Video Contour Tracking

Honglin Zheng 304947026 hlzheng@g.ucla.edu
Zhao Weng 304946606 w2280518650@gmail.com
Yao Xie 804946717 allenxie@cs.ucla.edu
Wenshan Li 105026914 helenali19@g.ucla.edu

Abstract

Both object tracking in video and contour extraction in image are active research in computer vision. However, state-of-the-art algorithms in both areas have limitations to some extent. For object tracking algorithms, most of them only capture the position and the general size of the moving object, which are represented as rectangle bounding box. For contour extraction models, most of them are applied on a single image. In this project, we propose a method to perform video contour tracking that incorporates two models to identify both the moving objects and their fine-grained contours in a video effectively.

1 Introduction

Video tracking is the process of locating a moving object (or multiple moving objects) overtime in consecutive video frames. It has been applied to various uses, including human-computer interaction, augmented reality, video communication and compression[1]. Traditional video tracking algorithms analyze sequential video frames and output the movement of targets between frames. The model generally extracts only coordinates and size information for each moving object. In this project, we combine the traditional video tracking model with another active research topic, contour extraction, by identifying object boundaries from the detected tracking, which is represented by a moving rectangle bounding box. Due to the great amount of data that is contained in video, video tracking can be a difficult and time consuming process, especially when the speed of moving objects exceeds the frame rate. Our model specifically performs automatic detection and motion-based tracking of moving objects in a video from a stationary camera.

The project is decomposed into two main phases: 1.object detection and tracking; 2.contour extraction. The first phase is further decomposed into two major steps: target representation and localization, as well as filtering and data association. To be more specific, we first detect moving objects in each frame, then associate the detection correspond to the same object over time. The detection of moving objects is based on a background subtraction algorithm using Gaussian mixture models. We apply blob analysis to detect groups of connected pixels, which are likely to form moving objects. The second phase, contour extraction, is divided into two parts, background subtraction, and boundary extraction. Topological structural analysis of digitized binary images by border following is used to extract contours for objects in each frame of the video. Background subtraction and bounding boxes are utilized and compared to create better contour extraction.

We show that our proposed model can detect multiple moving objects in video frames, as well as extract corresponding contours for each object. The learned model can be utilized in many computer vision applications, including security and surveillance, activity recognition, and traffic monitoring.

2 Related Works

There are a variety of algorithms in both video tracking and contour extraction, each having strengths and weaknesses. When performing the task, these models target at distinct scenarios. The following subsections will briefly illustrate how they work and analyze their advantages and disadvantages respectively.

[1] https://en.wikipedia.org/wiki/Video_tracking

2.1 Algorithms in Video Tracking

The process of video tracking is generally decomposed into two main steps: object detection, and data association. The first step is mostly considered as a bottom-up process, while the second step is usually a top-down process.

Common target localization algorithms include Kernel-based tracking and Condensation algorithm. Kernel-based tracking[3] is also known as mean-shift tracking. It performs iterative detection based on the maximization of a similarity measure. Condensation algorithm (Conditional Density Propagation)[2] is another popular probabilistic computer vision algorithm, which presents an alternate approach by iteratively detecting the object contour moving in a cluttered environment. The algorithm evolves the contour by minimizing the contour energy using gradient descent. Specifically, it starts with an initial contour from the previous frame, and evolves to its new position in the current frame.

After target representation and localization, we perform filtering and data association by incorporating prior information about the moving objects. Kalman filter[5] is an optimal recursive Bayesian filter that uses a series of measurements observed over time, and produces estimates of unknown variables. It estimates a jointly probability distribution over the variables for each time frame, which tends to be more accurate than those based on a single measurement alone. Since Kalman filter is a Bayesian filter for linear functions that subjects to Gaussian noise, its performance is less desirable when dealing with nonlinear and non-Gaussian processes.[4] Another algorithm, Particle filter[1], is useful for sampling the underlying state-space distribution in these situations. It uses a generic mutation-selection sampling approach, with a set of samples to represent the posterior distribution of some stochastic process given partial observations.

2.2 Algorithms in Contour Extraction

Contour extraction, also known as contour tracing or boundary following, is a technique to extract boundaries from digital images.

