Person Tracking from Videos Report

Project Objectives and Main Function Description

Project selection instructions: This project is "Option B - Person tracking from videos".

Project Objective: This project aims to build a system capable of real-time detection and tracking of multiple characters in video streams. Through this system, we strive to accurately identify each character and assign a unique number to each one. Meanwhile, the system can also dynamically record the movement trajectory of each target and visualise this information.

The system has core functions such as automatic detection of human targets, cross-frame continuous tracking and trajectory drawing, and the ability to be visualised on the front-end interface. It can accurately identify each character in the video, assign a unique number to each target, and continuously track the changes in their positions. Meanwhile, the system will draw the movement trajectory of the characters in real time in the video, allowing users to see the movement path of the characters intuitively.

Detailed Explanation of methods and steps

Overall Architecture Description

This system adopts PySide6 as the front-end graphical interface. The overall process of the back-end algorithm can be divided into four main links: video reading, character detection, target tracking, and trajectory drawing. The system first reads the video file to extract frames from the input video. Then, it uses the YOLOv11 model to detect the human target in each frame and

transmits the detection results to the tracking module. Combined with the inter-frame information, the uniqueness of the target identity is achieved. The system records and updates the movement trajectory of the target in each frame, and outputs the detection box, ID and trajectory after dynamic visualisation.

The system's core functions are realised through the following key modules: object detection, tracking, result information management, and visualisation modules.

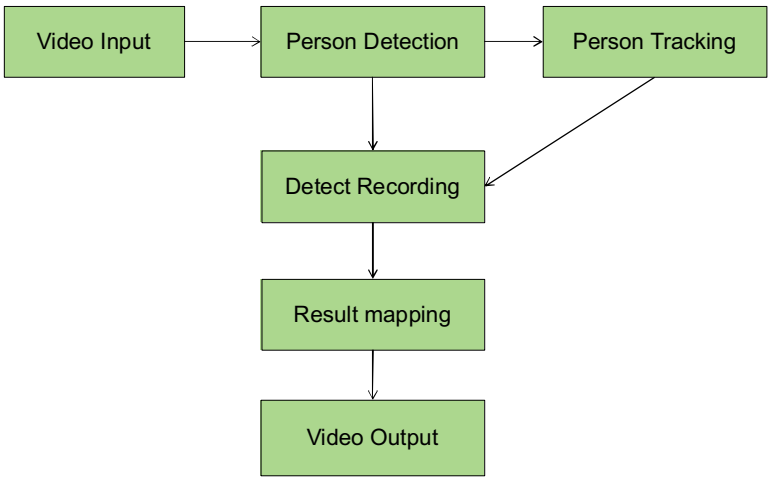

Figure 1. System Flowchart

Module Functions and Implementation Details

1. Character Detection Methods

This system adopts the YOLOv11 object detection model provided by the Ultralytics framework. YOLOv11 is a single-stage detector, featuring high precision and high speed. It is particularly suitable for scenarios with high real-time requirements, such as video surveillance and multi-target recognition and tracking tasks.

This system loads the officially trained weight file YOLOV11S.PT, so there is no need for retraining, and it can be directly deployed and used. During detection, YOLOv11 will process each video image frame, automatically identify the

human targets in the image, and return information such as each target's bounding box coordinates, CLASSES, and confidence. The system filters out non-human targets and low-credibility detection results by setting confidence thresholds and target categories to ensure the detection quality.

2. Tracking Algorithm Logic

After completing the character detection, the system must perform "identity association" for the same character in different frames, assigning and maintaining a unique ID for each target. This function is achieved through the built-in multi-object tracking module of YOLOv11, using the model.track() interface.

This interface automatically performs inter-frame association operations based on object detection. Its core logic includes:

Compare the relative position changes between the bounding boxes detected in the current and previous frames. Use the IOU (intersection-over-union ratio) as the key indicator to determine whether they are the same target. Match in combination with the visual features of the target (such as appearance) to avoid misidentification when the targets overlap or are close. Newly emerged targets are automatically assigned a new IDS, and the ID persistence within a certain number of frames is maintained after the target temporarily disappears to avoid ID switching caused by brief occlusion.

