Separación de haces de luz

Diego Zapata Hernández
Juan Diego González Ramos
Daniel Hagen González
Antonio Fernández Rubio
Sergio Guillén González

Introducción
¿Qué resolvemos?

- Queremos saber si alguno de los heliostatos se sale de su coordenada establecida en una pantalla receptora de los haces de luz.

```
[843354.5, 1625624.75, 770447.6875, 1113306.875]
Hay movimiento, creemos que es el heliostato  2
```

Introducción
¿Qué es OpenCV?

- **OpenCV** es una biblioteca open source para procesamiento de imágenes y visión computarizada.
- Disponible en Linux, Mac, y Windows.
- Tiene estructuras básicas de datos para operaciones con matrices y procesamiento de imágenes.
- En python usamos “NumPy” una bliblioteca de funciones matemáticas de alto nivel para las operaciones con vectores o matrices.

Índice:

Leyenda

- A lo largo del código del programa utilizaremos varios tipos de variables que comienzan por:
  - VC_: Estas son las variables de control
  - VU_: Estas son las variables que son utilizadas por el usuario.
  - C_: Estas variables son contadores
  - E_: Estas variables son estáticas (solo se modifican 1 vez durante la ejecución del programa
  - L_: Estas variables son listas

Índice:

Introducción de datos por parámetro

```
17 #INSTRUCCIONES PARA MODIFICAR EL COMANDO QUE LANZA LA APLICACION
18 #_____ prog -> para cambiar el nombre del comando que lanza este programa.
19 #_____description -> info sobre lo que hace el programa
20 parser = argparse.ArgumentParser(prog="calibration",description="Process for identifying heliostats.")
21
22 parser.add_argument("--numHeliostats",default="4",type=int,help="set the number of heliostats to calibrate
23 parser.add_argument("--pyrLevels",default="0",type=int, #nargs="?",
24                     help="set how many times the Pyramid method will be applied to source")
25 args = parser.parse_args()
26
27 # Variables obtenidas por el usuario
28 VU_cantHel = args.numHeliostats
29 VU_cantPyrD = args.pyrLevels
```

Índice:

Declaración de variables

```
31  # Cargamos el vídeo
32  E_camara = cv2.VideoCapture("Videos/varios_heliostatos.mp4")
33  E_fondo = None # Inicializamos el primer frame a vacío. Nos servirá para obtener el fondo
34
35  # Listas
36  L_listaHel = []
37  # Contadores
38  C_contHel = 0
39  C_iteraciones = 0
40  C_contAñadir = 0
41
42  # Variables de control
43  VC_comenzarRegistroMovimiento = False # Comenzamos de nuevo a detectar mas manchas cuando hay mas de 1 proyeccion
44  VC_comenzarEstablecerFondo = False
45  VC_add1Mancha = False
46  VC_add2Mancha = False
47  VC_noEntrar1Mancha = False
48  VC_comprPAP = False
```

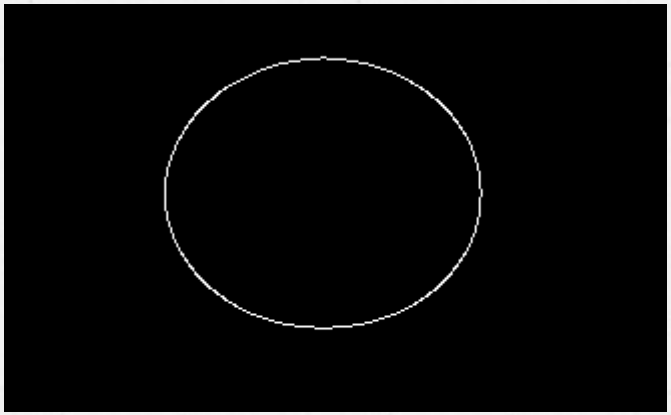
Índice:

Métodos principales

```
57  while True:
58      # Obtenemos el frame
59      (grabbed, frame) = E_camara.read()
60
61      # Si hemos llegado al final del vídeo salimos
62      if not grabbed:
63          break
64
65      # Bucle para reducir la imagen tantas veces como queramos
66      for x in range(0,VU_cantPyrD):
67          frame = cv2.pyrDown(frame)
68
69      # Convertimos a escala de grises
70      gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
71
72      # Aplicamos suavizado para eliminar ruido
73      gris = cv2.GaussianBlur(gris, (21, 21), 0)
74
75      # Si todavía no hemos obtenido el fondo, lo obtenemos. Será el primer frame que obtengamos
76      if E_fondo is None:
77          E_fondo = gris
78          continue
79
80      # Aplicamos un umbral
81      umbral = met.umbralizar(gris)
82
83      # Dilatamos el umbral para tapar agujeros
84      umbral = cv2.dilate(umbral, None, iterations = 2)
```

