EJEMPLO 3

Gestión de Eventos

- Manejador eventos Ratón
- Guardar imagen ventana al pulsar SHIT+LeftClick

https://docs.opencv.org/4.5.5/d7/dfc/group__highgui.html

Gestión de Eventos: módulo highgui

- Ejemplo (**e3.py**): (partiremos del código de e1b.py)
  - Partimos del código de captura de imágenes **e1.py**
- Manejador de Ratón:
  - Definir la función manejador:
    - def **onMouse** (**event**, **x**, **y**, **flags**, **param**):
  - Asignar Manejador:
    - cv.**setMouseCallback**(**winname**, **onMouse** [, **userdata**=None])
- Implementación Manejador:

```
// Mouse events handler  for image window
// event:        event type sent to the handler ->  cv.EVENT_MOUSEMOVE,
//          cv.EVENT_LBUTTONDOWN, cv.EVENT_LBUTTONUP, cv.EVENT_LBUTTONDBLCLK,
//          cv.EVENT_RBUTTONDOWN, cv.EVENT_RBUTTONUP, cv.EVENT_RBUTTONDBLCLK,
//          cv.EVENT_MBUTTONDOWN, cv.EVENT_MBUTTONUP, cv.EVENT_MBUTTONDBLCLK,
//          cv.EVENT_MOUSEWHEEL, cv.EVENT_MOUSEHWHEEL
// x:     X-coordinate position of the mouse in window
// y:     Y-coordinate position of the mouse in window
// flags: aditional flags sent to the handler ->
//          cv.EVENT_FLAG_SHIFTKEY, cv.EVENT_FLAG_CTRLKEY, cv.EVENT_FLAG_ALTKEY,
//          cv.EVENT_FLAG_LBUTTON, cv.EVENT_FLAG_RBUTTON, cv.EVENT_FLAG_MBUTTON,
// param: set in cv.SetMouseCallback
```

Gestión de Eventos: módulo highgui

- Ejemplo (**e3.py**):

```
WINDOW_CAMERA1 = '(W1) Camera 1'   # window id
CAMERA_ID = 0                                    # default camera
KEY_F5 = 7602176        # F5 unicode key code
ID_FILE = 1                    # filename id
```

```
#  Mouse events handler  for image window
def onMouse(event, x, y, flags, param):

    global capture, ID_FILE,  # global variables used in the mouse handler
    print(f"{event=}, {x=}, {y=}, {flags=}, {param=}")

    # on click left mouse button and SHIFT key, saves image
    if event ==cv.EVENT_LBUTTONDOWN and (flags & cv.EVENT_FLAG_SHIFTKEY) :
        filename = f"Image{ID_FILE}.jpg"
        print(f"Saving image window in file: {filename}")

        cv.imwrite(filename, capture)   # save window image
        ID_FILE += 1
```

```
# Setting Mouse Handler
cv.setMouseCallback( WINDOW_CAMERA1, onMouse, None)
```

Gestión de Eventos: módulo highgui

- Ejemplo (**e3.py**):

```
# while there are images ...
while True:
    ret, capture = camera.read()       # Capture frame-by-frame

    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break

    cv.imshow(WINDOW_CAMERA1, capture)     # Display the resulting frame

    # check keystroke to exit (image window must be on focus)
    key = cv.pollKey()
    if key == ord('q') or key == ord('Q') or key == 27:
        break

     elif key == KEY_F5 or key == ord(' ') :
        filename = f"Image{ID_FILE}.jpg"
        print(f"Saving image window in file: {filename}")
        cv.imwrite(filename, capture)   # save window image
        ID_FILE += 1

# End while (main loop)
```

EJERCICIO 3b

Interfaz de usuario. (e3b.py)

