```
##Camera calibration
import numpy as np
import cv2 as cv
import glob

# termination criteria
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

images = glob.glob(’*.jpg’)

for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (7,6), None)

    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)

        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners2)

        # Draw and display the corners
        cv.drawChessboardCorners(img, (7,6), corners2, ret)
        cv.imshow(’img’, img)
        cv.waitKey(500)

cv.destroyAllWindows()

ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

# undistort
dst = cv.undistort(img, mtx, dist, None, newcameramtx)

# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite(’calibresult.png’, dst)

##Data augmentation
#Filters
def get_random_number(inf_limit, superior_limit):
    return random.uniform(inf_limit, superior_limit)

#Filtro Gaussiano
```

```
def gaussian_blur(image, sigma):
    return image.filter(ImageFilter.GaussianBlur(radius=sigma))

#Color Jitter
def color_jitter(image):
    color_jitter = transforms.ColorJitter(
        brightness=(0.2, 1.0),  # Menor luz
        contrast=(0.2, 1.0),    # Contraste normal
        saturation=(0.2, 1.0),  # Menor saturación
        hue=(0.0, 0.1)          # Menor rango para HUE
    )
    transformed_image = color_jitter(image)
    return transformed_image
#Encoding Quality
def encoding_quality(image, quality):
    buffer = BytesIO()
    image.save(buffer, format="JPEG", quality=quality)
    buffer.seek(0)
    return Image.open(buffer)

#Unión de todos los filtros
def GB_EQ_CJ_filter(img):
    sigma = get_random_number(0.5, 5)
    img = gaussian_blur(img, sigma)
    quality = int(get_random_number(1, 30))
    img = encoding_quality(img, quality)
    img = color_jitter(img)

    return img

def apply_filters_on_folder(origin_img_folder, target_img_folder, filter):
    images = os.listdir(origin_img_folder)
    for i in range(len(images)):
        full_img_path = os.path.join(origin_img_folder, images[i])
        img = Image.open(full_img_path)
        img = filter(img)
        #SaveImage
        full_target_path = os.path.join(target_img_folder, images[i])
        img.save(full_target_path)
    print("Done applying filter")

def apply_darken_on_folder(origin_img_folder, target_img_folder, filter):
    images = os.listdir(origin_img_folder)
    for i in range(len(images)):
        full_img_path = os.path.join(origin_img_folder, images[i])
        img = filter(full_img_path)
        #SaveImage
        full_target_path = os.path.join(target_img_folder, images[i])
        img.save(full_target_path)
    print("Done applying filter")


##Dataset Split:
#Split of dataset
def split_dataset(main_folder, conditioned_db_folder):
```

```
images_folder = os.path.join(main_folder, "images")
labels_folder = os.path.join(main_folder, "labels")

# List directories
images_array = os.listdir(images_folder)
labels_array = os.listdir(labels_folder)

# Calculate the number of elements for training and validation
train = int(len(images_array) * 0.7)  # 70% for training
val = len(images_array) - train

# Select indexes for training and validation
train_indexes = random.sample(range(len(images_array)), train)
val_indexes = [i for i in range(len(images_array)) if i not in train_indexes]

# Print to check indexes
print("Train indexes:", train_indexes)
print("Number of training samples:", len(train_indexes))
print("Validation indexes:", val_indexes)
print("Number of validation samples:", len(val_indexes))

# Create directories for the conditioned database
train_conditioned = os.path.join(conditioned_db_folder, "train")
