Progetto Tecnologie Multimediali

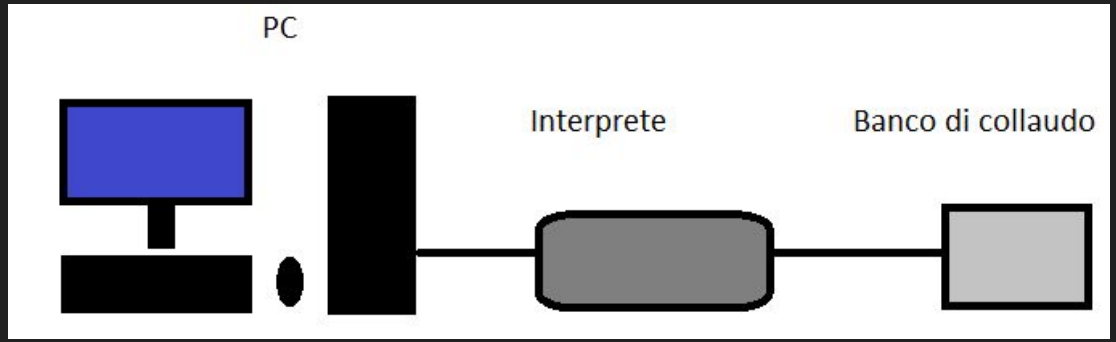
2022/06/15

OpenCV e Python per automatizzare il collaudo di un display

Collaudo

Il collaudo del display viene effettuato insieme al dispositivo su cui è montato, in questo caso un termostato.

Il pcb del termostato viene inserito nel suo banco di collaudo che è collegato ad un interprete che lo comanda traducendo lo script di collaudo dal computer a cui è collegato.



Problema

Il collaudo delle funzionalità del termostato è eseguito in completa automazione fatta eccezione per la verifica dei pulsanti e la verifica del display.

L’idea è quella di rendere automatica anche la verifica del display per evitare due problemi correlati:

- dopo ore di collaudo del display l’operatore è vittima di stanchezza degli occhi e, a seconda di come è posizionato il display nel banco di collaudo, del collo e della schiena
- per i motivi precedenti, o anche per motivi esterni, l’operatore può essere distratto e validare un display che non funziona o scartare un display funzionante

Come fare?

Per automatizzare il collaudo del display è necessario catturarne un'immagine e confrontarla con le immagini delle icone che dovrebbero essere accese in quel momento.

È quindi necessario identificare le immagini delle icone accese per ogni stato del display da verificare.

Camera

È necessario aggiungere una camera al banco di collaudo da cui poter catturare le immagini del display.

Diventa quindi necessario capire come installarla in modo da non ostruire il resto del collaudo e sviluppare qualcosa che comunichi con lo script di collaudo per validare o scartare il display.

Passiamo al secondo problema.

OpenCV

È una libreria open-source che mette a disposizione centinaia di algoritmi per la computer vision che possono essere integrati negli script di linguaggi quali C++, Java e Python.

https://opencv.org

Acquisizione

Acquisizione

Usando la funzione “VideoCapture”:

```
cap = cv.VideoCapture(0, cv.CAP_DSHOW)
ret, image = cap.read()
if image is None:
    print('immagine non caricata')
```

Usando la funzione “imread”:

```
image = cv2.imread('images/image.jpg')
if image is None:
    print("Immagine non caricata")
```

Acquisizione

In entrambi i casi l'immagine caricata sarà in BGR e dovrà essere trasformata in scala di grigi per essere processata dalle funzioni di OpenCV.

Verrà anche ridimensionata per concentrare l'informazione smussata con un filtro gaussiano.

```
    gray = cv2.cvtColor(resized_img,cv2.COLOR_BGR2GRAY)
    resized_img = cv2.resize(cropped_img, dim ,
interpolation=cv2.INTER_AREA)
    blurred = cv2.GaussianBlur(gray, (5,5), 0)
```

Individuazione ed estrazione icone

Estrazione dei bordi

Per estrarre i contorni dall'immagine OpenCV fornisce la funzione "Canny".

```
edged = cv2.Canny(blurred, 50, 150, 255)
```

L'algoritmo di Canny(https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html):

1 - Applica un filtro gaussiano 5x5 per rimuovere il rumore

2 - Trova l'intensità del gradiente dell'immagine(Sobel)

3 - Elimina i pixel che potrebbero non costituire un bordo controllando il massimo locale nei vicini sulla direzione del gradiente

4 - Fa una sogliatura d'isteresi per eliminare tutti i punti che non sono contorni

Contorni

Per trovare i contorni delle icone accese OpenCV mette a disposizione la funzione “findContours”

```
contours, hierarchy = cv2.findContours(draw_cnts, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
```

Questa funzione sfrutta l’algoritmo di Suzuki per identificare i contorni e differenziare tra contorni esterni ed interni.

