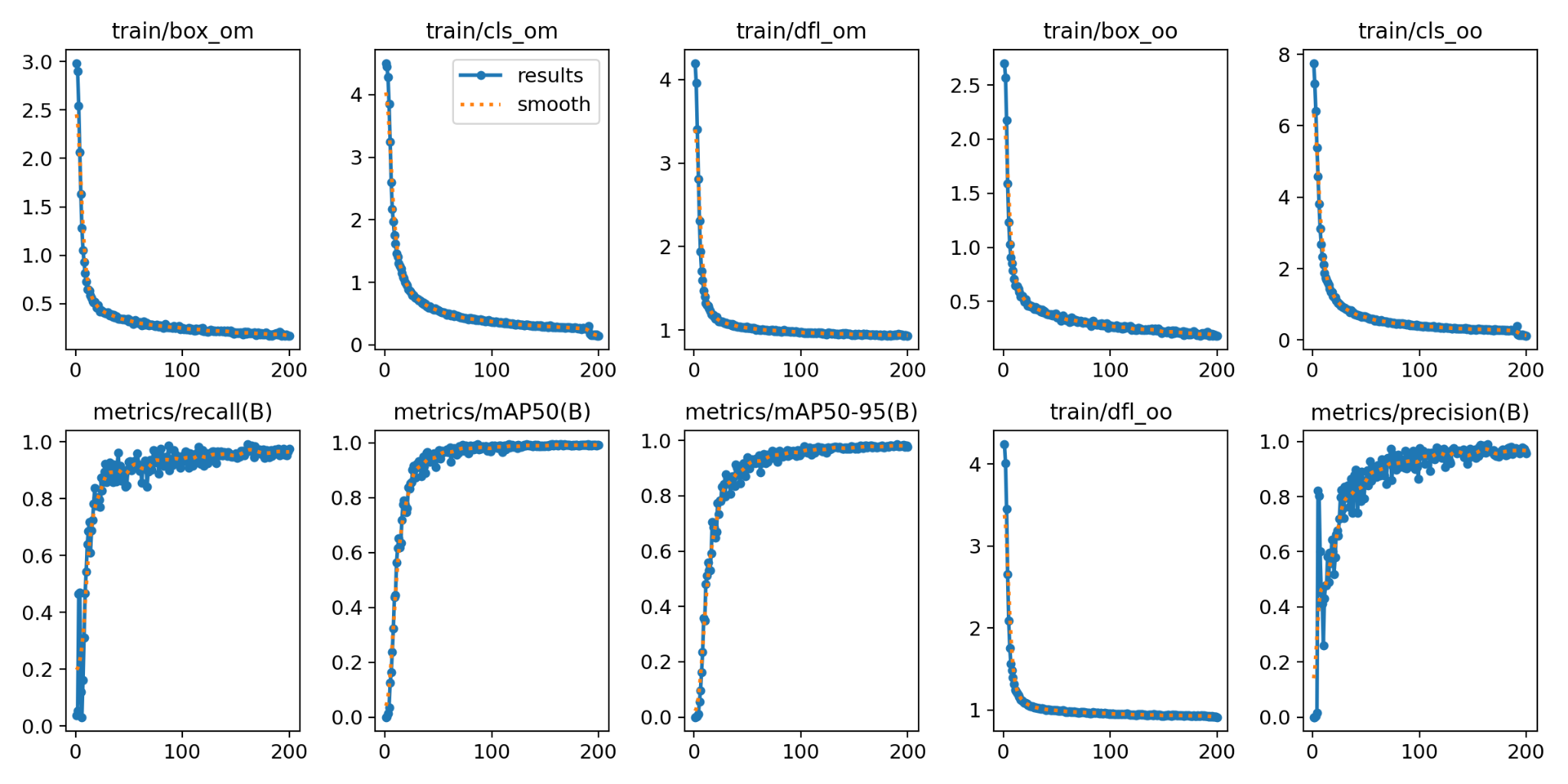
**Hazmat:** For detecting emergency signals, we use **YOLOv10**, a deep learning model based on convolutional neural networks that performs object detection in a single pass (single-shot detection). To train this model, we use a dataset consisting of a set of images and a set of annotations. The annotations are stored in a text file specifying the coordinates of the objects to be detected within the provided image. Therefore, each image in our dataset has its own annotation file.

