Big Brother: Multi-camera Object Tracking

Mohit Deshpande
The Ohio State University
deshpande.75@osu.edu

Brad Pershon
The Ohio State University
pershon.1@osu.edu

1 Introduction

Cameras are ever present in our society. They are on traffic lights, on top of buildings, inside buildings, and on vehicles. Millions of dollars go to law enforcement and cities to help stop crime using new surveillance systems [7]. Even universities are including cameras in their bus systems [5]. With this massive coverage of cameras, many of them overlap with each other, e.g., the field-of-view of a camera on a building might overlap with that of the camera on an adjacent building. In the event of a crime, law enforcement check all of these cameras and must manually track the perpetrator through potentially hundreds of cameras [7]. Instead, we present an automated solution, called `BigBrother`, that can follow a target across any number of cameras.

2 BigBrother

The problem of object tracking through multiple cameras $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$, given a target T in one camera, is to find and track that target in the other cameras. For simplicity, we assume that the target T is first identified in $\mathcal{C}_1$ and call this the **primary camera**. We immediately start mean-shift tracking for the centroid of the target's bounding box. This mean shift produces new coordinates for the target, and we can transform these coordinates to other cameras using a homography. We use a novel multi-scale variant of covariance tracking to match the target in the non-primary camera, where the initial covariance matrix is taken from the feature matrix of the target. Algorithm 1 describes our algorithm at a high level. The details are described in subsequent sections.

Algorithm 1 `BigBrother` Algorithm

1: **procedure** BigBrother($\mathcal{C}_1, \ldots, \mathcal{C}_N, T$)
2:     $\Sigma \leftarrow$ `feature_cov`($\mathcal{C}_1, T$)   ▷ Covariance matrix for other cameras
3:     $B_i \leftarrow \emptyset \quad \forall i \in \{2, \ldots, N\}$ ▷ Position of the target in other cameras
4:     **while** true **do**
5:         $T \leftarrow$ `meanshift`($\mathcal{C}_1, T$)   ▷ Update the target's position
6:         **for** $i \leftarrow 2$ to N **do**
7:             $T' \leftarrow \mathcal{H}(\mathcal{C}_1, \mathcal{C}_i) \cdot T$ ▷ Transform coordinates using homography
8:             **if** $T' \notin \mathcal{C}_i$ **then**
9:                 $B_i \leftarrow \emptyset$
10:                 **continue**
11:             **end if**
12:             $S \leftarrow$ `search_region`(T')   ▷ Define search region around mean-shift estimate
13:             $B_i \leftarrow$ `multiscale_cov_track`(Σ, S)   ▷ Find target in other cameras
14:         **end for**
15:     **end while**
16: **end procedure**

2.1 Homography

To track a target across multiple cameras, we must have either complete 3D knowledge of the space of the cameras and their positioning in that space or, more simply, a homography between the cameras. The homography is simpler but can only be computed if the field-of-view of the cameras overlap. This is a critical assumption we make. Given the large number of cameras with overlapping field-of-views, this assumption is reasonable.

We compute a homography offline by selecting corresponding pixels between the two camera views, and a corner or feature detector may make pixel selection easier. We require at least 4 corresponding coordinates to compute the homography, however, for a better homography we used 32 corresponding points. Each point is converted into homogeneous coordinates, and we solve the system of linear equations to produce the homography matrix H between two camera views. This is shown as $\mathcal{H}(\mathcal{C}_1, \mathcal{C}_i)$ in Algorithm 1.

For robustness, we use a normalized direct linear transform: we normalize our points to have zero mean and shift the average distance of the resulting points to the origin is $\sqrt{2}$.

2.2 Mean-Shift Tracking

After computing the homography and identifying the region of interest in one camera frame, we can start mean-shift tracking [1]. We assume the target's size is constant through the frames since we only track the centroid of the target. We use 16 color bins for the color quantization, circular neighbors with a radius of $r = 5$, and $h = 25$ for the Epanechnikov profile. The reason we use a smaller radius is because our dataset has a smaller resolution.

At each frame of the mean-shift tracking, we use the homography to transform the mean-shift-tracked centroid into the other cameras. If the homography is accurate, we can determine if the target is in a particular camera frame. If the transformed points are outside of bounds of a camera frame, we ignore that camera and know that target is not present. However, if the points lie inside of a camera frame, we know the target is present in that particular camera frame.

2.3 Covariance Tracking

To find the target in the other cameras, we use a novel multi-scale covariance tracking [6]. One primary issue with covariance tracking is speed: we have to search the entire image. However, we can the transformed mean-shift estimate to narrow our search space. We take the bounding box points from the target in the primary camera frame and transform them into other camera frames. Since the homography might produces points that are skew, we take the axis-aligned rectangle to be our search area. This approach drastically limits our search. On average, we found a 94.3% reduction in the number of pixels we must search.

