



UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES

Universidad Del Bío-Bío

Detección y Cambio de Dimensión en Grilla

Ingeniería Civil Informática

Jerson Antonio Palma Sarandona
Ingeniería Civil en Informática en formación

Introducción

El desarrollo de aplicaciones para la detección de grilla y ajuste de dimensiones de objetos debido a la humedad es esencial en diversas áreas, como la industria, la investigación y la agricultura. Este informe presenta una aplicación interactiva desarrollada en Python utilizando las bibliotecas OpenCV, NumPy, y Tkinter, que permite visualizar, analizar y procesar imágenes para detectar puntos de interés en una grilla.

Consideraciones.

Antes de embarcarse en este tutorial, es imperativo haber completado los tutoriales previos de nivelación en Python y el tutorial de detección de esquinas. El aprendizaje en este contexto es progresivo, y cada tutorial establece la base necesaria para comprender conceptos y técnicas más avanzados.

Además, es importante destacar que este tutorial hace uso de las bibliotecas fundamentales: OpenCV, NumPy y Tkinter. Estas bibliotecas fueron instaladas y configuradas durante los tutoriales anteriores. OpenCV se utiliza para el procesamiento de imágenes, NumPy para manipulación de arreglos y cálculos, y Tkinter para la creación de la interfaz gráfica de usuario.

Funcionalidades Principales.

1. Grilla Ideal

La función `crear_grilla` genera una grilla ideal con celdas de tamaño definido. Utiliza el algoritmo de Harris para detectar puntos de interés en la grilla y muestra la imagen resultante en la interfaz gráfica.

2. Humedad

La función `mostrar_humedad` permite cargar dos imágenes y aplicar el algoritmo de Harris para detectar puntos de interés en cada imagen. Además, calcula y muestra la distancia entre los puntos seleccionados, lo que facilita el análisis de cambios debidos a la humedad.

3. Fricción

La función `mostrar_friccion` permite cargar dos imágenes y aplicar el algoritmo de Harris para detectar puntos de interés en cada imagen. Además, calcula y muestra la distancia entre los puntos seleccionados, lo que facilita el análisis de cambios debidos a la

Código.

- Sección de importación: Aquí se están importando las bibliotecas que se utilizarán en el resto del script. Cv2, Numpy, tkinter, PIL, time, entre otros.

```
1  import cv2
2  import numpy as np
3  import tkinter as tk
4  from tkinter import ttk
5  from tkinter import filedialog
6  from tkinter import font
7  from PIL import Image, ImageTk, ImageDraw
8  import imutils
9  import time
10 import os
11 from tkinter import messagebox
12 import threading
13 import matplotlib.pyplot as plt
14 from matplotlib.ticker import MaxNLocator
```

Primero declararemos las variables sobre las cuales se guardarán las intersecciones encontradas en las imágenes y declarando un tamaño predeterminado, dichas variables se usaran más adelante

```
16 # Lista para almacenar los puntos
17 points = []
18 # Variables para el seguimiento de la selección de puntos algoritmo de harris
19 selected_points = []
20 selected_point_indices = []
21
22
23 # Tamaño deseado (750x750)
24 target_size = (750, 750)
```

Funciones.

LimpiarLabels () borra el contenido de las listas anteriormente definidas

```
27  def limpiarlabels():
28      points.clear()
29      selected_points.clear()
30      selected_point_indices.clear()
31      label1.config(text="", bg="gray")
32
```

Resize_image () cambia de tamaño la imagen tomando como argumento, el path de la imagen a cambiar, el tamaño deseado y retorna la imagen ya cambiada

```
34  def resize_image(image_path, target_size, background_color=(255, 255, 255)):
35      # Abrir la imagen con Pillow
36      original_image = Image.open(image_path)
37      # Crear una nueva imagen del tamaño deseado con el fondo especificado
38      new_image = Image.new("RGB", target_size, background_color)
39      # Calcular las coordenadas para centrar la imagen original en la nueva imagen
40      x_offset = (target_size[0] - original_image.width) // 2
41      y_offset = (target_size[1] - original_image.height) // 2
42      # Pegar la imagen original en la nueva imagen en las coordenadas calculadas
43      new_image.paste(original_image, (x_offset, y_offset))
44      return new_image
```

Update_imagen () Actualiza la imagen recibida para ser mostrada en la ventana

```
85  def update_image(label2, actualizacion):
86      # Convierte la imagen para mostrar en la ventana de Tkinter a formato RGB
87      image_to_show = cv2.cvtColor(actualizacion, cv2.COLOR_BGR2RGB)
88      image_to_show = Image.fromarray(image_to_show)
89      image_to_show = ImageTk.PhotoImage(image_to_show)
90      label2.configure(image=image_to_show)
91      label2.image = image_to_show
```

Umbralizacion () Umbraliza la imagen en escala de grises, recibe una imagen y la retorna umbralizada

```
99  def umbralizacion(grays, valor):
100      ret, thresh1 = cv2.threshold(grays, valor, 255, cv2.THRESH_BINARY)
101      return thresh1
```

Creación De Menú y Ventana:

- creación de ventana, configuración de dimensiones y label con el logo de la universidad, también se le asigna una fuente al texto.

```
914 # crea todo el menu de inicio
915 ventana = tk.Tk()
916 ventana.geometry("1920x900")
917 ventana.state("zoomed")
918 ventana.title("ANALIZADOR DE GRILLAS")
919 ventana.configure(bg="#212f4e")
920 fuente_texto = font.Font(family="Helvetica", size=16, weight="bold")
921 #ventana.resizable(0, 0)
922
923
924 global labellogo
925 ubblogo= Image.open("C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/escudo-monocromatico-oscuro.png")
926 ubblogo = ubblogo.resize((500, 329))
927 labellogo = tk.Label(ventana, bg="#212f4e")
928 labellogo.imagen_ubb = ImageTk.PhotoImage(ubblogo) # Guardar la imagen como un atributo del widget Label
929 labellogo.config(image=labellogo.imagen_ubb)
930 labellogo.place(x=700, y=80)
```