- `cv2.RETR_EXTERNAL`
  identifica solo i contorni esterni
- `cv2.CHAIN_APPROX_NONE`
  memorizza tutti i punti di contorno

ROI

Ottenuti i valori dei contorni posso ricavare, con la funzione “boundingRect”, un rettangolo per ciascuna catena che contenga l’elemento descritto.

```
(x,y,w,h) = cv2.boundingRect(c)
```

Risulta utile ottenere le coordinate e le dimensioni del rettangolo per

- isolare solo gli elementi di interesse
- definire una regione di interesse per ogni icona da estrarre

Icone

Posso salvare gli elementi di interesse utilizzando la funzione “imwrite”

```
cv2.imwrite(i_p, roi)
```

Le coordinate e le dimensioni degli elementi verranno memorizzate in un file .csv insieme ad un identificatore per l’elemento trovato

Matching

Matching

“matchTemplate” è la funzione che permette di cercare un template all’interno di una immagine

```
res = cv2.matchTemplate(final.copy(), icona, cv2.TM_CCOEFF_NORMED)
```

Codice creazione Template

Acquisizione e pre-processing

```
image = cv2.imread('images/pcb.jpg')
if image is None:
    print("Immagine non caricata")
cropped_img = image[100:700, 55:1225]
h, w = cropped_img.shape[:2]
width = 500
r = width/w
dim = (width, int(r*h))
resized_img = cv2.resize(cropped_img, dim , interpolation=cv2.INTER_AREA)
gray = cv2.cvtColor(resized_img,cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray, (5,5), 0)
```

