COMP3065
Computer Vision Coursework

Submitted May 5th 2024

Shujun JIANG
20320552

School of Computer Science University of Nottingham Ningbo China

Contents

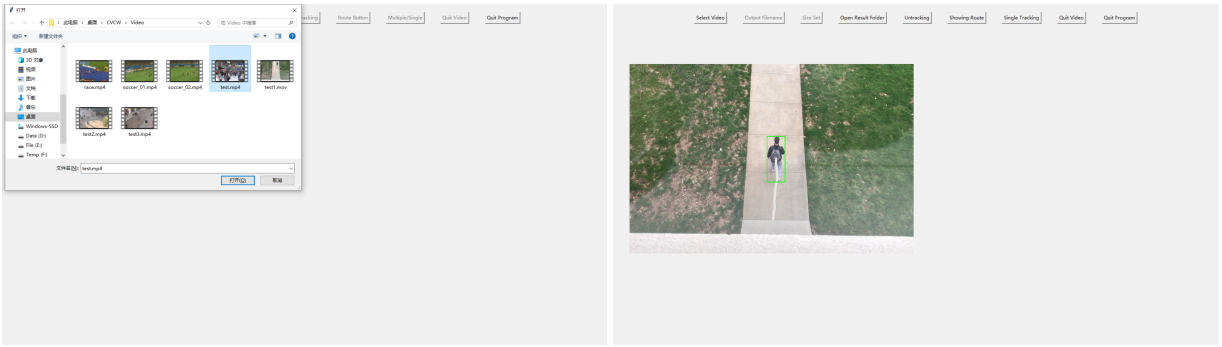
Description of key features of the implementation

Tkinter User Interface

In our project, we have used the Tkinter library to create a simple and intuitive GUI interface that provides various functions for video processing and tracking. In the following list are the detailed features provided by the UI.

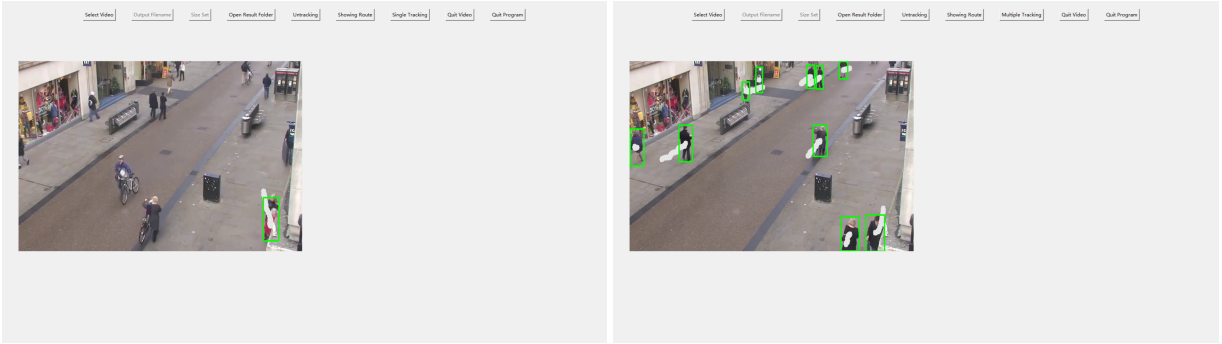
1. **Select Video File**: Click the file selection button on the interface, the system pops up a file selection dialog box, so that users can easily select the video file to be processed.
2. **Modify Output Filename**: The interface provides an input box where the user can specify the name of the generated output video to facilitate subsequent recognition and management.
3. **Set Video Dimensions**: The user can set the size of the output video through the buttons on the interface to meet the requirements of different application scenarios.
4. **Open Output Directory**: The user clicks the button to view the generated output video file and other related files.
5. **Enable Tracking**: A tracking button is provided on the interface to control whether the target tracking function is enabled or not.
6. **Display Path**: The user can click this toggle button to select whether to display the motion path of the target in the video.
7. **Switch Between Single/Multiple Tracking**: Click the single/multiple button to switch the single target tracking mode and the multiple target tracking mode.
8. **Close Video**: The user can click the button to stop the video playing and release the related resources.
9. **Close Program**: The user clicks this button to exit the program, close the interface window and release the occupied resources.

By clicking on these buttons, users can intuitively select video files, set parameters, and control the tracking process on the interface.



