Implementation of Object Detection Algorithm on Low Performance Embedded Systems

Karpov Alexey[*]
Moscow State University,
Department of mechanics and mechanics

Denis Ivanov[†]
Russian Systems Corporation

Abstract

Modern object detection algorithms easily achieve real-time performance without introducing quality tradeoff on todays desktop computers. However, many typical applications such as video surveillance, traffic management, various safety and aid systems require the use of embedded systems. This paper presents modifications to the algorithm by Viola and Jones [2001] and its analysis that allowed to achieve near real-time performance preserving detection accuracy rates on a real-life embedded system. The system is on test conducted on the Russian Railways locomotive.

Keywords: object detection, face detection, embedded system

1 INTRODUCTION

Effective in terms of time and quality implementation of various modern algorithms of computer vision require rather powerful hardware. Any modern desktop computer can suit well providing sufficient performance. At the same time real-life industrial computer system has to be not only computationally efficient but also capable to work in the harsh environment, rather small, energy-efficient and inexpensive (hence it has to have fewer components than a desktop). Usually such systems are up to 100 times slower than a modern desktop computer.

Object detection is one of the tasks that often has to be solved in the harsh environment with lack of space. This paper focuses on a real-life application of object detection – a face detection in the cabin of a vehicle. More specifically detection of the face of the train driver. This task is an important integral part of the more global problem – controlling over driver's behavior and action. The system constantly controls current state of the train and its driver. If something goes wrong (i.e. driver is missing, incorrect speed, etc.) the system will generate distress signal for the driver and contact traffic controller. Therefore the detection algorithm has to be rather accurate in order that the whole safety system be reliable.

The embedded system installed in the cabin of the train is based on a Texas Instruments TMS320VC33 chip. TMS320VC33 is 32-bit microprocessor running at 66 MHz with short pipeline. It has performance of 60 MIPS and nearly 120 MFLOPS. The size of the RAM is 1 megabyte. The aim was to achieve under one second processing time for the detection algorithm.

[*]karpov@fit.com.ru
[†]denis@rusys.ru

2 BOOSTED CASCADE ALGORITHM

We have chosen boosted cascade of simple features object detection algorithm as the base algorithm [Lienhart et al. 2003] [Viola and Jones 2001] and its implementation from OpenCV[1]. The algorithm uses a cascade of small Haar-like feature based classifiers each of them trained to detect the specified objects and to reject a significant fraction of the other objects. The input image is partitioned into many overlapping subwindows of different sizes. The algorithm classifies each subwindow into either representing the object or not.

The advantages of the algorithm are the following. On the one hand, relative computational simplicity, independence of time required to classify one subwindow from its size, overall robustness. On the other, its flexibility residing on a transparent parametrization and control of the quality and speed.

The algorithm easily achieved more than 25 fps on a desktop requiring 2 megabytes of the RAM[2] to run. Hence, the first obstacle we encountered in implementation of the algorithm on the embedded system was memory consumption. The problem became more complicated since the minimum size of a data type for the TMS320VC33 chip was 4 bytes. The difficulties were overcome using the following ways. The structure of the cascade representation was refined, data types were changed to smaller (and the correctness of the algorithm was confirmed), 8 bit images as well as auxiliary data were packed to 4 bytes blocks. As a result memory consumption on the embedded system decreased to 600 kilobytes.

However, time required for the algorithm to process single image with the same parameters as on a desktop was unacceptable. More specifically up to 6 seconds per frame. In order to increase processing speed the algorithm was modified and analyzed.

2.1 Algorithm Modifications

The algorithm was modified to solve the more certain task. In fact, it was required to determine the presence of at least one object (face) on the image. Therefore, it made possible to decrease the number of subwindows for classification. More specifically, as soon as a face was found (a subwindow classified as a face) algorithm stopped. However, such approach led to more false detections. In to order to reduce this negative effect several additional tests were performed. The subwindow was moved by 1 pixel from the initial position in different directions and classified. We have found out that 2 more detections was enough to confirm that the original detection was reliable.

Then it was discovered that "big faces" occurred more often than "small" . That is, most objects tended to be bigger in size than the minimum that could be detected. Thus, subwindows scanning

[1]http://opencvlibrary.sourceforge.net/
[2]on 160 by 120 pixels images

was modified to begin with bigger subwindows rather than smallest possible.