3. Trajectory Drawing Method

To enhance the visualisation effect and display the movement process of the target in the video, the system plots the movement trajectory of each target. The specific implementation methods are as follows:

For the targets detected in each frame, extract the coordinates of the centre points of their bounding boxes (center_x, center_y). Based on the tracking ID, set up a track_histories dictionary to record the historical location information (coordinate sequence) corresponding to this ID. In the image, the cv2. The line

() function of Opencv is used to connect each target's continuous trajectory points to achieve the trajectory's dynamic visualisation.

4. Code Details

Configuration file details：

- Video source configuration

VIDEO_SOURCE: Enter the video source, which can be the video file path (such as "1.mp4"), camera number (such as 0), or network video stream address (such as RTSP).

- Detection and tracking configuration

MODEL_PATH: The path of the YOLO11 model weight file (for example, "yolo11s.pt").

DEVICE: Specifies the device on which the model runs. "cpu" indicates CPU use, and "gpu" indicates GPU acceleration.

CONF_THRESH: Detect the confidence threshold. The higher the value, the stricter it is (filter out low-confidence targets).

IOU_THRESH: The IoU threshold of non-maximum suppression (NMS) removes overlapping targets.

CLASSES: Set the categories to be detected. 0 indicates that only "people" are detected (corresponding to the person category in the COCO dataset).

TRACK_HISTORY_LEN: The number of historical trajectory points saved for each target.

- Display configuration

SHOW_FPS: Whether to display real-time FPS (frame rate) in the picture.

"SHOW_TRACK: Whether to display the movement trajectory of the target."

LINE_THICKNESS: The thickness of the trajectory line.

FONT_SIZE: Display the font scaling size of the text (such as FPS number, target ID).

FONT_THICKNESS: The thickness of the font.

TRACK_FADE_FACTOR: The fading speed of the trajectory line. The smaller the value, the faster the fading; the larger the value, the longer the retention time of the trajectory.

- Colour configuration

TEXT_COLOR： The text colour adopts the BGR format. For example, white is (255, 255, 255).

- Output configuration

SAVE_OUTPUT: Whether to save the processed video file.

SOURCE_NAME: The file name is automatically extracted from the input video source and used to generate the output name.

OUTPUT_PATH: The output video's save path and file name (for example, "1_output.mp4").

OUTPUT_FPS: The frame rate of the saved video (such as 30 frames per second).

VIDEO_CODEC: The encoding format used to save the video. For example, "mp4v" applies to the .mp4 format.

The specific parameter Settings are shown in the following figure:

