Práctica 2:

Looking for something.

José Estévez Fernández.

30/10/2018

Contenido

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:

```
10  # Obtenemos todas las imagenes de ambas ubicaciones
11  templates = glob.glob("./templates/*.png")
12  capturas = glob.glob("./capturas/*.jpg")
```

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

```
# Recorremos cada imagen
for captura in capturas:
    f.write('Resultados de la lectura:')
    f.write(captura)
    f.write('\n')
    capturada = cv2.imread(captura, 0)
```

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.

```
24      # Definimos las regiones de la imagen que contienen la informacion
25      s1 = capturada[556:710, 1025:1440]
26      s2 = capturada[837:1003, 1025:1440]
27      s3 = capturada[1130:1300, 1025:1440]
28      secciones =  ['s1' , 's2', 's3']
```

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*

```
30      # Recorremos cada seccion para obtener los diferentes numeros
31      for seccion in secciones:
32          resultado = 0
33          resultado2 = 0
34
35          f.write("Seccion: ")
36          f.write(seccion)
37          f.write('\n')
38
39          roi = eval(seccion)
```

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.

```
41          # Definimos la region de cada uno de los numeros
42          n1 = roi[19:140, 16:92]
43          n2 = roi[19:140, 116:192]
44          n3 = roi[19:140, 216:292]
45          n4 = roi[19:140, 316:392]
46          numeros = ['n1', 'n2', 'n3', 'n4']
```

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.

```
48          # Recorremos cada uno de los numeros definidos
49          for numero in numeros:
50              num = eval(numero)
51
52              f.write("Numero: ")
53              f.write(numero)
54              f.write('\n')
55
56              im = 0
57              valor = 0
58              valor_dif = 0
59              mas_mejor = 0
60              mas_mejor_dif = 1
61              indice_final = 0
62              indice_final_dif = 0
63              indice = 0
```

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.

```
65              # Compara un numero con cada una de las plantillas
66              for templateImage in templates:
67                  template = cv2.imread(templateImage, 0)
68                  mejor = 0
69                  im = im + 1
70
71                  mejor_dif = 1
72
73                  f.write("Probabilidad de que sea la plantilla: ")
74                  f.write('\t')
75                  f.write(templateImage)
76                  f.write('\n')
77                  f.write("Metodo")
78                  f.write('\t\t\t\t')
79                  f.write('Min_Val')
80                  f.write('\t\t\t\t')
81                  f.write('Max_Val')
82                  f.write('\n')
83
84                  # Probamos cada metodo
85                  for meth in methods:
86                      method = eval(meth)
87
88                      res = cv2.matchTemplate(num, template,method)
89                      min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
90
91                      f.write(meth)
92                      f.write('\t')
93                      f.write(str(min_val))
94                      f.write('\t')
95                      f.write(str(max_val))
96                      f.write('\n')
```

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.

```
                    # Vemos el metodo de mayor probabilidad
                    if meth == 'cv2.TM_SQDIFF_NORMED':
                        if min_val < mejor_dif:
                            mejor_dif = min_val
                            valor_dif = indice
                    else:
                        if max_val > mejor:
                            mejor = max_val
                            valor = indice

                # Vemos el template de mayor probabilidad
                if mejor_dif < mas_mejor_dif:
                    mas_mejor_dif = mejor_dif
                    indice_final_dif = valor_dif

                if mejor > mas_mejor:
                    mas_mejor = mejor
                    indice_final = valor

                indice = indice + 1

            # Formamos el resultado
            if numero == 'n1':
                resultado = resultado + indice_final * 100
                resultado2 = resultado2 + indice_final_dif * 100

            if numero == 'n2':
                resultado = resultado + indice_final * 10
                resultado2 = resultado2 + indice_final_dif * 10

            if numero == 'n3':
                resultado = resultado + indice_final
                resultado2 = resultado2 + indice_final_dif

            if numero == 'n4':
                resultado = resultado + (indice_final * 0.1)
                resultado2 = resultado2 + (indice_final_dif * 0.1)
```

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.

```
            f.write('************************************************************')
            f.write('\n')
            f.write("El numero con mas probabilidad por cv2.TM_CCOEFF_NORMED, cv2.TM_CCORR_NORMED: ")
            f.write(str(indice_final))
            f.write('\n')
            f.write("El numero con mas probabilidad por cv2.TM_SQDIFF_NORMED: ")
            f.write(str(indice_final_dif))
            f.write('\n')
            f.write('************************************************************')
            f.write('\n')

        f.write('*****************************************************************************************')
        f.write('\n')
        f.write("El numero formado por cv2.TM_CCOEFF_NORMED, cv2.TM_CCORR_NORMED: ")
        f.write(str(resultado))
        f.write('\n')
        f.write("El numero formado por cv2.TM_SQDIFF_NORMED: ")
        f.write(str(resultado2))
        f.write('\n')
        f.write('*****************************************************************************************')
        f.write('\n')

f.close()
```

REPOSITORIO.

Se puede ver el desarrollo del proyecto en el siguiente repositorio.

https://github.com/OpenCVProjects/DigitalCounterReader.git

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