Measuring the Effectiveness of Bad Pixel Detection Algorithms Using the ROC Curve

Vittal Premachandran, *Graduate Student Member*, IEEE, Ramakrishna Kakarala, *Senior Member*, IEEE

Abstract — *Digital camera sensors usually contain defective pixels that either arise during the manufacturing process, or develop over time. These defects though small in number, are very noticeable to an experienced eye when looking at images captured from the camera. As a result it becomes extremely important for the sensor manufacturing companies to develop methods which would reduce these manufacturing errors. There have been a number of algorithms proposed for online detection of sensor defects in the literature, but there is no standard method of fairly comparing algorithms given that each algorithm uses thresholds in a different way. This paper presents a novel idea which makes use of the Receiver Operating Characteristics (ROC) curve for comparison among the various bad pixel correction algorithms. We demonstrate how the characteristics of the ROC curve can be used for determining the effectiveness of the algorithms and hence help us to decide which of the algorithms perform better than the others. Experimental results comparing two such bad pixel detection algorithms are provided*[1].

Index Terms —Defective Pixels, Camera Sensor Noise, ROC Curve, Algorithm Comparison.

I. INTRODUCTION

A camera sensor manufacturing process almost certainly introduces defects into solid state sensors, be they CCD sensors or CMOS sensors. The occurrences of these defective pixels may be caused by a number of factors. Some of them may be because of the non uniform temperature in the manufacturing unit, others due to impurities in the chemicals used during the sensor etching, etc. One or more combinations of these factors may also affect the behavior of the camera pixels. It becomes extremely difficult and costly to totally prevent the occurrences of these defects in the sensor array after a certain point. So, a limited number of defective pixels are allowed to occur in any given camera sensor. However, this number must be small and there should be a way to detect the defective pixels and correct them. The cameras that are being manufactured today have the capacity of capturing high resolution images which means that there are a large number of sensor pixels that are being squeezed onto the sensor array. As this number increases, it is only natural to assume that the number of defective pixels also increases. Hence, it is becoming all the more important for us to look at different ways in which we can reduce the number of defects in any given camera sensor.

Bad pixels may be classified into several categories: stuck-on or stuck-off pixels, for which the value does not change regardless of light, "hot'' pixels, which show a much higher reading than their neighbors for the same incident light, "cold'' pixels, which are similarly defined, and abnormally noisy pixels which may alternate randomly between being above or below the nominal reading for the incident light. Bad pixels may be detected during manufacturing, which results in either the sensor being rejected, or if the defects are sufficiently few, their locations being written into one-time-programmable (OTP) memory. However, bad pixels also develop in the field during use, either due to temperature, radiation, or other forms of damage. Sometimes pixels might not actually be "bad", but appear to be bad because of the smearing effect 0, which occurs when the sensor is acquiring the image of an extremely bright light source, such as the sun, headlight of a car, etc. A. Bosco, A. Bruna, F. Naccari, I. Guarneri also acknowledge the fact that defective pixels arise during the course of use of a camera and propose a way to dynamically identify these defects and correct them [1]. Dudas *et al.* [3] show that four different consumer cameras ranging in age from 0.5 years to 5 years have between 7 to 14 bad pixels. Their results are for high-end DSLR (Digital Single Lens Reflex) cameras, and therefore it is safe to assume the incidence of defects is higher for the inexpensive but widely-used cell phone cameras. Hence, some form of pixel defect detection that works in the field is highly desirable.

Different types of bad pixel correction algorithms have been proposed and they can be classified into two major categories: "online'', which works independently on each frame as the image data is being read out, or "offline'', which requires storage of pixel locations and measurements between images. Online methods are the ones that can be used to rectify the corrupted pixels straight away on the camera hardware. I. Baharav, R. Kakarala, X. Zhang, and D. W. Vook [4], and J. H. Stanback, X. Zhang, R. Kakarala, and B. Y. P. Ying [5] propose algorithms which fall in this category. Offline methods on the

[1] This work was supported in part by a Seed Grant provided by the Institute for Media Innovation (IMI).

Vittal Premachandran is a PhD student in the School of Computer Engineering, Nanyang Technological University, Singapore - 639798 (e-mail: vittalp@pmail.ntu.edu.sg).

Ramakrishna Kakarala is an Associate Professor in the School of Computer Engineering, Nanyang Technological University, Singapore - 639798 (e-mail: Ramakrishna@ntu.edu.sg).

other hand, make these corrections on the images outside the camera, on a computer, or, on the camera but using a database of images stored in a non-volatile memory. Papers by Y. P. Tan and T. Acharya [11], and, J. Dudas, C. Jung, G. H. Chapman, Z. Koren, and I. Koren [6] are examples of the offline method, in which the deviation of a pixel is determined by comparing its value to its neighbors, and, if this deviation persists over a number of independent images, the pixel is deemed defective. Such schemes, though highly accurate when sufficiently many unrelated images are taken, require pixel locations to remain in memory between frames. Furthermore, the memory used must be both writeable and non-volatile if the results are to be stable during the frequent turn on-off cycles of a typical consumer camera. Note also that the programmable non-volatile memory must be present in the same chip that processes the pixel data; in cell phone cameras, the pixel processing is often on the *same* chip as the image sensor, in which case the addition of on-chip, programmable non-volatile memory (excluding OTP) significantly changes the manufacturing process and adds considerable cost. Hence, cell phone camera sensors do not employ on-chip non-volatile memory, making offline methods infeasible.

