RASW: a Run-time Adaptive Sliding Window to Improve Viola-Jones Object Detection

Francesco Comaschi, Sander Stuijk, Twan Basten and Henk Corporaal
Electronic Systems Group, Eindhoven University of Technology, The Netherlands
{f.comaschi, s.stuijk, a.a.basten, h.corporaal}@tue.nl

***Abstract*—In recent years accurate algorithms for detecting objects in images have been developed. Among these algorithms, the object detection scheme proposed by Viola and Jones gained great popularity, especially after the release of high-quality face classifiers by the OpenCV group. However, as any other sliding-window based object detector, it is affected by a strong increase in the computational cost as the size of the scene grows. Especially in real-time applications, a search strategy based on a sliding window can be computationally too expensive. In this paper, we propose an efficient approach to adapt at run time the sliding window step size in order to speed-up the detection task without compromising the accuracy. We demonstrate the effectiveness of the proposed Run-time Adaptive Sliding Window (RASW) in improving the performance of Viola-Jones object detection by providing better throughput-accuracy tradeoffs. When comparing our approach with the OpenCV face detection implementation, we obtain up to 2.03x speedup in frames per second without any loss in accuracy.**

I. Introduction

Innovations in semiconductor technology and computer architectures are enabling new scenarios in the context of real-time computer vision systems where the capability of analyzing the content of a scene in real time is of crucial importance. A good example of a time-critical task which finds application in many fields, from video-surveillance to robot cameras, is the detection and identification of a specific object in a scene. Among the objects of possible interest, human faces have recently received significant attention in both academia and industry [1]. In the available literature, a wide variety of techniques have been proposed for face detection. The face-detection technique proposed by Viola and Jones in [2] represented a real break-through in this research area. Since its introduction, the Viola-Jones algorithm gained great popularity due to its high accuracy and solid theoretical basis. However the computational complexity of the Viola-Jones algorithm makes it still a challenging task to meet real-time requirements (e.g., throughput) even on powerful platforms. The scene-scanning speed heavily depends on the sliding-window step size Δ, which is the number of pixels by which the sliding window is shifted when scanning the image in search of objects. In most available implementations the step size Δ is constant and fixed at compile time; this implies that the sliding window is shifted by a constant amount of pixels regardless of the scene content. In this paper, we propose the *Run-time Adaptive Sliding Window* (RASW) approach which significantly improves the throughput without degrading the accuracy of the algorithm by adapting the sliding-window step size Δ at run time.

The paper is structured as follows. In the next section, we give an overview of the related work. In Sec. III, we review the Viola-Jones algorithm. In Sec. IV, the proposed RASW approach is presented. Experimental results are provided in Sec. V while Sec. VI contains the concluding remarks.

This paper is a slightly revised version of the paper with the same title published in *Proc. Seventh ACM/IEEE International Conference on Distributed Smart Cameras* (ICDSC 2013).

II. Related Work

Many hardware implementations of the Viola-Jones face detector have been proposed in recent years to obtain a high throughput [3]–[5]. However, a custom hardware design presents two main drawbacks: i) it requires a significant engineering effort, which also implies high non-recurring costs; ii) it does not provide flexibility, making it almost impossible to adapt the system to any changes in the application scenario.

Another possible solution to speed-up the algorithm is resorting to a GPU implementation [6]–[8]. However, the amount of power consumed by a GPU makes it an unpractical solution if low power is a concern, such as for embedded applications. Moreover, GPU implementations can be used in combination with the optimization proposed in this paper. In particular, features- and scale-parallelization are possible. For a more detailed discussion readers are referred to [6].

In [9] the authors proposed an OpenCV software implementation optimized for embedded environments, but the algorithm is left unaltered with respect to the original OpenCV implementation. Our algorithmic optimization improves the system throughput by reducing the required computation without degrading the system accuracy.

The first step in a typical object detection system is the scanning of the scene in search of candidate image regions. This scanning step is computationally intensive. RASW is a new approach in the scene-scanning process that exploits run-time information to speed-up the detection; therefore it relates also to recent works on optimal image search. Lampert et al. [10] propose an analytic approach known as Efficient Subwindow Search (ESS) to allow efficient object localization. However, the proposed approach refers to object localization, which implies that the number of objects of interest present in the scene must be known in advance. Such an approach can be extremely useful in contexts like image retrieval from the web, but from [10] it is not clear which performance can be expected in contexts where many objects of interest are simultaneously present in the scene. Moreover, in [10] a bounding function has to be built in order to apply ESS to an object detector, and in that work there is no example of such a function for a cascade classifier. The approach proposed in this paper is specifically targeted for cascade classifiers, does not require any additional information such as a bounding function and applies for any number of objects of interest in the scene.