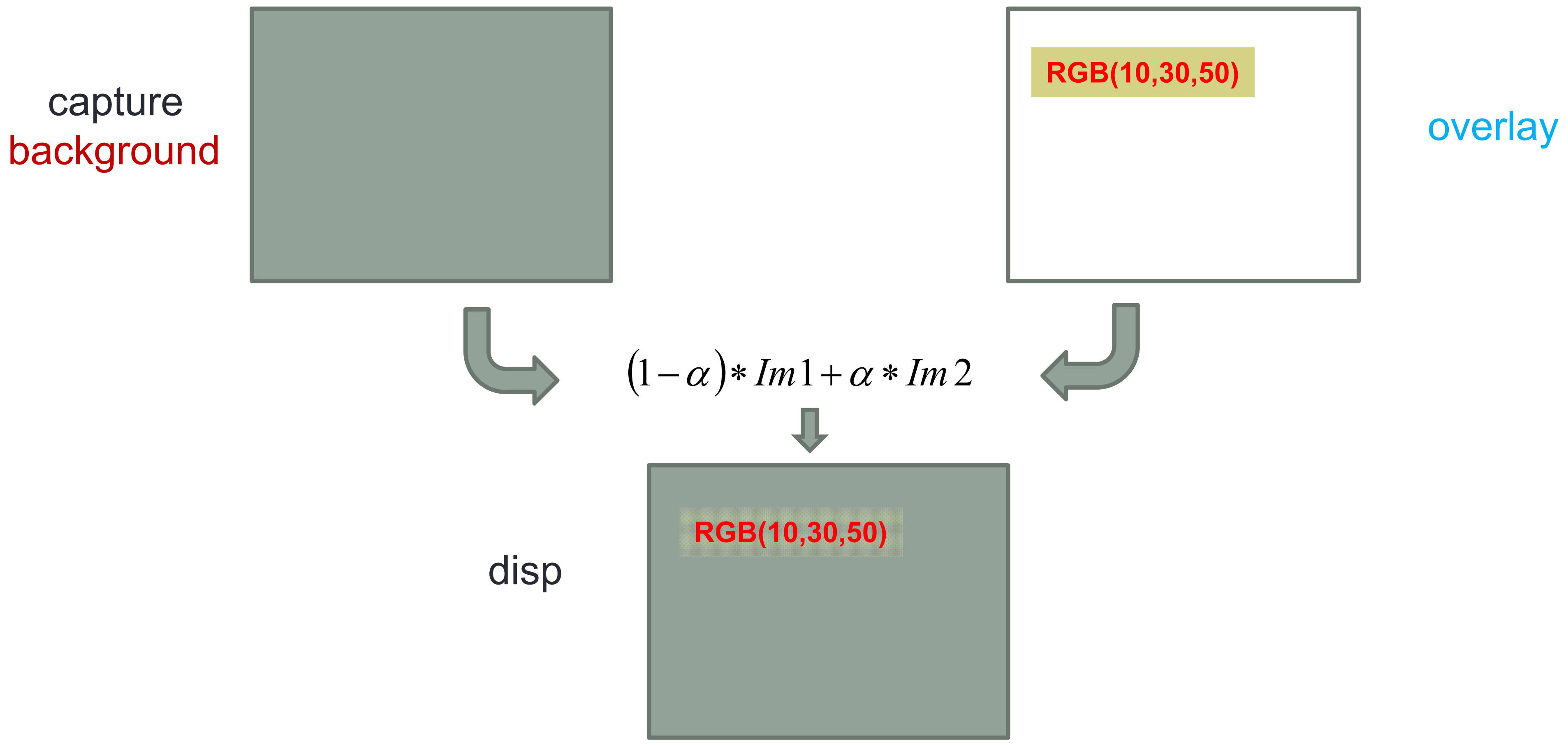
- Visualización del color en cada pixel bajo el cursor, mostrándolo en overlay semi-transparente sobre la imagen
- TrackBars: modificar parámetros de ejecución
- Botón de salida en overlay

https://docs.opencv.org/4.5.5/d7/dfc/group__highgui.html

https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

Interfaz Usuario: módulo core

- Ejercicio:
  - Visualización del color en cada pixel bajo el cursor, mostrándolo en overlay semi-transparente sobre la imagen



cv.**addWeighted(src1**, **alpha**, **src2**, **beta**, **gamma** [, **dst** [, **dtype**]] **)** → **dst**

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

Interfaz Usuario: módulo core

- Constructores adicionales clase *ndarray*:
  - **overlay** = np.**zeros**(capture.shape, **dtype**=np.uint8)
  - **overlay** = np.**zeros**((capture.shape[0], capture.shape[1], 3), **dtype**=np.uint8)

- Métodos adicionales *numpy*:
  - **overlay**[:] = 0 # borrar una imagen
  - **overlay**[:] = (b,g,r) # Asignar color una imagen

- Acceder al valor de un pixel:
  - **capture**[row, col] → scalar/tuple
  - **capture**[(row, col)] → scalar/tuple
  - **capture**.**item**(row, col, cha) → scalar (optimizado, solo devuelve el valor de un canal)

- Generar un string formateado en Python: *f-string*

```
CURSOR_POS = (0, 0)        # current position of the cursor over the window (row,col)

color = capture[CURSOR_POS];
cursorColor = f"RGB{color[::-1]}"          # reverse BGR tuple
print(cursorColor)
```