Virtually all camera sensor manufacturers have patents on online bad pixel detection, clearly indicating significant interest in such methods. We review the methods given in [4] and [5] in more detail later in this paper. Online methods operate by comparing each pixel's value to a prediction based on the immediate neighbors, and if the pixel value is significantly different (as measured by one or more thresholds), the pixel is judged bad and its value replaced by interpolation from neighbors. Online methods have their merits and drawbacks. On the plus side, they require no non-volatile memory, and indeed reuse the (volatile) SRAM memory which must be allocated anyway for line buffers required for "demosaicking''-- interpolating colors from the color filter array. Moreover, at low light levels, online bad pixel detection has the benefit of reducing noise extremes. The drawbacks include a loss of detail due to "good" pixels being mistakenly identified as "bad" and corrected, as well as potentially-inconsistent detection from frame to frame, which may result in a "blinking'' pixel effect in video.

Many of the existing algorithms successfully mask the defects caused by the bad pixels. Since the offline algorithms perform their task on the images outside of the camera, they can be complex and may require much higher computational resources than the online algorithms. But, the plus side of offline algorithms is that they may be more effective in their defect masking capabilities. While each of these categories of algorithms has their own advantages and disadvantages, there is no proper way for us to judge which one of the many available algorithms is better than the others. Most of these algorithms usually use one or more threshold parameters to fine tune their efficiencies. The units and magnitudes of these threshold parameters vary from algorithm to algorithm. Tweaking these parameters affects the algorithm's defect concealment efficiency. Some algorithms may fare better than other algorithms over the whole range of their threshold parameters, while some others may be superior only for a small range of their threshold parameters. This difference in the efficiencies of the algorithms over the complete threshold range makes it difficult to choose the best algorithm from a set.

In this paper we propose the novel use of the Receiver Operating Characteristics (ROC) curve to tackle the problem that is described above. We show how the ROC curve can be analyzed to make distinctions between two or more algorithms by reducing the ROC curves to a single comparable metric. This would provide us with a standardized way to make the comparison. For demonstration purposes, we implement two different algorithms and make a comparison between them using the ROC curve characteristics.

The rest of the paper is organized as follows. Section II describes in detail the two bad pixel algorithms that we use in our experiments. We also describe some modifications made to the original algorithms in order to generalize the correction process. In Section III we review the properties of the ROC curve and propose a new robust way of comparison among the different bad pixel algorithms. Experimental details and results which corroborate our claims are reported in Section IV and finally, in Section V we summarize the paper by highlighting the key contributions made.

II. BAD PIXEL ALGORITHMS

We choose two different bad pixel correction algorithms for the purposes of testing our approach and we give a brief description of them in this section. The algorithms we choose fall into the category of "online" defect correction algorithms. They can be built into the hardware circuitry of the camera. They do not use previously shot images from the same camera. The first algorithm that we make use of is the one suggested in [4]. The reason for choosing this is that, it is a fairly simple algorithm with minimal desire for system resources. This makes use of a "bad pixel corrector" which scans each pixel in the image one by one and makes the decision whether the pixel is "good" or "bad". The algorithm scans the pixels row wise and makes use of the data collected from the neighboring pixels and not just the pixel being scanned. It stores the data for *m* other neighboring pixels in order to assist the process of decision making. In our experiment we stored the data for a total of 13 pixels of which 6 of them are from the current line apart from the pixel under the scanner, and 7 from the previous line. To assist the determination process, the algorithm includes a feature buffer which stores the characteristic features of each of the pixels from a previously read-out line. In our experiment, the characteristic feature is made of either a *true* or a *false* value for every pixel which states whether the pixel from the previously read-out line was found to be a "good" or a "bad" pixel respectively.

The algorithm makes use of the neighboring pixel data that is stored in the buffer and obtains the local color correlation among the pixels. Details on how the color correlation between the various pixels is calculated can be found in [4]. A threshold is then selected from this color correlation and an estimate of the pixel under consideration is made based on the pixel values of the surrounding pixels. The difference between the actual pixel value and the estimated pixel value is obtained for comparison with the previously obtained threshold value. The algorithm then checks if the perpendicular pixel to the current pixel being scanned in the previous line is a "good" pixel or a "bad" pixel. If the perpendicular pixel is found to be "good", the difference is compared directly with the calculated threshold value. If the difference is greater than the threshold value, the pixel being tested is considered to be a "bad" pixel and is replaced by the estimated pixel value calculated above. If the difference is not greater than the threshold value, the pixel is considered to be a "good" pixel and is left unchanged. If on the other hand, the perpendicular pixel was found to be a "bad" pixel, then the algorithm as proposed in [4] does not permit the current pixel being scanned to be "bad". But, in our experiment we generalized the algorithm by making use of a second threshold, different from the first, to compare with the difference value when the perpendicular pixel was detected to be "bad". The second threshold was taken as a multiple of the first threshold and the multiplier used in our experiments was the value 2. This process is then repeated for all the pixels in the image and a decision is made on each of the pixels. A flowchart of the algorithm just described is given in Fig. 1.

