Edge Detection Using Streaming SIMD Extensions On Low Cost Robotic Platforms

Matthias Hofmann, Fabian Rensen, Ingmar Schwarz and Oliver Urbann*

***Abstract*— Edge detection is a popular technique for extracting features on images. To this end, the Sobel operator is one of the most widely used methods. However, building efficient applications that make use of Sobel on systems with restricted computational power is difficult. This is due to the fact that Sobel requires runtime-intensive computations such as the two-dimensional convolution. Hence, the goal of this work is to significantly speed up the computation time that is required for the Sobel operation. Thus, we use Streaming SIMD Extensions that define specific assembler commands. We compare the runtime of the SSE implementation with a standard implementation which does not make use of SSE. Our experiments show a relevant boost of performance.**

I. Introduction and Related Work

Processing image data on high frame rates plays a vital role in various domains, e.g. robot soccer. Efficient algorithms are especially important for applications with restricted computational power, e.g. low cost robotics platforms. For our work, we use the NAO robot from Softbank[1]. Our work aims to fulfill the soft real-time capability as best as possible. This means that the computation of two parallel HD images on 30 Hertz is desirable[2], but not critical if the algorithm does not provide results after 33ms. The algorithm is executed by the processor along with other algorithms, e.g. for artificial intelligence, motion control, and localization. These parts may require significant runtime in robotic applications as well.

Thus, it is desirable to minimize the runtime of the image processing system. Edge detection is a widely used approach for extracting features from the individual image. There is a vast amount of edge detectors, e.g. the Sobel operator [1], the Canny edge detection [2], and the Prewitt operator [3]. In the domain of robot soccer, these features may be used for further processing, i.e. detecting the ball, recognizing other robots, or extracting field lines. Many algorithms, such as the Hough transformation [4], or the Harris corner detection [5] are based on edge images. In the domain of smart vehicles, Gavrila and Philomin describe an image processing system based on edge detection [6] that is used to detect pedestrians. Since autonomous driving is a critical task, the system may even fulfill the hard real-time constraint. This means that if there is no result of an algorithm after a specific time frame, the system may cause a critical incident. For this reason, performing a runtime optimization is a reasonable approach. Kim et. al [7] describe such an optimization with respect to specific hardware architectures.

This paper is structured as follows: Section II describes the basics of SSE and the Sobel operator. A detailed explanation of our implementation is given in section III. Section IV shows our results and future work is outlined in section V.

II. Fundamentals

A. SSE in detail

SSE is able to operate on multiple data at the same time. For this purpose, it introduces eight new processor registers of one type, named XMM0 to XMM7. In comparison to its predecessor Multi Media Extension (MMX), SSE registers may hold floating point numbers. The size of each register is 128 bit[3], and stores one of the following combinations of different data types:

- 2 floating point numbers (each 64-bit, double precision)
- 4 floating point numbers (each 32-bit, single precision)
- 4 integers (each 32-bit, signed or unsigned)
- 8 integers (each 16-bit, signed or unsigned)
- 16 integers (each 8-bit, signed or unsigned)

Moreover, SEE defines the following set of standard commands that perform calculations on the registers:

- load and store
- pack and unpack
- logical operations (i.e. AND, NAND, OR, XOR)
- arithmetical operations (i.e. bit shifting, addition, average, maximum, minimum, multiplication, division, subtraction)
- comparison operations (i.e. equal, greater, less, greater equal, etc.)

Since *SSE* provides assembler commands, various compilers provide a set of *SSE* intrinsics. To make use of these functions in C++, one of the standard headers (Table I) must be included in the source code. The headers are backward-compatible. For a full include graph refer to the Clang documentation.[4].

For our study, we use *SSSE3* on the NAO v5 robot by Softbank robotics[5]. Most compilers (e.g. GCC, Clang, MSVC) perform optimization steps while translating the code into the corresponding *SSE* commands. Therefore, the resulting assembler code may differ from the C++ code.

