# A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation

K.T. Gribbon, C.T. Johnston, and D.G. Bailey
Institute of Information Sciences and Technology,
Massey University, Palmerston North, New Zealand
k.gribbon@massey.ac.nz, Christopher.Johnston.5@uni.massey.ac.nz, d.g.bailey@massey.ac.nz

## Abstract

This paper presents a novel FPGA implementation of a barrel distortion correction algorithm with a focus on reducing hardware complexity. In order to perform real-time correction in hardware the undistorted output pixels must be produced in raster order. To do this the implementation uses the current scan position in the undistorted image to calculate which pixel in the distorted image to display. The magnification factor needed in this calculation is stored in a special look-up table that reduces the complexity of the hardware design, without significant loss of precision. The application of bilinear interpolation to improve the quality of the corrected image is also discussed and a real-time approach is presented. This approach uses a novel method of row buffering to compensate for data bandwidth constraints when trying to obtain the required pixels for interpolation.

**Keywords**: FPGA, reconfigurable hardware, lens distortion, bilinear interpolation, camera calibration

## 1 Introduction

Image processing is often used to make non-contact measurements for real-time measurement applications. It is therefore vital to ensure that accurate measurements can be made from the captured images. If an analogue camera is used it must often be of greater resolution and quality than is needed for the particular application in order to compensate for losses incurred before digitisation of the image [1]. A digital camera facilitates early digitisation and therefore it is less crucial to compensate for these losses. This can lead to substantial cost savings since a digital camera with lower resolution can be used. However, the inexpensive and wide-angle lenses often used in low cost digital cameras are susceptible to barrel distortion, which can introduce significant errors into any measurements [2].

Barrel distortion occurs when the magnification at the centre of the lens is greater than at the edges. A higher quality lens can be used to correct for this but this comes at additional cost to the image capture system. Barrel distortion is primarily radial in nature, with a relatively simple one parameter model accounting for most of the distortion [3]. A cost effective alternative to an expensive lens is to algorithmically correct for the distortion using the model.

A microprocessor or DSP could be used as the implementation platform for such an algorithm if processing were done offline. However, to satisfy the operational constraints imposed by real-time processing at sixty frames per second the algorithm must be implemented in hardware. A fixed hardware approach using an application specific integrated circuit (ASIC) would have flexibility limitations making field programmable gate arrays (FPGA) a better choice.

An FPGA consists of a matrix of logic blocks that are connected by a switching network. Both the logic blocks and the switching network are re-programmable allowing application specific hardware to be constructed. As such, an FPGA offers a compromise between the flexibility of general-purpose processors and the hardware-based speed of ASICs. Performance gains are obtained by bypassing the fetch-decode-execute overhead of general-purpose processors and by exploiting the inherent parallelism of digital hardware, while at the same time maintaining the ability to change the functionality of the system with ease.

Design entry can be achieved using low-level development methods such as schematic capture or hardware description languages. Another alternative is to use system-level design languages such as Handel-C that take a more high-level approach [4].

## 1.1 Handel-C

Handel-C is a language developed by Celoxica that compiles algorithms written in a high-level C-like language directly into gate-level netlists. It is based on a subset of ANSI-C with syntax extensions for hardware design such as variable data widths, parallel processing and channel communication between parallel processing blocks. The language is designed to allow software engineers to express an algorithm without any knowledge of the underlying hardware.

Many algorithms are prototyped in higher-level programming languages like C and must be ported into VHDL or Verilog for implementation, which can increase the risk of errors. Handel-C avoids this problem by using a high-level language to design the algorithms and then directly compile them to hardware [5].

In Handel-C assignment statements take exactly one clock cycle. All other language statements such as control logic constructs are free and add zero additional clock cycles although they can increase combinatorial delays to the extent that the system clock cycle may need to be lengthened [4].

This paper presents an FPGA implementation of the barrel distortion correction and bilinear interpolation algorithms. Section two describes the essence of these two algorithms. Novel approaches to real-time hardware-based implementations of both algorithms are described in section three, with a focus on reducing hardware complexity. Finally, implementation results and conclusions are presented in sections four and five.

## 2 Algorithms

Barrel distortion occurs when the magnification of the lens decreases with axial distance causing each image point to move radially towards the centre of the image. This results in the characteristic “barrel” shape. The barrel distortion model [2] is:

$$r_u = r_d(1 + kr_d^2) \tag{1}$$

where $r_u$ and $r_d$ are the distance from the centre of distortion in the undistorted and distorted images respectively, as shown in Figure 1, and $k$ is the distortion parameter, which is specific to the lens.

