

# SUSAN RealTime - Structure Preserving Image Noise Reduction For Computer Vision On Autonomous Robots

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

This paper describes a new real time structure preserving noise reduction operator based on the SUSAN approach to low level image processing. As in the original paper, a correlation function is used to determine which parts of a pixel's neighborhood has to be included in the smoothing process, so that 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 in computer vision. However, in this new approach modifications in the correlation function and in the spatial weighing have enabled optimizations which yielded a manifold speedup in computational time, making the new filter suitable for real-time applications.

## 1 Introduction

With the recent development of autonomous mobile applications, embedded platforms have become a viable and convenient option for robotics. A good example for this is the commercial robot series "Aibo" from Sony, which recently reached its third generation. 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 computer vision applications.

### 1.1 SUSAN filtering

SUSAN is an acronym for Smallest Univalued Segment Assimilating Nucleus, and image processing operators based on this new principle has been first presented in [3] and later in [2]. In the latter article, the authors compared this filter with a multitude of linear and non-linear operators, proving its superior performance in term of noise suppression while preserving image features which are critical for computer vision like edges and corners. While the algorithm in itself is computationally efficient, as we will show in the *Performance Analysis* section it's not fast enough to be used under real time constraints on the robotic platforms which we used to develop our research.

## 2 Real Time Noise Reduction Filtering

In contrast to linear operators, which reduce noise by cutting the high frequency components of an image by performing a discrete convolution operation with a specified mask, 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, hence they retain some of the information which is carried in the high frequency part of the spectrum of the image, which otherwise would be lost.

## 2.1 SUSAN

The SUSAN noise reduction filter [2](from here on referred to simply as SUSAN), 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:

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

In equation 1  $c(p, p_0)$  represents the gaussian correlation function, where  $p_0 = (x_0, y_0)$  is the pixel which is going to be filtered (from now on referred to as *nucleus*),  $p = (x, y)$  is a pixel which belongs to the convolution mask around the nucleus, and  $t$  is a parameter (brightness threshold) which controls the width of the gaussian. In the spatial domain, the SUSAN filter also makes use of a gaussian weighing, and the authors suggest as the minimum mask size a  $3 \times 3$  matrix. Gaussians are considered optimal for this kind of filtering, since this function is the Fourier transform of itself, so it exhibits a smooth and progressive profile in both spatial and frequency domains, in contrast with sharp cutoff functions such as the  $rect_{2\sigma}$  (*box* in 2D) which are also quite popular as noise reduction filters, but exhibit ripples in the frequency spectrum. The overall equation of the filter is:

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

In equation 2,  $I'(x, y)$  is the filtered brightness value for a pixel,  $I(x, y)$  is the brightness before filtering,  $\sigma$  determines the spatial gaussian weighing, while  $t$  is the aforementioned brightness threshold. 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, as this kind of filtering is known to provide excellent results in presence of pulse noise [4].

## 2.2 Going Real Time

To implement such a filter in an efficient way, the non-linear correlation function can be precalculated (for a chosen  $t$  which is dependent from the expected amount of image noise) and stored in a look-up table: assuming to process 24 bit per pixel images (as in most commercially available cameras), with 3 color bands, 8 bit per band, the domain is limited to 511

values ( $I(x_0, y_0) - I(x, y) \in [-255..255]$ ), while the co-domain to 256, and using 32 bit floating point *Single Precision* numbers to represent it (the smallest format defined by the IEEE 754 specifications[5]), the total size of such a table is 2044 byte; in the following this will be referred to as  $LUT_{corr.}[i]$ . Then, it has to be specified the spatial scale of the smoothing  $\sigma$ , which will be also precalculated and stored (approximated) into the convolution mask of neighbors (which we'll refer to as  $LUT_{mask}[x][y]$ ). For each pixel in the mask of neighbors, the following operations have to be performed :