- Creación de botones, configuración de dimensiones, color, fuente y función a realizar

```
931 botonA = tk.Button(
932     ventana,
933     text="TRABAJO CON DATOS ALMACENADOS",
934     command=mostrar_menu,
935     relief=tk.GROOVE,
936     borderwidth=10,
937     bg="#d97675",
938     fg="BLACK",
939     font=fuente_texto,
940 )
941 botonA.place(x=690, y=550)
942 botonB = tk.Button(
943     ventana,
944     text="NUEVO PROYECTO",
945     command=mostrar_menu2,
946     relief=tk.GROOVE,
947     borderwidth=10,
948     bg="#d97675",
949     fg="BLACK",
950     font=fuente_texto,
951 )
952 botonB.place(x=820, y=710)
953 boton1 = tk.Button(
954     ventana,
955     text="1. GRILLA IDEAL",
956     command=mostrar_grilla,
957     relief=tk.GROOVE,
958     borderwidth=10,
959     bg="#8c75ad",
960     fg="BLACK",
961     font=fuente_texto,
962 )
963 boton1.place(x=830, y=850)
964
965 ventana.mainloop()
```

- **Ocultar_menu()** como el nombre lo describe, la función oculta todos los widgets de la ventana

```

93 def ocultar_menu():
94     # oculta todos los widgets de la ventana
95
96     for widget in ventana.winfo_children():
97         widget.place_forget()

```

- **Mostrar_menu()**, si se selecciona el botón “trabajo con datos almacenados”, primero se ocultan todos los widgets de la ventana excepto el logo, luego se muestran las dos opciones. trabajar con el experimento de grilla expuesta a humedad o grilla expuesta a fricción. También un botón para volver a la ventana principal

```

636 def mostrar_menu():
637     ocultar_menu()
638     labellogo.place(x=700, y=80)
639     fuente_texto = font.Font(family="Helvetica", size=16, weight="bold")
640     boton2 = tk.Button(
641         ventana, text="2: Humedad", command=mostrar_humedad,
642         relief=tk.GROOVE, borderwidth=10, bg="#d97675", fg="BLACK", font=fuente_texto,
643     )
644     boton2.place(x=860, y=575)
645     boton3 = tk.Button(
646         ventana, text="3: Flexion", command=mostrar_friccion,
647         relief=tk.GROOVE, borderwidth=10, bg="#f1c47c", fg="BLACK", font=fuente_texto,
648     )
649     boton3.place(x=870, y=700)
650     botonvolver = tk.Button(
651         ventana, text="VOLVER", command=mostrar_inicio,
652         relief=tk.GROOVE, borderwidth=5, bg="#d97675", fg="BLACK", font=fuente_texto,
653     )
654     botonvolver.place(x=10, y=10)

```

- **Mostrar_1** muestra una pantalla predeterminada.

```

47 def mostrar_1():
48     global label1
49     ocultar_menu()
50     label1 = tk.Label(ventana, background="gray")
51     label1.place(x=50, y=100, width=800, height=800)
52     limpiarlabels()
53

```

Mostrar humedad y flexión:

En este espacio inician las principales funcionalidades del programa, el cambio de dimensión de una grilla expuesta a humedad y flexión. Ambas trabajan de la misma manera, pero la principal diferencia está en los parámetros al llamar a las funciones “mostrar_imagen” y “on_canvas_click”

- Primero se muestra unos labels predeterminados a través de la función “mostrar_1”
- Se ejecuta la función limpiar labels para vaciar las listas con puntos (si es primera vez que se inicia esta función dichas listas estarán vacías)

```
414 def mostrar_humedad():
415     mostrar_1()
416     limpiarlabels()
```

- Se establecen variables globales
- Labels (títulos) y botones
 - Botón “VOLVER”. Vuelve a la pantalla principal

```
417 global boton_mostrar_imagen, cargar_Carpeta, Boton_calcularDistancias
418 botonMostarMenu = tk.Button(
419     ventana, text="VOLVER", command=mostrar_menu, fg="BLACK", bg="#EAFEDF"
420 )
421 botonMostarMenu.place(x=10, y=10, width=100, height=23)
422
```

- Botón “cargar” y respectivo label con título, carga la carpeta y primera imagen de dicha carpeta

```
423 titulo3 = tk.Label(
424     ventana,
425     text="Seleccionar carpeta",
426     fg="white",
427     bg="#212f4e",
428     font=("Arial", 12),
429 )
430 titulo3.place(x=920, y=70)
431 cargar_Carpeta = tk.Button(
432     ventana, text="Cargar", command=mostrar_primera_imagen, fg="BLACK"
433 )
434 cargar_Carpeta.place(x=920, y=100)
435
```

- Botón para mostrar la primera imagen

```
436 titulo4 = tk.Label(
437     ventana,
438     text="Cargar Primera Imagen",
439     fg="white",
440     bg="#212f4e",
441     font=("Arial", 12),
442 )
443 titulo4.place(x=920, y=170)
444
445 # Botón para mostrar la imagen
446 boton_mostrar_imagen = tk.Button(
447     ventana,
448     text="Cargar Imagen",
449     command=lambda: mostrar_imagen(72, 166, 3000),
450     borderwidth=3,
451 )
452
453 boton_mostrar_imagen.place(x=920, y=200)
454 boton_mostrar_imagen.config(state="disabled")
```

- Botón “iniciar análisis de Dimensión” dicho botón ejecuta el análisis de dimensión y calcula las distancias de los puntos seleccionados

```

456     titulo4 = tk.Label(
457         ventana,
458         text="iniciar análisis de Dimensión",
459         fg="white",
460         bg="#212f4e",
461         font=("Arial", 12),
462     )
463     titulo4.place(x=920, y=270)
464     Boton_calcularDistancias = tk.Button(
465         ventana, text="Iniciar", command=cal_distancias
466     )
467     Boton_calcularDistancias.place(x=920, y=300)
468     Boton_calcularDistancias.config(state="disabled")
469
470     # Etiqueta para mostrar los pixels entre los puntos seleccionados
471     resultadopixel = tk.Label(ventana, text="")
472     resultadopixel.place(x=250, y=920)

```

- Cuando ocurre el evento de hacer clic en uno de los puntos se ejecuta la función “on_canvas_click”

```

474     label1.bind(
475         "<Button-1>",
476         lambda event: on_canvas_click(
477             event,
478             imagen1,
479             label1,
480             resultadopixel,
481             selected_point_indices,
482             selected_points,
483             points,
484             10,
485             10,
486         ),
487     )

```

- Label con título:

```

489     titulo2 = tk.Label(
490         ventana,
491         text="Seleccionar Puntos A Analizar",
492         fg="white",
493         bg="#212f4e",
494         font=("Arial", 12),
495     )
496     titulo2.place(x=50, y=70)
497

