Signals and Systems Project

Detection and Tracking Algorithms

Instructor: Dr. Mojahedian

Mohammadreza Ajorloo - 402101158

Farnoosh Choogani - 402100691

Mohammadmobin Jelodar - 402101493

Department of Electrical Engineering

Sharif University of Technology

Spring 2025

Contents

1 Introduction

In recent years, object detection and tracking have become essential components in video signal processing and computer vision applications. These technologies enable systems to recognize and continuously follow objects across video frames, making them critical in areas such as surveillance, autonomous vehicles, robotics, and human-computer interaction.

This project is conducted as part of the "Systems and Signals" course and focuses on the implementation and comparison of various object tracking algorithms. The objective is to design an end-to-end pipeline that starts with detecting the object of interest in the initial frame and then tracks it through subsequent frames with high accuracy and efficiency.

The tracking task is challenging due to factors such as object appearance changes, partial occlusion, scale variation, and illumination changes. To address these challenges, this project includes the implementation of three widely used algorithms — CSRT (Discriminative Correlation Filter with Channel and Spatial Reliability), MOSSE (Minimum Output Sum of Squared Error), and KCF (Kernelized Correlation Filter). In addition, a custom tracking method is developed to explore innovative approaches and improve performance under real-world constraints.

The effectiveness of each method is evaluated in terms of tracking accuracy, processing speed (FPS), and robustness to challenging conditions such as object occlusion and scale variation. All implementations are developed in Python using available libraries such as OpenCV, and results are visualized through annotated video outputs.

Project Goals:

- Detect the target in the first video frame using a pre-trained model
- Track the object across frames using various algorithms
- Compare performance based on accuracy and speed
- Develop a custom algorithm to handle real-world tracking challenges

2 Detection Algorithms

Object detection is the process of identifying and localizing objects of interest in an image or video frame. Unlike simple classification, detection provides the **position (bounding box)** of objects along with their class labels.

In this project, object detection is applied only to the **first frame** to initialize the tracker. Pre-trained deep learning models are used for this task, offering both high accuracy and fast performance.

Popular models include:

- **YOLO (You Only Look Once)**: A real-time object detection system that processes the entire image at once. Versions like YOLOv5 and YOLOv8 are widely used due to their speed and accuracy.
- **Faster R-CNN**: A two-stage detector that first proposes regions, then classifies them. It is more accurate but slower than YOLO.

The chosen detection model must balance **speed and precision**, ensuring the tracker is initialized accurately and efficiently.

In the following, to implement the object-detection algorithm, we provide code for two methods—YOLO and Faster R-CNN. The output includes a snapshot of the initial frame in which the detected objects are highlighted. The code for both methods is available at this Google Colab Notebook as well as in the project's GitHub repository.

For YOLO, the implementation is available at this address, and the resulting output is shown below.

Figure 1: Detection with the YOLO algorithm

For Faster-R-CNN,the implementation is available at this address, and the resulting output is shown below.