images_train = os.path.join(train_conditioned, "images")
labels_train = os.path.join(train_conditioned, "labels")
os.makedirs(images_train, exist_ok=True)
os.makedirs(labels_train, exist_ok=True)

val_conditioned = os.path.join(conditioned_db_folder, "val")
images_val = os.path.join(val_conditioned, "images")
labels_val = os.path.join(val_conditioned, "labels")
os.makedirs(images_val, exist_ok=True)
os.makedirs(labels_val, exist_ok=True)

# Copy training images and labels
for i in train_indexes:
    # Images
    source_image = os.path.join(images_folder, images_array[i])
    destiny_image = os.path.join(images_train, images_array[i])
    shutil.copyfile(source_image, destiny_image)

    # Labels
    label_name = os.path.splitext(images_array[i])[0] + ".txt"
    source_label = os.path.join(labels_folder, label_name)
    destiny_label = os.path.join(labels_train, label_name)
    shutil.copyfile(source_label, destiny_label)

# Copy validation images and labels
for i in val_indexes:
    # Images
    source_image = os.path.join(images_folder, images_array[i])
    destiny_image = os.path.join(images_val, images_array[i])
    shutil.copyfile(source_image, destiny_image)

    # Labels
```

```
        label_name = os.path.splitext(images_array[i])[0] + ".txt"
        source_label = os.path.join(labels_folder, label_name)
        destiny_label = os.path.join(labels_val, label_name)
        shutil.copyfile(source_label, destiny_label)

    # Copy classes.txt if it exists
    source_classes = os.path.join(labels_folder, "classes.txt")
    if os.path.exists(source_classes):
        destiny_classes = os.path.join(conditioned_db_folder, "classes.txt")
        shutil.copyfile(source_classes, destiny_classes)

##Model training:
model_m = YOLO('yolo11n.pt')
dataPath = "/content/drive/MyDrive/train_yolo_models/conditioned_db/training.yaml"
trainedModel = model_m.train(data = dataPath, epochs = 45, imgsz= 640)

##YOLO model to tflite format
#Load the yolo model and save it as a tflite model
model =
YOLO('/content/drive/MyDrive/train_yolo_models/conditioned_db/runs/detect/train/weights/best.pt')
model.export(format='tflite')

##Vizualization and centers
def visualize_yolo(index, model):
    cap = cv.VideoCapture(index)
    while True:
        ret, frame = cap.read()
        results = model.predict(frame)

        for result in results:

            annotator = Annotator(frame)

            boxes = result.boxes
            for box in boxes:
                b = box.xyxy[0].numpy()
                c = box.cls
                annotator.box_label(b, model.names[int(c)])
                center_x = int((b[0] + b[2]) / 2)
                center_y = int((b[1] + b[3]) / 2)
                cv.circle(frame, (center_x, center_y), 5, (0, 0, 255), -1)
        img = annotator.result()
        cv.imshow('YOLO V8 Detection', img)
        key = cv.waitKey(1)
        if key == ord('q'):
            break
        return (center_x, center_y)

## Distance Obtainment
#Disparity = x1 -x2 //Coordinates of the center of the object in the image
#Disparity = np.abs(center_x1 - center_x2)
HAOV = 2*arctan((0.07*1920)/245)
HAOV = rad2deg(HAOV/2)
b = 0.07
disparity = np.abs(1151 - 906)
```

```
z = (b*1920)/(2*tan(HAOV)*disparity)
print("Distance to object [m]:" , z)

##Server and client transmision
#Version 2 Imágenes
SERVER_HOST = '192.168.226.171'  # Replace with the server's IP address
SERVER_PORT = 9999  # Port to connect to on the server
Luxes = np.load("Luxes_array.npy")

#Images path:
img_paths = []

# Set up the client socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_HOST, SERVER_PORT))
print("Connected to the server.")