In [11] the authors consider a method based on a model of visual search in humans for fast object detection. Comparing their work to the OpenCV implementation of the Viola-Jones face detector, they obtain an average 2x speedup, which comes

at the cost of a small loss in accuracy. The experimental results are reported for a single image, so it is difficult to understand if the reported results still hold when testing the proposed approach in a more extensive way. Moreover, the approach proposed by the authors in [11] applies to images containing one single face, and it is not clear how to apply such an approach to images with multiple instances of objects of interest. Even if some results are reported for different choices of the scaling factor s, the other parameters involved in the scene-scanning process are not systematically explored. The approach introduced in this paper can always reach a better accuracy than the OpenCV implementation at the same time speeding-up the detection task. Our results are presented for a standard database of faces and the influence of the different parameters involved is carefully analyzed.

In [12] the authors propose an interesting method to reduce the missed detections in a cascade face detector while increasing the step size in a sliding-window based technique. The authors present very promising results on several face databases. However, the approach proposed in [12] requires the off-line training of a decision tree which is adopted for location estimation. Also, the use of interests points adds some extra computation time to the detection task, thus making it more difficult to apply it in case of memory or processing power limitations. The algorithmic optimization presented in this paper is solely based on run-time information and it is obtained by performing the scene scanning in a more computationally efficient manner; therefore it does not require any extra training on the object classifier and does not add any computational overhead to the standard sliding-window techniques.

III. Viola-Jones Algorithm Overview

The Viola-Jones algorithm is used as a starting point in this work; therefore we present an overview of the algorithm describing its main steps. However, it should be noted that the proposed optimization may benefit every sliding-window based object detection algorithm adopting a cascade classifier. Since the proposed optimization relates to the scene-scanning process, without affecting the computations performed on each of the sliding windows, we focus in our discussion on the scene-scanning process and the merging of multiple detections. We also provide a brief description of the parameters influencing the run-time behavior of the algorithm and we briefly describe the working principles of the cascade classifier.

Scanning the detector: In Fig. 1 we provide a graphical representation of the scene-scanning process: in steps 1-4 a sliding window of fixed size is scanned across the complete image (left-to-right-top-to-bottom) according to a step size Δ. Each sliding window is provided as input to the cascade classifier (see Fig. 2, explained later). In step 5 the image is scaled-down according to a scale factor s to detect faces of various sizes. In step 6 the scanning is repeated on the scaled-down image. In step 7, when the image is smaller than the sliding window, the detection ends. In steps 8-9, multiple detections are merged using the ratio of the intersection/union of the detection areas.

Merging multiple detections: Following a standard implementation [12], [13], we combine multiple detections (of the same face/region) in he following fashion. The set of detections is first partitioned into disjoint subsets, where two

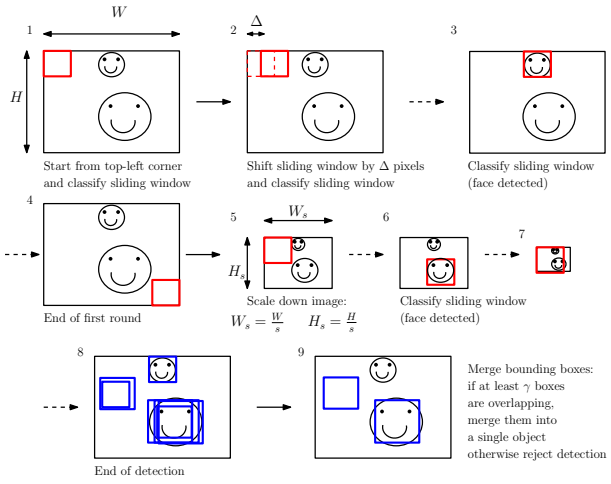


Fig. 1. Scene-scanning.

different bounding boxes are in the same subset if their overlap ratio is above a threshold α. If any detection is defined by a rectangular bounding box $R_i = (x_i, y_i, w_i, h_i)$, where x_i and y_i are the top left corner of the bounding box and w_i and h_i are its width and height respectively, then the overlap ratio between any two rectangles R_i and R_j is given by:

$$J(R_i, R_j) = \frac{R_i \cap R_j}{R_i \cup R_j} \tag{1}$$

where $R_i \cap R_j$ and $R_i \cup R_j$ stand for the area of their intersection and union respectively. The value of J ranges from 0 to 1 and we have experimentally determined a value of 0.45 for α. Once the entire set of detections has been divided into subsets of overlapping rectangles, we merge each subset of overlapping rectangles into a single rectangle (bounding box), where the corners of the final bounding box are the average of the corners of all detections in the subset. Since targets (and also false positives) are often detected multiple times, it is useful to introduce a configurable merging threshold γ which is defined as the minimum number of overlapping rectangles in each disjoint subset for accepting the merged detection. When a subset of overlapped rectangles contains less detections (rectangles) than γ, the subset will be rejected, i.e., the merging step reverts the decision of the cascade classifier and classifies the region as a region not containing a face. In steps 8-9 of Fig. 1 we provide an example where $\gamma = 2$ leads to a true positive (face correctly detected), a false positive (bounding box at output of merging step that contains no face) and a false negative (face not detected).

Cascade detector: To achieve rapid detection, Viola and Jones proposed to perform the detection task through a cascade of nodes, which are referred to as *strong classifiers*. The cascade structure is shown in Fig. 2. The cascade consists of n stages, which for our purposes can be seen as black boxes performing some computation on the input sliding window. Each stage is characterized by a stage-threshold and each stage computation results in a binary decision: if a certain condition is satisfied, then the sliding window is passed to the next stage for further processing; otherwise it is immediately rejected. Only sliding windows that can make it through the complete cascade are classified as faces. Since the computation

performed within each stage is quite costly, the structure of the cascade reflects an attempt to reject the majority of the sliding windows in the input as soon as possible, which reduces the amount of computation required. The parameters of the classifier have been obtained by training them on a set of positive and negative samples of fixed size. Since our focus is not on the training process, we have used a classifier already available from the OpenCV library [14]. This cascade classifier consists of $n = 25$ stages trained over sample images of 24 $\times$ 24 pixels.

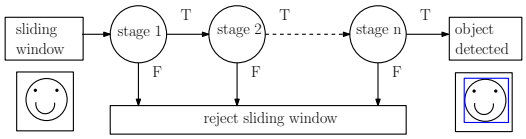


Fig. 2. Cascade classifier composed by n stages. Each stage results in a binary decision. If a stage returns false, the sliding window is rejected immediately and no further processing is required.

IV. Run-time Adaptive Sliding Window (RASW)

In Sec. III, we have seen that at each scaling of the original image a sliding window is scanned across the complete (scaled) image according to a step size Δ. For our purposes, we should distinguish between the step size along the x and the y axis, i.e. Δ_x and Δ_y. For sake of simplicity, we keep the Δ notation whenever a statement is valid for both directions. The sliding-window step size Δ affects both the detection accuracy and the system throughput. Generally speaking a higher value of Δ improves the system throughput, since a smaller number of detection sliding windows will have to pass through the cascade classifier for the detection task. On the other hand, this usually results in a worsening of the recall (i.e., the fraction of detected objects that are relevant to the search over the number of objects of interest present in the scene), since some objects of interest might be missed when moving the sliding window too fast. If we find a criterion to move the detection sliding window faster when scanning regions that do not contain any objects of interest, and move it slower in the vicinity of faces, we are able to speed-up the scene-scanning process without affecting the recall. Moreover, by moving the sliding window faster in uniform regions of the image, we reduce the chance for the classifier to wrongly classify a background region as a face, thus improving the precision (i.e., the fraction of detected objects that are relevant to the search over the total number of objects retrieved by the classifier).

In most existing implementations, the step size is fixed to a constant value, usually $\Delta_x = \Delta_y = 1$ or $\Delta_x = \Delta_y = 2$ since higher values would normally provide poor results in terms of recall. Our analysis reveals a correlation between the presence of faces in regions of the image and the stage of the cascade classifier where the sliding window is rejected (from now on referred to as the *exit-stage*). In particular, in background regions, the detection sliding windows are rejected in the earlier stages of the classifier, while in general the higher the proximity of faces, the higher the *exit-stage*. Fig. 3 shows an example image from the test set and for each of the sliding windows scanned across the image the corresponding *exit-stage* from the cascade classifier. If the top-left corner of the current sliding window is at position (x, y) of the image, the pixel in position (x, y) is displayed with an intensity which is inversely proportional to the *exit-stage*. In other words, the darker the pixel, the higher the *exit-stage*.