Next, we use **Google Colab**, a development environment where we train our model using a cloud-based GPU.

During the training process, the model was evaluated using the following graphs:



Afterward, we used this model in a test code using **OpenCV**. In this code, we loaded an image, applied the trained model, and displayed the results.

When displaying the results, instead of showing the signal names, the model displays numbers. This is because before training the model, the data.yaml file was not modified, and in the names list, only numbers were present. On the other hand, the numbers inside parentheses represent the class ID. We later use this data to display the correct signal name when applying real-time detection through the camera. 

The percentage number that appears after the class ID represents the confidence score, which indicates how certain the model is that the detected image corresponds to the assigned class.

After testing the model on an image, we loaded it into the camera to perform real-time detection. If something is detected, a snapshot is automatically taken, displaying the detected signal and its bounding box.

This is the code we used:

import cv2

from ultralytics import YOLO

import cvzone

import time

from datetime import datetime

import keyboard

cap = cv2.VideoCapture(1)

model = YOLO(r"C:\\Users\\marth\\OneDrive\\Desktop\\venv\\modelo2\\best.pt")

nombre = ["Blasting Agents", "Corrosive", "Oxidicer", "Oxigen", "Poison", "Radioactive", "Spontaneously Combustible",

"Dangerous when wet", "Explosives", "Flammables gas", "Flammable solid", "Fuel Oil", "Inhalation hazard",

"Non flammable Gas", "Organic Peroxide"]

ultimo\_tiempo = time.time()

while True:

ret, frame = cap.read()

if not ret:

print("Error al capturar la cámara")

break

results = model(frame)

for info in results:

for box in info.boxes:

x1, y1, x2, y2 = box.xyxy[0].numpy().astype(int)

confidence = round(box.conf[0].item() \* 100, 2)

classd = int(box.cls[0].item())

name = nombre[classd] if classd < len(nombre) else f"Clase {classd}"

if confidence > 90:

cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 255), 3)

cvzone.putTextRect(

frame, f"{name} ({classd}): {confidence}%",

(x1 + 8, y1 - 12),

thickness=2, scale=1.2,

colorR=(255, 0, 255), colorT=(255, 255, 255)

)

if keyboard.is\_pressed("F12"):

ruta\_guardado = f"detecciones/{name}.jpg"

cv2.imwrite(ruta\_guardado, frame)

print(f"Captura guardada: {ruta\_guardado}")

if time.time() - ultimo\_tiempo >= 3:

fecha\_hora\_actual = datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")

ruta\_guardado = f"detecciones/{name}\_{fecha\_hora\_actual}.jpg"

cv2.imwrite(ruta\_guardado, frame)

print(f"Captura guardada: {ruta\_guardado}")

ultimo\_tiempo = time.time()

cv2.imshow("Detección", frame)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

Libraries used:

### **OpenCV (cv2)**

OpenCV is a computer vision library used for image and video processing. It enables real-time applications like object detection, face recognition, and motion tracking. In this code, it captures video, draws detection boxes, displays results, and saves detected images.

### **Ultralytics YOLO**

YOLO is a deep learning model for real-time object detection. It analyzes images in a single pass, making it fast and efficient. Here, it detects hazardous material signs, providing their positions, class IDs, and confidence scores.

### **Time**

The time module manages time-related tasks like measuring intervals and adding delays. Here, it ensures images are saved every three seconds, preventing excessive captures.

### **Datetime**

The datetime module handles date and time operations. It generates timestamps for saved images, ensuring each has a unique name for better organization.

### **Keyboard**

The keyboard module detects key presses. In this code, pressing "F12" manually saves an image, allowing the user to capture specific detections instantly.