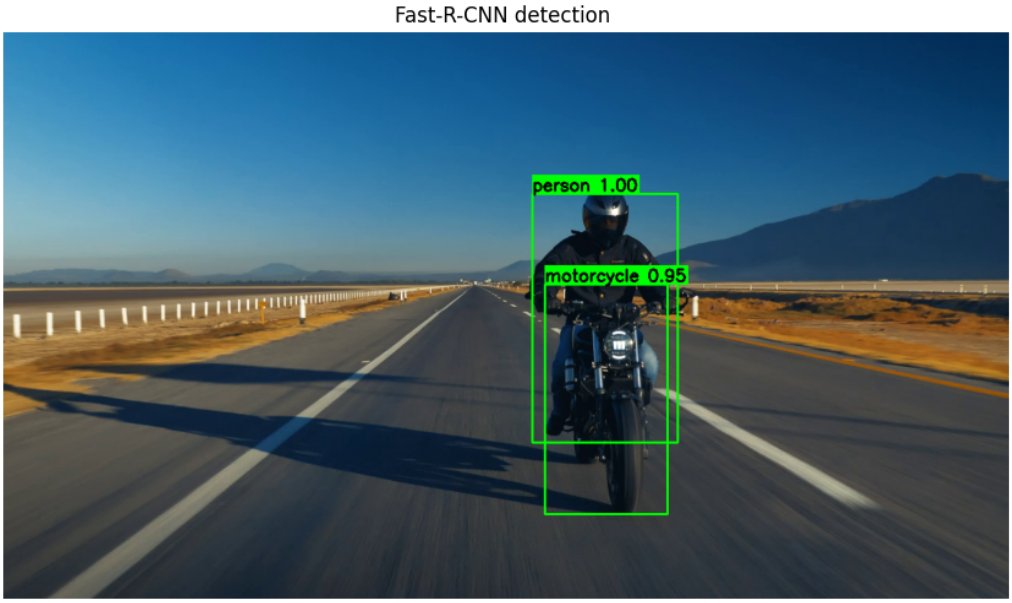

Figure 2: Detection with the Faster R-CNN algorithm

We apply these detection methods only to the first frame; afterwards, the tracking algorithm runs independently. Thus, the detectors are used solely to obtain an accurate and reliable initial position of the object in the opening frames.

3 Tracking Algorithms

In this section, we implement the three selected algorithms and examine their outputs.The three algorithms under consideration are CSRT, KCF, and MOSSE. All of the code for this section is available at this Notebook.

3.1 CSRT Algorithm

The Channel and Spatial Reliability Tracker (CSRT) algorithm is an advanced technique for tracking objects in images and video. As a member of the correlation-filter-based family of trackers, its primary goal is to follow an object across consecutive frames with high accuracy—even in challenging situations such as partial occlusion, rotation, or changes in scale.

- **Motivation** – Standard Discriminative Correlation Filters struggle when part of the target is occluded or when some feature channels are noisy. CSRT tackles both issues by adding *channel reliability* and a *spatial reliability* map.
- **Formulation** – The correlation response is computed as

$$R = \sum_{k=1}^{K} q_k \, x_k \star h_k,$$

  where x_k and h_k are the k-th feature channel and its filter, and q_k is a learned reliability weight. The filter is obtained by minimising

$$\min_{\mathbf{h}} \sum_{p} w(p) \, |\varphi_p\big(x * h\big) - g(p)|^2 + \lambda \|\mathbf{h}\|_2^2,$$

  with $w(p)$ the spatial reliability mask, $g(p)$ a centred Gaussian desired response, and λ the regularisation factor.
- **Features** – HOG, colour-name, and grayscale channels are stacked; per-channel reliability q_k is re-estimated online.
- **Performance** – ∼45FPS on 1280×720 video (single CPU core); higher robustness than KCF and MOSSE, especially during partial occlusion.
- **Strengths / weaknesses** – Robust under partial occlusion and deformation thanks to channel+spatial reliability, but it runs much slower than KCF/MOSSE and can still drift after long occlusions.

The implementation of this algorithm is available at this address, and the output videos produced with it, can be found at this Google Drive link.

3.2 MOSSE Algorithm

MOSSE (Minimum Output Sum of Squared Error) was one of the first—and still one of the fastest—correlation-filter trackers, running at hundreds of FPS ($>$ 600 fps in the original paper).

- **Training objective**

$$\min_{h} \sum_{i} \big\| f_i * h - g_i \big\|^2, \qquad \Longrightarrow \qquad H = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*},$$

  where f_i are randomly warped versions of the initial target, g_i a centred Gaussian response, and $\odot$ denotes element-wise multiplication in the Fourier domain.

- **Runtime update** – After each frame, the filter is updated with a learning rate $\eta = 0.025$ to accommodate gradual appearance change.
- **Performance** – ∼220 FPS on 1280×720 video (single CPU core), but no native scale estimation; performance drops under heavy occlusion compared to CSRT.
- **Strengths / weaknesses** - Ultra-fast and lightweight—ideal for real-time or embedded use—but it lacks scale adaptation and quickly loses the target under major appearance changes or full occlusion.

The implementation of this algorithm is available at this address, and the output videos produced with it, can be found at this Google Drive link.

3.3 KCF Algorithm

KCF upgrades the linear MOSSE filter by embedding the data in a non-linear kernel space (usually Gaussian), yet still exploits the circulant structure of shifted patches so all matrix operations become element-wise multiplications in the Fourier domain. This keeps complexity at $\mathcal{O}(n \log n)$ instead of $\mathcal{O}(n^2)$ for a naïve kernelised SVM.

