UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES

Universidad Del Bío-Bío

Detección De Esquinas

Ingeniería Civil Informática

Jerson Antonio Palma Sarandona
Ingeniería Civil en Informática en formación

Introducción

El desarrollo de aplicaciones para la detección de grilla es esencial en diversas áreas, como la industria, la investigación y la agricultura. Este tutorial 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. 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

Código.

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

```python
import cv2
import numpy as np
import tkinter as tk
from PIL import Image, ImageTk, ImageDraw
```

Primero definiremos listas vacías y variables de seguimiento, con las cuales trabajaremos para construir el algoritmo

```python
# Lista para almacenar los puntos
points = []
pointsB = []
# Variables para el seguimiento de la selección de puntos algoritmo de harris
selected_points = []
selected_point_indices = []
selected_pointsB = []
selected_point_indicesB = []
```

Se define un tamaño predeterminado para la función *resize_imagen ()*

Funciones:

LimpiarLabels () setea listas anteriormente definidas.

```python
def limpiarlabels():
    points.clear()
    pointsB.clear()
    selected_points.clear()
    selected_point_indices.clear()
    selected_pointsB.clear()
    selected_point_indicesB.clear()
```

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

```python
def resize_image(image_path, target_size, background_color=(255, 255, 255)):
    # Abrir la imagen con Pillow
    original_image = Image.open(image_path)

    # Crear una nueva imagen del tamaño deseado con el fondo especificado
    new_image = Image.new("RGB", target_size, background_color)

    # Calcular las coordenadas para centrar la imagen original en la nueva imagen
    x_offset = (target_size[0] - original_image.width) // 2
    y_offset = (target_size[1] - original_image.height) // 2

    # Pegar la imagen original en la nueva imagen en las coordenadas calculadas
    new_image.paste(original_image, (x_offset, y_offset))

    return new_image
```

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

```
61  def update_image(label2, actualizacion):
62      # Convierte la imagen para mostrar en la ventana de Tkinter a formato RGB
63      image_to_show = cv2.cvtColor(actualizacion, cv2.COLOR_BGR2RGB)
64      image_to_show = Image.fromarray(image_to_show)
65      image_to_show = ImageTk.PhotoImage(image_to_show)
66      label2.configure(image=image_to_show)
67      label2.image = image_to_show
```

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

```
44  def umbralizacion(grays):
45      valor = 166
46      ret, thresh1 = cv2.threshold(grays, valor, 255, cv2.THRESH_BINARY)
47      return thresh1
```

Creación De Menú y Ventana:

creación de ventana y la creación del menú estará dado por *mostrar_menu()* , dicha función muestra el menú primero limpiando ejecutando *LimpiarLabels() y "ocultar_menu"* se define el tamaño de la ventana, y se crean los botones con los cuales se navegara por el programa

```
386  ventana = tk.Tk()
387  mostrar_menu()
```

```
373  def mostrar_menu():
374      limpiarlabels()
375      ocultar_menu()
376      global boton1, boton2
377      ventana.geometry("1920x900")
378
379      boton1 = tk.Button(ventana, text="1: grilla ideal", command=mostrar_grilla)
380      boton1.pack()
381      boton2 = tk.Button(ventana, text="2: humedad", command=mostrar_humedad)
382      boton2.pack()
383
```

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

```
70  def ocultar_menu():
71      # Destruye todos los widgets de la ventana
72      for widget in ventana.winfo_children():
73          widget.destroy()
```

mostrar_1() una función que muestra los widgets predeterminados para las dos funcionalidades principales (crear grilla y grilla expuesta a humedad)

```python
def mostrar_1():
    global label1, label2
    ocultar_menu()
    botonMostarMenu = tk.Button(ventana, text="VOLVER", command=mostrar_menu)
    botonMostarMenu.place(x=10, y=10, width=100, height=23)
    label1 = tk.Label(ventana, background="gray")
    label2 = tk.Label(ventana, background="gray")
    label1.place(x=50, y=50, width=800, height=800)
    label2.place(x=1100, y=50, width=800, height=800)
    limpiarlabels()
```