Métodos principales

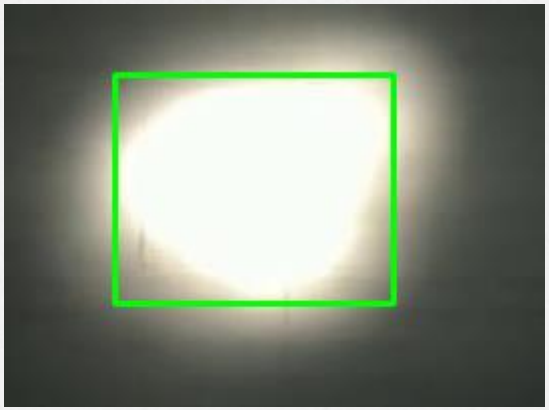
```
94      # Calculo de la diferencia entre el fondo y el frame actual
95      resta = cv2.absdiff(fondo, gris)
96      umbral = met.umbralizar(resta)
97
98      # Copiamos el umbral para detectar los contornos
99      contornosimg = umbral.copy()
100
101     # Buscamos contorno en la imagen. La variable contornosimg cambia! im = contornosimg
102     im, contornos, hierarchy = cv2.findContours(contornosimg,cv2.RETR_LIST,cv2.CHAIN_APPROX_NONE)
103
104     edges = cv2.Canny(umbral,255,255)
```



Índice:

Dibujo del contorno

```
102     # Recorremos todos los contornos encontrados
103     for c in contornos:
104         # Eliminamos los contornos más pequeño
105         contorno = cv2.contourArea(c)
106 
107         # Obtenemos el bounds del contorno, el rectángulo mayor que engloba al contorno
108         (x1, y1, w, h) = cv2.boundingRect(c) # w: es wigth y h: es heigh
109 
110         # Dibujamos el rectángulo del bounds
111         cv2.rectangle(frame, (x1, y1), (x1 + w, y1 + h), (0, 255, 0), 2)
```



Índice:

Detección de contornos

```
# Esta condicion se usa para ir añadiendo manchas.
elif comenzarRegistroMovimiento == False:
    # La logica es la siguiente: en el momento en que haya un contorno y aparezca otro,
    # añado el primer contorno a la lista.
    if cantHel == contHel:
        print("Comenzamos a comparar")
        comenzarEstablecerFondo = True
    elif len(contornos)>1:
        if contAñadir == 10:
            helCompr = []
            for t in contornos:
                (x1, y1, w, h) = cv2.boundingRect(t)
                #calculamos el punto (x2,y2):
                y2 = y1 + h
                x2 = x1 + w
                manchaNu = edges[y1:y2, x1:x2]
                helCompr.append(manchaNu)
        else:
            contAñadir+=1
        # Condicion para que no entre mas en la condicion de abajo, sino en la siguiente
        if noEntrar1Mancha == True:
            add1Mancha = True
        else:
            add2Mancha = True
    # Una vez que ya se han juntado las 2 primeras manchas en 1, las añadimos a listaHel
    elif len(contornos)==1 and add2Mancha == True:
        listaHel.append(helCompr[1]) # En la posicion 1 esta la mancha que ya estaba antes
        listaHel.append(helCompr[0]) # En la posicion 0 esta la mancha nueva
        add2Mancha = False
        noEntrar1Mancha = True
        contHel += 2
    # Este condicional solo añade una mancha
    elif len(contornos)==1 and add1Mancha == True:
        listaHel.append(helCompr[0])
        add1Mancha = False
        contHel += 1
```

Índice:

Detección de fondo

```
132         # Una vez hemos añadido las manchas establecemos el fondo. Damos 150 frame
133         elif VC_comenzarEstablecerFondo == True:
134             if VU_cantHel == C_contHel:
135                 met.visualizarLista(L_listaHel)
136                 if C_iteraciones==150:
137                     E_fondo = gris # El nuevo fondo sera gris
138                     umbralFondo = met.umbralizar(gris)
139                     VC_comenzarRegistroMovimiento = True
140                 while(C_iteraciones<150):
141                     C_iteraciones+=1
142                     (grabbed, frame) = E_camara.read()
```

```
58  # metodo para pintar las manchas
59  def visualizarLista(lista):
60      lista1 = ["mancha1.jpg","mancha2.jpg","mancha3.jpg","mancha4.jpg"]
61      i=0
62      for x in lista:
63          cv2.imwrite(lista1[i], x)
64          i+=1
```