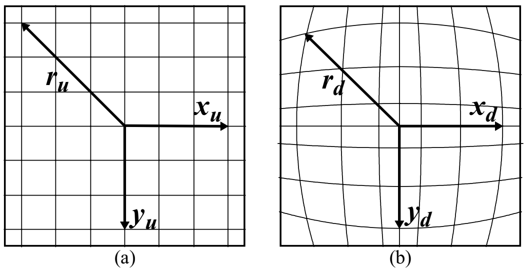


**Figure 1:** Illustration of barrel distortion model

Distortion correction calculates the location of the pixel in the distorted image that needs to be displayed. Unfortunately, these calculated coordinates are rarely integer values. This means that the location lies “between” the pixels in the original image. Bilinear interpolation is the method used to deal with this problem.

A block diagram of the complete system is shown in Figure 2. The system is driven entirely from the scan position of the display. Distortion correction is performed by using the current scan position and a magnification factor held in a look-up table (LUT) to calculate the address of the corresponding distorted pixel that is located in video RAM. This address is used as an input into the bilinear interpolation block which obtains the values of the pixels in the neighbourhood of the input address. Interpolation is performed and the result is output to the display.

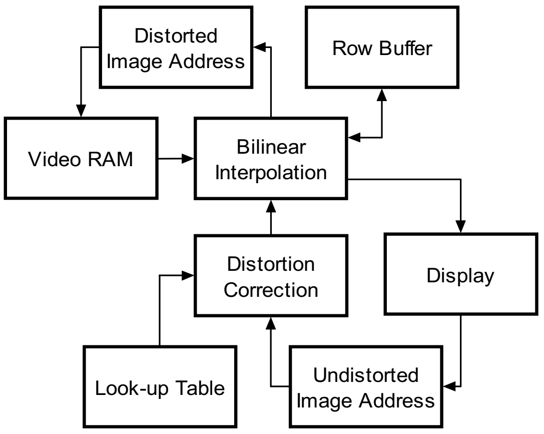


**Figure 2**: System diagram

### 2.1 Distortion Correction

The barrel distortion model effectively gives the coordinates in the undistorted image as a function of those of the distorted image. This form is unsuitable for real-time correction because it is necessary to produce the undistorted output pixels in raster order. In other words the coordinates in the undistorted image must be used to determine which pixel in the distorted image should be displayed. Therefore the equation needs to be of the form:

$$r_d = F(k, r_u) \tag{2}$$

As $r_u$ is calculated as:

$$r_u = \sqrt{x_u^2 + y_u^2} \tag{3}$$

which involves two multiplications and one square root, this function is very costly in terms of resources. Therefore it is preferable to have the model in terms of $r_u^2$. Equation (1) may be rewritten in terms of a radial dependent magnification:

$$x_d = x_u M(k, r_u^2), \quad y_d = y_u M(k, r_u^2) \tag{4}$$

where the magnification factor, $M(k,r_u^2)$ is:

$$M(k,r_u^2)=\frac{r_d}{r_u}=\frac{1}{1+kr_d^2} \qquad (5)$$

Unfortunately this equation is still in terms of $r_d^2$ rather than $r_u^2$. This can be solved by substituting equation (4) into equation (5), obtaining:

$$M=\frac{1}{1+kM^2r_u^2} \qquad (6)$$

where $M=M(k,r_u^2)$.

Equation (6) may be used to iteratively solve for $M$ until $M$ converges to the desired precision. It can be shown empirically that $M$ converges for $kr_u^2$ in the range $\frac{4}{27}$ to 4.

Figure 3 shows the resultant magnification function. Clearly, the mapping depends on the product $kr_u^2$, so this avoids the need to have a separate mapping for each $k$. A single mapping allows $M$ to be precalculated and placed in an LUT.

By normalising $x$ and $y$ coordinates based on the size of the image, $r_u^2$ can be constrained in the range of zero to one. After normalisation $k$ is also in the range of zero to one, where a value of one corresponds to severe distortion. This normalisation results in $kr_u^2$ also being in the range zero to one.

