2. Actividad

La resolución de actividad se encuentra en el fichero de `ACTIVIDAD.py`.

Introdución

En el fichero se encuentra resueltas las siguientes cuestiones:

1. Selección de una región dentro de una imagen
2. Captura del trozo guardado y detección de la actividad con respecto al trozo guardado.
3. Grabación de un video conteniendo 2 segundos antes de la actividad, hasta el fin de la misma. Opcionalmente, se ha realizado el recorte del objeto en cuestion, mediante una mascara.

Uso

La ejecución se realiza con los argumentos que se pasaría a `stream.py`. Los controles adicionales son los siguientes:

- 'c': Guarda la región seleccionada como muestra
- 'x': Eliminar la region seleccionada. La aplicación analizara la actividad ocurrida en la seccion seleccionada con respecto a la muestra tomada. En caso de detectar actividad realizara la grabación de un video. Así como mostrar el objeto recortado del fondo.

Se pueden ajustar la sensibilidad de recorte y detección en tiempo real mediante las tracbars.

Implementación

Para la implentación se han uso de las utilidades de `umucv`. En el codigo ha sido comentado adecuadamente y se procedera a expandir algunos fragmentos.

Buffer de video

Para almacenar los ultimos 2 segundos se hace uso de `deque`, con un tamaño de los FPS de la camara * 2. Por defecto es 60.

Creación de video

En la clase `ControlAct` se encuentra la utilidad para grabar video.

In [8]:
```
    def start_video(self):  # Inicia el video y guarda los ultimos x segu
        if self.video or len(self.last_frames) < 1:
            return
        self.video = Video(fps=FPS, codec="MJPG", ext="avi")
        self.video.ON = True
        for f in self.last_frames:
            self.video.write(f)
        self.last_frames.clear()

    def continue_video(self, f):  # Añade el frame actual
        if self.video:
            self.video.write(f)

    def stop_video(self):  # Detiene el video
        self.video.ON = False
        self.video.release()
        self.video = None
```

- `start_video` : Crea el video, graba en el los frames almecenados y el actual.
- `continue_video` : Graba el frame actual.
- `stop_video` : Detiene el video y lo cierra.