- *subtraction*:  $\Delta I(x, y) = I(x_0, y_0) - I(x, y)$
- *look-up*:  $c(x, y) = LUT_{corr.}[\Delta I] = e^{-\left(\frac{\Delta I(x, y)}{t}\right)^2}$
- *mult. (correlation)*:  $b(x, y) = I(x, y) \cdot c(x, y)$
- *look-up*:  $s(x, y) = LUT_{mask}[x][y] = e^{-\frac{(x-x_0)^2 + (y-y_0)^2}{2\sigma^2}}$
- *mult. (spatial)*:  $n(x, y) = b(x, y) \cdot s(x, y)$
- *mult. (pixel weight)*:  $d(x, y) = c(x, y) \cdot s(x, y)$

To calculate the final brightness of the given pixel:

- *sum (Num.)*:  $N(x_0, y_0) = \sum_{(x, y) \neq (0, 0)} n(x, y)$
- *sum (Denom.)*:  $D(x_0, y_0) = \sum_{(x, y) \neq (0, 0)} d(x, y)$

Now, if the denominator is not equal to zero:

- *division*:  $I'(x_0, y_0) = \frac{N(x_0, y_0)}{D(x_0, y_0)}$

Otherwise, the 8 closest neighbor's brightnesses have to be put in an array which has to be sorted, and the 2 median values averaged. For arrays of small size ( $\leq 25$  elements), *insertion sort* is the fastest algorithm [6]. Of the operations involved in the above process, the division is the most complex and expensive in terms of execution time, followed by the multiplication, as they both make use of simpler operations such as addition, shift, and bitwise logical instructions ([7], appendix H of [8]). Further, floating point instructions are more complex than integer ones (see also [5]), however the effective time required to perform an operation of one kind or another depends also from the hardware implementation. In choosing an efficient instruction selection, the number of execution units available on a target processor architecture play also an important role, beside instruction latency and overlapping<sup>1</sup>: the

<sup>1</sup>The number of cycles after which another instruction can begin to execute in the same unit

Intel NetBurst architecture, in use on the PentiumIV family of processors, can execute 4 integer additions per clock cycle, while only one division or multiplication (integer and floating point) can be executed at a time, as they share the same unit ([9]); in contrast, the MIPS R4000 can execute at most one integer and one fp instruction at a time ([10]). The look-up operations also play an important role in the overall performance, as the cost of a memory access is strongly dependent on the data location in the memory hierarchy (see [8]): ideally, the look-up tables should be small enough as to be accessible in most of the cases from the level of the hierarchy closest to the processor.

### 2.3 The New Algorithm

In order to avoid one of the 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  $\sigma$  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  $mod_n(\tau) = 0$ ) and require an additional shift operation of  $\Delta I$  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 second multiplication can be avoided by approximating the original gaussian spatial weighing with the following mask (figure1). No spatial weighing mul-

0	1	0
1	0	1
0	1	0

Figure 1: Mask of neighbors for the new algorithm, approximation of the original  $3 \times 3$  gaussian

tiplications are required, because the neighbors whose weight is 0 in the mask are simply not included in the numerator and denominator sums. 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:

$$I'(p_0) = \frac{\sum_{p \in M} I(p) \cdot rect_{2\tau}(I(p) - I(p_0))}{\sum_{p \in M} rect_{2\tau}(I(p) - I(p_0))} \quad (3)$$

In order to speed-up the division operation, we first have to note that the denominator of such a function can assume only 5 possible values:  $d(x, y) \in \{0, 1, 2, 3, 4\}$ , 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  $\tau$  of the correlation function: obviously, for  $\tau = 0 \xrightarrow{\forall(x,y)} d(x, y) = 0$ , while for  $\tau = 255 \xrightarrow{\forall(x,y)} d(x, y) = 4$ . Using a sample of 100 images captured from the on-board camera of our robot (Aibo ERS7) from real-world situations, we have measured the following incidence of cases (figure 2). As can be

