

[cols, 0],

[0, rows]])

pt2 = np.float32([[20, 20],

[cols - 120, rows / 4],

[cols / 4, rows - 120]])

M = cv2.getAffineTransform(pt1, pt2)

Cada pixel ubicado dentro del paralelogramo definido por los puntos de origen serán mapeados al paralelogramo definido por los puntos de destino.

Transformación en perspectiva

Esta transformación se calcula de manera similar a la anterior solo que deberemos indicar 4 puntos y usaremos las funciones cv2.getPerspectiveTransform() para obtener la matriz y cv2.warpPerspective() para aplicar la transformación.

**import** **math**

**import** **numpy** **as** **np**

**import** **cv2**

src = cv2.imread('left.jpg')

rows, cols = src.shape[:2]

pts1 = np.float32([[113, 137], [256, 136], [270, 337], [140, 377]])

pts2 = np.float32([[0, 0], [165, 0], [165, 223], [0, 223]])

M = cv2.getPerspectiveTransform(pts1, pts2)

dst = cv2.warpPerspective(src, M, (165, 223))

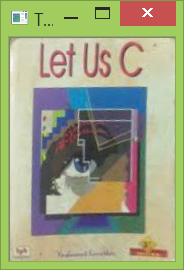
cv2.imshow('lena.jpg', src)

cv2.imshow('Transform', dst)

cv2.waitKey()



La primera serie de puntos es usada para definir el rectángulo que conforma el libro.



La segunda serie de puntos indica el rectángulo de destino en que deseamos ubicar el primer rectángulo, en este ejemplo logramos obtener una imagen de la portada del libro en perspectiva.

**12.- Girar imágenes**

Rotación con OpenCV

Para rotar una imagen usaremos la misma función que usamos para la traslación, solo debemos cambiar la matriz, anteriormente creamos la matriz manualmente pues era sencilla, la matriz de rotación puede ser un poca mas compleja por lo que nos apoyaremos en la función proporcionada por OpenCV cv2.getRotationMatrix2D() para crearla, ha dicha función debemos indicarle el punto pivote sobre el cual se realiza el giro o rotación, el ángulo y el factor de escala.

**import** **cv2**

**import** **numpy** **as** **np**

src = cv2.imread('lena.jpg')

rows, cols = src.shape[:2]

M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)

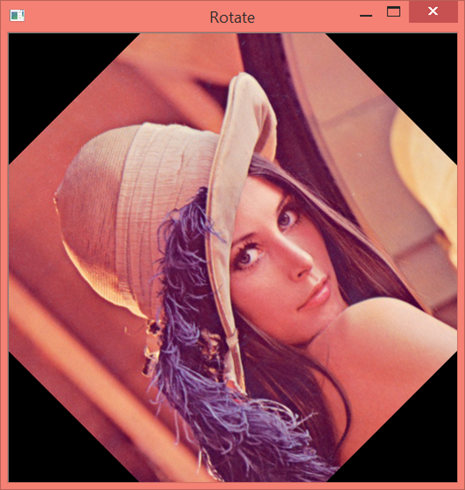
dst = cv2.warpAffine(src, M, (cols, rows))

cv2.imshow('lena.jpg', src)

cv2.imshow('Rotate', dst)

cv2.waitKey()

En este ejemplo el punto pivote de la rotación se encuentra en el centro de la imagen, el ángulo es de 45 grados y el factor de escala es 1, el resultado:



**13.- Cambiar el tamaño**

Puede cambiar el tamaño de una imagen. Esto es útil para reducir los recursos necesarios.

para procesar una imagen. Cuanto mayor sea la imagen, más memoria y CPU.

recursos necesitados. Para redimensionar una imagen, usamos el método resize (). los

los parámetros son la imagen que está escalando, las dimensiones deseadas como un

Tupla, y la interpolación.

La interpolación es el método matemático utilizado para determinar cómo

Manejar la eliminación o adición de píxeles. Recuerda, cuando trabajes con

Imágenes, realmente está trabajando con una matriz multidimensional que contiene

Información para cada punto, o píxel, que conforma la imagen. Cuando tú

Reduce una imagen, estás eliminando píxeles. Cuando amplías una imagen,

usted está agregando píxeles. La interpolación es el método por el cual esto ocurre.

Hay tres opciones de interpolación. INTER\_AREA es el mejor uso

por reducción. INTER\_CUBIC y INTER\_LINEAR son buenos para

Ampliando una imagen, siendo INTER\_LINEAR el más rápido de los dos. Si una

no se proporciona interpolación, OpenCV utiliza INTER\_LINEAR como predeterminado

Tanto para reducir como para agrandar.

La imagen de las tres bolas es actualmente de 800 × 533 píxeles. A pesar de esto

no es un tamaño grande, lo haremos un poco más pequeño. Vamos a hacer que sea la mitad de su actual

Tamaño para ambos ejes. Para ello, utilizaremos la interpolación INTER\_AREA.