The second algorithm that we use for experimentation is the one suggested in [5]. This algorithm is a fairly robust one which makes use of a 25 pixel neighborhood for its work area. A 25 pixel neighborhood is commonly used in cameras for demosaicking [7] and hence the algorithm can be combined with the interpolation process. As in the previous algorithm, this algorithm makes use of a "bad pixel corrector" which would decide whether a particular pixel is "good" or "bad". We store the data from the neighboring pixels, of the pixel under consideration and make use of them while taking the decision. The buffer includes the current row of sensor values, two neighboring rows of sensor values that are previous to the current row and two neighboring rows of sensor values that are subsequent to the current row. Overall a 5x5 block of sensor values are stored with the center pixel being the pixel that is currently being scanned. This is shown in Fig. 2. The algorithm sets two threshold parameters (denoted t_1 and t_2) which affect the threshold value. The algorithm then calculates the intensity ratios of the different color neighboring pixels, and makes use of these to compute the eight different estimate values denoted *E1* to *E8*. The equations to calculate the estimate values *En* are given in [5]. The maximum (E_{max}), minimum (E_{min}) and the mean (E_{mean}) of the eight estimate values just calculated are determined. We make use of E_{mean}, t_1 and t_2 to calculate the threshold value, *Th*. Two other parameters are computed namely, low_{value} and $high_{value}$. The low_{value} is computed using E_{min} and *Th*, while the $high_{value}$ is computed using E_{max} and *Th*.

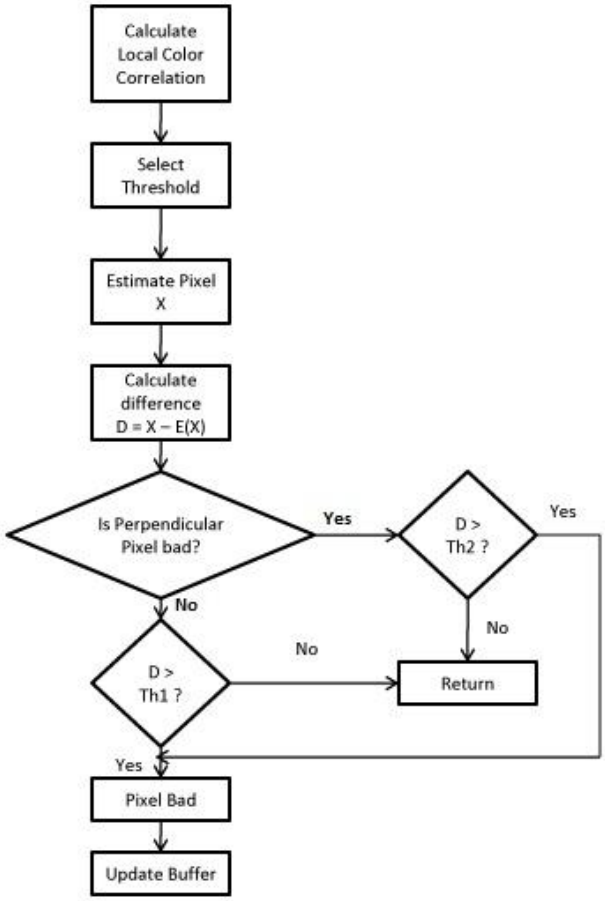


Fig. 1. The flowchart shows the modified Baharav *et al.* algorithm. It can be seen that we use a new and different threshold value *Th2* to make the decision on the current pixel, if the perpendicular pixel to the current pixel being scanned was labeled as "bad". The original algorithm in the patent used a single threshold value *Th1* and did not allow a pixel to be labeled "bad" if its perpendicular pixel was not "good".

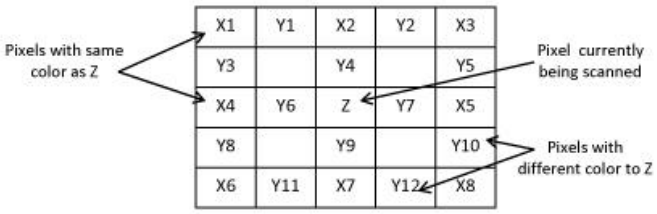


Fig. 2. A pixel neighborhood grid, which is used by the Stanback *et al.* algorithm, during the decision making step. The pixel currently being scanned is labeled as 'Z' in the figure. This is the pixel upon which the decision will be made i.e., "good" pixel or "bad" pixel. The pixels labeled with 'X's' are the pixels which capture the same color as the center pixel labeled 'Z'. The pixels labeled with 'Y's' are the pixels which trap a color different from the color trapped by the center pixel labeled 'Z'.

The current sensor value is compared with the low_{value} and

the $high_{value}$. If the current sensor value is between the low_{value} and $high_{value}$, then the pixel is assumed to be a "good" pixel, if not, it is considered to be "bad". If the pixel was found to be "bad", its value is then replaced by the mean estimate value (E_{mean}) in order to provide a more suitable value to the defective pixel.

III. ROC CURVE METHOD

In this section we describe the ROC curve method to evaluate the bad pixel correction algorithms. The ROC curves have been used by a lot of people in a number of different ways. There are also a quite a few papers in the literature which explain the various uses of the ROC curves. S. Dougherty, K. Bowyer, C. Kranenburg [8] used the ROC curves to evaluate the performance of various edge detection algorithms, while, F. Oberti, A. Teschioni, C.S. Regazzoni [10] used it for evaluating surveillance systems. Another recent use of this can be seen in the paper by T. Judd, K. Ehinger, F. Durand, A. Torralba [9], where the ROC curves are used for comparison between various automated systems which could predict where humans looked when given an image.