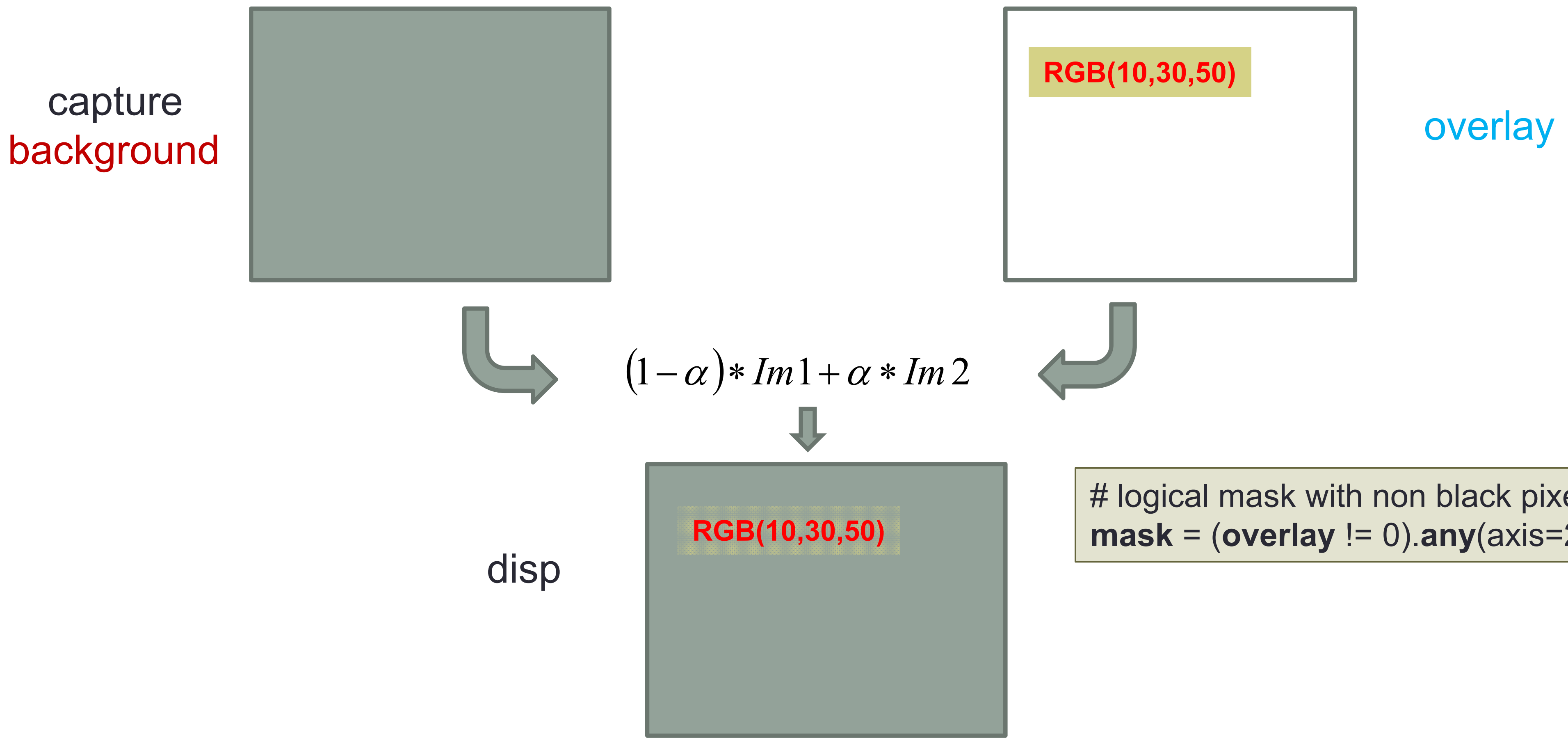
Interfaz Usuario: módulo core

- Event Handlers:
  - Código adicional para actualizar coordenadas del cursor

```
#  Mouse events handler  for image window
def onMouse(event, x, y, flags, param):

          # global variables used in the mouse handler
         global CURSOR_POS

         …….

         # on moving the cursor over the image
         if event == cv.EVENT_MOUSEMOVE :
                  CURSOR_POS = (y, x)  # save new cursor position in global variable (row,col)

         …….
```

