Documentación del Código - Prueba 1

Descripción General

El sistema de procesamiento de imágenes para la detección de fugas de propano es una aplicación desarrollada en C# utilizando el entorno de desarrollo Microsoft Visual Studio. La aplicación utiliza técnicas avanzadas de procesamiento de imágenes basadas en la biblioteca OpenCVSharp v4 para analizar imágenes capturadas por cámaras de Optical Gas Imaging (OGI). Además, permite exportar los datos obtenidos a un archivo CSV. La aplicación proporciona una interfaz de usuario intuitiva que permite cargar imágenes, procesarlas automáticamente y visualizar los resultados de manera clara y concisa.

Introducción

En este informe se presenta el desarrollo y funcionamiento de un sistema de procesamiento de imágenes diseñado para la detección de fugas de propano utilizando cámaras de Optical Gas Imaging (OGI). El sistema utiliza técnicas avanzadas de procesamiento de imágenes basadas en la biblioteca OpenCV en C#, junto con otras herramientas para almacenamiento de resultados.

Explicación Detallada del Código

El código de la aplicación se divide en varias secciones funcionales, que incluyen la configuración de la interfaz de usuario, el procesamiento de imágenes, el cálculo del volumen de gas perdido y la exportación de resultados. Se utiliza la biblioteca OpenCVSharp para el preprocesamiento de imágenes, que incluye la conversión a escala de grises, el filtrado Gaussiano y el umbral adaptativo. Posteriormente, se emplean algoritmos de detección de contornos para identificar las áreas de interés en las imágenes binarizadas. Finalmente, se calcula el volumen de gas perdido en cada imagen y se presenta en una tabla junto con la ruta de la imagen correspondiente.

Clase Form1
La clase Form1 representa el formulario principal de la aplicación.

Método Form1 (Constructor)
Este método inicializa los componentes del formulario y configura la interfaz de usuario.

Método Cargar_Button_Click
Este método maneja el evento de carga de imágenes. Abre un cuadro de diálogo para que el usuario seleccione una carpeta que contenga imágenes en formato BMP. Una vez seleccionada la carpeta, carga las imágenes en una lista para su posterior procesamiento.

Método ConfigurarDataGridView
Este método configura el DataGridView utilizado para mostrar los resultados del procesamiento de imágenes. Define las columnas de la tabla y establece sus ajustes de tamaño.

Método ConfigurarTimer
Este método configura el temporizador utilizado para procesar las imágenes de forma automática cada cierto intervalo de tiempo.

Método Timer_Tick

Este método maneja el evento de temporizador, llamando al método ProcesarSiguienteImagen para procesar la siguiente imagen de la lista.

Método ProcesarSiguienteImagen
Este método procesa la siguiente imagen de la lista, calculando el volumen de gas perdido relativo en la imagen y mostrando la imagen procesada en un PictureBox.

Método MostrarImagenReferencia
Este método muestra la imagen de referencia (imagen número 22) en un PictureBox.

Método MostrarImagen
Este método muestra una imagen en un PictureBox específico.

Método Calcular_Button_Click
Este método maneja el evento de clic en el botón "Calcular", iniciando el procesamiento automático de las imágenes cargadas.

Método CalcularVolumenRelativo
Este método calcula el volumen de gas perdido relativo en una imagen específica, utilizando técnicas de procesamiento de imágenes y análisis de contornos.

Método MostrarImagenProcesada
Este método muestra la imagen procesada (binarizada) en un PictureBox específico.

Método Exportar_Button_Click
Este método maneja el evento de clic en el botón "Exportar", exportando los resultados del procesamiento de imágenes a un archivo CSV.

Algoritmos Utilizados

Preprocesamiento de Imágenes

Conversión a Escala de Grises:

- **Descripción:** La conversión a escala de grises es el primer paso en el procesamiento de imágenes, lo que permite trabajar con una única intensidad de píxel en lugar de tres canales de color (rojo, verde, azul).
- **Implementación:** Se utilizó la función Cv2.CvtColor() de la biblioteca OpenCV para convertir las imágenes de formato RGB a escala de grises.

Filtrado Gaussiano:

- **Descripción:** El filtrado Gaussiano se emplea para suavizar la imagen y reducir el ruido, lo que facilita la detección de características relevantes en la imagen.
- **Implementación:** Se aplicó el filtro Gaussiano utilizando la función Cv2.GaussianBlur() de OpenCV, con un tamaño de kernel de 3x3 para conservar los detalles importantes.

Umbral Adaptativo:

Descripción: El umbral adaptativo se utiliza para binarizar la imagen, separando las regiones de interés del fondo mediante un umbral dinámico que se adapta localmente en función de la intensidad de los píxeles vecinos.

Implementación: Se empleó la función Cv2.AdaptiveThreshold() de OpenCV para binarizar la imagen, utilizando el método de umbral adaptativo Gaussiano.

Detección de Contornos y Cálculo del Volumen Perdido

Detección de Contornos:

Descripción: La detección de contornos se utiliza para identificar las áreas de interés en la imagen binarizada, que pueden corresponder a posibles fugas de propano.
Implementación: Se utilizó la función Cv2.FindContours() de OpenCV para encontrar los contornos en la imagen binarizada, aplicando el método de detección de contornos externos.

