Real-Time Structure Preserving Image Noise Reduction For Computer Vision On Embedded Platforms*

Walter Nistico, Uwe Schwiegelshohn, Matthias Hebbel, Ingo Dahm
Department of Electrical Engineering
Computer Engineering Institute
University of Dortmund
Otto-Hahn-Str. 4, D-44221, Dortmund, Germany
name.surname@udo.edu

Abstract

A camera is often the main sensor of autonomous robots. As those embedded platforms offer minor computing power only, it is a challenge to provide a fast and robust image processing. This paper describes a new real-time structure preserving noise reduction operator based on the so-called SUSAN filtering approach. We use a correlation function to determine which part of a pixel's neighborhood has to be included in the smoothing process. Thus, the smoothing process takes place only inside homogeneous regions of the image, without blurring edges and bi-dimensional features which are needed for object recognition.

1 Introduction

With the recent development of autonomous mobile applications, embedded platforms have become a viable and convenient option for robotics (e.g. Sony ERS-7).However, embedded robotic platforms are severely limited in terms of processing resources, due to the space and power constraints which are a consequence of autonomous operation. Furthermore, low power consumption cameras can exhibit significant amounts of noise in the images they capture [1]. Linear noise reduction convolution operators, such as Gaussian smoothing filters, are relatively inexpensive in terms of computing power. Yet, they significantly alter crucial features for computer vision such as edges and corners. Non-linear operators which selectively determine what part of the convolution kernel to include in the smoothing process, exhibit satisfactory performance in terms of image structure preservation while effectively reducing noise [2]. On the other hand, as we will show, the computational cost of such filters makes them not suitable for real-time applications on embedded platforms.

*This work was supported by the Deutsche Forschungsgemeinschaft DFG under program of emphasis SPP 1125

2 Noise Reduction Filtering

Conventionally, noise reduction is done by cutting the high frequency components of an image. Thus, all relevant information of the high frequency part of the spectrum of the image is lost. This is why non-linear operators try to preserve image structures by selectively determine which part of a pixel's neighborhood has to be included in the smoothing process.

2.1 SUSAN Noise Filtering

The SUSAN noise reduction filter achieves this by applying a correlation function to calculate the similarity of the brightness of a pixel to be filtered with a neighborhood defined by a fixed convolution mask [2]:

$$c(p, p_0) = e^{-\left(\frac{I(p)-I(p_0)}{t}\right)^2} \tag{1}$$

In Equation 1 $c(p, p_0)$ represents the Gaussian correlation function, where the so-called nucleus $p_0 = (x_0, y_0)$ is the pixel which is going to be filtered, $p = (x, y)$ is a pixel which belongs to the convolution mask around the nucleus, and t is the brightness threshold which controls the width of the Gaussian. In the spatial domain, the SUSAN filter also makes use of a Gaussian weighing. Accordingly, the overall equation of the filter is given by Equation 2:

$$I'(p_0) = \frac{\sum_{p \neq p_0} I(p) \cdot e^{-\frac{r^2}{2\sigma^2} - \left(\frac{I(p)-I(p_0)}{t}\right)^2}}{\sum_{p \neq p_0} e^{-\frac{r^2}{2\sigma^2} - \left(\frac{I(p)-I(p_0)}{t}\right)^2}} \tag{2}$$

Here, $I(x, y)$ is the brightness of a pixel before filtering, $I'(x, y)$ is the filtered brightness , and σ determines the spatial Gaussian weighing.In case the denominator of such a function for a given pixel is zero, it means that its whole neighborhood is uncorrelated with it, and hence it's treated as pulse noise. This is dealt with by using as an estimate for the pixel brightness the *median* of its 8 closest neighbors [3].