*Robotics Research Institute, TU Dortmund University, Germany <forname.surname>@tu-dortmund.de

[1]https://www.ald.softbankrobotics.com/en/cool-robots/nao

[2]The NAO camera provides at most 30 images a second.

[3]Recent Intel processors support Advanced Vector Extensions 256-bit registers. In our study, we use the NAO robot that offers 128-bit SSE registers.

[4]http://clang.llvm.org/doxygen/index.html

[5]https://www.ald.softbankrobotics.com/en/cool-robots/nao

TABLE I

COMPILER INTRINSICS AND CORRESPONDING C++ HEADERS.

Header file	Version
mmintrin.h	MMX
xmmintrin.h	SSE
emmintrin.h	SSE2
pmmintrin.h	SSE3
tmmintrin.h	SSSE3
smmintrin.h	SSE4.1
nmmintrin.h	SSE4.2
ammintrin.h	SSE4A
wmmintrin.h	AES
immintrin.h	AVX
x86intrin.h	All available intrinsics (Clang and GCC, x86)
intrin.h	All available intrinsics (MSVC)

B. Sobel Operator

The Sobel operator approximates the gradient (i.e. the first derivative) of the image intensity by calculating the 2-dimensional discrete convolution of a filter mask, and the image intensity values. There is a vast of amount of similar algorithms that utilize different filter masks, e.g. the Prewitt operator [3]. The filter masks of the Sobel operator are shown in equations 1 (horizontal) and 2 (vertical) [1].

$$M_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \tag{1}$$

$$M_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix} \tag{2}$$

For each pixel, the sum of the intensity values are multiplied by the corresponding values of the filter masks. Let I denote the 2-dimensional image. This way, the 2-dimensional convolution can be written as

$$G_x = M_x * I \tag{3}$$

$$G_y = M_y * I \tag{4}$$

If we combine the horizontal and vertical directions, the gradient magnitude is:

$$G = \sqrt{G_x^2 + G_y^2} \tag{5}$$

III. SSE IMPLEMENTATION OF THE SOBEL OPERATOR

An implementation of the Sobel operator in SSE requires loading the image into the SSE registers and computing the convolution. The following subsections cover the procedures and our implementation in detail.

A. Loading the Images Using SSE

We use the YUV422 color space to process images provided by the camera of the NAO robot[6] since it is its native format. In this color space, two adjacent pixels share the color information, but each pixel has an individual intensity value. Figure 1 shows the storage of the image in the main memory.

We exclusively load the intensity values of the pixels into the SSE registers as they serve as an input for the edge detection. For the load procedure, we utilize the SSE unpack operations. The unpack operation copies half of a first register, and half of a second register into a new allocated register in an alternating manner. Figure 2 visualizes the unpack operation for 8-bit interpreted registers.

We use the operations to deinterlace the YUV values, i.e. the intensity values are stored consecutively, and in the desired order. To this end, we load two successive chunks of 128 bit image data into two SSE registers, and perform four lower and three upper[7] unpack operations. We save the results of the operations in two registers: One contains the intensity values, and the other contains the color information. Figure 3 depicts the register contents during each unpack step.

B. Calculating the Convolution

We subsequently calculate the convolution. To this end, we load sixteen values into each individual register. Since the intensity values range from 0 to 255 (unsigned 8-bit), it is required to conduct an approximation to prevent overflows. The following step is executed for horizontal edge detection for every pixel $P_{i,j}$ at position (i, j):

$$\begin{aligned} G_x(P_{i,j}) = \; & P_{i+1,j-1} + 2 \cdot P_{i+1,j} + P_{i+1,j+1} \\ & - P_{i-1,j-1} - 2 \cdot P_{i-1,j} - P_{i-1,j+1} \end{aligned} \tag{6}$$

It follows that the value of the sums range from $-4 \cdot 255$ to $4 \cdot 255$. For edge detection, only the gradient magnitude is relevant. Thus, we split the sums into a negative and a positive part. Consequently, the two partial sums now range from 0 to $4 \cdot 255$ which makes dividing the initial mask by 4 sufficient to prevent from overflows.

