Light Random Sprays Retinex: Exploiting the Noisy Illumination Estimation

Nikola Banić and Sven Lončarić

***Abstract*—In this letter Light Random Sprays Retinex (LRSR), an improvement of the Random Sprays Retinex (RSR) algorithm is proposed. RSR is a white balancing algorithm for achieving local color constancy and image enhancement by using random sprays of the same size. The main problem of the original RSR is that the lower the number and size of the sprays, the greater the noise in the resulting image, which means that the number and size of sprays have to be relatively high in order to reduce the noise leading to a higher computation cost. The proposed improved algorithm is based on a new method to remove the noise in the resulting image thereby allowing only one spray of a smaller size to be used resulting in lower computation cost. By using interpolation the computation cost is reduced even further without a noticeable perceptual difference. The improvement is tested on a public database and is shown to outperform the original RSR in image quality and computation cost. The source code is available at http://www.fer.unizg.hr/ipg/resources/color_constancy/.**

***Keywords*—Color constancy, image enhancement, interpolation, noise removal, Random Sprays Retinex, retinex, white balance**

I. Introduction

THE human visual system's (HVS) perception of the color of an area depends on the surrounding visual scene, which is known as *locality of color perception*. One of the early and very well known models dealing with it is the Retinex model [1]. Various Retinex implementations provide dynamic range compression, color constancy and color and lightness rendition. Several examples of practical usage of Retinex for image enhancement have been reported [2][3][4][5]. A special group of Retinex implementations use a path-wise approach, which leads to the following problems: strong dependency on path geometry, high computation cost and sampling noise. All these problems were addressed and highly reduced in [6], which resulted in the Random Sprays Retinex (RSR) algorithm, which is faster than the path-wise Retinex algorithms, but not fast enough to be used for real time applications.

In this letter we propose an extension of the RSR algorithm, which allows significant reduction of computation cost and improves the image quality.

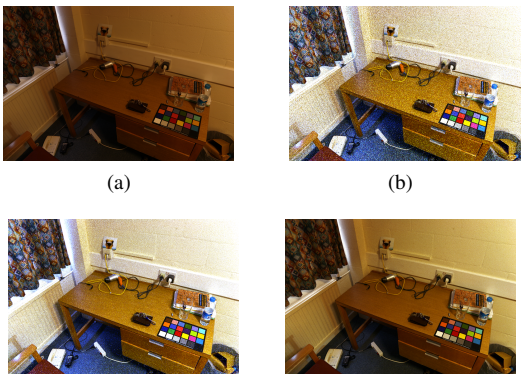
The paper is structured as follows: In Section II a simple explanation of the Random Sprays Retinex algorithm is given, in Section III the proposed method is described, in Section IV the experimental results are presented, and in Section V the proposed method is compared to RSR and similar algorithms in terms of result quality, difference and speed.

The authors are with the Image Processing Group, Department of Electronic Systems and Information Processing, Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia (email: nikola.banic@fer.hr; sven.loncaric@fer.hr).

II. Random Sprays Retinex algorithm

The Random Sprays Retinex algorithm was developed by taking into consideration the mathematical description of the Retinex model provided in [7]. First, the model is simplified and it is proved that the new pixel intensities can be calculated faster. The next step towards RSR in [6] is to notice three reasons for which paths should be replaced with something else: they are redundant, their ordering is completely unimportant, and they have inadequate topological dimension. This leads to use of 2-D objects as representations of the pixel neighbourhood, which is used when calculating the new pixel intensity. The random sprays are chosen as 2-D objects requiring tuning of several parameters.

The spray radius is set to the image diagonal. The identity function is taken as the radial probability density function. The minimal number of sprays (N) and the minimal number of pixels per spray (n) representing a trade-off between image quality and computation cost were determined through several experiments. Fig. 1(a) shows a test image from the ColorChecker image database [8]. RSR results for various parameters are shown in Fig. 1(b), Fig. 1(c) and Fig. 1(d).



