Preliminary Application Note

IMAPCAR2 Family

Pixel Remapping

Software

IMAPCAR2 Series

IMAPCAR2-200

IMAPCAR2-100

IMAPCAR2-50

Document No. U19951EE1V0AN00
Date Published: November 2009

Legal Notes

The information in this document is current as of November 2009. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".
  The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
  "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.

(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

Table of Contents

1 Purpose of the document

Pixel Remapping is an image transformation. It can be used in two typical cases:

- Geometrical transformation such as rotation or rescaling
- Distortion correction such as fisheye effect, as shown in the example below

In this document, we should describe the implementation of this algorithm to 1DC, the specific language of IMAPCAR2.

A general overview of the different implemented concepts is shown in the first part with block diagrams and the description of the software architecture. Performances are resumed in the second part and source codes are attached in the appendix.

2 Correction matrix generation

The first step of pixel remapping is to create a correction matrix for the vertical and horizontal components. This process is done **offline**.

Two possibilities:

- The two matrixes contain the vertical and horizontal shift, that means:

  out (x,y) = in (x+u,y+v)

- The two matrixes contain the pixel position, that means:

  out (x,y) = in (coord_x,coord_y)

Note that shift is done from destination

The following sections should describe how to generate such matrixes using MATLAB.

2.1 Format specification

As IMAPCAR2 supports .PGM format (Portable GrayMap), shift matrixes should be in this format.

If the original format is not .PGM, rescaling and conversion has to be done offline with MATLAB. Supported format are *bmp*, *gif*, *jpeg*, *png*, *ppm*, and *tiff*.

Remarks

- If shift values have been rescaled (for example to fit the [0,255] interval entirely), do not forget to change the corresponding *SCALE_FACTOR*, *SCALE_BITS*, and *CENTER_FACTOR* parameters in the header file *PixelRemappingParameters.h*
- IMAPCAR2 supports up to 12 bits images and 16 bits matrixes
- Do not forget to change the corresponding parameter set in the header file *PixelRemapping.h,* depending on the number of bytes per pixel for the image and matrixes

2.2 Standard matrix examples

2.2.1 Neutral matrix

No move is required. Matrixes are filled with zeros.

2.2.1.1 MATLAB Code

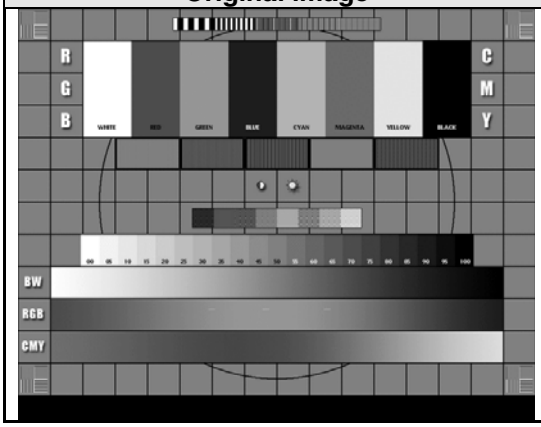
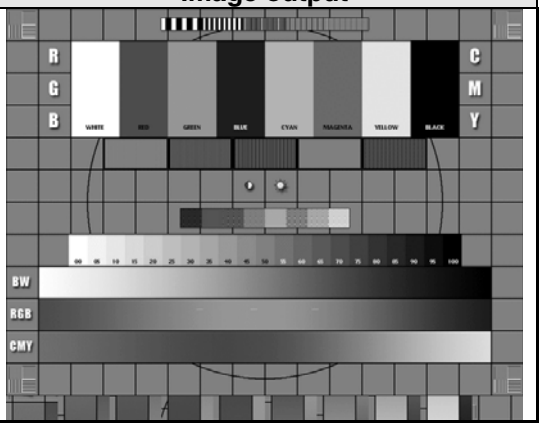
Matrix creation and rescaling

```
function [X,Y] = zero_matrix(width,height)
% This function generate zero-matrixes
% Inputs: matrix width and height
% Outputs:  shift matrixes

% Matrixes are entirely filled with zeros
X = zeros(height,width);
Y = zeros(height,width);

function [X_matrix]=rescale(X_matrix,scale_factor,center_factor)
 %This function rescales the matrix values
% For example to fill the [0,255] interval
% Inputs: matrix to rescale, scale factor and center factor
% To get the real values back: new = (old – center_factor) / scale_factor
% Output: resized and rescaled matrix

% Rescale between -128 and 128 in case of 8-bits matrix
X = floor(X*scale_factor);
% Center the values between 0 and 255 to get unsigned type as output
X = X + center_factor;
```

“main” function

```
% Matrix generation
 [X,Y] = zero_matrix(640,480);

% Rescaling
X1 = rescale(X,1,128);
Y1 = rescale(Y,1,128);

% if needed resize and rotate
X2 = zeros(512,640);
for i=1:480
    X2(i,:) = X1(i,:);
end
for i=481:512
    X2(i,:) = 0;
end

X2 = imrotate(X2,-90);

% Matrix save
imwrite(uint8(X1),'X_matrix_neutral.pgm','pgm','Encoding','ASCII');
imwrite(uint8(X2),'X_matrix_neutral_rotated.pgm','pgm','Encoding','ASCII');
imwrite(uint8(Y1),'Y_matrix_neutral.pgm','pgm','Encoding','ASCII');
```

2.2.1.2 Result example

Original image	Image output
	

2.2.2 Simple translation

This section shows how to generate translation matrixes. Each matrix is filled with a constant value.

2.2.2.1 MATLAB Code

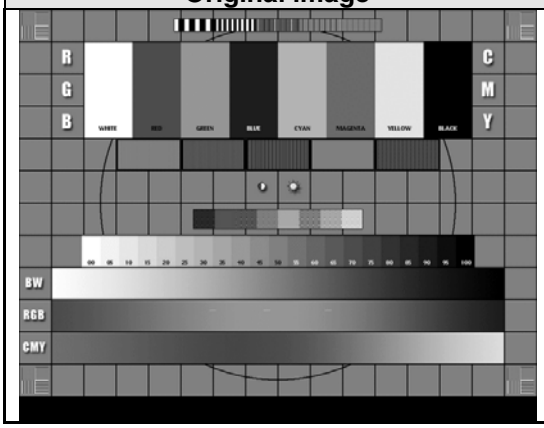
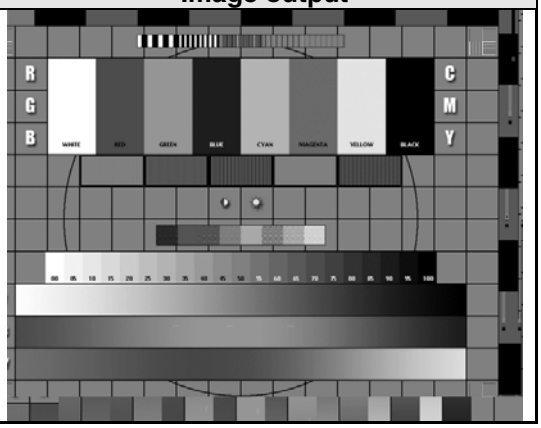
Matrix creation

```
function [X,Y] = simple_shift_matrix(width,height,x_shift,y_shift)
% This function generate constant matrixes

% Initialization
X = zeros(height,width);
Y = zeros(height,width);
% Fill-in
X(:,:) = -x_shift;
Y(:,:) = -y_shift;
```

“main” function

```
% Matrix generation
 [X,Y] = simple_shift_matrix(640,480,-27.5,18.25);

% Rescaling
X1 = rescale(X,4,120);
Y1 = rescale(Y,4,120);

% if needed resize and rotate
X2 = zeros(512,640);
for i=1:480
   X2(i,:) = X1(i,:);
end
for i=481:512
   X2(i,:) = 0;
end

X2 = imrotate(X2,-90);

% Matrix save
imwrite(uint8(X1),'X_matrix_translation.pgm','pgm','Encoding','ASCII');
imwrite(uint8(X2),'X_matrix_translation_rotated.pgm','pgm','Encoding','ASCII');
imwrite(uint8(Y1),'Y_matrix_translation.pgm','pgm','Encoding','ASCII');
```

2.2.2.2 Result example

Original image	Image output
	

2.2.3 Reduction or Zoom

This is to give an example of how to achieve an image reduction.

2.2.3.1 MATLAB Code

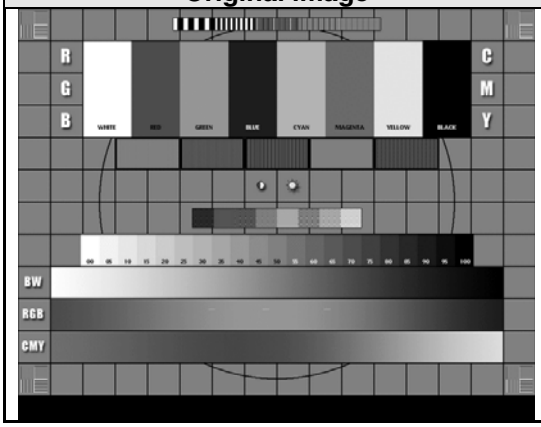
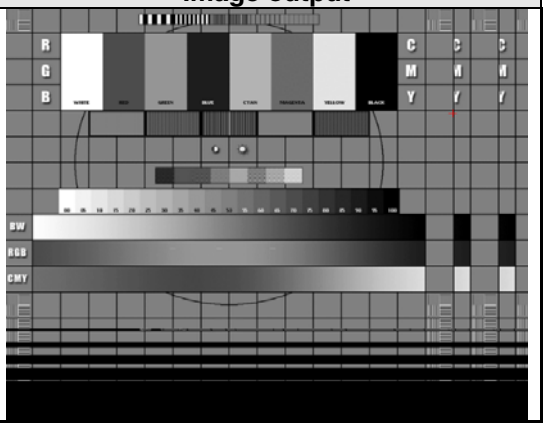
Matrix creation

```
function [X_shift,Y_shift] = reduction_matrix(width,height,x_scale,y_scale)
% This function generates reduction/zoom matrix
% Inputs: matrix width and height, as well as scaling factors
% Outputs: reduction matrixes

% Initialization
X_shift = zeros(height,width);
Y_shift = zeros(height,width);

% X-matrix fill
for u=1:width
   X_shift(:,u) = u*(1-x_scale);
end

% Y-matrix fill
for u=1:width
   for v=1:height
      Y_shift(v,u) = v*(1-y_scale);
   end
end
```

“main” function

```
% Matrix generation
 [X,Y] = reduction_matrix(640,480,5/6,3/4);

% Rescaling
X1 = rescale(X,8,0);
Y1 = rescale(Y,8,0);

% if needed resize and rotate
X2 = zeros(512,640);
for i=1:480
    X2(i,:) = X1(i,:);
end
for i=481:512
    X2(i,:) = 0;
end

X2 = imrotate(X2,-90);

% Matrix save
imwrite(uint16(X1),'X_matrix_reduction.pgm','pgm','Encoding','ASCII');
imwrite(uint16(X2),'X_matrix_reduction_rotated.pgm','pgm','Encoding','ASCII');
imwrite(uint16(Y1),'Y_matrix_reduction.pgm','pgm','Encoding','ASCII');
```

2.2.3.2 Result example

Original image	Image output
	

2.2.4 Simple Rotation

This section describes how to get shift matrixes for image rotation.

2.2.4.1 MATLAB Code

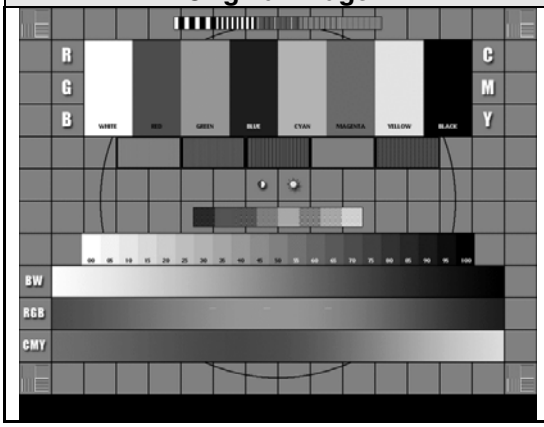
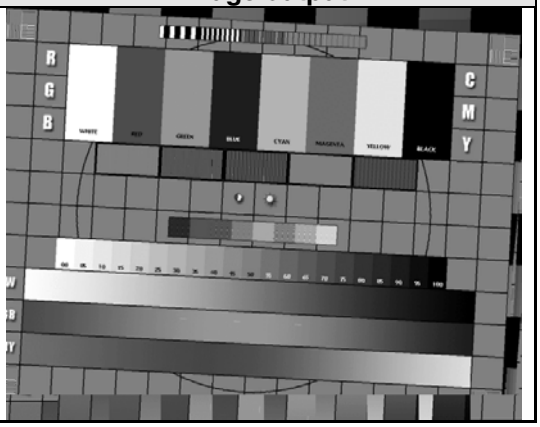
Matrix creation

```
function [X_matrix,Y_matrix] = rotation_matrix(angle,width,height)
% This function generates rotation matrixes
% Inputs: angle (in degrees), matrix width and height
% Outputs: generated matrixes

% Initialization
X_matrix=zeros(height,width);
Y_matrix = zeros(height,width);

 % Radian to Degree conversion
angle = angle*pi/180;

for v=1:height
   for u=1:width

      %2-D rotation by 'angle' degrees
      % Apply x' = x * cos(phi) - y * sin(phi)
      % and y' = x * sin(phi) + y * cos(phi)
      % Note that X -> new column y'
      %          Y -> new line x'
      %          u -> old column y
      %          v -> old line x
      X_matrix(v,u) = v * sin(angle) + u * cos(angle) - u;
      Y_matrix(v,u) = v * cos(angle) - u * sin(angle) - v;

   end
end
```

“main” function

```
% Matrix generation
 [X,Y] = rotation_matrix(3,640,480);

% Rescaling
X1 = rescale(X,4,140);
Y1 = rescale(Y,4,140);

% if needed resize and rotate
X2 = zeros(512,640);
for i=1:480
    X2(i,:) = X1(i,:);
end
for i=481:512
    X2(i,:) = 0;
end

X2 = imrotate(X2,-90);

% Matrix save
imwrite(uint16(X1),'X_matrix_reduction.pgm','pgm','Encoding','ASCII');
imwrite(uint16(X2),'X_matrix_reduction_rotated.pgm','pgm','Encoding','ASCII');
imwrite(uint16(Y1),'Y_matrix_reduction.pgm','pgm','Encoding','ASCII');
```

