J Real-Time Image Proc (2014) 9:463–478
DOI 10.1007/s11554-013-0330-9

SPECIAL ISSUE PAPER

Low-power DSP system for real-time correction of fish-eye cameras in automotive driver assistance applications

Mauro Turturici · Sergio Saponara · Luca Fanucci · Emilio Franchi

Received: 29 June 2012 / Accepted: 8 February 2013 / Published online: 3 March 2013

Abstract The development of an embedded system for real-time correction of fish-eye effect is presented. The fish-eye lens is applied to driver assistance video systems because of its wide-angled view. A large field of view can reduce the number of cameras needed for video system and their cost, installation, maintenance and wiring issues. On the other hand, this lens causes inherent radial distortion to image that has to be corrected in real-time with a low-cost and low-power processing platform. This paper proposes a solution that can be easily adapted to different types of lens and camera, and meets real-time constraints with a power budget within 100 mW and a board size of few cm^2. Starting from mathematical equations, given by the geometrical optics, a state-of-the-art correction method is presented, then optimizations are introduced at different levels: algorithmic level, where a real-time correction parameter calculation avoids extra non-volatile off-chip memory cards; data transfer level, where a new pixel pair management reduces memory access and storage burden; HW-SW implementation level, where a low-power board has been developed and tested in real automotive scenarios. Other applications of the developed system, such as multi-camera and multi-dimensional video systems, are finally presented.

Keywords Fish-eye · Video correction · Automotive · Driver assistance · Real-time video processing

M. Turturici (✉) · S. Saponara · L. Fanucci
Department of Information Engineering, University of Pisa,
Via G. Caruso 16, 56122 Pisa, PI, Italy
e-mail: mauro.turturici@for.unipi.it

E. Franchi
R.I.Co. srl, Via Adriatica 17, 60022 Castelfidardo, AN, Italy

1 Introduction

In last years, the use of cameras for automotive driver assistance has increased a lot [1–22]. Nowadays most of the car manufacturers offer video systems on their vehicles to give to the driver a better view of the so-called "blind spots", areas that are normally impossible to look easily in the driving position. Therefore, it is important to use an array of cameras mounted towards every directions or, as alternative, cameras equipped with wide-angled lens (referred in this paper as "fish-eye cameras").

Fish-eye lenses have been commonly used for surveillance applications [15, 17] due to their large field of view (FoV), but nowadays they find new useful applications in automotive driver assistance video systems [7–17]. In fact a single camera with a fish-eye lens can substitute, in some cases, up to four cameras with traditional lenses.

On the other hand, the fish-eye lens suffers of distortion problems, mainly radial [11, 15]. Since it is very important to give a correct view to the driver, with exact proportions and without any distortions, the correction of images captured by a fish-eye camera is required. Correction of the fish-eye effects has been treated in literature mainly for photography applications. For example, there are several software tools that led the photographer correct this distortion for a picture (e.g., PanoTools [23], Fisheye-Hemi Plug-In [24] for Photoshop and others). However, these solutions, like some other works for surveillance applications [16], refer to a software running on a personal computer. On the contrary, automotive applications call for the real-time correction of fish-eye cameras with hardware platforms compliant with use on-board a car. To this aim limited cost, size and power consumption are required; therefore, the platforms are constrained in terms of computational and memory capabilities.

This paper presents a low-cost, flexible and real-time solution for correcting video captured by fish-eye cameras. With respect to state-of-the-art fish-eye correction systems, the proposed platform provides a solution that can be easily adapted to different types of lens or camera and meets real-time constraints with a power budget within 100 mW and a board size of few cm^2. To reach the above targets, this work introduces optimizations at algorithmic level, where a real-time look-up table (LUT) correction technique is used, avoiding extra non-volatile (NV) off-chip memory cards, at data transfer level, where a new pixel pair management method reduces memory accesses and storage burden, and at HW-SW implementation level where a low-power board has been developed and tested in real automotive scenario.

The paper is organized as follows. Section 2 briefly reviews state-of-the-art video correction systems for fish-eye distortion effects. In Sect. 3, after discussing the basic principles of a fish-eye lens, the main equations to be used to perform the correction of the radial distortion are described. In Sects. 4 and 5 the calculation method for LUT-based correction and the new pixel management technique is presented. The design flow is shown in Sect. 6, while the porting on the low-power platform is discussed in Sect. 7. The main achieved performances plus tests from real automotive scenarios are discussed in Sect. 8. Furthermore, in Sects. 7 and 8 it is shown that the current implementation of the system can do a multi-dimensional correction (two space dimensions and time) and can also be used for substituting an array of standard cameras, since that a single fish-eye camera can reach the same FoV of few cameras with standard lenses. Conclusions are drawn in Sect. 9.

2 Review of state-of-the-art fish-eye correction systems

Fish-eye correction is a common feature of several professional photo retouching software. These programs, such as Fisheye-Hemi Plug-In for Photoshop, PanoTools or DxO Optics Pro, running on a personal computer, are mainly used for off-line correction of still pictures. Due to their heavy-duty tasks they are not suitable for the implementation of a low-power real-time fish-eye correction system. Another recent work [20] presented a fish-eye correction system for video stream at VGA resolution, still based on a personal computer (Intel Core 2 with 1 GB RAM). This system allows a correction of fish-eye video stream with a 15–17 fps rate and a rough power consumption of several tens of watts (e.g., the processor has power consumption up to 65 W [25]).

At the state-of-the-art few solutions have been already proposed for power-effective real-time correction of fish-eye images. Altera and Manipal Dot Net developed a system based on the Cyclone III FPGA device and Nios II processor [10]. This solution uses a pre-computed LUT and performs a remap of every pixel using the information stored in this table. Being pre-computed it needs an extra off-chip NV memory card which is specific for a defined camera and lens, so changing the camera resolution or the color space the solution cannot be reused but new and different external memories must be adopted.

A solution called logiVIEW is proposed by Xylon and targets Xilinx FPGA devices (Spartan6 and Virtex6 families) using Microblaze as soft processor [26]. Also in this case a first step for customization of the solution is required on the specific camera or lens type by the original equipment manufacturer (OEM). Therefore, SRAM-based FPGAs (like Altera and Xilinx) need an external NV memory device, which contains information about how to do the correction. These solutions implement just a fixed correction algorithm while, in order to adapt the solution to different types of lens, camera and display, a higher level of flexibility is required.

A similar solution [18] was presented in 2009 at the 17th IEEE Symposium on Field Programmable Custom Computing. Also this solution, based on high performance Virtex-6 has low flexibility towards different types of input–output video devices. Furthermore, the last two solutions use a logic unit whose cost is very high for automotive applications.

In 2011, a solution has been announced by Techwell (a part of Intersil) [27]. More information about the announced Techwell solution is not available, but notoriously the system is based on proprietary and custom Intersil Image Signal Processor, specifically designed for Techwell surveillance devices, and this reduces its flexibility and reusability to other video devices with different features and correction requirements.

The only cost-effective state-of-the-art solution has been proposed in 2011 [17] by NXP and it is based on proprietary PNX9530 media processor (basically PNX1005 for automotive market). Although this solution is proposed for bird's eye video system it can be used also for fish-eye correction. Also this system cannot be adapted to different input cameras without changing the software itself or use an external memory. The cost of this platform is not exactly known, but considering just some other similar products by NXP itself, we can suppose that our platform, being targeted only on fish-eye correction, can reach the same results with 30 to 40 % less costs.

3 Fish-eye distortion correction

3.1 Camera and lens: basic principles

The basic principle of a camera is well explained by the "pin-hole" camera model [28]. A scene, see Fig. 1, can be

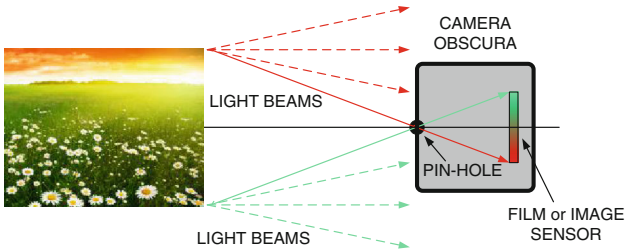


Fig. 1 Pin-hole camera model

reproduced on a film by means of a camera obscura with a tiny hole on one side. When a point of the scene is enlightened, rectilinear light beams are projected in many directions. Only a part of these light rays can enter through the pin-hole and so only a particular region of the film will be impressed. Most of the beams coming from the same point cannot impress the film because they are stopped by the camera obscura. So the result is a bi-dimensional picture, vertically and horizontally reversed towards the original tri-dimensional scene.