These modifications allowed to increase the performance of the algorithm up to 5 times on the images with faces. On the average performance increased up to 1.5-2 times depending on the parameters on our real-life cabin image sequence. See Figure 1 for the average time per image on a desktop.

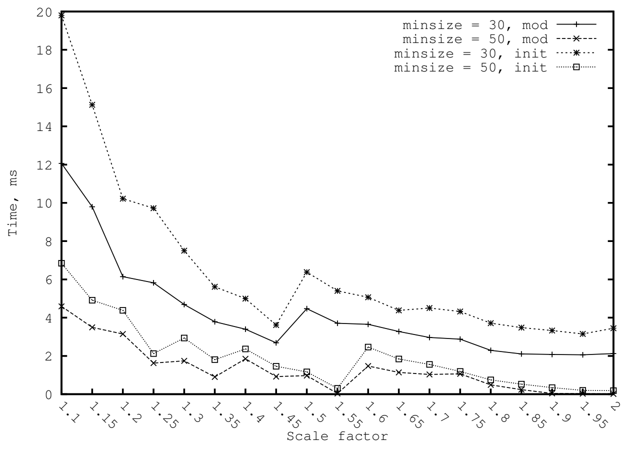

Figure 1: *Average processing time on a desktop before and after modifications*

2.2 Algorithm Analysis

The algorithm has a set of implicit and explicit parameters which control its quality and performance. The following analysis was done based on real-life image sequence (see Figure 2).

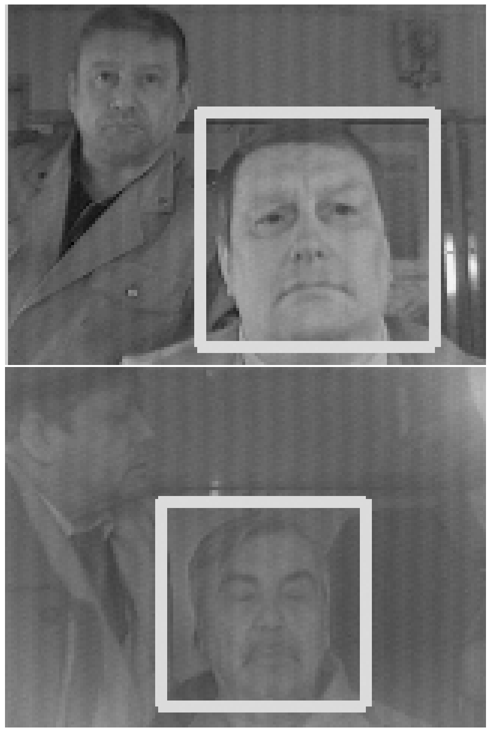

Figure 2: *Example images from real-life sequence*

At first, let us consider the following implicit parameter – number of stages in the cascade of classifiers, i.e. how many simple classifiers cascade contains. Obviously the more stages contains cascade the more time and memory is required to process a single subwindow. Two cascades were considered. First one had 20 stages, the second – 22 stages. As a result we have chosen cascade with 20 stages, since quality difference was insignificant, but memory consumption for the 20 stages cascade was 30 kilobytes less. The processing time difference was no more than 5% on the average.

The next implicit parameter is the minimum possible subwindow size that the cascade can process. It was fixed at 20 by 20 pixels. The parameters mentioned above can be changed only on the cascade training stage.

Another implicit parameter is the image size. It was fixed at 160 by 120 pixels. Initially, it was supposed that the size of the image would be 320 by 240 pixels, but we had to reduce it using simple interpolation due to severe memory amount restrictions. The processing speed increased thanks to the fact, but also we had to reduce minimum subwindow size as well as scale factor to maintain the quality. Overall, we believe that the fact did not affect the quality of the detection.

Let us consider the following explicit parameters:

- Minimum subwindow size. Defines a minimum subwindow that will be processed.
- Scale factor. Defines a factor used to scale minimum possible subwindow (defined in the cascade).

