Correction of Lens-Distortion for Real-Time Image Processing Systems

Using third-order transformation-polynomials and a 4x4 cubic-spline interpolation

*A.G.J. Nijmeijer, M.A. Boer, C.H. Slump, M.M. Samsom, M.J. Bentum,
G.J. Laanstra, H. Snijders, J. Smit, O.E. Herrmann*

Abstract

This paper presents the design of a real-time imaging system, with incorporated the correction of lens-distorted images. It may be used in medical applications (e.g. real-time X-ray image intensifiers), industrial robot vision products or consumer electronics. The system contains two different VLSI-circuits: a transformer and an interpolator. The transformer calculates an address that points to a pixel in the input image. Since this address hardly ever is an exact pixel position, a cubic-spline interpolator is used to calculate the pixel-intensity at the desired position.

1. INTRODUCTION

In the design of a real-time image processing system for low-light intensities with good image quality, two important factors play a role. On one hand, a distortion-free image is required; on the other hand, a high light-efficiency is desired. Lenses that meet these requirements, are complex and therefore expensive. Cheaper (and simple) lenses have a high light-efficiency, but suffer from distortion [1]. If distortion can be corrected in real-time, a better price/performance is accomplished.

Figure 1. Three kinds of distortion.

Distortion occurs, when the size of each point in the image plane is magnified in a different way. Distortions do not blur the image and, therefore can be corrected digitally. Lens-distortion is rotation-symmetric: the magnification varies with distance to the optical axis.

There are two kinds of distortion, isotropic and anisotropic distortion. When (in the first case) magnification increases from the axis, this aberration looks like a pincushion, otherwise it looks like a barrel (see Figure 1). The second, anisotropic case, is also referred to as pocket- handkerchief distortion. This distortion occurs in electron optics, if magnetic lenses are used.

To realize a real-time image processing system which can correct this kind of distortion, the system configuration (depicted in Figure 2) is proposed.

Figure 2. System Overview.

This Figure shows that the camera-picture is (digitally) saved in a double image-buffer. The image is corrected for distortion in the transformation block, using the parameters calculated in the parameter extraction block (software). The corrected image is stored in a double buffer and displayed after D/A conversion. The transformation can be divided into two parts: a transformer and an interpolator. The theory of both parts will be discussed briefly in the next section.

2. THEORY

Transformation

Distortion is a 3^{rd} order aberration [2] [3], which can be written in the form:

$$\begin{aligned}\Delta x &= r^2(D.u - d.v) \\ \Delta y &= r^2(D.v + d.u)\end{aligned}\quad (1)$$

With: Δx and Δy are pixel shifts due to distortion, r is the distance from the correct position to the optical axis, D and d are distortion coefficients (constant), u and v are correct pixel positions.

So, a distorted image can be corrected by using a 3^{rd} order (2-dimensional) polynomial. The more general case is presented here:

$$\begin{aligned}x &= (a_x + e_x v + i_x v^2 + m_x v^3) + (b_x + f_x v + j_x v^2 + n_x v^3) u \\ &+ (c_x + g_x v + k_x v^2 + o_x v^3) u^2 + (d_x + h_x v + l_x v^2 + p_x v^3) u^3\end{aligned}\quad (2)$$

With: x (and y) are coordinates in the distorted input image, u and v are coordinates in the corrected output image, $a_{x/y}, \dots, p_{x/y}$ are transformation coefficients.

These formulas generate x and y positions in the input image, as a function of the transformation parameters and location in the output image.

In Eq. (3), a_x and a_y govern x - and y translation of the image. e_x and b_y take care of scaling of the output image, while e_y and b_x enable the output image to rotate. Higher order terms generate perspective, shear and higher order transformations like distortion.

Interpolation

For every pixel in the output image the transformer calculates the corresponding coordinates in the input image. Unfortunately these coordinates will about never be on gridpoints of the input image. Because only the intensities of the pixels on gridpoints are known, an interpolation technique is required to determine the intensity of intermediate points. Figure 3 shows this interpolation scheme.