2.2.4.2 Result example

Original image	Image output
	

2.3 Particular case of camera calibration

2.3.1 Theory

In order to apply the inverse correction to the image, we need to find specific parameters. Indeed, each camera has its own physical parameters, namely:

- the focal length f_u
- the scale factor s_u
- the image center coordinates (u_0, v_0)
- the radial distortion coefficients k_1 and k_2
- the tangential distortion coefficients p_1 and p_2

From these parameters, we can obtain the corresponding parameters of the inverse model (a_1... a_8) using the following equation:

$$\mathbf{p} = \left(\mathbf{T}^T\mathbf{T}\right)^{-1}\mathbf{T}^T\mathbf{e}$$

where

$$\mathbf{u_i} = \left[-u'_i r_i^2, -u'_i r_i^4, -2\,u'_i v'_i, -\left(r_i^2 + 2\,u'^2_i\right), x_i r_i^4, x_i u'_i r_i^2, x_i v'_i r_i^2, x_i r_i^2\right]^T$$

$$\mathbf{v_i} = \left[-v'_i r_i^2, -v'_i r_i^4, -\left(r_i^2 + 2\,v'^2_i\right), -2\,u'_i v'_i, y_i r_i^4, y_i u'_i r_i^2, y_i v'_i r_i^2, y_i r_i^2\right]^T$$

$$\mathbf{T} = \left[\mathbf{u_1}, \mathbf{v_1}, \ldots, \mathbf{u_N}, \mathbf{v_N}\right]^T$$

$$\mathbf{p} = \left[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\right]$$

$$\mathbf{e} = \left[u'_1 - x_1, v'_1 - y_1, \ldots, u'_N - x_N, v'_N - y_N\right]^T$$

and

$$u'_i = \frac{u_i - u_0}{D_u s_u}$$

$$v'_i = \frac{v_i - v_0}{D_v}$$

$$r_i = \sqrt{u'^2_i + v'^2_i}$$

For detailed information about camera calibration, see the following paper:

http://www.vision.caltech.edu/bouguetj/calib_doc/papers/heikkila97.pdf

In our case, the calibration is done by a specific Matlab Toolbox.

2.3.2 Matlab Toolbox

This Toolbox is a modified version of the one which can be found at the following link:

http://www.vision.caltech.edu/bouguetj/calib_doc/

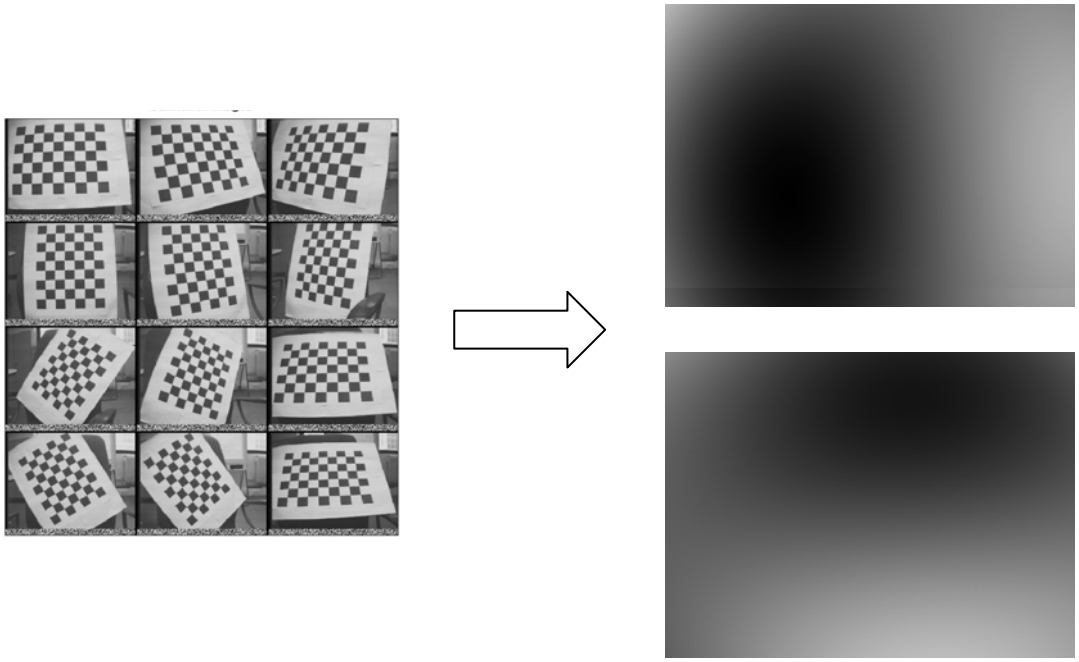
Indeed, the original toolbox calibrates the camera and provides corrected images as output. We only need the calibration part as we want to get the vertical and horizontal shift matrixes as output, so the toolbox was modified accordingly.

The use of this toolbox is very simple. We need to:

- provide a set of calibration image, generally consisting of chessboards in different positions
- click manually on the four corners of each image

Then, the toolbox can:

- guess the corners of the grid
- from this guesses, it computes the 8 physical parameters of the camera
- it computes the 8 parameters for the inverse model (see previous section)
- it provides finally the vertical shift and horizontal shift matrixes to apply for correcting the images



3 Pixel remapping middleware

3.1 IMAPCAR2 Implementation concepts

Several concepts are proposed. They have different advantages and drawbacks depending on the application we want to use.

3.1.1 Using 90 degrees rotations

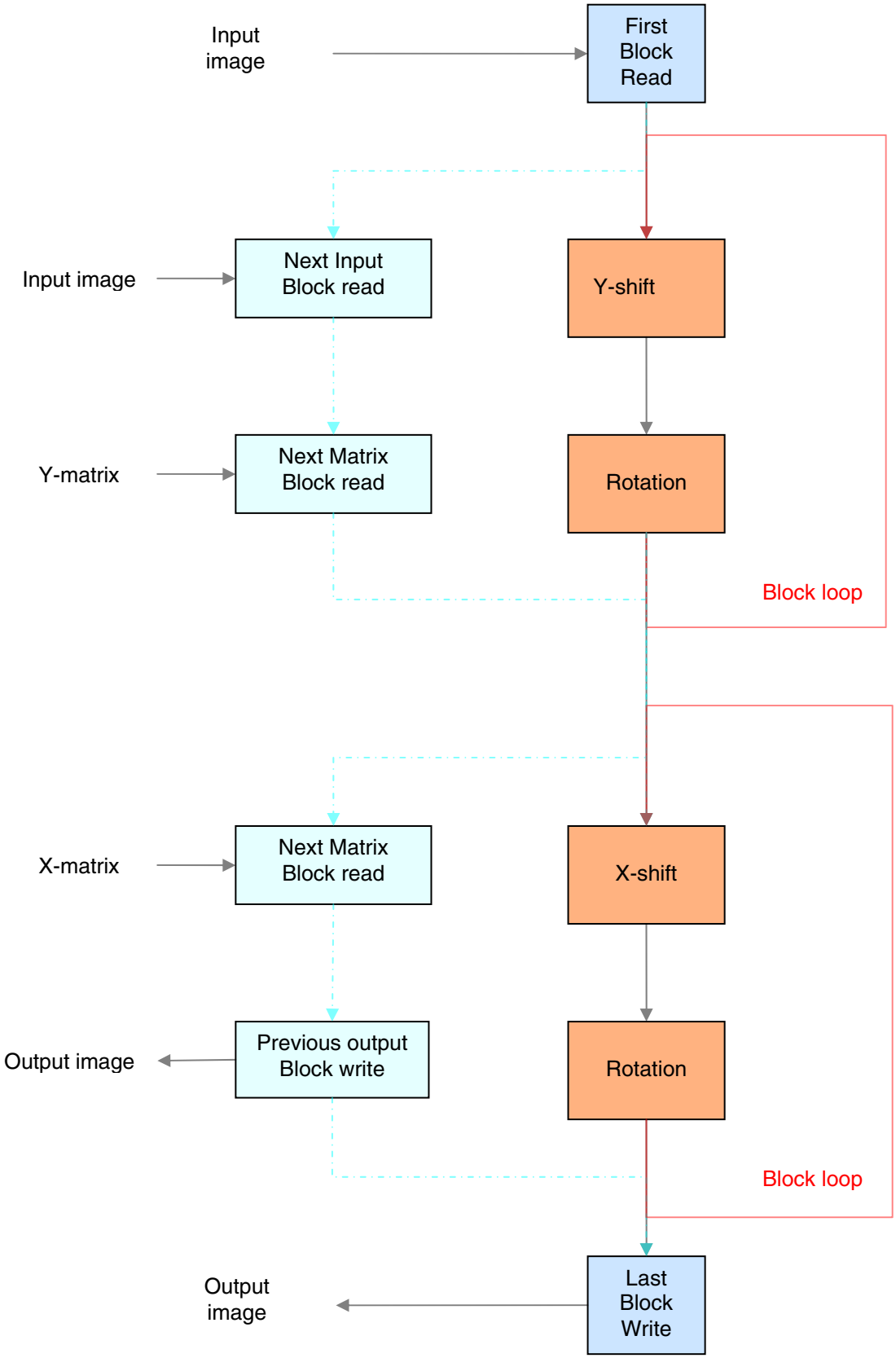
To use the processing elements array the more efficiently, process is entirely done in the vertical direction.

3.1.1.1 General diagram

To make the block diagram understandable, we need to add some remarks:

- The vertical and horizontal shifts are done separately but using rotation the process is nearly the same for both. Vertical shift is processed first.
- The image is processed per horizontal stripes. A horizontal stripe is a group of PENO lines of the image.
- Each stripe is divided in PENO*PENO square blocks in order to simplify the rotation process.
- Memory transfers are done in background as much as possible.

The overall process looks as follows:



Notes

- Vertical shift output is a uniform horizontal stripe. Therefore, **input blocks have different sizes** depending on Y-shift values. These sizes are computed offline.
- Although the horizontal stripe is processed per blocks, we wait for the entire horizontal stripe to be processed before applying the horizontal shift. Indeed, in case of big distortion (an important compression for example), we need several blocks to be available at the same time. That is why there are two loops instead of one.

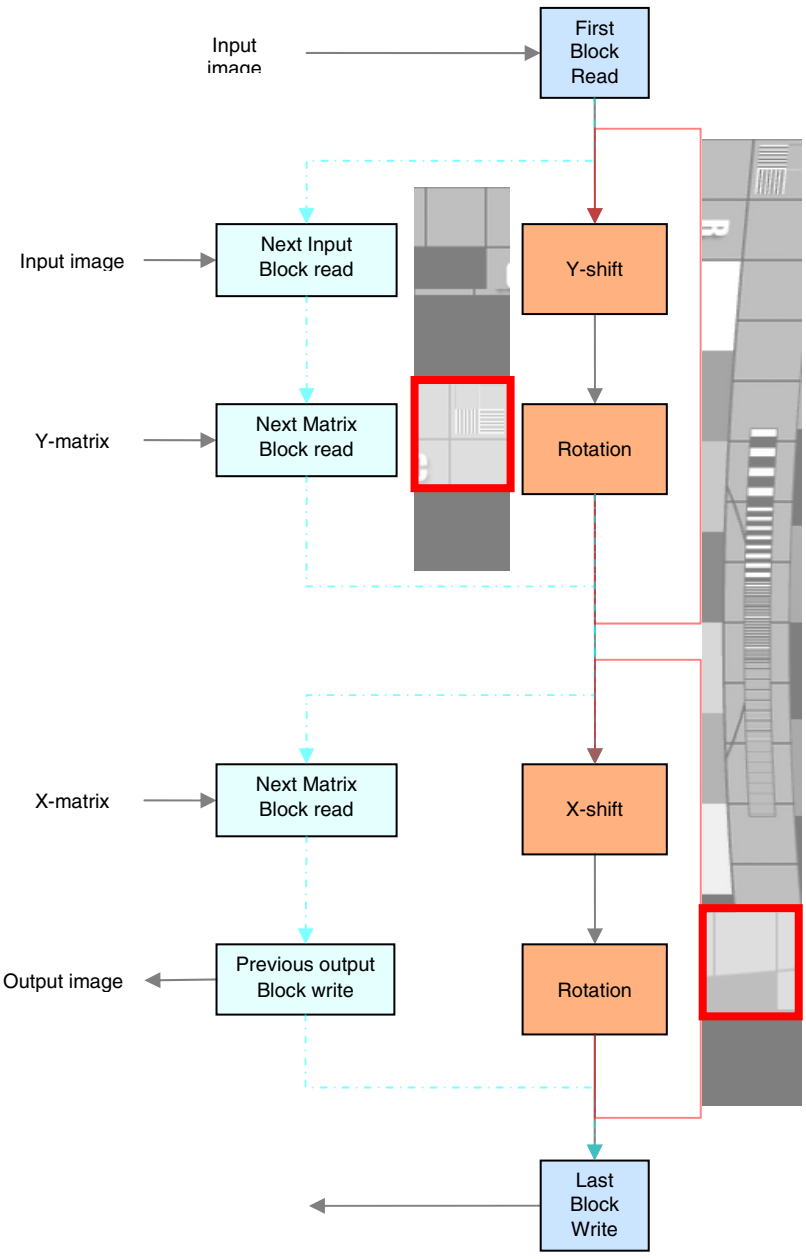
Therefore, we only need **3 internal buffers**:

- One rotating image buffer containing 2 blocks. One blocks is used for processing, the other for background transferring. It serves as **Y-shift input and X-shift output at the same time** so it is INPUT_SIZE*2 high (if we consider INPUT_SIZE to be the maximum size of the different input block sizes to read).
- Another image buffer containing an horizontal stripe, used as **Y-shift output and X-shift input**. Its size is therefore PENO*WIDTH (if we consider WIDTH as the image width).
- A rotating matrix buffer containing two square blocks PENO-wide. One block is used for processing, the other for background transferring.

