Práctica 2:

*Looking for something.*

José Estévez Fernández.

30/10/2018

Contenido

[Introducción. 2](#_Toc528785089)

[Detectar Objetos. 2](#_Toc528785090)

[Capturar imagen. 3](#_Toc528785091)

[Convertir la imagen a escala de grises. 3](#_Toc528785092)

[Filtrado del ruido en una imagen. 4](#_Toc528785093)

[Filtro Guassiano para la eliminación de ruido en imágenes. 5](#_Toc528785094)

[Detector de bordes. 6](#_Toc528785095)

[Detección de bordes con Sobel. 6](#_Toc528785096)

[Filtrado de bordes mediante la supresión non-maximun. 6](#_Toc528785097)

[Aplicar umbral por histéresis 6](#_Toc528785098)

[Detector de bordes Canny con OpenCV. 7](#_Toc528785099)

[Buscar los contornos de una imagen. 7](#_Toc528785100)

[Dibujar Rectángulos/Cuadrados. 8](#_Toc528785101)

# Introducción.

En base a una serie de fotogramas de tres contadores digitales, se propone una aplicación en ***C++*** y ***OpenCV*** bajo ***Qt*** capaz de establecer las lecturas de dichos contadores utilizando alguna(s) tecnica(s) de **matching template**.

Las entradas serán un conjunto de ficheros de imagen en un directorio y la salida un único fichero de texto que contenga línea a línea las lecturas de los contadores y el nombre del fichero de imagen.

# Matching Template.

***Matching Template*** es una técnica para encontrar las áreas en que una imagen contiene una imagen menor (plantilla). Para este propósito ***OpenCV*** proporciona la herramienta ***cv2.matchTemplate()***

## ¿Cómo funciona?

La técnica de ***Matching Templat*e** sigue los siguientes pasos:

* La imagen menor (plantilla) se desliza sobre la imagen de entrada.
* Se comparan la plantilla y el parche de la imagen de entrada debajo de la imagen de la plantilla.
* El resultado obtenido se compara con el umbral.
* Si el resultado es mayor que el umbral, la parte se marcará como detectada.

## Limitaciones.

A continuación vamos a explicar las limitaciones que presenta ésta técnica:

* Las incidencias de patrón deben conservar la orientación de la imagen del patrón de referencia (plantilla)
* Como resultado, no funciona para versiones rotadas o escaladas de la plantilla como un cambio en la forma / tamaño / corte, etc. del objeto w.r.t. La plantilla dará una falsa coincidencia.
* El método es ineficiente al calcular la imagen de correlación de patrón para imágenes medianas a grandes, ya que el proceso consume mucho tiempo.
* Para evitar el problema causado por los diferentes tamaños de la plantilla y la imagen original, podemos utilizar el ***multiscaling***. En caso de que las dimensiones de la plantilla no coincidan con las dimensiones de la región en la imagen que desea hacer coincidir, no significa que no pueda aplicar la coincidencia de plantillas.

# cv2.matchTemplate()

Para realizar la técnica Matching Template tenemos el método matchTemplate de OpenCV.

***matchTemplate(src\_img, template\_img[i], result\_mat, match\_method)***

MatchTemplate tiene los siguientes parámetros:

* **src\_img:** Imagen donde buscar la plantilla. Debe de ser de 8 bits o 32 bits en coma flotante.
* **template\_img[i]:** Plantilla a buscar, debe ser menor que ***src\_img*** y tener el mismo tipo.
* **result\_mat:** Mapa de comparación de resultados.
* **match\_method:** Método de comparación a aplicar.

# Práctica.

Para desarrollar la práctica debemos seguir los siguientes pasos:

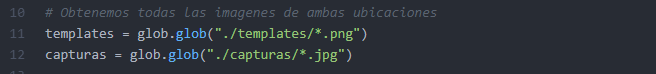
* Cargar las imágenes.
* Segmentar las tres zonas de los contadores.
* Procesar los contadores:
  + Segmentar los diferentes números que lo forman.
  + Procesar los números:
    - Hacer matching con todos los templates.
    - Quedarse con el más probable y encadenarlo
* Escribir una línea en el fichero de texto con el resultado.