A. Background

When faced with the problem of binary classification, there can be four different outcomes. We explain the four outcomes taking the defective pixel detection algorithms to be our classifiers. If the algorithm decides a pixel is bad when it is, in fact, bad, then we call that outcome as a True Positive (TP). However, if the algorithm classifies a pixel as bad when it is, in fact, good, we call the outcome to be False Positive (FP). If on the other hand the algorithm decides the pixel to be good when it is, in fact, good, we call the outcome as a True Negative (TN). Finally if the algorithm concludes that a pixel being scanned is good when it is, in fact, bad, we call the outcome to be a False Negative (FN).

The Receiver Operating Characteristics (ROC) [12] is a graphical plot of the *sensitivity vs. (1-specificity)*. The ROC curve can also be obtained by plotting the *true positive rate vs. false positive rate*. The true positive rate (TPR) is the ratio of the number of true positives to the sum of the true positives and the false negative. The false positive rate (FPR) is the ratio of the number of false positives to the sum of the false positives and the true negatives. These are given below in (1) and (2) respectively.

$$TPR = TP/(TP + FN) \quad (1)$$

$$FPR = FP/(FP + TN) \quad (2)$$

Ideally, we would expect the false negatives and the false positives to be zero. This would mean that, in the ideal case we obtain the true positive rate to be 1 and the false positive rate to be 0. This requirement gives us a way to judge the response of an algorithm. By looking at its ROC curve we can tell whether the algorithm is "good" or "bad" based on how close the curve is to the top left hand corner of the graph (which is where the ideal case of TPR = 1 and FPR = 0 exists). An ROC curve is a standardized way of representing the characteristics of an algorithm. The curve depends only on the values of the TPR and FPR, which is a direct measure of the defect detection capacity of the algorithms. Thus, by making use of the ROC curve for algorithm efficiency analysis, we eliminate the discrepancies that might arise by using other comparison techniques which depend on the values of the threshold parameters (whose units and magnitudes may vary across algorithms) to make the decision.

A perfect classification means that there are no false negatives (FN) and no false positives (FP). Such a classification would result in a single point at the co-ordinates (0,1) on the graph. A completely random guess would mean that we would obtain a point that would lie on the diagonal line running from the bottom left co-ordinate (0,0) to the top right hand corner (1,1) of the graph. This line is called the chance line. An algorithm can be considered reasonable if it is able to correctly classify more than 50% of the defective pixels as being defective and also, does not classify more than 50% of the "good" pixels as defective. This means that, if we plot the ROC curve for an algorithm, the curve should lie in the top left part of the graph i.e., above the chance line.

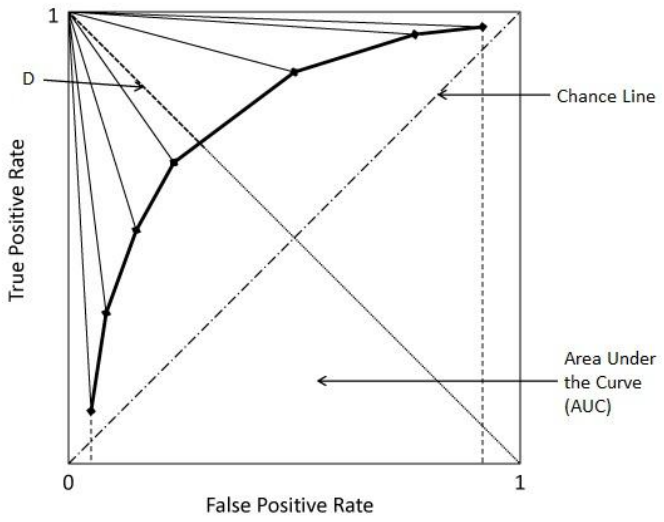


Fig. 3. ROC Curve depicting the different metrics explained in Section III. The area enclosed by the ROC curve and the dotted line from the end points onto the x-axis, is the metric Area Under the Curve (AUC). The length of the bold line from the point (0,1) to the ROC Curve is the metric which we call 'D'. Finally, the ACD can be found by taking an average of the lengths of the lines from the discrete points on the ROC Curve to the top-left point, (0,1).

We can obtain the ROC curves for different algorithms by varying their free parameters, which usually is the threshold value. By doing so, we can obtain the number of true positives and false positives that the algorithm produces in each iteration. Upon obtaining the TP and FP values, we calculate the TPR and FPR as given in (1) and (2) respectively. We now proceed to plot the ROC curves using the TPR and the FPR values just calculated for the different algorithms that we

would like to compare. Visual inspection of the graph can now be done and we can easily pick the curve that is 'closest' to the point (0,1). This curve would then represent the algorithm that would be the most efficient algorithm of the lot.

While this may be easily done by looking at a graph, we would like to make that decision using a mathematical model. Thus, after obtaining the ROC curve, we need to obtain a metric or a method to mathematically determine which of the various algorithms, based on their respective curves, is the best of the lot. For this, we can make use of a number of well established metrics [12] that can be obtained from the ROC curve. The distance between the point (0,1) and the intercept of the ROC curve with the line perpendicular to the chance line (say *D*). The smaller the value of this parameter 'D', the better is the algorithm corresponding to the ROC curve. Two other metrics that can be used are the area under the curve, and, the area between the ROC curve and the chance line. The area under the curve is also called as the *AUC* or *A'* (A-prime). Fig. 3 shows the different metrics that were mentioned above.