Detección de Actividad

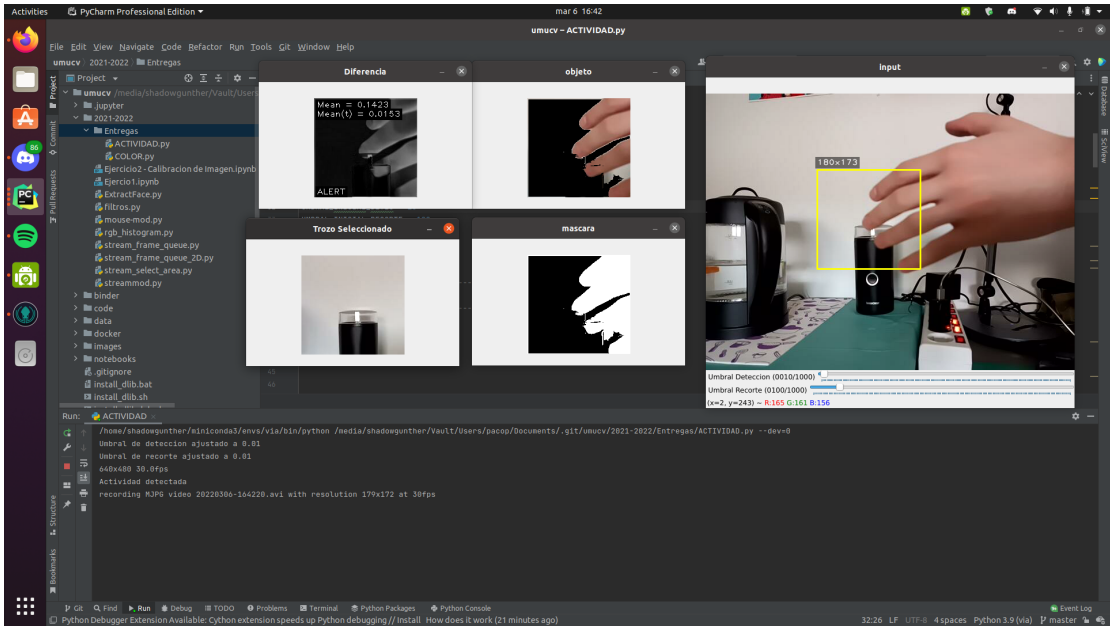
El código encargado de la detección de actividad es el siguiente:

```
            if data.saved_trozo is not None:  # Si hay trozo guardado,
empezamos con la vigilancia
                diff = cv.absdiff(data.saved_trozo, recorte)  #
Calculamos la diferencia del actual con el guardado
                diff = bgr2gray(diff)
                mean = np.mean(diff)
                means = mean  # La media nos da una aproximación de
cuanto ha cambiado con la de referncia

                if len(data.last_mean) >= SUAVIZAR_MEDIA:  # Si hay
suficientes medias las suavizamos conforme el tiempo
                    means = np.mean(data.last_mean)

                if np.abs(means - mean) <= data.umbral_deteccion:  #
Si la variación no es suficiente grande.
                    data.last_mean.append(mean)
                    data.last_frames.append(recorte)  # Actualizamos
las medias y guardamos frame
                    if data.estado is not Estado.END:  # Si venimos
de un estado de actividad
                        data.stop_video()  # Detenemos grabación y
destruimos ventanas residuales
                        print('Fin actividad')
                        data.estado = Estado.END
                        try:
                            cv.destroyWindow('mascara')
                            cv.destroyWindow('objeto')
                        except Exception:
                            pass
                else:  # Si Hay diferencia
                    mask = diff > data.umbral_recorte  # Creamos
mascara
                    cv.imshow('mascara', mask.astype(float))
                    mask = gray2bgr(mask)
                    objeto = np.zeros_like(recorte)  # Recortamos la
mascara
                    np.copyto(objeto, recorte, where=mask == 1)
                    cv.imshow('objeto', objeto)
                    putText(diff, 'ALERT', orig=(5, diff.shape[0] -
5))  # Notificamos la alerta

                    if data.estado is Estado.END:  # Si no estamos
grabando, empezamos
                        print('Actividad detectada')
                        data.start_video()
                        data.continue_video(recorte)
                        data.estado = Estado.ACTIVITY
                    elif data.estado is Estado.ACTIVITY:  #
Continuamos grabando
                        data.continue_video(recorte)
```

El primer `if` es una guarda para evitar analizar cunado no hay muestra. El proceso consiste en: Calculamos la diferencia absoluta de la muestra y el recorte (en blanco y negro) y calculamos la media. Si hay suficientes medias anteriores almacenadas se usa la media de las medias, esta decisión se ha tomado para suavizar el comportamiento de cambios de iluminación graduales temporales (pasa una nube), flickering de la camara o el autofocus. (*Durante la realización de las pruebas me dieron problemas en especial el autofocus*). Siguiente a eso, si la diferencia de la media actual con las ultimas mediciones no supera el umbral, se añade el frame al buffer y se añade esa media al buffer de ultimas medias, además si se estaba grabando se detiente la grabación y limpian las ventanas sobrantes. En caso contrario, la diferencia supera el umbral, se empieza a grabar, se recorta el objeto (gracias a que cualquier pertubación en la imagen se ve en gris o blanco en la imagen) facilitando así la creación de una mascara.

El proceso de deteción se realiza mediante una media aritmetrica, en este uso funciona decentemente bien debido a que el numero de pixeles es el mismo, y la variación del valor de gris de un grupo de pixeles afecta a la media de la imagen sustancialmente.

Ejemplo



3. Color

La resolución de actividad se encuentra en el fichero de `COLOR.py`.