## Cargar las imágenes.

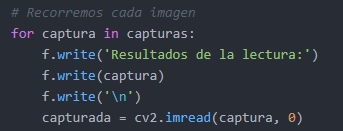
El primer paso es cargar las imágenes a procesar, para ello:

* Cargamos todas las imágenes de la carpeta ***templates*** en un vector.
* Cargamos todas las imágenes de la carpeta ***capturas*** en un vector.

Para realizar los pasos anteriores hacemos uso de la librería ***glob*** mediante las siguientes sentencias:



Abrimos una a una las imágenes del vector ***capturas*** para segmentar las tres zonas de los contadores.

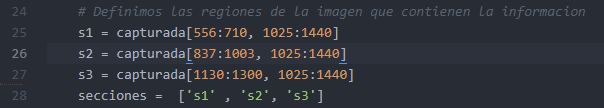


Al indicar 0 como parámetro en el método ***cv2.imread()*** abrimos la imagen en escala de grises. De esta forma ahorramos el tener que transformar una imagen en color a escala de grises.

Guarda en el fichero la imagen que se va a procesar.

## Segmentar las tres zonas de los contadores.

Al ser imágenes tomadas con una cámara fija podemos segmentar las zonas de cada uno de los contadores para obtener sus lecturas. Para ello definimos las coordenadas de cada una de las secciones y las guardamos en un array para posteriormente procesar cada una de las zonas.

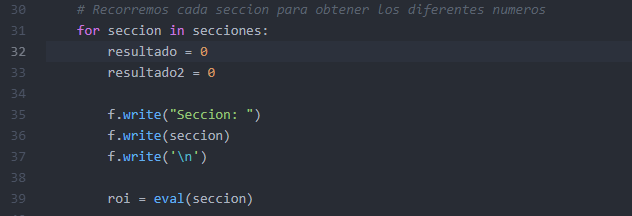


## Procesar los contadores.

Para procesar los contadores por cada una de las secciones segmentadas en el apartado anterior debemos seguir los siguientes pasos:

* Segmentar los diferentes números.
* Procesar cada uno de los números.

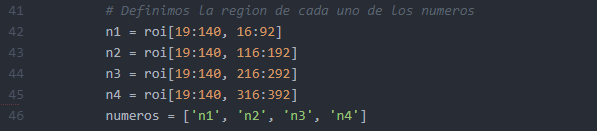
Para procesar cada una de las secciones recurrimos a un bucle ***for***



Escribimos una línea en el fichero indicando la sección que se va a procesar.

### Segmentar los diferentes números.

Al igual que hicimos con las secciones, debemos segmentar cada uno de los números de cada sección utilizando sus coordenadas.

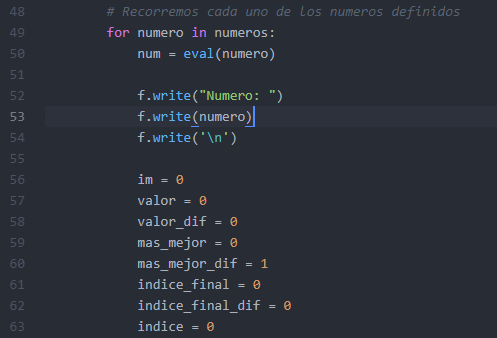


### Procesar cada uno de los números.

Para cada uno de los números debemos seguir los siguientes pasos para determinar a qué número se corresponde.

* Hacer ***matching*** con todos los ***templates***.
* Quedarse con el más probable y encadenarlo.

Para procesar cada número recurrimos a un bucle ***for*** y a una serie de ***variables*** para determinar la mejor opción.