```


Mostrar Flexión

Como se ha dicho es igual al apartado de “mostrar humedad”, solo cambian los parámetros de las funciones “mostrar_imagen” y “on_canvas_click”.

```
552 def mostrar_friccion():
553     mostrar_1()
554     limpiarlabels()
555     global boton_mostrar_imagen, cargar_Carpeta, Boton_calcularDistancias
556     botonMostarMenu = tk.Button(
557         ventana, text="VOLVER", command=mostrar_menu, fg="BLACK", bg="#EAFEDE"
558     )
559     botonMostarMenu.place(x=10, y=10, width=100, height=23)
560
561     titulo2 = tk.Label(
562         ventana,
563         text="Seleccionar Puntos A Analizar",
564         fg="white",
565         bg="#212f4e",
566         font=("Arial", 12),
567     )
568     titulo2.place(x=50, y=70)
569
570     titulo3 = tk.Label(
571         ventana,
572         text="Seleccionar carpeta",
573         fg="white",
574         bg="#212f4e",
575         font=("Arial", 12),
576     )
577     titulo3.place(x=920, y=70)
578     cargar_Carpeta = tk.Button(
579         ventana, text="Cargar", command=mostrar_primera_imagen, fg="BLACK"
580     )
581     cargar_Carpeta.place(x=920, y=100)
582
583     titulo4 = tk.Label(
584         ventana,
585         text="Cargar Primera Imagen",
586         fg="white",
587         bg="#212f4e",
588         font=("Arial", 12),
589     )
590     titulo4.place(x=920, y=170)
591
592     # Botón para mostrar la imagen
593     boton_mostrar_imagen = tk.Button(
594         ventana, text="Mostrar Imagen", command=lambda: mostrar_imagen(20.42, 160, 400)
595     )
596     boton_mostrar_imagen.place(x=920, y=200)
597     boton_mostrar_imagen.config(state="disabled")
598
599     titulo4 = tk.Label(
600         ventana,
601         text="iniciar análisis de Dimensión",
602         fg="white",
603         bg="#212f4e",
604         font=("Arial", 12),
605     )
606     titulo4.place(x=920, y=270)
607     Boton_calcularDistancias = tk.Button(
608         ventana, text="Iniciar", command=cal_distancias
609     )
610     Boton_calcularDistancias.place(x=920, y=300)
611     Boton_calcularDistancias.config(state="disabled")
612
613     # Etiqueta para mostrar los pixels entre los puntos seleccionados
614     resultadopixel = tk.Label(
615         ventana, text="", fg="white", bg="#212f4e", font=("Arial", 12)
616     )
```

```
612
613     # Etiqueta para mostrar los pixels entre los puntos seleccionados
614     resultadopixel = tk.Label(
615         ventana, text="", fg="white", bg="#212f4e", font=("Arial", 12)
616     )
617     resultadopixel.place(x=250, y=920)
618
619     label1.bind(
620         "<Button-1>",
621         lambda event: on_canvas_click(
622             event,
623             imagen1,
624             label1,
625             resultadopixel,
626             selected_point_indices,
627             selected_points,
628             points,
629             152,
630             190,
631         ),
632     )
633
```

Funciones para la detección de esquinas

Dichas funciones son aquellas que se usaron anteriormente durante la variación de dimensión

Mostrar_primera_imagen () crea una ventana emergente de la cual se obtendrá el path de la carpeta y la lista de archivos de esta, así también se le asigna a “primera_imagen” el path de la primera fotografía tomada

```
55 def mostrar_primera_imagen():
56     global primera_imagen, carpeta_seleccionada
57     boton_mostrar_imagen.config(state="normal")
58     # Obtener la carpeta seleccionada por el usuario
59     carpeta_seleccionada = filedialog.askdirectory()
60
61     if not carpeta_seleccionada:
62         return # El usuario ha cancelado la selección
63
64     # Obtener la lista de archivos en la carpeta seleccionada
65     lista_archivos = os.listdir(carpeta_seleccionada)
66
67     # Encontrar la primera imagen en la carpeta
68     primera_imagen = None
69     for archivo in lista_archivos:
70         if archivo.endswith(".jpg") or archivo.endswith(".png"):
71             primera_imagen = archivo
72             break
73     else:
74         print("No se encontraron imágenes en la carpeta seleccionada.")
```

Mostrar_imagen Muestra en el label la imagen con los puntos ya detectados y dibujados en la imagen.

1. Carga la imagen
2. Cambia las dimensiones de la imagen (según el “porcentaje” dicho en el argumento)
3. Pasa a escala de grises la imagen
4. Detecta y dibuja las intersecciones mediante el algoritmo de Harris, en el argumento se le pasa la imagen umbralizada y Se actualiza el label

```
310 def mostrar_imagen(porcentaje, valorumbral, cantidad_puntos):
311     global imagen1
312     limpiarlabels()
313     label1.config(text="", bg="gray")
314     Boton_calcularDistancias.config(state="normal")
315
316     imagen1 = cv2.imread(os.path.join(carpeta_seleccionada, primera_imagen))
317
318     alto_original, ancho_original = imagen1.shape[:2]
319     nuevo_ancho = int(ancho_original * porcentaje / 100)
320     nuevo_alto = int(alto_original * porcentaje / 100)
321     imagen1 = cv2.resize(imagen1, (nuevo_ancho, nuevo_alto))
322
323     gray = cv2.cvtColor(imagen1, cv2.COLOR_BGR2GRAY)
324
325     buscar_dibujar_puntos_harris(
326         umbralizacion(
327             gray,
328             valorumbral,
329         ),
330         imagen1,
331         cantidad_puntos,
332     )
333
334     update_image(label1, imagen1)
```

Buscar_dibujar_puntos_harris, mediante el algoritmo de Harris se detectan las intersecciones o “corners” de la imagen, y se eligen las primeras x cantidad mas posibles de que sean intersecciones.