Referring to Fig. 2, let $\boldsymbol{h}$ be the optical axis (i.e., the horizontal straight line that ideally goes perpendicular to the film, across the pin-hole) and consider a generic point P of the scene with a given distance from $\boldsymbol{h}$. We call θ the angle between the incoming light beam from P and $\boldsymbol{h}$. Other important variables are:

- R_p the distance between the projection of P and $\boldsymbol{h}$
- f the distance between the film and the pin-hole

The distance R_p on the film of the projected point from the optical axis $\boldsymbol{h}$, referred to the angle θ, can be computed with Eq. (1) for $K = 1$.

Cameras are usually used in conjunction with optical lenses that divert light rays and project on the image sensor (or on the film), an image that is different from the original. Mostly used lenses only change the FoV of the camera (basically, the area of the scene that is reported on the picture), but they do not change proportions between points of the picture. For those lenses, the formula that links the scene to the image sensor, also known as "mapping function", is still Eq. (1), but with a generic $K > 0$.

- R_p : the distance between the projection of P and **h**
- f : the distance between the film and the pin-hole

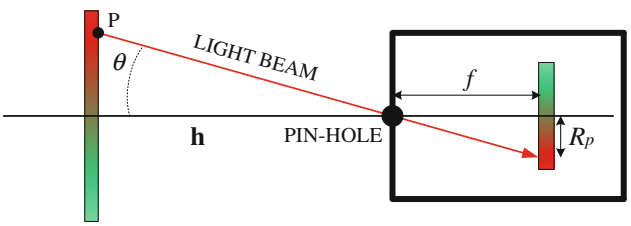


Fig. 2 Pin-hole camera basic principles

$$R_p = K \times f \tan(\theta) \tag{1}$$

3.2 Fish-eye optics introduction

A lens or an optics (i.e., a combination of more lenses) may also have a non-linear mapping function, so the representation of the scene on the image sensor can be very different from the normal view.

One of the most used types of non-linear optics is the so-called "fish-eye". A camera equipped with a fish-eye lens can easily capture points on the same plane of the lens and even behind, so it has an angular FoV wider than 180°. Due to its extremely wide angular FoV these optics find useful usage in photography (see Fig. 3a), surveillance (see Fig. 3b) and automotive applications. In photography fish-eye lenses are mainly used to give a particular effect to the picture, but currently their most emerging applications are in surveillance systems and automotive rear-view cameras. Indeed, thanks to their wide angular FoV, it is possible to look a large part of the scene with a single camera instead of using several cameras. This helps to reduce costs and also installation, maintenance and wiring issues which are key elements for the success of many electronic systems, in particular for the automotive market.

Moreover, using two or more cameras it can be necessary to combine images with mosaic techniques [11, 17] in order to make a "fit to be seen" image. Instead using a fish-eye lens, it is possible to develop a large FoV 3D image system (two space directions and time) with a single camera, replacing up to three cameras without the need of any combination of images.

On the other hand, the quality of images captured with a fish-eye lens cannot be as good as the one obtained with a standard lens; in fact a fish-eye lens creates distortion effects for the image, especially at the border. Images appear good in the middle part of the picture, but "compressed" at the borders (see Fig. 3b and c).

A very first explanation of the most evident effect given by the fish-eye lens can be the following: light beams arriving at the border of the lens are deflected more than those arriving at the center; therefore, the scene is much wider compared to a normal lens while objects in the center of the image look better than those at the borders (see Fig. 3b). This effect is known as radial distortion (see Fig. 3c). A fish-eye lens may also suffer from tangential distortion, due to fabrication defects, but this is usually negligible with respect to the radial one.

The mathematical formalization of this effect is given by the mapping function of the lens. The relation between the incoming angle θ of a light beam and its projection R_p (named R_{fish} for fish-eye cameras) on the image sensor can be quite different from Eq. (1). Some of the most common mapping functions, used to represent mathematical

Fig. 3 **a** Example of an extreme fish-eye optics for photography application; **b** a picture from a surveillance system equipped with fish-eye lens; **c** representation of the typical radial distortion introduced by a fish-eye lens

behavior of the fish-eye lenses, are proposed below in Eqs. (2), (3), (4) and (5) [10], in which we leave out the multiplier K.

$$\text{Linear scaled: } R_{\text{fish}} = f\theta \tag{2}$$

$$\text{Orthographic: } R_{\text{fish}} = f\ \sin(\theta) \tag{3}$$

$$\text{Equisolid angle: } R_{\text{fish}} = 2f \sin\left(\frac{\theta}{2}\right) \tag{4}$$

$$\text{Stereographic: } R_{\text{fish}} = 2f \tan\left(\frac{\theta}{2}\right) \tag{5}$$

Obviously the representation obtained with the above lenses will not have the same proportions as the one taken with a standard lens. Particularly, as already shown in Fig. 3b, fish-eye lenses introduce radial distortion, which occurs when proportions between points lying on the same radius are not respected before and after the lens.

3.3 Back-mapping method for fish-eye effect correction

The exact method to correct an image affected by a fish-eye distortion would be to use an optical lens opposite to the fish-eye one (i.e., that has a reversed mapping function) in front of the output screen. This is practically impossible for many reasons, so digital processing is the only way to achieve this goal. The simplest method to do a digital correction is the direct rearrangement of pixels, i.e., copy every $p_{j,k}$ pixel of the source image in its correct position to obtain a new corrected image by filling an empty output buffer. Referring to a stream of pixels given by a camera this can be done with the algorithm described below:

1. Initially allocate an empty buffer for the output images, with size of a full frame.
2. Receive a pixel $p_{j,\ k}$ from camera.
3. Use the coordinates $(j,\ k)$ of the current pixel $p_{j,\ k}$ to compute its correct position (n,m), rounded to the closest integer.
4. Copy the pixel $p_{j,\ k}$ at the correct position $(n,\ m)$ in the output buffer.
5. When a full frame has arrived, send the output buffer to the video output port.

The correct rearrangement of pixels is done by means of mapping function: for example, if we had a fish-eye lens represented by the equisolid mapping function (4), we know that a light beam that came with an angle θ with $\boldsymbol{h}$ (see Fig. 2) will be projected at a distance R_{fish} from $\boldsymbol{h}$ given by Eq. (4), thus we can compute θ by reversing the equation, as shown in (6). Then we have to put the pixel in the correct radial position, see Eq. (7).

$$\theta = 2 \sin^{-1}\left(\frac{R_{\text{fish}}}{2f}\right) \tag{6}$$

$$R_{\text{corr}} = f \tan\left[2 \sin^{-1}\left(\frac{R_{\text{fish}}}{2f}\right)\right] \tag{7}$$

So it is sufficient to copy every pixel with a distance of R_{fish} from the center to a distance R_{corr} along the same radius on the output buffer. For a digital image radius and distances are computed as number of pixels from the center of the film. Unfortunately, this method causes many blank pixels, because some pixels of the output buffer may not be filled as a result of the Eq. (7); furthermore, some pixels of the input stream can be put in the same position, being overwritten.

As an alternative, the same quality of the image, but no blank pixels, can be achieved with the "back-mapping" algorithm in Fig. 4 [10] acting as follows:

1. Initially allocate two empty buffers, with dimension of a full frame, one for the input stream and one for the output one.
2. Receive an entire frame from camera and store it in the input buffer.
3. Scan every $p_{j,k}$ location of the (empty) output buffer and compute for each one the position $(n,\ m)$ of the correct pixel of the input buffer to be copied there, or the nearest integer to that one.

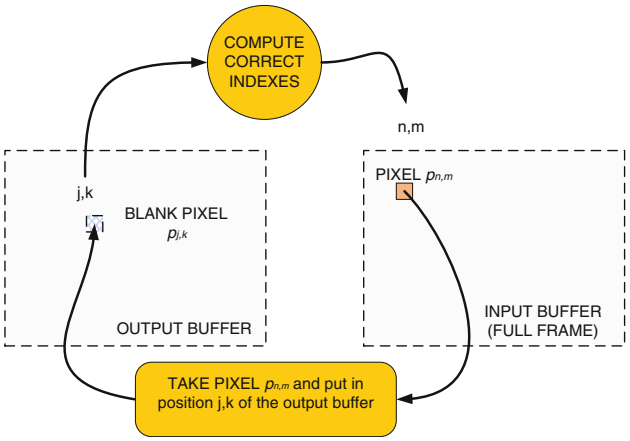


Fig. 4 Representation of the back-mapping method

4. Load the pixel *(n, m)* from the input buffer, and store it in the output buffer, in the current position *(j, k)*.
5. When the output frame is full, send it out to the video output port.