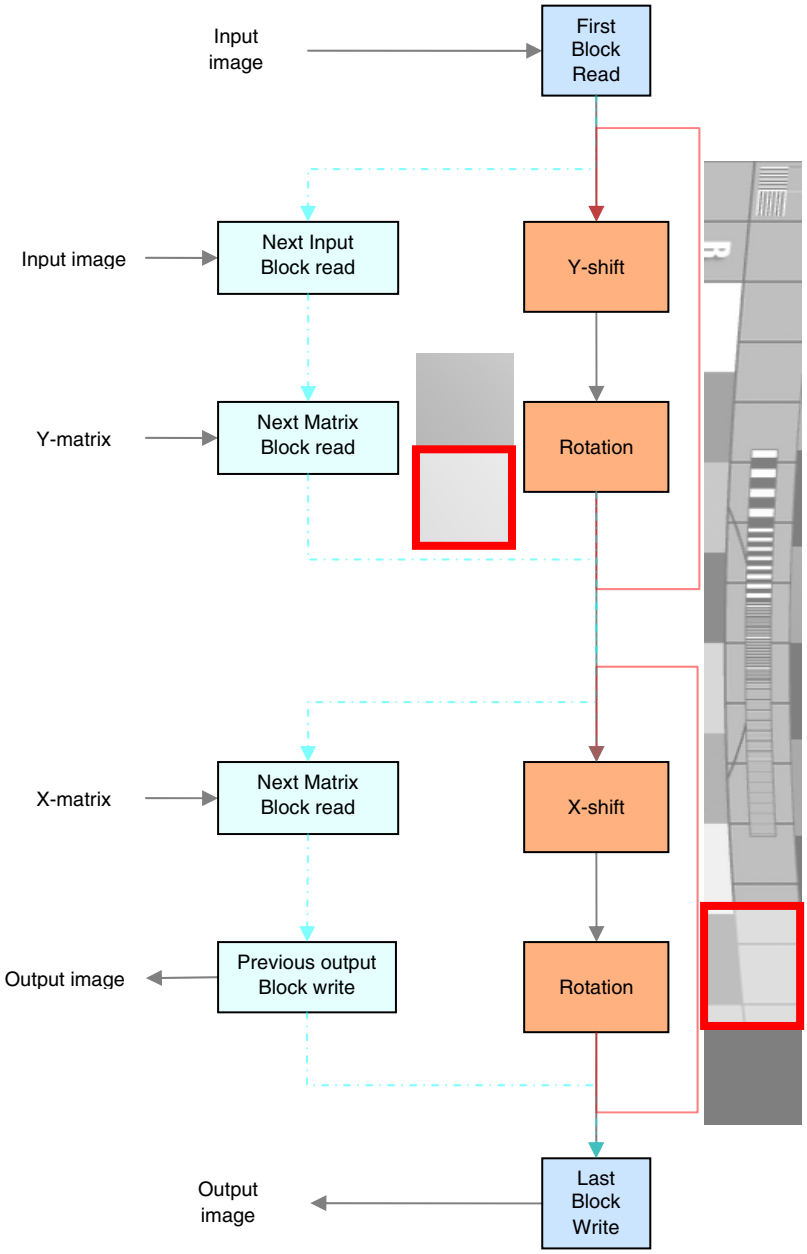
3.1.1.2 Step-by-step execution

To illustrate the different steps in details, here are the resulting images for each block

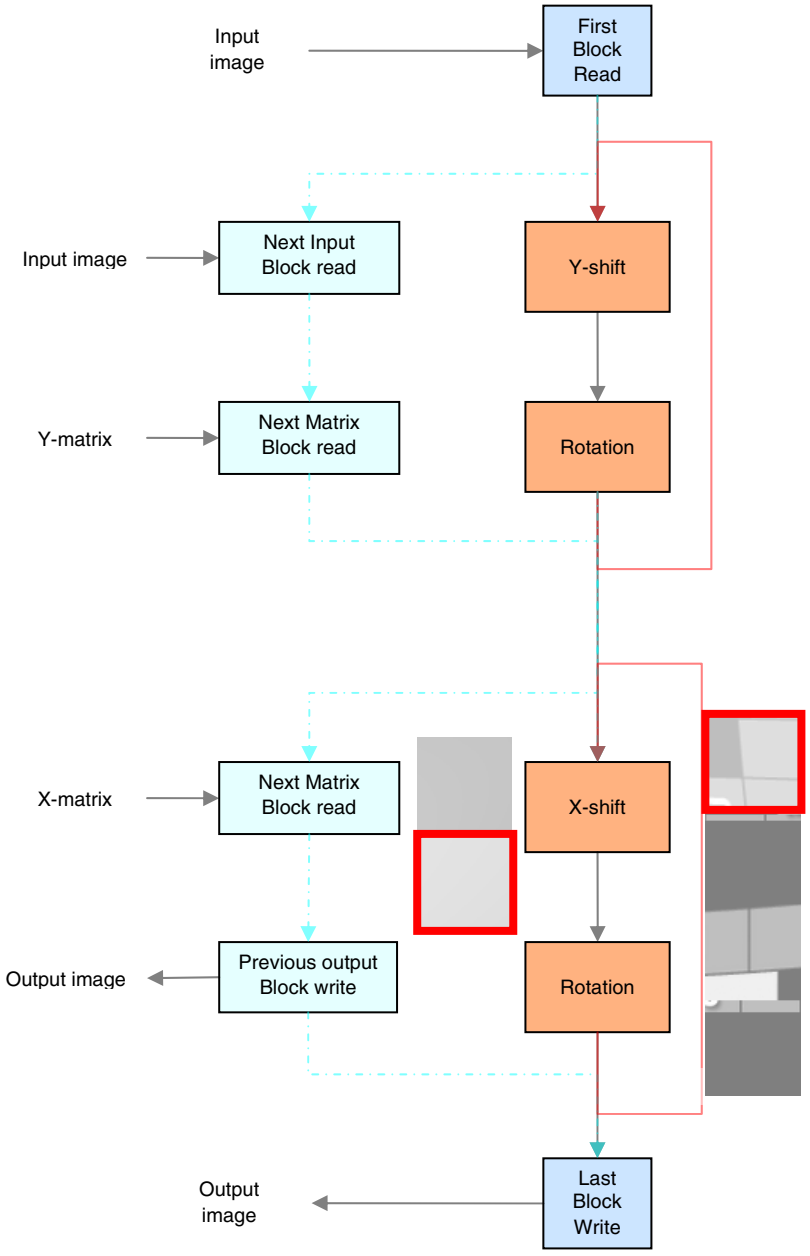
Y-shift and next input read



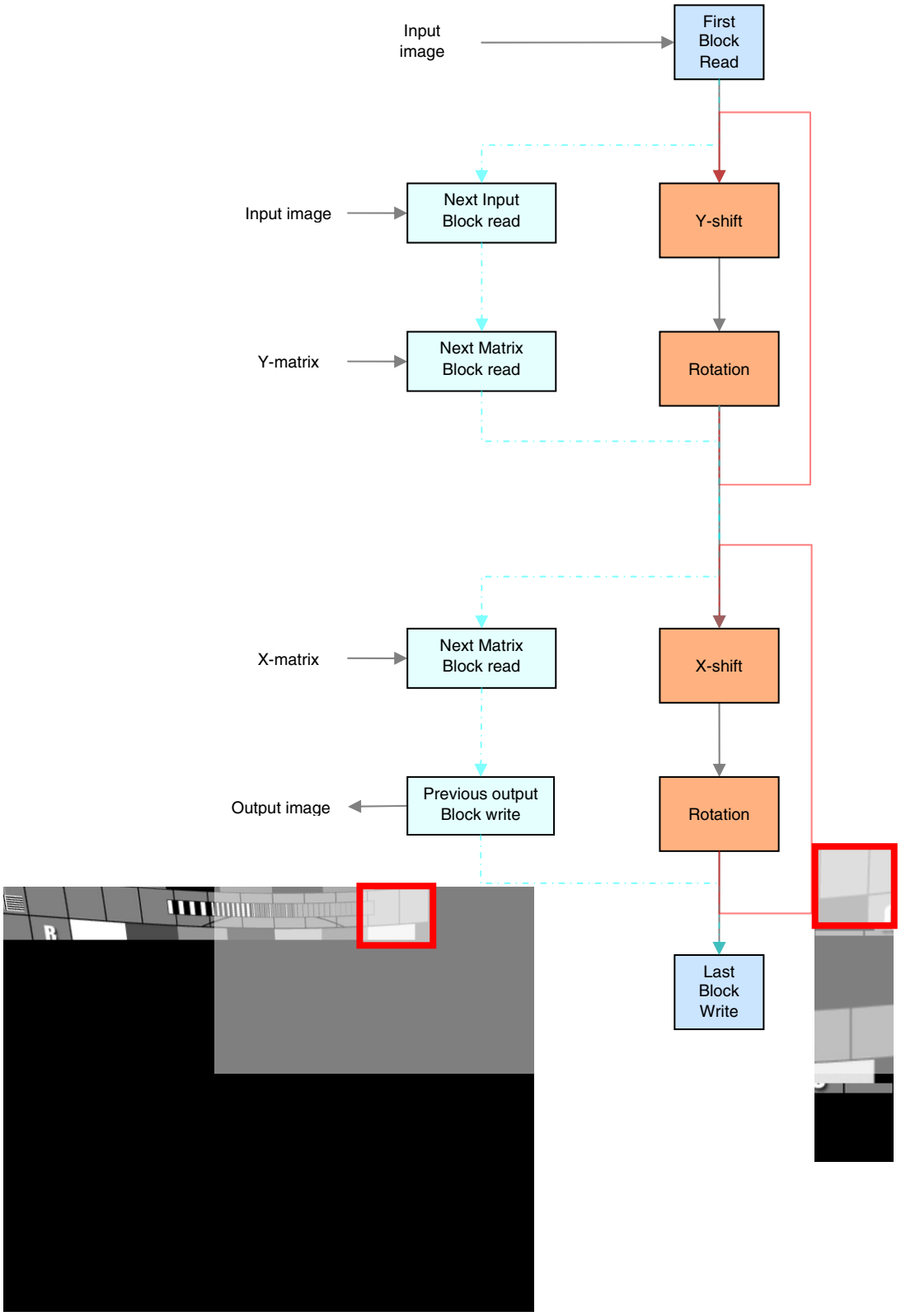
1st rotation and next Y-matrix read



X-shift and next X-matrix read



2nd rotation and previous output write



3.1.2 Using C-ring

The two rotations are slowing down the overall process. In this implementation, the horizontal shift is done per lines in the horizontal direction.

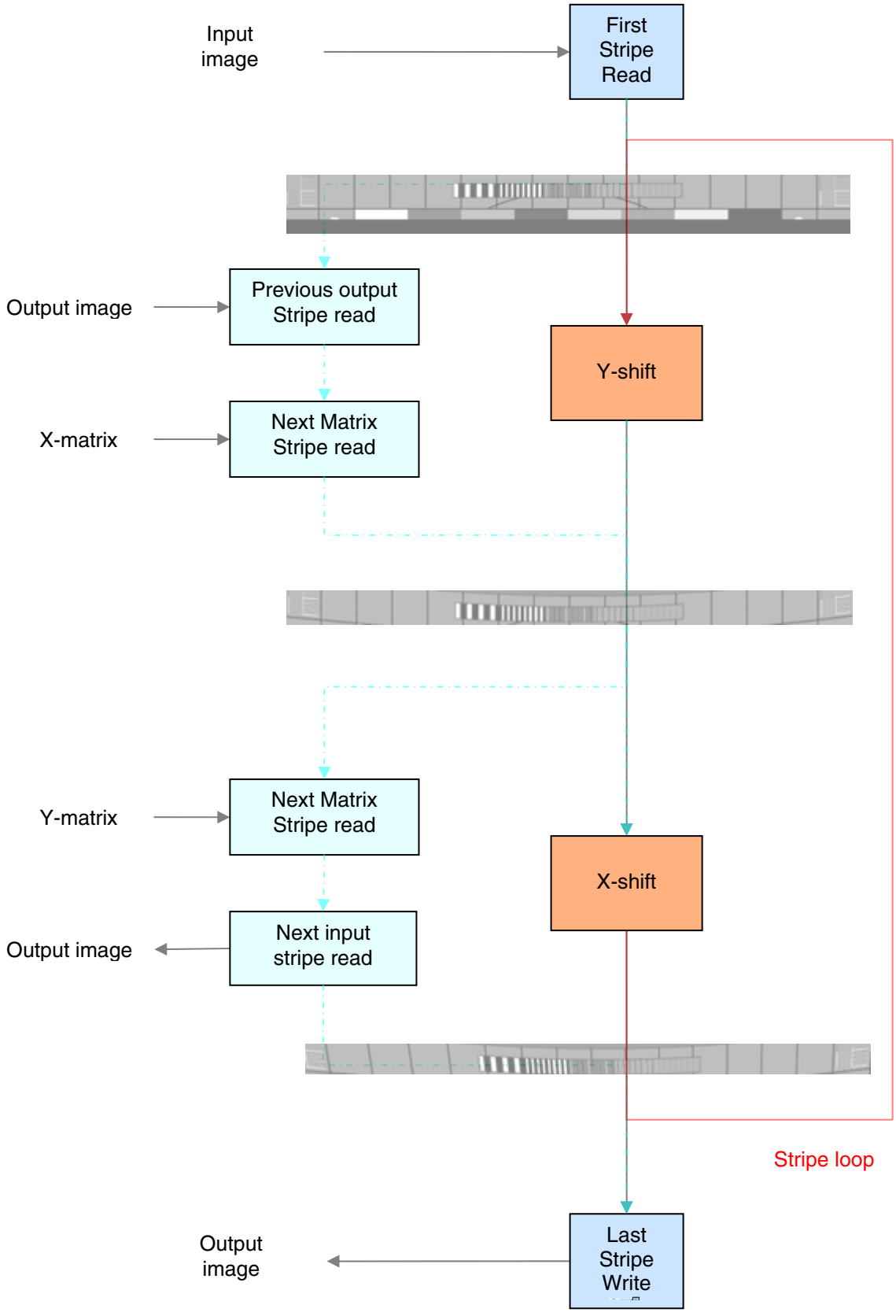
3.1.2.1 General diagram

Preliminary remarks:

- As in the previous concept, vertical shift is done first and then horizontal shift is processed
- Vertical shift is the same as the previous concept but horizontal shift is done using the C-ring
- The image is processed per horizontal stripes. A horizontal stripe is a group of STRIPE_SIZE lines of the image.
- Memory transfers are done in background as much as possible.
- As in the previous concept, input blocks have different sizes depending on Y-shift values. These sizes are computed offline.