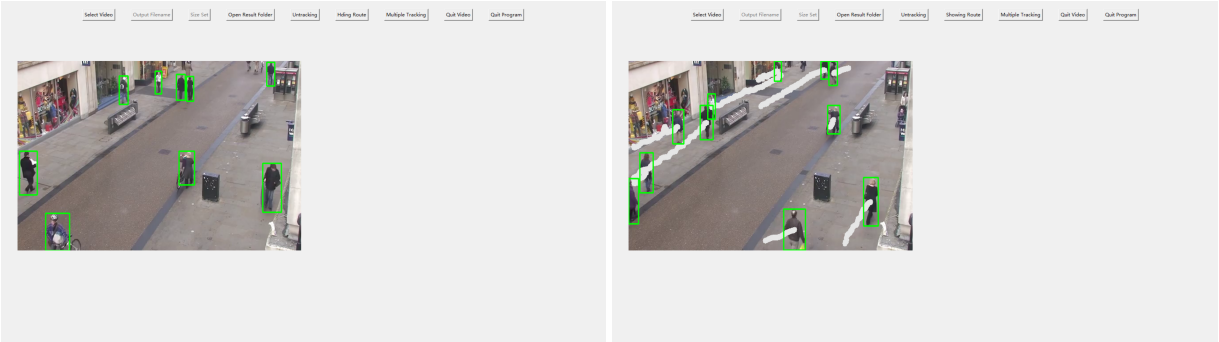
Single/Multi-Person Tracking Support

The program supports the user to choose single or multiple tracking mode. In multi-person tracking mode, the program can track multiple people at the same time and display their bounding boxes and trajectories (if trajectories are selected). This demonstrates the ability of the program to cope with complex environments and problems.



Show/Hide Motion Track Function

The program allows the user to choose whether to display the trace path or not. When this feature is selected, the tracked target will have a line behind it to show its movement path. This demonstrates the program's ability to keep working and track accurately.

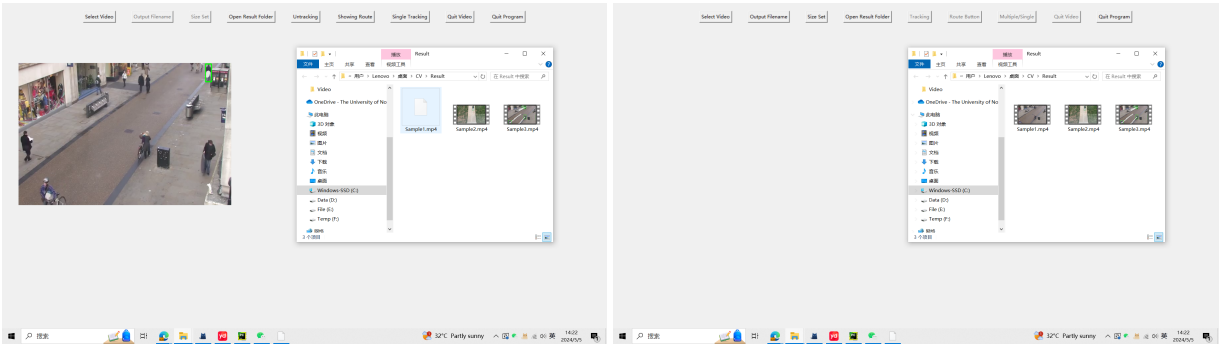


The following code shows how to process a video frame. The program draws the tracking path and detection box on the image by iterating over the detected target box and its tracking ID and category ID. Firstly, the program determines whether the target is a human or not, and then obtains the location information and tracking history of its target box. Next, the program adds the coordinates of the center point of the target box in the current frame to the tracking history. If object tracking and allowing paths are enabled, OpenCV's cv2.polylines() function will be used to draw the path of the object. Any box that is detected at the same time uses the cv2.rectangle() function to show the position and boundary of the object as a green rectangle. Through these operations, users can intuitively observe the motion trajectory and position change of the target, so as to better understand the behavior and motion pattern of the target.

```
# Draw the tracing path and detection box
for box, track_id, class_id in zip(boxes, track_ids, class_ids):
    if class_id == 0:  # The ID of the "person" class is usually 0, which you can determine based
        # on the class index of the YOLO model
        x, y, w, h = box
        track = track_history[track_id]
        track.append((float(x), float(y)))  # x, y center point

        # Draw a trace
        points = np.hstack(track).astype(np.int32).reshape((-1, 1, 2))

        if variables.Tracking:
            if variables.path:
                cv2.polylines(current_frame, [points], isClosed=False, color=(230, 230, 230),
                              thickness=10)

            # Draw the detection box
            cv2.rectangle(current_frame, (int(x - w / 2), int(y - h / 2)),
                          (int(x + w / 2), int(y + h / 2)),
                          (0, 255, 0), 2)
```

Real-time Video Processing

The OpenCV library was used to read the video frames, and the YOLOv8 model was run on each frame for object detection and tracking. After detecting the target, the program updates the video display on the interface in real time, and draws the bounding box and motion trajectory of the target to realize real-time video processing and display. While the video is being processed and displayed, the video is being written at the same time, and the processed video can be seen under the Results folder when the user closes the program.



