COMP-558

Fundamentals of Computer Vision

Face Detection and Tracking with Web Camera

Ruslana Makovetsky

and

Gayane Petrosyan

Contents

1 Introduction

Human-robot interaction receives an increasing attention among the researches in different areas of interest during last years. In the current project we address one of the computer vision tasks, involved in developing of such an interactive system. Namely, we are interested in detecting human user presence and tracking of his attention. Possible applications for this would include receptionist robot, which needs to perform particular actions when human approaches or leaves the system, interactive games, commercials etc.

The goal of the current project was to implement a commercial player, having the following specifications:

- automatically start playing a movie if human approaches the display,
- pause a movie if human loses his attention (for example, turns away),
- stop a movie if human leaves the location.

As a a result, we were looking for a face detection and tracking algorithm, which would enable us to detect faces in frames, forwarded from a web camera, and meeting the following requirements:

- real-time performance,
- variability in scale (person may move back and forward),
- in-plane rotations (person may shake his head),
- possible occlusions (person may perform actions, such as bringing a cup to his face, partially cover his face by hands, etc.)
- ability to distinguish between profile and non-profile faces - this requirement is particularly important for our application, since it enables to detect situations when user loses his attention.

The algorithm presented by Viola and Jones for real-time face detection [8] and its extension for multiple views [6] were adopted for the system implementation. The performance of the implemented system is reasonably good for the real time system - the systems processes about 8 frames per second. Demonstration videos for a single and multiple faces are available on the project website [4].

2 Algorithm

2.1 Outline

We start this section with the description of a framework for real-time face detection, presented by Viola and Jones [8]. This work had three key contributions:

1. It introduced a new, efficient, method for features calculation, based on an "integral image";

2. It introduced a method for aggressive features selection, based on AdaBoost learning algorithm; and finally

3. It introduced an idea of combining classifiers in a "cascade".

We will then describe an extension for this algorithm, presented by Viola and Jones [6], designed to deal with different head poses.

2.2 Viola-Jones Algorithm for Face Detection

2.2.1 Features

The problem of face detection can be viewed as a problem of binary classification of image frame as either containing or not containing a face. In order to be able to learn such a classification model, we first need to describe an image in terms of features, which would be good indicators of face presence or absence on a given image.

For their face detection framework Viola and Jones [8] decided to use simple features based on pixel intensities rather then to use pixels directly. They motivated this choice by two main factors:

- Features can encode ad-hoc domain knowledge, which otherwise would be difficult to learn using limited training data.

- Features-based system operates must faster than a pixel-based system.

The authors defined three kinds of Haar-like rectangle features (see Figure 1):

- Two-rectangle feature was defined as a difference between the sum of the pixels within two adjacent regions (vertical or horizontal),

- Three-rectangle feature was defined as a difference between two outside rectangles and an inner rectangle between then,

- Four-rectangle feature was defined as a difference between diagonal pairs of rectangles.

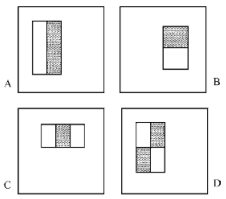


Figure 1: Rectangle features example: (A) and (B) show two-rectangle features, (C) shows three-rectangle feature, and (D) shows four-rectangle feature.

To enable an efficient computation of rectangle features, the authors presented an intermediate representation of an image, called an integral image. The value of integral image at location x, y is defined by the sum of the pixels from the original image above and to the left of location x, y inclusively (see Figure 2). Formally,

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'), \tag{1}$$

where $ii(x, y)$ is an integral image and $i(x, y)$ is an original image.

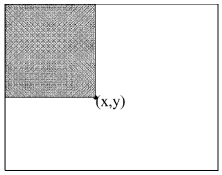


Figure 2: Integral image

Maintaining a cumulative row sum at each location x, y, the integral image can be computed in a single pass over the original image. Once it is computed, rectangle features can be calculated using only a few accesses to it (see Figure 3):

- Two-rectangle features require 6 array references,
- Three-rectangle features require 8 array references, and
- Four-rectangle features require 9 array references.