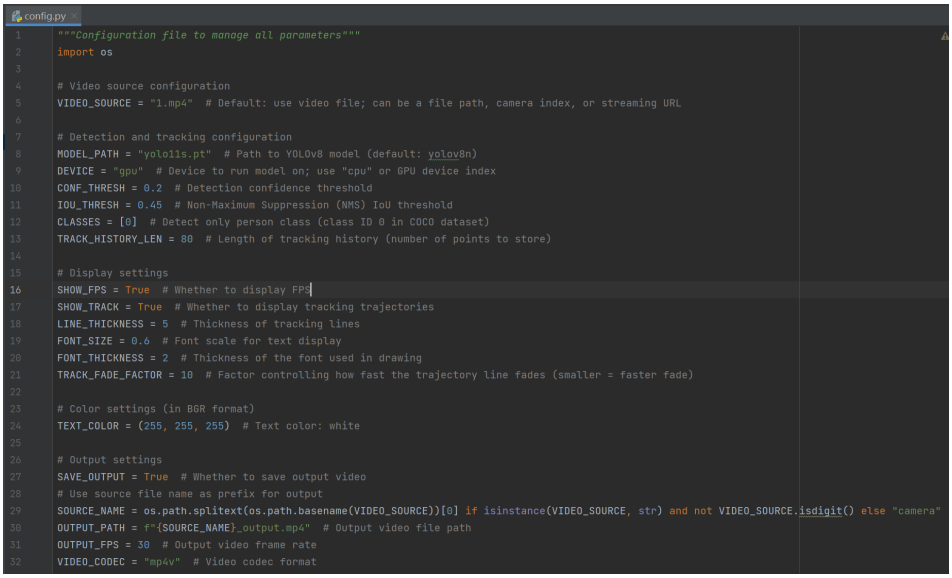

Figure 2. Detailed diagram of parameter Settings

Details of the main program code

Model loading: Load the YOLOv11 model for subsequent target detection and tracking.

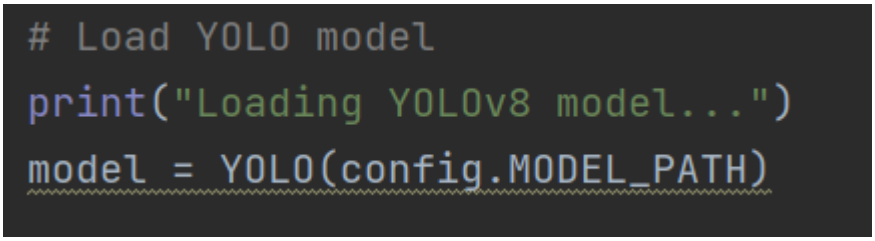

Figure 3. Model loading code

Open the specified video source (local file, camera, or network stream) and obtain the video's width, height, and frame rate for video input source reading

and basic information extraction.

```
# Get video source
print(f"Connecting to video source: {config.VIDEO_SOURCE}")
cap, width, height, fps = utils.get_video_source(config.VIDEO_SOURCE)
print(f"Video resolution: {width}x{height}, FPS: {fps}")
```

Figure 4. Video input processing code

The central processing loop (frame-by-frame processing) reads video frames, calculates real-time FPS, and uses Yolov11's track () method to perform "detection and tracking" on the current frame.

```
print("Starting video processing...")
while True:
    # Read one frame
    ret, frame = cap.read()
    if not ret:
        print("Video finished or source error")
        break

    # Calculate FPS
    frame_count += 1
    if frame_count % 10 == 0:  # Update FPS every 10 frames
        fps_value = utils.calculate_fps(start_time, frame_count)
        print(f"Processed {frame_count} frames, FPS: {fps_value:.2f}")

    # Perform object tracking
    results = model.track(
        source=frame,
        conf=config.CONF_THRESH,
        iou=config.IOU_THRESH,
        classes=config.CLASSES,
        persist=True,  # Enable cross-frame persistent tracking
        verbose=False
    )
```

Figure 5. Video frame detection and tracking code

Trajectory processing and drawing, tracking ID extraction and trajectory update, extracting each target's bounding box coordinates and tracking ID. Calculate the target centre point (center_x, center_y) and record it in the

historical trajectory through utils.update_track_history(). Draw the movement path on the image based on the historical trajectory of the current target.

```python
# Process tracking results
if results and len(results) > 0:
    result = results[0]  # Get the first result
    annotated_frame = result.plot()  # Get annotated frame

    # If tracking ID exists, update trajectory history
    if hasattr(result, 'boxes') and hasattr(result.boxes, 'id') and result.boxes.id is not None:
        boxes = result.boxes.cpu().numpy()
        for i, box in enumerate(boxes):
            # Get bounding box and tracking ID
            x1, y1, x2, y2 = box.xyxy[0].astype(int)
            track_id = int(box.id[0])

            # Calculate bounding box center
            center_x = (x1 + x2) // 2
            center_y = (y1 + y2) // 2

            # Update trajectory history
            track_histories = utils.update_track_history(
                track_histories, track_id, center_x, center_y
            )

        # Draw trajectory on the frame
        annotated_frame = utils.draw_tracks(annotated_frame, track_histories)
```

Figure 6 Video frame detection and tracking code3.