El algoritmo de Harris le asigna a cada píxel un valor según qué tan probable es que ese píxel sea una esquina, es por eso que se eligen los primeros x pixeles ya que son los que tienen mayor probabilidad de ser esquinas y en esas coordenadas se dibuja un círculo rojo

```
155 def buscar_dibujar_puntos_harris(gray, original, cantidad_puntos):
156     # Aplicar la detección de esquinas usando el algoritmo de Harris
157     dst = cv2.cornerHarris(gray, 2, 3, 0.04)
158     # Normalizar y aplicar un umbral para obtener las esquinas
159     dst_norm = cv2.normalize(dst, None, 0, 255, cv2.NORM_MINMAX)
160     threshold = 0.01 * dst_norm.max()
161     corners = np.column_stack(
162         np.where(dst_norm > threshold)
163     ) # Obtener coordenadas como una lista de tuplas
164     # Ordenar las esquinas por respuesta de Harris y tomar las primeras x = "cantidad de puntos"
165     corners = sorted(corners, key=lambda x: dst_norm[x[0], x[1]], reverse=True)[
166         :cantidad_puntos
167     ] # Supresión no máxima
168     valid_corners = []
169     for i, (y, x) in enumerate(corners):
170         if all(
171             np.linalg.norm(np.array([y, x]) - np.array([yc, xc])) > 10
172             for yc, xc in valid_corners
173         ): valid_corners.append((y, x))
174     # Dibujar los puntos en la imagen
175     for y, x in valid_corners:
176         cv2.circle(original, (x, y), 5, (0, 0, 255), -1)
177         points.append((x, y))
```

On_canvas_click Dicha función se acciona cuando se produce el evento de hacer clic en el label. Tiene por objetivo seleccionar dos puntos y calcular la distancia entre ellos.

1. Se compara si el lugar seleccionado corresponde a uno de los puntos ya analizados
2. Una vez que se confirma que el punto seleccionado corresponde a una de las intersecciones ya analizadas este cambia el color del punto rojo por verde
3. Cuando se seleccionan 2 puntos, se muestra la distancia entre ellos

```
213 def on_canvas_click(
214     event, imagencita, labelA, labelB, indices, puntos_seleccionados, puntos, xs, ys
215 ):
216     # por algun motivo las coordenadas que entrega el evento son x
217     # pixeles mas que las coordenadas de la imagen, por eso se le resta x
218     global punto1, punto2
219     x, y = event.x - xs, event.y - ys
220     for i, (px, py) in enumerate(puntos):
221         if abs(x - px) < 5 and abs(y - py) < 5:
222             indices.append(i)
223             puntos_seleccionados.append((px, py))
224             cv2.circle(imagencita, (px, py), 5, (0, 255, 0), -1)
225     if len(indices) == 2:
226         calculate_distance(puntos_seleccionados, labelB)
227         punto1 = puntos_seleccionados[0]
228         punto2 = puntos_seleccionados[1]
229         puntos_seleccionados.clear()
230         indices.clear()
231     # Actualiza la imagen en el widget de etiqueta
232     update_image(labelA, imagencita)
233
```

On_canvas_click2 funciona igual que “on_canvas_click” pero para el caso de la grilla.

```
190 def on_canvas_click2(event, imagencita, labelA, labelB):
191     # por algun motivo las coordenadas que entrega el evento son 25 pixeles mas que las coordenadas de
192     # la imagen, por eso se le resta 25
193     # (solo en el caso de la grilla ideal)
194     x, y = event.x - 25, event.y - 25
195     print(f"x: {x} y:{y}")
196
197     for i, (px, py) in enumerate(points):
198         if abs(x - px) < 5 and abs(y - py) < 5:
199             selected_point_indices.append(i)
200             selected_points.append((px, py))
201             cv2.circle(imagencita, (px, py), 5, (0, 255, 0), -1)
202
203     if len(selected_point_indices) == 2:
204         calculate_distance(selected_points, labelB)
205         selected_points.clear()
206         selected_point_indices.clear()
207
208     # Actualiza la imagen en el widget de etiqueta
209     update_image(labelA, imagencita)
```

Calculate_distance Calcula y muestra la distancia entre dos puntos

```
180 v def calculate_distance(selected_points, labelactualizado):
181 v     if len(selected_points) == 2:
182         point1 = selected_points[0]
183         point2 = selected_points[1]
184         distance = np.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]) ** 2)
185 v         labelactualizado.config(
186             text=f"Distancia entre los puntos seleccionados: {distance:.2f} píxeles"
187         )
```

Analizar_distancia_entre_puntos. Dicha función tiene como objetivo analizar la variación de distancia entre dos puntos de una grilla, la cual fue expuesta a humedad o flexión, se analiza cada una de las fotos y se guarda la distancia entre los puntos seleccionados, retorna la lista de con las distancias.

```
325 def analizar_distancia_entre_puntos(  
326     carpeta, punto1, punto2, valorhumbral, cantidad_puntos  
327 ):  
328     # Lista para almacenar las distancias calculadas  
329     distancias = []  
330     # Obtener la lista de archivos en la carpeta seleccionada  
331     lista_archivos = os.listdir(carpeta)  
332     # Iterar sobre los archivos en la carpeta  
333     for archivo in lista_archivos:  
334         # Cargar la imagen  
335         imagen = cv2.imread(os.path.join(carpeta, archivo))  
336         alto_original, ancho_original = imagen.shape[:2]  
337         nuevo_ancho = int(ancho_original * 20.42 / 100)  
338         nuevo_alto = int(alto_original * 20.42 / 100)  
339         imagen = cv2.resize(imagen, (nuevo_ancho, nuevo_alto))  
340         # Convertir la imagen a escala de grises  
341         grises = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)  
342         # Aplicar la umbralización  
343         umbral = umbralizacion(grises, valorhumbral)  
344         # Buscar y dibujar los puntos de Harris  
345         buscar_dibujar_puntos_harris(umbral, imagen, cantidad_puntos)  
346         margen = 10  
347         for px, py in points:  
348             if (  
349                 punto1[0] + margen > px  
350                 and punto1[0] - margen < px  
351                 and punto1[1] + margen > py  
352                 and punto2[1] - margen < py  
353             ):  
354                 for px2, py2 in points:  
355                     for px2, py2 in points:  
356                         if (  
357                             punto2[0] + margen > px2  
358                             and punto2[0] - margen < px2  
359                             and punto2[1] + margen > py2  
360                             and punto2[1] - margen < py2  
361                         ):  
362                             # Calcular la distancia entre los puntos  
363                             distancia = np.sqrt((px - px2) ** 2 + (py - py2) ** 2)  
364                             # Agregar la distancia a la lista  
365                             distancias.append(distancia)  
366                             punto1 = (px, py)  
367                             punto2 = (px2, py2)  
368                             break  
369                     break  
370             points.clear()  
371     return distancias  
372
```