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

```python
def crear_grilla():
    limpiarlabels()
    global grillacv
    # Tamaño de celda y tamaño de la imagen
    ancho_celda = 50
    alto_celda = 50
    ancho_imagen = (int(Sancho.get())) * ancho_celda  # Ajuste en el ancho
    alto_imagen = (int(Slargo.get())) * alto_celda  # Ajuste en el alto

    # Crea una nueva imagen
    imagengrilla = Image.new("RGB", (ancho_imagen, alto_imagen), color="white")
    dibujo = ImageDraw.Draw(imagengrilla)

    # Dibuja la grilla
    for i in range(0, ancho_imagen, ancho_celda):
        dibujo.line([(i, 0), (i, alto_imagen)], fill="black")

    for j in range(0, alto_imagen, alto_celda):
        dibujo.line([(0, j), (ancho_imagen, j)], fill="black")

    # Dibuja las líneas adicionales para el borde derecho e inferior
    dibujo.line(
        [(ancho_imagen - 1, 0), (ancho_imagen - 1, alto_imagen)], fill="black"
    )  # Borde derecho
    dibujo.line(
        [(0, alto_imagen - 1), (ancho_imagen, alto_imagen - 1)], fill="black"
    )  # Borde inferior
```

- Se guarda la imagen con el método ".save" y se define el nombre que llevará (dicha imagen se guardara en la misma carpeta en que se encuentra el archivo .py)

```
103
104     # Guarda la imagen generada
105     imagengrilla.save("grilla.png")
106     imagenideal = ImageTk.PhotoImage(imagengrilla)
107     # MUESTRA EN EL LABEL
108     label1.config(image=imagenideal)
109     label1.image = imagenideal
```

- 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.

```
110     # Convierte la imagen a escala de grises
111
112     grillacv = cv2.imread("C:/Users/jerpa/OneDrive/Escritorio/grilla.png")
113
114     image_path = "C:/Users/jerpa/OneDrive/Escritorio/grilla.png"
115
116     grillacv = resize_image(image_path, target_size)
117
118     # Guardar la nueva imagen
119     grillacv.save("C:/Users/jerpa/OneDrive/Escritorio/grilla.png")
120
121     grillacv = cv2.imread("C:/Users/jerpa/OneDrive/Escritorio/grilla.png")
122
123     gray2 = cv2.cvtColor(grillacv, cv2.COLOR_BGR2GRAY)
124     buscar_dibujar_puntos_harris(umbralizacion(gray2), grillacv)
125     update_image(label2, grillacv)
126
```

Buscar_dibujar_puntos_harris() funcion que aplica el algoritmo de Harris para detectar y dibujar las cruces de una imagen. Detecta los puntos en la imagen umbralizada , luego de obtener las coordenadas, las dibuja en la imagen original.

- Aplica el algoritmo de harris y guarda las coordenadas de la intersección (el valor 3000 de la línea 139 se refiere a tomar los primeros 3000 puntos que tengan mayor probabilidad de ser una esquina, este valor debe variar según con que imágenes se está trabajando y el tamaño de la grilla, es decir tiene que ser testeado)

```
128 def buscar_dibujar_puntos_harris(gray, original):
129     # Aplicar la detección de esquinas usando el algoritmo de Harris
130     dst = cv2.cornerHarris(gray, 2, 3, 0.04)
131     # Normalizar y aplicar un umbral para obtener las esquinas
132     dst_norm = cv2.normalize(dst, None, 0, 255, cv2.NORM_MINMAX)
133     threshold = 0.01 * dst_norm.max()
134     corners = np.column_stack(
135         np.where(dst_norm > threshold)
136     )  # Obtener coordenadas como una lista de tuplas
137
138     # Ordenar las esquinas por respuesta de Harris y tomar las primeras 100
139     corners = sorted(corners, key=lambda x: dst_norm[x[0], x[1]], reverse=True)[:3000]
```

- Dibuja los puntos en la imagen original

```
140
141     # Supresión no máxima
142     valid_corners = []
143     for i, (y, x) in enumerate(corners):
144         if all(
145             np.linalg.norm(np.array([y, x]) - np.array([yc, xc])) > 10
146             for yc, xc in valid_corners
147         ):
148             valid_corners.append((y, x))
149
150     # Dibujar los puntos en la imagen
151     for y, x in valid_corners:
152         cv2.circle(original, (x, y), 5, (0, 0, 255), -1)
153         points.append((x, y))
154
```