YOLOv8 Model Principle

Algorithm Overview

- YOLOv8 is a one-stage model for object detection, based on the concept that it only needs one pass to identify objects in an image, which is different from traditional two-stage methods.
- It divides the image into a grid of equal-sized cells and predicts a set of bounding boxes for each cell, along with a probability score for each box.
- Each grid cell is responsible for predicting the presence of objects in it, utilizing one-hot encoding for class prediction.
- By outputting all detection results at once, it eliminates the need for sliding Windows or region suggestion techniques.

Algorithm Principles

1. **Grid Segmentation and Bounding Box Generation**: The image is divided into a grid of cells and each cell predicts B bounding boxes, each characterized by a center coordinate, height, width, and confidence score.
2. **IOU Calculation**: The Intersection over Union (IOU) is used to measure the similarity between the predicted bounding box and the true bounding box.
3. **Confidence Calculation**: The confidence indicates whether an object is present in the predicted bounding box and is calculated as the product of the predicted confidence and IOU.
4. **Non-Maximum Suppression (NMS)**: It is used to remove overlapping bounding boxes and keep those with the highest confidence scores to prevent redundant detections.

YOLOv8 Model Structure

- YOLOv8 provides various models, including object detection networks with P5 640 and P6 1280 resolutions, and instance segmentation models based on YOLACT.
- Based on the YOLOv7 ELAN design, the C2f structure is used to adjust the number of channels for different scale models, and the backbone network and neck are fine-tuned.
- The first segment uses a decoupled first segment structure to transition from anchor base to anchor-free base.
- The loss is calculated using the TaskAlignedAssigner positive sample assignment strategy and the assignment focus loss, as well as the CIoU loss.

Mosaic Augmentation

- YOLOv8 uses Mosaic augmentation during training by concatenating four images to force the model to learn new object locations, partial occlusions, and changes in surrounding pixels.

Ensemble of YOLOv8 Models

In the following code, we load the pre-trained YOLOv8 model by leveraging the ultralytics library, including two different versions of the model :yolov8n and yolov8s. The two versions have slight differences in model structure and parameters to meet the requirements in different scenarios. In this way of model integration, we can combine the recognition results of the two models to improve the overall performance and accuracy of the model.

The benefit of model integration is that it can make full use of the advantages of multiple models and compensate for the disadvantages of their respective models. For example, yolov8n may have higher accuracy in some scenarios, while yolov8s may have faster inference speed. Through the combination of the two, the processing speed can be improved while ensuring accuracy, or the accuracy can be improved while ensuring speed. In this way, the comprehensive use of the advantages of different models can make the overall target detection and tracking effect more stable and reliable.

```python
# Loading the model
model = YOLO('./model_yolo/yolov8n.pt', verbose=False)
models = YOLO('./model_yolo/yolov8s.pt', verbose=False)

# Run YOLOv8 tracking on frames to continuously track objects between frames
results1 = model.track(current_frame, persist=True)
results2 = models.track(current_frame, persist=True)
# Ensemble model results
results = results1 + results2

if results and results[0].boxes:  # Check whether the result is empty

    if variables.multiple is False and len(results[0].boxes) > 1:
        results[0].boxes = results[0].boxes[:1]

    # Gets the box and trace ID
    boxes = results[0].boxes.xywh.cpu()
    try:
        track_ids = results[0].boxes.id.int().cpu().tolist()
        class_ids = results[0].boxes.cls.int().cpu().tolist()  # Get category information for the box
```

Two schemes that were not Adopted

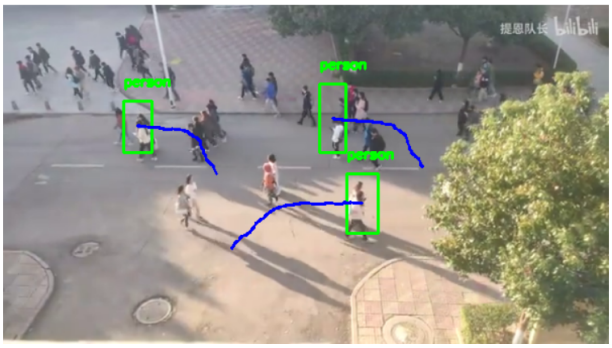
My project has a folder called Deprecated_Version, in this folder there are two folders Haar and SSD. These two folders are the algorithms and models I used before, if you need to test this part of my work, you can move the contents of these two folders to the root directory to test.

