**Single Camera Pedestrian Tracking**

Alex Fandrianto, Aman Neelappa, Rohan Maheshwari

## Abstract

Pedestrian tracking is an interesting problem with useful applications in fields like surveillance and pedestrian control at intersections. We implemented an algorithm for multiple pedestrian tracking using a single static camera. We then applied our algorithm on a single moving camera by adding a preprocessing step to compensate for camera motion. Our camera motion compensation achieved an average L-2 error of 0.49 pixels on a real dataset and 0.27 pixels on a simulated dataset. Our pedestrian tracking obtained a precision of 0.81 and recall of 0.86. When combined the tracker obtained a precision of 0.67 and recall of 0.84.

## Introduction

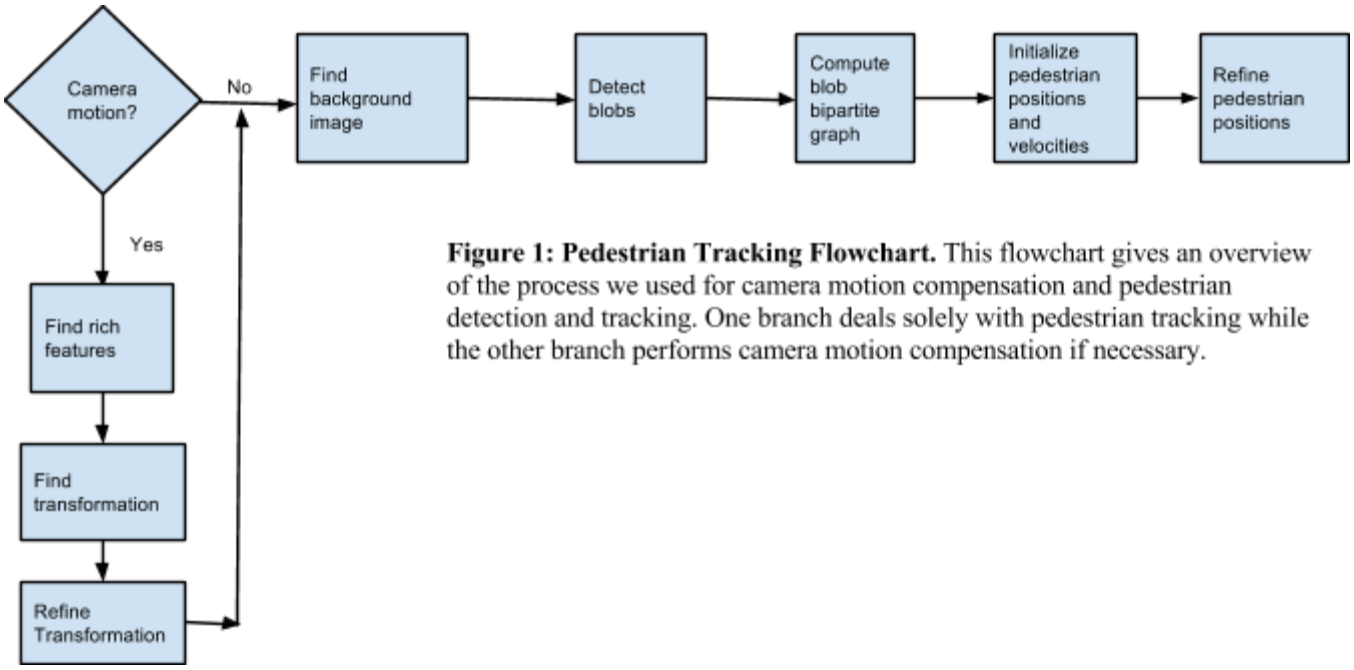



**Figure 1: Pedestrian Tracking Flowchart.** This flowchart gives an overview of the process we used for camera motion compensation and pedestrian detection and tracking. One branch deals solely with pedestrian tracking while the other branch performs camera motion compensation if necessary.

Given a stream of images taken from a single camera, the objective is to identify objects of interest and their motion across frames. Traditional techniques track objects based on the changes between frames, but this strategy is complicated when the camera is not static. We aim to study the performance of the former algorithm when combined with an algorithm to compensate for the ego-motion of the camera.