3 Approach

Our project is splitted into two major phases: object tracking and contour extraction. This section illustrates the methods adopted and explains the detailed approaches.

3.1 Object Detection

For implementation of object detection, we utilize Matlab Computer Vision System Toolbox[2], a computer vision framework which is designed for machine learning tasks including vision, cognition and reconstruction. The implementation consists of two parts: detecting moving objects in each frame, and associating the detections with corresponding objects across several frames.

3.1.1 Target Representation and Localization

We apply a background subtraction algorithm to identify moving targets. The algorithm is based on Gaussian mixture models. Specifically, the algorithm examines a number of consecutive video frames, and returns the foreground mask. The mask consists of ones and zeroes, where 1's stand for the detected object, and 0's stand for the background. We then apply blob analysis[3] to detect groups of connected pixels that are likely to form moving objects. The detection results in a region of an image in which some properties are constant or approximately constant.

We define a data structure *track*, shown in listing 1, representing a moving object in the video, with information used for detection to track assignment, termination and display.

```
function tracks = initializeTracks()
    % create an empty array of tracks
    tracks = struct(...
        'id', {}, ...
        'bbox', {}, ...
        'kalmanFilter', {}, ...
        'age', {}, ...
        'totalVisibleCount', {}, ...
        'consecutiveInvisibleCount', {})
    ;
end
```

Listing 1: Track Structure

The purpose of track is to maintain the state of a tracked object. The detections only in short-lived tracks are considered as noises, which will be removed from detection results. We only keep an object detection when its number of frames exceeds a certain threshold. We assume an object has left the field of view when there are no detections associated with the corresponding track for several consecutive time frames, that is the number of empty frames exceeds a specified threshold,

[2] https://www.mathworks.com/help/vision/index.html
[3] https://www.mathworks.com/help/vision/ref/blobanalysis.html

specified as *consecutiveInvisibleCount* in the *track* structure. The output of the algorithm is a binary mask of the same size as the input frame, with pixels of value 1 as foreground, and pixels of value 0 as background. The corresponding bounding boxes are extracted for detected objects. Kalman filter is adopted to predict the current location of the track, and the bounding box is updated so that its center is at the predicted location.

3.1.2 Detection Association and Filtering

After obtaining the targets in each video frame, our next step is to assign object detections in the current frame to one of the existing tracks. We associate the detections by minimizing the cost function, which is defined as the negative log-likelihood of a detection corresponding to a track. The assignment is further divided into two steps. First, we compute the cost of assigning every detection to each track. The cost computation is based on the Euclidean distance between the predicted centroid and the detected centroid, multiplied by the confidence of the prediction maintained by the Kalman filter. Then we solve the assignment problem by optimizing the cost. For each track, we consider assigning one of the detections to the track, or not assigning any detections to the track. The value for the cost of not assigning a detection to a track is chosen experimentally by considering the range of distance computed in the previous step. We compute the assignment by using the *assignDetectionsToTracks* function which minimizes the total cost by using Hungarian algorithm[4].

We update each track with corresponding detections by correcting the estimation of the object's location using the new detection. The predicted bounding box is replaced with the detected bounding box. The age and the visibility of the track is increased by 1. We also set the unassigned tracks as invisible, and increase its age by 1. We compute the ratio of the track's age for which it is visible, that is the track's visibility divided by its age. If the ratio is lower than the specified threshold, the track will be deleted. We create new tracks for unassigned detections. Any unassigned detection is considered as a start of a new track. The final output of the algorithm displays the bounding boxes and labels for each track on the video frame and the foreground mask.

[4]https://en.wikipedia.org/wiki/Hungarian_algorithm

3.2 Contour Extraction

Our framework generally uses the contour extraction algorithm in OpenCV. In order to achieve the better performance we want, we make some adjustment to the method. Here we introduce the basics about the contour extraction algorithm.

3.2.1 Background Subtraction

Existing Background Subtraction module in OpenCV takes each frame in the video and returns the mask on the existing frame to identify the background and foreground. There are different modes for background subtraction, and we choose BackgroundSubtractorMOG because it is most appropriate to identify the exact background contour. There are other modes which may use to identify the shadow of each foreground object. Those are unnecessary and not useful in our scenario. The Background Subtraction algorithm will take previous frames to help accurately identify the background in the current frame.