El programa también trabaja con otra función igual para el label() derecho, solo que cambian en donde se guardan los valores. En esencia es la misma.

```
156 def buscar_dibujar_puntos_harrisB(gray, original):
157     # Aplicar la detección de esquinas usando el algoritmo de Harris
158     dst2 = cv2.cornerHarris(gray, 2, 3, 0.04)
159     # Normalizar y aplicar un umbral para obtener las esquinas
160     dst_norm2 = cv2.normalize(dst2, None, 0, 255, cv2.NORM_MINMAX)
161     threshold2 = 0.01 * dst_norm2.max()
162     corners2 = np.column_stack(
163         np.where(dst_norm2 > threshold2)
164     )  # Obtener coordenadas como una lista de tuplas
165
166     # Ordenar las esquinas por respuesta de Harris y tomar las primeras 100
167     corners2 = sorted(corners2, key=lambda x: dst_norm2[x[0], x[1]], reverse=True)[
168         :3000
169     ]
170
171     # Supresión no máxima
172     valid_corners2 = []
173     for i, (y, x) in enumerate(corners2):
174         if all(
175             np.linalg.norm(np.array([y, x]) - np.array([yc, xc])) > 10
176             for yc, xc in valid_corners2
177         ):
178             valid_corners2.append((y, x))
179
180     # Dibujar los puntos en la imagen
181     for y, x in valid_corners2:
182         cv2.circle(original, (x, y), 5, (0, 0, 255), -1)
183         pointsB.append((x, y))
184
```

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*

```python
def mostrar_grilla():
    global Sancho, Slargo
    mostrar_1()
    Sancho = tk.Scale(ventana, from_=1, to=15, orient="horizontal")
    Sancho.place(x=300, y=860)
    Slargo = tk.Scale(ventana, from_=1, to=15, orient="horizontal")
    Slargo.place(x=430, y=860)
    botoncrear = tk.Button(ventana, text="CREAR", command=crear_grilla)
    botoncrear.place(x=550, y=860, width=100, height=23)
    resultadopixel = tk.Label(ventana, text="")
    resultadopixel.place(x=900, y=860)

    label2.bind(
        "<Button-1>",
        lambda event: on_canvas_click2(event, grillacv, label2, resultadopixel),
    )

```

on_canvas_click, una vez se haga click en un label se llama a la función on_canvas_click, y lo que hace es comparar si donde se hizo click es un punto de los puntos detectados anteriormente en por la función *Buscar_dibujar_puntos_harris*

```python
def on_canvas_click(
    event, imagencita, labelA, labelB, indices, puntos_seleccionados, puntos
):
    # por algun motivo las coordenadas que entrega el evento son 25 pixeles mas
    # que las coordenadas de la imagen, por eso se le resta 25 (solo en el caso de la grilla ideal)
    x, y = event.x - 6, event.y - 6
    print(f"x: {x} y:{y}")

    for i, (px, py) in enumerate(puntos):
        if abs(x - px) < 5 and abs(y - py) < 5:
            indices.append(i)
            puntos_seleccionados.append((px, py))
            cv2.circle(imagencita, (px, py), 5, (0, 255, 0), -1)

    if len(indices) == 2:
        calculate_distance(puntos_seleccionados, labelB)
        puntos_seleccionados.clear()
        indices.clear()

    # Actualiza la imagen en el widget de etiqueta
    update_image(labelA, imagencita)

```

La función *on_canvas_click2* es idéntica a la anterior solo cambia las variables

```python
def on_canvas_click2(event, imagencita, labelA, labelB):
    # por algun motivo las coordenadas que entrega el evento son
    # 25 pixeles mas que las coordenadas de la imagen, por eso se le resta 25
    # (solo en el caso de la grilla ideal)
    x, y = event.x - 25, event.y - 25
    print(f"x: {x} y:{y}")

    for i, (px, py) in enumerate(points):
        if abs(x - px) < 5 and abs(y - py) < 5:
            selected_point_indices.append(i)
            selected_points.append((px, py))
            cv2.circle(imagencita, (px, py), 5, (0, 255, 0), -1)

    if len(selected_point_indices) == 2:
        calculate_distance(selected_points, labelB)
        selected_points.clear()
        selected_point_indices.clear()

    # Actualiza la imagen en el widget de etiqueta
    update_image(labelA, imagencita)

```

Calculate_distance() Dicha función calcula la distancia euclidiana entre dos puntos

```python
def calculate_distance(selected_points, labelactualizado):
    if len(selected_points) == 2:
        point1 = selected_points[0]
        point2 = selected_points[1]
        distance = np.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]) ** 2)
        labelactualizado.config(
            text=f"Distancia entre los puntos seleccionados: {distance:.2f} píxeles"
        )


def calculate_distanceB(selected_pointsB, labelactualizado):
    if len(selected_pointsB) == 2:
        point1 = selected_pointsB[0]
        point2 = selected_pointsB[1]
        distance = np.sqrt((point2[0] - point1[0]) ** 2 + (point2[1] - point1[1]) ** 2)
        labelactualizado.config(
            text=f"Distancia entre los puntos seleccionados: {distance:.2f} píxeles"
        )