This way, equations (1) and (2) become

$$M'_x = \frac{1}{4} M_x = \begin{pmatrix} -\frac{1}{4} & 0 & +\frac{1}{4} \\ -\frac{1}{2} & 0 & +\frac{1}{2} \\ -\frac{1}{4} & 0 & +\frac{1}{4} \end{pmatrix} \tag{7}$$

$$M'_y = \frac{1}{4} M_y = \begin{pmatrix} -\frac{1}{4} & -\frac{1}{2} & -\frac{1}{4} \\ 0 & 0 & 0 \\ +\frac{1}{2} & +\frac{1}{2} & +\frac{1}{4} \end{pmatrix} \tag{8}$$

There is another restriction imposed by SSE. It performs computations in a strictly vertical manner, e.g. an addition is only possible between two values which are located at the same indexes of two individual registers. But equation (6) reveals that the computation for one resulting pixel requires access of eight neighbored values. We solve the problem by loading three rows of data[8], and then shifting these rows according to the convolution. Figure 4 visualizes the very procedure for the horizontal case. SSE offers bit-shifting on

[6] http://doc.aldebaran.com/2-1/family/robots/video_robot.html

[7] Only three higher unpacks are necessary since we do not evaluate the color information.

[8] One for each matrix row.

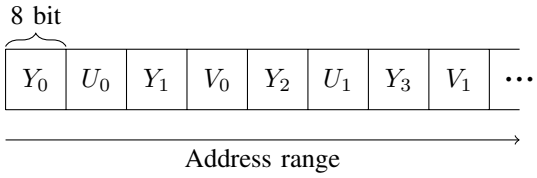

Fig. 1. The robot's image in the main memory.

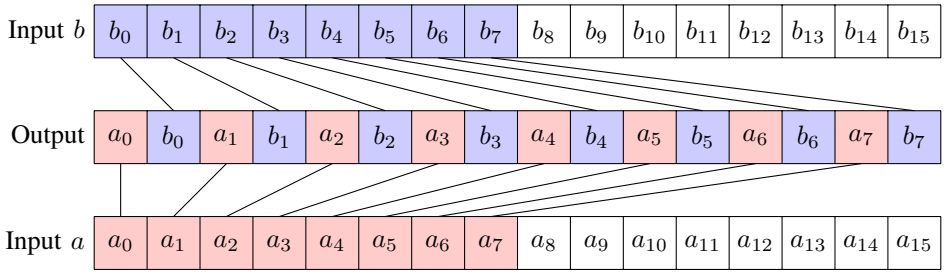

Fig. 2. Unpack lower halves for 8-bit values (`_mm_unpacklo_epi8(a,b)`).