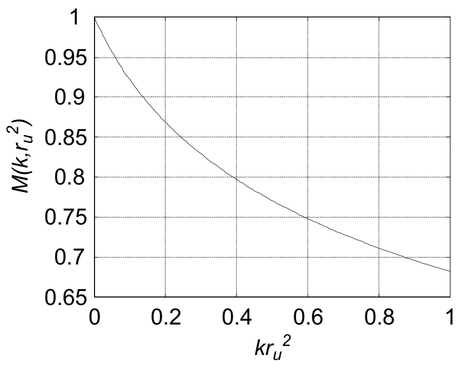


**Figure 3:** Magnification factor from pixel radius

## 2.2 Bilinear Interpolation

In a simplistic implementation, the calculated coordinates may be truncated so that the fractional component is discarded. Truncation, or alternatively rounding introduce substantial error in the pixel location, which in turn distorts lines by producing jagged-edge artefacts.

Bilinear interpolation gives improved results while providing a satisfactory compromise between computational efficiency and image quality [6]. The algorithm obtains the pixel value by taking a weighted sum of the pixel values of the four nearest neighbours surrounding the calculated location as illustrated in Figure 4 and equation (7) below.

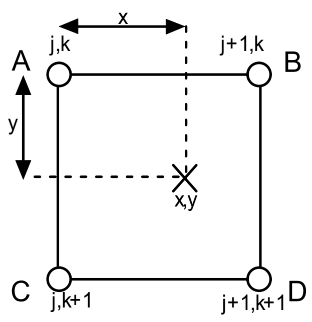


**Figure 4:** Bilinear interpolation neighbourhood

$$X_{x,y}=aA+bB+cC+dD \qquad (7)$$

where
$$a=(1-x)(1-y) \quad b=(1-y)x$$
$$c=(1-x)y \quad d=xy$$

## 3 Design and Implementation

The hardware used to support this implementation is the RC100 prototyping and development board from Celoxica, which incorporates a Xilinx Spartan-II FPGA, video decoder, offchip RAM and video DAC.

The Spartan-II device on the RC100 board is the XC2S200 which has the equivalent of 200,000 system gates and 14 blocks of 4Kbits RAM. The block RAM is synchronous and dual-ported allowing two simultaneous accesses per clock cycle.

A relatively low-level data flow approach at the register transfer level (RTL) was taken since this makes explicit the distortion correction and interpolation pipelines.

### 3.1 Fixed Point

A fixed point representation was used. Therefore all variables were represented as signed or unsigned integers and were commented to indicate the position of the binary point. When arithmetic operations were performed operands were explicitly shifted to ensure alignment.

### 3.2 Constraints

Distortion correction and bilinear interpolation are trivial tasks to perform “offline” or in software but a real-time implementation on the RC100 board presents a number of problems and constraints.

#### 3.2.1 Real-time Constraints

One pixel value must be provided to the VGA output each clock cycle since each statement in Handel-C takes one clock cycle to execute. This constrains the design into performing all of the required calculations each clock cycle. A pipelined approach is thus needed that outputs an interpolated pixel value to the video driver every clock cycle with several clock cycles of latency between the input and output. This delay can be accommodated by starting the pipeline calculations during the horizontal blanking period.

### 3.2.2 Memory Bandwidth Constraint

The implementation must also perform scan rate conversion because the input signal is composite PAL with a frame rate of 25 Hz which must then be processed and output to the VGA display at 60 Hz.

To perform this conversion two banks of single-port offchip RAM on the RC100 board are used. The stream from the video decoder is deinterlaced and written to one bank while two pixel values are read from the other bank and processed. When the video decoder reaches the end of a frame the roles of the banks are swapped.

Accesses to off-chip RAM (both reading and writing) take a single clock cycle to complete. However, bilinear interpolation requires simultaneous access to four pixels from the input image (see equation (7)). Constrained to only one read from video memory per clock cycle we need to obtain the values of the remaining neighbouring pixels. This implies the use of some method of buffering.

## 3.3 Distortion Correction

Due to the raster nature of the output, $y_u^2$ is constant for a line but $x_u^2$, and therefore $M(kr_u^2)$, changes for each pixel in the output. Obviously it is impractical to evaluate $M(kr_u^2)$ directly for each pixel and therefore a LUT implementation was considered using block RAM resources on the FPGA.

For a 512 pixel image, the magnification factor requires a precision of at least 10 bits, with more being necessary to enable bilinear interpolation within the distorted image. Each Block RAM can provide 256 look-up entries. Analysing the mapping in Figure 3 indicates that 256 entries covering $kr_u^2$ in the range from zero to one will only provide 8 bits of accuracy. To get 10 bits requires four block RAMs; while 12 bits will use 16 block RAMs, more than what is available on the target device.