1. Abra el IDE IDLE y cree un nuevo archivo.
2. Guarde el archivo como resize\_image.py.
3. Ingrese el siguiente código:

importar cv2

img = cv2.imread ('color\_balls\_small.jpg')

x, y = img.shape [: 2]

resImg = cv2.resize (img, (y / 2, x / 2), interpolación =

cv2.INTER\_AREA)

cv2.imshow ('imagen', img)

cv2.imshow ('redimensionado', resImg)

cv2.waitKey (0)

1. Guarde el archivo.
2. Abra una ventana de terminal.
3. Desplácese a la carpeta en la que guardó el archivo.
4. Ingrese a python resize\_image.py y presione Retorno.

Dos ventanas deberían haberse abierto. El primero tiene la imagen original. los

segunda muestra la imagen reducida. Pulse cualquier tecla para cerrar las ventanas.

**14.- Espacios de colores**

Uno de los elementos clave de trabajar con el color es el espacio de color, que

Describe cómo OpenCV expresa el color. Dentro de OpenCV, el color es

Representado por una serie de números. El espacio de color determina el

significado de esos números.

El espacio de color predeterminado para OpenCV es BGR. Esto significa que cada color es

descrito por tres enteros entre 0 y 255, que corresponden a la

Tres canales de color: azul, verde y rojo, en ese orden. Un color expresado

as (255,0,0) tiene un valor máximo en el canal azul, y ambos en verde

y los rojos son cero. Esto representa el azul puro. Dado esto, (0,255,0) es verde

y (0,0,255) es rojo. Los valores (0,0,0) representan negro, la ausencia de cualquier

color, y (255,255,255) es blanco.

Si ha trabajado con gráficos en el pasado, BGR es lo opuesto a lo que

es probable que estés acostumbrado a La mayoría de los gráficos digitales se describen en términos de RGB: Rojo, verde y azul. Por lo tanto, esto puede tomar un poco de tiempo para acostumbrarse.

El matiz representa el color en una escala de 0 a 180. La saturación representa cómo

blanco lejano, el color es de 0 a 255. El valor es una medida de cuán lejos de

el color negro es de 0 a 255. Si tanto la saturación como el valor son 0, el color

es gris Una saturación y un valor de 255 es la versión más brillante del tono.

es un poco más complicado. Está en una escala de 0 a 180, donde 0 y 180 son

ambos rojos Aquí es donde recordar la rueda de colores es importante. Si 0 y

180 se encuentran en la parte superior de la rueda en medio del espacio rojo, a medida que te mueves en el sentido de las agujas del reloj alrededor de la rueda, el tono = 30 es amarillo, el tono = 60 es verde, el tono = 90 es verde azulado, tono = 120 es azul, tono = 150 es púrpura y tono = 180 nos devuelve al rojo.

El que te encuentras con mayor frecuencia es la escala de grises. Escala de grises es

exactamente cómo suena: la versión en blanco y negro de una imagen. Es

utilizado por los algoritmos de detección de características para crear máscaras.

Filtro para objetos.

Para convertir una imagen a un espacio de color diferente, utiliza el cvtColor

método. Toma dos parámetros: la imagen y la constante de espacio de color.

Las constantes de espacio de color están integradas en OpenCV. Son de color

BGR2RGB, COLOR\_BGR2HSV y COLOR\_BGR2GRAY. Ves el

patrón allí? Si desea convertir del espacio de color RGB al

Espacio de color HSV, la constante sería COLOR\_RGB2HSV.

Convirtamos nuestra imagen de las tres bolas de colores en una imagen en escala de grises.

1. Abra el IDE IDLE y cree un nuevo archivo.
2. Guarde el archivo como gray\_image.py.
3. Ingrese el siguiente código:

import cv2

img = cv2.imread('color\_balls\_small.jpg')

grayImg = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

cv2.imshow('img', img)

cv2.imshow('gray', grayImg)

cv2.waitKey(0)

1. Guarda el archivo.
2. Abra una ventana de terminal.
3. Desplácese a la carpeta en la que guardó el archivo.
4. Ingrese a python gray\_image.py y presione Retorno.

Esto abre dos ventanas: una con la imagen en color original y una

Con la versión en escala de grises. Haga clic en cualquier tecla para salir del programa y cerrar las ventanas.

**15.- Filtros de colores**

El filtrado de un color requiere un código muy pequeño, pero al mismo tiempo,

puede ser un poco frustrante porque generalmente no está buscando un

Color, pero una gama de colores. Los colores son raramente puros y de un solo valor. Esto es por eso queremos poder cambiar entre espacios de color. Claro, podríamos mirar

Para un color rojo en BGR. Pero para hacer eso, necesitaríamos el rango específico para

cada uno de los tres valores. Y donde eso va a ser verdad con todo color

espacios, generalmente es más fácil marcar el rango que necesita en el espacio HSV.