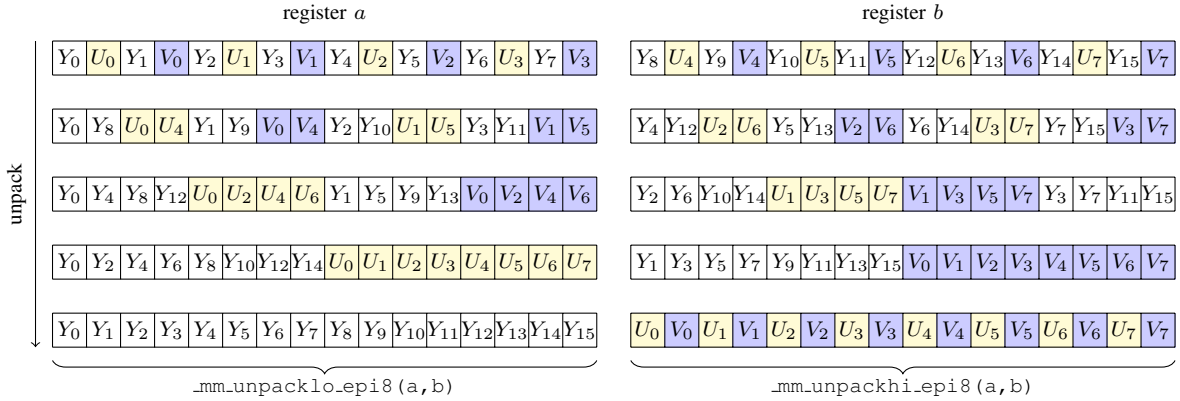

Fig. 3. Deinterlacing of YUV422 pixels using *SSE unpack* operations.

register level for some register sizes. However, we need to shift the register by the number of bits of one value (i.e. 8-bit). Since SSE does not provide this kind of bit-shifting operation, we implemented it.

Figure 5 further explains the calculation with bit shifted registers. In this regard, P_1 is the first element of the non shifted register, P_2 is the first element of the register shifted by one, and P_3 is the first element of the register shifted by two. If the registers are divided by $1/2$ and $1/4$ respectively (according to equations (7) and (8)) and cumulated, and all registers are cumulated, the resulting register contains the desired sums. A similar procedure using different shifts is necessary for the vertical sums.

The last step is the calculation of the magnitude of the gradients. This operation requires two power, and one square root operation. Due to integer intrinsics, it is not possible to calculate the exact values. Moreover, this would create another overflow issue. Hence, we need to perform an approximation according to Griffin [9]:

[9] `http://www.dspguru.com/dsp/tricks/magnitude-estimator`

$$\sqrt{G_x^2 + G_y^2} \approx \alpha \cdot \max(G_x, G_y) + \beta \min(G_x, G_y) \quad (9)$$

Choosing $\alpha = 1$ and $\beta = 0.25$ gives a largest error of approximately 11.61%, and a mean error of approximately 0.65%. Parameter studies for α and β yield better results, but setting $\alpha = 1$ does not impose a register to be altered, and $\beta = 0.25$ is merely bit shifted by two. This way, the magnitude estimation replaces two power and one square root operations with one maximum, one minimum, and one bit shift operation while still being close to the exact value.

IV. Results

We expect that the SSE approach accelerates the Sobel operator by more than 14 times compared to the sequential approach. That hypothesis follows from the simultaneously computation of 14 values, and the approximation of the gradient magnitude, from which we expect to be computed faster than the more complex square root, and exponentiation operations. Moreover, the classic approach does not make use of pointer arithmetic.

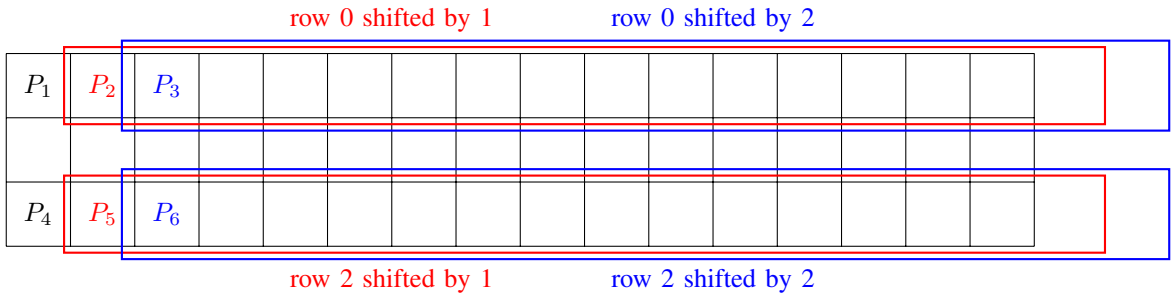

Fig. 4. Row shifting used for calculating the convolution, horizontal sum. P_1 to P_6: Pixel intensity values loaded into *SSE* registers.