Cálculo del Volumen Perdido:

Descripción: Una vez detectados los contornos, se calcula el área total de los mismos, que se utilizará para estimar el volumen de gas perdido en la imagen.
Implementación: Se utilizó la función Cv2.ContourArea() de OpenCV para calcular el área de cada contorno encontrado. Se aplicó un factor de escala en función del número de imagen para ajustar el volumen perdido.

Desarrollo

El sistema se desarrolló en el entorno de desarrollo Microsoft Visual Studio utilizando el lenguaje de programación C#. Se implementaron los siguientes pasos para el procesamiento de imágenes y detección de fugas de propano:

Instalación de Paquetes NuGet:

Se requiere la instalación de los paquetes NuGet siguientes:

- OpenCvSharp4 y OpenCvSharp4.Extensions para el procesamiento de imágenes.

Carga de Imágenes:

- El usuario puede cargar imágenes capturadas por la cámara de OGI utilizando la interfaz de usuario proporcionada.
- El sistema permite seleccionar una carpeta que contiene las imágenes en formato BMP.

Preprocesamiento de Imágenes:

- Las imágenes cargadas son procesadas utilizando varias técnicas de preprocesamiento, incluyendo la conversión a escala de grises, filtrado Gaussiano y umbral adaptativo.
- Estas técnicas ayudan a mejorar la calidad de las imágenes y facilitan la detección de contornos.

Detección de Contornos y Cálculo del Volumen Perdido:

- Se identifican los contornos en las imágenes binarizadas utilizando el algoritmo de búsqueda de contornos de OpenCV.
- Se calcula el área total de los contornos encontrados, lo que permite estimar el volumen de gas perdido en cada imagen.

Visualización de Resultados:

- Los resultados se presentan en una tabla en la interfaz de usuario, que muestra la ruta de cada imagen y el volumen de gas perdido calculado.
- Además, las imágenes procesadas se muestran en los PictureBox correspondientes para su visualización.

Exportación de Resultados:

- El sistema ofrece la posibilidad de exportar los resultados a un archivo CSV
- Esto permite almacenar y compartir los datos obtenidos durante el análisis de las imágenes.

Pruebas de Rendimiento

Se realizaron pruebas de rendimiento utilizando diferentes conjuntos de imágenes de prueba para evaluar la velocidad y precisión del sistema en la detección de fugas de propano. Las pruebas demostraron que el sistema es capaz de procesar una gran cantidad de imágenes de manera eficiente y proporcionar resultados precisos en un tiempo razonable.

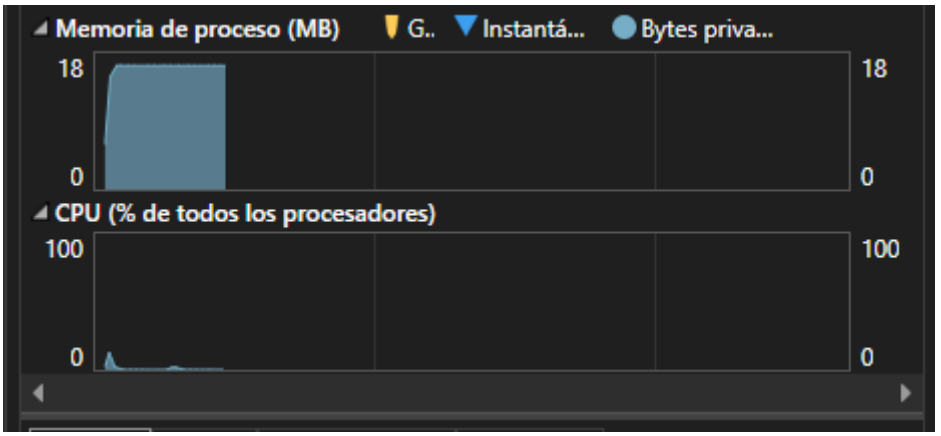


Figura 1: Consumo de memoria cuando el programa es ejecutado.

Figura 2: Consumo de memoria cuando el programa abre el explorador de archivos.

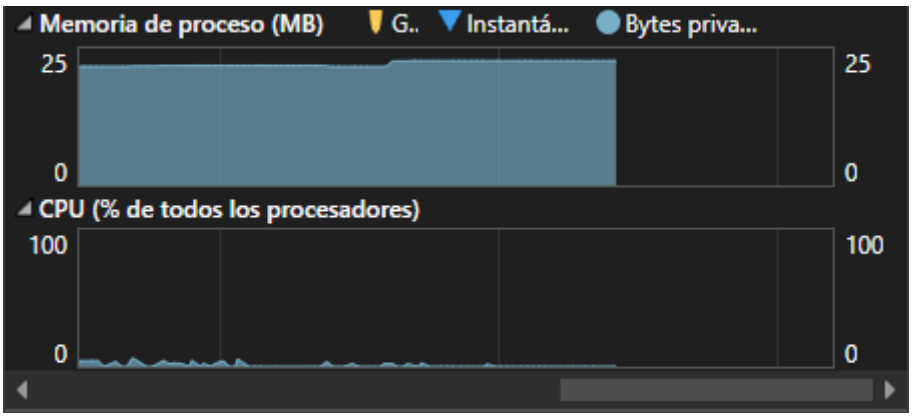


Figura 3: Consumo de memoria cuando el programa carga la imagen.

