Code Analysis Report

Benjosm/video-object-tracker

Generated on: Oct 14, 2025, 4:46 PM

Issue Summary

Potential Bugs	3
Style Issues	3
Security Concerns	0
Incomplete Code	1
Performance Concerns	1
Total Issues	**8**

Reports

Build Report

```
**Build Process Report**

**Command Executed:**
`pip install -r requirements.txt`

**Status:**
```

```
Failed (Terminated due to inactivity timeout)

**Duration:**
The process was active for over 1 minute before termination was manually
initiated.

**Build Output/Logs:**
```
Looking in links: https://download.pytorch.org/whl/cpu
Collecting torch
  Downloading torch-2.8.0-cp39-cp39-manylinux_2_28_x86_64.whl (888.0 MB)
```
The log shows that the process hung during the download of the `torch`
package. The download did not complete, and attempts to read the process
output later failed with an `ENXIO` error, indicating the process, while
listed as running, might not have been producing any output.

**Verification of Installation:**
After terminating the build process, `pip list` was run to check installed
packages. The output confirmed that none of the required dependencies (e.g.,
torch, torchvision, opencv-python-headless) were successfully installed. The
installed packages listed are the base system packages.

**Conclusion:**
The build process failed to complete due to the lengthy download time of
large dependencies (specifically PyTorch) which caused an inactivity
timeout. The environment was unable to install the required packages, and
the application cannot be executed in its current state. A more stable
network, increased timeout limits, or a pre-configured environment with
these dependencies is required to complete the build successfully.
```

Test Report

```
Ran the applications tests. No test files or testing frameworks were found.
The `README.md` is empty and provides no instructions on testing.
Conclusion: The application has no tests.
```

Run Report

```
Application testing summary: The video_tracker.py application has been
partially tested. Key findings: 1) The application successfully downloads
and loads the YOLOv5n model (yolov5n.pt) when not present. 2) Command-line
argument parsing works correctly as shown by the help output. 3) The SORT
```

```
tracker is properly configured with IOU threshold of 0.3 and max age of 10.
4) The application attempts to open video sources but fails in our
environment due to lack of webcam access and inability to download test
videos. 5) The core dependencies (OpenCV, PyTorch, ultralytics) appear to be
functional based on successful model loading. 6) The video processing
pipeline cannot be fully tested without a video source. The application is
structured correctly with video_tracker.py as the main entry point (main.py
is deprecated). The tracking system combines YOLO for object detection with
SORT for object tracking across frames.
```

File Descriptions & Issues

Video Object Tracker

This feature provides a complete solution for object detection and tracking in video sequences. It uses YOLOv5 nano for object detection and the SORT algorithm for tracking objects across frames. The video_tracker.py file serves as the main interface, while sort_tracker.py implements the SORT algorithm used for tracking. Additionally, it includes a standalone, non-web application for real-time object detection, classification, and tracking. The application uses video processing, headless operation, and is containerized with Docker. The project plan outlines the use of Python, OpenCV, PyTorch, and SORT for the implementation.

Video Object Tracker

This feature provides a complete solution for object detection and tracking in video sequences. It uses YOLOv5 nano for object detection and the SORT algorithm for tracking objects across frames. The video_tracker.py file serves as the main interface, while sort_tracker.py implements the SORT algorithm used for tracking. Additionally, it includes a standalone, non-web application for real-time object detection, classification, and tracking. The application uses video processing, headless operation, and is containerized with Docker. The project plan outlines the use of Python, OpenCV, PyTorch, and SORT for the implementation.

Video Object Tracker

This feature provides a complete solution for object detection and tracking in video sequences. It uses YOLOv5 nano for object detection and the SORT algorithm for tracking objects across frames. The video_tracker.py file serves as the main interface, while sort_tracker.py implements the SORT algorithm used for tracking.

video_tracker.py

This Python script is a video object tracker that uses the YOLOv5 nano model for object detection and the SORT (Simple Online and Realtime Tracking) algorithm for tracking objects across video frames. The script processes an input video file or camera feed, detects objects in each frame, tracks them using the SORT algorithm, and writes the tracked objects to an output video file. The script also provides real-time feedback on the processing progress, including the current frame number and frames per second (FPS).

The script starts by parsing command-line arguments for the input video file path or camera index, the output video file path, and a confidence threshold for object detection. It then loads the YOLOv5 nano model for object detection, which is optimized for CPU efficiency. The script opens the input video file and retrieves its properties, such as frame width, frame height, and FPS. It then initializes a VideoWriter object to write the tracked objects to the output video file.

The script initializes the SORT tracker with specific parameters, including the maximum age of a track, the minimum number of hits required to establish a track, and the intersection-over-union (IoU) threshold for associating detections with tracks. The script then enters a loop to process each frame of the input video. For each frame, it performs object detection using the YOLOv5 model, extracts the detections, and filters them based on the confidence threshold. The detections are then converted from the xyxy format to the xywh format and passed to the SORT tracker for updating the tracks.

The script draws bounding boxes and track IDs on the frame for each tracked object, using a color based on the class ID for better visualization. The frame is then written to the output video file. The script provides real-time feedback on the processing progress, including the current frame number and FPS, every 30 frames. The script also includes commented-out code for displaying the resulting frame in a window and allowing the user to quit the processing by pressing the 'q' key.