Instead of taking pixels from the source and put them in the correct position in the output buffer, this method computes for every blank pixel in the output buffer the "most suitable" pixel of the input buffer to put there; basically, the nearest to the exact one. This causes some little shape distortion, and that more than one empty location of the output buffer will be filled with the same pixel from the input buffer. No blank pixels will result in the output buffer. This method provides a good resulting image as it will be shown in next sections.

For example, refer to the equisolid mapping function in Eq. (4). In a picture taken with a standard lens a generic pixel far R_p from $\boldsymbol{h}$ represents the light beam coming with θ_i from $\boldsymbol{h}$, according to Eq. (1). Because of the fish-eye lens distortion *that* beam has been projected at a different distance, given by Eq. (4). The incoming angle θ_i of the correct light beam can be determined using Eq. (8), while the correct distance R_{source} can be determined using Eq. (9). For digital images, the parameter f can be computed with the Eq. (10) [10] where W is the image width in pixel and FOV_{horz} is the angular horizontal FoV in radians of the lens.

$$\theta_i = \tan^{-1}\left(\frac{R_p}{f}\right) \tag{8}$$

$$R_{\text{source}} = 2f \sin\left[\frac{1}{2}\tan^{-1}\left(\frac{R_p}{f}\right)\right] \tag{9}$$

$$f = \frac{W}{4\sin\left(\frac{\text{FOV}_{\text{horz}}}{2}\right)} \tag{10}$$

Therefore, if the lens mapping function is known, it is possible to perform the back-mapping correction of the radial distortion.

4 Fish-eye algorithm with new LUT process method

As seen in Sect. 3 the remapping of a single pixel involves the calculation of several equations with trigonometric functions. The real-time computing of every pixel coordinates is a heavy-duty task, not feasible under the power and cost budget constraints of automotive applications. On the other hand, real-time computing of correction parameters ensures high flexibility to every type of fish-eye lens and the possibility to apply other types of modification to the image such as digital zoom; these are eligible features for such a video system. Real-time state-of-the-art solutions use a fixed correction so, basically, a memory in which the results of the equations in Sect. 3 are stored, previously computed for every pixel. This solution is suitable just for a single type of lens at a time and changing the lens means changing the memory. For example, proprietary surveillance systems uses external NV memory card with a precomputed LUT that is right only for their proprietary lenses.

Instead the solution proposed in this paper has the flexibility to adjust to almost every type of available lens and does not need any dedicated external NV memory to contain the LUT. We use the same approach to do the remap, so we obtain the same quality as state-of-the-art methods. The remap indexes are computed once at start-up according to a parameter identifying the lens type. They are then stored in a small random access memory (RAM), the same for program and data, so we read correction indexes for the remapping process directly from RAM. This technique combines the good results of the back-mapping method to correct the images with a high flexibility.

4.1 Look-up table computation

The LUT is computed by means of three steps: in the first step, we compute the correction of radial distortion for every pixel; hence, we get both horizontal and vertical indexes for every pixel referring to the center of the image. In the second step, we compute the correct location of pixels to be taken for every empty location of the output buffer. Finally we compute memory addresses of pixels in the input buffer for direct load and store operations. The mathematical equations used to perform these actions are those described in Sect. 3.

For instance, consider the equisolid mapping function (4). In digital world distances are reported as pixel unity, so we must transform distances into pixel coordinates. Consider a generic point P and let (j, k) and (n, m), respectively, be the coordinates of pixels of the source (distorted) and target (undistorted) image. Since tangential distortion

is negligible (points of the source buffer have to be rearranged theoretically on the same radius) it results:

$$\frac{j}{k} = \frac{n}{m} \tag{11}$$

This implies that the corrected point coordinates can be computed by means of Eqs. (12) and (13).

$$n = \frac{2f\,j}{R_p} \sin\left[\frac{1}{2}\tan^{-1}\left(\frac{R_p}{f}\right)\right] \tag{12}$$

$$m = \frac{2f\,k}{R_p} \sin\left[\frac{1}{2}\tan^{-1}\left(\frac{R_p}{f}\right)\right] \tag{13}$$

where:

$$R_p = \sqrt{j^2 + k^2} \tag{14}$$

After the first step, we have, for every (j, k) blank pixel of the output buffer two floating-point numbers (n, m) that represent the indexes of the right pixel to be taken from the input buffer, referred to the center of the image. In the next step, we associate to these indexes their rounded integer location in the input buffer and then its memory address so we have exactly the RAM address of that pixel in the source buffer, not only its indexes. The LUT computation is briefly represented in Fig. 5.

As said before, a LUT that stores pointers to memory locations and not indexes can be used for load and store operations. Only simple instructions (memory access and sum) are used during real-time execution, while heavy-duty operations are concentrated at the start-up. Hence, after the start-up phase when the LUT is calculated, this algorithm mainly requires the use of RAM memory.

4.2 Real-time processing

After the start-up phase and after the calculation of the LUT, real-time operations can be easily explained with the pseudo-code below:

- Scan position *(j, k)* of the output buffer
- Access LUT at *(j, k)* position
- Read a data which represents the memory address of the source buffer to be taken (a pointer to that location)
- Copy that pixel to the current blank position of the output buffer
- Go to the next blank position $(j + 1, k)$ and repeat

Incoming frames are stored and remapped one by one; when a frame is ready this is sent to the video output port and a new frame is captured, see Fig. 6.

To further reduce the memory access a new pixel pair management strategy, exploiting chrominance sub-sampling, is introduced in Sect. 5, allowing the implementation of a low-cost and low-power embedded system.

5 Pixel pair management for reduced data transfer and storage complexity

5.1 Principles of chrominance sub-sampling

Data for image and video applications can be coded in many different ways. Color and position information about pixels can be also mixed to obtain a compression format suitable for the application. For example, an RGB signal carries information about brightness of the three fundamental colors that our eyes can perceive, i.e., red, green and blue. Other encoding methods can carry, for example, only brightness information (e.g., black/white images), or full color information plus transparency (images with alpha channel). Encoding method can be lossy or not vs. the original image captured by the sensor. One of the simplest color spaces is YCbCr 4:4:4; every pixel has the same information as RGB, but signals are carried as luminance (three colors mixed together) and chrominance (color components mixed together). Since human eye is more sensitive to the brightness intensity than colors, several ways to reduce information carried by video signals without noticeable loss of quality have been proposed in literature.

One of the most common video standards is ITU-R BT601 [29]. It specifies frame size, color space (YCbCr) and communication protocols; but also specifies a method to reduce the bandwidth of the carried signal, the so-called "4:2:2 horizontal chrominance sub-sampling". It basically consists of a reduction of chrominance information for horizontally paired pixels: luminance information (Y) is sampled at full rate, instead chrominance information (Cb and Cr) is sampled every two pixels. Thus, pixels are coupled two by two, every pixel has its own luminance but chrominance information is the same for each pair.