Many applications exist for pedestrian tracking. Traffic control, mobile robot navigation, and

surveillance all can make good use of a robust system. However, camera conditions are not always ideal. Environmental factors like wind can cause camera instability, while illumination varies throughout the day. Additionally, cameras may not be stable: for example, smartphones and robotic sensors. Since tracking algorithms generally require stable images, whenever possible, it would be useful to stabilize the frames by compensating for small perturbations in each frame.

## Literature Review

The problem of motion estimation is common in computer vision. 3D camera movement leads to complex changes in 2D images. The ability to follow the flow of pixels from one frame to another allows has applications in video compression and motion compensation. Two main methods exist to solve this problem. Pixel-based methods consider the pixel by pixel differences between image frames. One method, optical flow [6] attempts to model pixel movement based on gradients in the images. Feature-based methods attempt to find correspondences between higher level locations (such as minEigen features or Harris corners) in the images. These correspondences can be used to build a transformation between the two images; transformations can vary in complexity, from simple linear models to more complex polynomial ones. Common techniques used for motion estimation are linear regression and RANSAC [7]. The latter is more robust to outliers, but the former can be refined to filter out such outliers and reestimate the transformation, as in [2].

An alternative approach to pedestrian detection and tracking involves the use of class specific object detectors. Breitenstein et al [8] propose an algorithm that uses a first-order Markov model, considering only information from the current and the last time step, and integrates both class-specific and target-specific information in the observation model. A separate particle filter/tracker is initialized for each person detected with high confidence.

There are some key differences between the approaches that use object detectors for pedestrian detection and the one we use (pedestrian detection using background subtraction). The former is more robust to noise and illumination changes. However, it requires training specific detectors for different objects of interest. The latter approach can be used to detect multiple objects, though it is not as robust.

## Technical details

### Algorithm 1 (Person tracking using blobs) [1.]

The paper assumes a single fixed camera mounted at an arbitrary position. The images are processed at three levels: at the level of pixels, blobs and pedestrians. Below we outline the major steps in the process.

### Preprocessing the images

The first step in the pipeline is to find the *difference image* for each frame, the image that remains after subtracting the background. The background is established by using a few frames that have no objects of interest. During this phase, the maximum fluctuations in pixel values are noted, and the threshold is set to a value slightly higher than that. When objects of interest are present in a frame, this process gives rise to blobs.

**Blob tracking**

Individual blobs in each image are connected regions that can be extracted using boundary following. To track the movement of blobs, they are allowed to split, merge, appear and vanish across frames. Further, for each blob the following metrics are calculated *(1) area*: number of pixels, (2) *bounding box*: smallest rectangle holding the box, (3) *density*: area/bounding box area, and (4) *velocity*: pixels per second in horizontal and vertical directions

To model the movement of blobs, blobs from frame i are compared to the blobs from frame (i-1). The goal is to create the optimal bipartite graph with edges connecting the blobs in previous frame to the blobs in the current frame.

For computational simplicity, the following constraints are added to the graph:

**parent structure constraint**: A blob may either merge or split but not both. This forces the graph to be very disjoint; each connected component cannot have more than a single vertex of degree more than one.

**locality constraint**: vertices can be connected only if their corresponding blobs have a bounding box overlap area, which is at least half the size of the bounding box of the smaller blob.

Subject to these constraints, the authors find the bipartite graph the minimizes the following cost function:

$$\text{Cost}(G_i) = \Sigma[P_i \text{ in } G_i] \, |\text{Area}(P_i) - S_i(P_i)| \, / \, \max(\text{Area}(P_i), S_i(P_i))$$

where $S_i(P_i) = \Sigma[\text{Neighbor } N_i \text{ of } P_i] \, \text{Area}(N_i)$

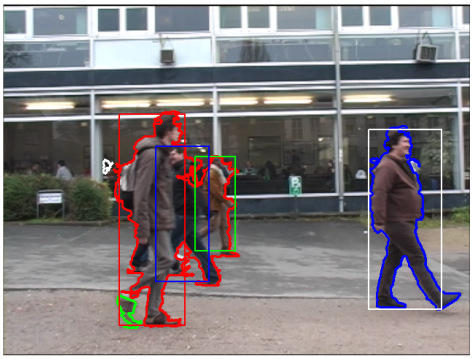
Here $P_i$ are the parent blobs in the graph $G_i$ where parent blobs are defined as those which have any of the following properties:

(1) degree > 1 (blob was split)

(2) degree 0 (blob vanishes)

(3) degree 1 but only appears in $i^{th}$ frame (blob appears)

The authors outline a simple algorithm to find the optimal graph. At the end of this stage, the velocity of each blob in the image reference frame is used to initialize the next stage of pedestrian tracking. Figure 2 shows an example of this algorithm on the TUD Campus dataset.



| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |

**Figure 2: Pedestrian Tracking via Blobs.** The 4 blobs detected in the left image are associated with the 3 in the right via the matrix to the left.2 blobs disappear (left white and green), [empty rows]. 1 blob appears (right red). [empty columns]Left red maps to right white and left blue with right green.We overlaid the bounding boxes of the ground truth pedestrians.

**Pedestrian tracking**

Pedestrians are modeled as rectangular patches with certain dynamic behavior. The dynamics of the pedestrians are modeled by the following equation:

$$x(t+1) = F\, x(t) + v(t)$$

x is a state vector [x1 y1 v1 v2] that models the position and velocity in the image plane.

F is the transition matrix of the system

v(t) is a sequence of zero mean, white Gaussian process noise with a certain covariance matrix to model the variance of acceleration.

Given the above dynamic model and the blob tracking results, pedestrian tracking is done in the following steps:

1. ***Relating pedestrians to blobs*:** Pedestrians are related to blobs using a simple rule: if a pedestrian was related to a blob in frame and that blob is related to another blob in the frame (through a split, merge, etc.), then the pedestrian is also related to the latter blob.
2. ***Initialize pedestrian positions**:* In this step we initialize the pedestrians in new blobs that appear in any image.
3. ***Predict pedestrian positions**:* We predict pedestrian positions in frames after they were initialized using the velocity estimates which are updated after each iteration.
4. ***Refine pedestrian positions:*** We refine the pedestrian positions by employing a 2D search to find the best position for overlap between the corresponding pedestrians and blobs.

**Algorithm 2 (Ego-Motion Compensation) [2.]**

Camera motion complicates background models by causing image-wide distortion between frames. Successful methods for dealing with this ego-motion generally attempt to construct a camera transformation between successive frames [3.], sometimes by assuming an affine or projective transformation. Then with this adjustment, the previous techniques can be used for object tracking.

Jung and Sukhatme's compensation algorithm involves using salient image features (e.g., corners and areas with rich textures) [4.] to compute the transform between the image at time t-1 to the image at time t. The feature correspondence between successive image frames is determined by the Lucas-Kanade method [5.], a method used for determining optical flow. In this paper, however, we use minEigen features to determine this correspondence.

With the correspondence of many features, it is possible to estimate the parameters of the camera transformation T. In this paper, Jung and Sukhatme select a bilinear model, instead of an affine or projective transformation. This nonlinear model, converting the image point $f^{t-1}$ to $f^{t}$, is reproduced below:

$$f_x^t = a_0 f_x^{t-1} + a_1 f_y^{t-1} + a_2 + a_3 f_x^{t-1} f_y^{t-1}$$
$$f_y^t = a_4 f_x^{t-1} + a_5 f_y^{t-1} + a_6 + a_7 f_x^{t-1} f_y^{t-1}$$

The $a_i$ parameters are estimated using least squares optimization. This provides an initial transform $T_0$, which estimates the true camera transform. The estimate suffers when features belonging to moving objects are corresponded. In order to base the transformation only on background objects, the algorithm filters out features with high error. The remaining correspondences $F_{in}$ are used to recompute a new estimate for the bilinear transform T.

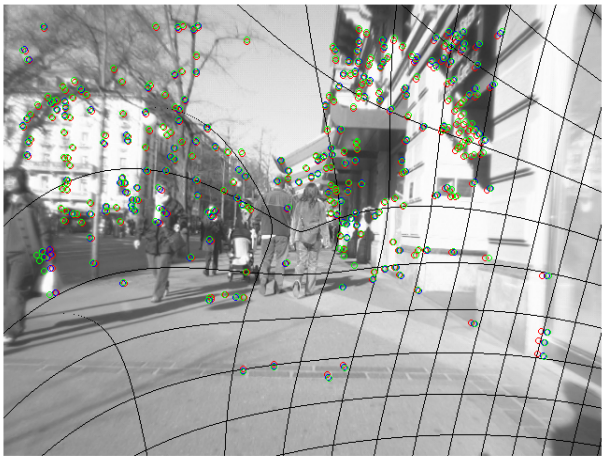
The final bilinear transform T (treated as matrix below) is used to transform points in the image at time t-1 to the image at time t. This compensated image can be used to compute image differences. Since in the general case, the compensated image will not overlap completely with the image at time t, the borders of the image frame cannot be used to detect or track moving objects.