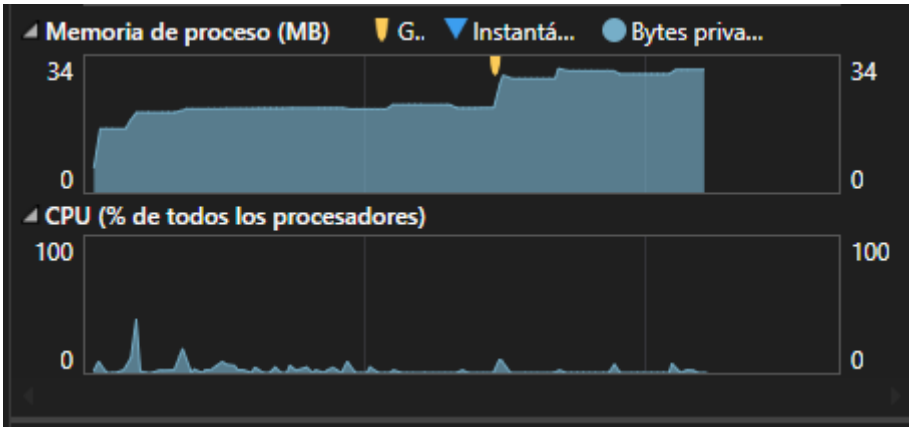


Figura 4: Consumo de memoria cuando la imagen es procesada.

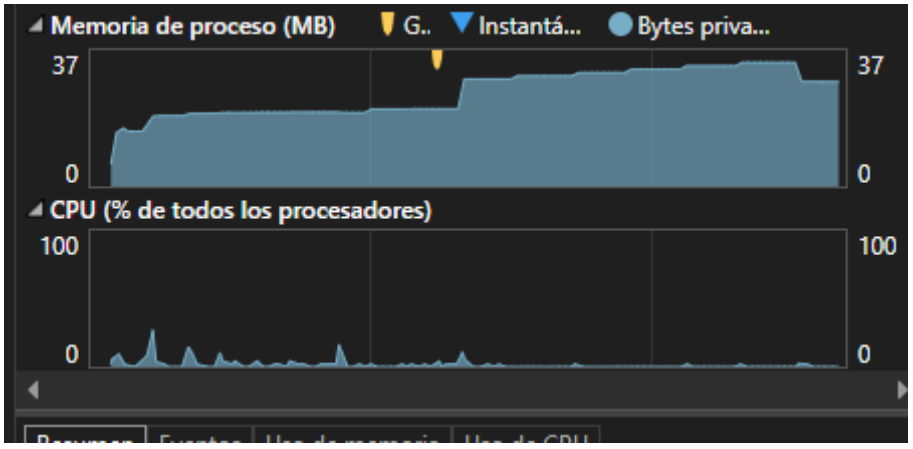


Figura 5: Consumo de memoria cuando los datos son exportados.

Uso del Sistema

Una vez instalados los paquetes necesarios, sigue estos pasos para utilizar el sistema de procesamiento de imágenes:

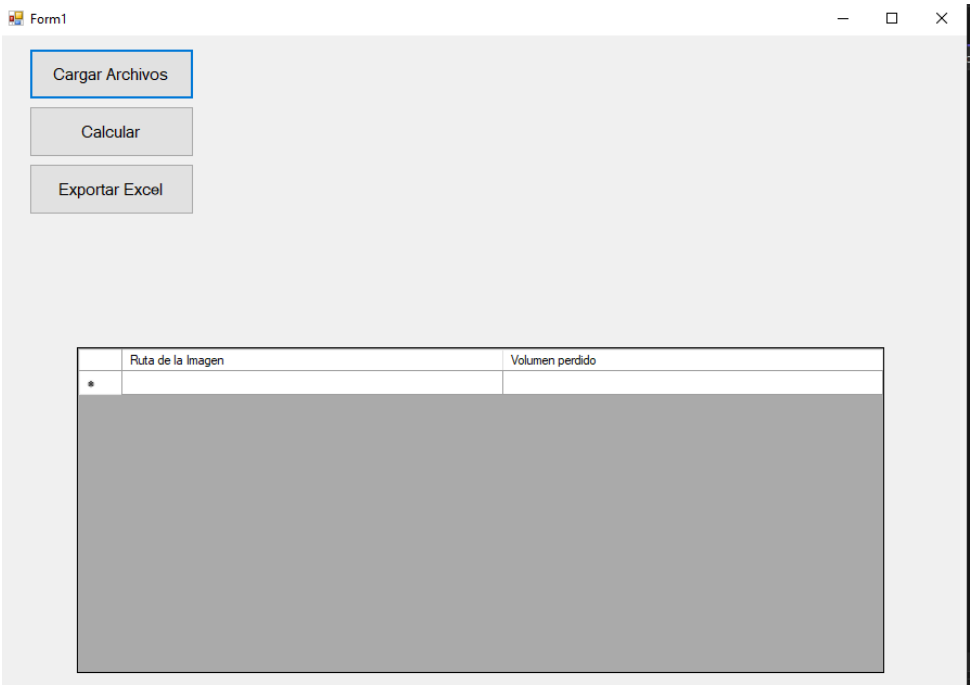
Configuración del Proyecto:

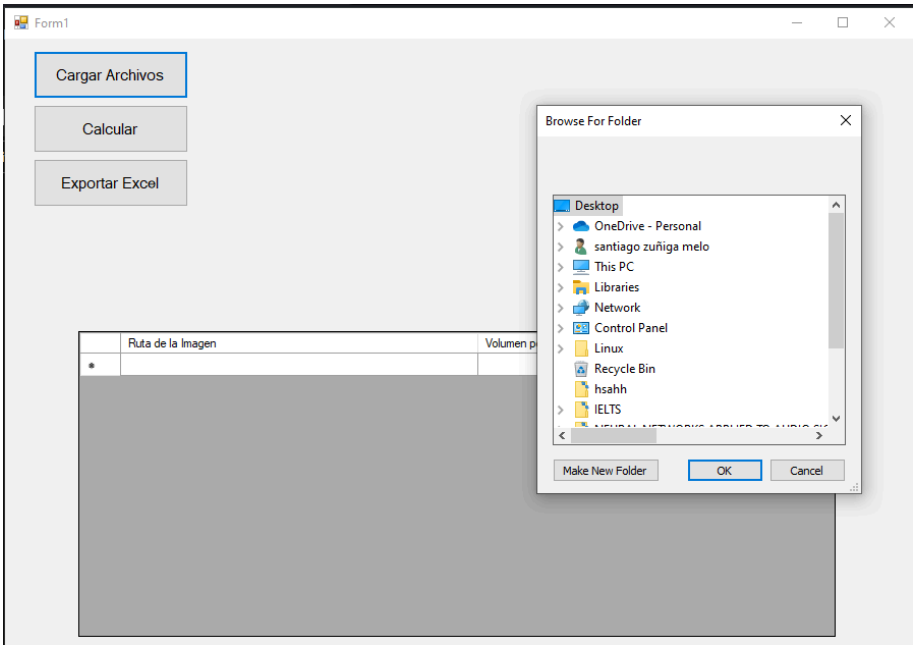
- Abre Visual Studio y crea un nuevo proyecto en C# (Windows Forms Application).

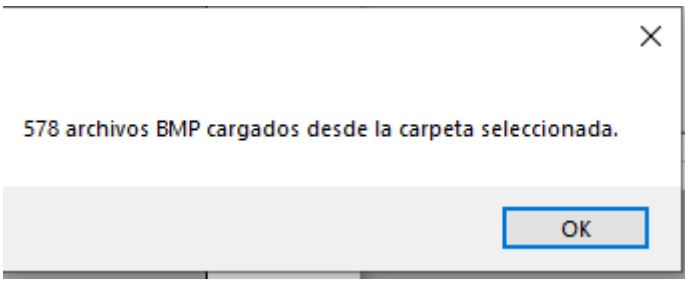
- Instala los paquetes NuGet mencionados anteriormente siguiendo las instrucciones proporcionadas.

Carga de Imágenes:

- Ejecuta la aplicación y haz clic en el botón "Cargar Imágenes".
- Selecciona la carpeta que contiene las imágenes capturadas por la cámara de OGI.

