import cv2
import numpy as np
import winsound
import cv2: This imports the OpenCV library, which is used for computer vision tasks such as processing images and videos.

import numpy as np: This imports the NumPy library, which is used for handling arrays and mathematical operations, especially important in image processing.

import winsound: This imports the winsound module, which is used to generate sound on Windows operating systems. It's specifically used to play a beep sound when an accident is detected.

The play_sound() function
```
def play_sound():
    frequency = 2500  # Set frequency to 2500 Hertz
    duration = 1000  # Set duration to 1000 milliseconds (1 second)
    winsound.Beep(frequency, duration)
```
def play_sound():: Defines a function called play_sound to handle the playing of a beep sound.

frequency = 2500: Sets the frequency of the beep to 2500 Hertz (a high-pitched sound).

duration = 1000: Sets the duration of the beep sound to 1000 milliseconds (1 second).

winsound.Beep(frequency, duration): This line plays the beep sound using the winsound.Beep() function with the specified frequency and duration.

The detect_accidents() function
def detect_accidents(input_video_path, output_video_path):
def detect_accidents(input_video_path, output_video_path):: Defines a function detect_accidents that accepts two parameters: input_video_path (the path to the input video) and output_video_path (the path where the output video will be saved).

Loading YOLO model and configuration
```
net =
cv2.dnn.readNet(r"C:\\Users\\chand\\OneDrive\\Documents\\PGM\\trash\\Accident_Detection\\yolov4.weights",
r"C:\\Users\\chand\\OneDrive\\Documents\\PGM\\trash\\Accident_Detection\\yolov4.cfg")
```
cv2.dnn.readNet(): Loads the YOLO model using the weights file (yolov4.weights) and the configuration file (yolov4.cfg). This function returns a neural network object used for object detection.

The paths provided are hardcoded to where the YOLO model files are located on the system.

Loading class names

with open("coco.names", "r") as f:

classes = f.read().strip().split("\n")

with open("coco.names", "r") as f:: Opens the coco.names file in read mode. This file contains a list of object class names that YOLO can detect, such as "person," "car," etc.

classes = f.read().strip().split("\n"): Reads the entire content of the file, removes any leading/trailing whitespace, and splits the content into a list of class names based on newlines.

Setting up video capture and output

cap = cv2.VideoCapture(input_video_path)

cv2.VideoCapture(input_video_path): Opens the video file located at input_video_path for reading.

frame_rate = int(cap.get(cv2.CAP_PROP_FPS))

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

cap.get(cv2.CAP_PROP_FPS): Gets the frame rate of the input video.

cap.get(cv2.CAP_PROP_FRAME_WIDTH): Gets the width of the frames in the input video.

cap.get(cv2.CAP_PROP_FRAME_HEIGHT): Gets the height of the frames in the input video.

Setting up video writer for saving output

fourcc = cv2.VideoWriter_fourcc(*'XVID')

out = cv2.VideoWriter(output_video_path, fourcc, frame_rate, (width, height))

cv2.VideoWriter_fourcc(*'XVID'): Creates a video codec using the 'XVID' codec for video compression.

cv2.VideoWriter(output_video_path, fourcc, frame_rate, (width, height)): Creates a VideoWriter object to save the processed video to the output_video_path with the specified codec, frame rate, and frame dimensions.

Randomly generating class colors

class_colors = np.random.uniform(0, 255, size=(len(classes), 3))

np.random.uniform(0, 255, size=(len(classes), 3)): Generates a random color for each class of object detected. Each color is represented as a 3-element RGB tuple, where each element is a random integer between 0 and 255.

Setting up variables for crash detection

prev_positions = {}

crash_detected = False

crash_frame = None

prev_positions = {}: Initializes a dictionary to store previous positions of bounding boxes for detecting moving objects (not fully used in this code).

crash_detected = False: Flag indicating whether a crash has been detected.

crash_frame = None: Stores the bounding box coordinates of the detected crash.

Frame processing loop

```
frame_skip = 5  # Process every 5th frame
frame_count = 0
```

frame_skip = 5: Defines how often frames are processed (every 5th frame). This helps to speed up processing.

frame_count = 0: Initializes a counter to track the current frame.

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
```

while cap.isOpened():: This loop runs as long as the video capture object is open and frames are available.

ret, frame = cap.read(): Reads the next frame from the video. ret is a boolean indicating success, and frame is the actual image data.

if not ret:: If no frame is returned (end of video), the loop breaks.

Frame skipping

```
frame_count += 1
if frame_count % frame_skip != 0:
    continue  # Skip processing this frame
```

frame_count += 1: Increments the frame counter.

if frame_count % frame_skip != 0:: Checks if the frame should be processed (every 5th frame).

continue: If the frame is skipped, the loop moves to the next iteration.

Creating blob for YOLO input