In Fig. 7a we see some pixels coded as 4:4:4. In Fig 7b,c and d these pixels are decomposed as their Y (luminance), Cb and Cr chrominance components. In Fig. 7f, g and h, we see instead the process of chrominance sub-sampling. As you can see luminance data of both images (Fig. 7b and f) are the same, but chrominance signals are different. The reconstruction of a 4:2:2 representation of the original image is shown in Fig. 7e. It is easy to see that, at first sight, it is similar to the original one (Fig. 7a) but, looking deeply, some pixels are significantly different. Please note that the image in Fig. 7a was expressly made for this example and due to this fact the 4:2:2 sub-sampling technique appears very lossy compared to the 4:4:4 image. For images taken from the real world, chrominance information has a slow variation, so the quality of images coded with this lossy compression results very high, as shown in Sect. 5B. ITU-R BT601 standard encodes also the data depth, 8 or 10 bits, for digital transmission. The use of 8-bit data

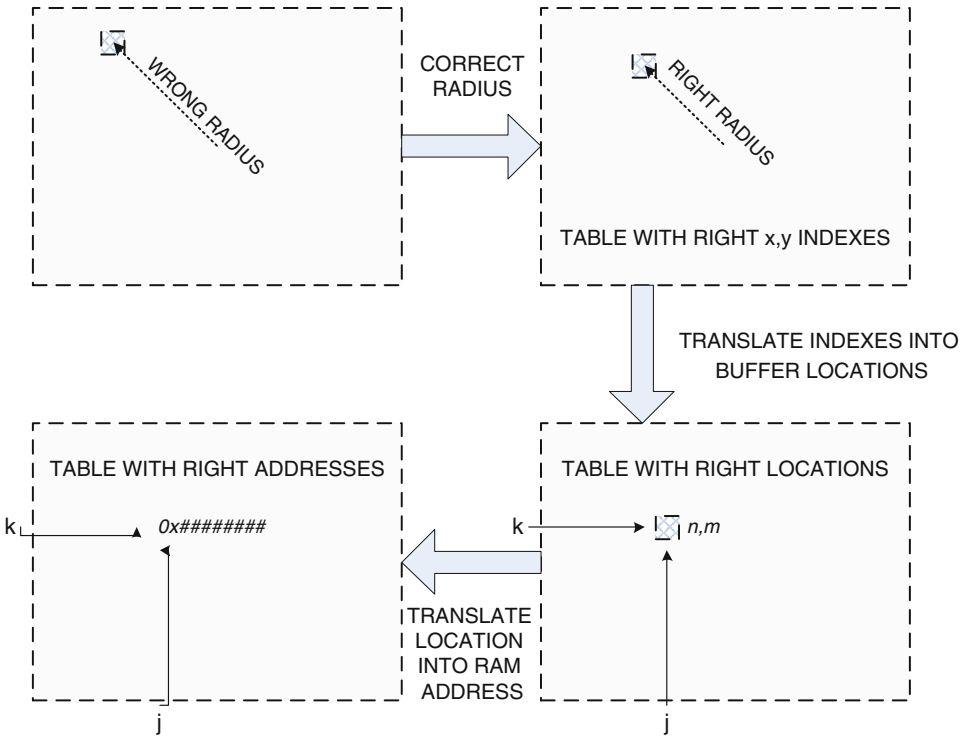


Fig. 5 Proposed three-step LUT creation

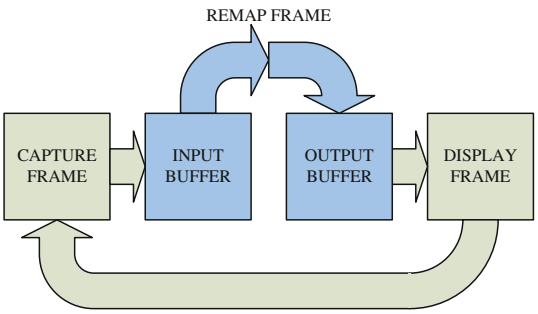


Fig. 6 Real-time data flow and processing

depth is the most common for standard video applications, including automotive, instead 10-bit depth is mainly used for high quality video such as movies. In many applications data are streamed with a single 8-bit parallel interface and typically arrive with the order $| Y_i | \mathrm{Cb}_{i,i+1} | Y_{i+1} | \mathrm{Cr}_{i,i+1} |$, this means that, starting from the first pixel, we get complete data of two pixels every 4 byte transfers; these two pixels have their own luminance information, but share the same chrominance information.

5.2 Pair by pair pixel remapping

While cameras used for surveillance applications are mostly mounted as overhead or above position (see Fig. 8a), for automotive application cameras are mounted to have the street as horizontal reference (Fig. 8b). This fact implies that optical axis is approximately parallel to the street, then horizontal distortion is less noticeable.

Using 4:2:2 encoding and 8-bit data depth, it is possible to store two paired pixels in a 32-bit word. Thus, an entire frame with N × M pixels is stored as ½(N × M) 32-bit words. Every word contains a pair of pixels, i.e., two 8-bit luminance information, one chrominance Cb (the same for both pixels) and one chrominance Cr (the same for both pixels again). Since our target application is a video system for automotive applications, we propose to do the correction for paired pixels and not for every single pixel. This can be made with the same load and store operations, but implies half operations than a complete rearrangement; furthermore, it will be shown that the achieved image quality is still good.

In Fig. 9 we see a real-scenario image captured with a fish-eye lens. Figure 10 shows the same image corrected with a complete rearrangement (pixel by pixel), instead Fig. 11 shows the image corrected with pair by pair remapping. It is easy to see that there is no noticeable difference between the two images. Only if we zoom on a particular region of the Figs. 10 and 11 (see Figs. 12 and 13 representing the particular of the solar reflection on the rear part of the parked black car, the second from left), we can see some differences due to the 4:2:2 chrominance sub-sampling. In fact, if we look at the first column of Figs. 12 and 13, we see that they are equal, but pixels in the second column are not the same, and this is true for all even column.

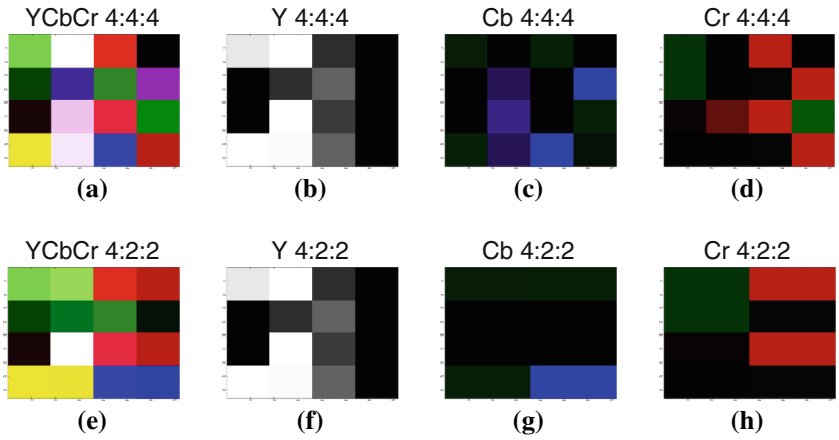


Fig. 7 Representation of 4:2:2 chrominance sub-sampling

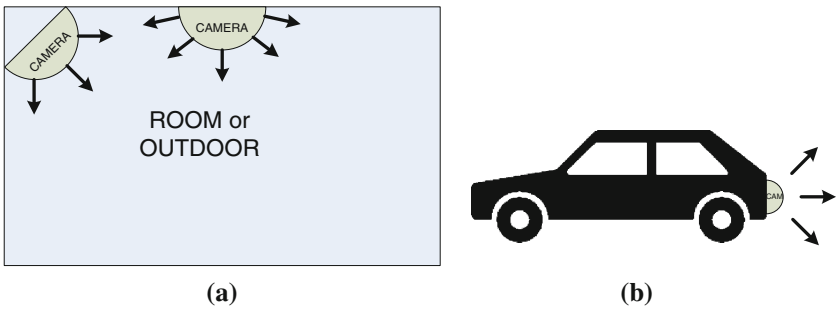


Fig. 8 Representation of the typical mounting of fish-eye cameras for **a** surveillance and **b** automotive applications

In our work, we made several examples of images taken from urban scenario. In each test we see that, for horizontally mounted cameras, the fish-eye effect correction of paired pixels gives results that are practically the same as those obtained with a complete rearrangement. Thus, the proposed method meets standard quality for rear-view automotive cameras with half load and store operations compared to a full rearrangement and this implies a less-costly and low-power hardware to be used.

Further chrominance compression, for example 4:1:1 and 4:2:0, would not provide significant enhancements compared to the 4:2:2 one.