The following two images show the main code in each of these files.

```python
def haar_detection():
    face_path = 'haarcascades\\haarcascade_frontalface_default.xml'
    eye_path = 'haarcascades\\haarcascade_eye.xml'
    smile_path = 'haarcascades\\haarcascade_smile.xml'
    diction = {'face': face_path, 'eye': eye_path, 'smile': smile_path}
    for i in diction:
        diction[i] = cv2.CascadeClassifier(diction[i])
    return diction


def video_detection(video_file, detection):
    video_capture = cv2.VideoCapture(video_file)
    if not video_capture.isOpened():
        print("Error: Could not open video file ", video_file)
        exit()

    while True:
        ret, frame = video_capture.read()
        if not ret:
            break
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        face_ret = detection['face'].detectMultiScale(gray_frame, scaleFactor=1.02, minNeighbors=5, minSize=(15, 15),
                                                      maxSize=(50, 50), flags=cv2.CASCADE_DO_CANNY_PRUNING)
        for (x, y, w, h) in face_ret:
            face_roi = gray_frame[y:y + h, x:x + w]
            eye_ret = detection['eye'].detectMultiScale(face_roi, scaleFactor=1.1, minNeighbors=3, minSize=(15, 15),
                                                        flags=cv2.CASCADE_SCALE_IMAGE)
            smile_ret = detection['smile'].detectMultiScale(face_roi, scaleFactor=1.1, minNeighbors=3, minSize=(15, 15),
                                                            flags=cv2.CASCADE_SCALE_IMAGE)
            for (xx, yy, ww, hh) in eye_ret:
                pt1 = (x + xx, y + yy)
                pt2 = (pt1[0] + ww, pt1[1] + hh)
                cv2.rectangle(frame, pt1, pt2, (255, 0, 0), 2)
            for (xx, yy, ww, hh) in smile_ret:
                pt1 = (x + xx, y + yy)
                pt2 = (pt1[0] + ww, pt1[1] + hh)
                cv2.rectangle(frame, pt1, pt2, (0, 255, 0), 2)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cv2.imshow('Video', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    video_capture.release()
    cv2.destroyAllWindows()
```

```python
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
print("[INFO] starting video stream...")
vs = cv2.VideoCapture(args["video"])
writer = None
trackers = []
labels = []
fps = FPS().start()
tracks = [[] for _ in range(len(trackers))]
while True:
    (grabbed, frame) = vs.read()
    if frame is None:
        break
    (h, w) = frame.shape[:2]
    width = 600
    r = width / float(w)
    dim = (width, int(h * r))
    frame = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    if args["output"] is not None and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 30,
                                 (frame.shape[1], frame.shape[0]), True)
    if len(trackers) == 0:
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(frame, 0.007843, (w, h), 127.5)
        net.setInput(blob)
        detections = net.forward()
        for i in np.arange(0, detections.shape[2]):
            confidence = detections[0, 0, i, 2]
            if confidence > args["confidence"]:
                # extract the index of the class label from the
                # detections list
                idx = int(detections[0, 0, i, 1])
                label = CLASSES[idx]
                if CLASSES[idx] != "person":
                    continue
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")
                t = dlib.correlation_tracker()
                rect = dlib.rectangle(int(startX), int(startY), int(endX), int(endY))
                t.start_track(rgb, rect)
                labels.append(label)
```

As you can see from the two images below, they don't perform very well. The accuracy of Haar detection is not enough, and it will be wrongly recognized. The SSD model also performs poorly when dealing with situations such as overlapping characters and fast camera movement. I tried to optimize them but they didn't work so I switched.