B. Average Curve Distance (ACD)

While some of the metrics described in the previous subsection look to be perfectly acceptable to base the decisions upon, the problem arises when we try to calculate their values. Most of the metrics require the equation of the curve to be known, which in our case is an unknown quantity because, we plot the curve as set of discrete points on the graph, which we get by varying the threshold values in discrete steps. So, in order to calculate the above metrics, we would have to make a number of approximations. For example, if we were to calculate the AUC, we would have to calculate it in parts and then sum them up, or if we wanted to find the distance between the point (0,1) and the intercept, we would have to approximate the curve to be a line segment between two discrete points on the graph in order to find the point of intersection. Because of these problems we introduce a new metric which would help us when we have only a discrete set of points and not the whole curve with its equation. We call it the Average Curve Distance (ACD) and it is calculated as given in (3).

$$ACD_j = \frac{1}{n_j}\sum_{i=1}^{n_j} dist_{i,j} \tag{3}$$

$$dist_{i,j} = \sqrt{\left(FPR_{i,j} - 0\right)^2 + \left(TPR_{i,j} - 1\right)^2} \tag{4}$$

Where ACD_j denotes the Average Curve Distance for curve *j*, $dist_{i,j}$ denotes the distance of a point *i* in curve *j* from the optimum point (0,1) as given in (4), and n_j is the number of discrete points in the curve *j*. n_j different points for the curve *j* are generated by setting the threshold parameter to different values, $\lambda_1,\ldots,\lambda_{nj}$.

The way to calculate $dist_{i,j}$ is given in (4) where, $FPR_{i,j}$ is the false positive rate at some threshold value λ_i and $TPR_{i,j}$ is the true positive rate at the same threshold value λ_i produced from some algorithm Ψ_j which generates the curve *j* in the ROC graph. Once we calculate ACDs for all the curves in the ROC graph, we find the ACD whose value is the minimum among the set of all the ACDs ($\min_j\{ACD_j\}$). Hence, the most efficient algorithm can then be selected as it corresponds to the one with the minimum most ACD.

One of the advantages of using the ACD is that it allows us to find out which curve is the closest to the ideal point (0,1) across different threshold values. Another advantage is that, this metric can be made use of even when the number of discrete points varies over the set of different algorithms. This means that even if the number of threshold values that we test the algorithms with varies from one algorithm to the other, we will still be able to compare their ACD values.

A limitation of using the ACD is that, it gives an accurate metric for comparison only if the set of threshold parameters are selected properly. The parameters should be selected in such a way that the ROC curve obtained from its respective algorithm for the selected parameter set sweeps across the graph completely in a given FPR range [FPR_α, FPR_β], which is along the *x* axis. In other words, only the ACDs measured for the ROC curves within the closed interval [FPR_α, FPR_β] which have discrete points on both the lines $x=FPR_\alpha$ and $x= FPR_\beta$ are comparable. This condition can usually be relaxed to a certain extent by making the lower level of the range $FPR_\alpha = 0$. Finding a threshold value for the algorithm which makes the FPR_α to be equal to 0 or at least close to 0 is usually an easy task to achieve. Once this is done, the new range becomes [0, FPR_β]. This is a much more realistic range for algorithm comparison as the sensor manufacturing companies usually allow a limited number of defective pixels to go undetected in any given detection algorithm for the reason stated in Section I. FPR_β can now be set to the allowable threshold value, Λ (which is the maximum allowable FPR and should not be confused with the algorithm's threshold parameters and threshold values).

Once these conditions are met, it would become easy to calculate the ACDs for the different algorithms and use it as a metric for comparison between them. The ACD would be a suitable metric even if the number of discrete points in the range [0, Λ] varies across algorithms. The more the number of points, the better would be the measure of the average distance of the curve from the ideal point (0,1).

C. A Mathematical Approach

Another more generic and mathematical way of comparing the two or more curves is given in this sub section. This method first tries to obtain an approximate equation of the curve and then tries to compare the different curves. The curve is assumed to be made up of a number of line segments. These line segments are the short lines between any two consecutive discrete points that are obtained to draw the ROC curve. Consider two algorithms, Ψ_i and Ψ_j where $i \neq j$ and $i,j \in \{1,2,3,\ldots,k\}$, where *k* is the total number of algorithms to be compared. Let $\Phi_i(x)$ denote the equation of the ROC curve corresponding to the algorithm Ψ_i, and $\Phi_j(x)$ be the equation of the ROC curve corresponding to the algorithm Ψ_j. Let N_i be the set of FPR values within the range [FPR_α, FPR_β], that are

obtained from the algorithm Ψ_i for the different threshold values. Again, the value of FPR_α could be 0. The set N_i can be written as, $N_i = \{x_1, x_2, ..., x_{ni}\}$, where x_t's are the intermediate values of the FPR and n_i is the total number of discrete points within the closed interval $[FPR_\alpha, FPR_\beta]$. Here $x_1 = FPR_\alpha$ and $x_{ni} = FPR_\beta$.

We can now write the equation of the curve as a set of equations as given below in (5).

$$\Phi_i(x) = \begin{cases} f_1(x) & if \quad x_1 \le x \le x_2 \\ \dots & \\ f_t(x) & if \quad x_t \le x \le x_{t+1} \\ \dots & \\ f_{n_{i-1}}(x) & if \quad x_{n_{i-1}} \le x \le x_{n_i} \end{cases} \tag{5}$$

Each of the $f_t(x)$'s in (5) is a straight line joining the points (FPR_t, TPR_t) and (FPR_{t+1}, TPR_{t+1}). Now that we have the equations of the lines within the required range, we can make use of them in any necessary way to decide which of the algorithms gives the best result.