Índice:

Análisis del cambio

```
# Si el contorno es tan pequeño suponemos es algo que no deberia estar...
if contorno <= 200:
    continue
# Si cumple los requisitos suponemos que es movimiento.
# Habria que poner una condicion de si se detecta el movimiento fuera de un cuadrado...
elif contorno > 200 and contorno<total and comenzarRegistroMovimiento == True:
    # Hacemos una especie de contador para que solo lo haga 1 vez
    if comprPAP == False:
        # Calculamos el punto (x2,y2):
        y2 = y1 + h
        x2 = x1 + w
        manchaNueva = edges[y1:y2, x1:x2]
        pos = 0
        pos, funciona = met.lorenzo(listaHel, manchaNueva)
        if funciona!=False:
            print("Hay movimiento, creemos que es el heliostato ",(pos+1))
        #time.sleep(10)
        if cont2==30 or funciona == False:
            comprPAP = True
            print("siguiente")
            cont2=0
            funciona = True
        else:
            cont2+=1
```

Análisis del cambio

```
99  def lorenzo(lista, mancha):
100     listaValores = []
101     listaValores2 =[]
102     a = 0
103     for x in lista:
104         img = x
105         cv2.imshow("Camara", img)
106         img2 = img.copy()
107         template = mancha
108         w, h = template.shape[::-1]
109         # Apply template Matching
110         res = cv2.matchTemplate(img,template, cv2.TM_CCOEFF) #'cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR','cv2.TM_CCORR_NORMED'
111         min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)
112         listaValores.append(max_val)
113     print(listaValores)
114     return listaValores.index(max(listaValores))
```

Índice:

Finalización del programa

```
185         # Capturamos una tecla para salir
186         key = cv2.waitKey(1) & 0xFF
187 
188         # Tiempo de espera para que se vea bien
189         time.sleep(0.015)
190 
191         # Si ha pulsado la letra s, salimos
192         if key == ord("s"):
193             break
194 
195 
196 # Liberamos la cámara y cerramos todas las ventanas
197 camara.release()
198 cv2.destroyAllWindows()
```

Índice:

Métodos fallidos

```
8  # Es como el anterior comprobarPAP pero reduzco la busqueda al borde izquierdo
9  def comprobarPAP2(array, mancha):
10     array2 = aumentarLista(array,None)
11     listaMancha = []
12     hora1 = time.time()
13     for k in array2:
14         hora5 = time.time()
15         lista = []
16         for x in range(0,3):#mancha.shape[0]
17             for y in range(0,k.shape[1]):
18                 cont = 0
19                 for x2 in range(0,mancha.shape[0]):
20                     for y2 in range(0,mancha.shape[1]):
21                         if (mancha.shape[1]+y)<=k.shape[1] and (mancha.shape[0]+x)<=k.shape[0]:
22                             if mancha[x2][y2] == k[x+x2][y+y2]:
23                                 cont+=1
24                         else:
25                             break
26                 lista.append(cont)
27         listaMancha.append(max(lista))
28         hora4 = time.time()
29         #print("Procesamiento de la mancha:", (hora4-hora5),"segundos.")
30     print("elementos en listaMancha",listaMancha[0],listaMancha[1],listaMancha[2],listaMancha[3])
31     hora2 = time.time()
32     #print("ComprobarPAP dura", (hora2-hora1), "segundos.")
33     return listaMancha.index(max(listaMancha))
```

Métodos fallidos

```
35  # Metodo para añadir una cantidad de ceros a la izquierda.
36  def aumentarLista(lista, cantidad):
37      if cantidad == None:
38          cantidad=2
39      listaFinal = []
40      listaFinalAux = []
41      lista1 = []
42      lista2 = []
43      # Llenamos una lista con la cantidad de 0 que queremos añadir
44      for x in range(0,cantidad):#mancha.shape[0]
45          lista1.append(0)
46
47      for x in lista:
48          for y in x:
49              lista2 = lista1.copy()      # Copiamos la lista llena de 0
50              lista3 = y.copy().tolist()  # Convertimos la numpy.ndarray en una lista
51              lista2.extend(lista3)       # Juntamos las dos listas
52              listaFinalAux.append(lista2)# Añadimos en una lista todas las listas con los 0s ya añadidos
53          listaFinal.append(np.array(listaFinalAux)) # Añadimos en una lista normal la lista de tipo ndarray
54          listaFinalAux = []
55      return listaFinal
```

