Informe de Uso de la Aplicación de Seguimiento de Lubinas

Propósito del Informe

El propósito de este informe es explicar el desarrollo de la aplicación y así ayudar a su usuario a utilizarla de manera eficaz.

I. Herramientas usadas

- **Lenguaje de Programación**: Python v3.11.12
- **Etiquetado de Imágenes**: CVAT (Computer Vision Annotation Tool)
- **Modelo de Detección y Seguimiento**: YOLOv8 (You Only Look Once version 8)

II. Funcionamiento de la aplicación

La aplicación ha sido desarrollada para analizar videos de un tanque que contiene un laberinto con 4 paredes. Estas paredes tienen ventanas y salidas en sus extremos, delimitando diferentes zonas numeradas. En la zona 1 del tanque se han colocado 5 lubinas, y en la zona 5 hay una botella de comida con una tapa azul.

III. Objetivo de la Aplicación

El objetivo principal de la aplicación es detectar y seguir cada lubina en el tanque, registrando el ID y el tiempo (en segundos) de la primera lubina que cruce una salida para llegar a una nueva zona, hasta que finalmente llegue a la última zona donde se encuentra la comida. En ese momento, el programa se detendrá

IV. Etapas del proyecto

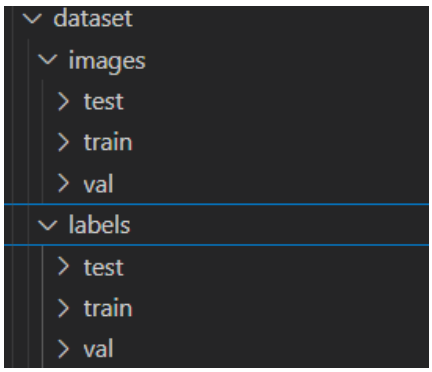
1. Etiquetado de las imágenes
2. Entrenamiento del modelo
3. Codificación de la aplicación y agregación de restricciones para ir mejorando su eficacidad.

1. Etiquetado de las imágenes

El etiquetado de las imágenes se llevó a cabo usando un sample de 2000 fotogramas provenientes de los 128 videos disponibles. Se utilizó CVAT para dibujar una caja (bounding box) alrededor de cada lubina en el tanque en cada fotograma, ayudando al modelo a reconocer las lubinas y distinguirlas de los reflejos en el agua y otros objetos en el video.

2. Entrenamiento del modelo

Tras el etiquetado, se entrenó el modelo YOLOv8 con estas imágenes. Los fotogramas se importaron desde CVAT y se organizaron en una carpeta llamada ‘dataset’, que contiene subcarpetas para ‘images’ y ‘labels’. Estas subcarpetas, a su vez, se dividieron en ‘Train’, ‘Validation’, y ‘Test’ para los procesos de entrenamiento y evaluación del modelo.



Se creó un script para lanzar el entrenamiento del modelo, ajustando parámetros y evaluando la precisión de este.

3. Codificación de la aplicación

a. **Importación de Librerías**: Se importaron todas las librerías necesarias para el funcionamiento del programa.

```
#this version opens videos from a folder and treats them one after the other
import os
import cv2 as cv
import numpy as np
import pandas as pd
from ultralytics import YOLO
import time
```

b. **Cálculo de Ángulos**: Se escribió una función para calcular el ángulo de una recta a partir de sus coordenadas, esencial para filtrar y mantener solo las líneas horizontales correspondientes a las paredes del tanque.

```
def line_angle(x1, y1, x2, y2):
    return np.degrees(np.arctan2(y2 - y1, x2 - x1))
```

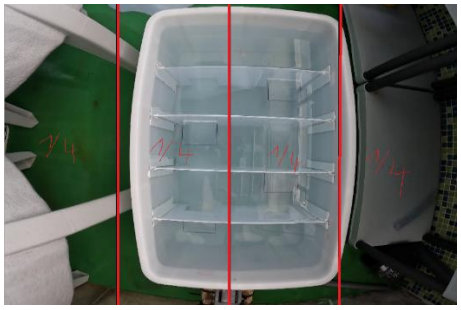
c. **Extracción de Fotogramas**: Una función para extraer fotogramas específicos del video. Esta función nos será útil para extraer las posiciones de las líneas de delimitación y la posición de la botella de comida en un fotograma dado del video y aplicar estos valores a todos los fotogramas del video.

```
def extract_frame(video, frame_number):
    if not video.isOpened():
        print("Error: Could not open video.")
        exit()
    video.set(cv.CAP_PROP_POS_FRAMES, frame_number)
    ret, frame = video.read()
    if ret:
        cv.imwrite('frame_5.jpg', frame)
    else:
        print(f"Error: Could not retrieve frame {frame_number}.")
    video.set(cv.CAP_PROP_POS_FRAMES, 0)
    return frame
```

d. **Cálculo de Coordenadas del Tanque**: Análisis de la posición del tanque en los videos para determinar las coordenadas aproximadas. Dado que esta posición queda céntrica a lo largo de los videos, se aproximó su posición, dividiendo una imagen del video.

```python
def tank_coordinates(frame):
    height, width,_ = frame.shape
    x1_tank = width / 4
    y1_tank = height / 12
    x2_tank, y2_tank = width - x1_tank, height - y1_tank

    return x1_tank, y1_tank,x2_tank, y2_tank
```



