Face Detection and Tracking using Optical Flow

Shubhanga Narasimha (111499997) and Vertika Vaid (111499526)

December 15, 2017

Abstract

In this project, a combination of Viola-Jones face detection and Optical Flow tracking is used to track a face in a video. The Viola-Jones face detector being a static face detector, fails abruptly in many frames of video. To increase the efficiency of tracking the face in all the frames of the video, temporal information from the previous frames are used to keep track of the face, which is then used when the Viola-Jones face detector fails.

1 Introduction

Viola-Jones face detection algorithm for face detection was first presented in 2001 [1]. Even though, this technique performed face detection in real-time by applying a classifier on several windows within the image, it does not make use of any temporal information in the previous frames of the video to track the face in the present frame. This makes the Viola-Jones face detection algorithm static, since it does not incorporate the temporal constraints to make an inference of the face detection. Also, in case that the face in a few frames of the video turn out to be distorted due to various reasons, the detector fails abruptly. Thus in a high frame-rate video, the Viola-Jones detector might fail in the successful detection of the face in multiple frames depending on the face it is tracking and its movements. Temporal information from the previous frames can aid in successful tracking in the new frames in such cases where the face detector fails, thus making the new face detector robust to distortions and face detection failure. Such a temporal technique can be used to track the face in subsequent frames until the face detection is successful again by the Viola-Jones face detector. Such a solution can be implemented using the Optical Flow techniques which we will discuss in detail.

2 Design

2.1 Viola-Jones face detector

Viola-Jones face detector works by using a cascade of classifiers which help in the real-time detection of faces. These classifiers use Haar-like features i.e. has horizontal and vertical kernels of ones and zeros that can analyze a window of information (of multiple sizes), one at a time. This window constitutes the hypothesis which is passed to the cascade of classifiers which is designed to pass it

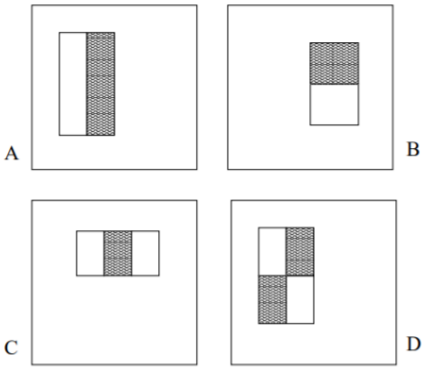

Figure 1: Example Rectangle features

onto the next level in the cascade, in the case that a face exists. If no face exists, it is discarded. If the window successfully passes through all the classifiers in the cascade, the input image is considered to contain a face. A video containing frames of images are passed one after the other as the input to the face detector and it can perform face detection with a fairly good accuracy at real-time. It assumes that the sequence of frames in the video as independent and does not use previous information for subsequent analysis.

Viola-Jones face detector uses haar like rectangular features as shown in Figure 1. Here to get the features from these rectangles the sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles.

To calculate these features very fast an integral image is used which is sum of the pixels above and to the left of a point x,y inclusive. So for example as given in Figure 2 the sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A+B, at location 3 is A+C, and at location 4 is A+B+C+D. The sum within D can be computed as $4 + 1 - (2 + 3)$.

2.2 Optical Flow

Videos however do make use of minor temporal changes to create an effect of motion. In essence, the information in the previous frames about the location and the size of the face can be used in the current frame to track the face better and faster. In this regard, multiple ideas are presented in [2] which use the Viola-Jones face detector once in every 20 frames of a video and using Optical Flow to track the face in a probabilistic manner by creating a likelihood map of the face, in between the 20 frames. This reduces a lot of complexity regarding the face detection in every frame which would reduce the performance. This is however achieved at the cost of increasing the complexity in the algorithm to handle certain errors at time of occlusion. However in view of simplicity, we

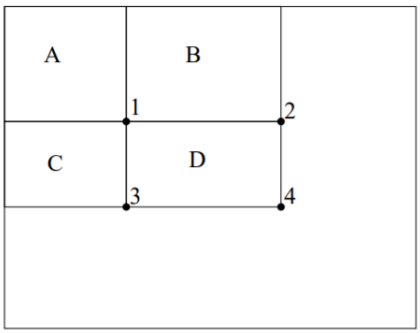

Figure 2: Example Integral Image

consider in reducing the complexity and use the Optical Flow technique only in the frames where the face detector fails us to provide a positive result.

2.3 Face detection and tracking using Optical Flow

The flowchart for real-time face detection and tracking using optical flow is given in the Figure 3.

Optical Flow works on the concept of brightness constancy and spatial coherence, which makes it easier to find a maximum likelihood map of the face in a frame where the face detector fails. The face detector gives us the location of the face in the frame with a width and height of the face in it. Using spatial coherence, we can safely assume that these parameters remain a constant in the next frame and that sudden motion is not possible at the current frame rate of the video. With such a practical assumption, if we find good features to track using the Shi-Tomasi corner detection within the window of this width and height, we can use optical flow to check its temporal motion to keep a track of the face. Here the likelihood map of the face is a simple equivalence to the width and the height of the face returned by the Viola-Jones face detector.