3.2.2 Contour Extraction

OpenCV has an existing implementation for contour extraction based on image mask of identified objects. Topological structural analysis of digitized binary images by border following is used to do border following to extract contours for objects in each frame of the video. Function findContour is utilized to form the pipeline with the output of Background Subtraction to extract the points that correspond to the contour of each foreground object. Since in our case, only outer contour not hierarchical contours are needed, we choose to use RETR_EXTERNAL for retrieval mode. Values for this retrieval mode parameter is listed below.

- RETR_EXTERNAL, retrieves only the extreme outer contours.
- RETR_LIST, retrieves all of the contours without establishing any hierarchical relationships.
- RETR_CCOMP, retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes.
- RETR_TREE, retrieves all of the contours and reconstructs a full hierarchy of nested contours.

In addition, to speed up drawing of contour, we choose to use CHAIN_APPROX_SIMPLE for approximation mode. That way, we connect the points together, we draw contours of each foreground object onto each frame for demonstration. The values for this parameter is listed below.

- CHAIN_APPROX_NONE, stores absolutely all the contour points.
- CHAIN_APPROX_SIMPLE, compresses horizontal, vertical, and diagonal segments and leaves only their end points.
- CHAIN_APPROX_TC89_L1, applies one of the flavors of the Teh-Chin chain approximation algorithm.
- CHAIN_APPROX_TC89_KCOS, applies another version of the flavors of the Teh-Chin chain approximation algorithm.

4 Method Refinement

In order to implement a more reasonable method and get better performances, we tried different frameworks for contour extraction. We divide the whole method refinement procedure into three steps as followings.

4.1 Find Contours Directly

Since we have our method of contour extraction, it is reasonable to apply the algorithm directly to the input. To be more specific, for every frame of the video, we first do the background subtraction to get rid of the background influences based on the several frames before and after the current frame. Second, we feed the result from background subtraction to the contour extraction algorithm to find contours of current frame. The general procedure is listed in Listing 2.

```
mask = backgroundSubtraction.apply(frame
    )
contours = findContour(mask)
```

Listing 2: Find Contour Directly

This method, as expected, finds almost every contour in the frame, and marks them on the output. However, we can easily find that, from the results presented in the experiments section, there are some contours in and around which there is no object at all. We name such contours as "ghosts" since you cannot find the actual object in the contour. They are like the ghosts that cannot be seen. To attempt solving such problem, we proposed the second version of our method.

4.2 Find Contours in Bounding Boxes

The problem that causes finding the ghosts is that why the contour extraction algorithm thinks there is a contour in that place. As we explore deeper into the implementation of the contour extraction algorithm [6], we find that the algorithm seems quite robust, so the problem did not arise from the contour extraction algorithm, but from the background subtraction method. It is possible that the background subtraction cannot have a good enough estimation of the background due to the videos themselves. For example, the quality of the video may affect the background estimation process. The number of frames used to get the background also has a great influence on the results. Other causes might be the movement of the camera or the moving speed of objects in the video. Under such circumstances, the background results might have several "objects" that were not to be objects and nothing where there should be an object.

In order to solve the problem, we decide to make the most of the results from the object detection phase. The object detection method can provide us with the bounding boxes of the objects. If we apply our method in the bounding boxes, there is no way that we would find contours outside a bounding box, which means we would not find contours if there is no objects present. So in this version, we first get the subarea bounded by the bounding boxes on the frame, and then apply the background subtraction method, followed by the contour extraction method. The general procedure is listed in Listing 3.

```
for boundingBox in boundingBoxes:
    subarea = frame[boundingBox]
    submask = backgroundSubtraction.
    apply(subarea)
    contours = findContour(submask)
```

Listing 3: Find Contour in Bounding Boxes

The results coming out from this method did not go as expected. It is true that we can no longer find the ghosts, but actually, we cannot find any contours at all. As we examine the outputs step by step, we find that, again, the problem is in the background phase. The bounding boxes are located correctly, but no contours is found. As a result, we must reschedule our framework.