Interfaz Usuario: módulo core

- Indexación clase *ndarray* mediante matrices lógicas:
  - Como índice podemos usar una matriz lógica (booleana). El valor **True** indica que el pixel es procesado → Máscara ROI (*Region Of Interest*)



```
# logical mask with non black pixels
mask = (overlay != 0).any(axis=2)
```

```
background[mask] = cv.addWeighted(background[mask], 1 - alpha, overlay[mask], alpha, 0)
```

$$dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma$$

Interfaz Usuario: módulos highgui/core

- Trackbars:
  - cv.**createTrackbar** (**trackbarname**, **winname**, **value**, **count**, **onChange**)

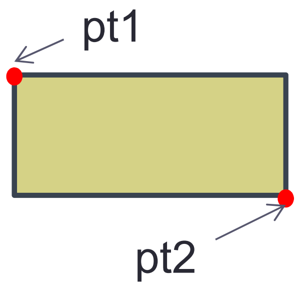
```
cv.createTrackbar ("Transp.", WINDOW_CAMERA1,  ALPHA,  100,  onTrackbar)
```

  - cv.**setTrackbarMax**(trackbarname, winname, **maxval**)
  - cv.**setTrackbarMin**(trackbarname, winname, **minval**)
  - cv.**setTrackbarPos**(trackbarname, winname, **pos**)
  - cv.**getTrackbarPos**(trackbarname, winname) → retval

```
def onTrackbar(x):
   global ALPHA
   ALPHA = x

ALPHA = 60
```

- Dibujo en pantalla:
  - cv.**line** (**img**, **pt1**, **pt2**, **color** [, **thickness**=1 [, **lineType**=cv.**LINE_8** [, **shift**=0]]])
  - cv.**rectangle** (**img**, **pt1**, **pt2**, **color** [, **thickness**=1 [, **lineType**=cv.**LINE_8** [, **shift**=0]]])
  - cv.**circle** (**img**, **center**, **radius**, **color** [, **thickness**=1 [, **lineType**=cv.**LINE_8** [, **shift**=0]]]

- Puntos (**pt1,pt2,center**): (x,y) | [x,y] | np.array([x,y])
- Tipo de líneas/**lineType**: cv.**LINE_8** 8-connected line, cv.**LINE_4** 4-connected line,
  cv.**LINE_AA** - antialiased line
- Grosor/**thinkness**: en objetos con área podemos especificar: cv.**FILLED**



```
BUTTON_SIZE = np.array((160, 25))   # overlay button Size (width, heigth)
BUTTON_POS = np.array((0, 0))  # overlay button upper left corner position (x, y)

cv.rectangle (overlay, BUTTON_POS, BUTTON_POS + BUTTON_SIZE,
                          (0,120,120),  cv.FILLED);
```

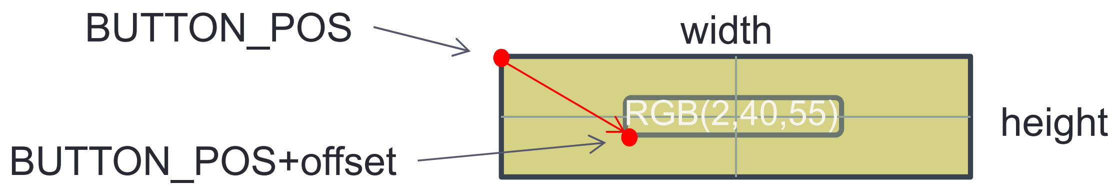
Interfaz Usuario: módulos highgui/core

- Texto:
  - cv.**getTextSize** (**text**, **fontFace**, **fontScale**, **thickness**) → size, baseLine

    - size: (width,height)
    - baseLine (int): y-coordinate of the baseline relative to the bottom-most text point.
  - cv.**putText** (**img**, **text**, **org**, **fontFace**, **fontScale**, **color** [, **thickness**=1 [, **lineType**=cv.LINE_8 [, **bottomLeftOrigin**=False]]])
    - Fuentes: cv.FONT_HERSHEY_SIMPLEX, cv.FONT_HERSHEY_PLAIN, cv.FONT_HERSHEY_DUPLEX cv.FONT_HERSHEY_COMPLEX, cv.FONT_HERSHEY_COMPLEX_SMALL, cv.FONT_HERSHEY_SCRIPT_SIMPLEX, cv.FONT_HERSHEY_SCRIPT_COMPLEX
    - Se pueden combinar con: | cv.FONT_ITALIC

```
cv.putText (overlay,  cursorColor,  BUTTON_POS + offset.astype(int) ,
                cv.FONT_HERSHEY_DUPLEX, 0.4, (255,255,255), 1,  cv.LINE_AA )
```

```
# offset para centrado de texto en un recuadro

textSize, baseline = cv.getTextSize(cursorColor, cv.FONT_HERSHEY_DUPLEX, 0.4, 1 )

offset = BUTTON_SIZE*0.5 + np.array(textSize)*[-0.5, 0.5]
```



