Spatio-temporal Object Tracking from Optical Flow

Andrew Pillsbury

Dartmouth College Computer Science Honors Thesis
June 4, 2015

Abstract

In this paper I introduce an algorithm that tracks any number of moving objects through a video. The algorithm is fed a video from a stationary camera and returns a set of objects which are represented as three-dimensional binary arrays consisting of a two-dimensional blob at each time interval that corresponds to a frame in the video. This algorithm does not make any assumptions about the objects it tracks, and can therefore track any type of object moving any direction. It is also lightweight and can run in near real time once the optical flows have been calculated. Unlike other object tracking algorithms, this algorithm does not depend on getting accurate information from any individual frame. It looks both forward and backwards, using the entire video to identify the most likely objects.

1. Introduction

Tracking and finding objects is an important task in computer vision and has applications in object recognition, surveillance, traffic monitoring, and video indexing [1]. This task consists of two main steps: object detection and object tracking from frame to frame. The algorithm that I propose uses optical flow between frames to accomplish both of these tasks.

My method reads in a video and returns a set of objects that it finds in the video. These objects correspond to objects moving in the video, where movement is defined as translation through the x and y coordinates of the video. The objects are represented as three-dimensional binary arrays consisting of an x coordinate, a y coordinate, and a time coordinate. Each point is equal to 1 if it is part of the object and is 0 otherwise. The objects are spatially and temporally continuous, meaning that they move smoothly through time and space from the time they first appear to the time they disappear. Although they are represented in discrete increments, they can be interpolated to arbitrarily small spatial and temporal increments.

My algorithm begins by finding objects using relatively large time intervals, and then at the end interpolates these objects to the same frame rate as the original video. The algorithm consists of five parts: finding blobs that are candidates for being objects in each time increment, assigning these blobs to existing objects or making new objects with them, combining multiple objects into one, interpolating

missing frames in objects, and finally interpolating the objects into smaller time intervals so they contain one two-dimensional matrix per frame. In order to allow others to use and build off of my work, I provide a public MATLAB implementation of my algorithm [2].

The frames that my algorithm analyzes are the ones corresponding to the times in the set defined in Equation 1, where the time increment is Δt, the video begins at t_{min}, and the video ends at t_{max}. I will call the video that has only these time increments the video skeleton, and the objects that use only these time increments object skeletons.

$$\{n * \Delta t | (n \in \mathbb{Z}) \wedge (n * \Delta t \geq t_{min}) \wedge (n * \Delta t \leq t_{max})\}$$

Equation 1: Set of times corresponding to frames in the video skeleton

My approach is founded on the assumption that no single frame contains enough information to accurately identify and track objects, and thus several frames must be used. Using the optical flow, I identify all possible objects by finding areas of movement that are the same in the forward flow and the backward flow. Because I do not depend on information from individual frames, I do not need to examine the optical flow from every frame, but instead can do my calculations at intervals (usually one second apart) and interpolate the results between these intervals. In Figure 1 we see that although this approach yields less detail, it still finds the major objects in the image, and uses a fraction of the computation time. The image on the left is the current frame, the image in the middle is the horizontal flow between the current frame and the frame one tenth of a second earlier, and the figure on the right is the horizontal flow between the current frame and the frame one second earlier. In both slides one can clearly see the man (the blue shape), and although there is more noise when the time interval is increased, this noise is eliminated by the rest of my algorithm.

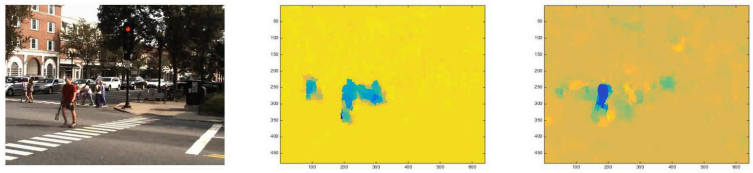

Figure 1: Optical flow between frames 1/10 second and 1 second apart

The problem of object tracking has been tackled in many different ways by others in the field of computer vision. I will refer the reader to [1] for an overview of many of these methods, but will briefly discuss some of the algorithms most similar to my own.