To overcome this problem, a single 256 entry LUT is used, with interpolation performed between the table values to improve accuracy. The top 8 bits of $kr_u^2$ ($kr_u^2|_{MSB}$) are used as an index to retrieve the magnification value from the look-up table. The next entry is also retrieved (the block RAM is dual port). The difference between these gives the slope, which is scaled by the lower 8 bits of $kr_u^2$ ($kr_u^2|_{LSB}$):

$$\begin{aligned} M(k,r_u^2) &= M(kr_u^2) \\ &\approx M(kr_u^2|_{MSB}) + \frac{kr_u^2|_{LSB}}{2^8}\left(M(kr_u^2|_{MSB}+1) - M(kr_u^2|_{MSB})\right) \end{aligned} \tag{8}$$

Since the most significant bit of $M$ is always 1, the look-up table does not need to store this, allowing the table to contain $M$ to 17 binary places. When combined with interpolation the accuracy of the approximation is about 15.5 bits for 16 bits precision of $kr_u^2$.

### 3.3.1 Hardware Reduction

The calculation of $x_u^2$ and $y_u^2$ can avoid using a multiplier by making use of the fact that

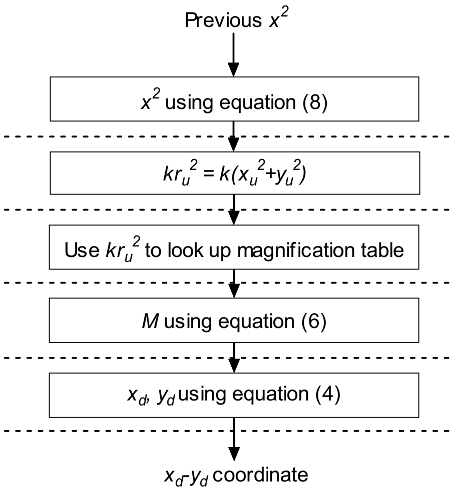
$$(x_u+1)^2 = x_u^2 + 2x_u + 1 \tag{9}$$

and the image is scanned in a raster fashion. In hardware this is equivalent to $x_u$ with 1 appended to the bottom bit and added to the previous $x_u^2$.

In Handel-C, functions can be shared by multiplexing the input and output. As $x_u^2$ and $y_u^2$ are never calculated at the same time the hardware for equation (9) can be shared.

### 3.3.2 Pipeline

Based on the discussion of the algorithm in sections 2.1 and 3.3 a five-stage pipeline shown below in Figure 5 is used to determine the coordinates of the pixel in the distorted image to be output. The dotted lines in Figure 5 represent registers.



Previous $x^2$
$x^2$ using equation (8)
$kr_u^2 = k(x_u^2+y_u^2)$
Use $kr_u^2$ to look up magnification table
$M$ using equation (6)
$x_d$, $y_d$ using equation (4)
$x_d$-$y_d$ coordinate

**Figure 5:** Pipeline for distortion correction algorithm

## 3.4 Bilinear Interpolation

The $x_d$-$y_d$ coordinates output from the distortion correction pipeline are not presented in a raster- based fashion. Instead they follow smooth "curves". This makes traditional row buffering ineffective because of the curved nature of the mapping.

The constantly changing magnification also means that the step size from one pixel to the next is not uniform. However, since the magnification is less than one, calculated coordinates taken from the distorted image will be less than one pixel apart and at an angle less than 45 degrees. Therefore, depending on the location of the previous coordinate we will always have at least one, and possibly up to four corner pixels in common with the previous calculation.

The design of the bilinear interpolation algorithm focused on finding a way of obtaining the remaining neighbouring pixels using only one video memory access per clock cycle and a novel row buffering method. To determine what pixel values are needed an analysis of the various scenarios relating to the current and previous coordinates was undertaken.

### 3.4.1 Scenarios

There are six possible relationships between the current and previous coordinates (see Figure 6).