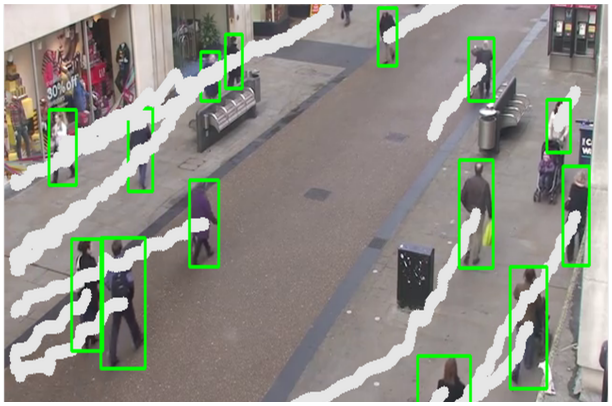
Parameter Passing Method

I have used a few main methods to pass parameters: The argparse library is used to process the command-line arguments. In main.py, you define an argument parser ap and then use the ap.add_argument() method to add several arguments, such as the path to the input video file, the path to the output video file, and the width and height of the video. Finally, the command line arguments are parsed with ap.parse_args() and the result is stored in args. In variable.py, you define global variables such as Processing, Video_Playing, END, cap, out, Tracking, multiple, and path. These variables can be accessed and modified throughout the program, as they store state information and configuration options to control UI synchronization, acting like "locks".

Explanation of the results obtained

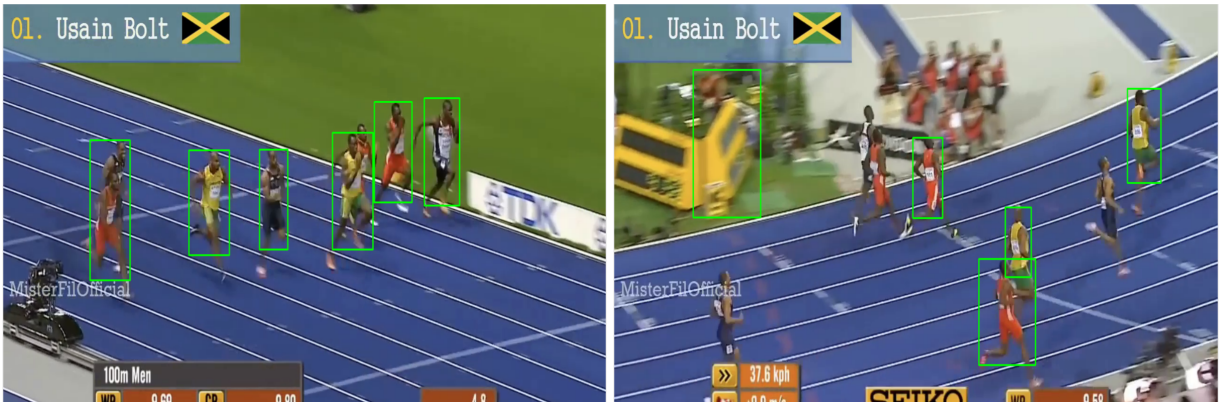
After running main.py, the UI buttons allow you to select videos and track people. In general, I completed the tasks specified in the CourseWork, and completed several additional tasks such as UI, model integration optimization, single-person multi-person tracking and so on. Through testing or watching the videos in Sample, we can know that the recognition and tracking functions of the program basically meet the expectations. Even in the station with a large number of people, such as test4.mp4, the program can almost cover and track a large number of people.

However, we found that it is difficult to capture distant people in this scene because the video itself is not high in pixels and distant people are more difficult to identify and track. In addition, the scene is complex and the characters are occluded, which may not be recognized by the program. Some special behaviors may not be recognized, such as the person pushing a cart in test3.mp4, when the body is superimposed on a bicycle, or when a baby is in a pram.

