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 even on buses. Millions of dollars go to law enforcement and cities to help stop crime [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 in a bus might overlap with that of the camera on a building that the bus is stopped at. 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 to other cameras. We use a novel multi-scale variant of covariance tracking to match the target in the non-primary camera. 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

procedure BIGBROTHER($\mathcal{C}_1, \ldots, \mathcal{C}_N, T$)
    $\Sigma \leftarrow$ `feature_cov`(T) ▷ Covariance matrix for other cameras
    $B_i \leftarrow \emptyset \quad \forall i \in \{2, \ldots, N\}$ ▷ Position of the target in other cameras
    while true **do**
        $T \leftarrow$ `meanshift`($\mathcal{C}_1, T$) ▷ Update the target's position
        for $i \leftarrow 1$ to N **do**
            $T' \leftarrow H(\mathcal{C}_1, \mathcal{C}_i) \cdot T$ ▷ Transform coordinates using homography
            if $T' \notin \mathcal{C}_i$ **then**
                $B_i \leftarrow \emptyset$
                continue
            end if
            $S \leftarrow$ `search_region`(T') ▷ Search region around mean-shift estimate
            $B_i \leftarrow$ `multiscale_cov_track`(Σ, S)
        end for
    end while
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. We can only compute the homography if the field-of-view of the cameras overlap.

We compute a homography offline by selecting corresponding pixels between the two camera views. We may use a corner or feature detector to make pixel selection easier. We require at least 4 corresponding coordinates to compute the homography, however, for a better homography we used 14 points.

Each point is converted into inhomogeneous coordinates, and we solve the system of linear equations to produce the homography matrix H between two camera views.

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 $\sqrt{2}$. We could have additionally improved this algorithm using Random Sample Consensus (RANSAC) [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. We only track the centroid of the target. We use 16 color bins for the color quantization. We use circular neighbors with a radius of $r = 7$ and $h = 25$ for the Epanechnikov profile.

At each frame of the mean-shift tracking, we use the homography to transform the mean-shift-tracked centroid into the other camera(s). If the homography is accuracy, we can determine the target is in a particular camera frame or not. If the transformed points are not in the frame, i.e., outside of bounds of the camera frame. However, if the points lie inside of the camera frame, we know that the target is present in a particular camera frame.

2.3 Covariance Tracking

To find the target in the other cameras, we use a novel multi-scale covariance tracking [6]. The 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 spaceand improves speed. We found a $XX\%$ reduction, on average, 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 a different scales. 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. This sliding window size change is also why we used the radial component of the feature vector. 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 [3]. However, doing this for K filters and flipped dimensions, we will have $2 \cdot K$ bounding boxes around the location of lowest distance. We simply use non-maximal suppression 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 runtime 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 the largest limitation since many cameras, particularly indoor ones, do not have a high enough vantage point to satisfy the co-planar points, 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 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 aspect ratios. However, if our target does not match one of these aspect ratios, then we may miss the detection. However, since they're computed as a function of the original target dimensions, we are likely to detect the target in the other cameras.

3 Results

For our dataset, 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 the dataset is shown in Figure 2. 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, specifically 2017b.

Figure 1 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 patch. This is where the bulk of the time is spent. The multi-scale covariance tracking

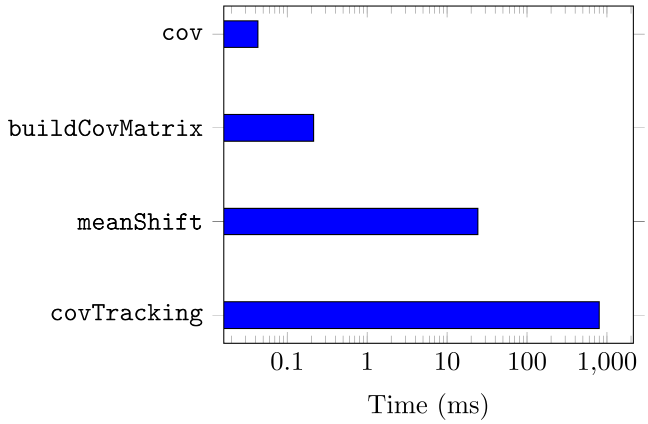

Figure 1: Performance of our components. Our multi-scale covariance tracking takes an order of magnitude longer than mean-shift tracking, as expected.

Figure 2: Example frame from our data. Our accuracy experiments track the moving white van.

clearly takes the longest amount of time. Mean-shift tracking, even with 50 iterations, takes an order of magnitude less time than covariance tracking. Its performance scales linearly if we reduce the number of scaling factors.

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 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 worked provided their own camera calibration matrices. Working with these matrices is not entirely straightforward at times. Furthermore, homographies can be quite time-consuming to compute. Automated approaches exist, but we did not investigate these.

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 object closest 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 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.

Triangulate new cameras. Cameras on moving objects, such as buses, are difficult to incorporate in this system. However, if we can identify the position and pose of a camera using VisualHull, we can compute an approximation of the moving camera in the 3D space of the fixed cameras. This immediately adds the moving camera to the network of fixed cameras. The moving camera is removed from the network when it moves outside of the 3D space of the fixed cameras.

References

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

[2] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981.

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

[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.