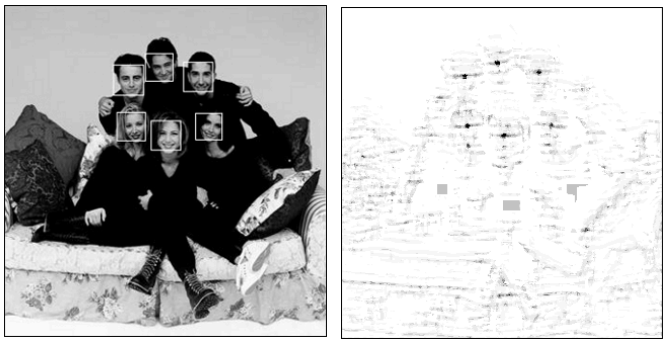


Fig. 3. The left plot is an example image from the test set. The right plot shows the correlation between regions of the image and the *exit-stage* from the cascade classifier when the sliding window passes over that region. The analysis suggests that the sliding window can be safely shifted by a bigger step in correspondence to low *exit-stage* values, without missing any faces

Our analysis suggests that the sliding window can be safely shifted by a bigger step in correspondence to low *exit-stage* values, since in most cases this implies that the current region of the image does not present any objects of interest. On the other hand, as the *exit-stage* value approaches the end of the cascade classifier, it is wise to slow-down the scanning process in order not to miss any faces, because a higher value of the *exit-stage* corresponds to a higher probability of being in the proximity of objects of interest. The basic idea behind the standard cascade classifier is to spend less time on image regions that are less likely to contain objects of interest. The proposed RASW approach directly discards non-promising regions by exploiting spatial data locality. Even though the discarded regions are those where computation would have been less expensive, the considerable reduction in the number of detection windows leads to a faster computation without affecting the recall. This is especially true because by directly discarding a window, we also avoid image normalization of the subwindow, which is a rather expensive pre-processing operation required by the Viola-Jones algorithm in order to minimize the effect of different lighting conditions [2], [9].

In Fig. 4 we provide a graphical representation of the intuition behind the proposed approach. In order to make the figures more readable, we distinguish between the x (left) and the y direction (right). The gray-scale blocks are a zoomed-in view of pixels from 204 to 213 (x-axis) and from 114 to 122 (y-axis) of the image reported in Fig. 3 (right). The black pixel at position (209, 117) corresponds to the bottom-right face detected in Fig. 3. For each of the approaches reported in Fig. 4, the corresponding geometric shape specifies the pixels where the sliding window is placed. In Fig. 4 (left) we can see how, by taking a constant step of $\Delta_x = 2$ or $\Delta_x = 3$, the detector fails to place the sliding window at the location where the face is present. On the contrary, the RASW approach automatically reduces the step size Δ_x as the sliding window approaches the face, thus allowing to place the sliding window properly. The same behavior can be observed in Fig. 4 (right), the only difference being that in this case also a constant step $\Delta_y = 3$ allows to place the sliding window properly.

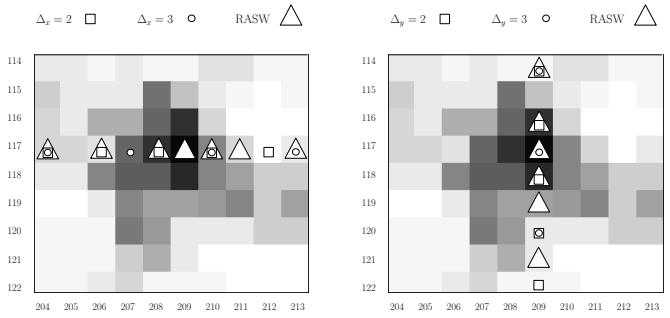


Fig. 4. Graphical representation of the proposed RASW method. To make the figure more readable, we distinguish between the x (left) and the y (right) direction. The squares and circles represent the position where the sliding window is placed by taking a step $\Delta = 2$ and $\Delta = 3$, respectively. The triangles represent the position where the sliding window is placed by the RASW approach. From the figure we can see how a dynamic adaptation of the step size along both directions allows to speed up the scene scanning in background regions while allowing to properly detect faces.