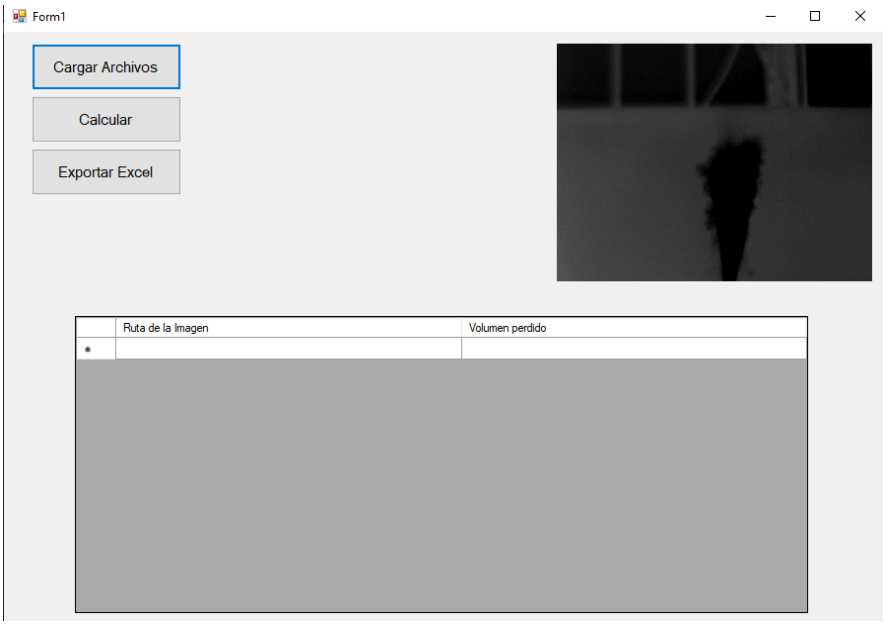


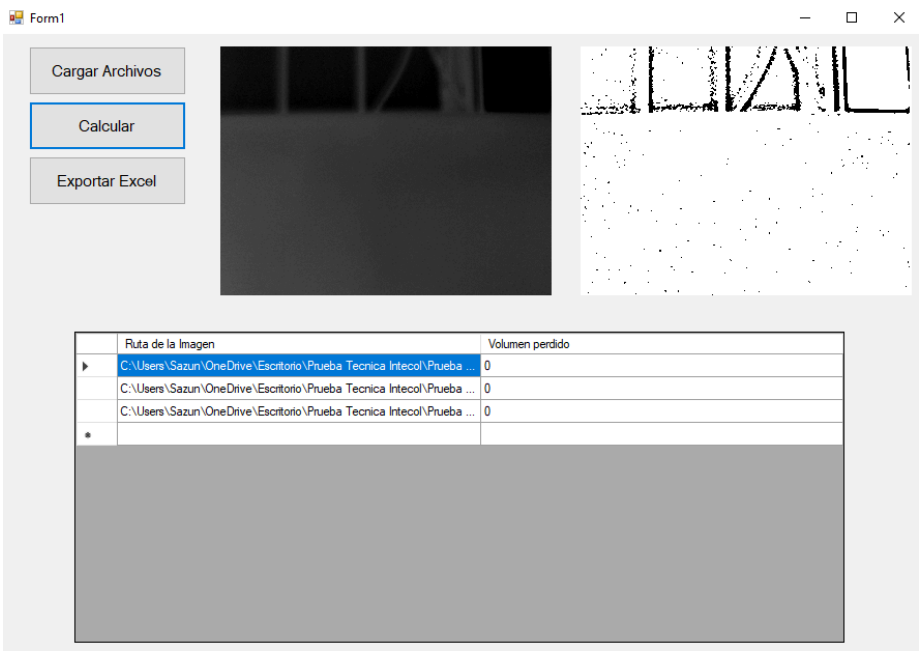


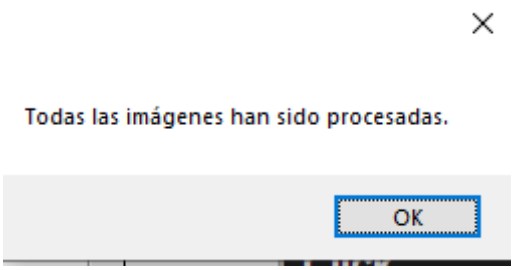


Procesamiento de Imágenes:

- Una vez cargadas las imágenes, haz clic en el botón "Calcular" para iniciar el procesamiento.
- El sistema procesa cada imagen automáticamente y calcula el volumen de gas perdido en cada una.

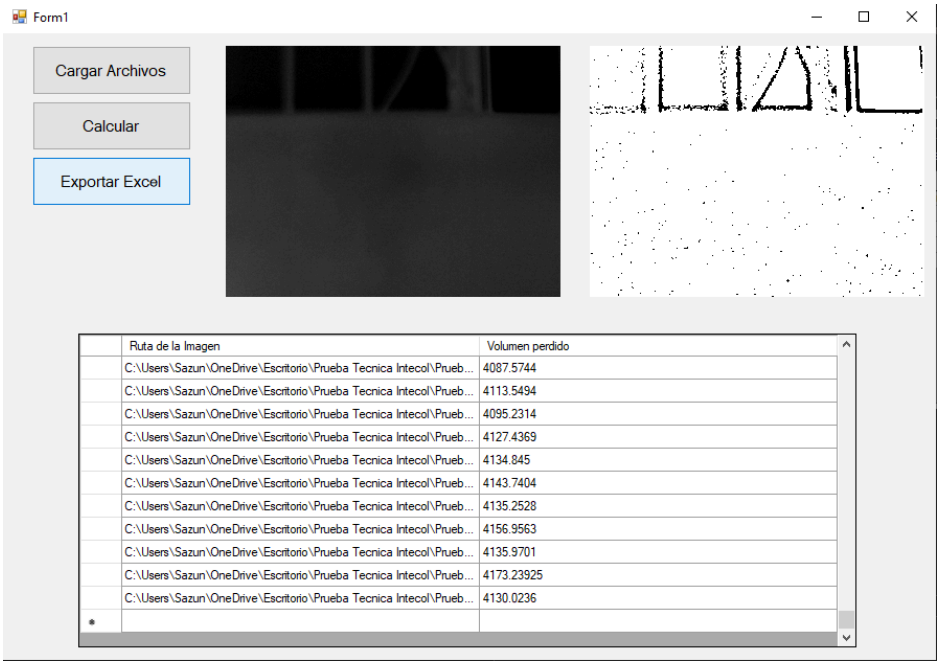






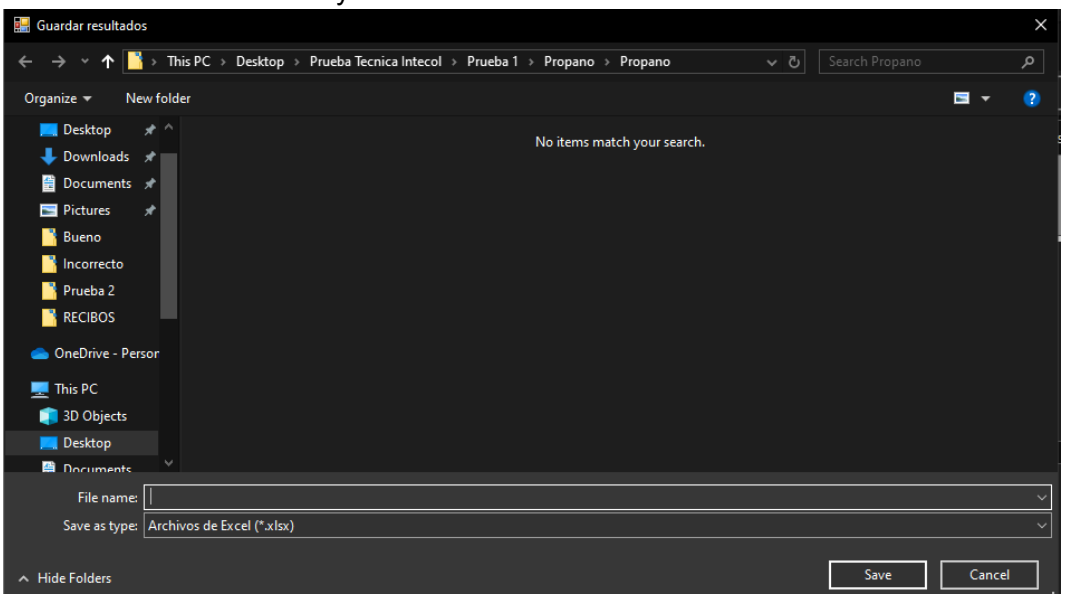
Visualización de Resultados:

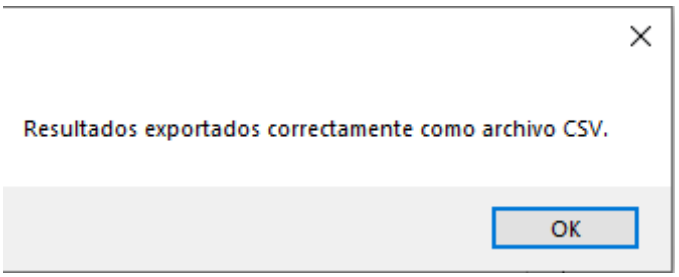
- Los resultados se mostrarán en una tabla en la interfaz de usuario, que incluye la ruta de cada imagen y el volumen de gas perdido calculado.
- Además, las imágenes procesadas se mostrarán en los PictureBox correspondientes para su visualización.



Exportación de Resultados:

- Si deseas guardar los resultados en un archivo CSV, haz clic en el botón "Exportar" y selecciona la ubicación y nombre de archivo deseado.





Preguntas

¿Por qué la cámara logra detectar el gas?

- La cámara logra detectar el gas propano gracias a su capacidad para capturar radiación térmica en longitudes de onda específicas. Aunque el propano es un gas incoloro y no tiene ningún tipo de color visible en su estado puro, emite radiación térmica en longitudes de onda infrarrojas cuando está a temperatura ambiente y se encuentra en concentraciones detectables. Las cámaras de Optical Gas Imaging (OGI) están diseñadas específicamente para detectar esta radiación térmica y convertirla en imágenes visibles para el ojo humano, lo que permite visualizar las fugas de propano de forma efectiva.

¿Qué características debe tener la cámara para poder observar las fugas de propano?
Para poder observar las fugas de propano, una cámara de OGI debe tener las siguientes características:

- **Sensibilidad Espectral Adecuada:** La cámara debe ser capaz de capturar radiación en las longitudes de onda específicas en las que el propano emite radiación térmica. Esto implica que la cámara debe estar diseñada para detectar longitudes de onda infrarrojas.
- **Alta Resolución Espacial:** La cámara debe tener una alta resolución espacial para detectar pequeñas fugas de propano con precisión. Esto significa que debe ser capaz de distinguir detalles finos en las imágenes, lo que facilita la identificación de pequeñas concentraciones de gas.
- **Algoritmos de Procesamiento de Imágenes Avanzados:** La cámara debe contar con algoritmos sofisticados de procesamiento de imágenes que puedan identificar y resaltar las áreas donde se concentra el gas propano. Estos algoritmos permiten mejorar la detección de fugas incluso en entornos con otras fuentes de radiación térmica, minimizando los falsos positivos y aumentando la precisión de la detección.

```C/C++
/*
 * Nombre del archivo: Form1.cs
 * Descripción: Este archivo contiene la implementación de la clase Form1, que representa la interfaz gráfica principal de la aplicación de procesamiento de imágenes.
 * Fecha de creación: 20/03/2024
 * Autor: Ing. Santiago Alejandro Zuñiga Melo
 * Descripción: Esta es mi solución a la prueba 1 para la entrevista en el cargo de analista de ingeniería para la empresa Intecol.
```

```
 *       La aplicacion permite cargar un conjunto de imagenes obtenidas por una
camara OGI para posteriormente procesarlas y asi usarlas para el calculo del
volumen
 *       de gas filtrado.
 */

using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Windows.Forms;
using OpenCvSharp;
using OpenCvSharp.Extensions;


namespace Prueba1
{
  /*
  * Clase: Form1
  * Descripción: Esta clase representa el formulario principal de la aplicación.
Contiene métodos para cargar una imagen BMP,
  *       procesarla y realizar varias operaciones de procesamiento de imágenes,
como aplicar filtros, detectar bordes
  *       y contar píxeles negros mediante segmentación.
  */
  public partial class Form1 : Form
  {
    // Lista que almacenará las rutas de los archivos BMP
    private List<string> archivos = new List<string>();
    // Lista que almacenará los volúmenes relativos calculados
    private List<double> volumenesRelativos = new List<double>();
    // Variable que almacenará la imagen de referencia
    private Bitmap imagenReferencia = null;
    // Índice utilizado para rastrear el archivo actual que se está procesando
    private int indiceProcesar = 0;
    // Temporizador utilizado para procesar imágenes periódicamente
    private Timer timer;

    /*
     * Constructor: Form1
     * Descripción: Inicializa una nueva instancia de la clase Form1.
     *       Configura el DataGridView y el temporizador.
     */
    public Form1()
    {
      InitializeComponent();
      ConfigurarDataGridView(); // Configurar el DataGridView
```

```
      ConfigurarTimer(); // Configurar el temporizador
    }

    /* Método: ConfigurarDataGridView
     * Descripción: Configura el DataGridView con las columnas necesarias.
     */

    private void ConfigurarDataGridView()
    {
      dataGridView1.ColumnCount = 2; // Establecer el número de columnas en el
DataGridView
      // Establecer los encabezados de columna
      dataGridView1.Columns[0].Name = "Ruta";
      dataGridView1.Columns[1].Name = "Volumen";
      // Ajustar el modo de tamaño automático de las columnas para que se ajusten al
contenido
      dataGridView1.Columns[0].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
      dataGridView1.Columns[1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
    }

    /* Método: ConfigurarTimer
     * Descripción: Configura el temporizador con un intervalo de 2 segundos y
asocia el evento Tick con Timer_Tick.
     */

    private void ConfigurarTimer()
    {
      timer = new Timer();
      timer.Interval = 2000; // Intervalo de 2000 milisegundos (2 segundos)
      timer.Tick += Timer_Tick; // Asociar el evento Tick del temporizador con
Timer_Tick
    }

    /* Evento: Timer_Tick
     * Descripción: Manejador de eventos para el evento Tick del temporizador.
     *       Procesa la siguiente imagen en la lista o detiene el temporizador si
todas las imágenes han sido procesadas.
     * Parámetros:
     *   - sender: El objeto que generó el evento.
     *   - e: Argumentos del evento.
     */
    private void Timer_Tick(object sender, EventArgs e)
    {
      if (indiceProcesar < archivos.Count)
      {
        ProcesarSiguienteImagen(); // Procesar la siguiente imagen en la lista
```

```
      }
      else
      {
        // Detener el temporizador si ya se procesaron todas las imágenes
        timer.Stop();
        MessageBox.Show("Todas las imágenes han sido procesadas.");
      }
    }

    /* Método: ProcesarSiguienteImagen
     * Descripción: Procesa la siguiente imagen en la lista de archivos.
     */

    private void ProcesarSiguienteImagen()
    {
      Bitmap imagenActual = new Bitmap(archivos[indiceProcesar]);
      double volumenRelativo = CalcularVolumenRelativo(imagenActual,
indiceProcesar);
      volumenesRelativos.Add(volumenRelativo);

      string rutaImagen = archivos[indiceProcesar];
      dataGridView1.Rows.Add(rutaImagen, volumenRelativo); // Agregar la ruta y
el volumen a DataGridView

      MostrarImagen(indiceProcesar, pictureBox1); // Mostrar la imagen actual en
pictureBox1

      indiceProcesar++; // Incrementar el índice para procesar la próxima imagen
    }
    /* Método: Cargar_Btn_Click
     * Descripción: Manejador de eventos para el botón de carga de imagen.
     *       Abre un diálogo para seleccionar una imagen BMP y la carga en una
PictureBox.
     * Parámetros:
     *   - sender: El objeto que generó el evento.
     *   - e: Argumentos del evento.
     */
    private void Cargar_Button_Click(object sender, EventArgs e)
    {
      FolderBrowserDialog folderBrowserDialog = new FolderBrowserDialog();

      if (folderBrowserDialog.ShowDialog() == DialogResult.OK)
      {
        string carpetaSeleccionada = folderBrowserDialog.SelectedPath;
        archivos = Directory.GetFiles(carpetaSeleccionada, "*.bmp").ToList();
        MessageBox.Show($"{archivos.Count} archivos BMP cargados desde la
carpeta seleccionada.");

        // Ordenar los archivos según el número en el nombre del archivo
```

```