Fig. 9 Original distorted example image

Fig. 10 Example image corrected pixel by pixel

According to the 4:1:1 compression (see Fig. 14c), chrominance information (both Cb and Cr) is sampled every four pixels; so if data are transmitted 8 bits a time, we get a group of pixels every 48 bits (6 byte transfers); every pixel has its own luminance (*Y*), but Cb and Cr are shared for four pixels. If we apply our method for every four pixels instead of two, we have to remap groups of 48 bits (one 32-bit word and a half) instead of one word a time. But it is common knowledge that, for a memory with 32-bit interface it would require two transfers, and there are 16 useless bits every two remap operations because pixel groups are not remapped next to each other, but they can lie on a different row and column. So applying our method to

Fig. 11 Example image corrected with paired pixels method

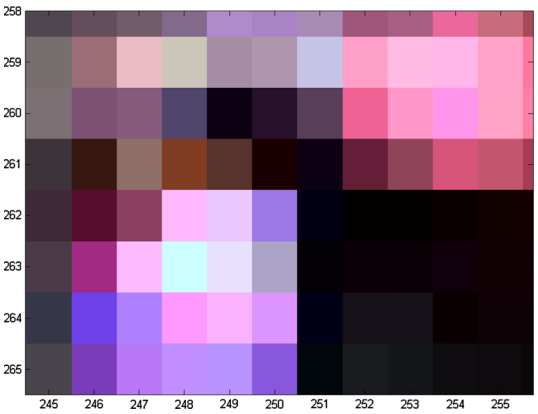


Fig. 12 Detail of Fig. 9 corrected with pixel by pixel rearrangement

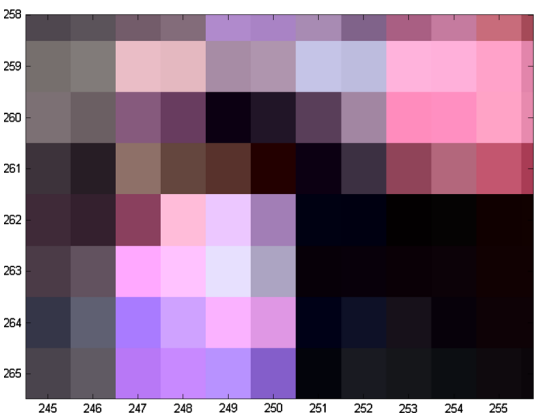


Fig. 13 Detail of Fig. 9 corrected with paired pixels method

4:1:1 compression will result in a loss of quality without allowing a better exploitation of resources vs. a 4:2:2 coding. This fact happens also for 4:2:0 compression: again a group of four pixels shares chrominance information, but this time the group is composed of two pixels that are close to one line and the two lying on the next line in the same position (see Fig. 14d). Also in this case we obtain just a loss of quality without any further enhancement about data transfer.

6 Design flow

This section discusses the design flow for the development of a real-time embedded system (first prototyping generation) for fish-eye correction, starting from the method proposed in Sects. 4 and 5. Its optimization in the final low-power platform is discussed in Sect. 7.

6.1 Algorithm development in MATLAB

The first task of our design flow was to test the quality of the proposed method (with the pixel pairing technique) applied to the rear-view parking video system. An image captured by a fish-eye automotive camera has been coded as 4:2:2 and then remapped as described in Sects. 4 and 5. Referring to the previous figures, we took an RGB picture with a standard rear-view camera equipped with fish-eye lens. Then it has been coded as 4:4:4 YCbCr (Fig. 9) without any loss of quality with respect to the original. As introduced before we performed a fish-eye correction pixel by pixel (Fig. 10). Then we coded the picture as 4:2:2 with a loss of quality, but without any noticeable defect. Finally, we made the correction pair by pair still obtaining a good quality image (Fig. 11).

The equations to be used and the quality of result images have been tested in MATLAB with still pictures and videos taken from urban scenarios; with this information we got parameters to perform some good LUTs, suitable for several types of lens. Many experiments in different urban scenarios confirmed the suitability of this approach for real-time correction of fish-eye camera in automotive assistance and, hence, we decided to start writing code for the development of a HW-SW low-power embedded system.

6.2 Code development in Linux and Qt IDE

After the validation performed in MATLAB we developed a C-code by means of Qt IDE and OpenCV libraries on a Linux platform. The used frame size was the most common for automotive rear-view camera, i.e., 720×480, but could have been selected any of the most common frame size for standard quality video, e.g., 720×486, 720×576, 640×480 and others. At this time we also performed a code analysis and optimization to get information about computational and memory resources needed for the application.

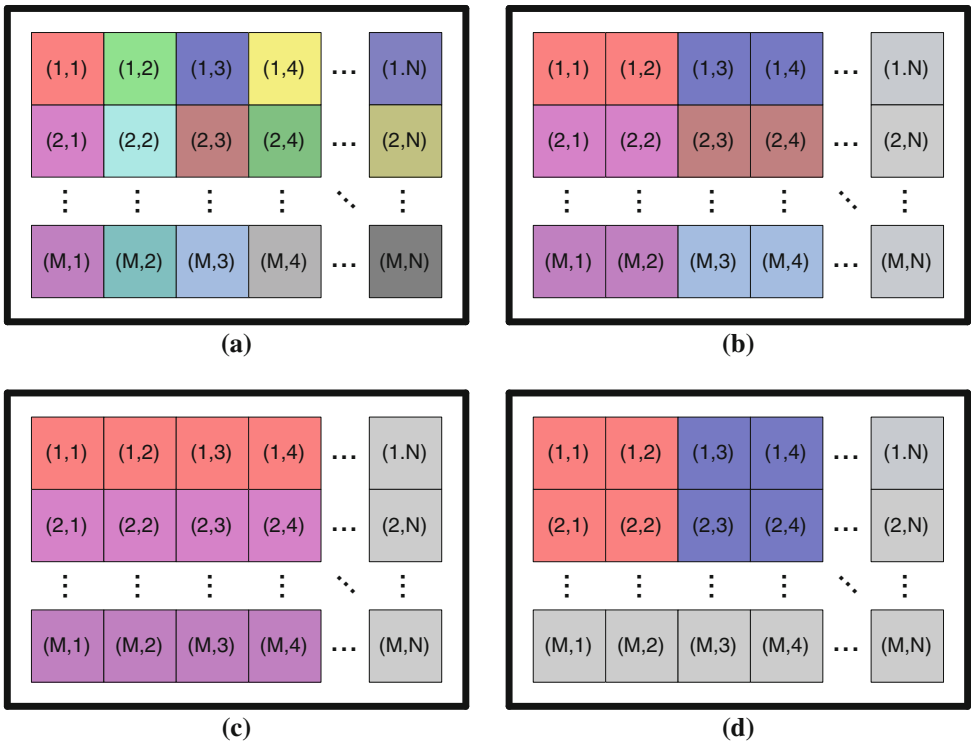


Fig. 14 Representation of some chrominance compressions referring to a screen with M × N resolution, **a** 4:4:4, **b** 4:2:2, **c** 4:1:1, **d** 4:2:0. Pixels in *white–gray* are not known referring to Fig. 14a

6.3 Porting on first generation prototyping high-end platform

The performed analysis allowed a first porting of the developed algorithm for real-time video correction on a high-end video processor. We used a Spectrum Digital EvmDM642 board equipped with high performance TMS320DM642 processor by Texas Instruments. It includes a VLIW (very long instruction word) computing core capable of up to eight 32-bit operations concurrently [30]. It features also high-speed interfaces for external memory connection and video peripherals. The EvmDM642 evaluation board is also equipped with 64 Mb of external DDR RAM and both with video decoder and encoder and other peripherals that are not used in this work. Realized in 130 nm CMOS technology the 720 MHz core requires a 1.4 V voltage supply and consumes roughly 1.5 W.

For the development of the embedded software we used the code written previously for Qt IDE and a TVP5146 video decoder (A/D converter) with 4:2:2 8-bit output. The first design step on this platform, although not meeting low-cost and low-power requirements, ensured a rapid development and prototyping of the application with the certainty to reach a result. The analysis carried out by means of TMS320DM642 processor allowed to easily port our code on a lower cost platform as described in Sect. 7.

7 Low-power embedded platform for fish-eye correction

Since this work aims at the implementation of a cost-effective fish-eye correction system, an important aspect of the research is the hardware targeting. The TMS320 DM642 processor (technically speaking a System-on-Chip) is not as low cost and low power as desired so it was necessary to perform a new porting of the software on a different platform, with lower cost and less power consumption.

Fish-eye effect correction is an emerging feature whose foreseeable market volume does not justify the design of an ASIC, hence a commercially available computing core has to be preferred. An affordable and cost-effective video correction system for automotive applications must have:

- flexibility to be adapted to several resolutions (input and output) and to an arbitrary lens, i.e., any mapping function,
- low cost,
- low-power consumption,
- enough performance for video processing up to 30 fps,
- high reliability, and
- good image quality,

Since a solution with a microcontroller does not provide enough computing power to face real-time video

processing, only FPGA and GPP solutions can be selected. Both GPP and FPGA technologies could ensure enough performance with power consumption suitable for the application. A GPP platform was finally selected for two main reasons: the first is that, we found several systems-on-chip (SoC) that integrate both input and output video port and other useful peripherals for image management (size formatter, D/A converter, blank pixel correction and other); secondly because porting C-code on GPP, such as ARM-based one, is simpler that implementing the algorithm on FPGA. These facts allow for lower hardware costs and reduced time for the software development, with negligible performance degradation.