Therefore, **5 internal buffers** are needed

- One image buffer WIDTH*INPUT_SIZE for input image read.
- One image buffer WIDTH*STRIPE_SIZE for X-matrix read
- One image buffer WIDTH*STRIPE_SIZE for Y-matrix read
- One image buffer WIDTH*STRIPE_SIZE for Y-shift output
- One image buffer WIDTH*STRIPE_SIZE for X-shift output


Input image
First Stripe Read
Output image
Previous output Stripe read
Y-shift
X-matrix
Next Matrix Stripe read
Y-matrix
Next Matrix Stripe read
X-shift
Output image
Next input stripe read
Stripe loop
Output image
Last Stripe Write

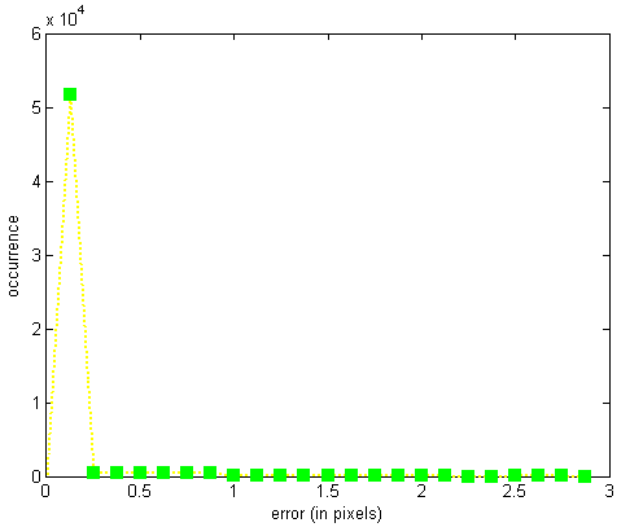
3.1.3 Using C-ring (speed-up version)

In the horizontal shift of the previous version, the pixels (x,y) and (x+1,y) are transferred using the C-ring for each pixel (x,y) in order to perform the bilinear interpolation correctly.

In this version, we transfer the pixel (x,y) and (x,y+1) to perform the horizontal shift faster. Therefore, for each pixel and each line y, the bilinear interpolation is done with the pixels x+shift(x) and x+1+shift(x+1). This can lead to error if the integer parts of shift(x) and shift(x+1) are different so the X-shift matrix need to be corrected offline.

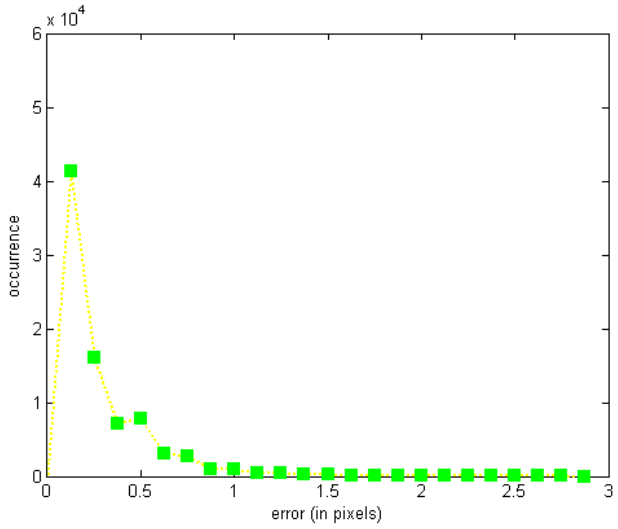
To allow the double transfer in the Y-direction, the X-shift matrix need to be recoded so that every two lines y, the integer parts of shift(y) and shift(y+1) are the same. Integer part is chosen so that error is minimized. Below is an example of the evolution of the error distribution on X coordinate (Inverse Perspective Matching algorithm):

Error on previous concept



Errors inferior to 0.5pixel are due to quantization.

Error on speed-up concept



Errors superior to 0.5 pixel are due to vertical recoding but number of inaccurate pixels is negligible for display applications (<1% of the whole image)

General diagram is the same as previously.

3.1.3.1 Matlab recoding

Horizontal correction

```
function [corrected_matrix,
          original_destination,
          destination_integer_part,
          corrected_destination] = recode_matrix_X_correction(original_matrix,height,width)
% This function corrects error done on bilinear interpolation
% when performing horizontal shift using C-ring optimized

original_destination = zeros(height,width);
destination_integer_part = zeros(height,width);
corrected_matrix=zeros(height,width);
corrected_destination=zeros(height,width);

for v=1:height
   for u=1:(width)
      % Get destination matrix
      original_destination(v,u) = original_matrix(v,u) + u;
      destination_integer_part(v,u) = floor(original_destination(v,u));
   end
end

% Recode matrix considering new bilinear interpolation
 for v = 1:height
   for u = 1:(width-1)
     % correction needed if integer part of the two neighboor pixels differ
     if(abs(destination_integer_part(v,u)-destination_integer_part(v,u+1))>0)
        new_decimal = (original_destination(v,u) - destination_integer_part(v,u))
          /(destination_integer_part(v,u+1)-destination_integer_part(v,u));
        corrected_destination(v,u) = destination_integer_part(v,u) + new_decimal;
        corrected_matrix(v,u) = corrected_destination(v,u) - u;
     else % no correction needed
        corrected_destination(v,u) = original_destination(v,u);
        corrected_matrix(v,u) = original_matrix(v,u);
     end
   end
 end
```

Vertical coding

```
function [Mx,Dest,Dest_int] = recode_matrix_Y_correction(Mx,height,width)
% Recodes matrix to enable transfer of (x,y) and (x,y+1) at the same time.
% that means for each pixel (x,y),
% integer_part(shift(x,y)) = integer_part(shift(x,y+1))

Dest = zeros(height,width);
Dest_int = zeros(height,width);

% Recoding
for v=1:2:height-1
   for u=1:width

      y = floor(Mx(v,u)); % integer part
      y1 = floor(Mx(v+1,u)); % integer part

      if(y<y1)
         Mx(v,u) = y1;
      elseif(y1<y)
         Mx(v+1,u) = y;
      end

   end
end

% compute new destination
for v=1:height
   for u=1:(width)
      Dest(v,u) = Mx(v,u) + u;
      Dest_int(v,u) = floor(Dest(v,u));
   end
end
```

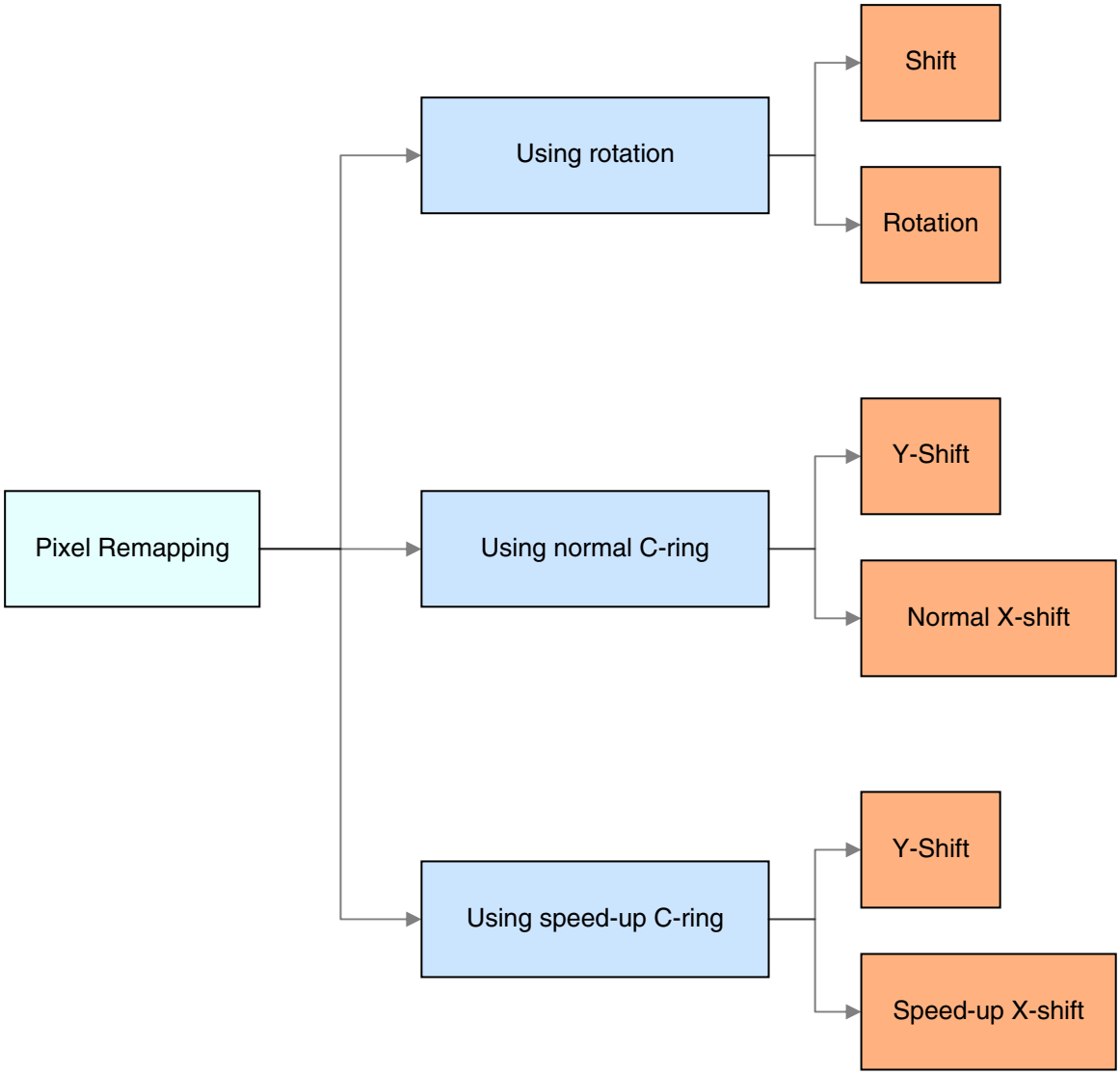
“Main” function

```
% Read input matrixes
original_matrix = imread('X_matrix_fisheye.pgm','PGM');

% Get real values
original_real_matrix = (original_matrix - 128) /4;

 % Horizontal bilinear interpolation correction
[corrected_matrix, original_destination, original_destination_int, corrected_destination] =
recode_matrix_X_correction(original_real_matrix, 480, 640);

% Vertical coding
[coded_matrix, coded_destination, coded_destination_int] =
recode_matrix_Y_correction(corrected_matrix, 480, 640);

% Rescale
[rescaled_matrix] = rescale(coded_matrix, 4, 128);

% Save
imwrite(uint8(rescaled_matrix),'X_matrix_fisheye_recoded.pgm','PGM','Encoding','ASCII');
```

3.2 Software architecture

3.2.1 Dependence graphs

The graph is simple. The method needs to be defined in the main function thanks to a flag parameter.



The main function is also calling IxThreshold to determine the variable input sizes

All these files are calling the same header file PixelRemappingParameters.h containing all the general parameters and declarations.

3.2.2 Functions and parameters description

3.2.2.1 Header file

We created a general header file PixelRemappingParameters.h for all files.

It contains:

- Method flags:

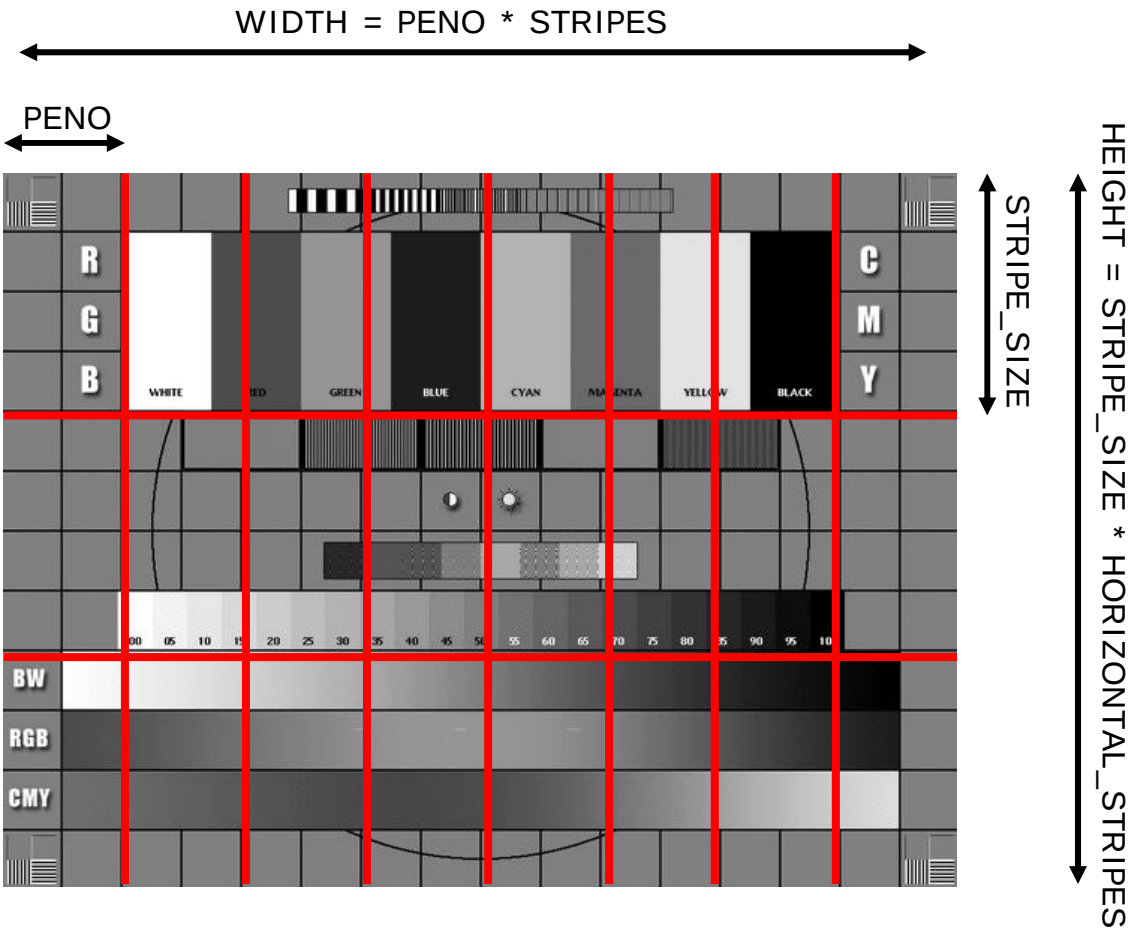
Parameter	Definition
ROTATION	Value is defined to 0
C_RING_NORMAL	Value is defined to 1
C_RING_SPEED_UP	Value is defined to 2