Figure 3. Pixel interpolation in the output image. $P_1 \dots P_4$ are pixels known from input image, P_{int} is unknown pixel in output image ($0 \leq m, n \leq 1$).

If the sampling of the object scene satisfies the Nyquist criterium, the exact value of P_{int} can be found by using an ideal 2D low pass filter. In the frequency domain this filter represents a block function, and in the spatial domain a sinc function. Unfortunately this function is defined for $-\infty < x, y < +\infty$. Therefore an infinite number of pixels are needed for the calculation of P_{int} . This is of course impossible to realize, so it is necessary to look at other interpolation functions.

It can be shown that interpolation methods such as nearest neighbor and bilinear interpolation have a poor high frequency behaviour [4]. In applications where sampling is done above the Nyquist frequency, these interpolation techniques may be sufficient. However, in most of the applications abrupt changes are present in the scene and the interpolation techniques mentioned will lead to images with aliasing artifacts. While truncating a sinc function even leads to worse images compared to bilinear interpolation [5], the so called high resolution cubic spline interpolation is used.

According to [5] this interpolation function approximates the ideal low pass filter very closely. In the one dimensional case it uses four pixel values to calculate a new intensity. Therefore the function is defined in the interval $[-2,2]$ and equals zero outside, see Eq. (4).

$$\begin{aligned} f(x) &= (a+2)x^3 - (a+3)x^2 + 1 & 0 \leq x \leq 1 \\ f(x) &= ax^3 - 5ax^2 + 8ax - 4a & 1 \leq x \leq 2 \end{aligned} \quad (3)$$

With a the behavior of the function can be controlled. Using $a = -0.5$ provides the best interpolation results for image processing [5].

Figure 4. 1-D interpolation, using the cubic spline function with $a = -0.5$.

A one dimensional interpolation with this function is illustrated in Figure 4. The Figure shows four known pixel values in the x-direction, and the cubic spline function placed symmetrically around the unknown pixel P_{int} . The value of P_{int} will be found by first multiplying each pixel intensity with a cubic spline function value, and then adding all four products. With $f(x)$ is the cubic spline function 4, the previous steps can be described as:

$$P_{int} = f(m-2)P_1 + f(m-1)P_2 + 3f(m)P_3 + f(m+1)P_4 \quad (4)$$

To use this technique for images, a two dimensional description is required. In that case 4×4 pixels are needed. The calculation can be split into five one dimensional interpolations. Four interpolations will be in the x-direction which results in four intermediate new pixel values $X_1 \dots X_4$. With these values the fifth interpolation, in the y-direction, is performed and the final value of P_{int} is found.

3. IMPLEMENTATION

Figure 5. System configuration.

The function blocks that were chosen to perform the image correction are shown in Figure 5. After digitization, the CCD picture (512×512 pixels and 25 images/s) is stored in a double image buffer. One side of the buffer is written by the camera,

while the other part is used for processing. The transformer blocks generate addresses, pointing to a certain position in the input image. The buffer returns a 4x4 pixel-kernel around this address. These pixels are used by the 4 row-interpolators, which generate 4 intensities. These four intensities are interpolated again, resulting in the final intensity for that pixel. These transformer and interpolator blocks will be explained next.

Transformation

A polynomial can be evaluated through forward differencing on condition that the function variable changes with constant steps. An architectural structure that implements the algorithm is presented in Figure 6.

Figure 6. Architectural structure of x-calculation, by forward differencing.

In the right-most column, each difference-block contains a multiplexer (symbolized as a switch), an adder, and three registers. The first register (A) contains the actual value of the difference-block. The second register (B) contains A's value at the start of a frame ($u=0$ and $v=0$). The third register (C) contains A's value at the start of an image-line ($u=0$). After the frame-sync signal, all A registers are loaded with their B registers. At this time (upper-left pixel in output image), the output (X) will point to the corresponding pixel in the input image. To generate the address of the subsequent pixel, all multiplexers are set upwards (u-direction). This way, all registers are increased with their higher-order derivative. When a line-sync is received, u changes from the end of a line (u_{max}), to the beginning of the next line ($u=0$). This is why the C registers are needed: All A registers can be loaded with these registers. Then a step in the v-direction has to be done. This works in the same way as described above, except all multiplexers are turned to the left (v-direction). Again, all C registers are loaded with the current A register values. Generation of following pixel addresses works as explained above.