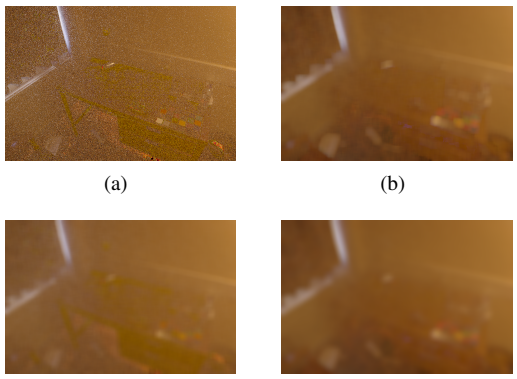
(a) (b) (c) (d)

Fig. 1: Examples of RSR results for various numbers (N) and sizes (n) of sprays: (a) original image, (b) $N = 1, n = 16$, (c) $N = 5, n = 20$, (d) $N = 20, n = 400$

III. Proposed method

A. Basic idea

Fig. 1(b) and Fig. 1(c) show that noise is a great disadvantage of minimizing the RSR computation cost by choosing lower values for parameters N and n. A naive solution would be to use a low-pass filter, but this would also blur the image so another solution has to be looked for.



(a) (b) (c) (d)

Fig. 2: Local intensity changes: (a) no processing, (b) filtered after calculation, (c) filtered before calculation, (d) filtered before and after

Despite the presence of the noise in Fig. 1(b) and Fig. 1(c), it can be observed that the change of color and brightness with respect to the original image changes slowly across the whole image. From the original image and the RSR resulting image, the relative change for a pixel i is calculated using the equation

$$C_c(i) = \frac{I_c(i)}{R_c(i)} \tag{1}$$

where $C_c(i)$ is the intensity change of pixel i for channel c, $I_c(i)$ is the original intensity, and $R_c(i)$ is the intensity after running RSR. By taking the intensity changes for all three image channels, it is possible to visualize them as seen in Fig. 2. The vector $\mathbf{e}(i) = [C_r(i), C_g(i), C_b(i)]^\top$ can be interpreted as RSR local illumination estimation and since it is not necessarily a unit vector, it also represents the local brightness adjustment, i. e. $\mathbf{e}(i) = b(i)\hat{\mathbf{e}}(i)$ where $\hat{\mathbf{e}}(i)$ is a unit vector and $b(i)$ is a scalar representing the brightness adjustment factor. The merged result of Eq. 1 for all channels calculated by using the RSR result from Fig. 1(b) is shown in Fig. 2(a). While the noise is still present, the difference between intensity changes of spatially close pixels is not large thus opening the possibility of applying a low-pass filter without tampering the intensity change too much:

$$C'_{c,k}(i) = (C_c * k)(i) \tag{2}$$

where i is the chosen pixel, c is the chosen channel, k is the averaging kernel, $*$ is the convolution operator and C_c is the intensity change calculated as described in Eq. 1. The application of a 25×25 averaging kernel to the image in Fig. 2(a) results in the image shown in Fig. 2(c). Another way to remove the noise is to use Eq. 1, but with blurred channels:

$$C''_{c,k}(i) = \frac{(I_c * k)(i)}{(R_c * k)(i)} \tag{3}$$

with the result for the same k as before in Fig. 2(b). By combining the two approaches of noise removal (result shown in Fig. 2(d)), the new intensity change is:

$$C^*_{c,k_1,k_2}(i) = (C''_{c,k_1} * k_2)(i). \tag{4}$$

Fig. 3: Resulting image after applying the proposed method.

The intensity change with much less noise can now be applied to the original image in order to get the result output image O where the intensity for a pixel i for a given channel c is calculated like

$$O_c(i) = \frac{I_c(i)}{C^*_{c,k_1,k_2}(i)}. \tag{5}$$

The result of applying Eq. 5 to the image shown in Fig. 1(a) with k_1 and k_2 being 25×25 averaging kernels is shown in Fig. 3. After experimenting with various Gaussian kernels and the averaging kernel it was visible on almost all used test images that the averaging kernel gives better results so this kernel type is used in the proposed method. Another advantage with averaging kernel is that its implementation can easily be made faster than for other kernel types. The proposed method summary is given in Algorithm 1.

Algorithm 1 Light Random Sprays Retinex

```
I = GetImage()
R = PerformRSR(I, parameters)
for all image channels c do
    calculate C*_{c,k1,k2}
    calculate O_c
end for
merge all O_c into final result O
```

B. Interpolation

Slow intensity changes are further exploited to minimize the computation cost by avoiding to calculate the intensity change

for every pixel. Instead, the row step r and the column step c are introduced meaning that only every cth pixel only in every rth row is processed and the rest is calculated by applying interpolation. In the experiments explained further in the text the nearest-neighbor interpolation was used.