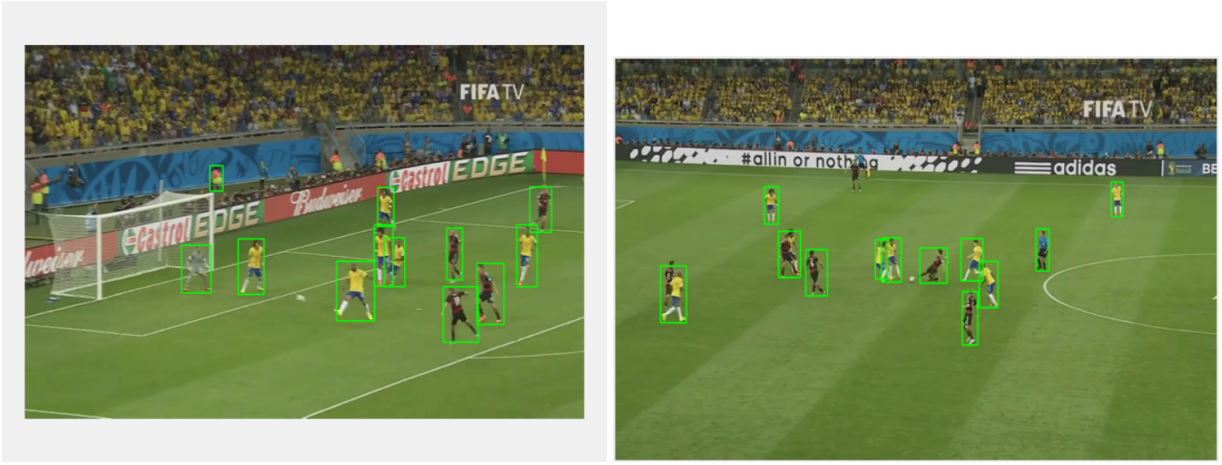


But in general, the excellent performance of the program can be proved when testing videos 1-4 in the Video folder.

Test videos 5-7 Since the camera is no longer stationary but moving, the function of drawing trajectories is not necessary. But from this, we can find that the program still has some shortcomings. For example, in test5.mp4, similar runners are identified together, and the baffler is mistakenly identified as a human body when the camera moves at a high speed and the shooting is not clear.



However, in soccer videos 6 and 7, perhaps due to the high quality of the video, the program easily recognized the people in the video even though the camera was moving, even though they were doing various soccer actions.



In summary, through comprehensive testing and observation, we find that the program can successfully realize the task of person recognition and tracking in most cases, and show good performance in the sample videos. The program not only completed the basic tasks specified in the course assignment, but also implemented additional tasks such as user interface design, model integration optimization, single-person multi-person tracking and so on. However, there are some limitations in certain situations, such as difficulty in capturing objects at long distances or with low video pixels, and possible occlusion of other objects when drawing the path. Moreover, under moving cameras, the performance of the program may suffer, leading to erroneous recognition or tracking results. However,

in general, the program performs well in most cases, meets most needs, and is more reliable when the video quality is high or stable.

Discussion of the strengths and weaknesses of the chosen approach and methods

Strengths of the Chosen Approach and Methods

1. **Real-time Video Processing Capability:** The program uses real-time video processing, combines OpenCV library to read video frames, and combines YOLOv8 model for target detection and tracking. This allows the program to display real-time detection and tracking results while processing the video, enhancing user interaction and responsiveness.

2. **Flexible Parameter Setting and Control:** The user interface provides rich parameter setting and control functions, such as selecting video files, setting output file names, adjusting video dimensions, enabling target tracking, and displaying trajectories. Users can adjust the parameters flexibly according to different requirements and scenarios to achieve the optimal detection and tracking effect, which improves the flexibility and applicability of the scheme.

3. **Multi-person Target Tracking Support:** The project supports single and multi-person target tracking modes. By toggling the tracking mode button, the user can flexibly choose to track a single target or multiple targets. In the multi-person tracking mode, the program can track multiple targets at the same time and display their bounding boxes and motion trajectories, which enhances the practicability and versatility of the program.

4. **Ensemble of Multiple Models Optimization:** By integrating multiple types of YOLOv8 models, the program integrates the advantages of each model in accuracy and speed, so as to improve the overall performance and stability of the model. This optimization method of multi-model integration can effectively improve the accuracy and robustness of object detection, so that the program can play a good effect in different scenes.

Weaknesses of the Chosen Approach and Methods

1. **Poor Robustness to Occlusions Between Targets:** When the tracked target is occluded or similar to the surrounding environment, the object detection and tracking performance of the program will be affected, resulting in missing or misjudging the target.

2. **Limited Processing Capability for Low-Quality Videos:** Programs may not

perform well when dealing with low-quality videos such as low-resolution or blurred videos. Especially in distant scenes in videos, targets are often difficult to accurately identify and track. Further optimization of the algorithm may be necessary in the future to improve the ability of the program to handle low-quality videos.

3. **Limited Adaptation to Moving Cameras:** In the case of fast camera motion, the object detection and tracking performance of the program may degrade, leading to wrong object recognition or tracking results. This is because video is prone to distortion due to camera motion, similar to the case in the previous article. This will affect the accuracy of the object detection and tracking algorithm. Further optimization of the algorithm is needed in the future to improve the ability of the program to process videos captured by moving cameras.