With these optical flow calculations, we can now have two options for the next frame in the video. We can choose to detect the face. In the inadvertent scenario that the face detector fails, we have the optical flow calculations of the previous frames with us to help us calculate the likelihood of the face in the frame. This is used in the next frame and the process continues. The Shi-Tomasi corners make sure that the features selected on the face are strong and unique which make errors less prone and makes this technique robust.

3 Implementation details

The implementation of this idea was done in OpenCV 3.3 using Python. The OpenCV libraries provide a wide variety of library functions that can be used for implementing this idea fairly simple. It also provides an interface to load videos, work on its frames as images and show them one after the other in a named

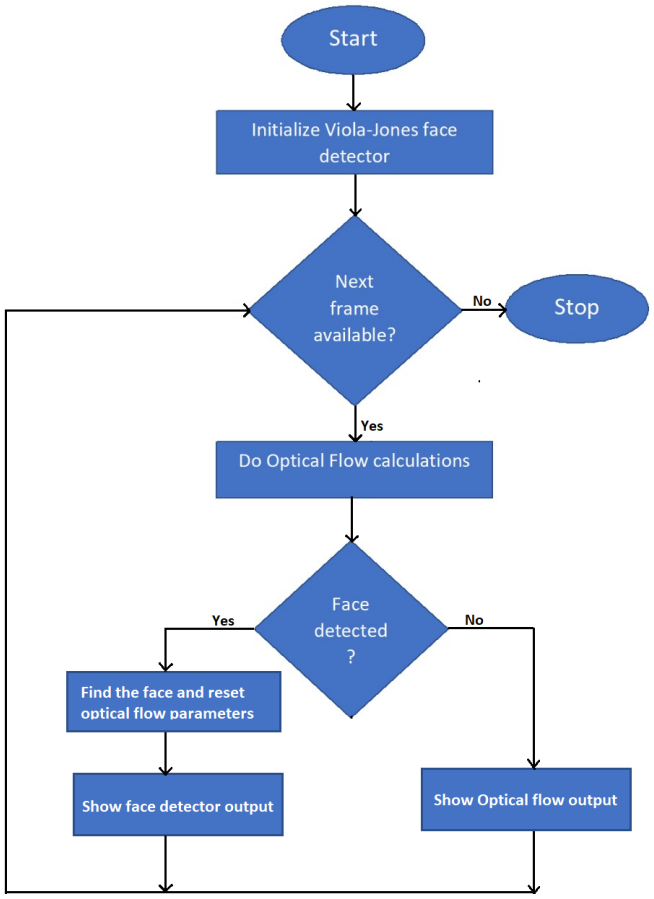

Figure 3: Flowchart of the algorithm

window with a certain delay, which makes it seem like a video being played. All in all, OpenCV provides a comfortable platform to create fast proof-of-concepts.

The face detector with the cascade classifier is created from the function `cv2.CascadeClassifier()`, which takes in an `xml` file containing the cascaded classifiers of Haar-like features, which is used for face detection. This functionality is then used frame by frame of the video which to get the coordinates and the dimension of the face. The frame of the video is converted to grayscale to have only the intensity information which is used for face detection. If no face is detected initially, we proceed to the first frame in which a face can be detected. With such an initial information about the face, we then proceed to retrieve the face information in the subsequent frames using optical flow.

With the first frame containing the face, we proceed to initialize the parameters for optical flow. A mask is first created to identify the area of the image that contains the face. We now proceed to use `cv2.goodFeaturesToTrack()` to get Shi-Tomasi corners within the mask of the face. In our case, we use best corners with a minimum distance of 10 and a quality level of 0.001. This ensures that the retrieved corners are really precise within the face and cannot be confused with other similar corners within the mask.

With this initialization, we proceed to retrieve one frame at a time from the video, convert it to grayscale and try to detect the face. If the face detection is successful, then the priori information about the face we had in the previous frame is updated for future use. If the face detection is not successful the calculations made from the optical flow about the location and the dimensions of the face are used to show the location of the face. The mean of the location of corners in the face that are getting tracked is calculated. The width and the height of the face is taken relative to the further most corner that is being tracked in all the four direction. In either case, optical flow calculations are done using `cv2.calcOpticalFlowPyrLK()` from the previous frame to the current frame, to identify the motion of the face in the video.

With every successive frame, the new parameters of the face are continuously updated to keep track of the movement of the face within the frame. With this method, we can clearly see that the optical flow calculations can be used to show the face location and dimension when the face detector works.

To run the program that is attached with this report, please use the following command in any computer that has Python installed:

`python OpticalFlowFaceTracking.py video1.avi`.

The first and the only argument to the program is the name of the video file for which the face detection and tracking has to be done.

