FICH, UNL - Departamento de Informática - Ingeniería Informática

Procesamiento Digital de Imágenes

Guía de Trabajos Prácticos 0

Introducción a Python y OpenCV

1. Instalación

Para la realización de las actividades prácticas esta cátedra sugiere el uso del lenguaje de programación **Python** (`https://www.python.org` o `http://www.python.org.ar`) y de la biblioteca de procesamiento de visión computacional **OpenCV** (`opencv.org/`). Le recomendamos que se mantenga actualizado (consultando en los sitios oficiales) respecto a las nuevas versiones ya que se presentan con avances significativos. Ambos han sido escogidos por su potencialidad y facilidad de uso, sin embargo, si usted prefiere utilizar otro lenguaje de programación y/o biblioteca de visión computacional puede hacerlo pero no podremos asistirlo respecto de las implementaciones.

1.1. Python

Instalar Python es trivial y todas las distribuciones de Linux incluyen una versión actualizada de éste. Sin embargo, si necesita descargarlo e instalarlo puede consultar aquí `https://www.python.org/downloads/`. Si no conoce el lenguaje de programación puede acceder a miles de tutoriales en diferentes sitios web, como por ejemplo en `https://wiki.python.org/moin/BeginnersGuide/Programmers`.

Usted puede trabajar con cualquier IDE de su preferencia (se recomienda *Spyder*) y puede utilizar la herramienta **pip** (`https://pypi.org/project/pip/`) para instalar las bibliotecas que serán necesarias como por ejemplo:

- pip install **numpy matplotlib**

Una alternativa es utilizar *Anaconda* (`www.anaconda.com`) que es una plataforma estándar para trabajar con Python, que por defecto tiene incluidas varias bibliotecas útiles, el IDE Spyder y Jupiter Notebook (`JupyterNotebookjupyter.org`) que es un entorno web para crear y compartir documentos con códigos, ecuaciones, visualizaciones y textos descriptivos. Esta última herramienta es muy útil para comentar códigos y analizar sus comportamientos, y para realizar la entrega de informes de trabajos prácticos. Para la instalación de bibliotecas se utilizará la herramienta *conda*.

1.2. OpenCV

Para instalar OpenCV, necesita utilizar la siguiente instrucción:

- pip install opencv-python

Si ustede utiliza la plataforma Anaconda, debe utilizar:

- conda install -c conda-forge opencv

2. Hello World

La prueba de funcionamiento la realizaremos cargando, obteniendo información y visualizando una imagen. Para ésto utilice el siguiente código:

```
# incluir la biblioteca
import cv2

# leer la imagen, mostrar sus dimensiones y tipo de dato
imagen = cv2.imread(‘‘OpenCV-Logo.png’’)
print("Dimensiones de la imagen: ", imagen.shape)
print("Tipo de dato de la imagen", img.dtype)

# mostrar la imagen en una ventana hasta que se presione una tecla y destruir ventana
cv2.imshow("Hello World", imagen)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

* Por defecto, OpenCV carga las imágenes en estructuras de tipo “uint8”, lo que debe tenerse en cuenta al operar con la imagen.

3. Creación, lectura y guardado

Para generar una imagen de zeros, se puede utilizar la instrucción np.zeros de la biblioteca *numpy* que debe incluirse. Si quiere crear una imagen a partir de otra, puede utilizar:

```
import cv2
imagen_nueva = np.zeros(imagen.shape, dtype = imagen.dtype)
ó
imagen_nueva = np.zeros_like(imagen)
```

Para cargar una imagen en escala de grises:

```
imagen_gray = cv2.imread(‘‘OpenCV-Logo.png’’, cv2.IMREAD_GRAYSCALE)
```

Para generar una imagen de grises a partir de una BGR:

```
imagen_gray = cv2.imread(‘‘OpenCV-Logo.png’’, cv2.IMREAD_BGR2GRAY)
```

- Nota: OpenCV utiliza el modelo BGR.

Para guardar una imagen (la función identifica el formato con base en la extensión del nombre):

```
cv2.imwrite(‘‘nueva_imagen.jpg’’, imagen)
```

4. Copiar, extraer ROIs y acceder al valor de un pixel

Para copiar imágenes es necesario utilizar la función *copy()*, ya que la instrucción = asigna la misma dirección de memoria a dos variables.