Sato et al. [3] describe a temporal spatio-velocity transform that finds pixel velocities in binary image sequences. This algorithm extracts blobs based on similarity of velocity and position, and is applied to tracking humans on a sidewalk. The algorithm first performs background subtraction and creates a one-dimensional binary image sequence, which then is thresholded to make human blobs. The trajectories are then calculated by examining the blobs over time, and then human interactions are discerned from the trajectories of the human blobs.

Aggarwal et al. [4] present a method for tracking moving objects in compressed MPEG videos. This algorithm uses background subtraction to locate objects, and requires that a user identify the object of interest. It then interpolates the results into predicted frames to obtain a smooth tracking.

Han et al. [5] track multiple objects through a number of frames, even under partial occlusion. They detect moving objects, then use a new weighted Kanade-Lucas-Tomasi tracker to track features within the moving object mask, and finally find the trajectory of the object. Using this method, they had a high degree of accuracy in identifying moving objects' trajectories.

2. Find Candidate Blobs

As the first step of my algorithm, I find all blobs in each frame of the video skeleton that could potentially be part of objects. There are two steps involved in this operation, which are detailed below. Note that I will refer to the video as V, and a frame in the video at time t as $V(t)$. Similarly, for an object O, I will refer to the two-dimensional spatial representation of the object at time t as $O(t)$.

2.1. Get binary mask of high flow areas

The first step in finding candidate blobs is to find high flow areas in the image. To do this, I use the algorithm designed and implemented by Sun et al. [6] to find the optical flow from $V(t)$ to $V(t+2*\Delta t)$ and the negative of the optical flow from $V(t)$ to $V(t-2*\Delta t)$. For each of these, I first add the absolute values of the horizontal and vertical flows to find areas of high flow. I then threshold these values by iteratively finding the median of all non-zero entries and setting those below the median to zero. The higher the number of iterations is, the higher the effective threshold is. Through experimentation, I have found that using two iterations yields good results.

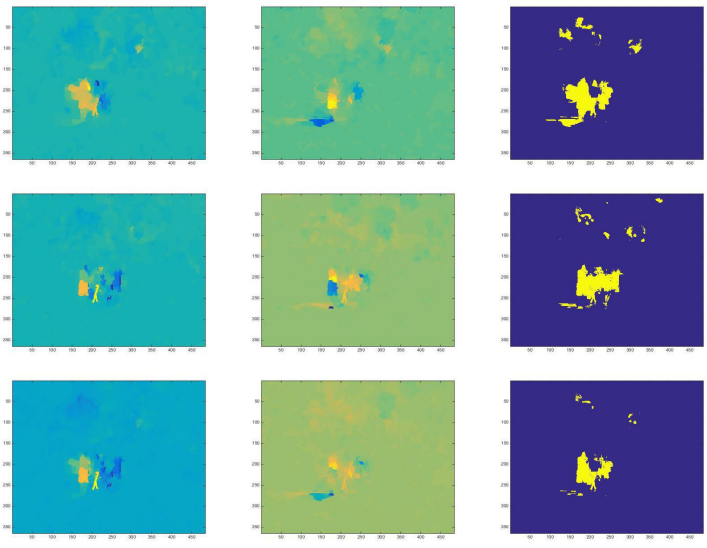

Figure 2: Forward and backward optical flow

After thresholding, the matrix is transformed into a binary image where all non-zero values are set equal to one.

Once a mask has been created for both the forward flow and the backward flow, I create a final mask that is the intersection of those masks, and create a final horizontal flow matrix and vertical flow matrix that are the sum of the forward flow and the

reverse of the backward flow. In Figure 2, the left column is horizontal optical flow, the middle column is vertical optical flow, and the right column is the mask created from the optical flow. The top row is the negative of the backward flow, the middle row is forward flow, and the bottom row is a combination of the first two using the sum of the flows and the intersection of the masks.

2.2. Split the mask into individual blobs

Once a binary mask has been found for a frame it is split up into individual masks based on connectivity so that each group of connected pixels (blob) is a separate mask (Figure 3, left). For each of these masks, if it is below a size threshold it is discarded. Otherwise, it is first expanded so that nothing is missed (Figure 3, middle). Then, the direction of motion of the blob is found. This is done by creating a matrix of directional values using the final horizontal and vertical flow matrices, and then finding the median of these values that are in the unexpanded blob. Finally, the expanded blob is filtered so that all entries that are not within a tolerance of the median direction of motion are eliminated (Figure 3, right). If the blob is too scattered (as determined by the mean standard deviation of the rows and columns of the filtered blob being below a threshold) the blob is determined to be noise and is eliminated.