#### Hacer ***matching*** con todos los ***templates***.

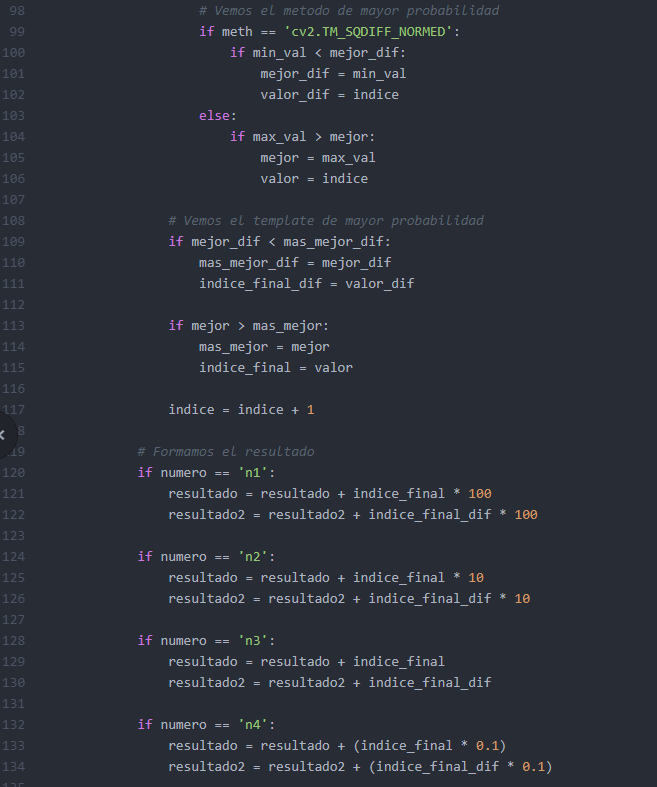
Para hacer ***matching*** utilizamos el método ***matchTemplate***. Se compara cada número segmentado con los diferentes ***templates*** para determinar qué número tiene más probabilidad de ser.



Tal y como puede verse en la imagen superior utilizamos todos los métodos del ***mathcTemplate*** para buscar la mejor opción.

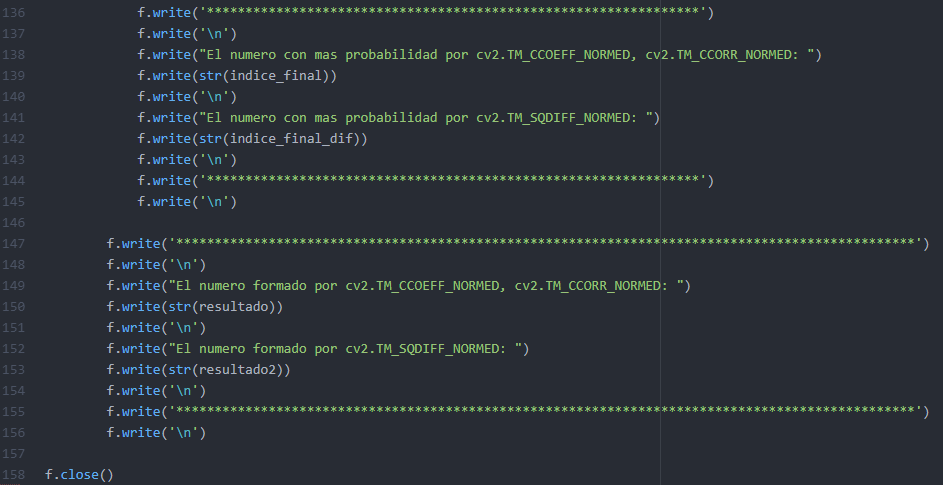
#### Quedarse con el más probable y encadenarlo.

Para ello utilizamos las variables definidas en el apartado anterior. Primero buscamos el método con mayor probabilidad y después comprobamos qué número tiene una mayor probabilidad.



## Escribir una línea en el fichero de texto con el resultado.

Por último escribimos una línea con el mejor resultado de cada ejecución en el fichero de texto y además el resultado final obtenido.



# Repositorio.

# WebGrafia.

<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_template_matching/py_template_matching.html>

<https://docs.opencv.org/3.4.1/de/da9/tutorial_template_matching.html>

<https://pythonprogramming.net/template-matching-python-opencv-tutorial/>

<https://www.youtube.com/watch?v=2CZltXv-Gpk>

<https://www.geeksforgeeks.org/template-matching-using-opencv-in-python/>

<https://docs.opencv.org/3.4.1/df/dfb/group__imgproc__object.html#ga586ebfb0a7fb604b35a23d85391329be>