- Image- and matrix-related parameters

Parameter	Definition
PR_MTX	Defines the matrix type
IDX_MTX	For internal line index computation depending on PR_MTX: 4 if 8-bits per pixel, 2 if 16-bits
PR_IMG	Defines the image type
IDX_IMG	For internal line index computation depending on PR_IMG 4 if 8-bits per pixel, 2 if 16-bits
SCALE_FACTOR	Scale factor to compute real shift values
CENTER_FACTOR	Center factor to compute real shift values
SCALE_BITS	Obtained by: 2^SCALE_BITS = SCALE_FACTOR
PR_SHIFT_CAST	Depending on the number of bytes of the image and SCALE_BITS, it defines the cast to apply for executing the bilinear interpolation in the shift functions (should be 16 bits for speed-up execution)

- Buffer sizes (see the following figure):

Parameter	Definition
HEIGHT	Image and matrix height. It must be a multiple of PENO for rotation method as it uses PENO square blocks
REAL_HEIGHT	Real height of image and matrix
WIDTH	Image and matrix width
STRIPES	Number of PENO-wide vertical stripes
*METHOD*_STRIPE_SIZE	Number of lines in a horizontal stripe (must be equal to PENO if rotation method used)
*METHOD*_HORIZONTAL_STRIPES	Number of *METHOD*_STRIPE_SIZE-high horizontal stripes
*METHOD*_INPUT_SIZE	Maximum size of variable input block size



Remark

*METHOD*_INPUT_SIZE should be determined by the returned value of IxThreshold function. ROTATION_STRIPE_SIZE is automatically equal to PENO but C_RING_STRIPE_SIZE should consider that internal memory is limited to 4KB per PE. The sum of the following must be inferior to 4000:

Image input: $\frac{\text{WIDTH}}{\text{PENO}}$*INPUT_SIZE

Matrix buffers: 2*$\frac{\text{WIDTH}}{\text{PENO}}$* STRIPE_SIZE*sizeof(PR_MTX)

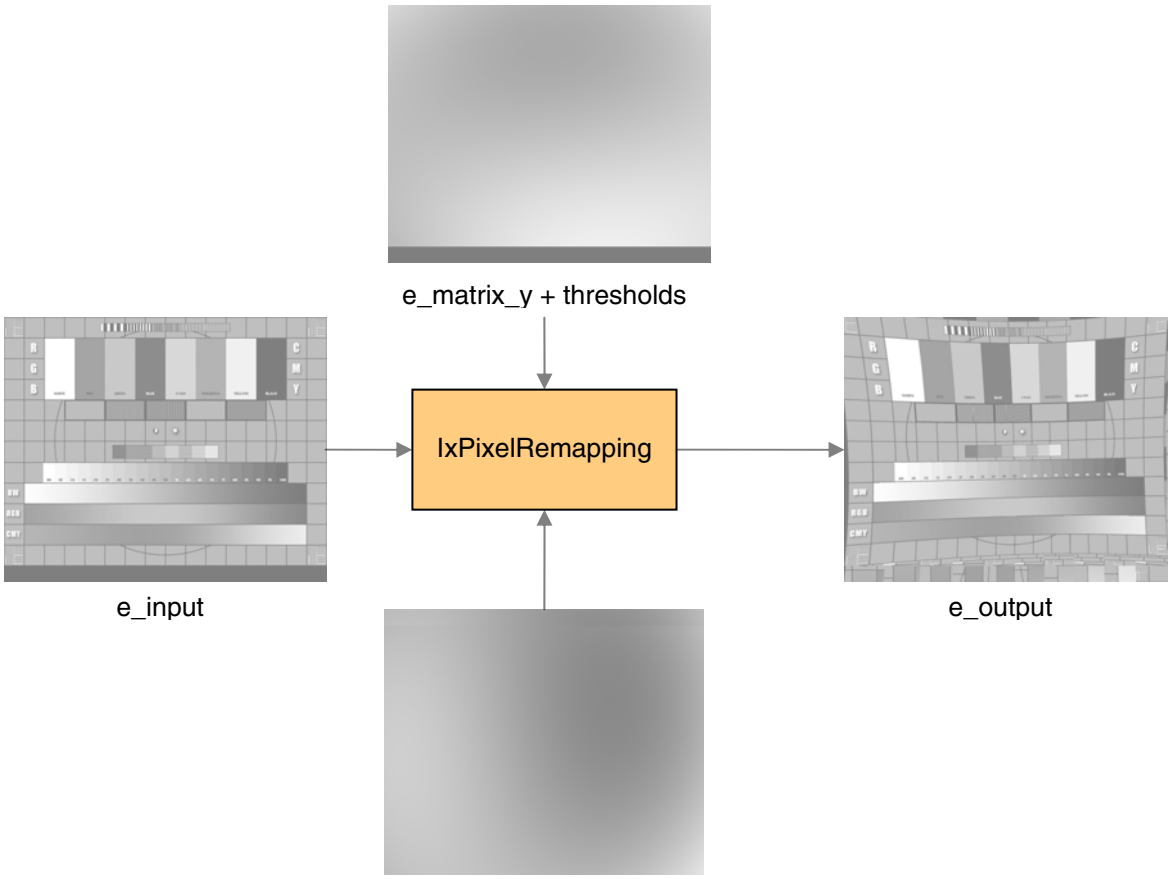
Image output buffers: 2*$\frac{\text{WIDTH}}{\text{PENO}}$*STRIPE_SIZE*sizeof(PR_IMG)

3.2.2.2 Pixel Remapping function

The IxPixelRemapping function contains the general steps of the pixel remapping process: background transfers, shifts and if necessary rotation.

The syntax should be as follows:

```
void IxPixelRemapping(outside PR_IMG * e_input,\
                      outside PR_MTX * e_matrix_y,\
                      outside PR_MTX * e_matrix_x,\
                      outside PR_IMG * e_output,\
                      uint * e_threshold_minc,\
                      uint * e_threshold_maxc,\
                      uint * e_threshold_minr,\
                      uint * e_threshold_maxr,\
                      uint flag);
```



No matter what method is used, Y-shift matrix is the same, only the X-shift matrix changes.

The parameters are the following:

Parameter	Definition	Type	Size
e_input	Image input, stored in external memory	outside PR_IMG *	HEIGHT*WIDTH
e_matrix_y	Vertical shift matrix, stored in external memory	outside PR_MTX *	HEIGHT*WIDTH
e_matrix_x	Horizontal shift matrix, stored in external memory	outside PR_MTX *	HEIGHT*WIDTH
e_output	Image output, stored in external memory	outside PR_IMG *	HEIGHT*WIDTH
e_threshold_minc	Output containing beginning of each input stripe to read for both method using C_RING	uint *	HORIZONTAL_STRIPES
e_threshold_maxc	Output containing beginning of each input stripe to read for both method using C_RING	uint *	HORIZONTAL_STRIPES
e_threshold_minr	Output containing beginning of each input block to read for method ROTATION	uint *	STRIPES*HORIZONTAL_STRIPES
e_threshold_maxr	Output containing beginning of each input block to read for method ROTATION	uint *	STRIPES*HORIZONTAL_STRIPES
flag	Method parameter: should be equal to C_RING_NORMAL, ROTATION or C_RING_SPEED_UP	uint	1

This function works with images up to 12 bytes per pixel and matrixes up to 16 bytes per pixel. **Do not forget to change the corresponding PR_IMG and PR_MTX parameters in the header file *PixelRemappingParameters.h***

3.2.2.3 Shift function when using rotation method

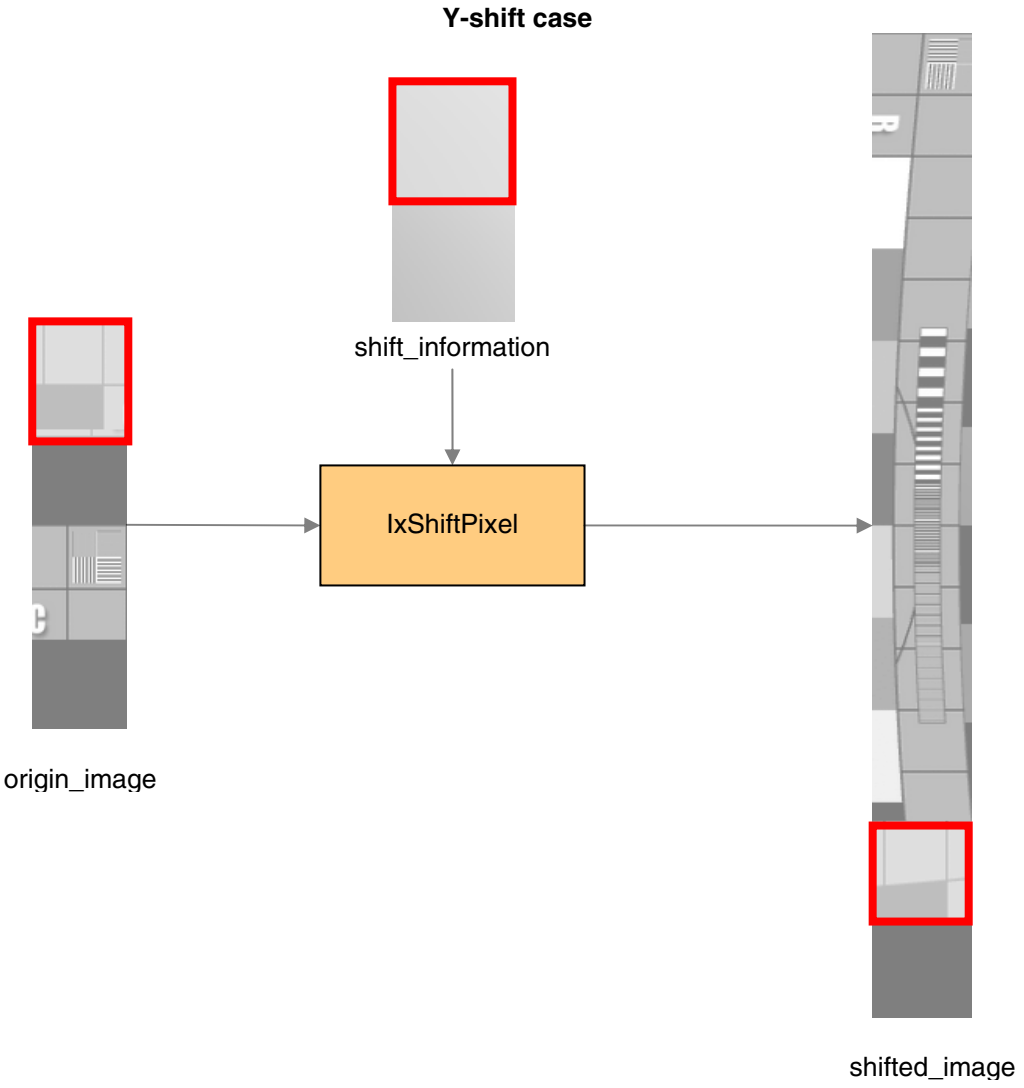
The IxShiftRotation function is used to shift pixels vertically per blocks of PENO*PENO pixels. That is why we use the combination of rotation and this function for the horizontal shift.

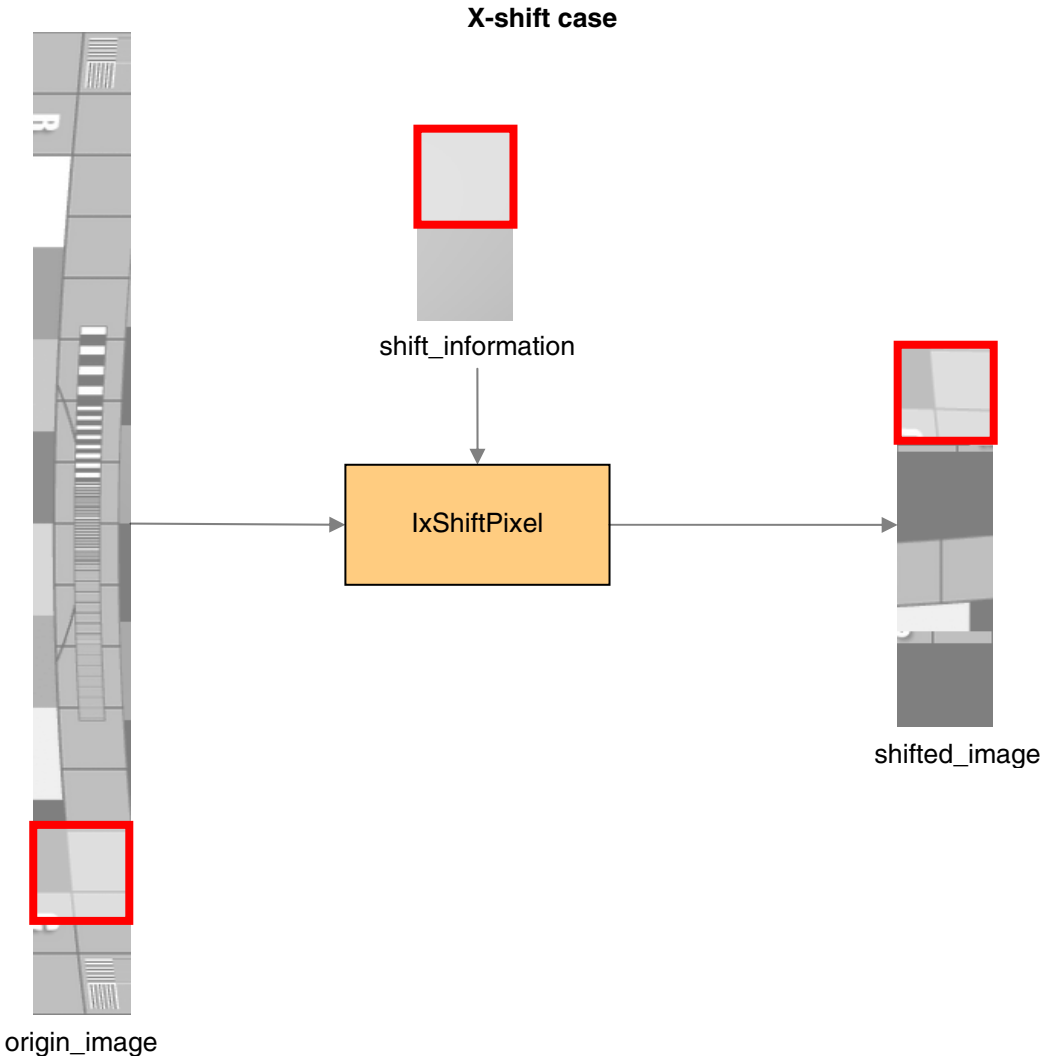
Note that it shifts the pixels from the destination, that means:

$$out(x,y) = in(x,y+k)$$

The syntax should be as follows:

*void IxShiftRotation(PR_IMG * origin_image,*
*PR_MTX * shift_information,*
*PR_IMG * shifted_image,*
uint lines)





The parameters are the following:

Parameter	Definition	Type	Size
origin_image	Input block	PR_IMG *	2*ROTATION_INPUT_SIZE high for Y-shift or WIDTH high for X-shift (see paragraph 3.1)
shift_information	Shift block	PR_MTX *	2*PENO high
shifted_image	Output block	PR_IMG *	2*ROTATION_INPUT_SIZE high for X-shift or WIDTH high for Y-shift or (see paragraph 3.1)
lines	Number of lines to process	uint	1

This function works with images up to 12 bytes per pixel and matrixes up to 16 bytes per pixel. **Do not forget to change the corresponding PR_IMG and PR_MTX parameters in the header file *PixelRemappingParameters.h***

3.2.2.4 Shift functions when using C-ring method

All the shift functions are used to shift pixels per horizontal stripes of WIDTH*C_RING_STRIPE_SIZE pixels.