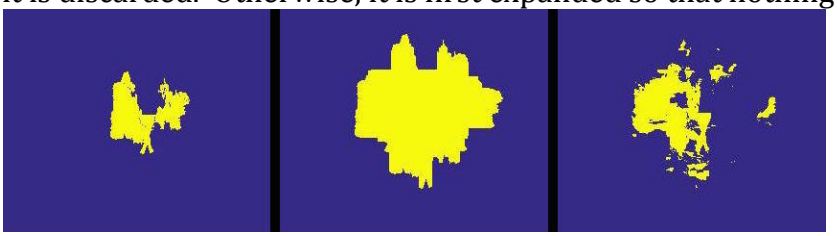

Figure 3: Split, expanded, and filtered blob

3. Assign Blobs to Objects

Once a blob *A* has been found, expanded, and filtered, it is assigned to an object. The likelihood that *A* is a part of an object *O* is calculated by finding the maximum possible intersection between *A* and *B*, *O*'s most recently found blob. This is calculated by translating *B* in a specified direction and recording the maximum intersection between *A* and the translated *B*. Figure 4 illustrates this process: the white arrow shows the direction of motion, the red area is *A*, the blue area is *B*, and the purple area is the intersection of *A* and *B*. The image on the left shows the overlap before sliding *B* and the image on the right shows the overlap after sliding *B*. Because the direction of motion is sometimes inaccurate, I

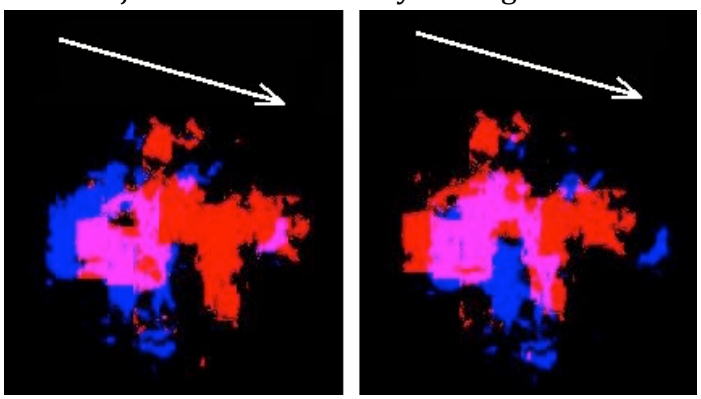

Figure 4: Finding overlap between objects

run this calculation with the directions of motion of A, B, and the last three blobs in O previous to B in order to mitigate any error. The likelihood that A is part of O is then calculated by dividing the maximum possible intersection (using any of the possible directions) by the size of the larger of A and B, as shown in Equation 2.

$$\frac{overlap(A,B)}{\max\left(size(A), size(B)\right)}$$

Equation 2: Likelihood that a blob is part of an object

If the largest likelihood across all objects is above a threshold, A is added to that object and the direction of motion of A is saved as whatever direction yielded the maximum overlap. If no likelihood is above the threshold, a new object is created and A is added to it.

4. Combine Objects

Once all blobs have been assigned to objects, I determine whether any two objects should be combined together into one object. To do this, I check every object O_1 against every object O_2 that was created after it. Let O_2 come into existence at time t, meaning that the first frame in O_2 which contains a blob is $O_2(t)$. Because of the order in which objects are created, this means that O_1 was created at a time no later than t. To determine the likelihood that O_1 and O_2 should be combined into a single object, a similar process is used as was employed to determine the likelihood that a specific blob belongs to an object.