The ROC curves, though monotonically increasing, may cross over other ROC curves. This makes the decision process even more difficult. It becomes necessary for us to check if there are any intersections between the ROC curves that we are comparing. We need to find the points of intersection (if any), that lie in the interval $[FPR_\alpha, FPR_\beta]$, between the curves we are comparing, which in our case are the curves $\Phi_i(x)$ and $\Phi_j(x)$. Since the curves are composed of line segments and since we know the equations of these line segments, we can compute the intersection points. We need to consider only the intersection points that lie on the curves and not the ones where the line segments meet each other outside their respective curves.

Two cases now arise out of this process. We may either find that the curves do not intersect with each other at all, or we may find that the curves in fact do intersect with each other (may be multiple number of times). Let us consider each case individually.

Case 1: No Intersection

In this case the two ROC curves do not intersect with each other over the range (FPR_α, FPR_β). Intersections at the end points (if at all) can also be put into this category. We now choose a test value for x between FPR_α and FPR_β, say, $x_{TEST}=(FPR_\alpha+FPR_\beta)/2$. Any other test value within the range (FPR_α, FPR_β) can also be chosen. If $\Phi_i(x_{TEST}) > \Phi_j(x_{TEST})$, we can conclude that the algorithm corresponding to the curve Φ_i produces a better efficiency than the algorithm corresponding to the curve Φ_j. This is because the curve with the greater value at x_{TEST} is closer to the ideal point (0,1) than the curve with a lower value. On the other hand, if $\Phi_i(x_{TEST}) < \Phi_j(x_{TEST})$, then the algorithm corresponding to the curve Φ_j is better than the algorithm corresponding to the curve Φ_i.

Case 2: With Intersection(s)

If there are points of intersection in the range (FPR_α, FPR_β), it means that the curves cross over each other. There could be a single cross-over point, or multiple cross-over points. The m intersection points are first found, where $m \ge 1$, and the complete range (FPR_α, FPR_β) is subdivided into multiple sub-ranges, $\{[FPR_\alpha, x_1], [x_1, x_2], ..., [x_m, FPR_\beta]\}$. For each sub-range, we find a test value for x. For sub-range l, the test value could be $x_{TESTl} = (x_{l-1} + x_l)/2$, where $l \in \{1,2,...,m+1\}$, $x_0=FPR_\alpha$ and $x_{m+1}=FPR_\beta$. If $\Phi_i(x_{TESTl}) > \Phi_j(x_{TESTl})$, it means that the algorithm corresponding to the curve Φ_i produces a better result in the sub-range $[x_{l-1}, x_l]$ than the algorithm corresponding to the curve Φ_j and vice-versa. This way, we can find out which algorithm produces the best result, and in which sub-range.

If we need to find out which algorithm produces the better result for a major part of the range, we can use a weight based calculation system. The procedure to calculate the weights is given in Fig. 4. The pixel-correction algorithm with more weight is the algorithm that is more efficient for a major part of the range $[FPR_\alpha, FPR_\beta]$.

$w_i = 0$
$w_j = 0$
for $l=1$ to $m+1$ **do**
  $dist_l = x_l - x_{l-1}$
  if $\Phi_i(x_{TESTl}) > \Phi_j(x_{TESTl})$ **then**
    $w_i = w_i + dist_l$
  else
    $w_j = w_j + dist_l$
  endif
endfor

Fig. 4. A weight based calculation method to determine which pixel-correction algorithm produces the better result for a major part of the range.

IV. EXPERIMENTAL RESULTS

In this section we explain the details of the experiments that were carried out to demonstrate the effectiveness of the ROC curve method in determining the most efficient algorithm from a given set. For the purposes of the experiment we used 5 different 501x501 RGB images of which four are shown in Fig. 5 and the fifth is shown in Fig. 6(a). The images were chosen as they contained a lot of high frequency components which would act as a major test for the chosen algorithms. Such images would invite the algorithms to make an increased number of false positive choices. The algorithm which would produce the least number of false positive choices and a highest percentage of true positive choices would be the most efficient one. This means that the ROC curve for such an algorithm should be closer to the top left hand corner of the ROC graph.

The images were converted to a pseudo RAW format in order to mimic the camera's sensor array. A detailed description of the design of camera sensor arrays can be found in [13]. The sensory unit of the camera is a combination of a

filter array and a sensor array. The filter array usually has the Bayer Pattern [14]. The pseudo RAW image was obtained by alternatively taking the Red and Green components of the pixels in an even row of the image, and by alternatively taking the Green and Blue components of the pixels in an odd row of the image. Defects were artificially introduced into the pseudo RAW image by adding a constant value of 100, to a total of 1681 (41*41) pixels in the image. Consecutive defects were added at a distance of 5 pixels, both in horizontal and vertical direction, from the previous defective pixel starting from pixel number (150,150) and going up to pixel number (350,350).

Fig. 5. Four of the total five images used in our experiments are shown here in this figure. The 5th image that was used is shown in Fig. 6(a). Each of these images was used at a resolution of 501x501. The images shown were chosen as they contained high frequency (edges) components in them.

A cropped pseudo RAW image of one of the images that was used for testing can be seen in Fig. 6(b), where the mosaicking effect can clearly be seen. The image should not be mistaken as a monochrome image and only appears so in Fig. 6(b) because the pseudo RAW image has only one plane with each pixel contributing to one of Red, Green or Blue color. Every alternate pixel in the same row represents one particular color. The defect induced image is shown in Fig. 6(c), which is a cropped out version of the original picture shown in Fig. 6(a). The image was cropped to better facilitate the visibility of the induced defects.