Our analysis demonstrated that the application is particularly memory-dominated, indeed the computing power is important only at the start-up (the LUT calculation involves many trigonometric functions to be computed), but in real time we do not need high computing capabilities, since the method proposed mainly exploits load and store instructions. Anyway, we cannot neglect the computing power at all, since it is important to keep low start-up time, about few seconds.

The complexity profiling analysis that has been carried out, allowed the selection of the TMS320DM36x processor family by Texas Instruments [31, 32]. This is realized in 65 nm CMOS technology, and is based on a 32-bit ARM926EJ-F core with 16 Kbytes instruction cache memory, 8 Kbytes data cache and further 32 Kbytes of on-chip RAM. The clock frequency of DM365 is 2.4 times lower than the DM642, and the one of DM368 (432 MHz) is roughly 1.6 times lower. These processors share the same architecture and on-board peripherals. The DM368 ensures more performance than the DM365, due to its higher core clock. On the other hand, the DM365 core can be powered at 1.2 V, thus allowing for lower power consumption, while the DM368 core requires a 1.35 V supply voltage. However, the DM365 and DM368 cores are pin-to-pin and software compatible, so both can be used for the final implementation.

Due to low-power considerations the DM365 has been selected to build the real-time fish-eye correction platform whose scheme is presented in Fig. 15. The input section of the prototyping platform is composed by a TI TVP5146 A/D converter, also referenced as "video decoder", that accepts signals in the most common analog formats: composite, Y/C (S-Video), RGB and YPbPr (Component). This way the system is flexible enough to be connected with different types of camera, both with analog and digital interface. This decoder provides the desired 8-bit 4:2:2 YCbCr output for subsequent digital processing. The decoder is not equipped with an internal buffer memory so it provides a continuous stream of data to the video input port of the processor. The digital stream is then conveyed to the digital input port of DM365. The processor performs the pixel rearrangement by means of an external small SDRAM (approx. 16 MB) and then subsequently copies the corrected frames to the video output port. The digital video is converted by internal D/A to component or composite analog video. Thus, the proposed architecture is suitable for both analog input and output.

Figure 16 provides more details of the internal processor data flow between its internal registers, video input port, video output port and the external SDRAM hosting the LUT plus the input and output frame buffers.

8 Implementation results

This section shows the implementation results of the developed system both hardware and software aspects.

8.1 Developed hardware platform

During this work a custom board equipped with the peripherals described previously has been developed with the industrial partner R.I.Co. srl. The board in Fig. 17 was designed for a wider video system usage and, thus, is equipped with other peripherals not used by the fish-eye correction application such as audio in/out, USB support, Ethernet support, RS-232 level shifter for UART communication and others. The devices circled in Fig. 17 by red lines represent those who have a primary functionality for the fish-eye correction, these are respectively (from top left to bottom right) RAM memory, DM365 SoC and NAND memory (to store the program), instead unnecessary devices are circled by yellow lines. As you can see the dimensions of the system for fish-eye correction can be reduced down to a 20 mm square (4 cm^2) PCB using small connectors for external peripherals.

The developed hardware has a cost of about 40$ for small production scale but, as previously said, it is equipped with some peripherals that can be removed for fish-eye system production, for which the expected cost can be reduced down to 20–25$ each. Indeed, the cost of the DM365 processor core for large volume market is in the order of few $. Such values are very interesting for the automotive market if compared to state-of-the-art solutions.

We made several tests for estimating power consumption; some of the not used components of the board have been disabled for the test, so only the key parts have been included in the measurement. We saw that the start-up phase is more power consuming than the execution one. This is because the calculation of the indexes involves many floating-point operations. Instead the

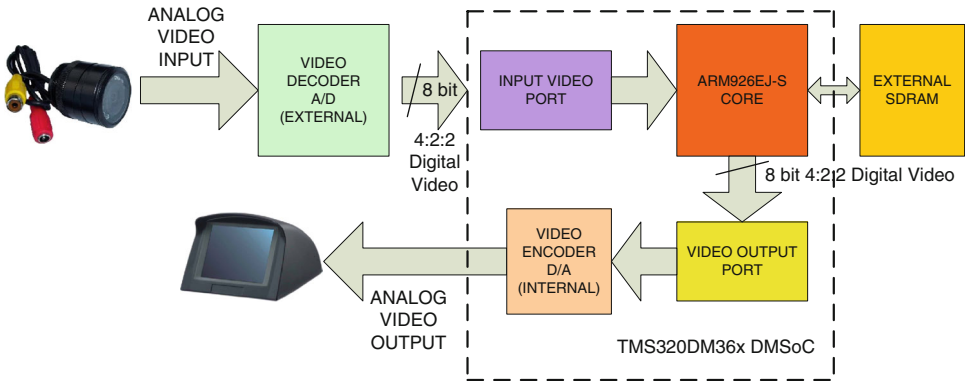


Fig. 15 Block diagram and signal flow of the fish-eye correction system

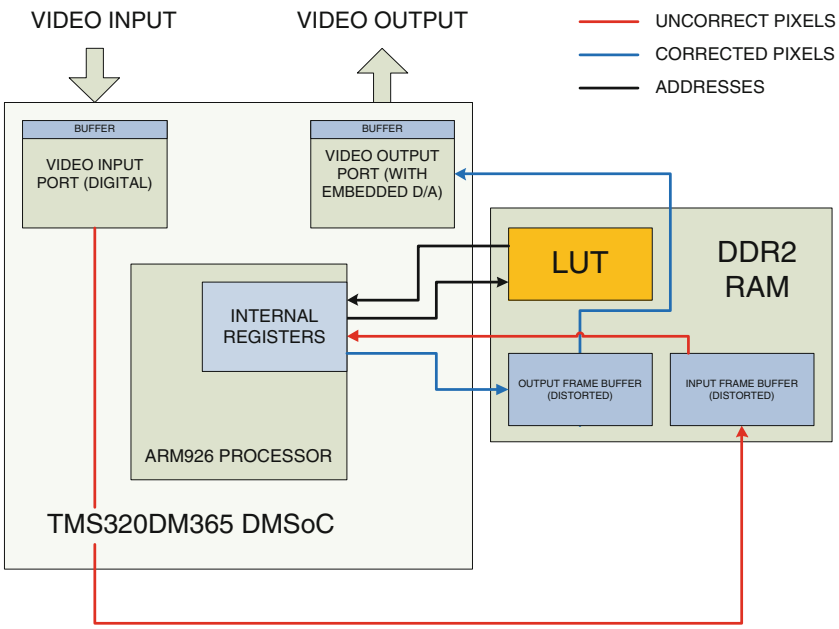


Fig. 16 Representation of internal processor data flow

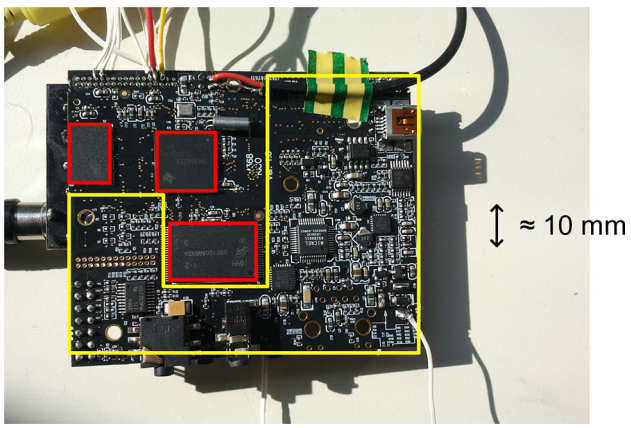


Fig. 17 Developed custom hardware board

remap process basically uses load and store instructions. Due to scaled voltage and frequency values, the simple core architecture and the custom board hardware design, we obtained a power consumption of about 100 mW. This quantity was measured by means of a laboratory power supplier. A typical power consumption graph is reported in Fig. 18.