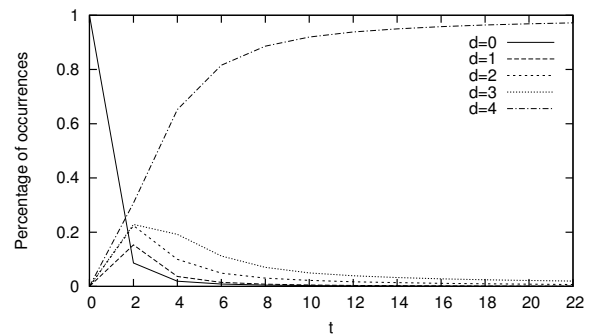
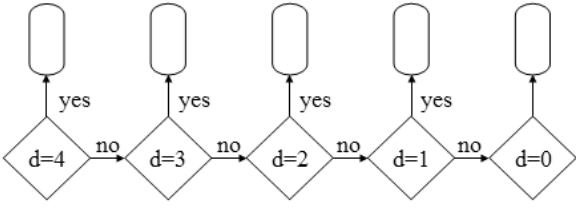


Figure 2: Occurrences of the 5 possible values that the denominator of equation 3 can assume.

seen, for values of  $\tau \geq 6$  case  $d(x, y) = 4$  totally dominates with more than 80% of occurrences: following the criterion 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. Because of the regularity of such conditional branches, even deeply pipelined out-of-order execution speculative architectures such as the PentiumIV can succeed in predicting the branching outcome in most of the cases, resulting in very little performance impact from the branches. Let's see what should be done in each case:

*case 4:* the numerator has to be divided by 4, which can be done efficiently with an instruction of shift right by 2 position:  $I'(x, y) = n(x, y) \gg 2$



0	1	0
1	8	1
0	1	0

1	2	1
2	4	2
1	2	1

Figure 4: Mask of neighbors, integer approximation of the original gaussians. Left:  $\sigma^2 \approx 0.36$ . The result must be normalized, dividing by the total weight of the mask (=12). Right:  $\sigma^2 \approx 0.64$ . Total weight of the mask = 16.

*case 3:* the numerator should be divided by 3, which cannot be accomplished with a shift; then, we add the nucleus to the numerator (since it's correlated with the other 3 points), so that we can divide by 4 using the shift right:  $I'(x, y) = (n(x, y) + I(x, y)) \gg 2$

*case 2:* the division is by 2, which can be done by shifting as:  $I'(x, y) = n(x, y) \gg 1$

*case 1:* there is nothing to divide

*case 0:* this is the special case, where a median filtering has to be performed. Having only 4 neighbors, there is no need to fully sort them, we just have to discard the maximum and minimum, then average the remaining 2.

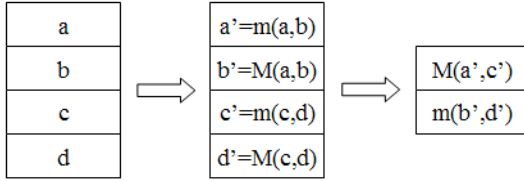


Figure 3: Calculating the median.  $\{a, b, c, d\}$  is the set of neighbours,  $m(x, y)$  stands for the minimum and  $M(x, y)$  for the maximum of elements  $x$  and  $y$ , the 2 final values have to be averaged.

### 3 Performance Analysis

In order to better evaluate the performance of our new approach, we have separated our tests into two categories: noise reduction measurements and runtime 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 ( $3 \times 3$ ), in order to minimize its running time;
- the original SUSAN, with the same mask as the SUSAN RealTime, which considers only 4 neighbors (we will refer to it as SUSAN "Lite"), to evaluate what is the impact of the new mask on the noise reduction capability as well as on the execution time;
- the Gaussian, with mask size also  $3 \times 3$ , in the 2 most popular and computationally efficient variants, as shown in figure 4;
- the Median, with mask size  $3 \times 3$ .

### 3.1 Running Time

We measured the running time of each filter over 100 images captured by the camera of the robots: for the tests conducted on the robots Sony Aibo ERS210(A), the camera has a resolution of  $176 \times 144$  pixel, 24 bit per pixel, while the robot Sony Aibo ERS7 has a resolution of  $208 \times 160$  pixel and the same images have been also processed on a desktop cpu, a 2.66GHz Intel PentiumIV. In all cases, the filters were running in normal conditions of operations, thus in parallel with other system threads. To evaluate the impact of our common-case optimization, we have included in the test the SUSAN RealTime in 3 variants: normal, with  $\tau = 14$  which we experimentally derived to be optimal for the noise present in the images of our robot's camera (the same applies for the normal SUSAN, however the threshold in this case has no performance impact);  $\tau = 0$  to test the worst case scenario, and  $\tau = 255$  best case; both these extremes have no practical use.