C. Complexity

The theoretical complexity of the proposed method displayed in the Big-O notation is $O(\frac{NnM}{rc})$ where M is the number of image pixels, which in case of using interpolation is lower than the original RSR complexity $O(NnM)$. The former Big-O notation does not contain any parts related to averaging as averaging can be implemented with $O(M)$. In Section IV the complexity is shown to be $O(\frac{nM}{rc})$ because N is fixed to 1.

IV. Experimental results

Several experiments were performed for parameters tuning. The proposed method inherits all the parameters from RSR and adds four new ones: the size of the two averaging kernels k_1 and k_2, r and c. For parameters tuning the original ColorChecker database described in [8] containing 568 sRGB images was used with the resolution of most of them being 874×583. The reason for not using the reprocessed version of the ColorChecker database [9] was that we were not measuring illumination estimation accuracy, but mainly the image quality so there was no linear model assumption and no need for linear RGB images. It may be necessary to retune some of the parameters for images with size different from the one of images in ColorChecker database. As a tuning tool the universal image quality index described in [10] was used:

$$Q = \frac{4\sigma_{xy}\bar{x}\bar{y}}{(\sigma_x^2 + \sigma_y^2)\left[(\bar{x})^2 + (\bar{y})^2\right]} \tag{6}$$

where x and y are the original gray-scale image and its modification, respectively. The dynamic range of Q is $[1, -1]$ with 1 being the best value obtained when x and y are equal. For RGB images the mean quality of all three channels was used.

A. Tuning the kernel sizes

When comparing the quality indices of images obtained by applying Eq. 2 and Eq. 3 using various kernel sizes, the former clearly outperforms the latter. The problem with Eq. 2 and Eq. 3 is that in some cases (e.g. when processing image shown in Fig. 1(a)) visual anomalies are still visible after the equation is applied. This is solved by applying Eq. 4 whose average quality index is slightly smaller then the one of Eq. 2, but the anomalies are eliminated so this is how the noise should be removed. In order to simplify further tuning, Eq. 4 is used with $k_1 = k_2$. Choosing a greater kernel size reduces the noise, but it also lessens the locality of RSR, while choosing smaller kernel sizes does the opposite. A good trade-off is achieved by choosing the lowest kernel size for which there is no more significant increment in quality when choosing greater kernel sizes. By looking at Fig. 4 it can be concluded that the value satisfying this constraint for all tested configurations is 15×15. However, after performing visual inspection, we concluded that a better choice is 25×25 as it leads to the same quality and it also removes visible anomalies that occurred in some test images during the visual inspection and were caused by RSR inherited noise.

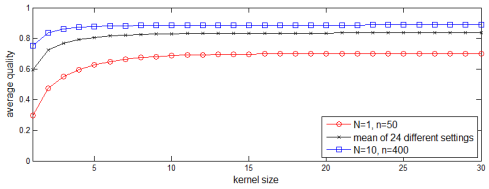


Fig. 4: The effect of kernel size on resulting image quality.

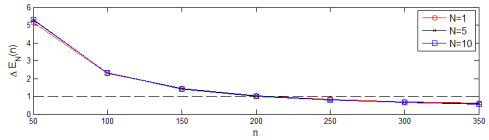


Fig. 5: $\Delta E_N(n)$ for different values of N.

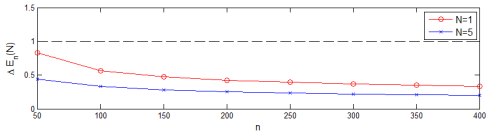


Fig. 6: $\Delta E_n(N)$ for different values of n.

B. Tuning N and n

In [6], parameters N and n were tuned by increasing N from 5 to 60 with a constant step of 5 and increasing n from 250 to 900 with a constant step of 50. Then followed the calculation of $\Delta E_N(n)$ and $\Delta E_n(N)$, the CIELab differences (approximation of perceptual difference) between the RSR results with a fixed value of N and two consecutive values of n, and vice versa, with N playing the role of n. The best choice of values for N and n was obtained by minimizing the product Nn with constraints $\Delta E_N(n) < 1$ and $\Delta E_n(N) < 1$. For consecutive values of N we chose 1, 5 and 10 and for consecutive values of n we chose the values from 50 to 400 with a constant step of 50. By looking at graphs in Fig. 5 and Fig. 6 it can be concluded that values satisfying all mentioned constraints are $N = 1$ and $n = 250$, which is better than values $N = 20$ and $n = 400$ that were obtained in [6] and can be attributed to a greater stability after the noise removal. Because Fig. 5 also clearly shows that the increase of the number of sprays has almost no effect, N can be freely set to 1. This reduces the original LRSR Big-O notation complexity to $O(\frac{nM}{rc})$, which is a significant improvement.