Cal_distancia, crea un gráfico con las distancias anteriormente guardadas, se importan las librerías de matplotlib para generar el gráfico.

```
458 def cal_distancias():
459     distancias = analizar_distancia_entre_puntos(
460         carpeta_seleccionada, punto1, punto2, 160, 400
461     )
462     import matplotlib.pyplot as plt
463     from matplotlib.ticker import MaxNLocator
464     from matplotlib.backends.backend_tkagg import (
465         FigureCanvasTkAgg,
466         NavigationToolbar2Tk,
467     )
468     from matplotlib.figure import Figure
469     import tkinter as tk
470
471     # Crear una figura y un objeto Axes
472     fig = Figure(figsize=(5, 4), dpi=100)
473     ax = fig.add_subplot(111)
474
475     # Configurar el objeto Axes
476     ax.xaxis.set_major_locator(MaxNLocator(integer=True))
477     ax.plot(
478         range(len(distancias)),
479         distancias,
480         color="blue",
481         linewidth=2,
482         linestyle="solid",
483         marker="o",
484         label="Distancias",
485     )
```

```
486     # Crear una ventana Tkinter
487     root = tk.Tk()
488     root.geometry("+1200+150") # Cambiar la posición de la ventana a (500, 500)
489     # Crear un canvas de matplotlib en la ventana Tkinter
490     canvas = FigureCanvasTkAgg(fig, master=root)
491     canvas.draw()
492     canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
493     # Añadir una barra de herramientas de navegación
494     toolbar = NavigationToolbar2Tk(canvas, root)
495     toolbar.update()
496     canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
497     # Mostrar la ventana
498     tk.mainloop()
```

Análisis de grilla ideal

Esta es otra de las funcionalidades principales del programa, la creación de detección de esquinas en una “grilla ideal” con dimensiones dichas por el usuario

mostrar_grilla() función que muestra los widgets una vez que se presione el botón “humedad” del menú principal.

- primero llama a la función **mostrar_1()** para mostrar los labels predeterminados
- crea spinbox para definir las dimensiones de la grilla creada
- se define el botón “crear” que al presionarse llamará a la función **crear_grilla**,
- se usa el método “.bind()” para que una vez que ocurra el evento de hacer click en el label de tal modo que se llame a la función **on_canvas_click2**

```
229 def mostrar_grilla():
230     global Sancho, Slargo, B_aplicarHarris
231     mostrar_1()
232     limpiarlabels()
233
234     botonMostarMenu = tk.Button(
235         ventana, text="VOLVER", command=mostrar_inicio, bg="#EAFEDE", fg="BLACK"
236     )
237     botonMostarMenu.place(x=10, y=10, width=100, height=23)
238     label2 = tk.Label(ventana, background="gray")
239     label2.place(x=1100, y=100, width=800, height=800)
240
241     L_dimenciones = tk.Label(
242         ventana,
243         text="Dimenciones de la grilla",
244         fg="white",
245         bg="#212f4e",
246         font=("Arial", 12),
247     )
248     L_dimenciones.place(x=300, y=910)
249     Sancho = tk.Scale(ventana, from_=1, to=15, orient="horizontal", bg="#EAFEDE")
250     Sancho.place(x=300, y=950)
251     Slargo = tk.Scale(ventana, from_=1, to=15, orient="horizontal", bg="#EAFEDE")
252     Slargo.place(x=430, y=950)
253     botoncrear = tk.Button(ventana, text="CREAR", command=crear_grilla)
254     botoncrear.place(x=550, y=950, width=100, height=23)
255     resultadopixel = tk.Label(ventana, text="")
256     resultadopixel.place(x=800, y=950)
257     titulo = tk.Label(
258         ventana,
259         text="Grilla ideal Creada por el Usuario",
260         fg="white",
261         bg="#212f4e",
262         font=("Arial", 12),
263     )
264     titulo.place(x=50, y=70)
265     titulo = tk.Label(
266         ventana,
267         text="Algoritmo de Harris Aplicado",
268         fg="white",
269         bg="#212f4e",
270         font=("Arial", 12),
271     )
```

```

272 titulo.place(x=1100, y=70)
273 L_aplicar = tk.Label(
274     ventana,
275     text="Aplicar Algoritmo de Harris",
276     fg="white",
277     bg="#212f4e",
278     font=("Arial", 12),
279 )
280 L_aplicar.place(x=1300, y=910)
281 B_aplicarHarris = tk.Button(
282     ventana, text="Aplicar!", command=lambda: aplicar_harris(label2)
283 )
284 B_aplicarHarris.place(x=1300, y=950)
285 B_aplicarHarris.config(state="disabled")
286
287 label2.bind(
288     "<Button-1>",
289     lambda event: on_canvas_click2(event, grillacv, label2, resultadopixel),
290 )

```

crear_grilla() función la cual crea una grilla, dibujándola con las dimensiones previamente dichas por el usuario y las medidas que tendrá cada cuadrado serán de 50 pixeles también se define el tamaño que tendrá dicha imagen.

- Se crea la imagen de color blanco con dicho tamaño, luego se dibujan las líneas horizontales y verticales

```

98 def crear_grilla():
99     limpiar_labels()
100     B_aplicarHarris.config(state="normal")
101     # Tamaño de celda y tamaño de la imagen
102     ancho_celda = 50
103     alto_celda = 50
104     ancho_imagen = (int(Sancho.get())) * ancho_celda # Ajuste en el ancho
105     alto_imagen = (int(Slargo.get())) * alto_celda # Ajuste en el alto
106     # Crea una nueva imagen
107     imagengrilla = Image.new("RGB", (ancho_imagen, alto_imagen), color="white")
108     dibujo = ImageDraw.Draw(imagengrilla)
109     # Dibuja la grilla
110     for i in range(0, ancho_imagen, ancho_celda):
111         dibujo.line([(i, 0), (i, alto_imagen)], fill="black")
112     for j in range(0, alto_imagen, alto_celda):
113         dibujo.line([(0, j), (ancho_imagen, j)], fill="black")
114     # Dibuja las líneas adicionales para el borde derecho e inferior
115     dibujo.line(
116         [(ancho_imagen - 1, 0), (ancho_imagen - 1, alto_imagen)], fill="black"
117     ) # Borde derecho
118     dibujo.line(
119         [(0, alto_imagen - 1), (ancho_imagen, alto_imagen - 1)], fill="black"
120     ) # Borde inferior
121
122     # Guarda la imagen generada
123     imagengrilla.save("grilla.png")
124     imagenideal = ImageTk.PhotoImage(imagengrilla)
125     # MUESTRA EN EL LABEL
126     label1.config(image=imagenideal)
127     label1.image = imagenideal
128

