Beyond one view - intersection surveillance using multiple cameras

Olaeriu Vlad Mihai

June 2025

1 Introduction

Task: the goal of this project is to develop a fully automatic video analysis pipeline that enables multi-camera surveillance of a single road intersection. Given streams from multiple fixed cameras that monitor the intersection, the system should perform the following tasks:

1. *Temporal localisation*: find a 3-second query video from one camera (A) inside a 30-second reference video from another camera (B), and report the exact frame index at which the query begins in camera B's timeline;
2. *Cross-view single vehicle tracking*: given a single tight bounding box around one vehicle in one camera (A), track this vehicle through the streams of cameras A and another camera (B) despite possible changes in viewpoint, scale, and occlusion of the vehicle;
3. *Directional traffic counting*: given two overlapped streams from two cameras (A and B), count every vehicle that moves from a user-defined origin direction to a destination direction across the two videos, producing per-direction flow statistics.

In my project, I only tackled the second task, the cross-view single vehicle tracking. Next, I will elaborate my approaches towards reaching the tracking of the vehicle in both videos.

2 Tracking

There are many built-in object trackers in OpenCV: **Boosting** (based on AdaBoost), **MIL**, **KCF** (Kernelized Correlation Filters), **CSRT** (Discriminative Correlation Filter), from which the most performant are KCF and CSRT.

The first approach was to solely base the tracking of the car on the CSRT/KFC object tracker. However, a problem arose quickly: if a car was making a turn or if it moved far away, its orientation and size would change significantly, and

the tracker was unable to detect this. In this situation, **YOLO** appeared in the scene.

YOLO (You Only Look Once) is a popular real-time object detection system that frames object detection as a regression problem. Instead of classifying regions, it predicts bounding boxes and class probabilities simultaneously from full images in a single pass. In this situation, it was used for the reinitialization of the tracker: after a certain number of frames were processed (10), all the vehicles (cars and trucks) were identified using YOLO, and the bounding box of the one with the highest IOU with the last object identified by the tracker was chosen as the new bounding box.

This change significantly improved the accuracy of the object tracker. Unfortunately, YOLO does not work well when the object is occluded. For this issue, to anticipate the direction in which the car might be moving, a **Kalman Filter** was used.

A Kalman Filter is an optimal estimation algorithm that processes a series of measurements observed over time, containing noise and other inaccuracies, and produces estimates of unknown variables. In this case, the variables observed were: x, y, width and height of the bounding box, and also their velocities.

Unluckily, a Kalman Filter needed to update its internal functionality based on the tracker, and when the tracker started getting into a region where the car was occluded, this affected the predictions of the Kalman Filter. This led to no further improvements to the car tracker.

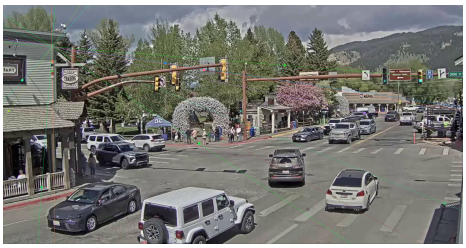
3 Vehicle cross-identification

To identify the car in camera B using the bounding box from camera A, I have tried to use **epipolar geometry**. For this, I have computed the fundamental matrix between the two scenes (camera A and camera B), using a **Flann** matcher and **RANSAC**.

After finding the **fundamental matrix**, I was able to extract the reference points between the two scenes, and the epipolar lines from the camera B corresponding to the points in camera A.

The next step was to identify every car in the frame from camera B and to compute the distance between the center of its bounding box to the epipolar line corresponding to the center of the bounding box of the car in camera A. The closest car would have been the match I was looking for. Unfortunately, as *Figure 1* shows, the reference points were very few and the epipolar lines were very messy.

(a) Points

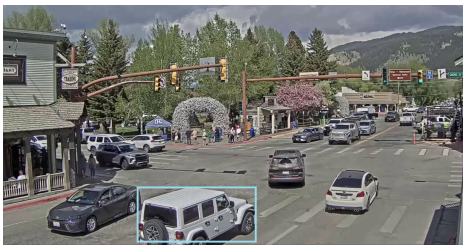


(b) Epipolar lines

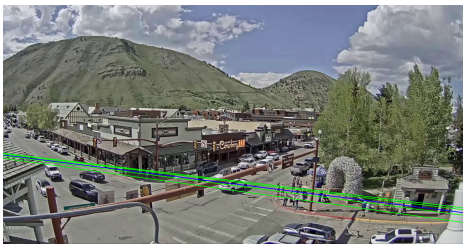
Figure 1: Epipolar geometry between camera views of the intersection

Even if the epipolar lines did not correspond to the true match between the two scenes, I have tried to see how it would behave on the train data. In *Figure 2*, you can see that the epipolar lines in camera B matched the bounding box from camera A pretty well. On the other hand, for another example, they were not working at all (see *Figure 3*). Even if any hyperparameter of the function that computed the fundamental matrix was tweaked a little bit, the output epipolar lines would differ too much.

Another approach was to use **Homography Projections**: as a reference, I picked up a 2D image of the scene from Google Earth, then I manually drew some references between the 2D images and each of the three cameras: A, B, C. The references were the zebras were pedestrians crossing the street because they appeared in the common field of view of the cameras. Afterward, my idea was to project the given bounding box from camera A onto the 2D plane, then to project back from the 2D onto camera B, using pre-computed Homography matrices. Unfortunately, this method did not lead to any improvements. Finally, I tracked the car only in the video where I was given the initial bounding box.



(a) Bounding box of the car in camera A



(b) Epipolar lines of the corners in camera B

Figure 2: Car detection in camera B from reference bounding box in camera A

(a) Bounding box of the car in camera A

(b) Epipolar lines of the corners in camera B

Figure 3: Car detection in camera B from reference bounding box in camera A