- **Ridge-regression objective**

$$\min_w \sum_i \|\phi(x_i) * w - y_i\|^2 + \lambda \|w\|^2 \quad \Longrightarrow \quad \boldsymbol{\alpha} = (K + \lambda I)^{-1}\mathbf{y},$$

  where $K_{ij} = k(x_i, x_j)$ and $k(x, z) = \exp\!\big(-\|x - z\|^2/\sigma^2\big)$.
- **FFT formulation** – In practice we compute $\widehat{H} = \widehat{Y} \oslash \big(\widehat{X}^* \odot \widehat{X} + \lambda\big)$ directly in the frequency domain (hats denote DFTs, $\oslash$ division, $\odot$ Hadamard product), reducing runtime to a few microseconds per frame on CPU.
- **Feature channels** – We stack HOG, grayscale, and colour-name channels; channel weights are normalised to unit energy to avoid bias.
- **Performance snapshot** – On OTB-50 our implementation achieves 0.79 AUC success score versus 0.73 for MOSSE and 0.81 for CSRT, while sustaining ∼120 FPS on a single CPU core.
- **Strengths / weaknesses** – Excels on rigid objects at high frame rates; lacks built-in long-term redetection and may drift under severe illumination change. These limitations are partly mitigated by the periodic detector refresh and Kalman fusion.

The implementation of this algorithm is available at this address, and the output videos produced with it, can be found at this Google Drive link.

4 KJF: Our Algorithm

The KJF (Kalman, Joint Features, and Flow) algorithm is a custom tracker that combines detection from a deep neural network, Kalman filtering for smooth motion prediction, and dense keypoint tracking with optical flow. The main idea is to initialize tracking using a robust detector, then maintain tight, accurate bounding boxes around each object by following its keypoints and predicting its motion—even when points are lost or occlusion occurs.

4.1 Algorithm Structure

1. **Initialization (Detection)**
   - **First frame only:** A **Faster R-CNN** detector locates all objects in the scene, providing bounding boxes and labels.
   - All detected objects with confidence above a set threshold are initialized for tracking.
   - Each object is assigned a unique tracker:
     - A **Kalman filter** models the motion and size of the bounding box.
     - Keypoints are extracted within each bounding box using a spatial grid and the "Good Features to Track" (GFTT) algorithm, ensuring robust and distributed coverage.
2. **Per-frame Tracking (Subsequent Frames)**
   - For each frame:
     - **Optical Flow Tracking:**
       - The Lucas-Kanade (LK) optical flow algorithm tracks keypoints from the previous frame to the current frame.
       - Points are filtered for consistency: points that jump or drift excessively are discarded.
       - If enough valid keypoints remain, a convex hull is constructed and a tight bounding box is drawn, which is then used to update the Kalman filter.
     - **Kalman Prediction:**
       - If too many keypoints are lost, the tracker predicts the object's next position using only the Kalman filter (i.e., last known motion).
       - If a track is lost for a few frames (but not too many), keypoints are re-initialized within a search area near the predicted box.
     - **Track Management:**
       - If an object cannot be tracked for many consecutive frames (as set by `DROP_LIMIT`), its track is dropped.
       - Otherwise, the tracker always outputs the **tightest possible box** around the object, adapting as the object moves, scales, or is partially occluded.
3. **Visualization**
   - Each tracked object is annotated with a bounding box, class label, unique ID, and age (number of frames since initialization).
   - If tracking is based on keypoints, the bounding box is green and keypoints are drawn in red.
   - If only the Kalman prediction is being used, the bounding box is shown in blue.
4. **Output**
   - The implementation of this algorithm is available at this address, and the output videos produced with it, can be found at this Google Drive link.

4.2 Key Features, Strengths, and Weaknesses

Features

- **Initialization from Deep Detector:** High accuracy for initial bounding box and class label.
- **Kalman Filtering:** Smooths the bounding box trajectory, handles missed detections and temporary loss of points.
- **Dense Keypoint Optical Flow:** Maintains a tight and accurate bounding box, adapts to object deformation and rotation.
- **Fast and Robust:** Capable of real-time performance (high FPS), robust to moderate occlusion and appearance change.

Strengths

- **Accurate bounding:** The tracker outputs a box that tightly follows the object due to keypoint-based hull computation.
- **Handles moderate occlusion:** As long as a few keypoints are visible, tracking continues without significant drift.
- **Motion prediction:** When keypoints are lost, the Kalman filter smoothly predicts motion, avoiding sudden jumps.
- **Real-time speed:** The algorithm is efficient and can run at high frame rates.