We explored also the possibility to develop a multi-camera correction with the same hardware, while the memory is enough to contain data for remapping up to eight cameras, the bus throughput is enough only for real-time correction of two cameras with a reduced frame size or a reduced frame rate. Since, the developed board can manage only one video signal at time, it is necessary to put an external multiplexer to mix the signals from the two cameras. The fish-eye correction is, hence, performed with time division frame by frame.

The maximum frame rate is actually 25 fps with the maximum frame size of 720×576 and 30 fps with 640×480. The process is quite fast, we calculated a rough delay of 100 ms for the entire remapping of a 720×576 frame, so it is quite negligible.

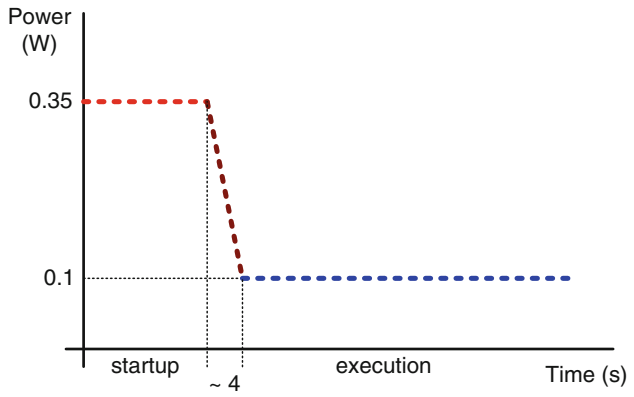


Fig. 18 Typical power consumption graph

8.2 Features

The developed system has the possibility to perform real-time fish-eye correction with a given remap function without the need of factory customization, by simply choosing the desired configuration before start-up. It is possible to use the same system with different types of camera by simply using one or more custom selector on the board, e.g., DIP switches, which let the OEM select some of the pre-compiled features:

- Standard of input and output (e.g., NTSC, PAL).
- Frame size, both input and output.
- Vertical and horizontal shift.
- Remapping function to be used for the correction or no correction.

So the OEM has the highest flexibility to adapt the system to many different types of camera and display. That operation could be ideally made also by the final user through a GUI interface running on the same display of the car infotainment system.

An example of the correction algorithm was previously shown in Fig. 11. More examples from real scenarios are provided in Figs. 19, 20 and 22. In Fig. 21 you can see a typical parking situation: a car is running in reverse obliquely towards a park line, but the uncorrected video (Fig. 22a) shows a stationary car as it is almost frontal. This misunderstanding can be the cause of an accident so it is important to correct the image (Fig. 22b) to see the correct angular orientation of the view and the real distance from the car.

The pictures were captured with the MT9V125 CMOS sensor by Micron Aptina operating at 25 fps with a resolution of 720 × 480 pixels per frame.

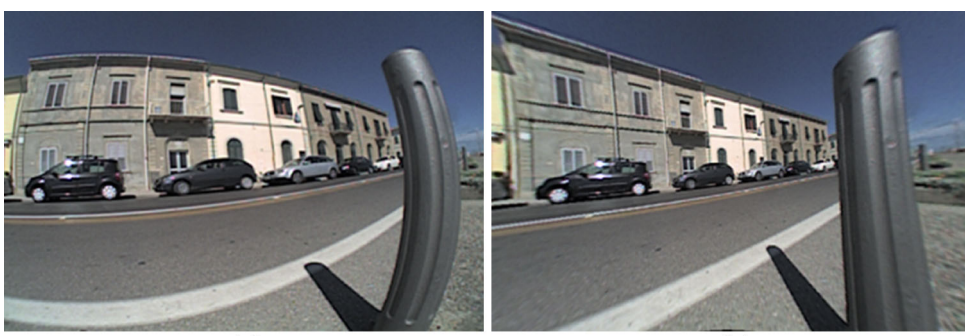
As introduced in the previous section, during start-up the software computes the address locations for all remap pixels (the so-called LUT) and stores them in an external RAM: this step would take an approximate time of some seconds (up to 4), depending on the type of mapping function to be used and the resolution of the capture and display buffer. After the initialization step the algorithm starts real-time acquisition, remap and display of the frames captured by the camera.

8.3 Memory resources

Regarding the memory (external RAM) occupation it depends on the frame size, see Tables 1 and 2. Table 1 shows the memory occupation of the computed LUT, necessary for the continuous remap of the pixels. This LUT must be stored in RAM during remap. Instead Table 2 shows the memory occupation of the input and output buffer. As seen before a full frame is stored in the input buffer, when captured by decoder, then a target (output) buffer is used to store the new, undistorted, image. Summarizing, the memory requirements can be easily faced by means of a small and affordable RAM module, like for example 16 MB. No other memory module is required for the remap operations. Memory occupation of the program itself is much lower than 1 MB.

8.4 Comparison with state-of-the-art solutions

Regarding software implementation, several advantages are obtained vs. the state-of-the-art solutions based on



(a) **(b)**

Fig. 19 Example result: **a** picture as captured from fish-eye camera, **b** corrected image

Fig. 20 Example result: **a** picture as captured from fish-eye camera, **b** corrected image

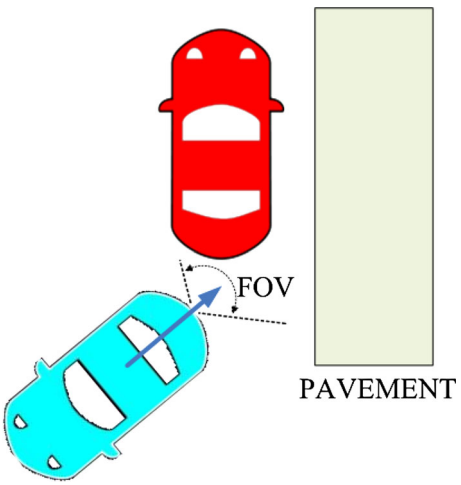


Fig. 21 Typical parking situation

FPGA. Being correction LUT computed during the initial transient phase no extra NV memory, such SD-card, is required. During the correction process it is possible to adaptively change the correction LUT and, hence, the applied correction effect. The system also allows for adapting camera and display with different frame size and can acquire region of interest of the input frame. Finally, it is possible to add an extra feature such as digital zoom, with the same hardware.

Table 1 LUT dimension for various frame resolutions

LUT dimension for various resolutions	
Resolution of the image	Dimensions of LUT (KB)
640 × 480 (VGA)	600
800 × 600 (SVGA)	937.5
720 × 480 (NTSC 16:9)	1350
720 × 576 (PAL 16:9)	1620

With respect to software-based fish-eye correction systems realized off-line [23, 24] or real-time on powerful processing platforms [20], thanks to algorithmic and architectural optimizations, the proposed system allows for the flexibility of a software solution but with power consumption of less than 100 mW, board size of few cm^2 and a cost of few tens of $. Another aspect to be noticed as important feature for automotive applications, where lighting conditions are often not optimal, is that the correction is not influenced by lighting conditions, since the algorithm operates just on pixel location and not on pixel value.

A direct comparison with commercial and state-of-the-art solutions shows that our system can be easily applied to different types of lens without hardware modifications, just sliding a simple DIP switch. With respect to every PC based solutions [20] the developed system can reach the

Fig. 22 Example result: **a** stationary car as captured from fish-eye camera, **b** corrected image that shows the exact angular position of the car

Table 2 Dimension of input and output buffers for various resolutions

Dimension of I/O buffers for various resolutions	
I/O color space and resolution	Total dimensions of two buffers (KB)
YCbCr 4:2:2 640 × 480 (VGA)	1200
YCbCr 4:2:2 800 × 600 (SVGA)	1875
YCbCr 4:2:2 720 × 480 (NTSC 16:9)	2700
YCbCr 4:2:2 720 × 576 (PAL 16:9)	3240

same performance within a processing power of few hundreds of mW, instead of few tens of Watts.

Another important aspect of the developed system, with respect to every state-of-the-art FPGA-based solution, is the reduced cost for hardware components and their integration due to the selection of a full integrated system-on-chip equipped with input and output video ports, D/A, and formatters.

8.5 Future enhancements

The DM36x chip integrates several on-chip peripherals not used for the fish-eye correction system, but dedicated to other video tasks such as H.264 coding. It also has USB support and UART ports. This way the proposed platform can be concurrently used for other processing tasks on-board the car. Indeed, future enhancements will be explored by testing the on-chip Enhanced DMA peripheral to let the processor being free to do other tasks. This can be useful to implement digital zoom and interpolation of pixels or manage multiple cameras. We are also planning to try a low-complexity real-time operating system to implement correction as a task and concurrently perform other tasks.