As can be seen, the SUSAN RT is  $\approx 5$  times faster than the original algorithm, and when this uses the same mask, it's still  $\approx 3$  times faster; even on the slowest processor (ERS210(A)), the new algorithm can process more than 50 frames per second, and having

PentiumIV 2.66GHz, 208 × 160 pixel image			
<i>Filter</i>	<i>Average(ms)</i>	<i>Min</i>	<i>Max</i>
SUSAN RT $\tau = 0$	2.6872	2.5	2.82
SUSAN RT $\tau = 14$	1.0248	0.92	1.26
SUSAN RT $\tau = 255$	0.932	0.64	0.94
SUSAN Lite $\tau = 12$	4.562	4.36	5
SUSAN $\tau = 12$	6.3304	6.24	6.58
Median	10.9004	10.3	11.88
Gaussian $\sigma^2 \approx 0.36$	0.6564	0.62	0.94
Gaussian $\sigma^2 \approx 0.64$	0.4824	0.3	0.94

ERS7 576MHz, 208 × 160 pixel image			
<i>Filter</i>	<i>Average(ms)</i>	<i>Min</i>	<i>Max</i>
SUSAN RT $\tau = 0$	14.187	14.05	14.3
SUSAN RT $\tau = 14$	9.904	9.8	10
SUSAN RT $\tau = 255$	9.883	9.75	9.95
SUSAN Lite $\tau = 12$	33.504	33.35	33.6
SUSAN $\tau = 12$	50.796	50.65	50.95
Median	42.82	42.6	42.95
Gaussian $\sigma^2 \approx 0.36$	5.089	4.95	5.2
Gaussian $\sigma^2 \approx 0.64$	5.864	5.75	6

a target processing 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.

### 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 have been 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 in the corner test

ERS210A 384MHz, 176 × 144 pixel image			
<i>Filter</i>	<i>Average(ms)</i>	<i>Min</i>	<i>Max</i>
SUSAN RT $\tau = 0$	27.505	27.35	27.6
SUSAN RT $\tau = 14$	18.215	18.1	18.4
SUSAN RT $\tau = 255$	18.071	18	18.25
SUSAN Lite $\tau = 12$	52.98	52.85	53.15
SUSAN $\tau = 12$	80.875	80.65	81.2
Median	79.917	79.2	80.3
Gaussian $\sigma^2 \approx 0.36$	8.285	8.15	8.4
Gaussian $\sigma^2 \approx 0.64$	11.384	11.25	11.5

(which represents a situation that includes edge preservation and noise filtering), giving the same weight to all noise types: 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 simultaneously inside real images, and the amount of edges and bi-dimensional structures is varying continuously depending on the objects present in the scene. As can be seen from table 1, the SUSAN RealTime

Noise Reduction, 10% of Noise, final $\sigma$			
<i>Filter</i>	<i>Gaussian</i>	<i>Pulse</i>	<i>Uniform</i>
SUSAN RT $\tau = 24$	3.66	5.10	3.99
SUSAN $\tau = 19$	2.65	9.17	3.29
SUSAN Lite $\tau = 21$	3.35	6.87	3.98
Median	2.73	0.54	4.05
Gaussian $\sigma^2 \approx 0.36$	4.9	15.55	5.38
Gaussian $\sigma^2 \approx 0.64$	3.25	10.13	3.57

Table 1: The reference image is 208 × 160, with a uniform brightness of 128. Lower numbers mean a better score.

provides good results in terms of noise filtering on all the 3 types of noise which have been used; however, the 3 operators which are using only 4 neighbors for filtering (SUSAN RT, SUSAN Lite and Gaussian  $\sigma^2 \approx 0.36$ ) have more limited smoothing capabilities than those which consider 8 neighbors, which is what we would expect from signal theory. On the other hand, increasing the mask size has a negative effect on the runtime performance, and we have already shown that the original SUSAN with a mask size of 3 × 3 is too slow for our requirements.