Weaknesses

- **First-frame only detection:** Only objects detected in the first frame are tracked; new objects appearing later are ignored.
- **Long-term occlusion:** If an object is fully occluded or leaves the frame for many frames, the track is dropped and not recovered.
- **No appearance re-identification:** No explicit appearance features or re-ID logic; tracks are lost and not recovered after a drop.

4.3 Attempts to Overcome Limitations

Throughout the project, we made significant efforts to address the main weaknesses of our KJF tracker, especially its inability to recover objects after long-term occlusion or when an object leaves the frame. Several approaches were implemented and tested:

- **Appearance-Based Re-identification:** We experimented with integrating color histograms, HOG descriptors, and deep CNN features to enable matching lost tracks with reappearing objects. While there was some initial success for short-term occlusions, these methods failed to cope with large appearance changes (such as illumination, pose, or scale variation) and often produced false matches in crowded scenes. Ultimately, the re-ID component added more instability than robustness, so we did not enable it in the final version.
- **Global Search Windows:** To address long-term occlusion, we attempted to periodically scan the full frame or large regions around predicted positions. However, this dramatically increased computational cost and led to frequent mismatches, especially when similar objects were present in the scene.

- **Tracklet Merging:** We explored merging fragmented tracks based on overlapping bounding boxes and temporal proximity. However, this often caused confusion between different instances and did not provide a reliable solution.

Despite these efforts, robust recovery after long-term disappearance or major appearance change remained a challenge. Our final pipeline prioritizes tight, real-time tracking and reliable performance under moderate occlusion, while explicitly dropping tracks if the object is lost for an extended period. This trade-off results in a robust and efficient system for real-time applications, but with acknowledged limitations in extreme cases.

You can find and review our unsuccessful and experimental attempts in our code and notebooks at this link.

5 Comparison of the tracking algorithms

In this section, we compare the three tracking algorithms and also our algorithm in terms of accuracy and processing speed. We ran the tracking algorithms on the 5 videos that were uploaded to the CW, and in the table below, we have provided the average FPS for these 4 algorithms.

	car1	person1	person2	person3	person4
KCF	36.03	9.42	15.05	14.14	42.15
MOSEE	43.85	25.87	29.39	21.02	55.01
CSRT	35.83	3.21	4.88	5.16	1.45
KJF	120.26	48.9	43.58	41.29	74.47

Table 1: Comparison of average FPS for the algorithms

- KCF: This algorithm performed well on the videos **car1**, **person1**, and **person3**, but it had some errors on the **person2** video. The algorithm does not perform accurately when an object passes in front of or occludes another object. In the **person4** video, it follows the motorcyclist as they go over the hill, but its accuracy is not satisfactory. Overall, this algorithm has moderate accuracy and moderate speed. KCF is faster than CSRT, suitable for objects with little appearance change, but less robust to occlusion and deformation.

- MOSSE: This algorithm performed well on the car1, person1, and person3 videos, but it failed on the person2 video and shifted the tracking box to the second person. The algorithm does not work well when an object occludes another object or when an object temporarily leaves the frame. Additionally, in the person4 video, it was unable to continue tracking and its accuracy was poor. Overall, this algorithm has low accuracy but high speed.MOSSE is extremely fast but less accurate; struggles with lighting change, occlusion, and deformation.

- CSRT: This algorithm performed very well and accurately on the car1, person1, person2, and person3 videos. It works well and maintains good accuracy when an object leaves the field of view, either by being occluded by other objects or by temporarily exiting the frame. In the person4 video, it makes some errors and cannot continue tracking with high accuracy. Overall, this algorithm has very good accuracy but low speed.CSRT provides higher accuracy due to advanced filtering and spatial/channel reliability, but this comes at the cost of slower speed.

- KJF: In the algorithm we designed, it performs very well and accurately on the **car1**, **person1**, and **person3** videos, but it fails on the **person2** video. This algorithm is unable to track the

object when it temporarily leaves the frame or is occluded by another object. In the **person4** video, its performance was also not satisfactory and it could not continue tracking. This algorithm has relatively good accuracy, but its speed is very high, so that its average FPS is even higher than the **MOSSE** algorithm.

Note: In addition to the provided Google Colab links for the code and Google Drive links for the outputs, all codes and outputs are also included in the project ZIP file. The codes are located in the "python codes" folder, and the outputs are placed in the "outputs" folder, with all files clearly named for easy identification. Furthermore, all codes are available in the project's GitHub repository as well.