Finally, this is how the captures taken by the detector look:

As we can see, the detector is relatively accurate; however, it makes some incorrect detections because it finds colors similar to those of the signs in the background. It also confuses similar signs, such as Oxygen and Oxidizer. To improve this detection, we will retrain our model with more images and more training epochs so that it becomes more accurate over time. We will also experiment with the confidence percentage to avoid any false detections.

**Qr code detector:**

This code captures video from a camera, detects QR codes in real-time, and stores the data from newly detected QR codes into a text file. It also provides functionality for saving images of QR code detections based on specific conditions, such as pressing a key or detecting multiple QR codes. Additionally, the code uses **OpenCV** for video processing and **pyzbar** to decode the QR codes.

First we import Libraries and declare camera:

import cv2

import numpy as np

from pyzbar import pyzbar

from datetime import datetime

import os

from func import \*

import keyboard

cap = cv2.VideoCapture(1)

We define the file path qr\_file, where detected QR codes will be saved:

qr\_file = "qr\_detections/qr\_code.txt"

Later is used a custom function cargar\_codigos\_existentes to load previously saved QR codes from the qr\_code.txt file. This prevents saving the same code multiple times.

def cargar\_codigos\_existentes(b):

if not os.path.exists(b):

return set()

with open(b, "r", encoding = "utf-8") as f:

return set(line.strip() for line in f)

codigos\_guardados = cargar\_codigos\_existentes(qr\_file)

Then we enter an infinite loop where frames are continuously captured from the camera using cap.read(). The frame is then passed to the prepare\_image function, which processes the image rezising it and converting it to gray scale so the detection can be done more precisely:.

def prepare\_image(a):

escala = 1.5

image= cv2.resize(a, (0, 0), fx=escala, fy=escala)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

sharp = cv2.convertScaleAbs(gray, alpha=1.5, beta=0)

Loop:

while True:

ret, image = cap.read()

prepare\_image(image)

if not ret:

print("Error al acceder a la camara")

break

We use pyzbar.decode(image) to detect and decode any QR codes present in the captured frame:

detection = pyzbar.decode(image)

For each detected QR code, we extract the coordinates of the bounding box (x, y, w, h) and draw a rectangle around the detected code. We also decode the QR code data into a text string:

for i in range(len(detection)):

x, y, w, h = detection[i].rect

cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)

text = detection[i].data.decode("utf-8")

If the "F11" key is pressed, an image of the current frame is saved with a timestamp in the filename:

if keyboard.is\_pressed("F11"):

fecha\_hora\_actual = datetime.now().strftime("%H-%M-%S")

ruta\_guardado = f"qr\_detections/presssed---{fecha\_hora\_actual}.jpg"

cv2.imwrite(ruta\_guardado, image)

If 8 or more QR codes are detected in a frame, an image is saved with a timestamp in the filename:

if len(detection) >= 8:

fecha\_hora\_actual = datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")

ruta\_guardado = f"qr\_detections/8---{fecha\_hora\_actual}.jpg"

cv2.imwrite(ruta\_guardado, image)

If the detected QR code is not already saved, it is added to the text file qr\_code.txt and the list of saved codes. The image is also saved, and a message is printed indicating a new code has been saved:

if text not in codigos\_guardados:

with open(qr\_file, "a", encoding="utf-8") as f:

f.write(text + "\n")

codigos\_guardados.add(text)

cv2.imwrite(ruta\_guardado, image)

print(f"Nuevo codigo guardo: {text}")

Finally we display the decoded text of the QR code on the image, just above the bounding box and then we show it:

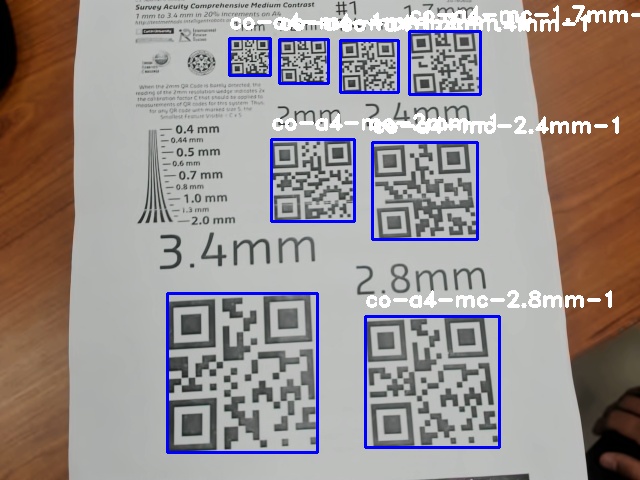
cv2.imshow("qr", image)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cv2.destroyAllWindows()

This is how the images are saved after the detection:



**CO2 Sensor:**

In this code we used the Adafruit SCD30 sensor to read CO2, temperature and humidity values, the prints those values on the serial monitor, then those values are received from the raspberry pi and printed out on its terminal.

First, we import Libraries:

#include <Adafruit\_SCD30.h>

#include <Servo.h>

Then we start the serial communication, and initializes the SCD30 sensor:

void setup() {

Serial.begin(115200);

Serial.println("Sensor test");

if (!scd30.begin()) {

Serial.println("Failed to find sensor");

while (1) delay(1000);

}

Serial.println("SCD30 found");

Serial.println("Measurement interval: " + String(scd30.getMeasurementInterval()) + " seconds");

}

After that, we repetitively check for the sensor data and print it on the serial port:

void loop() {

if (scd30.dataReady()) {

Serial.println("Data available:");

if (!scd30.read()) {

Serial.println("Error al obtener data");

return;

}

Serial.print("CO2: ");

Serial.print(scd30.CO2, 1);

Serial.print(" ppm, ");

Serial.print("Temperature: ");

Serial.print(scd30.temperature, 1);

Serial.print(" °C, ");

Serial.print("Humidity: ");

Serial.print(scd30.relative\_humidity, 1);

Serial.println(" %");

Serial.println("----------------------");