Edge Preservation, 10% of Noise, final gradient			
<i>Filter</i>	<i>Gaussian</i>	<i>Pulse</i>	<i>Uniform</i>
SUSAN RT $\tau = 24$	55.51	53.45	54.44
SUSAN $\tau = 19$	55.09	51.56	54.35
SUSAN Lite $\tau = 21$	55.1	50.87	53.96
Median	43.16	46.03	41.04
Gaussian $\sigma^2 \approx 0.36$	46.46	46.26	45.19
Gaussian $\sigma^2 \approx 0.64$	34.72	33.54	34.03

Table 2: The reference image is a perfect step edge of 300 pixel length, with gradient of ( $\Delta = 55$ ). Higher numbers mean a better score.

In all the experiments we have made, 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

Corner Test, 10% of Noise, final $\sigma$			
<i>Filter</i>	<i>Gaussian</i>	<i>Pulse</i>	<i>Uniform</i>
SUSAN RT $\tau = 24$	4.39	6.07	4.68
SUSAN $\tau = 19$	3.77	10.93	4.33
SUSAN Lite $\tau = 21$	4.02	8.00	4.61
Median	5.81	5.52	6.43
Gaussian $\sigma^2 \approx 0.36$	5.16	15.85	5.56
Gaussian $\sigma^2 \approx 0.64$	4.60	10.77	4.78

Table 3: The reference image is  $102 \times 102$ , with a uniform brightness of 112, which contains 100 squares of size  $5 \times 5$  and brightness 135, so the feature strength ( $\Delta = 25$ ) is low compared to the amount of noise. Lower numbers mean a better score.

of the sharper correlation function (which is the reason why the SUSAN RT performs even slightly better than the SUSAN Lite). The two Gaussian filters performed badly, reducing the strength of edges and corners ( $\sigma^2 \approx 0.64$ ) and/or filtering very little noise ( $\sigma^2 \approx 0.36$ ); the Median performed respectably, however we have shown that this filter is slow, and falls clearly behind the SUSAN variants in terms of image structure preservation. The original SUSAN overall provided the best filtering, however the new SUSAN RealTime scored very close to it using only a small fraction of the computational time, and proved to be very well balanced in all test conditions.

## 4 Application

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 to 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.

## 5 Summary

The Performance of the suggested approach has been analysed 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 compara-

ble to linear ones. We presented a practical application for the RoboCup Four-Legged Soccer League, which has been used by the Microsoft Hellhounds - a member of the GermanTeam. Furthermore, we showed that the limits of this approach (the lack of scalability in terms of noise filtering due to the spatial constraints) cannot be overcome with traditional approaches as well without significantly increasing the computational time. Compared to existing algorithms, the suggested approach provides an excellent ratio between image quality and runtime behavior. As a final conclusion, this efficient algorithm 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., 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] S. Smith, "Flexible filter neighbourhood designation," in *Proc. 13th Int. Conf. on Pattern Recognition*, vol. 1, pp. 206–212, 1996.
- [4] M. Singh and P. K. Bora, "Two-dimensional linear prediction based median filtering," 2002.
- [5] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.
- [6] R. Sedgewick, *Algorithms*, ch. 9, pp. 120–125. Addison-Wesley, 2nd ed., 1984.
- [7] D. A. Patterson and J. L. Hennessy, *Computer Organization & Design. The Hardware/Software Interface*, ch. 4. Morgan Kaufmann, 1994.
- [8] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*, ch. 3, 5, Appendix H. Morgan Kaufmann, 3rd ed., 2003.
- [9] Intel corp., *IA-32 Intel Architecture Optimization*, 2003.
- [10] J. Heinrich, *MIPS R4000 Microprocessor User's Manual*, ch. 2, 6. MIPS Technologies Inc., 1994.