

Tracking Data Logger for Real-Time Chain-Snatching Detection

1. Project Overview

This project implements a **Tracking Data Logger**, which is the foundational module of a **Real-Time Chain-Snatching Detection System**.

The goal of this module is to convert raw video input into **structured tracking data** by detecting people, tracking them across frames, and logging their movement information in a CSV file.

This structured output is later used for:

- Motion analysis
- Interaction detection
- Chain-snatching verification

2. Why Tracking Data Logger is Needed

Traditional video analysis processes each frame independently and lacks memory. However, activities like chain snatching are **events over time**, not single-frame objects.

The Tracking Data Logger provides:

- Temporal continuity (memory)
- Persistent person identities
- Structured movement data

Without this module, reliable behavior analysis is not possible.

3. Libraries Used and Their Purpose

3.1 ultralytics (YOLOv8)

Purpose: Person detection

- Used to detect people in each video frame
- Pre-trained on the COCO dataset
- Provides bounding boxes around detected persons

Why used:

- Fast
- Accurate
- No training required

3.2 supervision

Purpose: Tracking and detection handling

- Converts YOLO output into a standardized format
- Integrates **ByteTrack** for multi-object tracking

3.3 ByteTrack (via supervision)

Purpose: Person identification (tracking)

- Assigns a **persistent ID** to each person
- Maintains the same ID across consecutive frames
- Uses motion consistency to match detections

Why needed:

- Detection alone cannot remember people
- Tracking enables identity preservation

3.4 opencv-python (cv2)

Purpose: Video processing

- Reads video frame-by-frame
- Handles video input in Kaggle
- Provides frame data to the detection model

3.5 pandas

Purpose: Data storage

- Collects tracking data

- Saves output in CSV format
- Enables easy verification and reuse

3.6 os

Purpose: File handling in Kaggle

- Used to locate uploaded video files inside /kaggle/input

4. Execution Environment: Kaggle

The project was implemented and tested on **Kaggle**, which is a cloud-based notebook environment.

Important characteristics of Kaggle:

- Runs on remote servers
- No GUI (headless environment)
- No access to user's local file system
- Input must be uploaded through Kaggle interface

5. How Input Video is Extracted in Kaggle

Step 1: User uploads video

The user uploads a video file (e.g., cctv.mp4) using:

Kaggle Notebook → Add Data → Upload

Kaggle automatically mounts the uploaded file inside:

/kaggle/input/<dataset-name>/

Step 2: Code locates the video

The code scans the /kaggle/input directory to find video files:

```
INPUT_DIR = "/kaggle/input"
```

The first video file with extensions .mp4, .avi, .mov, or .mkv is selected automatically.

This allows the system to work with **any uploaded video** without hardcoding paths.

6. How the Code Works (Step-by-Step)

Step 1: Video is opened

```
cap = cv2.VideoCapture(video_path)
```

- Video is read frame by frame
- Each frame is processed independently at first

Step 2: Person detection

```
results = model(frame)
```

- YOLOv8 detects all persons in the frame
- Outputs bounding boxes for each detected person

At this stage:

- Persons are detected
- No identity is assigned yet

Step 3: Filtering only persons

```
detections = detections[detections.class_id == 0]
```

- COCO class 0 corresponds to person
- Other objects are ignored
- This improves tracking stability

Step 4: Multi-object tracking (core fix)

```
detections = tracker.update_with_detections(detections)
```

This is the **key step**.

What happens here:

- Detections from the current frame are matched with previous frames

- ByteTrack assigns a stable track_id
- The same person keeps the same ID across frames

This step fixes the earlier issue where:

- IDs were missing
- IDs changed every frame

Step 5: Tracking data logging

For every detected person, the following data is recorded:

frame_number, track_id, x1, y1, x2, y2

This represents:

- When the person appeared (frame)
- Who the person is (track_id)
- Where the person was (bounding box)

7. How Output is Saved in Kaggle

Output file location

Kaggle allows saving outputs only inside:

/kaggle/working/

The CSV is saved as:

/kaggle/working/tracking_data.csv

CSV Structure

frame, track_id, x1,y1,x2,y2
1,0,120,85,245,430
1,1,310,90,415,440
2,0,125,88,248,435

This file:

- Is automatically generated

- Can be downloaded from Kaggle
- Is reusable for later analysis

8. How We Verified the Output

The CSV file was verified by:

- Checking column names
- Ensuring repeated track_id values across frames
- Confirming multiple frames were logged

Verification code:

```
df.head()
```

Repeated IDs confirm that tracking is working correctly.

9. What This Module Achieves

By implementing this project, we achieved:

- Conversion of raw video into structured data
- Persistent person identification
- Reliable movement logging
- Foundation for behavior analysis

This module does **not** detect crimes directly, but it enables all higher-level intelligence.

Role in Full Chain-Snatching Detection System

Tracking Data Logger



Motion & Speed Analysis



Interaction Detection



Pose-Based Verification



Chain Snatching Alert

Thus, the Tracking Data Logger is the **backbone** of the entire system.

Code:

```
import cv2

import pandas as pd

from ultralytics import YOLO

import supervision as sv

# DIRECT VIDEO PATH (KAGGLE)

VIDEO_PATH = "/kaggle/input/cctv-videos/Chain_Snatching_at_PDA.mp4"

OUTPUT_CSV = "/kaggle/working/tracking_data.csv"

# LOAD MODELS

model = YOLO("yolov8n.pt")

tracker = sv.ByteTrack()

# OPEN VIDEO

cap = cv2.VideoCapture(VIDEO_PATH)

if not cap.isOpened():

    raise RuntimeError("Cannot open video")

frame_id = 0

log_data = []

# PROCESS VIDEO

while True:

    ret, frame = cap.read()
```

```

if not ret:

    break

frame_id += 1

results = model(frame)[0]

detections = sv.Detections.from_ultralytics(results)

detections = detections[detections.class_id == 0]

detections = tracker.update_with_detections(detections)

for bbox, track_id in zip(detections.xyxy, detections.tracker_id):

    if track_id is None:

        continue

    x1, y1, x2, y2 = map(int, bbox)

    log_data.append([frame_id, int(track_id), x1, y1, x2, y2])

cap.release()

df = pd.DataFrame(log_data, columns=["frame", "track_id", "x1", "y1", "x2", "y2"])

df.to_csv(OUTPUT_CSV, index=False)

print("CSV saved to:", OUTPUT_CSV)

print("Frames processed:", frame_id)

```

Conclusion

The Tracking Data Logger successfully transforms unstructured video input into structured, time-based tracking data. By integrating person detection with multi-object tracking and CSV-based logging, the system provides reliable temporal information required for real-time surveillance intelligence. This implementation demonstrates system-level understanding and forms a strong foundation for advanced crime detection applications.

Code executed in Kaggle link:

<https://www.kaggle.com/code/anushreeu04/notebook5bd861a1b4>