4.3 Subtract and Find in Bounding Boxes

The key reason that causes the method to find no contours is that we subtract background within the

bounding box area, which is completely wrong. As we know, the background subtraction algorithm requires the knowledge of previous and afterwards frames of the current frames. If we estimate the background of the bounding box area, since the bounding box is moving almost all the time, we can never get the correct or the nearly correct background estimation. That is to say, the background we subtract within the bounding box is literally not a background, or even worse, it might just be a bunch of irregular pixels.

Now the reason is clear, all we have to do is to exchange the position of the background subtraction phase and the bounding box area extraction phase. Thus, we first subtract background of the whole frame, and then get the subarea in the bounding box from the previous output, and finally find contours in the bounding box. The general procedure is listed in Listing 4.

```
mask = backgroundSubtraction.apply(frame
    )
for boundingBox in boundingBoxes:
    submask = mask[boundingBox]
    contours = findContour(submask)
```

Listing 4: Subtract and Find in Bounding Boxes

This method produces rather good results, almost no ghosts and almost all the contours are marked in the video. It goes as expected that, with background subtraction on the whole frame, we get a reasonable mask, and with bounding boxes on the mask, we exclude the possible subarea where the ghosts might exist. This method actually somewhat has fewer contours detected, which could be seen as balance achievement between the number of detection and the number of ghosts. This is the end of our method refinement.

5 Experiments

We first tried our method on the test video when making adjustments to the method as mentioned in the previous section. Then we test our method on the data set CAVIAR to get the general performance of the method.

5.1 Results and Comparison on Test Video

As presented in Figure 1, the top-left sub-figure shows the positions of the bounding boxes. The top-right sub-figure corresponds to the first version of our method, which apply the background subtraction on the whole image, and directly perform the contour extraction algorithm. The bottom-left sub-figure shows the result of the second version of our method, in which the background subtraction and contour extraction are all performed within the bounding box area. The bottom-right sub-figure is the result of the final version of the method, which first performs background subtraction on the whole image, and then does the contour extraction within the bounding box area.

We can see from the result that, we might find ghosts if we directly find the contours on the video as explained in the previous section. The movement of the objects makes the background subtraction module not able to get the perfect background, leading to the incorrect judgment of the contours. If we perform the background subtraction and the contour extraction within the bounding box area, we would not be able to find the contours since bounding boxes are almost moving and changing all the time, which causes the background subtraction to go wrong. Finally, exchange the positions of background subtraction and subarea extraction, we get the more reasonable results.

5.2 Results on CAVIAR

We experiment our model on CAVIAR Test Case Scenarios Data Set. It consists of a number of video clips of different scenarios, including people walking alone, meeting with others, window shopping, entering and exiting shops, fighting and passing out, leaving a package in a public place, etc.

We can see from the results that our method get a reasonably good performance on the data set. However, it is not as perfect as we expected. We have not come up with a proper metric to evaluate the method, but by human estimation, the accuracy of the contour detection, computed as follows,

$$\text{Accuracy} = \frac{C_{\text{detection}}}{C_{\text{objects}}},$$

where $C_{\text{detection}}$ denotes the number of detected contours and C_{objects} denotes the number of actual objects in the video, is around 90%. While the completeness of the contour, which means how well the contour matches the shape of the objects, is not perfect. We could see some distance between the contours and the actual shapes of the objects. But in general, the method successfully complete the task we would like to do.