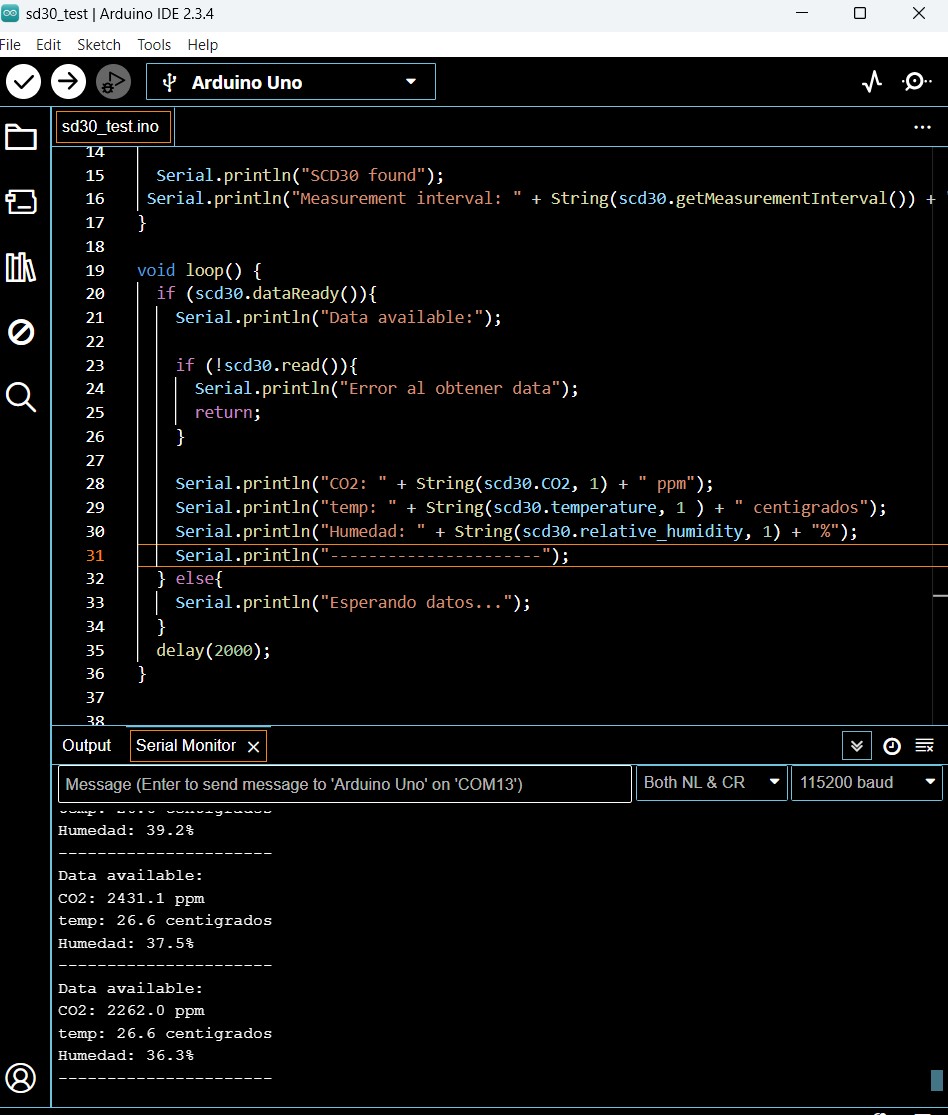
}

}

This code simply sends data to the Raspberry Pi through the serial port. Right now, we are only printing the information in the terminal, but in the future, we plan to store it in a variable so that we can display it in the interface and assign it a specific function.

Our long-term goal is not only to show real-time environmental data but also to analyze and compare environmental changes over a specific time interval. This will allow us to track variations and gain deeper insights into the collected data.

Finally, this is the information that the sensor read after some tests:



**Motion Tracker:**

This code implements a motion detection system using OpenCV. It captures video frames from a camera, detects movement by comparing consecutive frames, and highlights detected motion with green rectangles.

First we import the library  **Opencv** and initialize the camera:

import cv2

cam = cv2.VideoCapture(1, cv2.CAP\_DSHOW)

Then it captures two consecutive frames to compare changes:

ret, frame1 = cam.read()

ret, frame2 = cam.read()

Detection Loop:

while cam.isOpened():

ret, frame2 = cam.read()

if not ret:

Break

Then we process frame differences:

diff = cv2.absdiff(frame1, frame2)

gray = cv2.cvtColor(diff, cv2.COLOR\_BGR2GRAY)

blur = cv2.GaussianBlur(gray, (5, 5), 0)

\_, thresh = cv2.threshold(blur, 40, 255, cv2.THRESH\_BINARY)

dilated = cv2.dilate(thresh, None, iterations=3)

contours, \_ = cv2.findContours(dilated, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

Later, the box is drawed:

for contour in contours:

(x, y, w, h) = cv2.boundingRect(contour)

if cv2.contourArea(contour) < 700:

continue

cv2.rectangle(frame1, (x, y), (x + w, y + h), (0, 255, 0), 2)

Finally display the result and release the camera:

cv2.imshow('motion tracker', frame1)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

frame1 = frame2.copy()

cam.release()

cv2.destroyAllWindows()

This tracker is not as efficient as we would like because it is highly sensitive, often detecting image noise as motion. To improve its accuracy, we plan to experiment with different frame comparison techniques and explore the implementation of **Optical Flow**, which provides more precise motion detection.

So far, we have been testing the **Lucas-Kanade method**, and in the future, we aim to integrate its algorithm for a more reliable tracker. The main challenge will be minimizing false detections caused by noise. To achieve this, we will focus on selecting the right features for the objects we want to track, ensuring the program distinguishes between actual movement and irrelevant background changes.