Presentation and interpretation of experimental results

System interface display

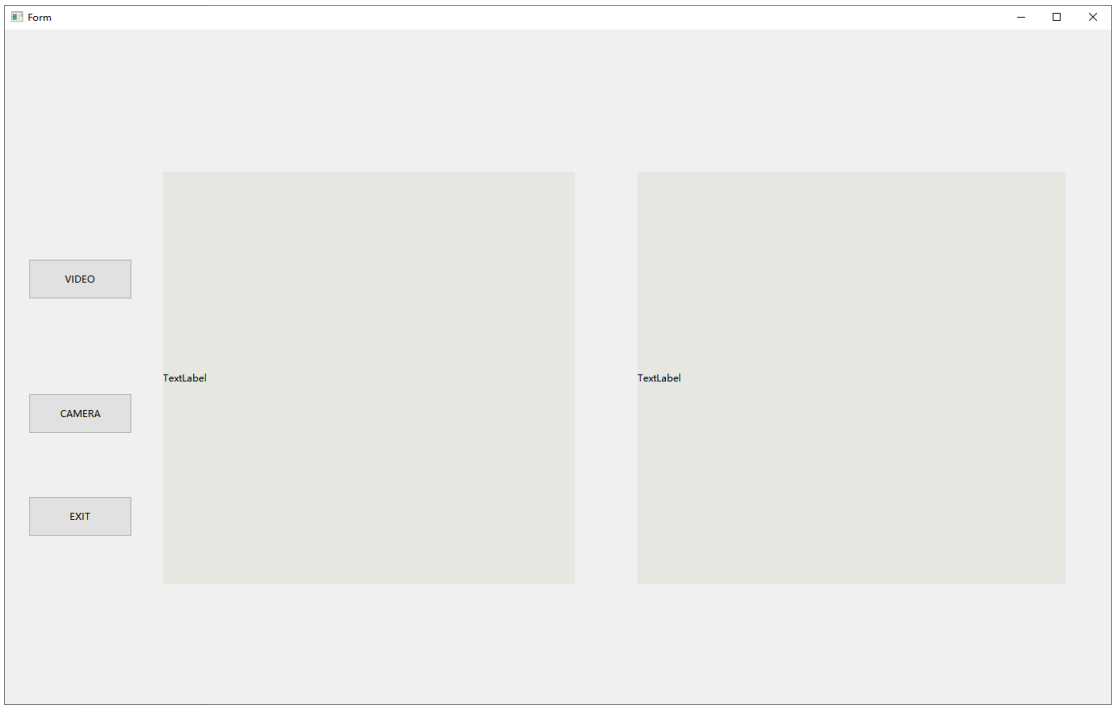

Figure 7. Graphical user interface display

The interface is divided into a file input and video display areas. In the file input area, users can input video files or directly obtain video streams from the camera. The video display area is divided into left and right sections. The left area shows the original video, and the right shows the detection and tracking results.

The following are the results of the three testing videos:

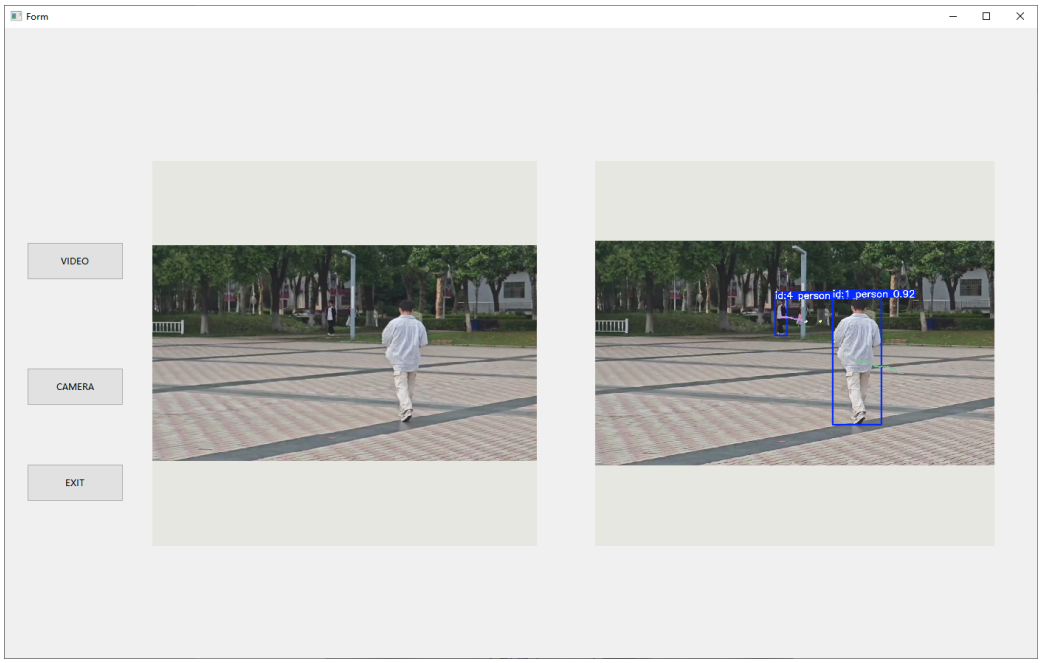

Figure 8 Test Video 1 Effect demonstration

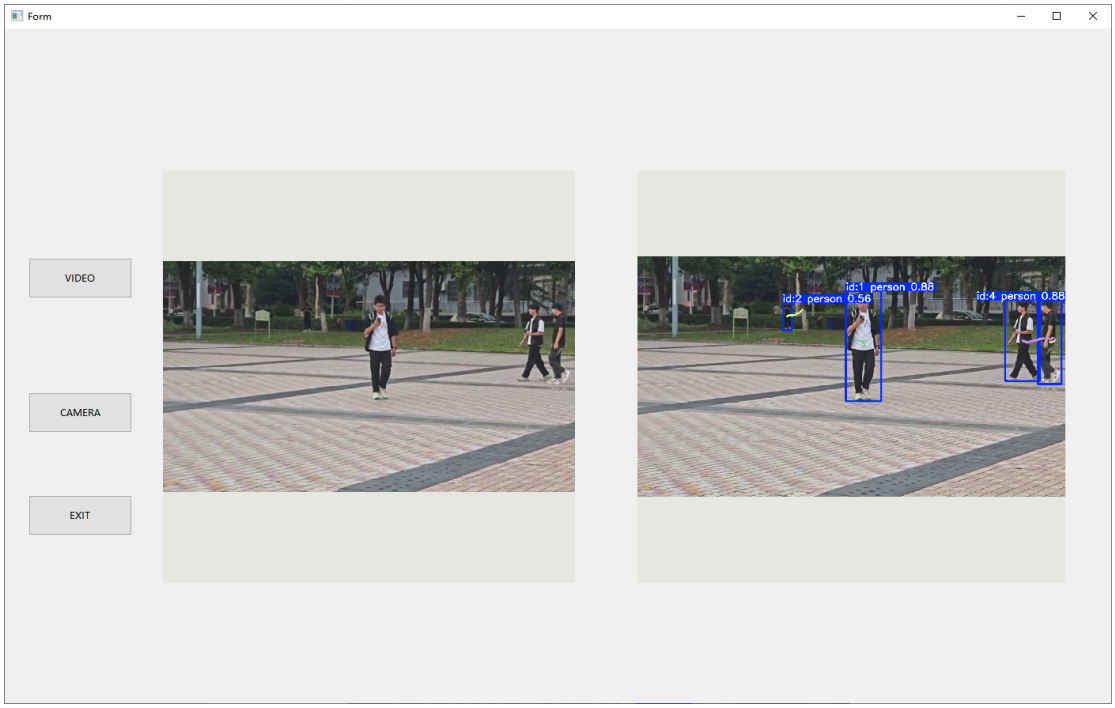

Figure 9 shows the effect of Test Video Two

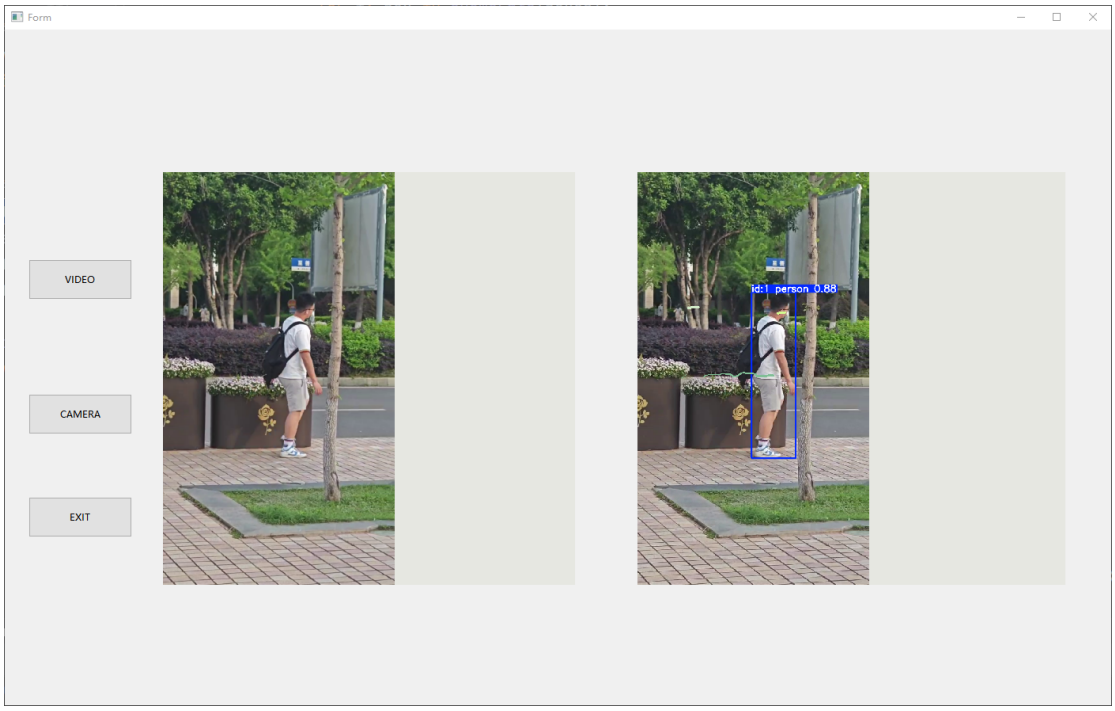

Figure 10. shows the effect demonstration of the test video three

The test results of the three videos show that the system can successfully detect and track characters in different scenarios and has a good real-time response capability. The detection borders of each target are accurate, the ID allocation is unique and remains unchanged during the cross-frame process, and the trajectory drawing is precise and continuous. Especially in scenarios where multiple people appear simultaneously and personnel move frequently, the system can still correctly label the ID of each character and draw independent trajectories.

Critical Evaluation of the Advantages and Disadvantages of the Method

This system performs stably in various scenarios and has good detection and tracking capabilities. In an environment with clear targets, slow movement and uniform lighting, the system can accurately identify multiple characters, assign stable IDS and draw continuous trajectories. For instance, in open squares or areas with sparse foot traffic, the detection border closely adheres to the person,

the trajectory line is smooth, and the ID hardly changes, making it suitable for outdoor monitoring, foot traffic analysis and other application scenarios. However, in some complex environments, the system performance slightly declines. Specifically including:

- When multiple people block, the person in the foreground may block the target behind, causing the blocked person to be temporarily unable to detect it. Once it reappears, the system may misjudge its ID as a new target, resulting in ID confusion or trajectory breakage.
- When drastic changes in lighting occur, such as strong backlighting or shadow coverage, they may affect the detection confidence, causing the target border to shift or be missed.
- When a character hurries or exits the drawing and re-enters, it is easy to be assigned a new ID, losing the continuity of the original trajectory.