2.2 Going Real-Time

An efficient implementation of such a filter is to precalculate the non-linear correlation function for a chosen t in a look-up-table L. Using the Sony ERS-7 as a reference implementation, the total size of such a table is 2044 byte. In the following, this will be referred to as

$$L_{corr.}[\Delta I] := exp\left(-\left(\frac{\Delta I(x,y)}{t}\right)^2\right) \qquad (3)$$

Further, it has to be specified the spatial scale of the smoothing σ, which can be also precalculated and stored in the convolution mask of neighbors:

$$L_{mask}[\Delta x][\Delta y] := exp\left(-\frac{(\Delta x)^2 + (\Delta y)^2}{2\sigma^2}\right) \qquad (4)$$

Now for each pixel in the mask of neighbors, applying Equation 2 can be done at the cost of 2 look-ups, a division and several multiplications and additions(depending on the mask size).

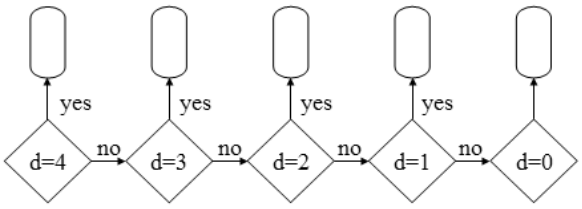
If the denominator of Equation 2 equals zero, the eight closest neighbor's brightnesses have to be put in an array which has to be sorted, and the 2 median values averaged. Due to efficiency reasons, we recommend the use of the insertion sort algorithm [4].

In all other cases, the division is the most complex and expensive operation in terms of execution time, followed by the multiplication [5, 6].

In order to avoid one source of multiplications involved in the original algorithm, we chose to replace the correlation function with a $rect_{2\tau}(p, p_0)$: having a sharper cutoff, the optimal threshold σ is somehow more dependent from the amount of noise present in the images ([2]), however having a binary co-domain for the function means that we can represent it with integer values, and use the smallest data type offered by the processors for that purpose, which is the byte, reducing the size of the look up table to $\frac{1}{4}$ of the original, for a total of 511 byte. This can be further reduced, in case of processors with very limited cache amounts, by introducing a quantization $n : 1$ (with $n = 2^i$, works well for $i = 1, 2$) in the domain of the function, which has the only negative effects of an increase of granularity of the threshold value (which can then assume only values $mod_n(t) = 0$) and require an additional shift operation of ΔI by i positions. The multiplication can be avoided, by using $c(p, p_0) = -rect_{2\tau}(p, p_0) \in \{0, -1\}$ whose domain is represented in signed byte format respectively as 00000000 and 11111111, hence the correlation weighing can be performed with a simple bitwise logical operation $AND\,\langle I(p), c(p, p_0)\rangle$. The remaining multiplications can be avoided by approximating the original Gaussian spatial weighing with a special mask that is illustrated by Figure 2. Then, no spatial weighing multiplications are required, because the neighbors whose weight is 0 in the mask are simply not included in the numerator and denominator sums n and d. As a result, let $M = \{(1, 1); (1, -1); (-1, 1); (-1, -1)\}$ represent the neighbors considered by the algorithm, the equation of the new filter is:

$$I'(p_0) = \frac{\sum\limits_{p\in M} I(p) \cdot rect_{2\tau}\left(I(p) - I(p_0)\right)}{\sum\limits_{p\in M} rect_{2\tau}\left(I(p) - I(p_0)\right)} \qquad (5)$$

In order to speed-up the division operation, we first have to note that the denominator d of such a function can assume only 5 possible values: $d \in \{0, 1, 2, 3, 4\}$, which represents how many neighbors are "similar" to the nucleus for a given pixels. The statistical incidence of each case is dependent on the threshold τ of the correlation function: obviously, for $\tau = 0 \overset{\forall(x,y)}{\Rightarrow} d(x, y) = 0$, while for $\tau = 255 \overset{\forall(x,y)}{\Rightarrow} d(x, y) = 4$. Using a sample of 100 images captured from the on-board camera of our robot (Aibo ERS7) from real-world situations, we have measured that for values of $\tau \geq 6$ case $d(x, y) = 4$ totally dominates with more than 80% of occurrences (which become $> 91\%$ for $d(x, y) = 10$): following the criterium of making the common case fast, we have replaced the division with a series of nested conditional branches, such that in the most common case only one conditional branch is performed, followed by the second most common with 2, and so on.