Let A be $O_2(t)$ (the first blob that is associated with O_2) and B be the last blob associated with O_1 before time t. Then, in a manner similar to the one discussed above, the maximum possible overlap is calculated between A and B using the last three directions of motion from O_1 before t (including that of B) and the first three directions of motion from O_2 (including that of A). Instead of only using the overlap to determine likelihood, however, the difference between the median direction used from O_1 and the median direction used from O_2 is also taken into account using Equation 3, where *directionDiff* is the difference between the directions and *maxDiff* is the maximum possible difference between directions.

$$\frac{overlap(A,B)}{\max\left(size(A), size(B)\right)} * \left(1 - \frac{directionDiff}{maxDiff}\right)^2$$

Equation 3: Likelihood that two objects should be combined

This value is calculated for every O_2 that was created after O_1, and the object O_3 yielding the maximum likelihood is recorded. If this likelihood is above the threshold (the same threshold that was used for assigning blobs to objects) then O_1 is merged with O_3, meaning that for every t where $O_3(t)$ contains a blob, $O_1(t)$ is overwritten by $O_3(t)$. After this happens, O_3 is deleted.

5. Interpolate Missing Blobs and Delete Non-persistent Objects

In order to weed out noise, any object that has fewer than a minimum number of blobs associated with it is deleted so that only objects that persist through a significant amount of time are left.

After that, every object O is examined in order to interpolate missing blobs. A blob at time t is defined as missing from object O if t is a time corresponding to part of the video skeleton and there is both a blob at $O(t_{first})$, where $t_{first} < t$, and at $O(t_{last})$, where $t_{last} > t$. Every missing blob is interpolated by finding the offset between $O(t_{first})$ and $O(t_{last})$ and translating $O(t_{first})$ by the value calculated in Equation 4.

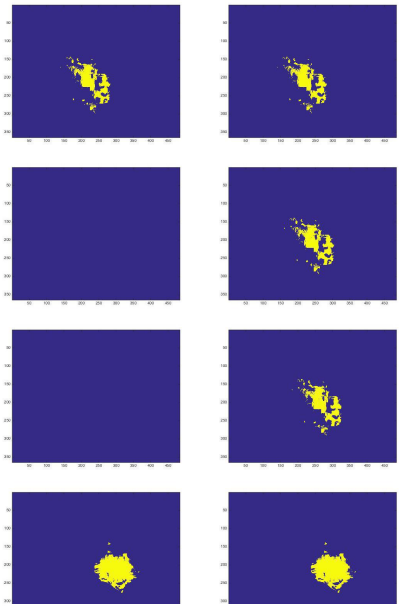

Figure 5: Interpolating blobs

$$\frac{offset * (t - t_{first})}{(t_{last} - t_{first})}$$

Equation 4: Calculating offset for interpolated blobs

In Equation 4, *offset* is found by using the previously discussed function to find the maximum overlap between $O(t_{first})$ and $O(t_{last})$ and recording the x and y offset values that yielded the maximum overlap. Figure 5 shows four frames of an object before and after interpolation, where the top row is at t_{first}, the bottom row is at t_{last}, the left column is the object before interpolation, and the right column is the object after interpolation.

Because I combine objects and interpolate missing blobs, short-term occlusions do not pose a problem to tracking objects, because the object gets combined from before and after the occlusion and the occluded frames get interpolated.

6. Create Final Objects

In order to transform the object skeleton into an object that has a blob corresponding to every frame of the original video, I interpolate the skeleton into smaller time intervals.

Let the current object be O and the current time be t, and the time of the new blob that needs to be created be t' where $t < t' < t+\Delta t$. $O(t')$ is interpolated in the same manner as missing blobs, except the shape used for the interpolated blobs is a combination of the current blob, the next blob, and previous blob, i.e. $O(t)$, $O(t+\Delta t)$, and $O(t-\Delta t)$. This shape is found by translating $O(t-\Delta t)$ and $O(t+\Delta t)$ to maximize their overlap with $O(t)$, and then defining the shape of $O(t')$ as the space where at least two of the three blobs intersect. In Figure 6, $O(t-\Delta t)$ is blue, $O(t+\Delta t)$ is green, and $O(t)$ is red. The other colors correspond to overlap between blobs (for example,

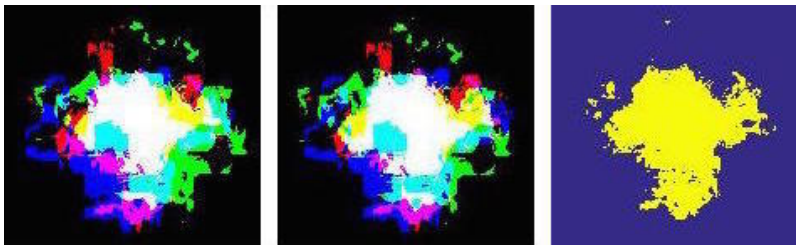

Figure 6: Using overlap from before and after for a more accurate shape

purple is the overlap of red and blue, and white is the overlap of all three). The leftmost image shows the blobs before being translated, the middle figure shows the blobs after being translated, and the rightmost image is the final blob resulting from the overlap. This final shape is then translated using the values from the same calculation as is described in Equation 4.