Note that it shifts the pixels from the destination, that means:

$$out(x,y) = in(x,y+k)$$

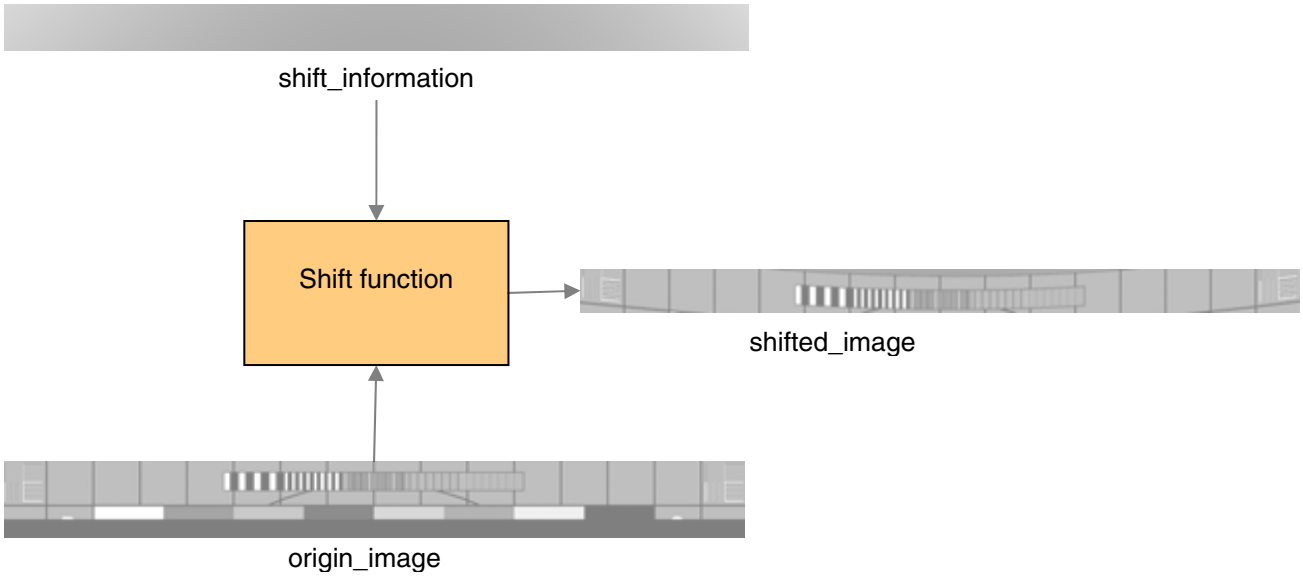
The generic syntax is as follows:

*void IxShiftFunction(PR_IMG * origin_image,*

*PR_MTX * shift_information,*

*PR_IMG * shifted_image)*

Y-shift case



For Y-shift, IxVerticalShiftCRing is used.

For X-shift, IxHorizontalShiftNormal or IxHorizontalShiftSpeedUp is used depending on the flag defined in IxPixelRemapping

The parameters are the following:

Parameter	Definition	Type	Size
origin_image	Input block	PR_IMG *	INPUT_SIZE*WIDTH for Y-shift or C_RING_STRIPE_SIZE*WIDTH for X-shift
shift_information	Shift block	PR_MTX *	C_RING_STRIPE_SIZE*WIDTH
shifted_image	Output block	PR_IMG *	C_RING_STRIPE_SIZE*WIDTH

4 Middleware description

4.1 Project tree

The main folder contains a generic project PixelRemapping with the general library functions and some application examples in the PR_tests folder.

The overall structure looks as follows:

Level 0	Level 1	Level 2	Level 3	Level 4
PixelRemappingMW				
	PixelRemapping			
		Data		
			Mire_VGA.png	
			Y_matrix_rotation.pgm	
			X_matrix_rotation.pgm	
			X_matrix_rotation_rotated.pgm	
			X_matrix_rotation_recoded.pgm	
		Matlab		
			rotation_matrix_generation.m	
			rescale.m	
			rotation_matrix_generation_main.m	
			recode_matrix_Y_correction.m	
			recode_matrix_X_correction.m	
			matrix_recoding_main.m	
		ImapPixelRemapping		
			ipr_Threshold.lc	
			ipr_PixelRemapping.lc	
			ipr_BilinearInterpolation.lc	
		PixelRemappingParameters.h		
		PixelRemappingMain.lc		
		PixelRemapping.tcl		
	PR_tests			
		Fisheye		
			Data	
				Mire_VGA.png
				double_coded_position_X.pgm
				Y_matrix_fisheye.pgm
				X_matrix_fisheye.pgm
				X_matrix_fisheye_rotated.pgm
				X_matrix_fisheye_recoded.pgm
				X_error_speed_fisheye.pgm
			Matlab	
				rescale.m
				recode_matrix_Y_correction.m
				recode_matrix_X_correction.m
				matrix_recoding_main.m
				double_coded_matrix.m
				reference_generation.m
				error_histogram.m
				reference_comparison_main.m

			PixelRemappingParameters.h	
			PixelRemappingMain.lc	
			PixelRemapping.tcl	
		IPM		
			Data	
				Mire_VGA.png
				double_coded_position_X.pgm
				Y_matrix_ipm.pgm
				X_matrix_ipm.pgm
				X_matrix_ipm_rotated.pgm
				X_matrix_ipm_recoded.pgm
				X_error_speed_ipm.pgm
			Matlab	
				rescale.m
				recode_matrix_Y_correction.m
				recode_matrix_X_correction.m
				matrix_recoding_main.m
				double_coded_matrix.m
				reference_generation.m
				error_histogram.m
				reference_comparison_main.m
			PixelRemappingParameters.h	
			PixelRemappingMain.lc	
			PixelRemapping.tcl	
		Reduction		
			Data	
				Mire_VGA.png
				double_coded_position_X.pgm
				Y_matrix_reduction.pgm
				X_matrix_reduction.pgm
				Y_matrix_reduction_rotated.pgm
				X_matrix_reduction_recoded.pgm
				X_error_speed_reduction.pgm
			Matlab	
				rotation_matrix_generation.m
				rescale.m
				rotation_matrix_generation_main.m
				recode_matrix_Y_correction.m
				recode_matrix_X_correction.m
				matrix_recoding_main.m
				double_coded_matrix.m
				reference_generation.m
				error_histogram.m
				reference_comparison_main.m
			PixelRemappingParameters.h	
			PixelRemappingMain.lc	
			PixelRemapping.tcl	
		Rotation		
			Data	
				Mire_VGA.png
				double_coded_position_X.pgm
				Y_matrix_rotation.pgm
				X_matrix_rotation.pgm
				X_matrix_rotation_rotated.pgm
				X_matrix_rotation_recoded.pgm
				X_error_speed_rotation.pgm

			Matlab	
				rotation_matrix_generation.m
				rescale.m
				rotation_matrix_generation_main.m
				recode_matrix_Y_correction.m
				recode_matrix_X_correction.m
				matrix_recoding_main.m
				double_coded_matrix.m
				reference_generation.m
				error_histogram.m
				reference_comparison_main.m
			PixelRemappingParameters.h	
			PixelRemappingMain.lc	
			PixelRemapping.tcl	
		Translation		
			Data	
				Mire_VGA.png
				double_coded_position_X.pgm
				Y_matrix_translation.pgm
				X_matrix_translation.pgm
				X_matrix_translation_rotated.pgm
				X_matrix_translation_recoded.pgm
				X_error_ speed_translation.pgm
			Matlab	
				translation_matrix_generation.m
				rescale.m
				translation_matrix_generation_main.m
				recode_matrix_Y_correction.m
				recode_matrix_X_correction.m
				matrix_recoding_main.m
				double_coded_matrix.m
				reference_generation.m
				error_histogram.m
				reference_comparison_main.m
			PixelRemappingParameters.h	
			PixelRemappingMain.lc	
			PixelRemapping.tcl	

Remarks

In the Matlab folder:

- xxx_matrix_generation_main is used to generate shift matrixes for translation, rotation or reduction/zoom and calls xxx_matrix_generation and rescale functions
- matrix_recoding_main is used to recode X-shift matrix if necessary and calls recode_matrix_X_correction as well as recode_matrix_Y_correction
- reference_comparison_main is used to generate a MATLAB reference and compute error distribution on the X coordinate. This calls reference_generation and error_histogram

ImapPixelRemapping contains functions library and is linked to each application example project:

- ipr_Threshold for offline thresholds computing functions
- ipr_PixelRemapping for general pixel remapping process
- ipr_BilinearInterpolation for all shift functions

4.2 Memory consumption

4.2.1 Rotation method

The rotation method uses 3 internal buffers:

Buffer	Use	Type	Size
Image Buffer 1	Input before vertical shift as well as output after horizontal shift	PR_IMG *	2*INPUT_SIZE*PENO
Image Buffer 2	Output after vertical shift, as well as input before horizontal shift	PR_IMG *	STRIPES*PENO*PENO
Matrix Buffer	Contains shift blocks	PR_MTX *	2*PENO*PENO

Therefore, the maximum internal memory consumption is 1.7KB per PE (case of 64PEs, 1280*1024 resolution)

4.2.2 C-ring method

The C-ring method uses 5 internal buffers:

Buffer	Use	Type	Size
Image Buffer 1	Input before vertical shift	PR_IMG *	INPUT_SIZE*WIDTH
Image Buffer 2	Output after vertical shift, as well as input before horizontal shift	PR_IMG *	C_RING_STRIPE_SIZE*WIDTH
Image Buffer 3	Output after horizontal shift	PR_IMG *	C_RING_STRIPE_SIZE*WIDTH
Matrix Buffer 1	Contains X-shift input	PR_MTX *	C_RING_STRIPE_SIZE*WIDTH
Matrix Buffer 2	Contains Y-shift input	PR_MTX *	C_RING_STRIPE_SIZE*WIDTH

Therefore, memory consumption is adjustable by defining INPUT_SIZE and C_RING_STRIPE_SIZE properly.

4.3 Execution time

For the overall process we can compare the time performance depending on:

- the high-end and low-end version of IMAPCAR2
- the different type of images and matrixes (size considered is VGA)
- the method used

8 and 16 bits images were tested. For 8 bits matrix, fisheye correction matrixes were chosen and for 16 bits matrixes we used inverse perspective matching matrixes.

- For IMAPCAR2-200 (64PE, 132Mhz), we obtain:

Concept	**8-bits image and matrixes e.g Fisheye**	**16-bits image and matrixes e.g. Inverse Perspective Matching**
Using rotation	2.97ms	3.11ms
Using C-ring (normal version)	3.46ms	3.9ms
Using C-ring (speed-up version)	2.3ms	2.65ms

- For IMAPCAR2-50 (32PE, 66Mhz), we obtain:

Concept	**8-bits image and matrixes e.g. Fisheye**	**16-bits image and matrixes e.g. Inverse Perspective Matching**
Using rotation	14.22ms	15.02ms
Using C-ring (normal version)	13.59ms	15.32ms
Using C-ring (speed-up version)	8.94ms	10.48ms

Therefore, if distortion is not important, using the speed-up C-ring version is much faster. Else, we can use rotation or normal C-ring method to get precise results.

Remarks

In this document, only gray-value images up to 16 bits were considered. To use this algorithm with color images, the 3 color coordinates can be coded (using YC422 for example). Therefore, time performance is the same for pixel remapping, only coding and decoding time is added.

In case of several cameras configuration, camera and image parameters should be the same so the same process can be applied to all.