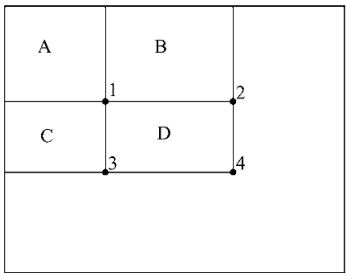


Figure 3: Calculation example. The sum of the pixels within rectangle D can be computed as 4 + 1 - (2 + 3), where 1-4 are values of the integral image.

The authors defined the base resolution of the detector to be 24x24. In other words, every image frame should be divided into 24x24 sub-windows, and features are extracted at all possible locations and scales for each such sub-window. This results in an exhaustive set of rectangle features which counts more than 160,000 features for a single sub-window.

2.2.2 Learning Classification Functions

The complete set of features is quite large - 160,000 features per a single 24x24 sub-window. Though computing a single feature can be done with only a few simple operations, evaluating the entire set of features is still extremely expensive, and cannot be performed by a real-time application.

Viola and Jones assumed [8] that a very small number of the extracted features can be used to form an effective classifier for face detection. Thus, the main challenge was to find these distinctive features. They decided to use AdaBoost learning algorithm as a feature selection mechanism.

In its original form, AdaBoost is used to improve classification results of a learning algorithm by combining a collection of weak classifiers to form a strong classifier. The algorithm starts with equal weights for all examples. In each round, the weight are updated so that the misclassified examples receive more weight.

By drawing an analogy between weak classifiers and features, Viola and Jones decided to use AdaBoost algorithm for aggressive selection of a small number of good features, which nevertheless have significant variety.

Practically, the weak learning algorithm was restricted to the set of classification functions, which of each was dependent on a single feature. A weak classifier $h(x, f, p, \theta)$ was then defined for a sample x (i.e. 24x24 sub-window) by a feature f, a threshold θ, and a polarity p indicating the direction of the

inequality:

$$h(x, f, p, \theta) = \begin{cases} 1 & \text{if } pf(x) < p\theta, \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The key advantage of the AdaBoost over its competitors is the speed of learning. For each feature, the examples are sorted based on a feature value. The optimal threshold for that feature can be then computed in a single pass over this sorted list. To achieve this, we need to maintain four sums for each element in the sorted list:

- T^+ - the total sum of positive example weights,
- T^- - the total sum of negative example weights,
- S^+ - the sum of positive weights below the current example, and
- S^- - the sum of negative weights below the current example.

The error for a threshold which splits the range between current and previous example in the sorted list is the minimum of the error of labeling all the examples below the current example as negative (and all other examples as positive) versus the error of labeling all the examples below the current example as positive (and all other examples as negative). Formally,

$$e = \min(S^+ + (T^- - S^-), S^- + (T^+ - S^+) \tag{3}$$

In their paper [8], Viola and Jones show that a strong classifier constructed from 200 features yields reasonable results - given a detection rate of 95%, false positive rate of 1 to 14,084 was achieved on a testing dataset. These results are promising. However, authors realized that for a face detector to be practical for real applications, the false positive rate must be closer to 1 in 1,000,000. The straightforward technique to improve detection performance would be to add features to the classifier. This, unfortunately, would lead to increasing computation time and thus would turn the classifier into inappropriate for real-time applications.

2.2.3 Detectors Cascade

There is a natural trade-off between classifier performance in terms of detection rates and its complexity, i.e. an amount of time required to compute the classification result.

Viola and Jones [8], however, were looking for a method to speed up performance without compromising quality. As a result, they came up with

an idea of detectors cascade (see Figure 4). Each sub-window is processed by a series of detectors, called cascade, in the following way. Classifiers are combined sequentially in the order of their complexity, from the simplest to the most complex. The processing of a sub-window starts, then, from a simple classifier, which was trained to reject most of negative (non-face) frames, while keeping almost all positive (face) frames. A sub-window proceeds to the following, more complex, classifier only if it was classified as positive at the preceding stage. If any one of classifiers in a cascade rejects a frame, it is thrown away, and a system proceeds to the next sub-window. If a sub-window is classified as positive by all the classifiers in the cascade, it is declared as containing a face.