Algorithm 1 Run-Time Adaptive Sliding Window

Require: scaled input image X, classifier cascade S
Ensure: vector V of detected faces (bounding boxes)
1: **for** $y \leftarrow 0$ to $X.height - subwindow.height$ **do**
2: $\quad\Delta_x \leftarrow 1$
3: $\quad$**for** $x \leftarrow 0$ to $X.width - subwindow.width$ **do**
4: $\quad\quad\Delta_x \leftarrow \Delta_x - 1$
5: $\quad\quad\Delta_y[x] \leftarrow \Delta_y[x] - 1$
6: $\quad\quad$**if** $\Delta_x = 0$ AND $\Delta_y[x] = 0$ **then**
7: $\quad\quad\quad$*exit-stage* $\leftarrow S(x, y)$
8: $\quad\quad\quad$**if** *exit-stage* $= n$ **then**
9: $\quad\quad\quad\quad R \leftarrow (x, y, width, height)$
10: $\quad\quad\quad\quad$push R into V
11: $\quad\quad\quad$**end if**
12: $\quad\quad\quad$**if** *exit-stage* $< \Delta_{x,t1}$ **then**
13: $\quad\quad\quad\quad\Delta_x = \Delta_{x,max}$
14: $\quad\quad\quad$**else if** $\Delta_{x,t1} \leq$ *exit-stage* $< \Delta_{x,t2}$ **then**
15: $\quad\quad\quad\quad\Delta_x = \Delta_{x,nom}$
16: $\quad\quad\quad$**else**
17: $\quad\quad\quad\quad\Delta_x = \Delta_{x,min}$
18: $\quad\quad\quad$**end if**
19: $\quad\quad$**else if** $\Delta_x = 0$ **then**
20: $\quad\quad\quad\Delta_x \leftarrow \Delta_{x,min}$
21: $\quad\quad$**else if** $\Delta_y[x] = 0$ **then**
22: $\quad\quad\quad\Delta_y[x] \leftarrow \Delta_{y,min}$
23: $\quad\quad$**end if**
24: $\quad$**end for**
25: **end for**

Algorithm 1 presents the implementation of RASW. Every time we run the cascade classifier on a sliding window (line 7 of algorithm 1), the stage where the window has been rejected is returned. A face is detected when *exit-stage* $= n$. Whenever a face is detected (line 8), a rectangular bounding box R is created and pushed into the vector of detected faces V (line 10). The sliding window step size Δ_x can switch between three different values: $\Delta_{x,max}, \Delta_{x,nom}$ and $\Delta_{x,min}$, depending on the *exit-stage* from the cascade classifier of the latest evaluated sliding window (the same holds for the y direction). In order to assign a proper value to Δ_x and Δ_y, we need to store the information related to the result of the latest evaluated detection sliding window. Such information can be directly stored in the step sizes Δ_x and Δ_y. In our implementation the sliding window is scanned in a left-to-right-top-to-bottom fashion; therefore the information related to the x-direction can be stored in one single short integer variable (Δ_x). On the other side, every time a sliding window reaches the end of a row, the information related to the step size across the y-direction must be preserved for the entire row; therefore Δ_y is an array of short integers whose size is the width of the current scaled-down image X. At each iteration of the algorithm, both Δ_x and $\Delta_y[x]$ are reduced by one. Only when both Δ_x and $\Delta_y[x]$ reach zero the current sliding window is evaluated. If the condition is met by only one of the two step sizes, then such step size is reinitialized to the minimum (lines 20 and 22). The transition from one value of Δ to another is determined for both directions at run time based on four threshold parameters: $\Delta_{x,t1}$, $\Delta_{x,t2}$, $\Delta_{y,t1}$ and $\Delta_{y,t2}$. These parameters are set at compile time to one of the stages of the cascade classifier; therefore they can assume the integer values in the range $[0;\ n+1]$, with $\Delta_{x,t1} \leq \Delta_{x,t2}$ and $\Delta_{y,t1} \leq \Delta_{y,t2}$. We include the corner cases where Δ_x is constant and equal to either $\Delta_{x,min}$ ($\Delta_{x,t1} = \Delta_{x,t2} = 0$), $\Delta_{x,nom}$ ($\Delta_{x,t1} = 0$, $\Delta_{x,t2} = n+1$) or $\Delta_{x,max}$ ($\Delta_{x,t1} = n+1$). The same holds for Δ_y. For the sake of brevity, in lines 12-18 of algorithm 1 we report only the code related to the x direction, but the same code should be reported for the y direction, although with some variations. Different choices for the threshold parameters provide several accuracy-throughput tradeoffs. Depending on the performance requirements of the application, by properly tuning the thresholds together with the other parameters involved in the scene-scanning process (scale factor s and merging threshold γ), different design points in the accuracy-throughput space can be reached. For the values of $\Delta_{max}, \Delta_{nom}$ and Δ_{min} we chose 3, 2 and 1 respectively (for both directions), since in our experiments higher values have proven to provide poor recall results (below 70%).

In the next section we show that, by assigning different values to the threshold parameters, the proposed RASW approach provides new configurations in the scene-scanning design space which dominate the configurations provided by the existing implementations.

V. Experimental Results

In this section we demonstrate the effectiveness of RASW in improving the performance of Viola-Jones object detection by comparing it with an implementation that assigns a static value to Δ_x and Δ_y, and to the two approaches adopted in the OpenCV 2.4 library, which implement simpler mechanisms of run-time adaptation. The experiments have been run on a quad-core Intel Core i7 with a 3.07 GHz clock featuring an 8.00 MB level-3 cache. The selected test set is the CMU+MIT face database set [15].