The script handles exceptions that may occur during processing and ensures that all resources are properly released, including the input video file, the output video file, and any OpenCV windows. Finally, the script prints a summary of the processing, including the total number of frames, the total processing time, and the average FPS.

sort_tracker.py

• **Potential Bugs:**

- 1. The `predict` method in the `KalmanBoxTracker` class checks if `(self.kf.x[6] + self.kf.x[2]) <= 0` and sets `self.kf.x[6] *= 0.0`. However, this condition might not be sufficient to handle all cases of negative scale or width, and the fix might not be appropriate for all scenarios.
- 2. The `update` method in the `Sort` class includes a condition `(self.frame_count >= self.min_hits or self.frame_count <= self.min_hits)` to determine whether to include a tracker in the output. This condition is likely a bug and should be `(self.frame_count >= self.min_hits)`.
- 3. The `associate_detections_to_trackers` method in the `Sort` class might not handle cases where there are no detections or no trackers correctly, as it directly returns empty arrays in those cases.

• **Style Issues:**

- 1. The `iou` function and the `KalmanBoxTracker` class are not documented with docstrings for their parameters and return values.
- 2. The `update` method in the `Sort` class has a comment that mentions returning `[x,y,w,h,track_id,class]`, but the actual return value is `[x,y,w,h,track_id,cls]`. The comment should be updated to match the actual code.
- 3. The `associate_detections_to_trackers` method in the `Sort` class has a complex logic for handling matches, unmatched detections, and unmatched trackers. This logic could be simplified and made more readable.

• **Performance Concerns:**

- 1. The `associate_detections_to_trackers` method in the `Sort` class uses a nested loop to compute the IoU matrix, which has a time complexity of O(n*m), where n is the number of detections and m is the number of trackers. This could be a performance bottleneck for large numbers of detections or trackers. Consider using a more efficient data association algorithm, such as the Hungarian algorithm, which has a time complexity of O(n^3) in the worst case but can be optimized for specific cases.

Video Object Tracker (1)

This feature is a standalone, non-web application for real-time object detection, classification, and tracking. It uses video processing, headless operation, and is containerized with Docker. The project plan outlines the use of Python, OpenCV, PyTorch, and SORT for the implementation. The main.py file is deprecated and has no functionality, but it is included in the project for historical reasons.

main.py

This file is a deprecated Python script. It serves no functionality as it is marked with a comment indicating that it has been deprecated and users should use 'video_tracker.py' instead. The file does not contain any code or logic, making it effectively empty and non-functional.

project_plan.txt

This file is a project plan for a standalone, non-web application that processes live video from a webcam or a saved video to detect, classify, and track objects in real-time. The application will run in a headless Docker container and output annotated video files and object tracking data to stdout. The project uses Python 3.9 with OpenCV for video I/O, PyTorch CPU-only for inference, and a custom SORT implementation for tracking. The plan includes detailed sections on user requirements, missing details filled in, technologies and setup, module and file plan, implementation strategy, run and build instructions, and a completion checklist.

Project Infrastructure

This feature includes the project's infrastructure files, such as the .gitignore file for version control, the requirements.txt file for dependency management, and the README.md file for project documentation. These files are essential for setting up the development environment and understanding the project's purpose and dependencies.

Project Infrastructure

This feature includes the project's infrastructure files, such as the .gitignore file for version control, the requirements.txt file for dependency management, and the README.md file for project documentation. These files are essential for setting up the development environment and understanding the project's purpose and dependencies.

.gitignore

This file is a .gitignore configuration file for a Python project. It specifies which files and directories should be ignored by Git, meaning they won't be tracked or committed to the repository. The file is organized into several sections, each targeting a specific type of file or directory.

The first section targets Python-specific files and directories, such as byte-compiled files (e.g., .pyc, .pyo, .pyd), optimized files (.Python), and virtual environment directories (env/, build/, etc.). This helps keep the repository clean and focused on the source code.

The second section is commented out and targets virtual environment files. If the project uses a virtual environment, these lines should be uncommented and adjusted to match the setup.

The third section targets IDE and editor-specific files, such as .idea/ (JetBrains IDE), .vscode/ (Visual Studio Code), and temporary files (*.swp, *~, .DS_Store).

The fourth section targets build artifacts, such as build/ and dist/ directories.

The fifth section targets temporary/generated files, such as output.mp4 and *.log files.

The sixth section targets video output files with various extensions (e.g., .mp4, .avi, .mov, etc.).

The seventh section targets PyTorch-specific cache files and directories, such as .torch/, torch/_C/*.so, and **/__pycache__.

requirements.txt

This file is a Python requirements file used to specify the dependencies needed for a project. It lists the external libraries and packages required for the project to function correctly. The file includes specific versions for some packages and version ranges for others. The packages listed are primarily used for machine learning, computer vision, and data analysis tasks.

README.md

This file is a README.md file for a project named "video-object-tracker". The README.md file is typically used to provide an overview of the project, its purpose, and instructions for setting up and using it. The content of the README.md file is currently empty, as indicated by the "=== README.md start ===" and "=== README.md end ===" markers.

- **Incomplete Code:**
  - The README.md file is incomplete as it contains no content.