```
imagen2 = imagen.copy()
```

Si se quiere copiar sólo una ROI de la imagen, se puede utilizar:

```
imagen2 = imagen[x0:x1,y0:y1].copy()
```

Para leer y escribir en un pixel (x,y) se puede utilizar:

```
valor_px = imagen[y,x]
imagen[y,x] = valor_px
```

- Nota: valor_px será un “uint8” para imágenes de grises mientras que serán 3 valores para imágenes color.

5. Herramientas para dibujar y visualizar

Algunas de las funciones que nos permiten dibujar (línea, círculo y rectángulo) en una imagen son:

- `cv2.line(imagen, start_point, end_point, color, thickness, line_type)`
- `cv2.circle(imagen, center_point, radius, color, thickness, line_type)`
- `cv2.rectangle(imagen, start_point, end_point, color, thickness)`

5.1. Mostrar imágenes con matplotlib.pyplot

Esta biblioteca tiene las funcionalidades similares a las que brinda Matlab, se incluye con:

```
import matplotlib.pyplot as plt
# Las figuras se declaran como en Matlab
plt.figure()
# Para cargar una imagen en la figura
plt.imshow(imagen)
# Se pueden asignar falsos colores a imágenes de grises
plt.imshow(imagen_gray, cmap= ’gray’)
```

Se pueden utilizar subfiguras con *plt.subplot(num_filas,num_columnas,actual):*

```
plt.subplot(1,3,1)
plt.imshow(imagen)
plt.subplot(1,3,2)
plt.imshow(imagen_gray, cmap = 'gray')
plt.subplot(1,3,3)
plt.imshow(imagen2)
plt.show()
```

6. Avanzados

6.1. Eventos del mouse

Este es un ejemplo acerca de la utilización del eventos del mouse para dibujar una línea, en este caso se utiliza el botón izquierdo al presionar y al soltar.

```
# Primero se define la función:
def click(event, x, y, flags, param):
     # si se presiona el botón izquierdo, se guarda la localización (x,y)
     if event == cv2.EVENT_LBUTTONDOWN:
          refPt = [(x, y)]
     # si se libera el botón, se guarda la localización (x,y)
     elif event == cv2.EVENT_LBUTTONUP:
          refPt.append((x, y))
          # dibuja la linea entre los puntos
          cv2.line(imagen, refPt[0], refPt[1], (0, 255, 0), 2)
          cv2.imshow(str_win, imagen)

# defino una ventana y le asigno el manejador de eventos
str_win=``DibujaWin''
cv2.namedWindow(str_win)
cv2.setMouseCallback(str_win, click)

while True:
     # muestra la imagen y espera una tecla
     cv2.imshow(str_win, imagen)
     key = cv2.waitKey(1) & 0xFF
     # si la tecla c es presionada sale del while
     if key == ord(``c''):
          break
```

6.2. Interacción con trackbars

A veces es útil modificar parámetros mientras se visualizan las imágenes, se puede hacer mediante consola, utilizando el teclado o utilizando herramientas gráficas como trackbars en la ventana. En el ejemplo se combinan dos imágenes (blending) variando los pesos de cada una, cada vez que se mueve la barra se procesa.

```
# como usa valores naturales, ajusto la cantidad de valores a elegir
alpha_slider_max = 100

# se utilizan las 2 imágenes y se calculan los parámetros para combinarlas
def on_trackbar_alpha(val):
     global imagen1, imagen2
     alpha = val / alpha_slider_max
     beta = ( 1.0 - alpha )
     dst = cv2.addWeighted(imagen1, alpha, imagen1, beta, 0.0)
     cv2.imshow(title_window, dst)

cv2.namedWindow(title_window)
cv2.createTrackbar('Alpha', title_window, int(alpha_slider_max/2), alpha_slider_max,
on_trackbar_alpha)

while True:
     cv2.imshow(title_window, dst)
     key = cv2.waitKey(1) & 0xFF
     # presione c para salir
     if key == ord(''c''):
          break
```

6.3. Pasaje de parámetros al programa

Una biblioteca que permite el manejo del pasaje de parámetros al programa es *argparse* y se debe incluir como `import argparse`

```
# se crea el analizador de parámetros y se especifican
ap = argparse.ArgumentParser()
ap.add_argument(''-ig'', ''--imagen_gris'', required=True, help=''path de la
imagen de grises'')
ap.add_argument(''-ic'', ''--imagen_color'', required=True, help=''path de
la imagen color'')
args = vars(ap.parse_args())

# se recuperan los parámetros en variables o directamente se utilizan
nombre_imagen = args[''imagen_gris'']
imagen_2 = cv2.imread(args[''imagen_color''])
```