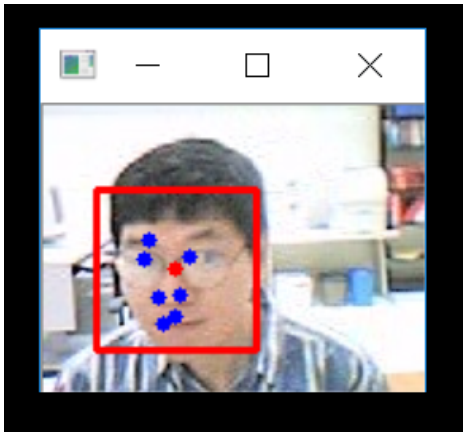

Figure 4: Output of successful face detection and corner points on face

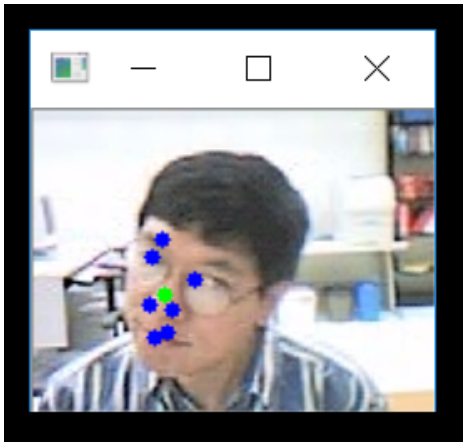

Figure 5: Output of unsuccessful face detection and the mean of all the corner points on face

4 Observations

The implementation of optical flow along with Viola-Jones face detector proved to be very efficient in terms of the face tracking for every frame. Let us first set the conventions clear in the video that is displayed in the output. This is shown in the Figure for understanding

- We output a red square equal to the dimensions of the face around it, whenever the face detector is successful. Also, the centre of the square is marked by a red dot. This can be seen in Figure 4.
- The blue dots on the face represent the best corners found using the Shi-Tomasi corner detectors. The features under the blue dots represent the features that we track in the frames of the video. This can be seen in all the frames of the video and can be seen in both Figure 4 and 5.
- When the face detector fails, we remove the red square and the red dot. The face detection is now shown by optical flow calculations, whose output is marked by a green dot. The green dot is the mean in both the directions of all the blue dots, that represent the best corners in the face. This can be seen in Figure 5.

We tested the implemented algorithm on 5 different videos with the subjects making different motions. The motions include moving away/towards the camera (scale), rotation, translation, occlusion and sudden movement. For each of the videos, the following observations are made. We also mention the number of frames in which the face detector was successful and the number of frames in which optical flow aided us in tracking the face.

Video - 1: In this video, the robustness of the algorithm wrt. scaling and translation can be seen. The corners in the face keep tracking the face successfully even though the subject makes complex scaling and translation movements simultaneously. It can be observed that during many of such movements, the face detector fails, however the optical flow fires in to show the green dot precisely at the same spot where the red dot had been. Also below is the statistics for the video.

```
$python OpticalFlowFaceTracking.py video1.avi
No of frames in which face is detected =  169
No of frames in which optical flow output is used =  87
Total frames =  256
```

Video - 2: In this video, the subject tests the algorithm with occlusion using the hand and also in the process of wearing spectacles. During the above two actions, it can be observed that some of the corners that are tracked lose focus, go out of the frame and are lost. However, the remaining corner points still cling on to the face and their mean still points correctly to the face centre. From this we observe that even though there is a lot of distortion in the video, the face detection is successful. Here is the statistics for the video.

```
$python OpticalFlowFaceTracking.py video2.avi
No of frames in which face is detected =  157
No of frames in which optical flow output is used =  91
Total frames =  248
```

Video - 3: In this video, the same subject tests the algorithm by occluding the face by wearing spectacles. The algorithm is proved robust even so, by the following results.

```
$python OpticalFlowFaceTracking.py video3.avi
No of frames in which face is detected =  185
No of frames in which optical flow output is used =  42
Total frames =  227
```

Video - 4: In this video, the subject makes sudden rotational movements. The algorithm still tracks the face using optical flow techniques even though the face detection fails frequently, since the spatial coherence is still within the limits of assumed variability between the frames. The results are as shown below.

```
$python OpticalFlowFaceTracking.py video4.avi
No of frames in which face is detected =  208
No of frames in which optical flow output is used =  137
Total frames =  345
```

Video - 5: In this video, the subject makes sudden scale movements i.e., the subject moves away/towards the camera rapidly. As in the case with video 4, similar results can be observed.

```
$python OpticalFlowFaceTracking.py video5.avi
No of frames in which face is detected =  225
No of frames in which optical flow output is used =  112
Total frames =  337
```

5 Results

In the above observations for the 5 videos, it can be clearly seen that the optical flow algorithm helps in successful face tracking in 100% of the frames. The face detection fails for a large number of frames, however the optical flow algorithm tracks the face for **ALL** the remaining number of frames. The efficiency is increased to 100%, which is a huge achievement. For all the 5 videos, here is the efficiency improvement to reach 100%, respectively:34%, 37%, 18%, 40% and 33%.

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1, pp. I–I, IEEE, 2001.

[2] A. Ranftl, F. Alonso-Fernandez, and S. Karlsson, "Face tracking using optical flow," in *Biometrics Special Interest Group (BIOSIG), 2015 International Conference of the*, pp. 1–5, IEEE, 2015.