Object Tracker using OpenCV

1. Introduction

This documentation provides an overview of various object tracking algorithms available in OpenCV 4.8.0 and focuses on the Kernelized Correlation Filters (KCF) algorithm. Object tracking plays a crucial role in computer vision applications such as surveillance, augmented reality, and autonomous navigation.

2. Available Algorithms

a. BOOSTING (Booosting)

This tracker is based on an online version of AdaBoost — the algorithm that the HAAR cascade based face detector uses internally. This classifier needs to be trained at runtime with positive and negative examples of the object. The initial bounding box supplied by the user (or by another object detection algorithm) is taken as a positive example for the object, and many image patches outside the bounding box are treated as the background.

Given a new frame, the classifier is run on every pixel in the neighborhood of the previous location and the score of the classifier is recorded. The new location of the object is the one where the score is maximum. So now we have one more positive example for the classifier. As more frames come in, the classifier is updated with this additional data.

Pros:

- ✧ Robust performance in various scenarios.
- ✧ Suitable for tracking objects with complex motion patterns.
- ✧ Handles occlusion and background clutter effectively.
- ✧ This algorithm is a decade old and works ok , but I could not find a good reason to use it especially when other advanced trackers (MIL, KCF) based on similar principles are available.

Cons:

- ✧ Sensitive to changes in lighting conditions and object appearance.
- ✧ Prone to drift when tracking objects with non-rigid motion.
- ✧ Tracking performance is mediocre. It does not reliably know when tracking has failed.

b. MIL (Multiple Instance Learning)

This tracker is similar in idea to the BOOSTING tracker described above. The big difference is that instead of considering only the current location of the object as a positive example, it looks in a small neighborhood around the current location to generate several potential positive examples. You may be thinking that it is a bad idea because in most of these “positive” examples the object is not centered.

This is where Multiple Instance Learning (MIL) comes to rescue. In MIL, you do not specify positive and negative examples, but positive and negative “bags”. The collection of images in the positive bag are not all positive examples. Instead, only one image in the positive bag needs to be a positive example.

In our example, a positive bag contains the patch centered on the current location of the object and also patches in a small neighborhood around it. Even if the current location of the tracked object is not accurate, when samples from the neighborhood of the current location are put in the positive bag, there is a good chance that this bag contains at least one image in which the object is nicely centered.

Pros:

- Effective tracking of objects with appearance variations.
- Robust to partial occlusions and changes in lighting conditions.
- Well-suited for tracking objects with consistent textures or patterns.
- The performance is pretty good. It does not drift as much as the BOOSTING tracker and it does a reasonable job under partial occlusion.
- If you are using OpenCV 3.0, this might be the best tracker available to you. But if you are using a higher version, consider KCF.

Cons:

- Limited scalability for tracking multiple objects simultaneously.
- Requires sufficient training data to adapt to object appearance changes.
- Tracking failure is not reported reliably. Does not recover from full occlusion.

c. KCF (Kernelized Correlation Filters)

KFC stands for Kernelized Correlation Filters. This tracker builds on the ideas presented in the previous two trackers. This tracker utilizes the fact that the multiple positive samples used in the MIL tracker have large overlapping regions. This overlapping data leads to some nice mathematical properties that are exploited by this tracker to make tracking faster and more accurate at the same time.

Pros:

- ✧ Robust tracking performance in various scenarios, including scale and rotation changes.
- ✧ Efficient implementation suitable for real-time applications.
- ✧ Handles occlusion and cluttered backgrounds effectively.
- ✧ Utilizes a kernelized correlation filter for accurate and robust tracking.
- ✧ Accuracy and speed are both better than MIL and it reports tracking failure better than BOOSTING and MIL. If you are using OpenCV 3.1 and above, I recommend using this for most applications.

Cons:

- ✧ Sensitive to abrupt changes in object appearance.
- ✧ May suffer from drift when tracking objects with irregular motion patterns.
- ✧ Does not recover from full occlusion.

d. **TLD (Tracking, Learning, and Detection)**

TLD stands for Tracking, learning, and detection. As the name suggests, this tracker decomposes the long term tracking task into three components — (short term) tracking, learning, and detection. From the author's paper, "The tracker follows the object from frame to frame. The detector localizes all appearances that have been observed so far and corrects the tracker if necessary.