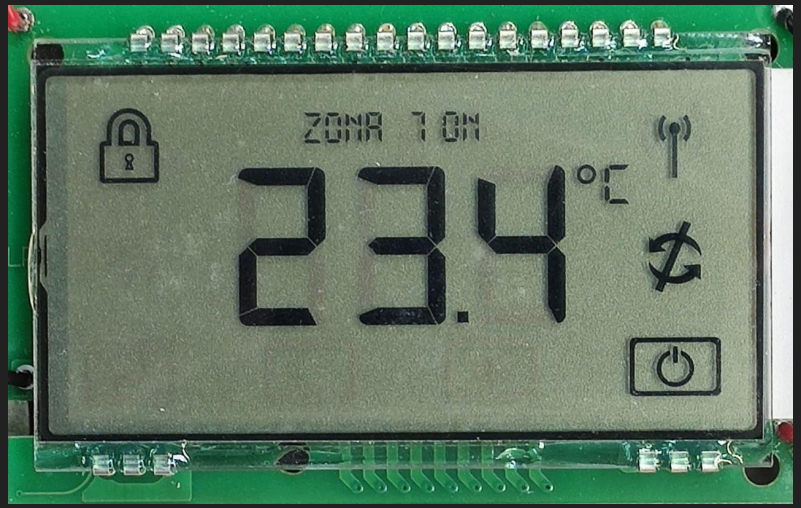


Immagine acquisita

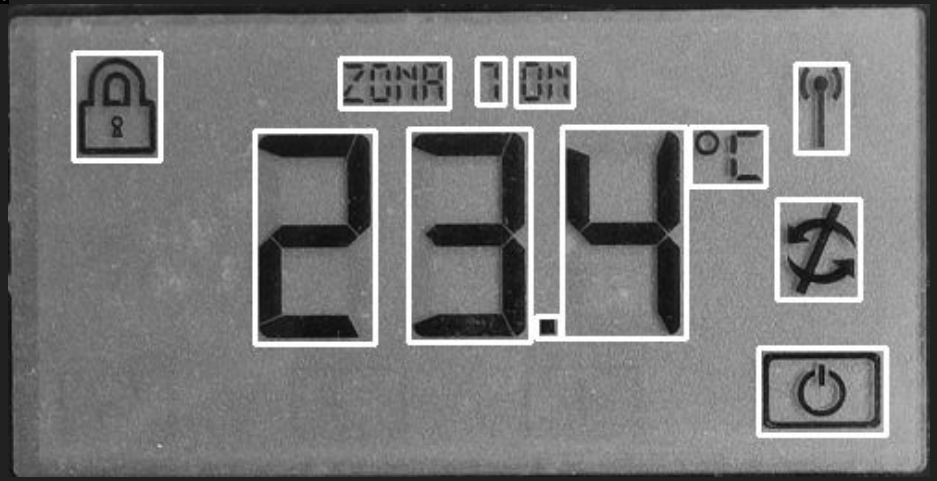
Immagine dopo preprocessing

Estrazione descrittori e definizione delle ROI

```
edged = cv2.Canny(blurred, 50, 150, 255)
kernel = np.ones((5,5), np.uint8)
dilation = cv2.dilate(edged.copy(), kernel, iterations=1)
draw_cnts = dilation.copy()
contours, hierarchy = cv2.findContours(draw_cnts, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
img_with_icons = gray.copy()
icons = []
icons_bounds = []
for c in contours:
    (x,y,w,h) = cv2.boundingRect(c)
    if (5 < x < 495) and (5 < y < 245) :
        if (w > 8) and (h > 7):
            if h < 200:
                cv2.rectangle(img_with_icons,(x,y),(x+w,y+h),(255,0,0),2)
                bound = (x,y,w,h)
                icons.append(c)
                icons_bounds.append(bound)
```

Immagine dopo il filtro Canny

Immagine con Roi



Salvataggio Templates

```
root = os.path.dirname(os.path.abspath(sys.argv[0]))
p = pathlib.Path(root)
p = p.parent.parent.parent
p_icons = os.path.join(p,'images','icons')
p_icons.replace('\\', '/')
print(p)
print(p_icons)
cont = 0
icons_legend = os.path.join(p_icons + '\\icons_legend.csv')
print(icons_legend)
if os.path.exists(icons_legend):
    os.remove(icons_legend)
with open(icons_legend, 'w', newline='') as i_l:
    csv_writer = csv.writer(i_l)
    header = ['id', 'x', 'y', 'w', 'h']
    csv_writer = csv.DictWriter(i_l, fieldnames=header)
    csv_writer.writeheader()
    for ib in icons_bounds:
        name = "icona_"+str(cont).zfill(3)+".jpg"
        roi = resized_img.copy()[ib[1]:ib[1]+ib[3], ib[0]:ib[0]+ib[2]]
        i_p = os.path.join(p_icons + '/' + name)
        cv2.imwrite(i_p, roi)
        row = {'id' : name, 'x' : str(ib[0]), 'y' : str(ib[1]), 'w' : str(ib[2]), 'h' : str(ib[3])}
        csv_writer.writerow(row)
        cont += 1
```

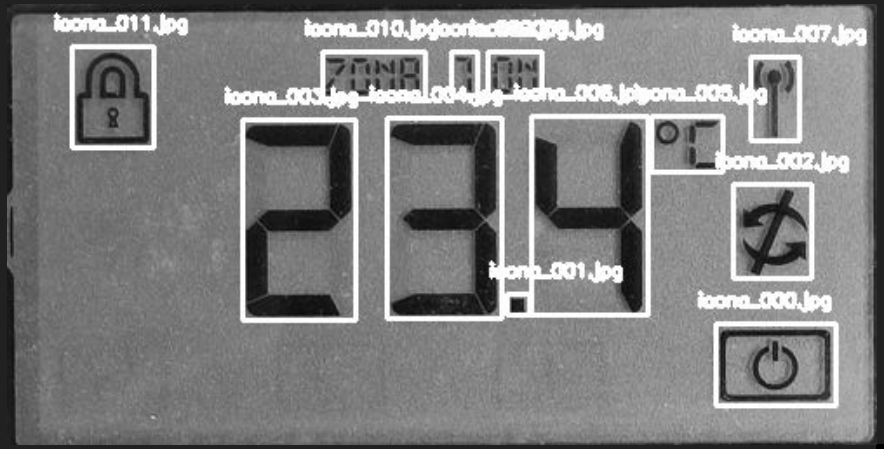
Codifica Matching Template

Preparazione variabili

```
csv_reader = pandas.read_csv(icons_legend)
main_dir = p
res_path = os.path.join(main_dir,'results.txt')
print(res_path)
final = gray.copy()
s_res= []
cnt = 0
if os.path.exists(res_path):
    os.remove(res_path)
```