In this paper, we select a quadratic model over the bilinear model because of the resultant reduction in mean L2 error. Mean L2 error indicates the average distance the transformed feature points are from their corresponding feature points in the next frame. The lower the error, the better the transformation. It is worth being cautious of overfitting, but due to the large number of feature correspondences, the fits seem to be quite good. When refining the transformation, we use the correspondences in the bottom 85 percentile of L2 error. In practice, this threshold causes matching features located on moving

pedestrians to become outliers.

a)

b)



c)

**Figure 3: Camera Motion Compensation**. The previous frame's feature points are in red (a), with the next frame's in green (b). The quadratic transformation (c) shows the compensated red features as blue. Most of the blue points are overlapped by the green features, indicating the transformation is quite accurate. Black lines help illustrate how the image is stretched.

## Experiments

### Datasets

The TUD Campus [8] and Stadtmitte [9] datasets contain a stationary camera video of pedestrians crossing a street. Approximately 7 people are present per image. They walk in front of the camera and occasionally past each other. This is ideal for our pedestrian tracking algorithm.

The ETH Zurich Sunny Day dataset is the video taken by a mobile robot traveling along a sidewalk. Pedestrians tend to move towards or away from the robot in this dataset, making it less ideal for our current pedestrian tracking scheme. Approximately 5 pedestrians are present per frame.

### Pedestrian Tracking (Stationary camera)

We compute the precision and recall metrics (we consider a match if the ground truth pedestrian bounding box and the one computed by us is above a threshold). We report a precision of 0.81 and a recall of 0.86.

**Figure 4: Pedestrian Tracking**. Sample images from our dataset with tracked pedestrians in red and ground truth in green

**Camera Motion Compensation**

We use the ETH Zurich Sunny Day dataset and compensate motion across frames. As described, we compute a transformation between frames using corresponding feature points and then refine the transformation. The original correspondences had a mean L2 error of 2.22 pixels, while the final mean L2 error after motion compensation is 0.49 pixels. See Figure 3 for an example transformation.

**Camera Motion Compensation Simulation**

The ETH Zurich dataset involves too much camera motion for the Pedestrian Tracking algorithm, which requires a background to be computed. We simulate random camera motion in the form of random affine transformations (scaling, skewing, and translation) on the TUD Stadtmitte dataset in order to test the camera motion algorithm.

a)  b) 

c) 

**Figure 5: Camera Motion Compensation Simulation**. The original color image (a) had a random transformation applied to it to obtain (b). Except for pixels lost during the transformation, the motion compensation algorithm (c) restores the original image with an average L2 error of 0.27 pixels.

Random transformations are applied to each of the frames of the dataset. Then camera motion compensation adjusts the images back to their original state so that they can be used for pedestrian tracking, as in Figure 5.

a)  b) 

**Figure 6: Simulated Background Image**. A new background image (b) was estimated for the TUD Stadtmitte dataset after the simulation. For comparison, the old background (a) is provided. The backgrounds are computed by taking the mode of the pixels over the dataset in order to remove the transient artifacts of pedestrians in the scene. The simulation does not degrade the background much, and most dark spots are near the edges of the frame.

**Pedestrian Tracking (Moving camera simulation)**

We compute the precision and recall metrics (we consider a match if the ground truth pedestrian bounding box and the one computed by us is above a threshold of 0.5). We report a precision of 0.67 and a recall of 0.84. The simulation's background image (Figure 6) could have caused the decrease in performance. Pedestrians are harder to detect on the sides and center of the image due to the additional blackened regions of the image.