C. Tuning r and c

In order to simplify r and c tuning, let i be the common value for r and c so that $r = c = i$ and let $\Delta E_k(i)$ be the average CIELab difference between images processed with kernel of size 25×25 with no interpolation and images processed with kernel size $k \times k$ and interpolation determined by the parameter i. Now i is maximized by choosing a value of k so that $\Delta E_k(i) < 1$ and from the graph shown in Fig. 7 it can be concluded that a good candidate is $i = 6$ with $k = 175$. Raising r and c without performing averaging spreads the initial noise and drastically reduces the image quality, which leads to conclusion that using this kind of subsampling and interpolation on the original RSR algorithm should be avoided.

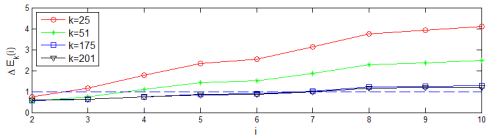


Fig. 7: $\Delta E_k(i)$ for different values of k.

V. Comparison to RSR and other similar algorithms

The comparison between Retinex algorithms is an open problem for reasons described in detail in [6]. Nevertheless, some simple comparisons concerning speed, resulting image average quality and average CIELab difference between the result and the original and between different algorithms were performed. For the comparison we used C++ implementations of RSR, LRSR and some other algorithms that apply local illuminant correction and have publicly available implementations: fast version of Automatic Color Equalization (ACE) [11][12], Multi-Scale Retinex with Color Restoration (MSRCR) [3][13] and Franke-McCann Retinex (FMR) [14]. For FMR we created our own C++ implementation based on the publicly available Matlab implementation [14]. The well-known Local Space Average Color algorithm [15] was not used because unlike the mentioned algorithms, LSAC does not adjust the brightness and due to the fact that one of the factors used in Eq. 6 is the closeness of the mean luminance as described in [10], the comparison would not be fair. All algorithms were used with default parameter values of their implementations. FMR was used with 4 iterations, which is the same as for the test images on the implementation web page. Several LRSR parameter sets were tested: LRSR_A ($N = 20, n = 400, k_1 = k_2 = 175, r = c = 6$), LRSR_B ($N = 1, n = 250, k_1 = k_2 = 25, r = c = 1$) and LRSR_C ($N = 1, n = 250, k_1 = k_2 = 175, r = c = 6$). LRSR_A represents the fast approximation of RSR with its default parameters, while LRSR_B and LRSR_C have default spray size for LRSR without and with interpolation, respectively. LRSR_C also represents the default LRSR parameter set.

A. Speed, CIELab difference and quality comparison

The speed of the proposed method, RSR and other selected algorithms was measured on Ubuntu 11.10 operations system on a computer with i5-2520M CPU (only one core was used) by running the corresponding implementations against 100 images from ColorChecker database. This was repeated several times for each algorithm to calculate the average. The speed test results are shown in Table I together with average resulting image quality and CIELab difference between the original and the result. It is obvious that LRSR is faster than all other tested algorithms, especially when used with LRSR_C parameters set, which was shown to be 229 times faster than RSR. Approximating the exact RSR results with LRSR_A parameters set is 32 times faster. The average CIELab difference between the original and processed images for all methods is in all cases high above the just-noticeable difference (JND) threshold of 2.3 [16] with RSR and LRSR_A producing the smallest difference. It is interesting to mention that even though MSRCR and FMR have similar average difference between the original and the result, their average image qualities differ significantly.

TABLE I: Methods average execution times on 100 images and average resulting image qualities and average CIELab differences from the original for all images

	RSR	LRSR_A	LRSR_B	LRSR_C	ACE	MSRCR	FMR
time (s)	3524.73	108.14	133.02	15.33	274.65s	111.08s	140.16 s
quality	0.786	0.8854	0.8641	0.862	0.6146	0.3673	0.5168
ΔE^*_{ab}	9.93	9.88	11.32	11.31	21.75	34.15	32.55

B. Perceptual difference

The average CIELab difference between results of the tested algorithms is shown in Table II and the fact that the difference between RSR and LRSR with various parameters set is below the JND proves that LRSR approximates RSR very well. All tested methods differ significantly between themselves.