Interfaz Usuario: módulo core

- Event Handlers:

```
#  Mouse events handler  for image window
def onMouse(event, x, y, flags, param):
        global CURSOR_POS, EXIT, BUTTON_POS, BUTTON_SIZE
        …..
        # on moving the cursor over the image
        if event == cv.EVENT_MOUSEMOVE :
                CURSOR_POS = (y, x)  # save new cursor position in global variable (row,col)

        # on click left mouse button
        if event == cv.EVENT_LBUTTONDOWN :
                # checks if Exit button is clicked
                if (x > BUTTON_POS[0] and x < (BUTTON_POS[0] + BUTTON_SIZE[0]) and
                        y > BUTTON_POS[1] and y < (BUTTON_POS[1] + BUTTON_SIZE[1]) ):
                    EXIT = True

# Trackbar events handler
def onTrackbar(x):
    global ALPHA
    ALPHA = x
```

```
# Global variables
ALPHA = 60            # % level of transparency
EXIT = False          # exit the program
CURSOR_POS = (0, 0)     # current position of the cursor over the window (row,col)
BUTTON_SIZE = np.array((160, 25))   # Overlay button Size (width,heigth)
BUTTON_POS = np.array((0, 0))      # overlay button upper left corner position (x,y)
```

Interfaz Usuario: módulo core

- Inicialización:

```
// enables a trackbar associated to variable ALPHA
cv.createTrackbar ("Transp.", WINDOW_CAMERA1,  ALPHA,  100,  onTrackbar)
```

- Función DrawOverlay:

```
def drawOverlay( image, alpha=0.6):
     global CURSOR_POS, BUTTON_POS, BUTTON_SIZE

     background = image.copy()     # creates a copy to preserve original image

    #Allocates memory for overlay image of the same size as background
    overlay = np.zeros((image.shape[0], image.shape[1], 3),  dtype=np.uint8)

    # if input image  is not BGR, it is converted to BGR
    if image.ndim < 3:
         background = cv.cvtColor(background, cv.COLOR_GRAY2BGR)

    …………………………………..
```

Interfaz Usuario: módulo core

- Función DrawOverlay:

```
def drawOverlay( image, alpha=0.6):

    ………………………………

   // Draws the overlay image
  cv::rectangle(overlay, BUTTON_POS, BUTTON_POS + BUTTON_SIZE, (0, 120, 120), cv.FILLED)
  cv::rectangle(overlay, BUTTON_POS, BUTTON_POS + BUTTON_SIZE, (0,255,255), 1)

   # text with the pixel color under the cursor
   color = image[CURSOR_POS]
   if np.ndim(color) == 0:        # color is scalar
       cursorColor = f"Gray[{color}]"  # Gray scale level
   else:
       cursorColor = f"RGB{color[::-1]}"  # reverse BGR tuple

     ………………………………………………
```

Interfaz Usuario: módulo core

- Función DrawOverlay:

```
def drawOverlay( image, alpha=0.6):

  ………………………………

  # offset to draw text centered in the rectangle
  textsize, baseline = cv.getTextSize(cursorColor, cv.FONT_HERSHEY_DUPLEX, 0.4, 1)
  offset = BUTTON_SIZE*0.5 + np.array(textsize)*[-0.5, 0.5]

  cv.putText(overlay, cursorColor, BUTTON_POS + offset.astype(int),
             cv.FONT_HERSHEY_DUPLEX, 0.4, (255, 255, 255), 1, cv.LINE_AA)

  # blending both images
  mask = (overlay!=0).any(axis=2)    # logical mask with non black pixels (0,0,0) in overlay
  background[mask] = cv.addWeighted(background[mask], 1-alpha, overlay[mask], alpha, 0)

  return background
```

Interfaz Usuario: módulo core

- Bucle Principal:

```
# while there are images ...
while True:


    .....



    # draws overlay with pixel color under cursor on capture image
   disp  = drawOverlay(capture, float(ALPHA)/100)

   cv.imshow(WINDOW_CAMERA1, disp)     # Display the frame


    key = cv.pollKey()
    if key == ord('q') or key == ord('Q') or key == 27 or EXIT:
        break
    ……

# End while (main loop)
```