Práctica 2: Desarrollo del mapa electrónico de la carretera

GRUPO 7:

Jonathan Bautista → bk008

Jorge Veleriano → bq0063

Objetivo general:

El principal objetivo de esta práctica 2 es extender la aplicación desarrollada en la práctica 1 para que sea capaz de representar en pantalla la posición suministrada por el receptor GPS. Se capturará una imagen del Google Earth del Campus Sur y otra del INSIA, se georeferenciará en la pantalla del ordenador y se mostrará junto a la posición del receptor GPS en tiempo real. Esta imagen debe ser coherente con la posición suministrada por el receptor GPS. La aplicación debe realizar las funciones de visualización similares a las de un navegador comercial. Las pruebas finales de esta práctica se realizarán con un vehículo instrumentado con GPS en la pista de ensayo del INSIA.

Componentes de la aplicación

A continuación , se va a explicar el código generado, el cual se enfoca en crear una aplicación gráfica utilizando la librería Tkinter, que muestra un mapa con puntos actualizados a partir de las coordenadas GPS recibidas en tiempo real a través de un puerto serial:

En primer lugar se ha definido las siguientes librerías:

```
6
7   import tkinter as tk
8   from tkinter import ttk
9   from PIL import Image, ImageDraw, ImageTk
10  import threading
11  import queue
12  import serial
13  import pyproj
```

t**kinter y ttk**: tkinter es una librería estándar de Python para crear interfaces gráficas de usuario (GUI). En este caso, se utiliza para crear la ventana principal, los componentes y los eventos de la interfaz.

ttk es una sublibrería que ofrece widgets adicionales para mejorar la apariencia de los componentes de la GUI, como las etiquetas (Label), que se usan para mostrar la imagen en la ventana.

PIL (Python Imaging Library): PIL es una librería para manipulación de imágenes. En este caso, se usa la clase Image para abrir y trabajar con la imagen base, ImageDraw para dibujar en la imagen, y ImageTk para convertir la imagen de PIL en un formato que pueda ser mostrado en Tkinter.

threading: esta librería se utiliza para ejecutar tareas en paralelo. En este código, se usa para crear un hilo que lee continuamente los datos del GPS sin bloquear la interfaz gráfica.

queue: la librería queue se utiliza para manejar la comunicación entre el hilo que lee los datos del GPS y la interfaz gráfica. En este caso, una cola (Queue) se utiliza para almacenar las coordenadas GPS que se reciben.

serial: esta librería permite la comunicación con dispositivos a través de puertos seriales, como un GPS. Aquí se utiliza para leer los datos del GPS desde el puerto COM7.

pyproj: pyproj es una librería para realizar transformaciones de coordenadas geográficas. En este caso, se usa para convertir las coordenadas GPS (en formato GGA) a coordenadas UTM, que son más adecuadas para trabajar con mapas en 2D.

En segundo lugar, se ha creado las siguientes funciones:

transformar_gga_a_utm(latitud_gga, longitud_gga):

```
16  # Función para transformar las coordenadas GGA a UTM:
17  def transformar_gga_a_utm(latitud_gga, longitud_gga):
18      # Convertir latitud y longitud a grados decimales
19      latitud_decimal = float(latitud_gga[:2]) + float(latitud_gga[2:]) / 60
20      longitud_decimal = -1 * (float(longitud_gga[:3]) + float(longitud_gga[3:]) / 60)
21
22      # Definición de los sistemas de coordenadas WGS84 y UTM
23      sistema_wgs84 = pyproj.CRS('EPSG:4326')
24      sistema_utm = pyproj.CRS.from_epsg(32630)
25
26      # Crear el transformador entre sistemas de coordenadas
27      transformador = pyproj.Transformer.from_crs(sistema_wgs84, sistema_utm, always_xy=True)
28
29      # Transformar las coordenadas
30      easting, northing = transformador.transform(longitud_decimal, latitud_decimal)
31      return easting, northing, '30T'
32
```

Esta función convierte las coordenadas GPS en formato GGA (latitud y longitud) a coordenadas UTM. Primero convierte las coordenadas a grados decimales y luego las transforma del sistema de referencia WGS84 a UTM usando pyproj

convertir_utm_a_imagen(coordenadas_utm, imagen_objeto):

```python
def convertir_utm_a_imagen(coordenadas_utm, imagen_objeto):
    x_utm, y_utm = coordenadas_utm

    esquina_noroeste = (446175.44, 4471052.89)  # Coordenadas UTM de la esquina superior izquierda
    esquina_sureste = (446573.82, 4470710.86)  # Coordenadas UTM de la esquina inferior derecha

    ancho_imagen, alto_imagen = imagen_objeto.size

    x_pixel = (x_utm - esquina_noroeste[0]) / (esquina_sureste[0] - esquina_noroeste[0]) * ancho_imag
    y_pixel = (1 - (y_utm - esquina_sureste[1]) / (esquina_noroeste[1] - esquina_sureste[1])) * alto_

    return x_pixel, y_pixel

```

Convierte las coordenadas UTM a píxeles en la imagen que se muestra en la interfaz. Calcula la relación entre las coordenadas UTM y las dimensiones de la imagen base para determinar la posición del punto en la imagen.