TABLE II: Average CIELab difference between various methods

	RSR	LRSR_A	LRSR_B	LRSR_C	ACE	MSRCR	FMR
RSR	-	1.3290	1.8849	1.9942	16.0306	28.4463	24.5847
LRSR_A	1.3290	-	1.9226	1.6831	16.0570	28.5986	24.5927
LRSR_B	1.8849	1.9226	-	0.8833	15.1503	27.5285	23.4992
LRSR_C	1.9942	1.6831	0.8833	-	15.2240	27.6693	23.5448
ACE	16.0306	16.0570	15.1503	15.2240	-	18.1339	22.9748
MSRCR	28.4463	28.5986	27.5285	27.6693	18.1339	-	26.9623
FMR	24.5847	24.5927	23.4992	23.5448	22.9748	26.9623	-

VI. Conclusion

LRSR significantly reduces computation cost of RSR and gives very similar results, but of higher quality. It therefore allows a much faster usage of the Retinex model in image enhancement. Due to the noise removal, it becomes stable faster than RSR allowing it to have stronger effects on image enhancement.

Acknowledgments

The authors acknowledge the advice of Brian Funt on proper implementation of the Frankle-McCann Retinex algorithm. This work has been supported by the IPA2007/HR/16IPO/001-040514 project "VISTA - Computer Vision Innovations for Safe Traffic."

REFERENCES

[1] E. H. Land, J. J. McCann *et al.*, "Lightness and retinex theory," *Journal of the Optical society of America*, vol. 61, no. 1, pp. 1–11, 1971.

[2] Z.-u. Rahman, D. J. Jobson, and G. A. Woodell, "Multi-scale retinex for color image enhancement," in *Image Processing, 1996. Proceedings., International Conference on*, vol. 3. IEEE, 1996, pp. 1003–1006.

[3] D. J. Jobson, Z.-u. Rahman, and G. A. Woodell, "A multiscale retinex for bridging the gap between color images and the human observation of scenes," *Image Processing, IEEE Transactions on*, vol. 6, no. 7, pp. 965–976, 1997.

[4] Z.-u. Rahman, G. Woodell, and D. J. Jobson, "Retinex image enhancement: Application to medical images," in *The NASA Workshop on New Partnerships in Medical Diagnostic Imaging, Greebelt, Maryland*, 2001.

[5] Z.-u. Rahman, D. J. Jobson, and G. A. Woodell, "Retinex processing for automatic image enhancement," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 100–110, 2004.

[6] E. Provenzi, M. Fierro, A. Rizzi, L. De Carli, D. Gadia, and D. Marini, "Random spray retinex: a new retinex implementation to investigate the local properties of the model," *Image Processing, IEEE Transactions on*, vol. 16, no. 1, pp. 162–171, 2007.

[7] E. Provenzi, L. De Carli, A. Rizzi, and D. Marini, "Mathematical definition and analysis of the Retinex algorithm," *JOSA A*, vol. 22, no. 12, pp. 2613–2621, 2005.

[8] P. V. Gehler, C. Rother, A. Blake, T. Minka, and T. Sharp, "Bayesian color constancy revisited," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[9] B. F. L. Shi. (2013, May) Re-processed Version of the Gehler Color Constancy Dataset of 568 Images. [Online]. Available: http://www.cs.sfu.ca/colour/data/

[10] Z. Wang and A. C. Bovik, "A universal image quality index," *Signal Processing Letters, IEEE*, vol. 9, no. 3, pp. 81–84, 2002.

[11] A. Rizzi, C. Gatta, and D. Marini, "From Retinex to Automatic Color Equalization: issues in developing a new algorithm for unsupervised color equalization," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 75–84, 2004.

[12] P. Getreuer, "Automatic Color Enhancement (ace) and its Fast Implementation," *Image Processing On Line*, no. 2012, 2012.

[13] F. Pelisson. (2013, Sep.) Multi-scale retinex with color restoration. [Online]. Available: http://read.pudn.com/downloads152/sourcecode/graph/texture_mapping/660686/Retinex/retinex.cpp__.htm

[14] B. Funt, F. Ciurea, and J. McCann, "Retinex in MATLAB," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 48–57, 2004.

[15] M. Ebner, "Color constancy based on local space average color," *Machine Vision and Applications*, vol. 20, no. 5, pp. 283–301, 2009.

[16] M. Mahy, L. Van Eycken, and A. Oosterlinck, "Evaluation of uniform color spaces developed after the adoption of CIELAB and CIELUV," *Color research and application*, vol. 19, no. 2, pp. 105–121, 1994.