The covariance matrix initialized from the original target in the primary camera. For our feature vector, we use the radial vector described in [6] for each pixel in the search space.

$$\mathbf{f} = [||(x', y')|| \quad I(x, y, 1) \quad I(x, y, 2) \quad I(x, y, 3)]$$

where $||(x', y')||$ is the radial distance from the center of the patch to the jth pixel in the patch. The last 3 components are the RGB components of the pixel.

The covariance matrix is initialized with a fixed patch in the primary camera. However, in other cameras, the target may have different scales. The scale of the target will most likely be different in another camera, but there's a relation between the scale of the target in another camera and the primary camera. We propose a new multi-scale approach, inspired by the Region Proposal Network [9], that changes the scale of the sliding window in the search space. We define K factors that scale the dimensions of the sliding window, equally along both dimensions. In addition to scaling the initial window, we also flip the dimensions, e.g., we use a sliding window of size 5×3 as well as 3×5, for every K scaling factor.

We perform covariance tracking by searching for the best, i.e., minimum-distance, match, computed using the metric described in [2]. However, doing this for K filters and flipped dimensions, we will have $2 \cdot K$ bounding boxes in the search space. We simply use non-maximal suppression, with an overlap ratio threshold of 0.05, to reduce this number and select the bounding box of the smallest distance. The speed of this algorithm depends on the size of the search space $|S|$ and the number of scaling factors K. However, in practice, K is much smaller compared to $|S|$, which dominates the time complexity.

We do not start mean-shift tracking in the other camera frame since we want to maintain the relationship of the target between several cameras. If we were to mean-shift track in all cameras, these would operate independently. This makes it impossible to pick up a target in a new camera since mean-shift would stop tracking after the target leaves a frame. To remedy this, we use the homography to connect the primary camera to the other cameras. If the target moves into the field-of-view of another camera, we will notice the project points will lie in the camera's field-of-view. This allows us to drop the target from a camera if it falls out of view and pick up the target in a camera if it comes into view.

2.4 Work Allocation

Mr. Pershon computed the homography H between the two cameras. This was computed by manually matching correspondence points and solving the linear system of equations as described in Section 2.1. He also implemented a highly-vectorized variant of mean-shift tracking for the primary camera.

Mr. Deshpande implemented the multi-scale covariance tracking and cross-camera interaction described in Section 2.3.

2.5 Challenges

Co-planar points. Homographies or other measurements are not entirely accurate; there is always some error associated with them. When transforming points between cameras, the transformed points are not that accurate. However, we can ameliorate this inconsistency by using more points when computing our homography. Since we use homographies for translating between different cameras, we only get reasonable results when the target is co-planar with the ground-plane. This is a limitation since many cameras, particularly indoor ones, do not have a high enough vantage point to satisfy the co-planar points requirements, or even approximate it reasonably.

Loss of target in primary camera. One assumption our algorithm makes is that the target remains in the same camera the entire during of our approach. If we lose the target in one camera but it is still present in another camera, we can simply switch from covariance tracking to mean-shift; that camera becomes our primary camera.

Vastly varying target sizes. The covariance matrix is computed from a patch of fixed size. When we perform the search, we compute the covariance of several scales. However, if our target does not match one of these scales, then we may miss the detection. However, since they're computed as a function of the original target's dimensions, we are likely to detect the target in the other cameras.

3 Results

For our experiments, we used a video sample from the PETS 2001 dataset. The video has two surveillance cameras in the daytime, each with 384×144 resolution. In particular, we track a moving van between these two cameras. A frame from both cameras at the same time step is shown in Figure 1. Figure 2 shows an example frame from our algorithm. All experiments, accuracy and performance, were conducted on an Intel i7@3.2GHz processor with 16GB of RAM and no parallelization. Our algorithm was implemented in MATLAB.

Figure 1: Example frames from our dataset. Our experiments track the moving white van.

Figure 2: Example frames from our algorithm. The blue box represents the projected search area, and the red box is the location of the target.

Figure 3 shows our algorithm's performance. `buildCovMatrix` is a subroutine inside of `covTracking` that constructs the covariance variance for each sliding window patch. `cov` is a subroutine inside of `buildCovMatrix` simply computes the covariance matrix Σ. It should be noted that `buildCovMatrix` and `cov` are called thousands of times, on average, since we must construct the covariance matrix for each location in the search area. This is where the bulk of the time is spent. The multi-scale covariance tracking clearly takes the longest amount of time. Mean-shift tracking, even with a large number of iterations, takes an order of magnitude less time than covariance tracking. Our algorithm's performance scales linearly if we reduce the number of scaling factors.

Table 1 shows our algorithm's precision, recall, accuracy, as well as the confusion matrices, as a function of the neighborhood radius of the mean-shift tracking algorithm. Since we had no ground-truth data, we randomly sampled 40 frames and had a human judge if the bounding box was on the target, i.e., the van. In our specific case, precision means, from the detected events, i.e., the "positives", the target is correctly inside of the bounding box, and, recall means our algorithm correctly detected the target whenever it was actually there.