The following case differentiation shows, what oper-

ations should performed according to the denominator value d in order to achieve optimized processing speed:

- d=4: replace division by a shift-right instruction by two positions: $I'(x,y) = n(x,y) >> 2$
- d=3: approximation: add nucleus to numerator and divide by four $I'(x,y) = (n(x,y) + I(x,y)) >> 2$
- d=2: replace devision by a shift-right instruction by one bit $I'(x,y) = n(x,y) >> 1$
- d=1: nothing to divide
- d=0: perform median filtering: just discard the maximum and minimum, then average the remaining.

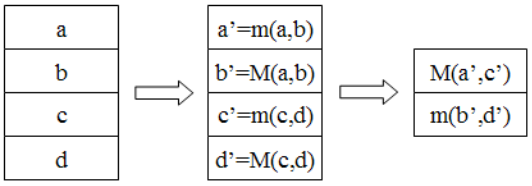


Figure 1: Calculating the median. $\{a, b, c, d\}$ is the set of neighbours, $m(x,y)$ stands for the minimum and $M(x,y)$ for the maximum of elements x and y, the 2 final values have to be averaged.

3 Performance Analysis

To better evaluate the performance of our new approach, we have separated our tests into two categories: structure preservation in noisy images, and run-time measurements. The latter is particularly important for our application domain, since the filter is meant to be used on a robot with limited computational resources in a dynamic and competitive environment, where it has to quickly react to the changing situations. As a reference, we have compared our filter, that here will be referred to as "SUSAN RealTime", with the following operators:

- the original SUSAN, presented with the smallest mask suggested by its authors (3×3), in order to minimize its running time;

- the Gaussian, with mask size also 3×3, in the most popular and computationally efficient variant, as shown in Figure 2.

0	1	0
1	0	1
0	1	0

1	2	1
2	4	2
1	2	1

Figure 2: Masks of neighbors, integer approximation of the original gaussians. Left: SUSAN RealTime mask. Right: Gaussian with $\sigma^2 \approx 0.64$.

3.1 Running Time

We measured the running time of each filter over 100 images captured by the camera of the robots: on the robot Sony Aibo ERS7 the camera has a resolution of 208×160 pixel, 24 bit per pixel. In all cases, the filters were running in normal conditions of operations, thus in parallel with other system threads. While the SUSAN RealTime execution time is dependent on the brightness threshold used, we have empirically derived the optimal for the noise present in the images ($\tau = 14$), and we have verified that its speed is only $\approx 1\%$ slower than the best case which is ($\tau = 255$).

ERS7 576MHz, 208×160 pixel image			
Filter	*Average(ms)*	*Min*	*Max*
SUSAN RT $\tau = 14$	9.904	9.8	10
SUSAN $\tau = 12$	50.796	50.65	50.95
Gaussian $\sigma^2 \approx 0.64$	5.864	5.75	6

As can be seen, the SUSAN RT is ≈ 5 times faster than the original algorithm, and when this uses the same mask, it's still ≈ 3 times faster; even on the slowest processor (ERS210(A)), the new algorithm can process more than 50 frames per second, and having a target rate of 20fps, there are about 32 ms per frame left for other tasks. Compared to the gaussians, it's between 1.5 and 2.2 times slower, which is a very good result for a non-linear filter. A more detailed comparison can also be found in [7].

3.2 Noise Reduction

To evaluate the noise reduction and structure preservation capabilities, we have followed a similar approach as in the original SUSAN article ([2]), but with a notable exception: the results will be measured after a single application of each filter, this because in our domain the total execution time is critical, making unfeasible a repeated iteration until the error variance reaches 0. For each filter which requires to determine a threshold, we have used in all tests the value which provided the overall best score: using different thresholds in different tests would yield higher scores,

but this doesn't reflect the real usage, since several kinds of noise are present simultaneuosly inside real images, and the amount of edges and bi-dimensional structures is varying continuously depending on the objects present in the scene.