The two algorithms that were mentioned above in Section II were then tested on these defect induced pseudo RAW images. While testing the algorithm proposed in the patent [5], the parameters were varied thus: the threshold parameter t_2 was kept constant at 0 and the threshold parameter t_1 was varied from 10 to 60 in steps of 5. And for the algorithm proposed in the patent [4], the parameter *t* was kept constant at 10 and the parameter *n* was varied from 1 to 3.5 in steps of 0.1. The ROC curves generated from the two algorithms for each of the 5 images can be seen in Fig. 7(a) and Fig. 7(b) respectively. The ACDs were calculated as given in (3) and (4) for each of the images and for each algorithm. We then took an average of the ACDs for each algorithm to see its response over the 5 different images. The average ACD for the algorithm from patent [5] was found to be 0.0666 and the average ACD for the algorithm from patent [4] was found to be 0.5271. Since the average ACD of the former is smaller than the latter, we conclude that the algorithm proposed in [5] is better than the algorithm proposed in [4]. Visual inspection of the ROC graphs also validates our conclusion as the curves in Fig. 7(a) are much closer to the ideal case than the curves in Fig. 7(b). Also, notice that the scale of the axis in Fig. 7(b) is in the order of 10^{-3} showing that the algorithm given in [5] has produced far less false positive values when compared to the algorithm given in [4].

We now provide the simulation times for the two algorithms that were used in our experiments. Here we provide the time taken for the execution of the algorithms over the complete image. The simulation was performed in MATLAB using the same set of images as shown above in Fig. 5 and Fig. 6(a). TABLE I lists the simulation times that were obtained for the five different test images while simulating the Baharav algorithm, while TABLE II does the same for the Stanback Algorithm. The time taken is shown in seconds and is for the complete 501x501 image. Though the algorithms work on a pixel by pixel basis, the times are shown for complete images because it is easier for comparison purposes. If we were to take the time taken to check each pixel, we would then have had times in the micro second range.

TABLE I
SIMULATION TIMES FOR BAHARAV ALGORITHM

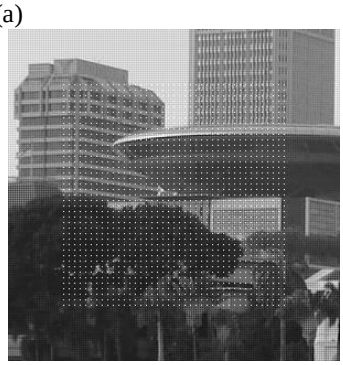
Image #	Time in seconds
Image 1	8.471
Image 2	8.490
Image 3	8.499
Image 4	8.477
Image 5	8.494

TABLE II
SIMULATION TIMES FOR STANBACK ALGORITHM

Image #	Time in seconds
Image 1	2.281
Image 2	2.263
Image 3	2.226
Image 4	2.277
Image 5	2.275

(a)

(b)



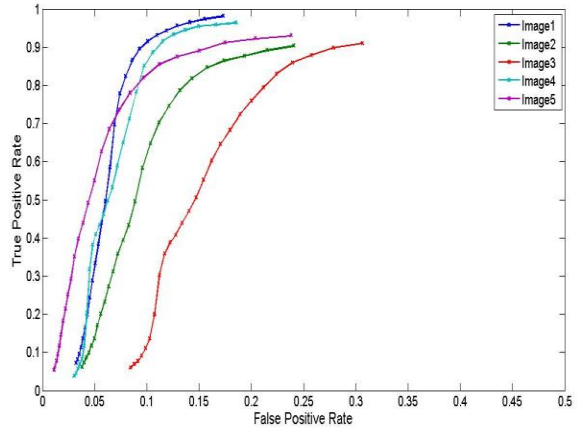
(c)

Fig. 6. (a) A 501x501 cropped image of the buildings along the Singapore River. (b) A portion of the pseudo RAW image, mimicking the Bayer Filter Array (BFA), obtained from the image shown in (a). We can clearly see that the conversion to pseudo RAW format introduces the mosaicking effect in the images. (c) A cropped out region of the pseudo RAW image with defects artificially added in the center of the image ranging from pixel (150,150) to pixel (350,350). The defects were introduced in this range by increasing the intensity of every 5 pixels by a constant value 100.

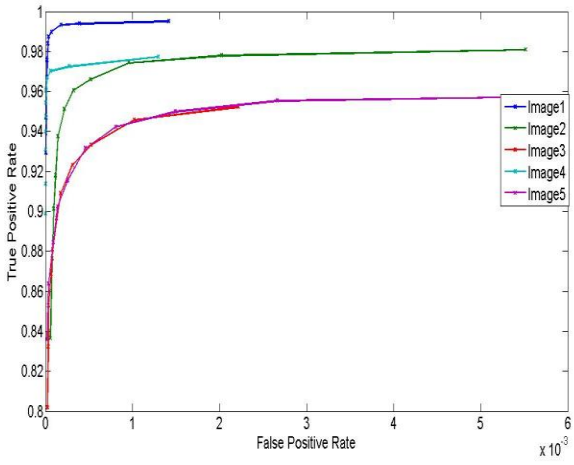
From the simulation times presented in the two tables we can see that the Baharav algorithm takes much longer to execute than the Stanback algorithm. This is because the complexity present in computing the color correlation between the neighboring pixels, which is required by the former, is much higher than the "simple" computations such as *min*, *max* and *mean*, which are required by the latter. Moreover, it is worth stating that these algorithms are usually hardwired in the camera's circuitry and run at pixel rate i.e. at the same rate in which the pixels are read out of the Analog to Digital Converter (ADC). Hence the times given above are purely simulation times performed on a computer. The algorithms when hardwired will run in real-time inside the camera.