The approaches that we have taken into account are the following: (i) static: Δ_x and Δ_y constant and equal to either 1, 2 or 3 (therefore leading to nine possible combinations); (ii) OpenCV 1: $\Delta_x = 2$ for every region of the image, unless the previous sliding window has detected a face. In the latter case, the step size Δ_x is reduced to 1. Δ_y is kept constant and equal to 1; (iii) OpenCV 2: $\Delta = 1$ if the sizes of the current scaled-down image are less than half of the original image sizes, $\Delta = 2$ otherwise. In this case a change in the step size affects both Δ_x and Δ_y.

We remark here that the first 2 approaches (static and OpenCV 1) used for our comparison are sub-cases of the pro-

posed RASW approach. In particular, every combination of the static approach can be derived from Algorithm 1 by assigning proper values to Δ_{t1} and Δ_{t2} (as already mentioned in the previous section). The second approach (OpenCV 1) can be obtained by setting $\Delta_{x,t1} = 0$, $\Delta_{x,t2} = n$, $\Delta_{y,t1} = \Delta_{y,t2} = 0$.

In order to illustrate the advantages achievable through RASW, we tested the different approaches for several values of the two other scene-scanning parameters: the scaling-factor s and the merging threshold γ. In particular we have chosen the following settings: $s \in \{1.1, 1.2, 1.3, 1.4, 1.5\}$ and $\gamma \in \{1, 2, 3, 4, 5\}$. Larger values for any of these parameters provided poor results in terms of recall. Having five possible values for s, five different values for γ and eleven different approaches on Δ (nine static combinations plus the two OpenCV approaches), we end up with 275 different configurations for the existing approaches. Regarding RASW, different choices are possible for the threshold parameters $\Delta_{x,t1}$, $\Delta_{x,t2}$, $\Delta_{y,t1}$ and $\Delta_{y,t2}$. Since the number of stages of the selected classifier is $n = 25$, and in order to perform an extensive testing of our method in reasonable time we pruned the search space by assigning to the threshold parameters a limited set of values in the range $[0; n + 1]$. In particular $\Delta_{x,t1}, \Delta_{x,t2}, \Delta_{y,t1}, \Delta_{y,t2} \in \{0, 5, 10, 15, 20, 25, 26\}$, with the constraints $\Delta_{x,t1} \leq \Delta_{x,t2}$ and $\Delta_{y,t1} \leq \Delta_{y,t2}$, which leads to 28 possible combinations for each direction. Multiplying by the possible values for s and γ we end up with a total of 19,600 configurations.

To analyze the performance of the proposed algorithm, we use two metrics often adopted when comparing object detection algorithms (in particular these are the same metrics adopted in the Pascal VOC challenge [16]). These two metrics are the $recall$ and the $precision$, and they are defined as:

$$recall = \frac{TP}{TP + FN}, \quad precision = \frac{TP}{TP + FP} \tag{2}$$

where TP is the number of *true positives*, FN is the number of *false negatives* while FP is the number of *false positives*. Hereby we want to show that the proposed RASW approach provides the designer with new design points which dominate the existing solutions. Among all the generated configurations, we show only the so-called *Pareto points*, which are locally optimal solutions that capture all the optimal tradeoffs in the design space[1] [17]. A configuration in the design space is said to dominate another configuration if it is at least as good in each of the adopted metrics. Typically, we want to remove elements that do not contribute interesting realization options. A configuration that is dominated by another one is not interesting. A set of configurations that does not contain any dominated configuration, is said to be *Pareto minimal*. Thus, all individual elements of a Pareto-minimal set of configurations are Pareto points.

In Fig. 5(a) we report the Pareto points obtained through the baseline (static, OpenCV 1, OpenCV 2) and the RASW approaches in the 2-dimensional space $recall/precision$. The Pareto-minimal configuration set from the baseline implementations contains 27 Pareto points while the Pareto-minimal configuration set from the RASW approach contains 48 Pareto points. From Fig. 5(a) we can see that all the Pareto-minimal configurations from the baseline design space are dominated by points from the RASW design space. This implies that for every colored cross or x-mark (baseline configuration) in Fig. 5(a) we can find at least one red circle (RASW configuration) that is at least as good in each of the adopted metrics. In practice, for a given recall, the RASW approach always provides at least one configuration point in the scene-scanning parameters design space which is at least as precise as any configuration point from the baseline implementations with the same (or better) recall.