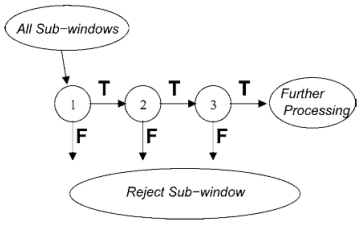


Figure 4: Detectors cascade

Since most of sub-windows do not contain faces, only a small number of them are processed by all the classifiers in a cascade. The majority of sub-windows are rejected during first few stages, involving simple classifiers solely, which results can be evaluated very rapidly. Thereby, the overall detection performance meets the requirements of real-time processing, and yet it yields high detection rates, comparable with alternative, much slower, techniques.

2.3 Extension of Viola-Jones Algorithm for Multiple Views

The main disadvantage of an algorithm, originally presented by Viola and Jones [8], is that it can only detect frontal, upright faces with some moderate deviation for in plane and out of plane rotations. Viola and Jones proposed an extension for the algorithm designed to detect faces from multiple view points [6].

One of the possible approaches would be to train a single classifier to detect all poses of a face. However, authors state that such an approach is unlearnable given an existing classifiers. They performed informal experiments when a classifier was trained on all poses, and the classification

performance hopelessly inaccurate.

An alternative approach, selected by Viola and Jones [6], is to divide the space of poses into various classes and train different detectors for each pose class. In order to avoid the computational expense of evaluating every detector on every window, authors used a two stage approaches which first estimated the pose of the face, and then evaluated only the detector trained on that pose.

Given that the frontal face detector [8] handles approximately ± 15 degrees of in-plane rotation (i.e. 30 degrees of a space), Viola and Jones trained 12 different detectors for frontal faces in 12 different rotation classes (covering the full 360 degrees of possible rotations) [6].

In addition, Viola and Jones [6] trained a right profile detector. Since in-plane rotations are quite common for the profile faces, originally defined two, three, and four rectangle features were not sufficient to detect faces in these poses with sufficiently high accuracy. To address this issue, Viola and Jones [6] created a new type of rectangle features that focused on diagonal structures in the image (see Figure 5 for an example).

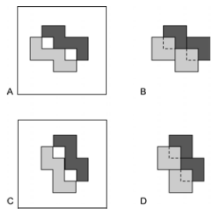


Figure 5: Diagonal features example

To compute such feature, 16 references to the integral image are required.

3 Implementation

This section describes an implementation of the system for face detection and tracking algorithm, which would enable us to detect faces in frames, forwarded from a web camera. This section also discusses some of the issues we met along the path and decisions made to overcome them.

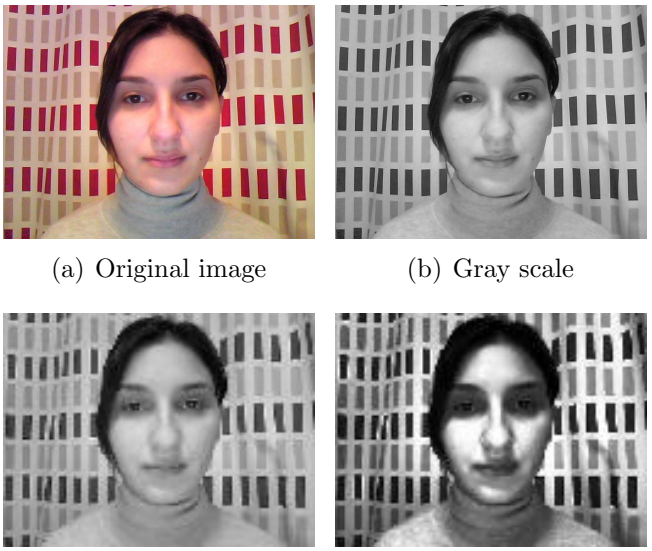
For implementation we used JavaCV [2] as a wrapper for OpenCV [3]. The library provides trained classifiers for frontal face and right profile. One way to obtain an ability to detect other poses, would be to train additional classifiers, as it was proposed by Viola and Jones [6] and discussed in section 2.3, Though technically it is clear how to do that, it would require

significant amount of resources. We would need large database of positively and negatively labeled images and probably weeks to train a classifier for that dataset.