We see that using a smaller mean-shift neighborhood radius produces better precision and recall in the other cameras. However, in the primary camera, mean-shift tracking suffers as a result. The smaller neighborhood radius is also more appropriate for the dimensions of the camera feed.
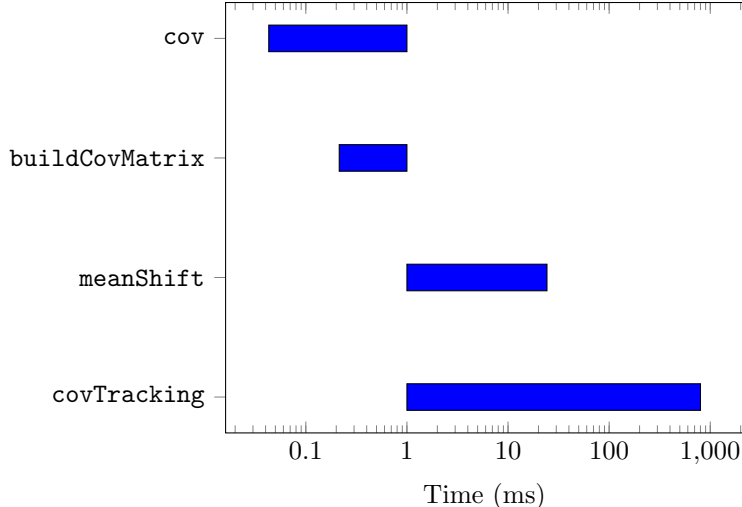
Figure 3: Performance of our components. Our multi-scale covariance tracking takes an order of magnitude longer than mean-shift tracking.

Radius	tp	fp	tn	fn	Precision	Recall	Accuracy
3	8	7	15	10	0.5333	0.4444	0.5750
5	10	7	12	11	0.5882	0.4762	0.5500
7	5	5	12	18	0.5000	0.2174	0.4250
20	5	5	10	20	0.5000	0.2000	0.2500

Table 1: Confusion matrices for varying neighborhood sizes. Smaller neighborhood size results in better quality.

3.1 Lessons Learned

Vectorize code. Vectorized code tends to run faster than non-vectorized code due to parallelism and efficient algorithms for large matrix multiplication. We learned that a good approach is to start without vectorized code and remove loops by condensing data structures into matrices or tensors. Tensor arithmetic is also noticeably faster than non-vectorized code.

Use camera calibration files. Many datasets we found with provided their own camera calibration numbers. Working with these matrices is not entirely straightforward at times. Furthermore, point matching for computing the homography can be quite time-consuming. We need to manually select good corresponding points to use for the homography if feature detectors do not select good homography points. Automated approaches exist, but the perspective difference between the two camera is large enough that many of these don't work.

Account for drift. Mean-shift tracking point tends to drift in the primary camera. This can drastically affect the projection of the points into another camera frame. A small change in position in the primary camera might lead to a large movement in another camera. This drastically affects the quality of the result: if we use a constant size search area, then a large jump may cause the target to be outside of the search region. One solution is varying the search area size as a function of the Euclidean distance between the last two positions of mean-shift tracking, i.e., use a larger search area if the tracked point jumped far and vice-versa.

4 Future Work

Narrow search space with an object detector. Instead of creating the search space from the projected points, we can reduce our search space even further by running a object detector, such as Faster R-CNN [9], YOLO [8], or Single-Shot Detector [4], on each non-primary camera. These models draw bounding boxes on each object in a frame, depending on the frame rate. Then we can check the box of the k-closest objects to the transformed points.

Construct a visual hull. Knowing the locations of cameras in 3D space allows us to use an algorithm, such as VisualHull [3], to construct a complete 3D wireframe of the target. This provides additional information, such as target height, that may be used to help narrow searching or constructing a description of the target.

Dynamic camera switching. One useful feature for an application of multi-camera tracking is dynamically switching between cameras that have lost and captured the target. For example, if a target is no longer visible in one camera, remove the camera feed from the application dashboard. If it is picked up in another camera, add the camera feed to the dashboard. This allows the user to keep track of all cameras on the target without having to search through many different camera feeds.

References

[1] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on pattern analysis and machine intelligence*, 2003.

[2] W. Förstner and B. Moonen. A metric for covariance matrices. pages 299–309, 2003.

[3] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on pattern analysis and machine intelligence*, pages 150–162, 1994.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. 2016.

[5] New cameras enhance cabs security. `https://ttm.osu.edu/cabscameras`.

[6] F. Porikli, O. Tuzel, and P. Meer. Covariance tracking using model update based on lie algebra. *CVPR*, 2006.

[7] Privacy fears grow as cities increase surveillance. `http://www.nytimes.com`.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. 2016.

[9] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. 2015.