La forma más fácil de demostrar esto es caminando a través de él. El código en realidad es bastante simple una vez que sabes lo que está pasando.

1. Abra el IDE IDLE y cree un nuevo archivo.
2. Guarde el archivo como blue\_filter.py.
3. Introduce el siguiente código:

import cv2

img = cv2.imread("color\_balls\_small.jpg")

imgHSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

Chapter 9 An Introduction to OpenCV

323

lower\_blue = np.array([80,120,120])

upper\_blue = np.array([130,255,255])

blueMask = cv2.inRange(imgHSV,lower\_blue,upper\_blue)

res = cv2.bitwise\_and(img, img, mask=blueMask)

cv2.imshow('img', img)

cv2.imshow('mask', blueMask)

cv2.imshow('blue', res)

cv2.waitKey(0)

1. Guarda el archivo.
2. Abra una ventana de terminal.
3. Desplácese a la carpeta en la que guardó el archivo.
4. Ingrese a python blue\_filter.py y presione Retorno.

**16.- Detección de manchas (Blobs)**

Un blob es una colección de píxeles similares. Podrían ser cualquier cosa desde un

Círculo monótono a una imagen jpeg. Para una computadora, un píxel es un píxel, y

no puede distinguir entre una imagen de una bola y una imagen de un plano.

Esto es lo que hace que la visión por ordenador sea tan desafiante. Hemos desarrollado

muchas técnicas diferentes para tratar de extrapolar información sobre una imagen; Cada uno tiene compensaciones en términos de velocidad y precisión.

La mayoría de las técnicas utilizan un proceso llamado extracción de características, que es un término general para una colección de algoritmos que catalogan pendientes

características de una imagen, como líneas, bordes, amplias áreas de color, etc.

adelante. Una vez extraídas estas características, se pueden analizar o comparar.

Con otras características para realizar determinaciones sobre la imagen. Así es como

Funciones como detección de rostro y trabajo de detección de movimiento.

Vamos a utilizar un método más simple para rastrear un objeto. Más bien luego de extraer las características detalladas y analizarlas, utilizaremos el color.

Técnicas de filtrado de la sección anterior para identificar un área grande de color. Luego usaremos funciones integradas para recopilar información sobre el grupo de píxeles. Esta técnica más simple se llama detección de manchas.

**17.- Seguimiento de manchas (Blob Tracking)**

Vimos en la sección anterior que las coordenadas x e y del centro de un blob se devuelven como parte de los puntos clave, que se utiliza para rastrear el blob.

Para rastrear el blob, necesita usar la transmisión de video en vivo desde la cámara del robot y, a continuación, defina qué significa el seguimiento para su proyecto. La forma más simple de seguimiento es simplemente mover el círculo generado con la mancha.

1. Abra el IDE IDLE y cree un nuevo archivo.
2. Guarde el archivo como blob\_tracker.py
3. Ingrese el siguiente código:

import cv2

import numpy as np

cap = cv2.VideoCapture(0)

# setup detector and parameters

params = cv2.SimpleBlobDetector\_Params()

params.filterByColor = False

params.filterByArea = True

params.minArea = 20000

params.maxArea = 30000

params.filterByInertia = False

params.filterByConvexity = False

params.filterByCircularity = True

params.minCircularity = 0.5

params.maxCircularity = 1

det = cv2.SimpleBlobDetector\_create(params)

# define blue

lower\_blue = np.array([80,60,20])

upper\_blue = np.array([130,255,255])

while True:

ret, frame = cap.read()

imgHSV = cv2.cvtColor(frame, cv2.COLOR\_BGR2HSV)

blueMask = cv2.inRange(imgHSV,lower\_blue,upper\_blue)

blur= cv2.blur(blueMask, (10,10))

res = cv2.bitwise\_and(frame, frame, mask=blueMask)

# get and draw keypoint

keypnts = det.detect(blur)

cv2.drawKeypoints(frame, keypnts, frame, (0,0,255),

cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_

KEYPOINTS)

cv2.imshow('frame', frame)

cv2.imshow('mask', blur)

for k in keypnts:

print k.size

if cv2.waitKey(1) & 0xff == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

1. Guarda el archivo.
2. Abra una ventana de terminal.
3. Desplácese a la carpeta en la que guardó el archivo.
4. Ingrese sudo python blob\_tracker.py y presione return.

Dos ventanas abiertas: una que muestra la máscara utilizada para filtrar el color y uno con el flujo de video. Se debe dibujar un círculo alrededor de la mancha.

He habilitado filterByArea y filterByCircularity para asegurarme de que solo estoy recibiendo la pelota. Es probable que necesite realizar ajustes en los parámetros del detector para afinar su filtro.