Alternatively, we decided to use only provided two classifiers to cover major part of head poses. The algorithm implementation, as well as pre-processing optimization and post-processing steps required by our implementation are discussed in the following sections.

3.1 Pre-processing

Before passing each sequential frame to face detecting algorithm the frame is processed to make the algorithm more efficient and more precise.



(a) Original image (b) Gray scale

(c) Sampling (d) Histogram enhancement

Figure 6: Pre-processing steps.

First, frame is converted from RGB format to GRAYSCALE (see Figures 6(a) and 6(b)). Then, image size is reduced by performing pixels sampling. This helps the algorithm to work faster. In our implementation, the original image was scaled down by factor of 5 (see Figure 6(c)).

The last pre-processing step is aimed to improve precision of face detection algorithm. During this step the brightness is normalized and contrast is increased. As a result, the black and white features become significantly more noticable (see Figure 6(d)).

3.2 Face detection

After pre-processing the image, frame is forwarded to the next step of the algorithm, where different face poses are detected. As we mentioned before, classifiers for frontal face and for right profile are available within the OpenCV library. Three more detectors were introduced by the algorithm based on the ones already provided: left profile detector, tilt to right detector, and tilt to left detector.

Left profile detector is based on the same classifier as right profile detector. However, before using the classifier the image is mirrored and only after that the classifier is applied. If the face is found the coordinates of box containing the image are flipped back to correspond to actual image.

Almost the same idea is used for two other new detectors. Tilt to right and tilt to left detectors are based on the frontal face classifier. However, in this case, the images are rotated by 30 degrees to left or to right. If the original image contained the face with in-plain rotations, then after the image is rotated in the opposite direction, the face appears to have the frontal pose. Therefore standard frontal face detector can be applied. In this case as well, for detected faces the coordinates of face boxes are appropriately processed back to match to initial image.

Now we have 5 classifiers to apply on images. If we assume that image frames contain only one face, then we can try these classifiers sequentially until one of them report positive result. However, if we want to support multi-face image frames the sequential application of 5 classifiers might be quit slow and not usable. For that purposes we added multithreading to our system to be able to support multi-face image frames. Each classifier is applied to the same pre-processed image within different thread. When all of them finish to work, the resulting sets of faces are combined and provided as an input for the following post-processing step.

3.3 Post-processing

After detection algorithm is applied we have a set of all detected faces with corresponding classifier type and coordinates of box containing the face. One of the major problems of the original algorithm is the huge number of false positives.

To overcome this problem, S. Wang and A. Abdel-Dayem [9] suggested pre-processing and post-processing methods based on skin color filter and showed that the post-processing method works better. They used filters derived by J. Kovac et al. [7] to distinguish between face and not face regions of the image. The suggested algorithm is simple for implemtation. It tests

every pixel in the face detection box. If the number of “skin“ pixels over number of all pixels is smaller than the threshold then the box does not contain a face. We implemented the suggested post-processing method for each detected face to eliminate some of the false positives. This method significially improved the problem with false-positives, but yet not eliminated all of them.

As can be seen on from Figure 8, some head poses are detected by more than one classifier. For this reason, the second post-processing step is required to merge detection which refer to the same region of the image, and thus to the same face. In order to perfrom the match, we look for intersection of detection boxes, based on their coordinates. Different heuristics can be applied at this step to selecting one of the intersecting faces and to eliminate the other. We introduced an idea of priority between the face poses and made decision based on that priority. For five detectors the priorities are assigned from highest to lowest in the following order: frontal face, tilt left, tilt right, left profile, right profile. From pair of intersecting faces we choose the one that has higher priority. By changing the heuristic the behavior of the system might change, but the chosen heuristic is fast, easy to implement and reasonable in practice for webcam applications.

3.4 Media Player

The implementation of media player was based on Java Media Framework (JMF) [1]. The only disadvantage of this framework is that it is somewhat deprecated, and for this reason does not support advanced video formats. We used MPG video format along with the implemented media player.

The player implements the following logic. While at least one frontal or tilted face is detected, player plays the movie. If only profile faces were detected, the movie should be paused. Otherwise, the movie stops.