V. SUMMARY

In this paper we propose a relatively simple and robust method for comparing two or more algorithms by the novel use of the ROC curve. By determining the average distance of the curve from the ideal point, we are able to predict how good the algorithm actually is compared to other similar algorithms. Its effectiveness can be seen in the experimental results, which corroborates our claim that the ACD can be used a metric for decision making during the ROC curve analysis. This method can be used for comparison between any two similar entities, not just algorithms. Also we present a mathematical approach which enables a more generic comparison which can be used when the constraints required for the ACD calculation cannot be met.



(a)



(b)

Fig. 7. The different curves within a graph show the response of the respective algorithm over different test images. The label 'Image 1' corresponds to the image in Fig. 6(a), while the labels 'Image 2-5' correspond to the images in Fig. 5 from top-left to bottom-right, respectively. (a) ROC Curves generated by the modified Baharav *et al.* algorithm for 5 different images. (b) ROC Curves generated by the modified Stanback *et al.* algorithm for the same 5 different images as chosen for the Baharav *et al.* algorithm. The horizontal scale in (b) is in the range 10^{-3} which shows that the algorithm produced far fewer False Positive values than the one in (a) demonstrating its superiority.

REFERENCES

[1] Young Seok Han, Euncheol Choi and Moon Gi Kang. "Smear removal algorithm using the optical black region for CCD imaging sensors," *IEEE Transactions on Consumer Electronics*, vol.55, no.4, pp.2287-2293, Nov. 2009

[2] A. Bosco, A. Bruna, F. Naccari, I. Guarneri. "Hardware/software solution for high precision defect correction in digital image sensors." in *Proc. IEEE International Symposium on Consumer Electronics*, 14-16 April 2008, pp.1-4.

[3] J. Dudas, L. Wu, C. Jung, G. H. Chapman, Z. Koren, and I. Koren. "Identification of in-field defect development in digital image sensors." in *Proc. SPIE-IS&T Electronic Imaging*, 2008, *SPIE Vol. 6502*.

[4] I. Baharav, R. Kakarala, X. Zhang, and D. W. Vook. "Bad pixel detection and correction in an image sensing device." U.S. Patent 6,737,625, Jan. 2, 2003.

[5] J. H. Stanback, X. Zhang, R. Kakarala, and B. Y. P. Ying. "System and method for detecting and correcting defective pixels in a digital image sensor." U.S. Patent 7,460,688, Jun. 15, 2006.

[6] J. Dudas, C. Jung, G. H. Chapman, Z. Koren, and I. Koren. "Robust detection of defects in imaging arrays." in *Proc. SPIE-IS&T Electronic Imaging*, 2006, *SPIE Vol. 6059*.

[7] B. K. Gunturk, J. Glotzbach, Y. Altunbasak, R. W. Schaffer, and R. M. Merserau, "Demosaicking: color filter array interpolation," *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 44-54, January 2005.

[8] S. Dougherty, K. Bowyer, C. Kranenburg. "ROC curve evaluation of edge detector performance," in *Proc. International Conference on Image Processing*, 4-7 Oct 1998, vol.2, pp.525-529.

[9] T. Judd, K. Ehinger, F. Durand, A. Torralba. "Learning to predict where humans look," in *Proc. International Conference on Computer Vision and Pattern Recognition*, Sept. 29 2009-Oct. 2 2009, pp.2106-2113.

[10] F. Oberti, A. Teschioni, and C.S. Regazzoni. "ROC curves for performance evaluation of video sequences processing systems for surveillance applications," in *Proc. International Conference on Image Processing*, 1999, vol.2, pp.949-953.

[11] Yap-Peng Tan and T. Acharya. "A robust sequential approach for the detection of defective pixels in an image sensor," in *Proc. International Conference on Acoustics, Speech, and Signal Processing*, 15-19 Mar 1999, vol.4, pp.2239-2242.

[12] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Singapore: Wiley, 2004.

[13] R. Lukac, K.N. Plataniotis. "Color filter arrays: design and performance analysis," *IEEE Transactions on Consumer Electronics*, vol.51, no.4, pp. 1260- 1267, Nov. 2005

[14] B. E. Bayer, "Color Imaging Array", U.S. Patent 3,971,065, Jul. 20, 1976.

BIOGRAPHIES

Vittal Premachandran is a PhD student from the School of Computer Engineering at Nanyang Technological University. He became a Graduate Student Member of the IEEE in 2010. Born in Bangalore, India in 1988, he holds a B.E. degree in Computer Science and Engineering from Visveswaraya Technological University, Belgaum, India. His research interests are in the areas of Image Processing and Computational Photography. Contact email: vittalp@pmail.ntu.edu.sg

Ramakrishna Kakarala (SM '01) is an Associate Professor in the School of Computer Engineering at Nanyang Technological University. His research interests include computational photography, with emphasis on image acquisition and on scientific study of photographic aesthetics. Prior to joining NTU, he was a Master Scientist for Avago Technologies, and before that, a Senior Lecturer at the University of Auckland. He has a Ph.D. in Mathematics from the University of California, Irvine, and B.S. and M.S. in Electrical Engineering from the University of Michigan, Ann Arbor. Contact email: Ramakrishna@ntu.edu.sg