Introdución

En el fichero se encuentra resueltas las siguientes cuestiones:

1. Selección de una región dentro de una imagen
2. Representación en vivo del color de la region mediante histograma, (mostrado en lineas).
3. Detección de similitud de la región con un grupo de muestra.

Uso

La ejecución se realiza con los argumentos que se pasaría a `stream.py`. Los controles adicionales son los siguientes:

- 'c': Guarda la región seleccionada como muestra
- 'x': Elimina la selección de la región
- 'r': Elimina los modelos almacenados
- 'n': Selecciona el siguiente método de comparación La aplicación muestra la cantidad de color en la region seleccionada, la capacidad de captura de la misma como muestra y la selección de la muestra más parecida a la selección actual.

Implementación

Para la implentación se han uso de las utilidades de `umucv`.

Creacion de histogramas

Las funciones de codigo encargadas de la creación de histogramas son las siguientes.

In [9]:
```
def make_histogram(c, size):  # Crea un histograma normalizado y el adapt
    h, b = np.histogram(c, np.arange(0, 257, 4))
    x = 2 * b[1:]
    yn = h / np.sum(h)
    y = size - h * (size / h.max())
    xy = np.array([x, y]).T.astype(int)
    xyn = np.array([x, yn])
    return xy, xyn
```

Recibe los canales de color (c) y la altura de la region, para crear un histograma normalizado `xyn` y el que se muestra `xy`. Esta función es usada por `make_rgb_histogram` para crear los histogramas de cada color.

In [10]:
```
def make_rgb_histogram(f, size):  # Hace los histogramas para cada canal
    blue, green, red = cv.split(f)
    blue = make_histogram(blue, size)
    green = make_histogram(green, size)
    red = make_histogram(red, size)
    return blue, green, red
```

Separa los canales de color de la region seleccionada y crea los histogramas para cada canal.

Comparación de histogramas

Para la comparación de histogramas se han implementado dos metodos. Intersección de histogramas (*El que mayor valor de intersección devuelve es el más probable que sea el más cercano*). Y por diferencia absuluta(*Cuanto menor sea la diferencia más acerca al modelo*).

In [11]:
```
def hg_diff(hg1, hg2):  # Comparacion de histogramas por diferencia absol
    hgy1 = hg1[:, 1]
    hgy2 = hg2[:, 1]
    diff = np.abs(hgy1 - hgy2)
    return diff


def hg_intersect(hg1, hg2):  # Comparacion de histogramas por interseccio
    mini = np.minimum(hg1, hg2)
    result = np.true_divide(np.sum(mini), np.sum(hg2))
    return result
```

Selección de modelo

La seleccíon del modelo más parecido se apoya de las siguietes dos funciones:

In [12]:
```
def select_candidate(values, mode):  # Seleciona que canal se diferencia
    if mode == 'min':
        return np.max(values)
    elif mode == 'max':
        return np.min(values)


def is_better(value, last_value, mode):  # Comprueba si el modelo actual
    if last_value is None:
        return True
    elif mode == 'min':
        return value < last_value
    elif mode == 'max':
        return value > last_value
```

La primera, selecciona el valor más significativo de los tres canalaes de color, el modo define como se comporta. Por ejemplo en caso de la intersección el valor más pequeño es el más distinto, mientras que en diferencias el más alto es el más distinto. La segunda devuelve si el modelo actual es mejor candidato que el anterior candidato elegido.

In [13]:
```
def select_most_like_model(data, hgn, method):  # Selecciona el mejor moa
    values = list()
    index = 0
    last_min = None
    b, g, r = hgn
    for i in range(0, len(data.patrones)):
        bp, gp, rp = data.patrones[i].color_info
        aux_b = method.fun(bp, b)
        aux_g = method.fun(gp, g)
        aux_r = method.fun(rp, r)
        value = select_candidate([aux_b, aux_g, aux_r], method.selector)
        values.append(value)
        if is_better(value, last_min, method.selector):
            index = i
            last_min = value

    return values, data.patrones[index]
```

La función de seleccion del modelo más parecido, buscando entre todos los candidatos.

Ejemplo