e. **Detección de Líneas**: Usando los métodos GaussianBlur y Canny para extraer curvas y HoughLinesP para especificar todas las líneas de la imagen.

```python
def extract_lines(frame):
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    blurred = cv.GaussianBlur(gray, (5, 5), 0)
    edges = cv.Canny(blurred, 50, 150, apertureSize=3)
    lines = cv.HoughLinesP(edges, 1, np.pi / 180, threshold=150, minLineLength=50, maxLineGap=10)
    return lines
```

f. **Filtrado de Líneas:** Un algoritmo para mantener esencialmente las líneas de las paredes del laberinto, basado en ángulos y posiciones dentro del tanque.

```python
def detect_line_areas(df, lines, x1_tank, y1_tank, x2_tank, y2_tank):
    if lines is not None:
        for line in lines:
            x1, y1, x2, y2 = line[0]
            angle = line_angle(x1, y1, x2, y2)
            if -7 <= angle <= 7:
                if (x1 > x1_tank and x2 < x2_tank) and (y1 > y1_tank and y2 < y2_tank):
                    df = pd.concat([df, pd.DataFrame([{"x1": x1, "y1": y1, "x2": x2, "y2": y2}])], ignore_index=True)
    df_sorted = df.sort_values(by="y1")
    yy = 0
    first_row = df_sorted['y1'].iloc[0]
    last_row = df_sorted['y1'].iloc[-1]
    j = (last_row - first_row) / 4
    areas = pd.DataFrame(columns=['x1', 'y1', 'x2', 'y2'])
    for index, row in df_sorted.iterrows():
        if np.abs(row['y1'] - yy) > j:
            yy = row['y1']
            areas = pd.concat([areas, df_sorted[df_sorted['y1'] == yy]], ignore_index=True)
    areas = areas.sort_values(by="y1")
    return areas
```

g. **Detección de la Comida**: Un script para detectar la botella de comida con la tapa azul y devolver su fotograma con coordenadas

```python
def detect_blue_cap(frame, x1_tank, width):
    # Detect the blue cap (bottle) using color segmentation
    hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
    lower_blue = np.array([100, 150, 0])
    upper_blue = np.array([140, 255, 255])
    mask = cv.inRange(hsv, lower_blue, upper_blue)

    # Find contours for the detected blue areas
    contours, _ = cv.findContours(mask, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

    # List to store positions of bounding boxes
    positions = []

    for contour in contours:
        # Compute the bounding box for the contour
        x, y, w, h = cv.boundingRect(contour)
        # Filter small contours
        if w * h > 500 and (x > x1_tank and w + x < width - x1_tank):
            # Draw bounding box around detected blue cap
            cv.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
            cv.putText(frame, 'Bottle', (x, y - 10), cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
            # Store the position of the bounding box
            positions.append((x, y, w, h))

    return frame, positions
```

h. **Procesamiento de Videos**: Importación del modelo entrenado y análisis de los videos, generando ficheros de texto con resultados específicos. Se especifica que esos ficheros deben estar colocados en el mismo camino que sus asociado pero a partir de la carpeta local.

```python
# Load the YOLO model and video
model = YOLO("D:/Chris/Lubinas/runs/train/exp/weights/best.pt")


# Define the base directory to replace in the video paths
base_path_to_replace = "L:\\ENVIROBASS_NOE\\TESTS\\APRENDIZAJE"

# Get the current working directory
current_dir = os.getcwd()

# Read video paths from links.txt
with open("video_paths.txt", "r") as file:
    video_paths = [line.strip() for line in file.readlines()]

for video_path in video_paths:
    video = cv.VideoCapture(video_path)
    if not video.isOpened():
        print(f"Error opening video file {video_path}")
        continue
    # Replace the base part of the path
    relative_path = video_path.replace(base_path_to_replace, '').strip("\\")
    video_path = os.path.join(current_dir, relative_path)


    # Extract the directory path and video name from the video_path
    video_dir = os.path.dirname(video_path)
    video_name = os.path.splitext(os.path.basename(video_path))[0]

    # Ensure the directory exists
    if not os.path.exists(video_dir):
        os.makedirs(video_dir)
```

i. **Generación de Ficheros de Resultados**: Creación de varios ficheros (resultados.txt, posLineas.txt, BoundingBoxes.txt, Botella.txt) para cada video analizado, conteniendo demás de los resultados que hemos comentado antes, informaciones relevante como la posición de la líneas de delimitación, la posición de las bounding boxes and la posición de la botella de comida respectivamente.

```
# Define paths for output files
resultados_path = os.path.join(video_dir, f"{video_name}_resultados.txt")
position_lineas_path = os.path.join(video_dir, f"{video_name}_posLineas.txt")
bounding_boxes_path = os.path.join(video_dir, f"{video_name}_boundingBoxes.txt")
botella_position_path = os.path.join(video_dir, f"{video_name}_botella.txt")
with open(resultados_path, "w") as zones_file, \
    open(position_lineas_path, "w") as lines_file, \
    open(bounding_boxes_path, "w") as boxes_file, \
    open(botella_position_path, "w") as bottle_file:
```

V. Instrucciones para el Uso de la Aplicación

1. **Preparar el Video**: Asegúrate de tener un video del tanque listo para su análisis.
2. **Ejecutar la Aplicación**: Abre la aplicación y selecciona el video para iniciar el análisis.
3. **Monitoreo del Proceso**: La aplicación comenzará a detectar y seguir las lubinas automáticamente. Puedes ver el progreso en la interfaz de la aplicación.
4. **Revisar Resultados**: Una vez que el programa se detenga, revisa el fichero resultados.txt para ver el ID y el tiempo de las lubinas que cruzaron cada salida.