Edge Preservation, 10% of Noise, final gradient			
Filter	*Gaussian*	*Pulse*	*Uniform*
SUSAN RT $\tau = 24$	55.51	53.45	54.44
SUSAN $\tau = 19$	55.09	51.56	54.35
Gaussian $\sigma^2 \approx 0.64$	34.72	33.54	34.03

Table 1: The reference image is a perfect step edge of 300 pixel length, with gradient of ($\Delta = 55$). Higher numbers mean a better score.

Corner Preservation, 10% of Noise, final σ			
Filter	*Gaussian*	*Pulse*	*Uniform*
SUSAN RT $\tau = 24$	4.39	6.07	4.68
SUSAN $\tau = 19$	3.77	10.93	4.33
Gaussian $\sigma^2 \approx 0.64$	4.60	10.77	4.78

Table 2: The reference image is 102×102, with a uniform brightness of 112, which contains 100 squares of size 5×5 and brightness 135, so the feature strength is ($\Delta = 25$). Lower numbers mean a better score.

For all the experiments we have made, it must be noted that the SUSAN RealTime is performing better than the original in presence of pulse noise: this is a consequence of having a smaller mask, so that the chances of finding two similar noise spikes inside it are lower, and because of the sharper correlation function. The gaussian filter performed badly, reducing the strength of edges and corners; the original SUSAN overall provided the best filtering, however the new SUSAN RealTime scored very close to it, and proved to be very well balanced in all test conditions.

4 Results

We have used the SUSAN RealTime filter in the image processor of our RoboCup team *Microsoft Hellhounds* (which is part of the GermanTeam that won RoboCup 2004 in Lisbon) to take part at several RoboCup competitions: GermanOpen 2004, AustralianOpen 2004, AmericanOpen 2004, and JapanOpen 2004. The average running time of the whole image processor, on the Sony ERS7 robot, was 17ms for a frame processing rate of 28.5 fps, which was more than adequate to track fast moving objects.

The performance of the suggested approach is analyzed both in terms of noise filtering and structure preservation as well as in terms of running time on different CPU architectures. We proved that this filter combines the benefits of non linear structure preserving operators with processing times comparable to linear ones. We presented a practical application for the RoboCup Four-Legged Soccer League, which was used by the Microsoft Hellhounds. Furthermore, we showed that the limits of this approach (the lack of scalability in terms of noise filtering due to the spatial constraints and the rippling behavior in the high frequency range) cannot be overcome with traditional approaches as well within the typical processing constraints of real-time applications on embedded platforms. Compared to existing algorithms, the suggested approach provides an excellent ratio between image quality and runtime behavior. As a final conclusion, this efficient algorithm for image processing helps to save processing power which can be used for less time-critical applications like AI or task scheduling.

References

[1] J. Bunting, S. Chalup, M. Freeston et.al., "Return of the NUbots! The 2003 NUbots Team Report," Tech. Rep. N.N., Newcastle Robotics Laboratory, The University of Newcastle, Australia, 2003.

[2] S. M. Smith and J. M. Brady, "SUSAN - A New Approach to Low Level Image Processing," *Int. Journal Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[3] M. Singh and P. K. Bora, "Two-Dimensional Linear Prediction Based Median Filtering," 2002.

[4] R. Sedgewick, *Algorithms*, ch. 9, pp. 120–125. Addison-Wesley, 2nd ed., 1984.

[5] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design. The Hardware/Software Interface*, ch. 4. Morgan Kaufmann, 1994.

[6] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*, ch. 3, 5, Appendix H. Morgan Kaufmann, 3rd ed., 2003.

[7] W. Nistico, U. Schwiegelshohn, M. Hebbel, and I. Dahm, "SUSAN RealTime - Structure Preserving Noise Reduction For Computer Vision," Tech. Rep. 1004, University of Dortmund, 2004.