4.4 Source codes

4.4.1 Header file

All the parameters are listed in the header file PixelRemappingParameters.h

```
#ifndef PIXELREMAPPINGPARAMETERS_H_
#define PIXELREMAPPINGPARAMETERS_H_

#include <stddef.h>
#include <stdio.h>
#include <imapio.h>
#include <stdimap.h>
#include <libipl.h>

/************************************************************
* Choose method
************************************************************/
#define ROTATION              0
#define C_RING_NORMAL         1
#define C_RING_SPEED_UP       2

/************************************************************
* Matrix parameters
************************************************************/
#define PR_MTX                sep uchar
#define SCALE_FACTOR          4
#define CENTER_FACTOR         128
#define SCALE_BITS            2
#define IDX_MTX               4 // 4 if 8 bits, 2 if 16 bits

/************************************************************
* Image parameters
************************************************************/
#define PR_IMG                sep uint
#define PR_SHIFT_CAST         sep ulong // 16bits if (image bytes+scaling bytes<16), else
32bits
#define IDX_IMG               2 // 4 if 8 bits, 2 if 16 bits
```

```
/*************************************************************
* Sizes
*************************************************************/
#define HEIGHT                512L ///< Image and matrix height (multiple of PENO)
#define REAL_HEIGHT           480L ///< Image and matrix real height (not multiple of PENO)
#define WIDTH                 640L ///< Image and matrix width
#define STRIPES               (WIDTH/PENO) ///< number of blocks in a horizontal stripe

/*************************************************************
* Rotation method parameters
*************************************************************/
#define ROTATION_STRIPE_SIZE  PENO ///< size of block to adjust in case of 16-bits images
#define ROTATION_HORIZONTAL_STRIPES   (HEIGHT/ROTATION_STRIPE_SIZE)
#define ROTATION_INPUT_SIZE   (2*PENO) ///< to adjust in case of 16-bits images

/*************************************************************
* C-Ring method parameters
*************************************************************/
#define C_RING_STRIPE_SIZE    20 ///< size of block to adjust in case of 16-bits images
#define C_RING_HORIZONTAL_STRIPES (REAL_HEIGHT/C_RING_STRIPE_SIZE)
#define C_RING_INPUT_SIZE     (120) ///< to adjust in case of 16-bits images

/*************************************************************
* General flags
*************************************************************/
#if PENO == 64
       #define SCALE_PENO     6 // in case of 64 PE: 6
#elif PENO == 32
       #define SCALE_PENO     5
#endif

#define CLOCKWISE             1
#define ANTI_CLOCKWISE        0
#define READ                  0
#define WRITE                 1
#define FG                    1 ///< foreground transfer
#define BG                    0 ///< backgroung transfer

#endif /*PIXELREMAPPINGPARAMETERS_H_*/
```

4.4.2 Main file

```
#include "PixelRemappingParameters.h"

/* global variables */
outside PR_IMG e_input[STRIPES*HEIGHT]; // Image input
outside PR_MTX e_shift_y[STRIPES*HEIGHT]; // Vertical matrix
outside PR_MTX e_shift_x[STRIPES*HEIGHT]; // Horizontal matrix
outside PR_IMG e_output[STRIPES*HEIGHT]; // Image output

void main(){

        uint e_thres_minc[C_RING_HORIZONTAL_STRIPES];
        uint e_thres_maxc[C_RING_HORIZONTAL_STRIPES];

        uint e_thres_minr[ROTATION_HORIZONTAL_STRIPES*STRIPES];
        uint e_thres_maxr[ROTATION_HORIZONTAL_STRIPES*STRIPES];

        int max_input_size;

        max_input_size=IxThreshold(e_shift_y,
                                   e_thres_maxc,
                                   e_thres_minc,
                                   e_thres_maxr,
                                   e_thres_minr,
                                   C_RING_NORMAL);

        printf("min INPUT_SIZE parameter: %d",max_input_size);

        IxPixelRemapping(e_input,
                         e_shift_y,
                         e_shift_x,
                         e_output,
                         e_thres_minc,
                         e_thres_maxc,
                         e_thres_minr,
                         e_thres_maxr,
                         C_RING_NORMAL);

}
```

5 Configuration examples (How To)

5.1 Using rotation

This is the most simple to configure:

1. Set the flag in the IxThreshold and IxPixelRemapping functions to ROTATION in the main function

```
max_input_size=IxThreshold(e_shift_y, e_thres_maxc, e_thres_minc,
                           e_thres_maxr, e_thres_minr, ROTATION);


IxPixelRemapping(e_input, e_shift_y,
                 e_shift_x, e_output,
                 e_thres_minc, e_thres_maxc,
                 e_thres_minr, e_thres_maxr, ROTATION);
```

2. All the image and matrix parameters should be carefully defined, especially:

- update IDX_IMG (resp. IDX_MTX) to 4 if image (resp. matrix) is 8 bits per pixel, or 2 if 16 bits per pixel.
- PR_SHIFT_CAST depends on the number of bytes per pixel of the images and the number of scaling bits of the matrixes. It is more time consuming if set to *sep ulong*

Therefore, **image bits + scale bits should be inferior to 16 bits** if possible

```
/*************************************************************
* Matrix parameters
*************************************************************/
#define PR_MTX                sep uchar
#define SCALE_FACTOR          4
#define CENTER_FACTOR         140
#define SCALE_BITS            2
#define IDX_MTX               4


/*************************************************************
* Image parameters
*************************************************************/
#define PR_IMG                sep uint // image is 12 bits per pixel
#define PR_SHIFT_CAST         sep uint
#define IDX_IMG               2
```

3. ROTATION_INPUT_SIZE parameter is provided by the IxThreshold offline function

```
#define ROTATION_INPUT_SIZE   (100)
```

5.2 Using C-ring

The first steps are the same as rotation.

1. Set the flag in the IxThreshold and IxPixelRemapping functions to C_RING_NORMAL (resp. C_RING_SPEED_UP) in the main function

```
max_input_size=IxThreshold(e_shift_y, e_thres_maxc, e_thres_minc,
                           e_thres_maxr, e_thres_minr, C_RING_NORMAL);


IxPixelRemapping(e_input, e_shift_y,
                 e_shift_x, e_output,
                 e_thres_minc, e_thres_maxc,
                 e_thres_minr, e_thres_maxr, C_RING_NORMAL);
```

2. All the image and matrix parameters should be carefully defined, especially:

- Do not forget to update IDX_IMG (resp. IDX_MTX) to 4 if image (resp. matrix) is 8 bits per pixel, or 2 if 16 bits per pixel.
- PR_SHIFT_CAST depends on the number of bytes per pixel of the images and the number of scaling bits of the matrixes. It is more time consuming if set to *sep ulong*

Therefore, **image bits + scale bits should be inferior to 16 bits** if possible

```
/*************************************************************
* Matrix parameters
*************************************************************/
#define PR_MTX                sep uint
#define SCALE_FACTOR          8
#define CENTER_FACTOR         5120
#define SCALE_BITS            3
#define IDX_MTX               2


/*************************************************************
* Image parameters
*************************************************************/
#define PR_IMG                sep uchar // image is 12 bits per pixel
#define PR_SHIFT_CAST         sep uint
#define IDX_IMG               4
```

3. C_RING_INPUT_SIZE parameter is also provided by the IxThreshold offline function

```
#define C_RING_INPUT_SIZE     80
```

4. C_RING_STRIPE_SIZE parameter should be carefully chosen because of internal memory limitation.

The sum of:

Image input: $\frac{WIDTH}{PENO}$*INPUT_SIZE,

Matrix buffers: 2*$\frac{WIDTH}{PENO}$* STRIPE_SIZE*sizeof(PR_MTX) and

Image output buffers: 2*$\frac{WIDTH}{PENO}$*STRIPE_SIZE*sizeof(PR_IMG)

must be inferior to 4000 (see section 3.2.2)

```
#define C_RING_STRIPE_SIZE    40
```

Here, if considering VGA image and 64 PEs configuration, internal memory consumption is equal to 3.2KB

6 Application examples

The PR_tests folder contains preconfigured examples of pixel remapping.

Each configuration example is structured like the PixelRemapping project described above except for the data folder which contain also the error distribution on X coordinate.

Examples are the following:

- Translation in both X and Y directions
- Reduction or zoom in both X and Y directions
- Rotation
- Fisheye correction
- Inverse Perspective Matching

6.1 Translation configuration

6.1.1 Header file

In this case, an example is given with 8 bits image and matrixes. Pixel Remapping is done using rotation

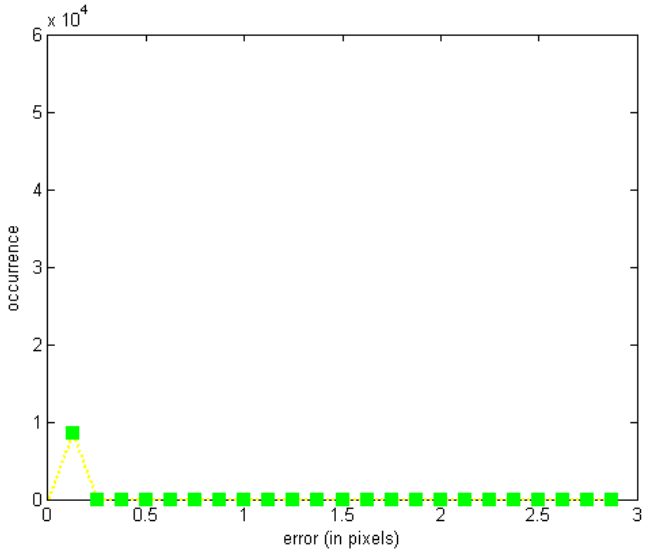
```
/*******************************************************
* Matrix parameters
*******************************************************/
#define PR_MTX          sep uchar
#define SCALE_FACTOR    4
#define CENTER_FACTOR   128
#define SCALE_BITS      2

#define IDX_MTX         4 // 4 if 8 bits, 2 if 16 bits

/*******************************************************
* Image parameters
*******************************************************/
#define PR_IMG          sep uchar
#define PR_SHIFT_CAST   sep uint
#define IDX_IMG         4 // 4 if 8 bits, 2 if 16 bits
```

6.1.2 Results

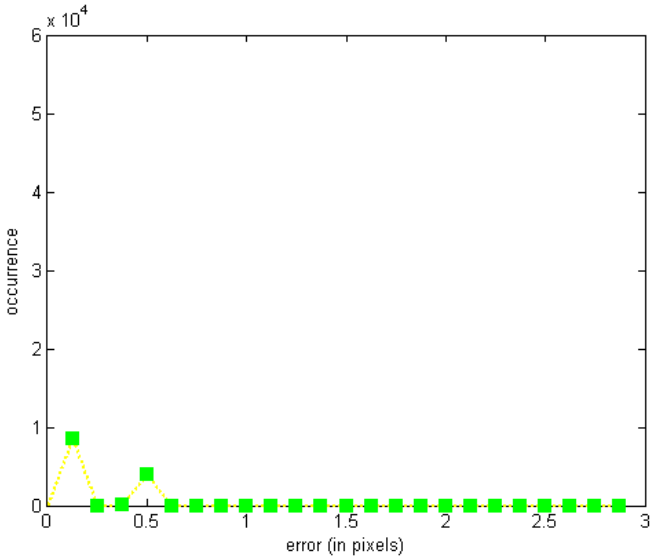
Image example is provided in the corresponding project.

Original image	Image output
	

Error on normal concept



Error on speed-up concept



6.2 Reduction / Zoom

6.2.1 Header file

In this case 8 bits image example is provided and 16 bits matrixes are generated. We used speed-up version of Pixel Remapping.

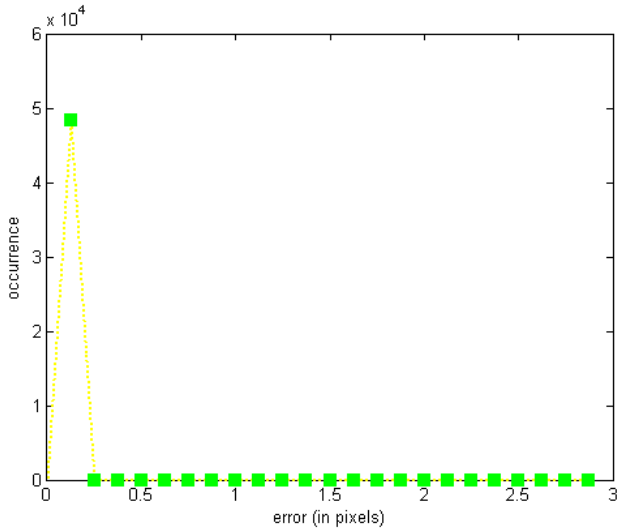
```
/*******************************************************
* Matrix parameters
*******************************************************/
#define PR_MTX          sep uint
#define SCALE_FACTOR    8
#define CENTER_FACTOR   0
#define SCALE_BITS      3

#define IDX_MTX         2 // 4 if 8 bits, 2 if 16 bits

/*******************************************************
* Image parameters
*******************************************************/
#define PR_IMG          sep uchar
#define PR_SHIFT_CAST   sep uint // 16bits if (image bytes+scaling bytes<16), else 32bits
#define IDX_IMG         4 // 4 if 8 bits, 2 if 16 bits

…

/*******************************************************
* C-Ring method parameters
*******************************************************/
#define C_RING_STRIPE_SIZE            16
#define C_RING_HORIZONTAL_STRIPES (REAL_HEIGHT/C_RING_STRIPE_SIZE)
#define C_RING_INPUT_SIZE             (120)
```

6.2.2 Results

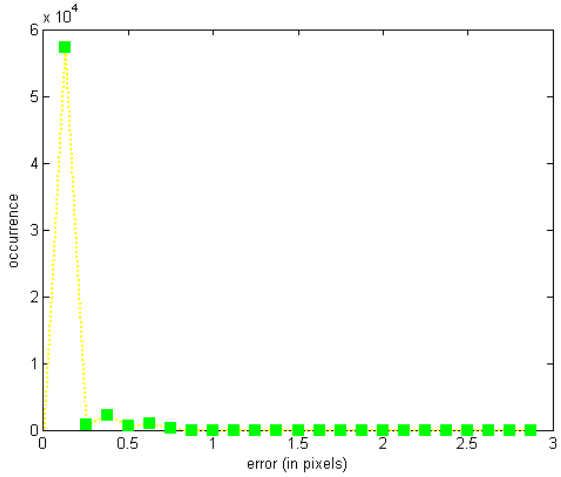
Image example is provided in the corresponding project.