```


Se llama a la imagen a través del path y se le asigna a una variable, luego esta se pasa a escala de grises y se llama a la función `buscar_dibujar_puntos_harris()`, se le pasa como argumento la imagen umbralizada y la imagen original para aplicar el algoritmo de Harris y así detectar las esquinas.

```
130 def aplicar_harris(label2):
131     global grillacv
132     # Convierte la imagen a escala de grises
133
134     grillacv = cv2.imread("C:/Users/jerpa/OneDrive/Escritorio/grilla.png")
135
136     image_path = "C:/Users/jerpa/OneDrive/Escritorio/grilla.png"
137
138     grillacv = resize_image(image_path, target_size)
139
140     # Guardar la nueva imagen
141     grillacv.save("C:/Users/jerpa/OneDrive/Escritorio/grilla.png")
142
143     grillacv = cv2.imread("C:/Users/jerpa/OneDrive/Escritorio/grilla.png")
144
145     gray2 = cv2.cvtColor(grillacv, cv2.COLOR_BGR2GRAY)
146     buscar_dibujar_puntos_harris(umbralizacion(gray2, 166), grillacv, 2000)
147     update_image(label2, grillacv)
148
```

Mostrar_inicio Muestra el menú de inicio con sus respectivos widgets.

```
600 def mostrar_inicio():
601     ocultar_menu()
602     labellogo.place(x=700, y=80)
603     botonA = tk.Button(
604         ventana,
605         text="TRABAJO CON DATOS ALMACENADOS",
606         command=mostrar_menu,
607         relief=tk.GROOVE,
608         borderwidth=10,
609         bg="#d97675",
610         fg="BLACK",
611         font=fuente_texto,
612     )
613     botonA.place(x=690, y=550)
614     botonB = tk.Button(
615         ventana,
616         text="NUEVO PROYECTO",
617         command=mostrar_menu2,
618         relief=tk.GROOVE,
619         borderwidth=10,
620         bg="#d97675",
621         fg="BLACK",
622         font=fuente_texto,
623     )
```

```
624     botonB.place(x=820, y=710)
625     boton1 = tk.Button(
626         ventana,
627         text="1. GRILLA IDEAL",
628         command=mostrar_grilla,
629         relief=tk.GROOVE,
630         borderwidth=10,
631         bg="#8c75ad",
632         fg="BLACK",
633         font=fuente_texto,
634     )
635     boton1.place(x=830, y=850)
```

Nuevo Proyecto

esta es otra de las funcionalidades del programa, el objetivo de esta función es para tomar fotos cada x cantidad de tiempo definido por el usuario, para su posterior análisis.

Camara Este fragmento de código define una función llamada camara que parece estar diseñada para iniciar una cámara utilizando OpenCV.

```
641 def camara():
642     global capture
643     capture = cv2.VideoCapture(0)
644     iniciarcamara()
645     botondetenercam.config(state="normal")
646     botoncamara.config(state="disabled")
647     if "button_start" in globals():
648         button_start.config(state="normal")
649
```

iniciarcamara Esta función captura y muestra frames de video de una cámara en tiempo real en una interfaz de usuario de Tkinter.

```
651 def iniciarcamara():
652     global capture, variable_control
653     variable_control = 1
654
655     if capture is not None:
656         ret, frame = capture.read()
657         if ret:
658             frame = imutils.resize(frame)
659             frame = imutils.resize(frame)
660             ImagenCamara = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
661             im = Image.fromarray(ImagenCamara)
662             img = ImageTk.PhotoImage(image=im)
663             labelIZQ.configure(image=img)
664             labelIZQ.image = img
665             labelIZQ.after(1, iniciarcamara)
666
667             # Actualiza la interfaz gráfica
668             labelIZQ.update_idletasks()
669         else:
670             labelIZQ.image = ""
671             capture.release()
```

Mostrar_2 esta función configura y muestra el menú del proyecto de monitoreo de cámara.

```
809 # muestra menu nuevo proyecto
810 def mostrar_2():
811     ocultar_menu()
812     global labelIZQ, entry_interval, botoncamara, botondetenercam, C_pathFotos
813     botonvolver = tk.Button(
814         ventana, text="VOLVER", command=mostrar_menu2, bg="#EAFEDE", fg="BLACK"
815     )
816     botonvolver.place(x=10, y=10)
817     labelIZQ = tk.Label(ventana, bg="gray")
818     labelIZQ.place(x=50, y=100, width=700, height=700)
819     texto1 = tk.Label(
820         ventana,
821         text="Monitoreo De Camara",
822         bg="#212f4e",
823         fg="WHITE",
824         font=("Helvetica", 11, "bold"),
825     )
826     texto1.place(x=50, y=70)
827     botoncamara = tk.Button(ventana, text="INICIAR CAMARA", command=camara)
828     botoncamara.place(x=50, y=850)
829     botondetenercam = tk.Button(
830         ventana, text="DETENER CAMARA", command=detener_camara, fg="BLACK"
831     )
832     botondetenercam.place(x=300, y=850)
833     botondetenercam.config(state="disabled")
834
835     # Etiqueta y campo de entrada para el intervalo de tiempo
836     label_interval = tk.Label(ventana, text="Intervalo de tiempo (segundos):")
837     label_interval.place(x=800, y=100)
838     entry_interval = tk.Entry(ventana)
839     entry_interval.place(x=800, y=130)
840
841     # caja de texto con los path de las fotos que se estan tomando
842     C_pathFotos = tk.Text(ventana)
843     C_pathFotos.place(x=1100, y=100, width=700, height=700)
844     C_pathFotos.insert(tk.END, "Fotos Tomadas:\n\n")
```

actualizarCajaPath() actualiza la caja de texto con los path de las fotos que se están tomando

```
848 def actualizarCajaPath(pathfotos):
849     C_pathFotos.delete("2.0", tk.END)
850     C_pathFotos.insert(tk.END, "Path de las fotos que se estan tomando:\n")
851     for root, dirs, files in os.walk(pathfotos):
852         for filename in files:
853             C_pathFotos.insert(tk.END, os.path.join(root, filename) + "\n")
854     C_pathFotos.see("end")
855     C_pathFotos.update_idletasks()
```

Star_capture() / *star_capture2* diseñada para iniciar un proceso de captura de imágenes en un intervalo de tiempo especificado y guardarlas en una carpeta específica.

```
763 def start_capture():
764     C_pathFotos.delete("2.0", tk.END)
765     output_folder = "C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad"
766     interval = int(entry_interval.get())
767     hours = int(entry_hours.get())
768     minutes = int(entry_minutes.get())
769     seconds = int(entry_seconds.get())
770     duration = hours * 3600 + minutes * 60 + seconds
771
772     threading.Thread(
773         target=capture_and_save_phototime, args=(output_folder, interval, duration)
774     ).start()
```

```
756 def start_capture2():
757     global termina
758     C_pathFotos.delete("2.0", tk.END)
759     button_start.config(state="disabled")
760     botondetenercam.config(state="disabled")
761     detenerCapturas.config(state="normal")
762     output_folder2 = "C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Flexion"
763     interval = int(entry_interval.get())
764     termina = True
765     threading.Thread(
766         target=capture_and_save_phototime_flexion, args=(output_folder2, interval)
767     ).start()
```