        archivos.Sort((a, b) =>

Convert.ToInt32(Path.GetFileNameWithoutExtension(a).Replace("image(",
"").Replace(")", ""))

.CompareTo(Convert.ToInt32(Path.GetFileNameWithoutExtension(b).Replace("imag
e(", "").Replace(")", ""))));

        // Cargar la imagen de referencia (imagen número 22)
        int indiceImagenReferencia = archivos.FindIndex(f =>
f.Contains("image(22)"));
        if (indiceImagenReferencia == -1)
        {
          MessageBox.Show("No se encontró la imagen número 22 en los archivos
cargados.");
          return;
        }
        else
        {
          imagenReferencia = new Bitmap(archivos[indiceImagenReferencia]);
          MostrarImagenReferencia(imagenReferencia);
        }
      }
    }

    /* Método: MostrarImagenReferencia
     * Descripción: Muestra la imagen de referencia en el PictureBox pictureBox2.
     * Parámetros:
     *   - imagenReferencia: La imagen de referencia a mostrar.
     */

    private void MostrarImagenReferencia(Bitmap imagenReferencia)
    {
      pictureBox2.Image = (Bitmap)imagenReferencia.Clone();
    }
    /* Método: MostrarImagen
     * Descripción: Muestra la imagen en el PictureBox especificado.
     * Parámetros:
     *   - indice: El índice de la imagen en la lista de archivos.
     *   - pictureBox: El PictureBox en el que se mostrará la imagen.
     */

    private void MostrarImagen(int indice, PictureBox pictureBox)
    {
      if (indice < archivos.Count && File.Exists(archivos[indice]))
      {
        Bitmap imagen = new Bitmap(archivos[indice]);
        pictureBox.Image = (Bitmap)imagen.Clone();
```

```
      }
      else
      {
        MessageBox.Show($"No se pudo encontrar la imagen: {archivos[indice]}");
      }
    }

    /* Método: Calcular_Button_Click
     * Descripción: Manejador de eventos para el botón de cálculo de volúmenes.
     *       Inicia el temporizador para comenzar el procesamiento de imágenes cada
2 segundos.
     * Parámetros:
     *   - sender: El objeto que generó el evento.
     *   - e: Argumentos del evento.
     */
    private void Calcular_Button_Click(object sender, EventArgs e)
    {
      if (archivos.Count == 0 || imagenReferencia == null)
      {
        MessageBox.Show("Por favor carga los archivos primero.");
        return;
      }

      // Iniciar el temporizador para comenzar el procesamiento de imágenes cada 2
segundos
      timer.Start();
    }

    /* Método: CalcularVolumenRelativo
     * Descripción: Calcula el volumen relativo de la imagen actual.
     * Parámetros:
     *   - imagenActual: La imagen actual a procesar.
     *   - indiceImagen: El índice de la imagen en la lista de archivos.
     * Retorna: El volumen relativo calculado.
     */
    private double CalcularVolumenRelativo(Bitmap imagenActual, int
indiceImagen)
    {
      double volumenReferencia = 5.0; // Volumen de referencia en litros (imagen
22)
      double factorEscala;

      if (indiceImagen < 20)
      {
        factorEscala = 0.0; // Si es una de las primeras 20 imágenes, el volumen es 0
      }
      else
      {
```

```
        // Factor de escala para el volumen perdido
        factorEscala = Math.Max((indiceImagen - 21) * 0.0001, 0);
      }

      Mat imagenMat =
OpenCvSharp.Extensions.BitmapConverter.ToMat(imagenActual);
      Mat imagenGrayscale = new Mat();
      Cv2.CvtColor(imagenMat, imagenGrayscale, ColorConversionCodes.BGR2GRAY);
      Cv2.GaussianBlur(imagenGrayscale, imagenGrayscale, new
OpenCvSharp.Size(3, 3), 0);
      Mat imagenBinarizada = new Mat();
      Cv2.AdaptiveThreshold(imagenGrayscale, imagenBinarizada, 255,
AdaptiveThresholdTypes.GaussianC, ThresholdTypes.Binary, 11, 2);
      OpenCvSharp.Point[][] contornos;
      HierarchyIndex[] jerarquia;
      Cv2.FindContours(imagenBinarizada, out contornos, out jerarquia,
RetrievalModes.External, ContourApproximationModes.ApproxSimple);

      double areaTotal = 0;
      foreach (var contorno in contornos)
      {
        double areaContorno = Cv2.ContourArea(contorno);
        if (areaContorno > 100)
        {
          areaTotal += areaContorno;
        }
      }

      double volumenPerdido = areaTotal * factorEscala;

      if (indiceImagen == 21)
      {
        volumenPerdido = volumenReferencia;
      }

      MostrarImagenProcesada(imagenBinarizada);

      return volumenPerdido;
    }

    /* Método: MostrarImagenProcesada
     * Descripción: Muestra la imagen procesada en el PictureBox pictureBox2.
     * Parámetros:
     *   - imagenProcesadaMat: La imagen procesada en formato Mat.
     */
    private void MostrarImagenProcesada(Mat imagenProcesadaMat)
    {
      Bitmap imagenProcesadaBitmap =
OpenCvSharp.Extensions.BitmapConverter.ToBitmap(imagenProcesadaMat);
```

```
      pictureBox2.Image = (Bitmap)imagenProcesadaBitmap.Clone();
    }

    /* Método: Exportar_Button_Click
     * Descripción: Manejador de eventos para el botón de exportar resultados.
     *       Exporta los datos del DataGridView como un archivo CSV.
     * Parámetros:
     *   - sender: El objeto que generó el evento.
     *   - e: Argumentos del evento.
     */
    private void Exportar_Button_Click(object sender, EventArgs e)
    {
      if (dataGridView1.Rows.Count == 0)
      {
        MessageBox.Show("No hay datos para exportar.");
        return;
      }

      SaveFileDialog saveFileDialog = new SaveFileDialog();
      saveFileDialog.Filter = "Archivos CSV|*.csv";
      saveFileDialog.Title = "Guardar resultados";

      if (saveFileDialog.ShowDialog() == DialogResult.OK)
      {
        try
        {
          using (StreamWriter sw = new StreamWriter(saveFileDialog.FileName))
          {
            // Escribir encabezados de columna
            foreach (DataGridViewColumn column in dataGridView1.Columns)
            {
              sw.Write(column.HeaderText + ",");
            }
            sw.WriteLine();

            // Escribir datos de cada fila
            foreach (DataGridViewRow row in dataGridView1.Rows)
            {
              foreach (DataGridViewCell cell in row.Cells)
              {
                sw.Write(cell.Value + ",");
              }
              sw.WriteLine();
            }

            MessageBox.Show("Resultados exportados correctamente como archivo
CSV.");
          }
```

```
        }
        catch (Exception ex)
        {
          MessageBox.Show($"Error al exportar: {ex.Message}");
        }
      }
    }
  }
}
```