The following formula was derived in order to estimate processing speed of the algorithm:

$$\sum_{p=F_1}^{F_2} X(p)Y(p),$$

$$F_1 = \left\lceil \log_s \frac{w}{o_w} \right\rceil = \left\lceil \log_s \frac{h}{o_h} \right\rceil,$$

$$F_2 = \left\lfloor \min\left(\log_s \frac{I_w - 10}{o_w}, \log_s \frac{I_h - 10}{o_h} \right) \right\rfloor,$$

$$X(p) = \left[\frac{I_w - o_w s^p}{\max(2, s^p)} \right],\ Y(p) = \left[\frac{I_h - o_h s^p}{\max(2, s^p)} \right],$$

where $\lceil\rceil$ is ceiling of a number, $\lfloor\rfloor$ – flooring of a number, $[]$ – rounding a number to the closest integer, o_w, o_h – minimum possible width and height of a subwindow defined in the cascade, I_w, I_h– width and height of the image, w, h – explicitly specified minimum width and height of a subwindow, s – scale factor. The formula shows the number of subwindows that will be processed during detection. However, it did not estimate well processing time of the algorithm, since it did not take into account additional calculations performed when the subwindow was scaled. When the subwindow is scaled the cascade classifiers are scaled accordingly. Therefore the formula was slightly changed:

$$\sum_{p=F_1}^{F_2} X(p)Y(p) + C(F_2 - F_1).$$

Here the constant C scales time required to do additional calculations into "subwindows number" measure. We have found from experimental results that $C = 1137$. This formula estimates well the processing time of the algorithm. But it is fair to say that still the formula is a estimation on the average. It estimates only a simple case. By simple case we mean the following. The modifications from 2.1 are not taken into account, every subwindow is considered to be processed only once, since the common version of the algorithm uses first walk through subwindows to eliminate the non-objects using only few first classifiers of the cascade. The second pass is used by the algorithm to process only remaining subwindows with the whole cascade. Nevertheless, high correlation occurs between the average processing time and the calculation result of the formula. See normalized results in comparison on Figure 3.

Since we have obtained such high correlation it is possible to predict performance of the algorithm with a set of parameters. Figure 4 shows the result of calculation for the formula for a wider set of parameters.

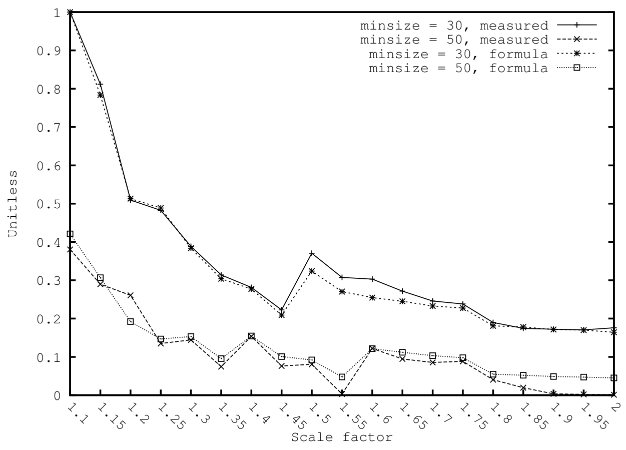

Figure 3: *Average processing time vs calculated time normalized*

One can see that minimum subwindow size greatly affects the processing time. Decreasing minimum subwindow height and width for 10 pixels lead to 50% and more increase of the processing speed in the most cases. Scale factor also has strong influence on the processing time. At first the processing time decreases rapidly, but after approximately factor 1.3 decreasing slows down, almost stabilizing at the certain level in the end. There is an interesting effect of increasing the processing time while increasing the scale factor. The fact is that sometimes increasing the scale factor gives the more possible subwindow sizes, therefore adding number of them.

