```python
from ultralytics import YOLO
import cv2, math, cvzone, time, serial
from sort import *
import numpy as np
import matplotlib.pyplot as plt
matplotlib.use('Agg')
import pandas as pd
from datetime import datetime
import streamlit as st

def main():
    st.title("AI Driven Ultrasound Fencing")

    frame_placeholder = st.empty()
    st.sidebar.title("South Zone TN")
    time_now = datetime.now().strftime("%d-%m-%Y")
    st.sidebar.write(f"Date: {time_now}")

    coordinates_placeholder = st.sidebar.empty()
    tracking_placeholder = st.sidebar.empty()
    distance_placeholder = st.sidebar.empty()
    escaped_placeholder = st.sidebar.empty()
    graph_placeholder = st.sidebar.empty()

    distance_hisory = []
    pixel_distance_hisory = []
    all_x_points = []
    all_y_points = []

    camera_option = 0

    classNames = ["Elephant"]
    escaped_animal = []

    ori_width, ori_height = 1280, 720
    target_width, target_height = 180, 180

    esp = None
    try:
        esp = serial.Serial("COM24", 9600, timeout=1)
        time.sleep(2)
        esp.flushInput()
        st.sidebar.success("ESP8266 connected successfully")
    except Exception as e:
        st.sidebar.error(f"Error connecting to ESP8266: {e}")
        esp = None

    def map_coordinates(x, y):
        if camera_option == 0:
            new_x = target_width - int((x * target_width) / ori_width)
        else:
            new_x = int((x * target_width) / ori_width)
        new_y = int((y * target_height) / ori_height)
        return new_x, new_y

    def process_esp_data():
        current_distance = None
        if esp and esp.in_waiting > 0:
            try:
                line = esp.readline().decode('utf-8', errors='replace').strip()
                if line.startswith("DIST:"):
                    try:
                        distance = float(line[5:])
                        current_distance = distance
                    except ValueError:
                        pass
            except Exception as e:
```

```
                st.sidebar.warning(f"Error reading from ESP8266: {e}")
                esp.flushInput()
        return current_distance

    model = YOLO('../../YOLO_weights/best_hand_n.pt')
    tracker = Sort(max_age=40, min_hits=3, iou_threshold=0.3)
    boundary_line = [0, 450, 1280, 450]

    cap = cv2.VideoCapture(camera_option)
    cap.set(3, ori_width)
    cap.set(4, ori_height)

    last_detection_time = time.time()
    default_position_sent = False
    current_distance = None

    run_status = True

    while run_status:
        all_x_points = []
        all_y_points = []
        distance = process_esp_data()
        if distance is not None:
            current_distance = distance

        success, img = cap.read()
        if not success:
            st.error("Failed to capture frame from camera")
            break

        results = model(img, stream=True)
        dets = np.empty((0, 5))
        object_detected = False
        cur_cls = None

        for i in results:
            boxes = i.boxes
            for j in boxes:
                x1, y1, x2, y2 = j.xyxy[0]
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                w, h = x2 - x1, y2 - y1
                cx, cy = x1 + w // 2, y1 + h // 2

                conf = math.ceil((j.conf[0] * 100)) / 100
                cls = classNames[int(j.cls[0])]

                if cls in ["Elephant"] and conf >= 0.6:
                    cur_arr = np.array([x1, y1, x2, y2, conf])
                    dets = np.vstack((dets, cur_arr))
                    cur_cls = cls
                    object_detected = True
                    last_detection_time = time.time()
                    cvzone.putTextRect(img, f"conf:{conf}", (max(0, x1), max(30, y2 + 40)),
offset=2)

        result_tracker = tracker.update(dets)
        cv2.line(img, (boundary_line[0], boundary_line[1]), (boundary_line[2],
boundary_line[3]), (0, 0, 255), 1)
        shortest_obj_id = None
        min_dist = float('inf')
        closest_coords = None

        for i in result_tracker:
            x1, y1, x2, y2, id = i
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            w, h = x2 - x1, y2 - y1
            cx, cy = x1 + w // 2, y1 + h // 2