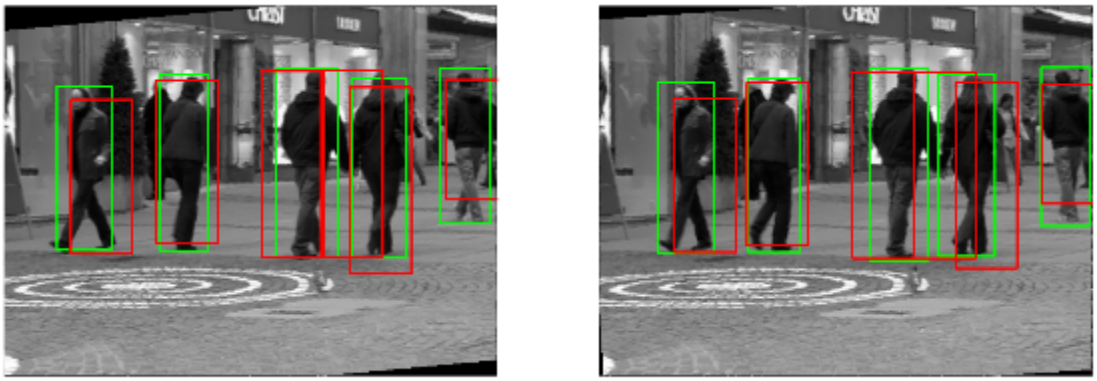


**Figure 7: Pedestrian Tracking with compensation for simulated camera motion**. Sample images from our new dataset with tracked pedestrians after compensating for the simulated camera motion in red and ground truth in green

## Conclusion

In general, we find that blob tracking provides decent results. For pedestrian tracking recall and precision were 0.86 and 0.81 respectively, but our implementation degrades in accuracy as we track more frames. One reason for this error is that we are tracking in 2D instead of 3D, which makes our velocity estimates less accurate. Occluded pedestrians present a difficult challenge as it is not clear how long they should be tracked based on last velocity before being dropped. For our specific dataset, the ground truth simply dropped them as soon as they were occluded, which artificially hurts our precision score.

The current pedestrian tracking algorithm is general. It can track not only pedestrians, but also cars and other moving objects. However, the use of blobs means the tracker can get confused between different object classes. Worse, targets that do not move cannot be detected under this system, and a background image is required to compute the blob locations. Use of pedestrian detectors could increase accuracy of the pedestrian tracking system.

Camera motion compensation is a very promising preprocessing step for the current algorithm. High accuracy transformations (with less than 0.5 pixel average error) can be achieved in real world and simulated datasets without hampering the performance of the pedestrian tracker (only a small drop in performance). Unfortunately, at this time, the algorithm's reliance on a background image forces camera motions to be localized to a common area in order to maintain good performance.

**References**


1. Masoud Osama, and Nikolaos P. Papanikolopoulos. "A novel method for tracking and counting pedestrians in real-time using a single camera." *Vehicular Technology, IEEE Transactions on* 50.5 (2001): 1267-1278.
2. Jung, Boyoon, and Gaurav S. Sukhatme. "Detecting moving objects using a single camera on a mobile robot in an outdoor environment." *International Conference on Intelligent Autonomous Systems*. 2004.
3. Yilmaz, Alper, Omar Javed, and Mubarak Shah. "Object tracking: A survey."*Acm Computing Surveys (CSUR)* 38.4 (2006): 13.
4. Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Pittsburgh, PA, April 1991.
5. Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In Proceedings of the 7th International Joint Conference on Artificial Intelligence, pages 674697, 1981.
6. Martin A. Fischler and Robert C. Bolles (June 1981). "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". *Comm. of the ACM* **24** (6): 381–395. doi:10.1145/358669.358692
7. David J. Fleet and Yair Weiss (2006). "Optical Flow Estimation". In Paragios et al.*Handbook of Mathematical Models in Computer Vision*. Springer. ISBN 0-387-26371-3
8. Breitenstein, Michael D., et al. "Online multiperson tracking-by-detection from a single, uncalibrated camera." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.9 (2011): 1820-1833.
9. TUD Campus and Crossing <https://www.d2.mpi-inf.mpg.de/node/382>
10. TUD Stadtmitte < http://www.d2.mpi-inf.mpg.de/node/428 >
11. ETH Zurich Sunny Day <http://www.vision.ee.ethz.ch/~aess/dataset/>