dibujar_punto(coordenadas_utm, imagen_actual):

```python
# Función para dibujar un punto en la imagen:
def dibujar_punto(coordenadas_utm, imagen_actual):
    # Crear un objeto de dibujo en la imagen
    diseño = ImageDraw.Draw(imagen_actual)
    tamaño_punto = 3  # Tamaño del punto que se dibujará

    # Mapear las coordenadas UTM a las coordenadas de la imagen
    x_pixel, y_pixel = convertir_utm_a_imagen(coordenadas_utm, imagen_actual)

    # Dibujar un círculo rojo en la imagen
    diseño.ellipse([x_pixel - tamaño_punto, y_pixel - tamaño_punto, x_pixel + tamaño_punto, y_pixel +

    return imagen_actual

```

Dibuja un punto sobre la imagen. Usa ImageDraw para crear un círculo rojo en la ubicación correspondiente a las coordenadas UTM, que previamente se han convertido a píxeles.

leer_gps(cola_datos, puerto_serial):

```python
def leer_gps(cola_datos, puerto_serial):
    while True:
        # Leer una línea de datos GPS desde el puerto
        linea_gps = puerto_serial.readline().decode('utf-8').strip()

        # Verificar si es una trama 'GPGGA'
        if 'GPGGA' in linea_gps:
            partes = linea_gps.split(',')
            print("Tramas recibidas:", partes)

            # Verificar que los datos de la latitud y la longitud sean válidos
            if len(partes) > 5 and partes[2] and partes[4]:
                latitud = partes[2]
                longitud = partes[4]

                # Conversión a UTM y poner las coordenadas en la cola
                easting, northing, zona = transformar_gga_a_utm(latitud, longitud)
                cola_datos.put((easting, northing))

```

Lee datos del GPS desde el puerto serial especificado. Extrae las coordenadas de latitud y longitud de las tramas GGA recibidas y las convierte a coordenadas UTM antes de almacenarlas en la cola (queue).

actualizar_grafico(ventana, cola_datos, imagen_base, etiqueta_imagen, ultimo_punto):

```python
def actualizar_grafico(ventana, cola_datos, imagen_base, etiqueta_imagen, ultimo_punto):
    try:
        while True:
            # Procesar todos los elementos en la cola
            easting, northing = cola_datos.get_nowait()
            nuevo_punto = (easting, northing)

            # Solo actualizar si el nuevo punto es diferente al último
            if nuevo_punto != ultimo_punto[0]:
                imagen_actualizada = dibujar_punto(nuevo_punto, imagen_base)
                mostrar_imagen(imagen_actualizada, etiqueta_imagen)
                ultimo_punto[0] = nuevo_punto  # Actualizar el último punto

    except queue.Empty:
        pass
    finally:
        # Llamar nuevamente a la función para actualizar la interfaz
        ventana.after(100, actualizar_grafico, ventana, cola_datos, imagen_base, etiqueta_imagen, ulti
```

Esta función es llamada periódicamente para actualizar la interfaz gráfica con los nuevos datos de coordenadas. Lee los datos de la cola, dibuja el punto en la imagen y actualiza la visualización si el punto es diferente del último dibujado.

mostrar_imagen(imagen, etiqueta):

```python
def mostrar_imagen(imagen, etiqueta):
    imagen_tk = ImageTk.PhotoImage(imagen)
    etiqueta.configure(image=imagen_tk)
    etiqueta.image = imagen_tk

# Configuración inicial de la interfaz gráfica
ventana_principal = tk.Tk()
ventana_principal.title("Visualización de Coordenadas GPS")

# Cargar la imagen base sobre la cual se dibujarán los puntos
ruta_imagen = "Fotos/insia.png"
imagen_base = Image.open(ruta_imagen)
imagen_tk = ImageTk.PhotoImage(imagen_base)

# Crear la etiqueta donde se mostrará la imagen
etiqueta_imagen = ttk.Label(ventana_principal, image=imagen_tk)
etiqueta_imagen.pack()

# Configuración de la cola y el puerto serie COM7
cola_datos = queue.Queue()
puerto_serial = serial.Serial('COM7', 4800, timeout=1)

# Iniciar un hilo para leer los datos GPS
hilo_gps = threading.Thread(target=leer_gps, args=(cola_datos, puerto_serial), daemon=True)
hilo_gps.start()
```

Toma la imagen actualizada y la convierte al formato adecuado para ser mostrada en el widget Label de Tkinter. La imagen es luego renderizada en la interfaz gráfica.

Flujo del programa

El flujo del programa comienza con la configuración de la interfaz gráfica, donde se inicializa la ventana principal de Tkinter y se carga la imagen base sobre la que se dibujarán los puntos GPS. Se crea una etiqueta para mostrar esta imagen. Luego, se configura el puerto serial (COM7) para leer los datos GPS, utilizando una cola para almacenar las coordenadas leídas, lo que permite que el hilo de lectura y la interfaz gráfica operen de manera independiente. Un hilo en segundo plano se encarga de leer constantemente los datos del GPS, transformarlos a coordenadas UTM y colocarlos en la cola. La función actualizar_grafico se ejecuta periódicamente para actualizar la interfaz con las nuevas coordenadas, dibujando un punto en la imagen si las coordenadas son diferentes de las anteriores. Finalmente, el bucle principal de Tkinter mantiene la aplicación en ejecución y permite la interacción continua con el usuario. Este código demuestra cómo integrar datos GPS en tiempo real con una interfaz gráfica, utilizando varias librerías de Python para manejar la imagen, las coordenadas geográficas y la comunicación con el puerto serial, todo mientras mantiene una experiencia fluida y actualizada en la interfaz gráfica.