7. Results

Figures 7-9 show the results when this algorithm is tested on a short video clip of an intersection that includes pedestrians crossing the street at the beginning and a truck driving through at the end. These results were created by taking one frame per second from the video and masking that frame with the corresponding blob from the object. Figure 7 shows the first object, which tracks the pedestrians crossing the street. Figure 8 shows the second object, which also tracks the pedestrians briefly. Figure 9 shows the third object, which tracks the truck towards the end of the clip.

Figure 7: Object 1 found in a short video of an intersection

Figure 8: Object 2 found in a short video of an intersection

Figure 9: Object 3 found in a short video of an intersection

8. Conclusion and Future Work

In this paper, I have described an algorithm that uses video from a stationary camera to find and track moving objects through time. This algorithm is generalized, so it can track any type of object moving any direction. It is also lightweight and can run in near real time once the optical flows have been calculated. The advantage that my algorithm has over other algorithms is that it does not depend on getting accurate information from any one frame, or from only looking at previous frames to find objects. My algorithm looks both forward and backwards, using the entire video to identify the most likely objects.

There are many ways that this project could be extended in the future in order to make it more accurate, specialize it for different uses, or both. A priority for those looking to build upon this project should be to test the algorithm on a variety of situations. As of the writing of this paper I have only tested this code on videos of street intersections, but it should be tested on videos containing a variety of types of objects in many different environments.

A possible improvement to the algorithm is using smaller flow increments in addition to the larger ones to find faster moving objects and objects that don't move in a consistent direction, like a person's arms and legs.

The speed of this algorithm could also be improved by using large spatial increments as well as temporal increments when finding the object skeletons, and then interpolating the missing x and y values in the same way that the missing time frames are interpolated at the end.

Another way in which this project could be extended is to enable it to use a moving camera rather than just a stationary one. This task would require that optical flow due to the motion of the camera be eliminated so that only objects moving independently of the camera motion remain. This is difficult, however, because objects move in different directions due to the motion of a camera, so no one adjustment could be applied to the whole optical flow to eliminate this motion.

Another feature that could be added is to add image segmentation to create more precision. If more precision is needed than the blobs provide, an image segmentation algorithm could be applied which only segments the image inside the blobs, thus yielding the same precision of image segmentation but with much less computational cost because the whole image need not be segmented.

9. Acknowledgements

Thank you to my thesis advisors Richard Granger and M. Douglas McIlroy for providing guidance and advice throughout this project.

10. References

[1] Yilmaz, A., Javed, O., and Shah, M. 2006. Object tracking: A survey. *ACM Comput. Surv. 38*, 4, Article 13 (Dec. 2006), 45 pages. Print.

[2] github.com/APILLSBURY/tsp-thesis/tree/master/optical_flow_object_tracking

[3] Sato, Koichi, and J.k. Aggarwal. "Temporal Spatio-velocity Transform and Its Application to Tracking and Interaction." Computer Vision and Image Understanding (2004): 100-28. Print.

[4] Aggarwal, Ashwani, Susmit Biswas, Sandeep Singh, Shamik Sural, and A. K. Majumdar. "Object Tracking Using Background Subtraction and Motion Estimation

in MPEG Videos." Computer Vision – ACCV 2006 Lecture Notes in Computer Science: 121-30. Print.

[5] Han, Bing, Christopher Paulson, Taoran Lu, Dapeng Wu, and Jian Li. "Tracking of Multiple Objects under Partial Occlusion." Automatic Target Recognition XIX (2009). Print.

[6] Sun, D.; Roth, S. & Black, M. J. "Secrets of Optical Flow Estimation and Their Principles" IEEE Int. Conf. on Comp. Vision & Pattern Recognition, 2010