Max litvinenko
Winter 2016

Detection and Tracking of Vehicles and Pedestrians

Table of Contents

1. Abstract

There are many approaches to solve computer vision problems of “detecting” and “tracking” objects using camera (or video stream). This paper describes approaches implemented to detect and track “vehicles & pedestrians”. The code was implemented [1] in C++ using an open source OpenCV 3.1.0 library. This particular implementation focuses on detecting cars and people from a traffic surveillance camera. While there are many solutions to this problem, some algorithms may work perfect for detecting objects from static cameras, but completely fail to detect objects in other conditions (ex: camera is static vs moving; camera is at different angle to cars and/or pedestrians). Other algorithms might not detect 100% of objects, but will work in multiple environments. This approach is using ‘Haar’ detection in conjunction of ‘Median Flow’ tracking algorithm and was meant to be reused in multiple environments, however some settings would need to be changed and possibly some re-training for haar features would need to be done.

2. Tracking

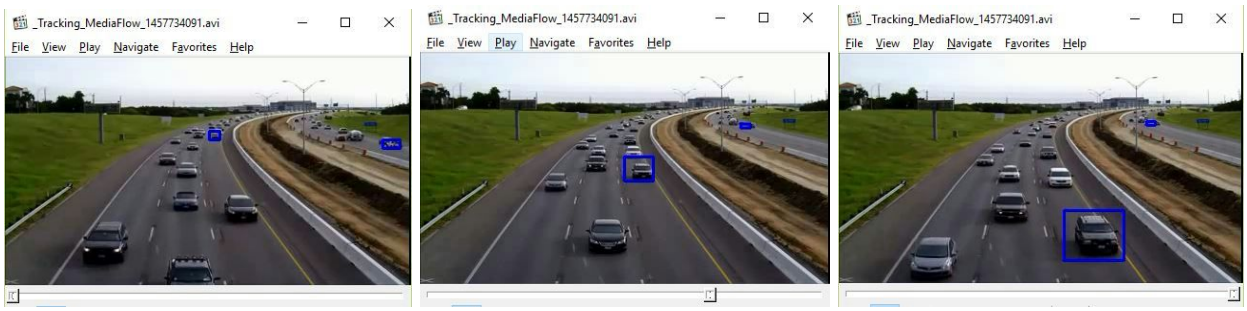
OpenCV 3.1 provides handful number of tracking algorithms [2] that are already implemented for us to use. After testing all of them using tutorial provided by opencv [3] for tracking multiple objects at once, best candidates for the final implementation became “KCF” [4] and “Median Flow” [5].

KCF

KCF (Kernelized Correlation Filters) is a novel tracking framework that utilizes properties of circulant matrix to enhance the processing speed. It is extended to KCF with color-names features. [4]

Median Flow

The tracker is suitable for very smooth and predictable movements when object is visible throughout the whole sequence. [5]



Decision

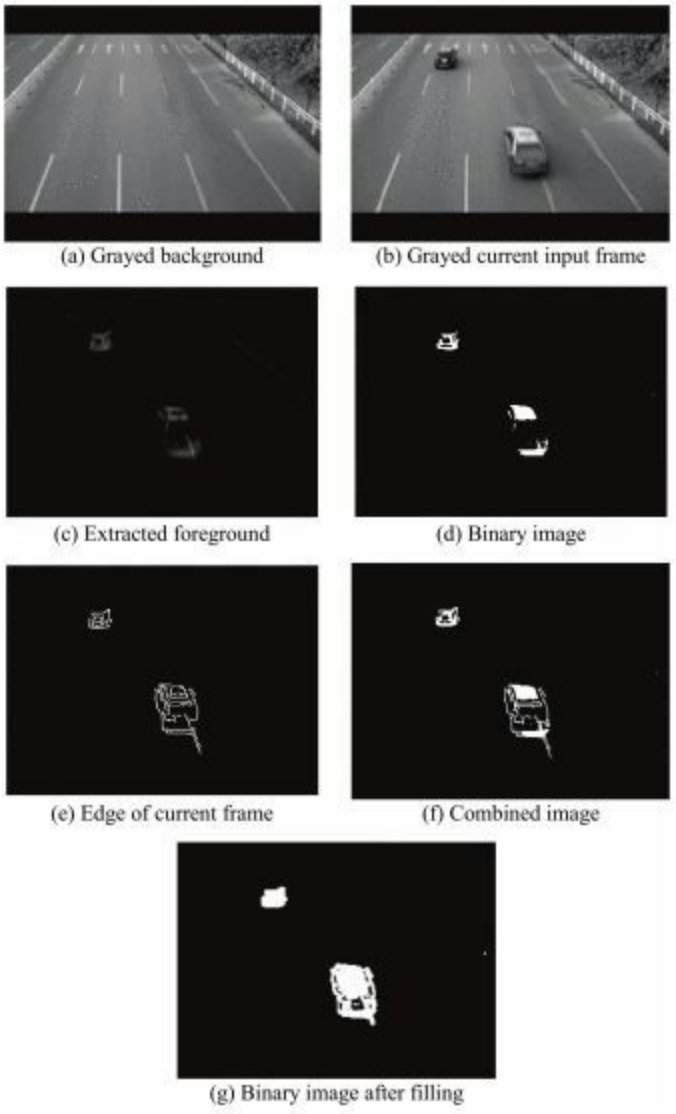
There are a few reasons why these two particular algorithms were chosen. One reason is that they both were significantly faster in comparison to all other ones (“Boosting”, “MIL”, and “TLD”). They also performed a good job with tracking objects that are visible on all of the frames which is what was needed. The final decision was made to use “Median Flow”, because it was tracking the size of the object as well. KCF was seen to get smaller *in some rare cases, in some videos* when the car was going farther, but when the car comes closer to the camera, KCF it was not tracking that. “Median Flow”, on the other hand, was perfectly recognizing and tracking when a car moves farther away and/or closer to the camera. In addition to that, “Median Flow” was a bit faster than “KCF”.