4 Results

4.1 Head poses coverage

4.1.1 Out-of-plane rotation

As it was already discussed in the implementation section, face detection algorithm uses two trained classifiers - one for frontal faces and another for right profiles. The algorithm is built on the assumption that these two classifiers are enough to cover main variety of poses for webcam applications.

To verify this assumption, an experiment has been run on a set of images to check the detection coverage for out of plane rotations of the head. Experimental dataset is shown on Figure 7. It consists of 42 sequential photos, which have been made with pose difference of approximately 4.2 degrees between two consequential images, and cover from -90 degree to 90 degree of out of plane face rotations. This density represents all real world head rotations. Detectors for frontal face, left profile and right profile have been applied on this experimental dataset of 42 images.

The results of the experiment run are shown on Figure 8. Horizontal axis represents different angles of out of plane rotation, and vertical bars represent the detection result for different classifiers. Note that false positives are removed from the results as they do not contribute to current coverage experiment.

It can be seen that classifiers cover almost [-66 62] region of the rotations leaving out extreme profile cases. The most important result is that between the range of images detected by frontal classifier and profile classifier there is no gap. Even more, the ranges covered by frontal view and profile classifiers are firmly intersected. Therefore, it can be claimed that by frontal classifier and by profile classifier the algorithm can cover all intermediate poses.

4.1.2 In-plane rotation

For the in-plane rotations, as mentioned before, the same frontal classifier was used, but on image rotated by 30 degree left or right. According to the authors [8], the original algorithm by Viola and Jones can cover up to ± 15 degrees of in-plane rotations. Therefore, by adding one 30 degree and one -30 degree detectors, the algorithm can cover ± 45 degrees of in plane rotation. Based on the application of the algorithm more rotations can be considered. However, for webcam applications ± 45 is sufficient.

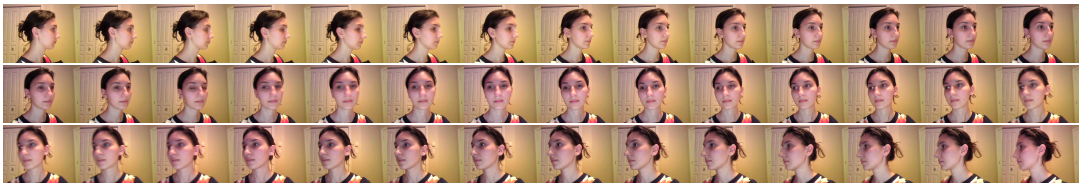


Figure 7: Head poses data set

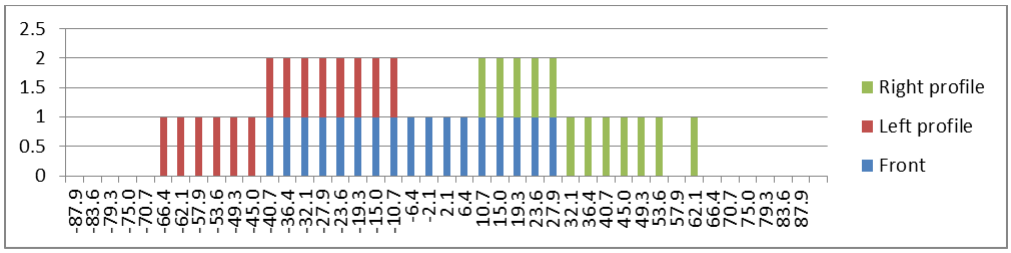


Figure 8: Head poses detection results

Experiment	Seconds per frame	Frames per second
Original frame, single thread	0.3	3.3
Sampled image, single thread (frontal faces only)	0.018	55.5
Sampled image, single thread (no faces)	0.071	14
Sampled image, multiple threads	0.1	10
Player system	0.12	8.3

Table 1: Time performance results

4.2 Time performance

Viola and Jones reported the performance of processing 15 frames per second [8]. In order to compare the performance of our system to Viola and Jones results, we performed a line of experiments. Experiments run on Intel Core i3-2310 CPU 2.10Ghz processor, and results can be seen in Table 1.

All cases, except of the last, report the performance of detection algorithm isolated from rest of the system. In other words, neither movie was played, nor detection face frame was rendered during the experiments. Last row in the table refers to the performance of the algorithm, integrated with the player and face detection rendering.