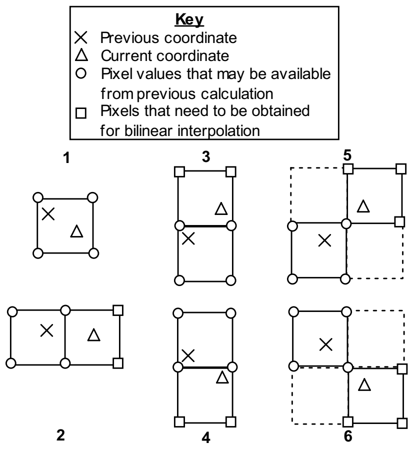

Key
× Previous coordinate
△ Current coordinate
○ Pixel values that may be available from previous calculation
□ Pixels that need to be obtained for bilinear interpolation
1
2
3
4
5
6

**Figure 6:** Scenarios

Scenario one is the simplest as all four pixel values from the previous interpolation calculation are available. In scenarios two, three and four only two pixels are available and in scenario five and six only one pixel is available from the previous calculation.

### 3.4.2 Buffering

The buffer is constructed using on-chip RAM blocks with a length equal to the number of horizontal pixels. The integer part of the $x_d$-coordinate is used as the address with a row offset and pixel value stored at the corresponding location.

The RAM is dual-ported and the timing cycle altered so that pixel values can be read and updated in a single Handel-C clock. The pixel value to be written into the buffer on each clock cycle must be chosen carefully in order to maximise the likelihood of it being used for interpolation on the next line.

Since output to the display is performed in a raster-based fashion, coordinates also appear going from left to right and downwards even if a curved path is taken for the line. Based on Figure 6 it is useful to write the bottom-right pixel value used in the current interpolation to the buffer. The bottom-right pixel is more likely to be used in subsequent calculations because the image is being scanned from top to bottom and from left to right. The next row will be less than one pixel apart and therefore only a single bit of the $y_d$-coordinate is needed for the $y$-offset.

Figure 7(a) shows one possible scenario and how to retrieve the four surrounding pixels in a single clock cycle. The left pixel values are obtained from the previous interpolation. After reading the buffer contents of the address corresponding to the current location, the $y$-offset indicates that it corresponds to the top-right pixel of the neighbourhood. Thus we need to read the bottom-right pixel from video RAM and write this result into the buffer RAM.

In some scenarios it may not possible to retrieve all four neighbouring pixel values and therefore a three-point interpolation must be used, an example of this scenario is shown in Figure 7(b).

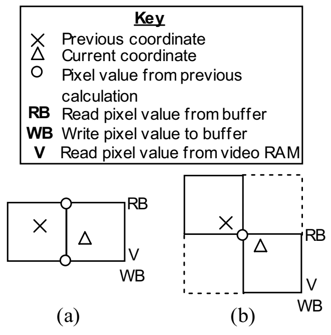

Key
× Previous coordinate
△ Current coordinate
○ Pixel value from previous calculation
RB Read pixel value from buffer
WB Write pixel value to buffer
V Read pixel value from video RAM
RB
V
WB
(a)
RB
V
WB
(b)

**Figure 7:** Scenario examples

### 3.4.3 Pipeline

Based on the discussion in the preceding sections a four-stage pipeline was developed for the interpolation algorithm, shown below in Figure 8.

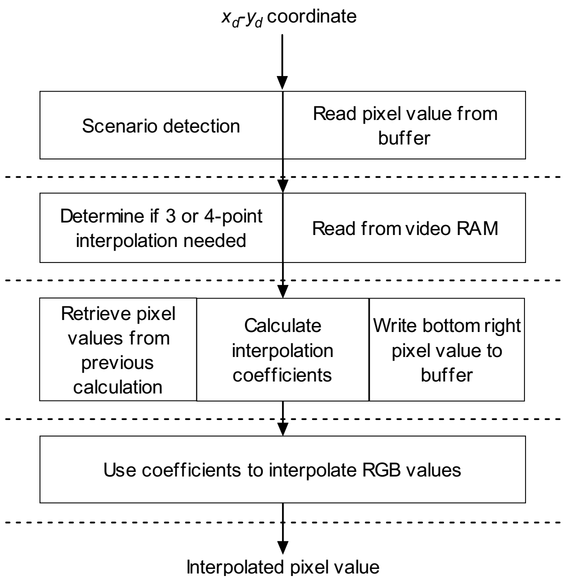

$x_d$-$y_d$ coordinate
Scenario detection
Read pixel value from buffer
Determine if 3 or 4-point interpolation needed
Read from video RAM
Retrieve pixel values from previous calculation
Calculate interpolation coefficients
Write bottom right pixel value to buffer
Use coefficients to interpolate RGB values
Interpolated pixel value

**Figure 8:** Pipeline for bilinear interpolation algorithm

When starting a frame the buffer will not have the values needed for the first line. Therefore it is necessary to prime the buffer by sending the first row of coordinates twice and discarding the erroneous interpolated pixels. This can be accomplished during the vertical blanking period.