3. Detection

There were two main algorithms that were considered for detecting objects in this project. Blob detection and using Haar cascades algorithms.

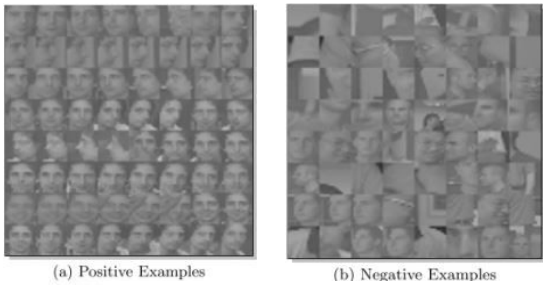
Blob

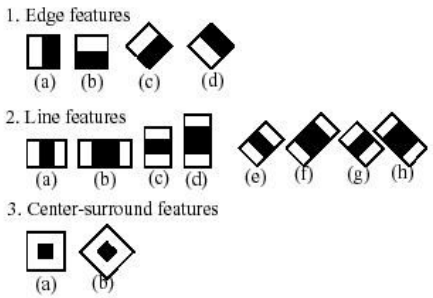
Blob detection is one of common algorithms that is used to detect the foreground object that is moving from a background image that is static. See image on a right side that explains one of the varieties of blob detection algorithm that could be used for this project. (Image is taken from scholarly written article [6]). Main idea of this algorithm is to work in grayed image (a, b), find a background image by subtracting pixels that are changing from static pixels that most of the times do not change(a). When you found a background image, you need to subtract a current frame from the background frame to get the difference between the two(b). There will also be some other steps and settings performed that will adjust the image by removing the noise pixels. This algorithm requires some ‘learning time’ where it finds the background first.


(a) Grayed background
(b) Grayed current input frame
(c) Extracted foreground
(d) Binary image
(e) Edge of current frame
(f) Combined image
(g) Binary image after filling

Once you have the blobs, you can pass the blob object to a tracker algorithm to track it.

Haar


(a) Positive Examples (b) Negative Examples



Haar detection algorithm [7] also requires some time for training the classifier. However it's much much larger and a separate process. Basically what is being done is a classifier is being trained by taking a very large set of images of the same size for a particular object (Car, Pedestrian, etc). First step is to train for positives and second step is to train for negative images (a, b) [8]. Its being trained by applying special training Haar-like features shown on the image below. After the haar classifier is trained, the xml file generated. In our implementation we re-used publicly available haarcascades found from different sources [9].

XML files are loaded to a base cascade classifier and then being used by passing ‘gray frame’ of the video(or image) in OpenCV’s function that is already implemented for us. There, you will have to specify needed parameters. The most important ones need to be adjusted for each video stream is the ‘scale factor’ and ‘min and max sizes’:

```
virtual void cv::BaseCascadeClassifier::detectMultiScale  (
        InputArray                  image,
        std::vector< Rect > &       objects,
        std::vector< int > &  numDetections,
        double                scaleFactor,
        int                         minNeighbors,
        int                         flags,
        Size                        minSize,
        Size                        maxSize
)
```

This method will go through the whole frame in small chunks (of minSize), probably from ‘left to right’ and from ‘top to bottom’ and will detect objects. Detected objects will be ‘rectangle objects’ returned to an argument called ‘objects’. These will be detected objects in current frame. This these objects could be passed to a tracker to track them.

Decision

Overall, both algorithms would be acceptable for the problem of ‘detecting cars and people from a traffic surveillance camera’. However, HAAR detection algorithm was chosen because it seemed more advanced and more reusable in different environments, compared to blob detection. Reusability means, haar could be used in cameras that are moving (and/or background is changing), which is crucial for blob detection. However, there needs to be more training performed to output a good number of objects for different environments.