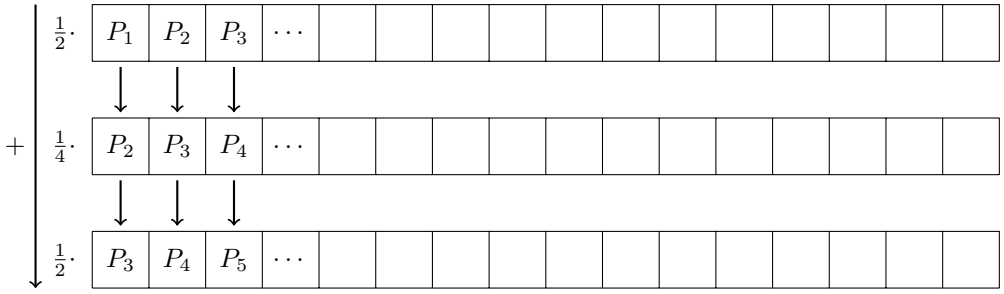

Fig. 5. The three resulting registers from bit shifting. The top row can be used to calculate the correct sum without using random access.

TABLE II
RESULTS OF THE TIMING ANALYSIS ON A NAO V5

Approach	Minimum (ms)	Maximum (ms)	Average (ms)	Frequency Avg (fps)
Classic	370.157	379.786	375.283	2.435
8-bit *SSE*	9.490	11.166	10.234	29.654

The performance evaluation was executed on the NAO robot during a test game of two minutes. To this end, a profiling analysis with respect to the SSE Sobel implementation was conducted. Table II shows the timing results for processing the Sobel image based on the image of the upper camera (1280×960 intensity values) of the robot.

It is cognizable that our SSE Sobel implementation speeds up the computation by more than 14 times. This makes it easier to calculate an edge image with respect to soft real-time constraint. With an average runtime of 10.234 ms per frame, and a frequency of 29.654 (which includes the overall runtime of the cognition frame per second), we are now able to compute at least all images provided by a single (either upper, or lower) camera. Figures 6 and 7 are example images: The first one is processed without SSE Sobel while the second one is processed with our implementation. There is no visual difference between the images.

Our implementation makes it possible to perform the Sobel operation on specific regions of the image. Moreover, it is possible to calculate edge images by utilizing the horizontal or vertical filter mask exclusively. This is needed for algorithms such as the Harris corner detection [5]. Our implementation is publicly available at `https://github.com/NaoDevils/SSESobel`.

V. FUTURE WORK

It is possible to transfer our approach to cutting-edge processors that use more recent SSE versions, and larger registers. Using a processor with 256-bit registers may result into a doubling of performance of the Sobel operation. Another idea for future work is to pre-calculate a matrix containing the computation results of the gradient magnitude. This would replace the magnitude estimation (see equation 9) by a lookup table. However, the disadvantage is that it would consume more main memory.

REFERENCES

[1] D. Luo, *Pattern Recognition and Image Processing*. Amsterdam: Elsevier, 1998.

[2] Y. Luo and R. Duraiswami, "Canny edge detection on nvidia cuda," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, June 2008, pp. 1–8.

[3] J. M. S. Prewitt, "Object enhancement and extraction."

[4] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972. [Online]. Available: http://doi.acm.org/10.1145/361237.361242

[5] C. Harris and M. Stephens, "A combined corner and edge detector," in *In Proc. of Fourth Alvey Vision Conference*, 1988, pp. 147–151.

[6] D. M. Gavrila and V. Philomin, "Real-time object detection for smart vehicles," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, 1999, pp. 87–93 vol.1.

[7] C. G. Kim, J. G. Kim, and D. H. Lee, "Optimizing image processing on multi-core cpus with intel parallel programming technologies," *Multimedia Tools and Applications*, vol. 68, no. 2, pp. 237–251, 2014. [Online]. Available: http://dx.doi.org/10.1007/s11042-011-0906-y

Fig. 6. Example image that is computed by Sobel without using SSE extensions.

Fig. 7. Example image that is computed by Sobel by using SSE extensions.