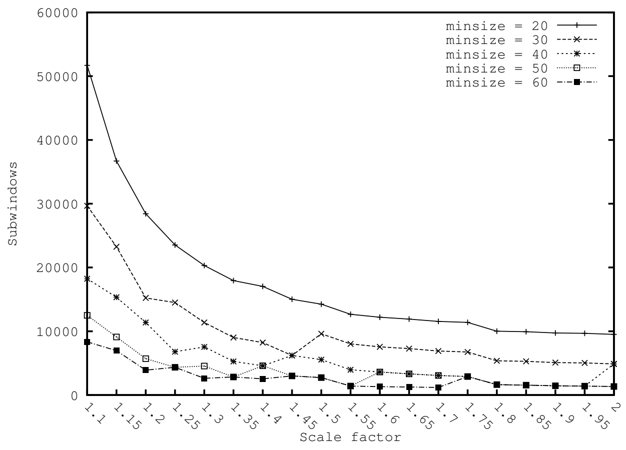

Figure 4: *The formula calculation result*

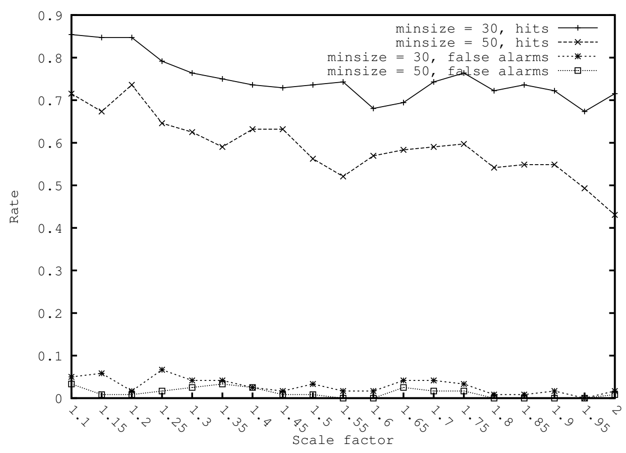

Figure 5: *Hit vs false alarm rates*

Let us discuss how parameters affect the quality of the detection results. We have focused on the cases when the minimum height and width of a subwindow equals to 30 and 50 pixels. The choice of these parameters came from the real-life situation we had come up against. At first it was required to detect faces with height and width of at least 50 pixels. Later the requirements changed due to widen field of view of the camera to more severe – height and width had to be at least 30 pixels. Simple changing from the minimum size of subwindow from 50 to 30 pixels without changing the scale factor led to a dramatic (more than 3 times) drop in the time performance. From nearly half a second per frame to an unacceptable level of two seconds per frame. One of the ways to reduce processing time was increasing the scale factor. However we had to know how it would affect detection quality.

We have measured quality of the detection on our real life image sequence (see Figure 5). At first, let us consider hit and false alarm rates for the different minimum subwindow sizes. One can see that hit rates decrease up to 20% percent when the minimum subwindow size is increased from 30 to 50 pixels, but false alarms rates decrease not too much in the most cases. The sequence contains lots of rather big faces taken before widening the camera's field of view. So we expect bigger difference in hit rates between 30 and 50 pixel minimum subwindow size cases.

Now let us consider scale factor influence. One can see that hits rates generally decline while increasing the scale factor. However, that decrease is not so fast compared to the decrease of the average processing time (see Figure 3). False alarm rates also tend to decrease.

We now turn back to the question of maintaining the same performance of the detection algorithm after we had to decrease the minimum subwindow size from 50 to 30. One can see, that setting the minimum size to 30 and the scale factor to 1.85 makes the algorithm to detect better and while being a bit slower than the initial choice nevertheless achieve under one second performance (see Table 1). For our purposes it is very important to have small number of false alarms. Because it is better to miss detection of the train driver than detect him when one is absent.

Parameters	minsize = 50, scale = 1.25	minsize=30, scale = 1.25	minsize = 30, scale = 1.85
number of subwindows	4353	14495	5283
hit rate	0.65	0.79	0.74
false alarm rate	0.02	0.07	0.01

Table 1: *Performance comparison for the different parameters*

3 FUTURE WORK AND CONCLUSION

Although we have achieved under one second performance, we will try to further decrease the processing time in order to use more aggressive parameters. This can be done using many ways. On of the most evident and still not exploited ways can be derived from the fact that we detect faces in a video stream. Hence, we can further decrease the number of subwindows by scanning only changed image area.

All of the techniques mentioned above focused on increasing the processing speed when the object is presented on the image. Therefore it would be useful to develop a quick way to detect images that clearly does not contain the object.

Another aim is derive more precise formula for the processing time estimation.

The paper has presented techniques that allowed to achieve near real-time performance of the boosted cascade object detection algorithm on a computer system with a 120 MFLOPS performance rating. Using thorough analysis of the detection method we have succeeded to preserve high detection accuracy.

The method have been implemented on the embedded system installed on the Russian Railways train. Currently testing of the system is conducted. It will be an important part of driver's action control system. The system is in turn part of the more global railway safety system.

References

LIENHART, R., KURANOV, A., AND PISAREVSKY, V. 2003. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *"DAGM-Symposium"*, "297–304".

VIOLA, P., AND JONES, M. 2001. Rapid object detection using a boosted cascade of simple features.