```
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True, crop=False)
```

cv2.dnn.blobFromImage(): Converts the input frame into a format suitable for YOLO. This involves resizing the image to 416x416 pixels, normalizing pixel values by dividing by 255.0, and preparing it as a blob.

Performing forward pass through YOLO

net.setInput(blob)
layer_outputs = net.forward(net.getUnconnectedOutLayersNames())
net.setInput(blob): Sets the input blob for the YOLO network.

net.forward(net.getUnconnectedOutLayersNames()): Performs a forward pass through the network and retrieves the outputs of the last layer.

Extracting bounding boxes, confidences, and class IDs
boxes = []
confidences = []
class_ids = []
boxes, confidences, class_ids: Initializes empty lists to store the bounding box coordinates, confidence scores, and class IDs of detected objects.

Processing YOLO layer outputs
```
for output in layer_outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
```
for output in layer_outputs:: Iterates over each output layer from YOLO.

scores = detection[5:]: The first 5 elements of detection represent object location (x, y, width, height) and confidence score. scores contains the confidence values for each class.

class_id = np.argmax(scores): Finds the class with the highest score (most probable object).

confidence = scores[class_id]: The confidence score of the predicted class.

Bounding box calculation
```
if confidence > 0.5:
    center_x = int(detection[0] * width)
    center_y = int(detection[1] * height)
    w = int(detection[2] * width)
    h = int(detection[3] * height)
    x = int(center_x - w / 2)
    y = int(center_y - h / 2)
```
if confidence > 0.5:: Filters out detections with a confidence lower than 50%.

center_x, center_y, w, h: Extracts the center coordinates and dimensions of the bounding box, scaling them to the original frame size.

x, y: Computes the top-left corner coordinates of the bounding box.

Storing bounding box info

```
boxes.append([x, y, w, h])
confidences.append(float(confidence))
class_ids.append(class_id)
```

boxes.append([x, y, w, h]): Adds the bounding box coordinates to the boxes list.

confidences.append(float(confidence)): Adds the confidence score to the confidences list.

class_ids.append(class_id): Adds the class ID to the class_ids list.

Non-Maximum Suppression

```
indices = cv2.dnn.NMSBoxes(boxes, confidences, score_threshold=0.5,
nms_threshold=0.4)
```

cv2.dnn.NMSBoxes(): Applies Non-Maximum Suppression to eliminate overlapping bounding boxes and keep the most confident ones.

Drawing bounding boxes and crash detection

```
for i in indices.flatten():
    box = boxes[i]
    x, y, w, h = box
    label = f"{classes[class_ids[i]]}: {confidences[i]:.2f}"
    color = (0, 255, 0)  # Default color is green
```

for i in indices.flatten():: Iterates through the bounding boxes selected after NMS.

x, y, w, h: Retrieves the coordinates and dimensions of each bounding box.

label: Creates a label string showing the object class and its confidence score.

color: Sets the default color of the bounding box to green.

Handling high-confidence detections

```
if confidences[i] > 0.9:
    color = (0, 0, 255)  # Change color to red
    if confidences[i] > 0.99:
        crash_detected = True
        crash_frame = box  # Store the coordinates of the crash frame
```

if confidences[i] > 0.9:: If the confidence score is above 90%, change the color to red.

if confidences[i] > 0.99:: If the confidence score is greater than 99%, mark this as a potential crash and store the coordinates.

Drawing the bounding box

```
cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)
cv2.putText(frame, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

cv2.rectangle(): Draws a rectangle around the detected object with the specified color and thickness.

cv2.putText(): Adds the label text (class name and confidence) above the bounding box.

Crash detection and sound

```
if crash_detected:
    cv2.rectangle(frame, (crash_frame[0], crash_frame[1]), (crash_frame[0] + crash_frame[2], crash_frame[1] + crash_frame[3]), (0, 0, 255), 2)
    cv2.putText(frame, "CRASH DETECTED!", (crash_frame[0], crash_frame[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 2)
    play_sound()
    crash_detected = False
    crash_frame = None
```

if crash_detected:: If a crash has been detected, draw a red rectangle around the crash area and display the text "CRASH DETECTED!".

play_sound(): Play a sound to alert the user.

crash_detected = False: Reset the crash detection flag after handling the crash.

crash_frame = None: Reset the stored crash frame coordinates.

Saving and displaying the frame

```
out.write(frame)
cv2.imshow('frame', frame)
```

out.write(frame): Saves the processed frame to the output video file.

cv2.imshow('frame', frame): Displays the current processed frame in a window.

Breaking the loop if 'q' is pressed

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

cv2.waitKey(1): Waits for a key press for 1 millisecond.

if cv2.waitKey(1) & 0xFF == ord('q'):: If the 'q' key is pressed, exit the loop.

Releasing resources and closing windows

```
cap.release()
out.release()
cv2.destroyAllWindows()
```

cap.release(): Releases the video capture object.

out.release(): Releases the video writer object.

cv2.destroyAllWindows(): Closes all OpenCV windows.

Main execution block

```
if __name__ == "__main__":
    input_video_path = r"C:\\Users\\chand\\OneDrive\\Documents\\PGM\\trash\\Accident_Detection\\Videos\\303.mp4"  # Path to input video
    output_video_path = "C:\\Users\\chand\\OneDrive\\Documents\\PGM\\trash\\Accident_Detection\\output_video.mp4"  # Path to save output video
    detect_accidents(input_video_path, output_video_path)
```

if __name__ == "__main__":: This block ensures the program runs only if it's executed directly (not imported as a module).

input_video_path and output_video_path: Set the file paths for the input video and output video.

detect_accidents(input_video_path, output_video_path): Calls the detect_accidents() function to start processing the video.