```

Mostrar_imagen Obtiene las fotos seleccionadas en la lista desplegable y las muestra en los labels

```
269  def mostrar_imagen():
270      limpiarlabels()
271      global imagen1, imagen2
272      label1.config(text="", bg="gray")
273
274      foto1 = opciones_imagenes[opcion_seleccionada.get()]
275      foto2 = opciones_imagenes[opcion_seleccionada2.get()]
276
277      imagen1 = cv2.imread(foto1)
278      alto_original, ancho_original = imagen1.shape[:2]
279      nuevo_ancho = int(ancho_original * 73 / 100)
280      nuevo_alto = int(alto_original * 73 / 100)
281      imagen1 = cv2.resize(imagen1, (nuevo_ancho, nuevo_alto))
282
283      imagen2 = cv2.imread(foto2)
284      alto_original, ancho_original = imagen2.shape[:2]
285      nuevo_ancho = int(ancho_original * 73 / 100)
286      nuevo_alto = int(alto_original * 73 / 100)
287      imagen2 = cv2.resize(imagen2, (nuevo_ancho, nuevo_alto))
288
289      gray = cv2.cvtColor(imagen1, cv2.COLOR_BGR2GRAY)
290      gray2 = cv2.cvtColor(imagen2, cv2.COLOR_BGR2GRAY)
291      buscar_dibujar_puntos_harris(umbralizacion(gray), imagen1)
292      buscar_dibujar_puntos_harrisB(umbralizacion(gray2), imagen2)
293      update_image(label1, imagen1)
294      update_image(label2, imagen2)
295
```

*opcion_cambiada(*arg)* Funciones para las variables de control del menú desplegable (no realizan ninguna acción)

```
297  def opcion_cambiada(*args):
298      print(".")
299
300
301  def opcion_cambiada2(*args):
302      print(".")
303
```

Mostrar_humedad() Muestra los widgets de la ventana “humedad”, crea una lista para con las opciones del menú desplegable, se les asigna los path de las imágenes a cada una de las imágenes

```python
def mostrar_humedad():
    mostrar_1()

    global opcion_seleccionada, opcion_seleccionada2, opciones_imagenes
    # opciones para el menu desplegable
    opciones_imagenes = {
        "original": "C:/Users/jerpa/OneDrive/Escritorio/imagenes de cuadricula/IMG_20231023_090543.jpg",
        "11:00": "C:/Users/jerpa/OneDrive/Escritorio/imagenes de cuadricula/IMG_20231023_090551.jpg",
        "12:30": "C:/Users/jerpa/OneDrive/Escritorio/imagenes de cuadricula/IMG_20231023_090557.jpg",
        "13:00": "C:/Users/jerpa/OneDrive/Escritorio/imagenes de cuadricula/IMG_20231023_090602.jpg",
    }

    opciones = list(opciones_imagenes.keys())

    # Variable de control para el menú desplegable
    opcion_seleccionada = tk.StringVar(ventana)
    opcion_seleccionada.set(opciones[0])  # Establecer la opción predeterminada
    opcion_seleccionada.trace("w", opcion_cambiada2)
    # Crear el menú desplegable
    menu_desplegable = tk.OptionMenu(ventana, opcion_seleccionada, *opciones)
    menu_desplegable.place(x=925, y=60)

    # Variable de control para el menú desplegable
    opcion_seleccionada2 = tk.StringVar(ventana)
    opcion_seleccionada2.set(opciones[0])  # Establecer la opción predeterminada
    opcion_seleccionada2.trace("w", opcion_cambiada)
    # Crear el menú desplegable
    menu_desplegable2 = tk.OptionMenu(ventana, opcion_seleccionada2, *opciones)
    menu_desplegable2.place(x=925, y=360)
```

- define el evento de click y que una vez que haga click en uno de los dos labels se ejecuta la función *on_canvas_click*

```python
    label1.bind(
        "<Button-1>",
        lambda event: on_canvas_click(
            event,
            imagen1,
            label1,
            resultadopixel,
            selected_point_indices,
            selected_points,
            points,
        ),
    )
    label2.bind(
        "<Button-1>",
        lambda event: on_canvas_click(
            event,
            imagen2,
            label2,
            resultadopixel2,
            selected_point_indicesB,
            selected_pointsB,
            pointsB,
        ),
    )
```