Original image	Image output
	

Error on normal concept



Error on speed-up concept



6.3 Rotation

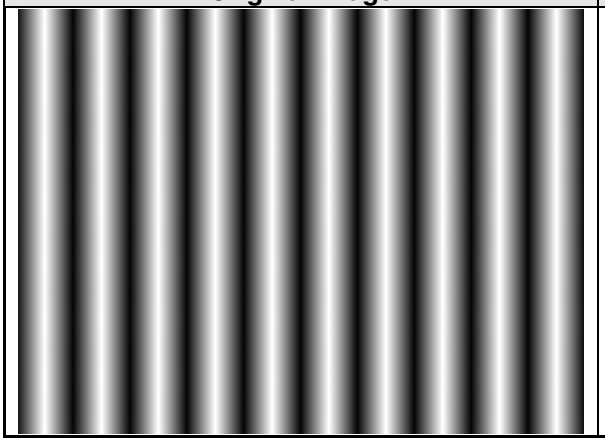
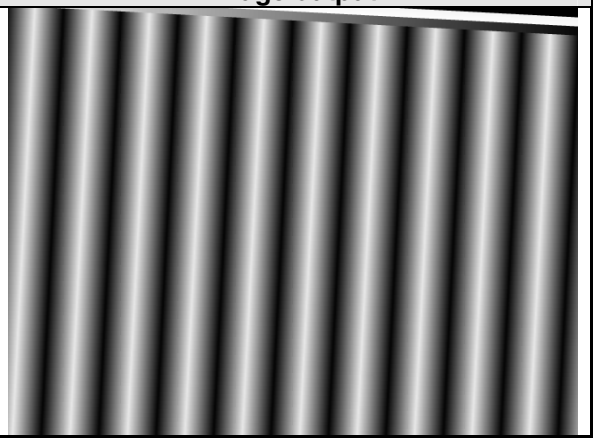
6.3.1 Header file

Here, matrixes are 8 bits per pixel but image example is 16 bits per pixel. We used speed-up method.

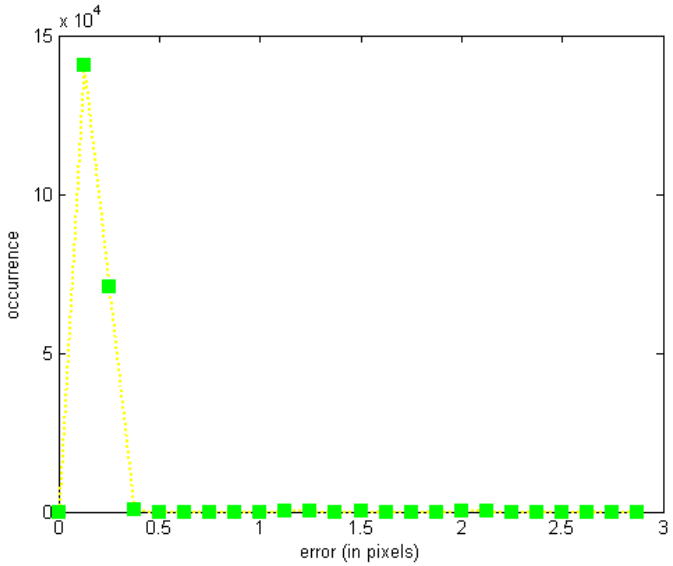
```
/*******************************************************
* Matrix parameters
*******************************************************/
#define PR_MTX          sep uchar
#define SCALE_FACTOR    4
#define CENTER_FACTOR 140
#define SCALE_BITS      2

#define IDX_MTX         4 // 4 if 8 bits, 2 if 16 bits

/*******************************************************
* Image parameters
*******************************************************/
#define PR_IMG          sep uint
#define PR_SHIFT_CAST   sep ulong
#define IDX_IMG         2 // 4 if 8 bits, 2 if 16 bits

…

/*******************************************************
* C-Ring method parameters
*******************************************************/
#define C_RING_STRIPE_SIZE            20
#define C_RING_HORIZONTAL_STRIPES (REAL_HEIGHT/C_RING_STRIPE_SIZE)
#define C_RING_INPUT_SIZE             (80)
```

6.3.2 Results

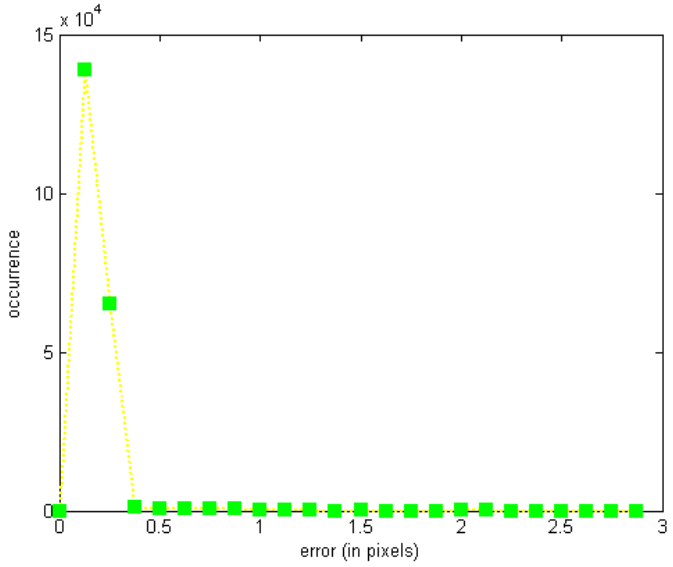
Image example is provided in the corresponding project.

Original image	Image output
	

Error on normal concept



Error on speed-up concept



6.4 Fisheye

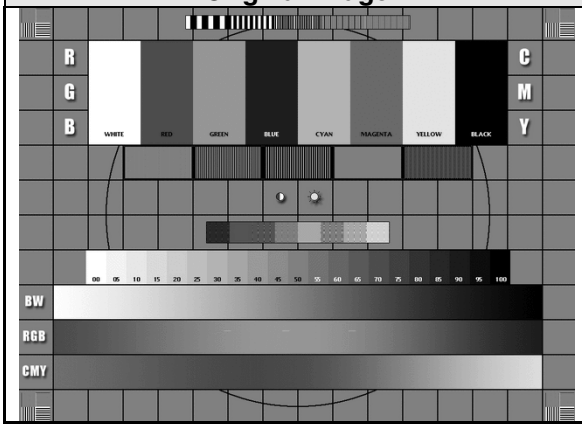
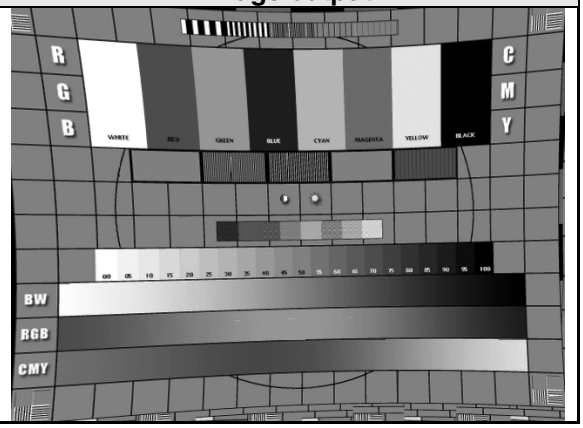
6.4.1 Configuration

In this case, we load 8 bits image and matrixes and rotation method is chosen.

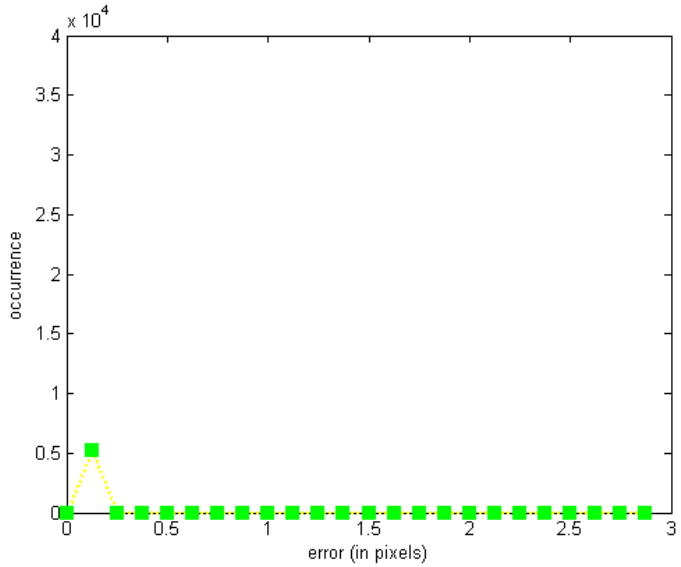
```
/*************************************************************
* Matrix parameters
*************************************************************/
#define PR_MTX         sep uchar
#define SCALE_FACTOR   4
#define CENTER_FACTOR 128
#define SCALE_BITS     2

#define IDX_MTX        4 // 4 if 8 bits, 2 if 16 bits

/*************************************************************
* Image parameters
*************************************************************/
#define PR_IMG         sep uchar
#define PR_SHIFT_CAST  sep uint
#define IDX_IMG        4 // 4 if 8 bits, 2 if 16 bits
```

6.4.2 Results

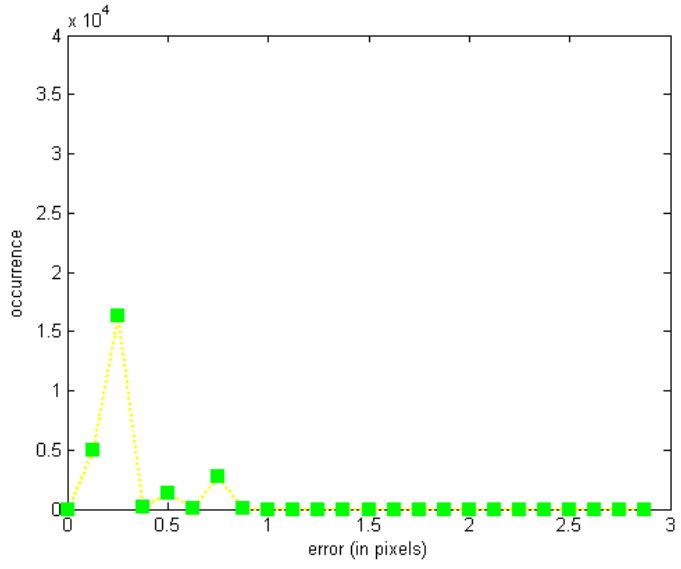
Image example is provided in the corresponding project.

Original image	Image output
	

Error on normal concept



Error on speed-up concept



6.5 Inverse Perspective Matching

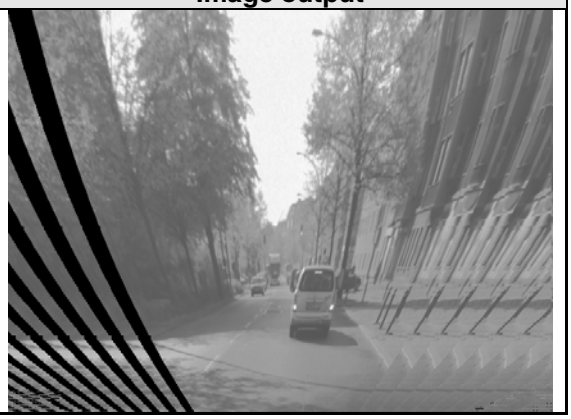
6.5.1 Header file

Here, 16 bits image and matrixes are used. We choose C-ring normal method.

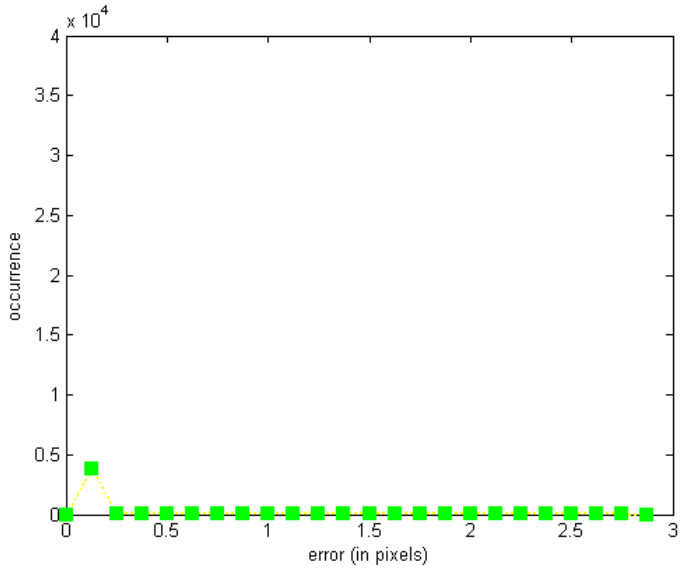
```
/*************************************************************
* Matrix parameters
*************************************************************/
#define PR_MTX         sep uint
#define SCALE_FACTOR   8
#define CENTER_FACTOR  5120
#define SCALE_BITS     3

#define IDX_MTX        2 // 4 if 8 bits, 2 if 16 bits

/*************************************************************
* Image parameters
*************************************************************/
#define PR_IMG         sep uint
#define PR_SHIFT_CAST  sep uint
#define IDX_IMG        2 // 4 if 8 bits, 2 if 16 bits

…

/*************************************************************
* C-Ring method parameters
*************************************************************/
#define C_RING_STRIPE_SIZE           10
#define C_RING_HORIZONTAL_STRIPES    (REAL_HEIGHT/C_RING_STRIPE_SIZE)
#define C_RING_INPUT_SIZE            (80)
```

6.5.2 Results

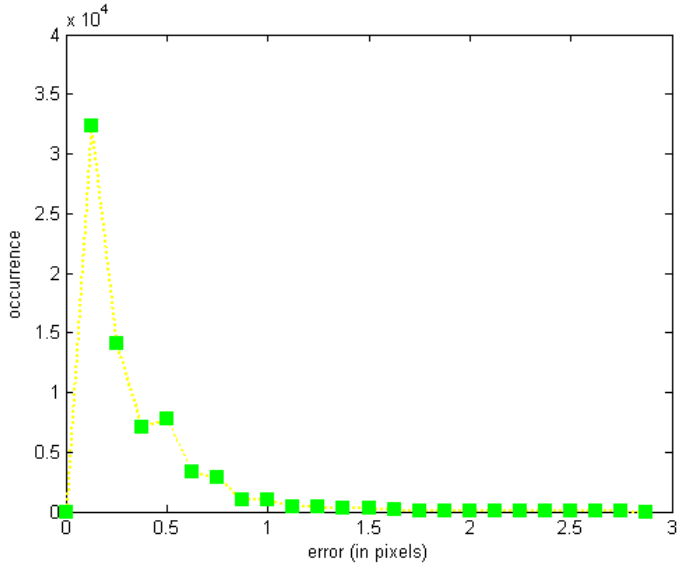
Image example is provided in the corresponding project.

Original image	Image output
	

Error on normal concept



Error on speed-up concept



7 Revision history

Version	Date	Document number	Description
1.0	Nov 2009	U19951EE1V0AN00	Official release