In order to reduce the amount of registers, a new concept is presented in Figure 7. The design is split into two parts, a u- and v structure. The u-structure calculates the pixel- address, by continuously shifting four registers (containing x , Δx , $\Delta^2 x$ and $\Delta^3 x$) This way, all registers are incremented with their higher order derivative (except $\Delta^3 x$, which is incremented with 0, when the switch is set to '0'). The other part of the design is the left-hand 'v' structure. This structure calculates all difference values that change only once per line. All results that are required to calculate the next line, are stored in the pipeline registers between the v- and u structures. A controller should generate the appropriate commands to the registers, so that they are loaded with the correct values. When a line-sync is received, these values are copied into the u-structure. The external line-FIFO contains difference registers that only have to be changed once per line. The frame-FIFO allows transformation of more than one image, with different sets of parameters.

Figure 7. Simplified architectural structure.

Interpolation

As mentioned in section 2 the cubic spline interpolation can be described as follows:

$$\begin{aligned} P_{int} &= f(n-2)X_1 + f(n-1)X_2 + f(n)X_3 + f(n+1)X_4 \\ X_i &= f(m-2)P_{i,1} + f(m-1)P_{i,2} + f(m)P_{i,3} + f(m+1)P_{i,4} \end{aligned} \quad (5)$$

Where: P_{int} is the interpolated pixel, $f(x)$ is the cubic spline function, P_i is a pixel intensity, m and n are fractions in the x and in the y-direction, respectively.

Since P_{int} is calculated for every pixel in the output image, these equations must be solved in one pixel time interval. In this time interval 20 multiplications and 15 additions must be executed for the interpolation function, and another 40 multiplications and 20 additions for the cubic spline function evaluations. Then the necessary computing power for images at 25 images/s is about 620 MOPS (Mega Operations Per Second). This rather high performance makes it useful to search for some optimizations.

The first optimization is provided by the transformer. The transformer calculates with fixed-point representations, which leads to a certain length of the fraction part of the coordinates. The number of fraction bits determines the number of possible locations between the gridpoints. In our case we assumed that four fraction bits are sufficient, so there are 16 (2^4) possible intermediate locations.

Only for the intermediate points, the function values of the high resolution cubic spline must be known. Therefore it is possible to store these values in a LUT (Look Up Table). So the number of operations is reduced with the cubic spline evaluations. This results in a much smaller computing power of 230 MOPS.

The second optimization can be found from Formula 5. Since the two-dimensional interpolation exists of five one-dimensional interpolations, it is possible to split the design in five similar ASICs. Four ASICs for the interpolation in the x-direction, and one in the y-direction. The required computing power per ASIC is in this way reduced to about 45 MOPS. So now in each ASIC the following function must be implemented:

$$P_{int} = P_1c_1 + P_2c_2 + P_3c_3 + P_4c_4 \quad (6)$$

Where: P_i a known pixel value and c_i a Cubic spline coefficient

A direct implementation in hardware requires 4 multipliers and 3 adders. Using parallel multipliers results in a very large ASIC with the drawback of yield risks. Therefore the possibility for using other types of multipliers is considered. For this design a serial/parallel multiplier with an extension is developed. A serial/parallel multiplier calculates the product by repeated addition. The algorithm for the calculation of the product of P and c is:

```
Prod = 0
for i = 0 to n - 1 do
    Prod = Prod + 2i P ci
end
```

Where: P the pixel intensity, c the cubic spline coefficient, n the number of bits of c , and c_i is bit i of c .