```

```
            new_x_, new_y_ = map_coordinates(cx, cy)
            all_x_points.append(ori_width - cx)
            all_y_points.append(ori_height - cy)

            escaped_id = [i[0] for i in escaped_animal]
            if boundary_line[1] - 20 < cy < boundary_line[1] + 20 and id not in escaped_id:
                escaped_animal.append([int(id), cur_cls])
                now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                st.warning(f"{cur_cls} is likely to escaped the marked boundary. {now} +5:30
UTC", icon="⚠")

            dist = ori_height - cy

            if dist < min_dist:
                min_dist = dist
                shortest_obj_id = id
                closest_coords = (cx, cy)

            cvzone.cornerRect(img, (x1, y1, w, h), )
            cvzone.putTextRect(img, f"id:{id}, dist:{dist}", (max(0, x1), max(30, y1 - 10)),
offset=2)
            cv2.circle(img, (cx, cy), 5, (255, 0, 255), cv2.FILLED)
            cv2.line(img, (cx, cy), (cx, ori_height), (0, 255, 0), 1)

        if shortest_obj_id is not None and closest_coords is not None:
            cx, cy = closest_coords
            new_x, new_y = map_coordinates(cx, cy)
            coordinates_placeholder.write(f"Original: ({cx},{cy}) → Altered: ({new_x},
{new_y})")

            tracking_info = f"ID: {shortest_obj_id}, Pixel distance: {min_dist:.1f}"
            if current_distance is not None:
                tracking_info += f", Ultrasonic: {current_distance} cm"
                distance_hisory.append(current_distance)
                pixel_distance_hisory.append(min_dist)
                chart_data = pd.DataFrame({
                    "Ultrasonic distance": distance_hisory,
                    "Pixel distance": pixel_distance_hisory
                })
                graph_placeholder.line_chart(chart_data, x_label="Frames", y_label="Distance")
                # fig, ax = plt.subplots()
                # ax.set_xlim(0, 180)
                # ax.set_ylim(0, 180)
                # ax.scatter(new_x, new_y, color="red", s=100)
                # distance_placeholder.pyplot(fig)
            tracking_placeholder.write(tracking_info)

            escaped_placeholder.write(f"Total escaped: {len(escaped_animal)}")

            if esp:
                data = f"{new_x},{new_y}\n"
                esp.write(data.encode())

            if current_distance is not None:
                cvzone.putTextRect(img,
                                   f"id:{shortest_obj_id}, dist:{min_dist}, distance:
{current_distance}cm, escaped:{len(escaped_animal)}",
                                   (max(0, 0), max(30, 0)), offset=2)
                default_position_sent = False
            else:
                cvzone.putTextRect(img, f"id:{shortest_obj_id}, dist:{min_dist}, escaped:
{len(escaped_animal)}",
                                   (max(0, 0), max(30, 0)), offset=2)
                default_position_sent = False

        elif time.time() - last_detection_time > 3 and not default_position_sent and esp:
            data = "90,90\n"
```

```
            esp.write(data.encode())
            default_position_sent = True
            # status_placeholder.text("Sent default position")

        else:
            cvzone.putTextRect(img, f"No Objects Detected to track, escaped:
{len(escaped_animal)}",
                               (max(0, 0), max(30, 0)), 3, 3, offset=2)
            coordinates_placeholder.write("No coordinates available")
            tracking_placeholder.write("No objects detected")
            distance_placeholder.empty()
            graph_placeholder.empty()
            escaped_placeholder.write(f"Total escaped: {len(escaped_animal)}")

        if all_x_points:
            fig, ax = plt.subplots()
            ax.set_xlim(0, ori_width)
            ax.set_ylim(0, ori_height)
            ax.scatter(all_x_points, all_y_points, color="red", s=100)
            distance_placeholder.pyplot(fig)
            plt.close(fig)

        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        frame_placeholder.image(img_rgb, channels="RGB", use_container_width=True)

    if esp:
        esp.close()
    cap.release()


if __name__ == "__main__":
    main()
```