# Use a consistent window name
WINDOW_NAME = "Processed Frame"

# Initialize the OpenCV window
cv.namedWindow(WINDOW_NAME, cv.WINDOW_NORMAL)

# Vamos a intentar con las otras imágenes
main_folder = "new_images"
second_folders = os.listdir(main_folder)
for i in range(len(second_folders)):
    dir_1 = os.path.join(main_folder, second_folders[i])
    third_folders = os.listdir(dir_1)
    for j in range(len(third_folders)):
        dir_2 = os.path.join(dir_1, third_folders[j])
        images = os.listdir(dir_2)
        #Ordenamos de manera numérica
        images = sorted(os.listdir(dir_2), key=lambda x: int(''.join(filter(str.isdigit, x))))

        for k in range(len(images)):
            image_full_path = os.path.join(dir_2, images[k])
            img_paths.append(image_full_path)
            img = cv.imread(image_full_path)

            # Encode the frame as JPEG
            _, encoded_frame = cv.imencode('.jpg', img)
            encoded_frame_bytes = encoded_frame.tobytes()

            # Send frame size followed by the frame data to the server
            frame_size = len(encoded_frame_bytes)
            client_socket.sendall(frame_size.to_bytes(4, byteorder="big") + encoded_frame_bytes)

            # Receive the processed frame size from the server
            data = b''
            while len(data) < 4:
                packet = client_socket.recv(4096)
                if not packet:
                    break
                data += packet
            if not data:
                break

            # Extract the processed frame size
            processed_frame_size = int.from_bytes(data[:4], byteorder="big")
            data = data[4:]

            # Receive the processed frame data based on the processed frame size
            while len(data) < processed_frame_size:
                data += client_socket.recv(4096)

            # Decode the processed frame data
            processed_frame_data = data[:processed_frame_size]
            data = data[processed_frame_size:]
            processed_frame = cv.imdecode(np.frombuffer(processed_frame_data, dtype=np.uint8),
cv.IMREAD_COLOR)

            # Resize the processed frame for display
            processed_frame = cv.resize(processed_frame, (640, 480))

            # Overlay the image name and lux value
            image_name = images[k]
            lux_value = Luxes[k]
            text = f"Name: {image_name}, Lux: {lux_value}"
            cv.putText(
                processed_frame,
                text,
                (10, 30),  # Position of the text (x, y)
                cv.FONT_HERSHEY_SIMPLEX,  # Font type
                0.7,  # Font scale
                (255, 255, 255),  # Font color (white)
                2,  # Thickness
                cv.LINE_AA  # Line type
            )

            # Display the processed frame
            cv.imshow(WINDOW_NAME, processed_frame)
            if cv.waitKey(1) & 0xFF == ord('q'):
                break

client_socket.close()
cv.destroyAllWindows()

#Vamos a usar un modelo en tflite de 16 bits para hacer el proceso más ligero
model = YOLO("models/new_best_yolov8n_float16.tflite")
# model = YOLO("models/new_best_yolov8n.pt")
# Server configuration (Receiver)
HOST = '0.0.0.0'  # Listen on all network interfaces
PORT = 9999       # Port to listen on

# Set up the server socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(1)
print("Waiting for connection...")
```

```
# Accept a client connection
conn, addr = server_socket.accept()
print(f"Connected to {addr}")

data = b''  # Buffer for received data
payload_size = 4  # Size of the header (int indicating the frame size)

while True:
    # Receive the frame size from the transmitter
    while len(data) < payload_size:
        packet = conn.recv(4096)
        if not packet:
            break
        data += packet
    if not data:
        break

    # Extract the frame size
    frame_size = int.from_bytes(data[:payload_size], byteorder="big")
    data = data[payload_size:]

    # Receive the frame data based on the frame size
    while len(data) < frame_size:
        data += conn.recv(4096)

    # Extract and decode the frame data
    frame_data = data[:frame_size]
    data = data[frame_size:]
    frame = cv.imdecode(np.frombuffer(frame_data, dtype=np.uint8), cv.IMREAD_COLOR)

    # Process the frame (for example, converting it to grayscale)
    results = model.predict(frame)

    for result in results:

        annotator = Annotator(frame)
        boxes = result.boxes
        for box in boxes:
            b = box.xyxy[0].numpy()
            c = box.cls
            annotator.box_label(b, model.names[int(c)])
            #center_x = int((b[0] + b[2]) / 2)
            #center_y = int((b[1] + b[3]) / 2)
            #cv.circle(frame, (center_x, center_y), 5, (0, 0, 255), -1)
        img = annotator.result()
        #cv.imshow('YOLO V8 Detection', img)
        #key = cv.waitKey(1)
        #if key == ord('q'):
            #break
    #centers = [center_x, center_y]
    #processed_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    processed_frame = img

    # Encode the processed frame
    _, encoded_processed_frame = cv.imencode('.jpg', processed_frame)
    encoded_processed_frame_bytes = encoded_processed_frame.tobytes()

    # Send the processed frame size and data back to the transmitter
    processed_frame_size_bytes = len(encoded_processed_frame_bytes).to_bytes(4, byteorder="big")
    conn.sendall(processed_frame_size_bytes + encoded_processed_frame_bytes)

# Close the connection
conn.close()
server_socket.close()
```