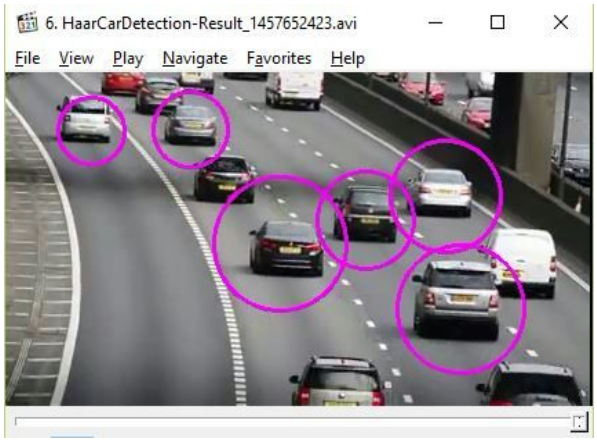
Notice, in blob detection we do not discuss the classification of which object is detected, a car, pedestrian, a dog, or a big shadow, which is also a big problem for blob detection.

4. Detection + Tracking

Discussing both processes of HAAR detection and MEDIAFLOW tracking, they would eventually to need to be combined to complete this project.

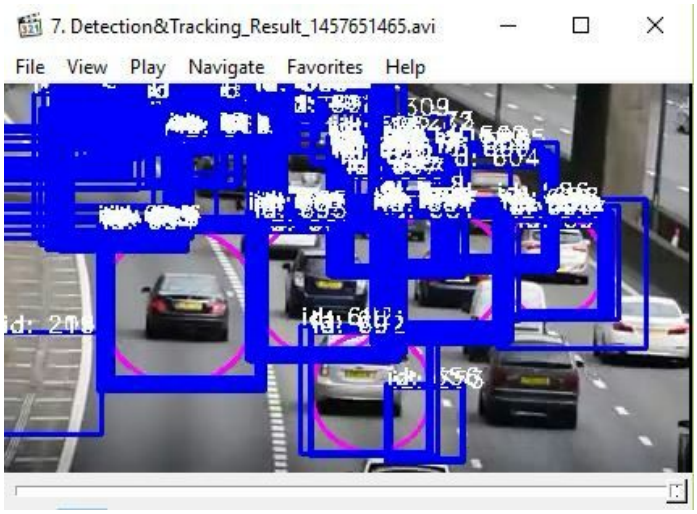
Implementation

The complete implementation [1] uses the following algorithm. For infinite loop in the camera, or the number of frames in the video, we use HAAR detection to detect cars and people (in our implementation it also marks them in pink circle, see image 1). Then, we pass these objects to multi-tracker, which will automatically track these objects in next frames (in our implementation it also marks the tracking objects in blue rectangles, see image 2).



Problems

Since we keep detecting the same object (same car or pedestrian) in next frames, we keep passing them to tracker as a *new* recognized object. Current implementation lacks of a 'Smart Tracker' that will compare a current new object to the list of already tracking objects, if they are within a threshold, we would ignore this new object.



Another problem is that current tracker never removes the old tracked objects, thus we continue to see blue squares on the screen.

See image For output videos please take a look here [10]

5. Conclusion

Solving the computer vision problem of 'Detection and Tracking of Vehicles and Pedestrians', HAAR cascade detection and MEDIANFLOW tracking algorithms perform a remarkably great job working together in theory and in implementation provided (it is visible even when the implementation of 'smart tracker' is not 100% complete). However, there might be other algorithms out there that may beat current implementation in some environments. One of the test would be to use blob tracking instead of haar, explained here in "Real-time moving vehicle detection, tracking, and counting system implemented with OpenCV" [6]

6. References

1. https://github.com/maxpdx/cs410-opencv-detection-tracking/blob/master/OpenCV%20-%20Object%20detection%20and%20tracking/Main.cpp
2. http://docs.opencv.org/3.1.0/d0/d0a/classcv_1_1Tracker.html#gsc.tab=0
3. http://docs.opencv.org/trunk/d5/d07/tutorial_multitracker.html#gsc.tab=0
4. http://docs.opencv.org/3.1.0/d2/dff/classcv_1_1TrackerKCF.html#gsc.tab=0
5. http://docs.opencv.org/3.1.0/d7/d86/classcv_1_1TrackerMedianFlow.html#gsc.tab=0
6. Real-time moving vehicle detection, tracking, and counting system implemented with OpenCV
7. http://docs.opencv.org/3.1.0/d5/d54/group__objdetect.html#gsc.tab=0
8. http://www.diva-portal.org/smash/get/diva2:699483/FULLTEXT01.pdf
9. https://github.com/maxpdx/cs410-opencv-detection-tracking/tree/master/OpenCV%20-%20Object%20detection%20and%20tracking/haarcascades
10. https://github.com/maxpdx/cs410-opencv-detection-tracking/tree/master/Project%20Presentation/Demos