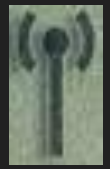
Algoritmo con funzione di Matching

```
for images in os.listdir(p_icons):
    if(images.endswith(".jpg")):
        i_p = os.path.join(p_icons + '\\' + images)
        icona = cv2.imread(i_p)
        if images == csv_reader['id'] [cnt]:
            x,y,w,h = (csv_reader['x'][cnt],csv_reader['y'][cnt],csv_reader['w'][cnt],csv_reader['h'][cnt])
            res = cv2.matchTemplate(final.copy(), icona, cv2.TM_CCOEFF_NORMED)
            cv2.rectangle(res, (x,y), (x+w, y+h), 255, 2)
            s_res.append(res)
            min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
            top_left = max_loc
            bottom_right = (top_left[0]+w, top_left[1]+h)
            if (x-2 < top_left[0] < x+2 ) and (y-2 < top_left[1] < y+2 ):
                found = open(res_path, "a")
                found.write("trovato " + images + '\r\n')
                found.close()
                cv2.rectangle(final, top_left, bottom_right, 255, 2)
                cv2.putText(final, images, (top_left[0]-10, top_left[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.35, 255, 2)
    if cnt < 12:
        cnt += 1
```

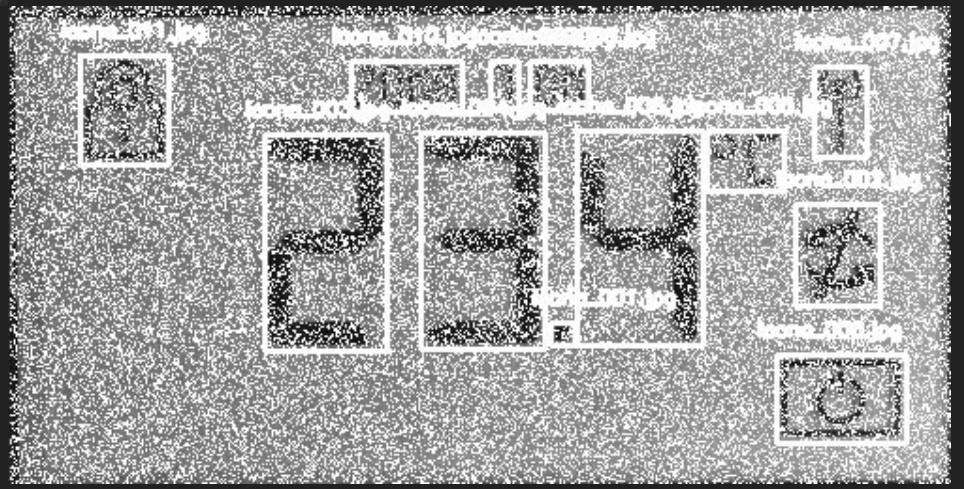


Icone trovate ed evidenziate sull’immagine

Output funzione “matchTemplate” per l’icona 007



Match su display parzialmente ostruito



Match su rumore sale e pepe

Conclusioni

Sul problema iniziale

- OpenCV e Python dimostrano la possibilità di automatizzare la verifica dell’operatività del display
- Il sistema è stato provato anche in condizione di rumore e la risposta è robusta

Riflessioni

- sarà probabilmente necessario migrare il sistema su C++ o Java per interfacciarlo al meglio con lo script di collaudo
- i test sono stati fatti su immagine acquisita da file e non catturata da una camera: cosa comporta?(problemi di luce, di acquisizione, di risoluzione) È forse possibile sfruttare un altro approccio?
- il sistema utilizza delle immagini template per trovare le icone accese, posso effettuare questa ricerca con successo usando invece i contorni?
- il sistema sembra scalabile, ma può essere applicato su display diversi? È necessario chiamare in causa il machine learning?