First row (original frame) refers to the performance results when our algorithm was applied on the frame, forwarded from the camera, without any pre-processing. As can be seen, this result is about 5 times slower than the result reported by Viola and Jones. However, it is still acceptable for the real time application.

Second and third rows (sampled image) refer to the performance results

when the detectors were invoked sequentially on the sampled image (as it appears on Figure 6(c)). In this scenario next classifier is envoked only if no face was detected by the previous detector. "Frontal faces only" condition stands for the best case scenario, when a single classifier for the frontal faces is always invoked. "No faces" condition stands for the worst case scenario, when all five classifiers are invoked. The best case, which is in fact equvalent to the experiments run by Viola and Jones, outperforms thier results in more than 3 times, whereas worst case scenario performance is comparable with thier results. This difference may be explained by more advanced processor used in our experiments. Unfortunately, Viola and Jones did not report processor details in thier paper [8], but considering the rapid development of processors industry, such explaination is more than probable.

Forth row refers to the performance of the algorithm when each detector runs in a separate thread. Note that in this case the performance of the algorithm is worse than the performance of a single thread worse scenario. This can be attributed to the following factors: overhead of threads switching, and post-processing steps required to merge multiple detections.

Last row in the table (player system) refers to the performance of entire system. It can be seen that even when integrated with the player, the system performs reasonably well in real time.

5 Discussion

During the last decade the algorithm presented by Viola and Jones [8] has become one of the most popular algorithms for face detection in real time. The main advantage of their approach is uncompetitive detection speed while relatively high detection precision, comparable to much slower algorithms. However, the approach also has some disadvantages such as exteremely long training time, limited head poses, and high rate of false positive detections. In the current project we overcame these difficulties by using already trained classifiers available in OpenCV library [3], using multi-view approach [6], and filtering detected faces by skin color [9].

Our system processes about 8 frames per second, which is almost twice less than the results reported by Viola and Jones [8]. However, considering that our system is integrated with media player and we use 5 different face detectors, whereas Viola and Jones used only one, the performance is reasonably well and acceptable for real time application.

Our systems does not make an assumption of a single user. It enables to detect multiple faces and track all users' attention. However, when single user assumption holds, the system performance may be improven even further by

applying techniques proposed by Ephraim et al. [5].

Our system makes an assumption about limited range of head poses, in particualar for in-plane rotations. However, this assumption is reasonable for the web camera application, where it is very unlikely to encounter, for example, rotation of 180 degrees, or even less extereme rotations.

Currently, our system handles all frames independently of each other. Obviously, since we have a stream of frames forwarded from the camera, these frames should be connected. In order to use this information properly, Markov Random Fields (MRF) may be used to model transtition between system states, i.e. faces presence and poses, based on the observatios, i.e. face detections by different classifiers.

Demonstration videos and other supporting materials for the current project can be found on the project website [4].

References

[1] Java Media Framework (JMF). `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html`.

[2] JavaCV. `http://code.google.com/p/javacv/`.

[3] OpenCV. `http://opencv.willowgarage.com/wiki/`.

[4] Project website. `http://cs.mcgill.ca/~gpetro6/comp558_project.html`.

[5] Theo Ephraim, Tristan Himmelman, and Kaleem Siddiqi. Real-time viola-jones face detection in a web browser. In *Proceedings of the 2009 Canadian Conference on Computer and Robot Vision*, CRV '09, pages 321–328, Washington, DC, USA, 2009. IEEE Computer Society.

[6] M Jones and P Viola. Fast multi-view face detection. *Mitsubishi Electric Research Lab TR2000396*, (July), 2003.

[7] J. Kovac, P. Peer, and F. Solina. Human skin color clustering for face detection. In *EUROCON 2003*, volume 2, pages 144–148, 2003.

[8] Paul Viola and Michael Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57:137–154, 2004.

[9] Shanshan Wang and Amr Abdel-Dayem. Improved viola-jones face detector. In *Proceedings of the 1st Taibah University International Conference on Computing and Information Technology*, ICCIT '12, pages 321–328, 2012.