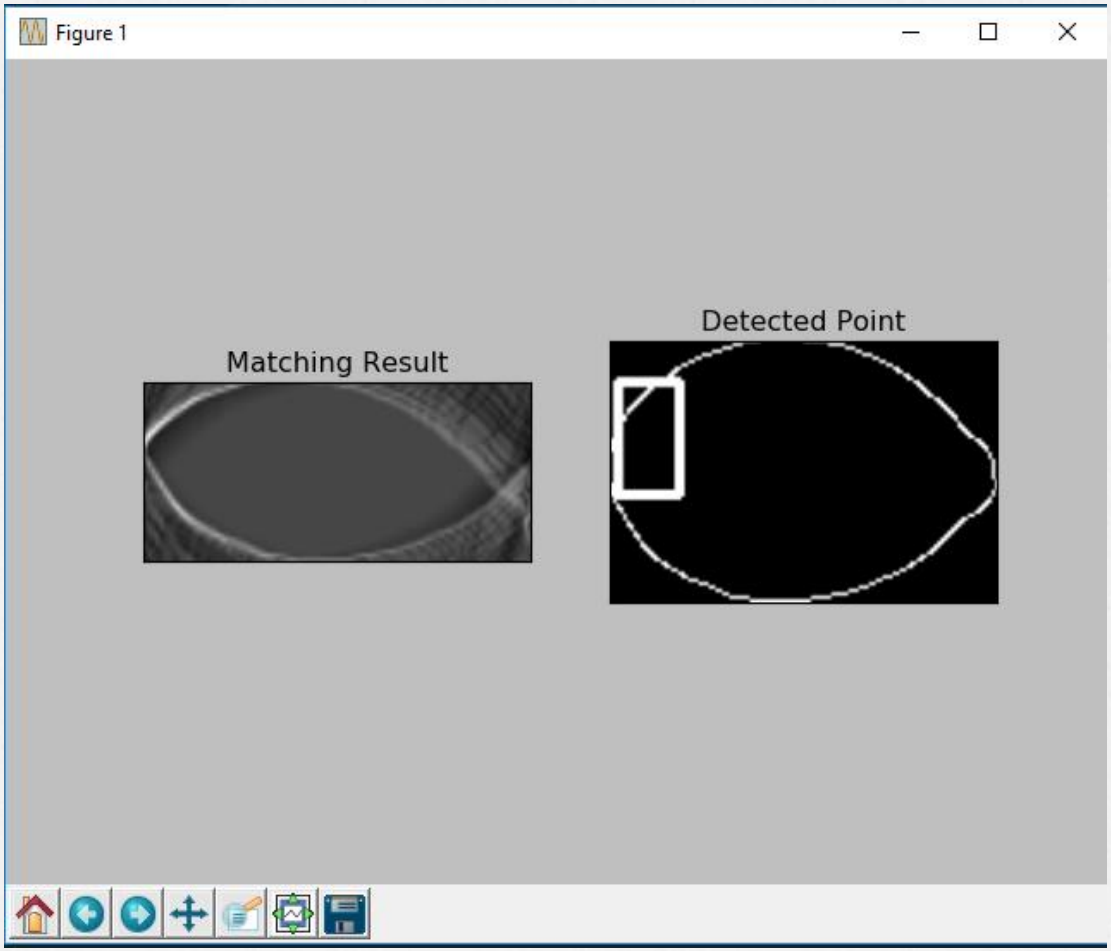
Índice:

Coste algorítmico

- Coste algorítmico n^3 en la clase conjuntoCodigo.py. En la línea 153 nos encontramos 3 bucles anidados. Ya que es un while dentro de un for, de un while.
- En la clase métodos.py en la línea 84 el método “Lorenzo” tiene un coste n^3

En conjunto un coste n^5 ya que el método Lorenzo está dentro de dos bucles anidados.

Resultado



¿Preguntas?

Diego Zapata Hernández
Juan Diego González Ramos
Daniel Hagen González
Antonio Fernández Rubio
Sergio Guillén González

GitHub --> diegozahe/light-beams-unmixing