As said before our system has the capability to manage up to two different fish-eye cameras, this can result in a 4D (three space dimensions and time) video system, allowing the development of anti-collision and distance detector for automotive application.

Another potential application is in conjunction with a four-way video multiplexer: managing up to four different video signals, an all-round vision of the car (either front or rear according to the selected cameras) can be obtained.

9 Conclusions

This paper presented a flexible and low-cost solution for the real-time correction of the fish-eye distortion effect for automotive application. With respect to the state-of-the-art the proposed platform provides a solution that can be easily adapted to different types of lens or frame size, meeting real-time constraints with a power budget within 100 mW and a board size of few cm^2. Experimental results from real automotive scenarios confirm the quality of the corrected images. These results were achieved through optimization adopted at different levels: at algorithmic level, where a LUT correction technique with a new LUT exploitation is introduced avoiding extra NV off-chip memory cards; at data transfer level, where a new pixel pair management method reduces memory access and storage burden; at HW-SW implementation level, where a low-power board has been developed and tested in a real automotive scenario.

Acknowledgments We would like to thank Prof. M. Diani who lent us the EvmDM642 board and R.I.Co. srl that supported the research.

References

1. Yamada, K., Soga, M.: “A compact integrated visual motion sensor for ITS applications”. In: IEEE transactions on intelligent transportation systems, vol. 4, no. 1, pp. 35–42 (2003)
2. McCall, J.: “Video-based lane estimation and tracking for driver assistance: survey, system, evaluation” In: IEEE transactions on intelligent transportation systems, vol. 7, no. 1, pp. 20–37 (2006)
3. Soga, M., Kato, T., Ohta, M., Ninomiya, Y.: “Pedestrian detection with stereo vision”, ICDE Workshops 2005
4. He, Z. et al.: “Video-based measure of traffic volume parameter”, IEEE International conference automation and logistics, pp. 421–425 (2007)
5. Saponara, S., et al.: Algorithmic and architectural design for real-time and power-efficient Retinex image/video processing. J. Real-Time Image Process. **1**(4), 267–283 (2007)
6. Marsi, S., et al.: Integrated video motion estimator with Retinex-like pre-processing for robust motion analysis in automotive scenarios: algorithmic and real-time architecture design. J. Real-Time Image Process. **5**(4), 275–289 (2010)
7. Maddalena, S., Darmon, A., Diels, R.: “Automotive CMOS image sensors”, VDI-Buch, advanced microsystems for automotive applications, Part 6, pp. 401–412 (2005)
8. Römer, M., Heimann, T.: “Real-time camera link for driver assistance applications”, VDI-Buch, advanced microsystems for automotive applications, Part 3, pp. 299–310 (2009)
9. Azzopardi, M., et al.: A high speed tri-vision system for automotive applications. Eur. Transp. Res. Rev. **2**, 31–51 (2010)
10. Manipal Dot Net with Altera Corporation: “A flexible architecture for fisheye correction in automotive rear view cameras”, white paper (2008)
11. Manipal dot Net and Altera: “generating panoramic views by stitching multiple fisheye images”, white paper (2009)
12. Dhane, P., et al.: “A generic non-linear method for fisheye correction”, Int. J. Comput. Appl., vol. 51, no. 10, August (2012)
13. Wei, J., et al.: “Fisheye video correction”, IEEE Trans. Vis. Comput. Graph. **18**(10), pp. 1771–1783 (2012)
14. Kun, B., et al.: “An image correction method of fisheye lens base on bilinear interpolation”, In: fourth international conference on intelligent computation technology and automation (2011)
15. Hughes, C., Glavin, M., Jones, E., Denny, P.: Wide-angle camera technology for automotive applications: a review. IET Intell. Transp. Syst. **3**(1), 19–31 (2009)

16. Friel, M., Hughes, C., Denny, P., Jones, E., Glavin, M.: Automatic calibration of fish-eye cameras from automotive video sequences. IET Intell. Transp. Syst. **4**(2), 136–148 (2010)
17. Thomas, B., et al.: "Development of a cost effective bird's eye view parking assistance system". In: International journal of advanced research in computer science and software engineering (IJARCSSE) 2011, pp. 461–466
18. Bellas, N.: "Real-time fisheye lens distortion correction using automatically generated streaming accelerators". In: 17th IEEE symposium on field programmable custom computing machines (2009)
19. Hughes, C., et al.: "Review of geometric distortion compensation in fish-eye cameras", ISSC, pp. 162–167 (2008)
20. Bangadkar, S., et al.: "Mapping matrix for perspective correction from fish eye distorted images", IEEE ICRTIT, pp. 1288–1292 (2011)
21. Saito, M., et al.: "People detection and tracking from fish-eye image based on probabilistic appearance model", IEEE SICE, pp. 435–440 (2011)
22. La Hung Manh et al.: "A small-scale research platform for intelligent transportation systems", IEEE ROBIO, pp. 1373–1378 (2011)
23. Panorama Tools [Online]. Available: http://panotools.sourceforge.net
24. Image Trends Fisheye-Hemi Plug-In [Online]. Available: http://www.imagetrendsinc.com/products/prodpage_hemi.asp
25. Intel Core 2 processor datasheet http://download.intel.com/design/processor/datashts/318732.pdf
26. Xylon logiBrics Technology, logiVIEW [Online] Available: http://www.logicbricks.com/Products/logiVIEW.aspx
27. Intersil Fisheye Image Correction Technology [Online]. Available: http://www.intersil.com/video/fisheye.asp
28. Salomon, D.: Transformations and projections in computer graphics, Springer, Berlin, (2006)
29. ITU Recommendation BT.601 [Online] http://www.itu.int/rec/R-REC-BT.601-7-201103-I/en
30. TI, "TMS320DM642 Video/Imaging Fixed-Point DSP", Oct. 2010, [Online] http://focus.ti.com/docs/prod/folders/print/tms320dm642.html
31. TI, "TMS320DM365 Digital Media System-on-Chip", June 2011 [Online] http://focus.ti.com/docs/prod/folders/print/tms320dm365.html
32. TI, "TMS320DM368 Digital Media System-on-Chip", June 2011 [Online] http://focus.ti.com/docs/prod/folders/print/tms320dm368.html

Author Biographies

Mauro Turturici received the Master of Science degree in Electronic Engineering in 2010 from the University of Pisa. Currently he has a research contract with the Department of Information Engineering, University of Pisa, working on electronic systems for healthcare and vision automotive systems. He is also a Ph.D. candidate in Information Engineering at the University of Pisa.

Sergio Saponara got the Master of Science degree, cum laude, and the Ph.D. in Electronic Engineering from the University of Pisa. In 2002 he was with IMEC, Leuven (B), as Marie Curie Research Fellow. Since 2001 he collaborates with Consorzio Pisa Ricerche. He is Associate Professor at University of Pisa in the field of electronic circuits and systems for telecom, multimedia, space and automotive applications. He co-authored more than 160 scientific publications and holds 10 patents. Sergio Saponara is also research associate of CNIT and INFN and served as guest editor of special issues on international journals and as program committee member of international IEEE and SPIE conferences. He is associate editor of the Journal of Real-Time Image Processing, Springer.

Luca Fanucci got the Master of Science and the Ph.D. degrees in Electronic Engineering from the University of Pisa in 1992 and 1996, respectively. From 1992 to 1996, he was with ESA/ESTEC, Noordwijk (NL), as research fellow. From 1996 to 2004, he was a senior researcher of the CNR in Pisa. He is Professor of Microelectronics at the University of Pisa. His research interests include VLSI architectures for integrated circuits and systems. Prof. Fanucci co-authored more than 180 scientific publications and he holds more than 20 patents. He was program chair of IEEE Euromicro DSD 2008 and IEEE DATE Designer's Forum. He is the director of the Microelectronics division of Consorzio Pisa Ricerche scarl.

Emilio Franchi received the Master of Science degree in Electronic Engineering from University of Pisa. Currently he is R&D manager of RICO srl, Castelfidardo (AN), Italy and collaborates with CUBIT technology lab, Polo Tecnologico di Navacchio, Italy and with the University of Pisa. In the past he worked for MITSUBA and he collaborated with main semiconductor companies, including Texas Instruments. His research interests are focused on electronic systems for industrial and automotive applications.