Capture_and_save_phototime_flexion esta función captura imágenes de una cámara a intervalos regulares y las guarda en una carpeta específica. Cada foto se nombra con un timestamp para asegurar que cada foto tenga un nombre único. La ruta de cada foto se inserta en una caja de texto en la interfaz de usuario.

```
733 def capture_and_save_phototime_flexion(output_folder, interval):
734     global termina
735     # Crear una nueva carpeta con la fecha y hora actual como nombre
736     timestamp = time.strftime("%Y%m%d-%H%M%S")
737     folder_name = f"Photos_{timestamp}"
738     folder_path = os.path.join(output_folder, folder_name)
739     os.makedirs(folder_path)
740     while termina:
741         # Capturar una foto utilizando la cámara
742         camera = capture
743         return_value, frame = camera.read()
744         # Generar un nombre de archivo único basado en la fecha y hora actual
745         timestamp = time.strftime("%Y%m%d-%H%M%S")
746         filename = f"{timestamp}.jpg"
747         # Guardar la foto en la carpeta de salida especificada
748         output_path = os.path.join(folder_path, filename)
749         cv2.imwrite(output_path, frame)
750         C_pathFotos.insert(tk.END, output_path + "\n")
751         # Esperar el intervalo de tiempo especificado antes de tomar la siguiente foto
752         time.sleep(interval)
```

Capture_and_save_phototime captura imágenes de una cámara a intervalos regulares durante un período de tiempo especificado y las guarda en una carpeta específica. También muestra una barra de progreso en la interfaz de usuario para indicar el progreso de la captura de imágenes.

```
683 def capture_and_save_phototime(output_folder, interval, duration):
684     L_infoProgreso = tk.Label(
685         ventana,
686         text="Progreso captura de fotos",
687         bg="#212f4e",
688         fg="WHITE",
689         font=("Arial", 11, "bold"),
690     )
691     L_infoProgreso.place(x=800, y=350)
692     barraprogreso = ttk.Progressbar(
693         ventana, orient="horizontal", length=270, mode="determinate")
694     barraprogreso.place(x=800, y=378)
695     threading.Thread(target=avanzar_barra, args=(duration, barraprogreso)).start()
696     start_time = time.time()
697     # Crear una nueva carpeta con la fecha y hora actual como nombre
698     timestamp = time.strftime("%Y%m%d-%H%M%S")
699     folder_name = f"Photos_{timestamp}"
700     folder_path = os.path.join(output_folder, folder_name)
701     os.makedirs(folder_path)
702     while True:
703         # Comprobar si ha pasado el tiempo de duración
704         if time.time() - start_time >= duration:
705             break
706         # Capturar una foto utilizando la cámara
707         camera = capture
708         return_value, frame = camera.read()
709         # Generar un nombre de archivo único basado en la fecha y hora actual
710         timestamp = time.strftime("%Y%m%d-%H%M%S")
711         filename = f"{timestamp}.jpg"
712         # Guardar la foto en la carpeta de salida especificada
713         output_path = os.path.join(folder_path, filename)
714         cv2.imwrite(output_path, frame)
715         C_pathFotos.insert(tk.END, output_path + "\n")
716         # Esperar el intervalo de tiempo especificado antes de tomar la siguiente foto
717         time.sleep(interval)
718     messagebox.showinfo("Captura finalizada", "La captura de fotos ha finalizado.")
```

Avanzar_barra Función para avanzar barra de progreso

```
674 def avanzar_barra(tiempo, barraprogreso):
675     avance = 100 / tiempo
676     while tiempo > 0:
677         barraprogreso.step(avance)
678         tiempo -= 1
679         time.sleep(1)
680     barraprogreso.stop()
```

Detener_camara() función para detener Cámara

```
779 def detener_camara():
780     capture.release()
781     print("camara detenida")
782     botoncamara.config(state="normal")
783     botondetenercam.config(state="disabled")
784     button_start.config(state="disabled")
```

Grabar humedad Permite al usuario especificar la duración de la captura en horas, minutos y segundos, y proporciona un botón para iniciar la captura.

```
836 def grabar_humedad():
837     mostrar_2()
838     global entry_hours, entry_minutes, entry_seconds
839     # Etiqueta y campos de entrada para la duración de captura
840     label_duration = tk.Label(ventana, text="Duración de captura:")
841     label_duration.place(x=800, y=200)
842
843     frame_duration = tk.Frame(ventana)
844     frame_duration.place(x=800, y=230)
845
846     entry_hours = tk.Entry(frame_duration, width=2)
847     entry_hours.pack(side=tk.LEFT)
848     label_hours = tk.Label(frame_duration, text="horas")
849     label_hours.pack(side=tk.LEFT)
850
851     entry_minutes = tk.Entry(frame_duration, width=2)
852     entry_minutes.pack(side=tk.LEFT)
853     label_minutes = tk.Label(frame_duration, text="minutos")
854     label_minutes.pack(side=tk.LEFT)
855
856     entry_seconds = tk.Entry(frame_duration, width=2)
857     entry_seconds.pack(side=tk.LEFT)
858     label_seconds = tk.Label(frame_duration, text="segundos")
859     label_seconds.pack(side=tk.LEFT)
860     # Botón para iniciar la captura de fotos
861     button_start = tk.Button(ventana, text="Iniciar Captura", command=start_capture)
862     button_start.place(x=800, y=300)
```

Grabar humedad Permite al usuario especificar cada cuanto se tomarán las fotos.