The multiplication is done in n -steps. In every step i , P is added to the intermediate result with a significance determined by 2^i , if c_i equals one. All four product terms in Eq. 6 can be calculated this way, and can be added afterwards. The addition of all four terms after multiplication can be expressed as follows:

$$\begin{aligned} & (P_1c_{1,n-1})2^{n-1} + (P_1c_{1,n-2})2^{n-2} + \dots + (P_1c_{1,0})2^0 + \dots + \\ & (P_4c_{4,n-1})2^{n-1} + (P_4c_{4,n-2})2^{n-2} + \dots + (P_4c_{4,0})2^0 \\ & \frac{(P_4c_{4,n-1} + P_3c_{3,n-1} + P_2c_{2,n-1} + P_1c_{1,n-1})2^{n-1}}{(P_4c_{4,n-2} + P_3c_{3,n-2} + P_2c_{2,n-2} + P_1c_{1,n-2})2^{n-2}} + \dots + \quad (7) \\ & (P_4c_{4,0} + P_3c_{3,0} + P_2c_{2,0} + P_1c_{1,0})2^0 \end{aligned}$$

The answer has still a serial/parallel structure. Only this time the term as expressed in Eq. 8, in stead of a single pixel intensity, is added in each step i to the intermediate result with the significance 2^i .

$$P_1c_{1,i} + P_2c_{2,i} + P_3c_{3,i} + P_4c_{4,i} \quad (8)$$

If Eq. 8 can be calculated in real-time for every step i , equation 6 can be obtained in n -steps, with one extended serial/parallel multiplier. Figure 8 shows the block diagram, in which the interpolator is implemented with the multiplier.

Figure 8. interpolator block diagram

4. Design Methodology and Results

The design of the real-time imaging system is split into a PCB part and an ASIC part. Each part is developed by a design team. The ASIC part is divided into a transformer ASIC and an interpolator ASIC. First of all a literature study is done on the two basic problems: the distortion correction and the cubic spline interpolation. Hence, algorithms were developed to perform the high level simulations, from which design-constraints were obtained. Also, a C- program was written, which implements the transformation and interpolation. Results, obtained with this program are shown in Figure 9.

Figure 9. Distorted and corrected image of a Fujinon lens (F=0.85).

After the specifications were defined several design alternatives were investigated. A MoDL¹ hardware description was made, using the best solution obtained. Based on the MoDL descriptions, which did meet the required specifications, the hardware implementation was completed using COMPASS design software [6]. Using CMOS 1 μm technology and a 55 MHz clock frequency, the resulting areas of the chips is 10 vs. 25 mm^2 for the interpolator and transformer, respectively.

References

- [1] G.S. Monk, *Light, Principles and Experiments*, 2nd, Dover Publications, New York, USA, 1963.
- [2] P.W. Hawkes, E. Kasper, *Principles of Electron Optics*, 2nd, Academic Press, London, Great Britain, 1989, pp. 378-382.
- [3] M. Born, E. Wolf, *Principles of Optics: Electromagnetic Theory of Propagation, Interference, and Diffraction of Light*, Pergamon Press, New York, USA, 1975.
- [4] M.J. Bentum, R.G.J. Arendsen et al., Design and Realization of High Speed Single Exposure Dual Energy Image Processing, *The proceedings of the Fifth Annual IEEE Symposium Computer-based Medical Systems*, Durham, USA, 1992, pp. 25-34.
- [5] J.A.Parker, R.V.Kenyon et al., Comparison of Interpolating Methods for Image Resampling, *IEEE Transactions on medical imaging*, vol. MI-2, No. 1, 1983, pp. 31-39.
- [6] COMPASS Design automation, Les Taissounières, Route des Dolines, 06560 Sophia Antipolis, France.

Acknowledgements

The authors thank the EUROCHIP VLSI Design Action and the Dutch PICO project for supplying CAD software and processing facilities.

A.G.J. Nijmeijer, M.A.Boer et al.
University of Twente
Department of Electrical Engineering
Laboratory for Network Theory
P.O. Box 217
NL-7500 AE Enschede
e-mail: sammie@nt.el.utwente.nl

¹MoDL (Modeling and Design Language) is a high level description and simulation language developed at the University of Twente