Especially for embedded applications, where the images have to be processed under stringent real-time constraints, a third aspect is of paramount importance: *throughput*, which can be defined as the number of processed Frames Per Second (*fps*). In order to be able to visualize the Pareto-minimal solutions when taking also the system throughput into account, we adopt the *F1-score*, which is a widely adopted metric in the context of information retrieval and accounts for the system accuracy by combining precision and recall together [18]. The *F1-score* can be interpreted as a weighted average of the precision and the recall, and it reaches its best value at 1 and worst score at 0:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3}$$

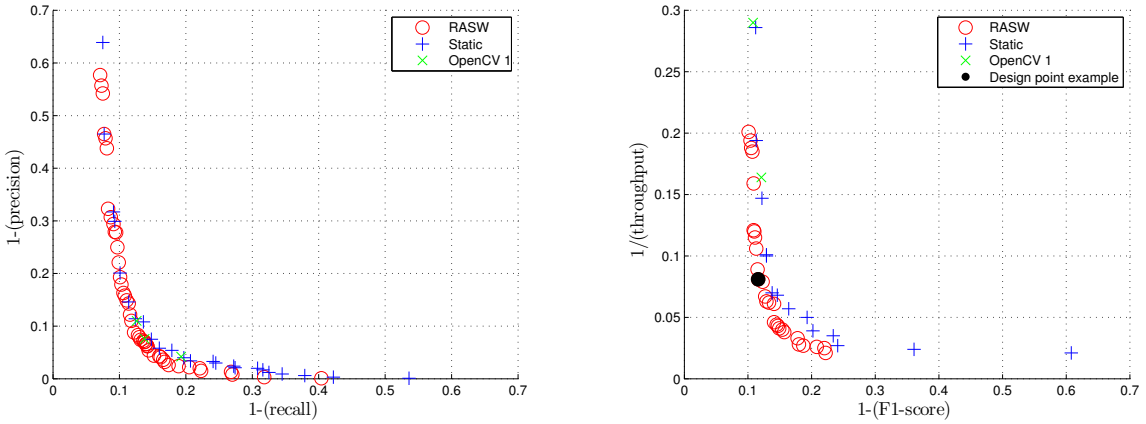
In Fig. 5(b) we report the Pareto points obtained through the baseline and the RASW approaches in the 2-dimensional space $F1\text{-}score/throughput$[2]. The Pareto-minimal configuration set from the baseline implementations contains 16 Pareto points while the Pareto-minimal configuration set from the RASW approach contains 28 Pareto points. From Fig. 5(b) we can see that also in this case the Pareto-minimal configuration set from the baseline design space is dominated by the Pareto-minimal configuration set from the RASW design space. In practice, this implies that for a given accuracy, the RASW approach always provides at least one configuration point, which provides a throughput at least as high as the one provided by any configuration point from the baseline implementations with the same (or better) accuracy.

The results in Fig. 5 show that, in the design space of the parameters involved in the scene-scanning process, RASW provides new configurations that dominate the configurations achievable through the existing implementations. This means that the same (or better) recall can be achieved with more precision and that the same (or better) accuracy can be achieved with less computation. For example, consider the Pareto-minimal configuration achievable through the OpenCV 1 implementation by setting $s = 1.2$ and $\gamma = 4$ (second x-mark from the top in Fig. 5(b)). Such a point leads to $F1\text{-}score = 87.9$ and $throughput = 6.1\, fps$. By selecting $\Delta_{x,t1} = 5$, $\Delta_{x,t2} = 10$, $\Delta_{y,t1} = 5$, $\Delta_{y,t2} = 5$, $s = 1.3$ and $\gamma = 4$, the RASW approach leads to a better accuracy ($F1\text{-}score = 88.4$) with a 2.03x speedup ($throughput = 12.4\, fps$). This point is plotted with a black circle in Fig. 5(b).

The gain in performance achievable through the RASW approach is not constant, i.e. it varies depending on the position

[1]To provide our results so that they are in line with the traditional definition of Pareto point [17], we use $1-recall$ instead of recall, $1-precision$ instead of precision. In this way the theoretically best possible configuration coincides with the origin of the axes.

[2]Also in this case we use $1-$*F1-score* and $1/throughput$ in order to report the theoretically best possible solution in coincidence with the origin of the axes.



(a) Recall/precision trade-off space.

(b) F1-score/throughput trade-off space.