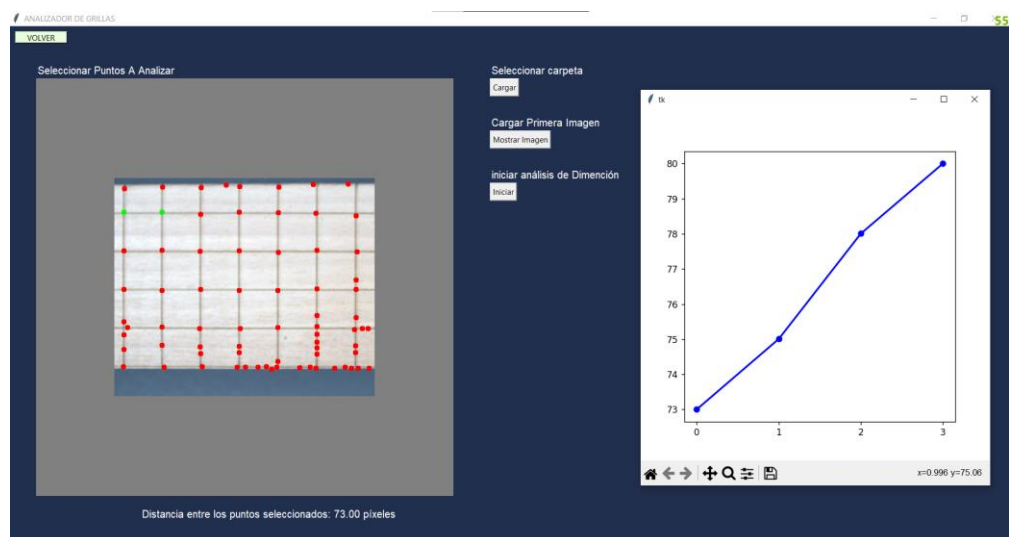
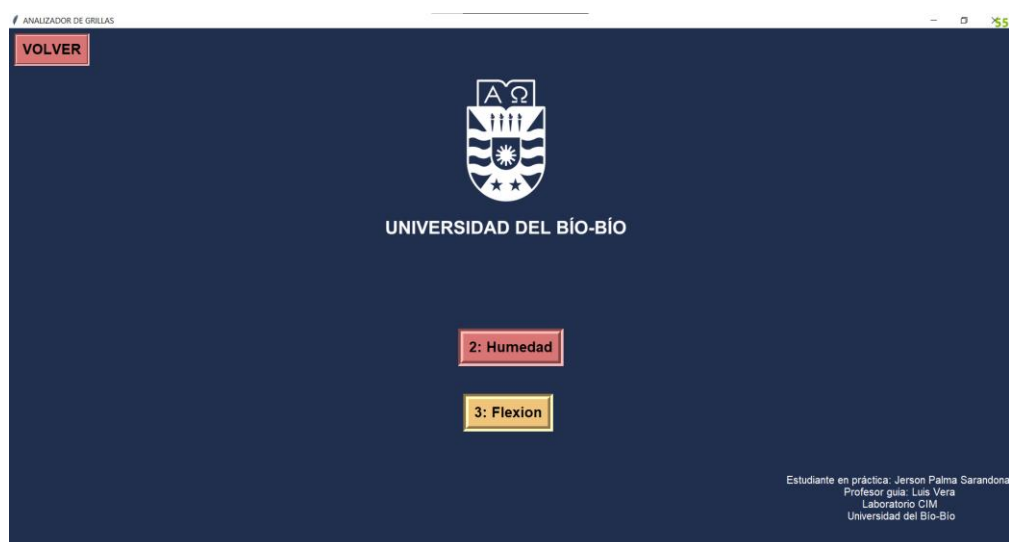
```
865 def grabar_friccion():
866     mostrar_2()
867     global detenerCapturas, button_start
868     button_start = tk.Button(ventana, text="Iniciar Captura", command=start_capture2)
869     button_start.place(x=800, y=300)
870     detenerCapturas = tk.Button(
871         ventana, text="DETENER CAPTURAS", command=v_control, fg="BLACK"
872     )
873     detenerCapturas.place(x=800, y=350)
874     detenerCapturas.config(state="disabled")
875     button_start.config(state="disabled")
876
```

Mostrar_menu2 muestra el menú principal del programa.

```
878 def mostrar_menu2():
879     ocultar_menu()
880     # apaga la camara una vez que vuelve al menu
881     if "variable_control" in globals():
882         capture.release()
883     labellogo.place(x=700, y=80)
884     boton2 = tk.Button(
885         ventana,
886         text="2: Humedad",
887         command=grabar_humedad,
888         relief=tk.GROOVE,
889         borderwidth=10,
890         bg="#d97675",
891         fg="BLACK",
892         font=fuente_texto,
893     )
894     boton2.place(x=860, y=575)
895     boton3 = tk.Button(
896         ventana,
897         text="3: Flexion",
898         command=grabar_friccion,
899         relief=tk.GROOVE,
900         borderwidth=10,
901         bg="#f1c47c",
902         fg="BLACK",
903         font=fuente_texto,
904     )
```

```
905     boton3.place(x=870, y=700)
906     botonvolver = tk.Button(
907         ventana,
908         text="VOLVER",
909         command=mostrar_inicio,
910         relief=tk.GROOVE,
911         borderwidth=5,
912         bg="#d97675",
913         fg="BLACK",
914         font=fuente_texto,
915     )
916     botonvolver.place(x=10, y=10)
917     botonVolverInicio = tk.Button(
918         ventana,
919         text="VOLVER AL INICIO",
920         command=mostrar_inicio,
921         relief=tk.GROOVE,
922         borderwidth=5,
923         bg="#d97675",
924         fg="BLACK",
925         font=fuente_texto,
926     )
927     botonVolverInicio.place(x=10, y=10)
```


Al compilar y ejecutar el programa se verá así:



ANALIZADOR DE GRILLAS

VOLVER

Monitoreo De Camara



Intervalo de tiempo (segundos)

2

Duración de captura:

0 horas3 minutos4 segundos

Iniciar Captura

Progreso captura de fotos

INICIAR CAMARA

DETENER CAMARA

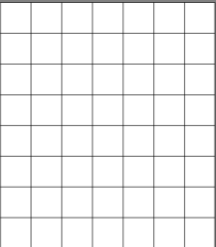
Fotos Tomadas:

C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202933.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202936.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202936.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202940.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202942.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202944.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202946.jpg
C:/Users/jerpa/OneDrive/Escritorio/CIM_grilla/Fotos_Nuevo_proyecto/Fotos_Humedad\Photos_20240305-202933\20240305-202948.jpg

ANALIZADOR DE GRILLAS

VOLVER

Grilla ideal Creada por el Usuario



Dimensiones de la grilla

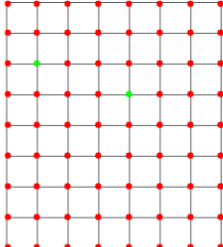
7

8

CREAR

Distancia entre los puntos seleccionados: 158.11 píxeles

Algoritmo de Harris Aplicado



Aplicar Algoritmo de Harris

Aplicar