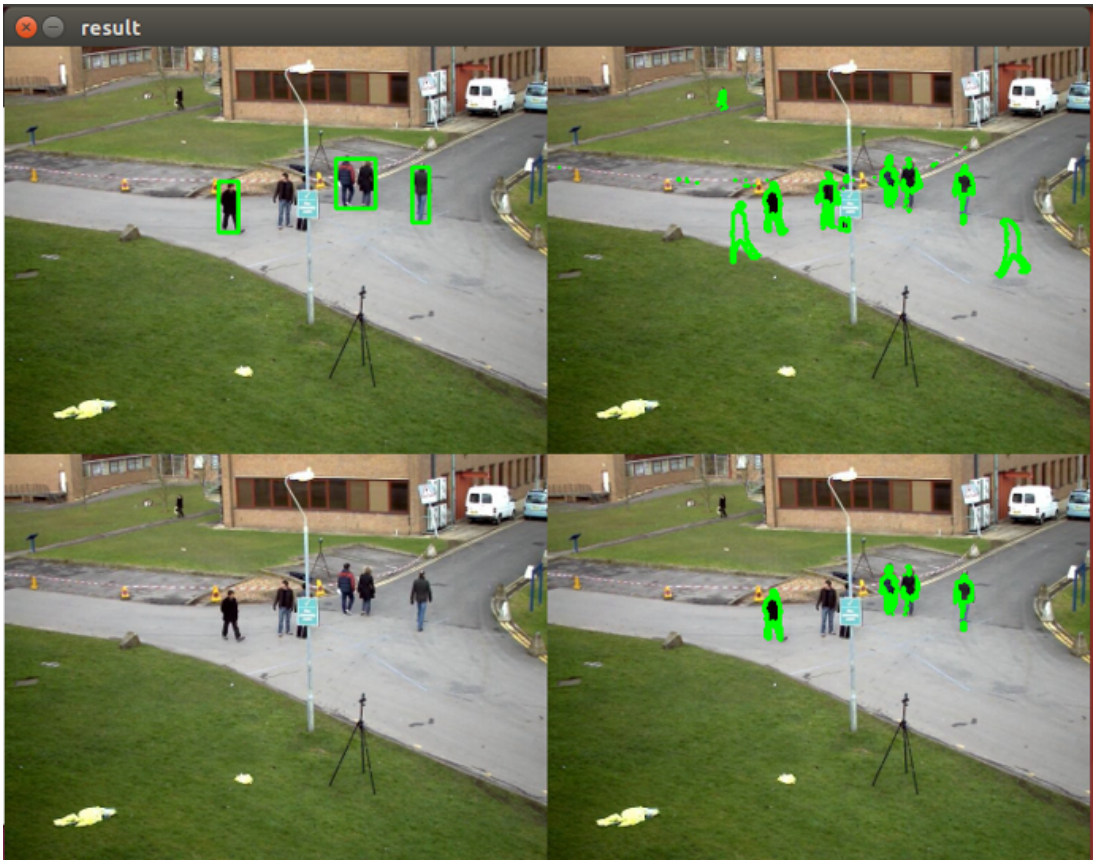

Figure 1: Comparison between different versions of method

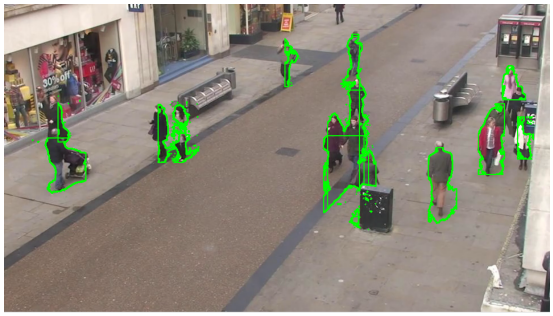

Figure 2: Example result 1 from CAVIAR

Figure 3: Example result 2 from CAVIAR

6 Conclusion

In this project, we implemented a method for object contour tracking in videos. We combined the object detection algorithm and the contour extraction algorithm to reach our goal. We made the most of the bounding box results from the object detection phase, and designed a framework of finding contours on every frame, by first applying background subtraction on the whole frame, and then performing contour extraction within the bounding box areas. As we evaluated our method on the data set, we obtained reasonably good results, which marked the success of the task.

For future work, we would suggest working on the following several aspects. First, the background subtraction phase makes the most contribution to the results, that is, its performance has direct influence on the contour detected in the final step. From our perspective, a better way to discover the temporal relation between the frames should be investigated in order to get a better background estimation. Second, we could try other methods for object detection and contour extraction. The neural network, which is a rather hot

topic, would be a nice choice. Third, we could refine the current results by handling the overlaps of objects and shrinking the contours to the exact shapes. What mentioned above are all possible ways to improve the general performance of the method.

References

[1] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[2] Andrew Blake and Michael Isard. The condensation algorithm-conditional density propagation and applications to visual tracking. In *Advances in Neural Information Processing Systems*, pages 361–367, 1997.

[3] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Kernel-based object tracking. *IEEE Transactions on pattern analysis and machine intelligence*, 25(5):564–577, 2003.

[4] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–194. International Society for Optics and Photonics, 1997.

[5] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[6] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.