Fig. 5. Pareto points in the *recall/precision* (a) and in the $F1\text{-}score/throughput$ space (b). The crosses and x-marks are Pareto-minimal solutions from the static and OpenCV 1 approaches, respectively (none of the Pareto points comes from the OpenCV 2 approach, therefore no symbol has been plotted in its correspondence), the red circles are the Pareto-minimal points provided by the RASW approach. From the figure we can see that when designing an object-detection system, the proposed RASW approach can always provide better solutions with respect to a baseline implementation.

in the design space. The purpose of this paper is to show that, through the RASW approach, the designer of an object detector has now the possibility to choose new design points that better fit the target application. For example, suppose that the target application requires a throughput of at least $10\ fps$, with a recall greater than 80% and a precision above 95%. Then, by choosing the set of parameters mentioned above for the RASW approach the designer will get the design point corresponding to the black circle in Fig. 5(b), characterized by $recall = 82.4\%$, $precision = 95.2\%$, $throughput = 12.4\ fps$, which meets the application requirements. This point is not achievable through any of the baseline implementations (the Pareto-minimal configuration from the OpenCV 1 implementation mentioned above leads to $recall = 82.6\%$, $precision = 94\%$ and $throughput = 6.1\ fps$).

VI. Conclusions

This paper proposes the use of a Run-time Adaptive Sliding Window (RASW) for improved object detection. We present our results for an implementation of the Viola-Jones face detector, but any sliding-window based object detection algorithm using a cascade classifier can benefit from our optimization. When compared to the existing approaches, RASW provides better design points both in the recall/precision design space and in the F1-score/throughput design space, which is particularly relevant when targeting real-time applications. The proposed optimization allows to achieve better accuracy than the existing implementations with less computation, and can be implemented on top of architecture-level optimizations or combined with parallelizing techniques to improve the performance even further. Since RASW leads to a large design space, in future work we plan to develop suitable algorithms which automatically search for the parameters setting that best fits the application requirements. The proposed algorithm exploits spatial locality to improve face detection: when capturing images from video, temporal locality can be also taken into account to reduce the required computation even further.

Acknowledgment

This work was supported in part by the COMMIT program under the SenSafety project.

References

[1] C. Zhang and Z. Zhang, "A survey of recent advances in face detection," in *Technical report MSR-TR-2010-66*, 2010.
[2] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, 2001.
[3] J. Cho, B. Benson, S. Mirzaei, and R. Kastner, "Parallelized architecture of multiple classifiers for face detection," in *ASAP*, 2009.
[4] M. Hiromoto, H. Sugano, and R. Miyamoto, "Partially parallel architecture for Adaboost-based detection with haar-like features," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, pp. 41–52, 2009.
[5] B. Brousseau and J. Rose, "An energy-efficient, fast FPGA hardware architecture for OpenCV-compatible object detection," in *FPT*, 2012.
[6] D. Hefenbrock, J. Oberg, N. Thanh, R. Kastner, and S. Baden, "Accelerating Viola-Jones face detection to FPGA-level using GPUs," in *FCCM*, 2010.
[7] D. Oro, C. Fernández, J. R. Saeta, X. Martorell, and J. Hernando, "Real-time GPU-based face detection in HD video sequences," in *ICCV Workshops*, 2011.
[8] S. C. Tek and M. Gokmen, "GPU accelerated real-time object detection on high resolution videos using modified census transform," in *VISAPP*, 2012.
[9] L. Acasandrei and A. Barriga, "Accelerating Viola-Jones face detection for embedded and SoC environments," in *ICDSC*, 2011.
[10] C. Lampert, M. Blaschko, and T. Hofmann, "Beyond sliding windows: object localization by efficient subwindow search," in *CVPR*, 2008.
[11] N. J. Butko and J. R. Movellan, "Optimal scanning for faster object detection," in *CVPR*, 2009.
[12] V. B. Subburaman and S. Marcel, "Alternative search techniques for face detection using location estimation and binary features," *Comput. Vision Image Understanding*, vol. 117, no. 5, pp. 551–570, 2013.
[13] V. Jain and E. Learned-Miller, "FDDB: A benchmark for face detection in unconstrained settings," University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009, 2010.
[14] G. Bradski, "The OpenCV Library," *Dr. Dobb's J. Softw. Tools*, 2000.
[15] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 20, pp. 23–38, 1998.
[16] M. Everingham, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, pp. 303–338, 2010.
[17] M. Geilen, T. Basten, B. D. Theelen, and R. Otten, "An algebra of pareto points," *Fundam. Inform.*, vol. 78, no. 1, pp. 35–74, 2007.
[18] C. Goutte and É. Gaussier, "A probabilistic interpretation of precision, recall and *f*-score, with implication for evaluation," in *ECIR*, 2005.