The learning estimates detector's errors and updates it to avoid these errors in the future." This output of this tracker tends to jump around a bit. For example, if you are tracking a pedestrian and there are other pedestrians in the scene, this tracker can sometimes temporarily track a different pedestrian than the one you intended to track. On the positive side, this track appears to track an object over a larger scale, motion, and occlusion. If you have a video sequence where the object is hidden behind another object, this tracker may be a good choice.

Pros:

- ✧ Adaptive tracking with online learning capabilities.
- ✧ Handles significant changes in appearance and abrupt motion.
- ✧ Incorporates a detection module for re-detection of lost objects.
- ✧ Works the best under occlusion over multiple frames. Also, tracks best over scale changes.

Cons:

- ✧ Prone to false positives, especially in cluttered scenes.
- ✧ High computational complexity, limiting real-time performance on resource-constrained devices.
- ✧ Lots of false positives making it almost unusable.

e. **MEDIANFLOW (Median Flow)**

Internally, this tracker tracks the object in both forward and backward directions in time and measures the discrepancies between these two trajectories. Minimizing this ForwardBackward error enables them to reliably detect tracking failures and select reliable trajectories in video sequences.

In my perspective I found this tracker works best when the motion is predictable and small. Unlike, other trackers that keep going even when the tracking has clearly failed, this tracker knows when the tracking has failed.

Pros:

- ✧ Robust tracking of objects with moderate motion.
- ✧ Handles occlusions and abrupt changes in scale.
- ✧ Suitable for real-time applications due to its computational efficiency.
- ✧ Excellent tracking failure reporting. Works very well when the motion is predictable and there is no occlusion.

Cons:

- ✧ Limited effectiveness for tracking objects with fast or erratic motion.
- ✧ May struggle with objects undergoing significant deformations.
- ✧ Fails under large motion.

f. **CSRT (Channel and Spatial Reliability Tracking)**

In the Discriminative Correlation Filter with Channel and Spatial Reliability (DCF-CSR), we use the spatial reliability map for adjusting the filter support to the part of the selected region from the frame for tracking. This ensures enlarging and localization of the selected region and improved tracking of the non-rectangular regions or objects. It uses only 2 standard features (HoGs and Colornames). It also operates at a comparatively lower fps (25 fps) but gives higher accuracy for object tracking.

Pros:

- ✧ Robustness to challenging scenarios like occlusion and fast motion.
- ✧ Accurate and stable tracking performance.
- ✧ Combines spatial and channel reliability for improved tracking accuracy.

Cons:

- ✧ Higher computational complexity compared to some other algorithms.
- ✧ May require more computational resources, limiting real-time performance on low-end hardware.

3. Selected Algorithm: KCF

Ease of Use: KCF is relatively easy to implement and use, especially with libraries like OpenCV providing ready-to-use implementations. Developers can quickly integrate KCF into their projects without extensive knowledge of complex algorithms.

Efficiency: KCF is computationally efficient, allowing it to achieve real-time performance on a variety of hardware platforms, including embedded systems and mobile devices. This efficiency is crucial for applications requiring low-latency tracking, such as robotics or augmented reality.

Versatility: KCF is versatile and can handle various tracking scenarios, including objects with different appearances, scales, and motion patterns. This flexibility makes it suitable for a wide range of applications, from surveillance to sports analysis.

Adaptability: KCF can adapt to changes in the tracked object's appearance over time. This adaptability is valuable in dynamic environments where lighting conditions, occlusions, or background clutter may vary.

Accuracy: KCF provides accurate tracking results, minimizing errors and reducing the likelihood of losing track of the target object. This accuracy is essential for applications where precise object localization is crucial, such as medical imaging or autonomous vehicles.

Community Support: KCF is a well-established algorithm with a strong community of developers and researchers. This community support ensures ongoing development, bug fixes, and optimizations, making KCF a reliable choice for long-term projects.

Parameter Tuning: KCF offers various parameters that developers can tune to optimize tracking performance for specific applications or environments. This flexibility allows developers to fine-tune the algorithm to achieve the desired balance between tracking accuracy and computational efficiency.

Integration with Existing Tools: KCF integrates seamlessly with existing computer vision libraries and frameworks, such as OpenCV, making it accessible to a wide range of developers familiar with these tools. This integration simplifies the development process and reduces implementation time.

By considering these developer perspective reasons, we make informed decisions about using KCF for object tracking in our projects.