## 4 Results

At present the distortion correction algorithm has been successfully implemented and tested on the RC100 development board (Figure 9(a) and (b)). The interpolation algorithm has been implemented but is presently being tested. The utilisation of the device is shown in Table 1.

**Table 1:** Resource utilisation of device (XC2S200)

| | **CLBS (1172 total)** | **Block RAM (14 total)** |
|---|---|---|
| **Keyboard interface** | 129 (11%) | 1 |
| **Video decoder / VGA** | 235 (20%) | 4 |
| **Correction algorithm** | 270 (23%) | 1 |
| **Bilinear interpolation** | 329 (28%) | 3 |
| **Total** | 971 (83%) | 9 (64%) |

The correction and interpolation algorithms together use approximately half the logic resources of the device, with most of this being used to implement the large multipliers. If this were implemented on an FPGA such as the Virtex-II that incorporates embedded multipliers the resource utilisation would be significantly reduced.

## 5 Conclusion

The conversion from a software algorithm to one that runs in hardware in real-time presents a number of difficulties. These include the inability to do offline processing, data bandwidth constraints and the need to minimise logic gate count.

The use of a look-up table with interpolation can reduce the complexity of the hardware design without significant loss of precision compared to calculating values at run-time.

## 6 Acknowledgements

The Celoxica University Programme for generously providing the DK1 Design Suite.

## 7 References

[1] Bainbridge-Smith, A. & Dunne, P., "FPGAs in computer vision applications", *Proceedings Image and Vision Computing New Zealand 2002*, pp 347-352 (2002).

[2] Bailey, D.G., "A new approach to lens distortion correction", *Proceedings Image and Vision Computing New Zealand 2002*, pp 59-64 (2002).

[3] Li, M., Lavest, S.M., "Some aspects of zoom lens camera calibration", *IEEE Trans on PAMI*, 18(11): 1105-1110 (1996).

[4] Alston, I., Madahar, B., "From C to netlists: hardware engineering for software engineers?", *IEE Electronics & Communication Engineering Journal*, pp 165-173 (August 2002).

[5] Celoxica Ltd., *HANDEL-C Language Overview*, (2002).

[6] Gonzalez, R.C., Woods, R.E., *Digital Image Processing,* 2nd Edition, Prentice-Hall, New Jersey (2002).

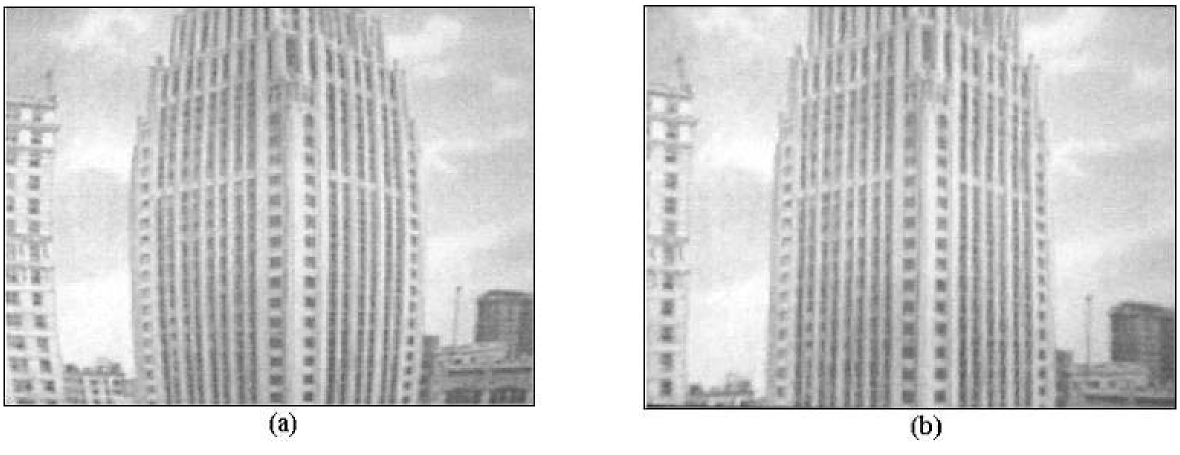


**Figure 9:** Results from correction algorithm