Interactive Camera Distortion Correction

B. Kh. Barladyan, L.Z. Shapiro, I.V. Valiev, A.G. Voloboy
Keldysh Institute of Applied Mathematics RAS, Moscow

Abstract

Here interactive software and algorithm of camera distortion correction is considered. Elaboration of the camera distortion correction is needed during car parking system design. Specific algorithm representation as the set of six scale coefficients tables is introduced. This representation takes into account specific of cameras used in automobile industry and possibility of hardware implementation of given interactively created algorithms. The goal of the interactive software is to provide to car parking system designer a tool for elaboration of camera with desirable (reasonable) distortion. The scale coefficient tables are transferred to camera manufacturer for designed camera creation.

Keywords: *parking camera design, camera distortion, distortion correction, lens design*

1. INTRODUCTION

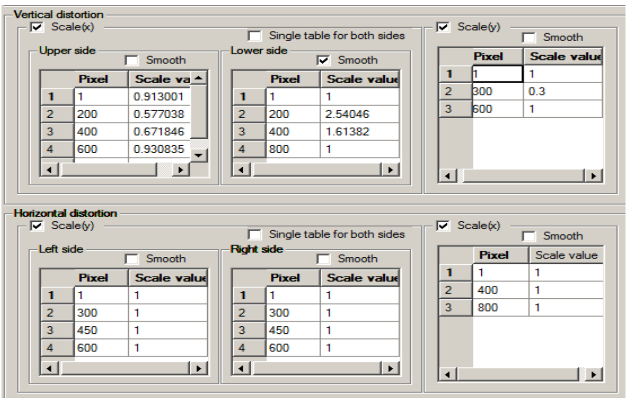
Using different cameras especially reversing cameras in the car parking system becomes almost standard in modern cars. There are a large number of manufacturers offering such system on the market [1, 2 and 3]. These cameras have wide-angle objective about 180 degree or even more to provide maximal visible area. Typically only horizontal extra wide angle is needed in parking systems. Moreover requirements to the visibility of the lower and upper hemispheres are different. Typically the visibility of the lower part of hemisphere is the most essential in parking systems from the driver point of view. So camera makers design cameras with asymmetrical view field. In common case view field may be asymmetrical in horizontal direction also. It is reasonable approach for cameras placed on the left and right bumper sides or on side rearview mirror. The cameras with such extra wide angles and asymmetrical view field unavoidable have large distortions, in common case asymmetrical ones, which should be corrected to better environment understanding by car driver. There are a number of algorithms about distortion correction [4-9]. Some of them are implemented in commercial software [10-12].

During design of parking system developers tune the camera position and orientation, taking into account the given car specific, select cameras with appropriate optical and electric specification. To provide maximally useful and effective overview of critical areas around the car the parking system designers would like to have custom distortion correction for used cameras. In general cases this correction can have own specific for each camera depending on the camera position. Some areas in the image should be magnified and other ones should be reduced.

The parking systems become currently a mass product, so the most reasonable and effective solution becomes embedding of distortion correction algorithm directly in the camera electronics. Camera makers can now implement almost any distortion correction algorithm in camera image processing but the algorithm for given specific camera should be elaborated by parking system designer and passed to the camera maker company in acceptable form. It should be pointed that in some practical cases camera distortions cannot be completely corrected in principle. Typical example is the camera with the view angle more than 180 degree. So the task of parking system designer is elaboration of camera image in form optimized to control the car vicinity by car driver but not distortion correction itself. It is the main purpose of utility where suggested algorithm was implemented.

2. SCALE ALGORITHM REPRESENTATION

Taking into account possible asymmetric of camera distortions, the default image produced by camera is split from correction point of view on four parts by vertical and horizontal axes. The axes intersection point is the fixed point of distortion correction – the position of this pixel is not changed during correction. For rest pixels two type of scaling coefficients are defined. For vertical distortion the scale in given image point is defined as the function of X coordinate – ScaleY(x). These functions are defined separately for upper and lower sides (relatively to horizontal splitting axis). Addition scale function ScaleY(y) defines the scale coefficients as the function of coordinate **y** independently of **x** coordinate. Applying the scaling of the all these three functions does not move the pixels along horizontal splitting axis. The same three functions are defined for horizontal image scaling – ScaleX(y) for left and right image sides and ScaleX(x). Example of User Interface implemented in our software where these three functions in the table form can be set for image with resolution 800x600 is shown on Fig.1.


Vertical distortion
[✓] Scale(x)
[ ] Single table for both sides

Upper side — [ ] Smooth

	Pixel	Scale va
1	1	0.913001
2	200	0.577038
3	400	0.671846
4	600	0.930835

Lower side — [✓] Smooth

	Pixel	Scale value
1	1	1
2	200	2.54046
3	400	1.61382
4	800	1

[✓] Scale(y) — [ ] Smooth

	Pixel	Scale value
1	1	1
2	300	0.3
3	600	1

Horizontal distortion
[✓] Scale(y)
[ ] Single table for both sides

Left side — [ ] Smooth

	Pixel	Scale value
1	1	1
2	300	1
3	450	1
4	600	1

Right side — [ ] Smooth

	Pixel	Scale value
1	1	1
2	300	1
3	450	1
4	600	1

[✓] Scale(x) — [ ] Smooth

	Pixel	Scale value
1	1	1
2	400	1
3	800	1

Fig.1. Example of six scale tables.

Scale coefficients are defined for some set of pixels and are interpolated between them. By default linear interpolation is used. Apart linear interpolation the spline interpolation can be applied as well. Scale functions are displayed in the graphical form as it is showed on Fig. 2. Also it is possible to use the same scale function for both image sides (horizontal or vertical).

Interface, presented on Fig. 1, provides edition of all six scale functions. So, practically arbitrary distortion correction can be created in this way.

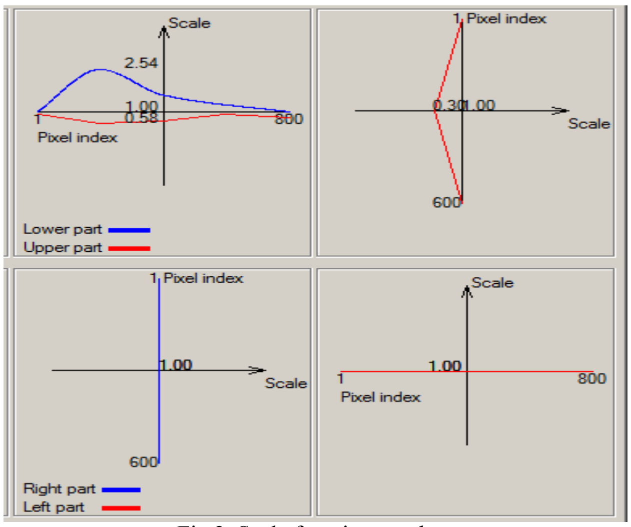


Fig.2. Scale function graphs.

3. DIRECT IMAGE CORRECTION

Specification of distortion correction in numerical form is not very convenient for parking camera system designer. It is more desirable to provide interface directly on image, so that user can move the image point from one position to another one, while the scale functions will be created for this correction automatically. This feature was provided in our software (Fig. 3).

In direct image correction mode the auxiliary grid provides convenient visual control how current distortion functions affect on original image. A designer can select color and step of this grid. Green contour provides presentation of original image size. Also it is possible to directly move selected pixel to the new position by mouse drag and drop. Application provides appropriate changes of scale tables and real time image correction.

Additional slider provides convenient control of scale along X or Y axis according selected line in appropriate ScaleY(y) or ScaleX(x) table.

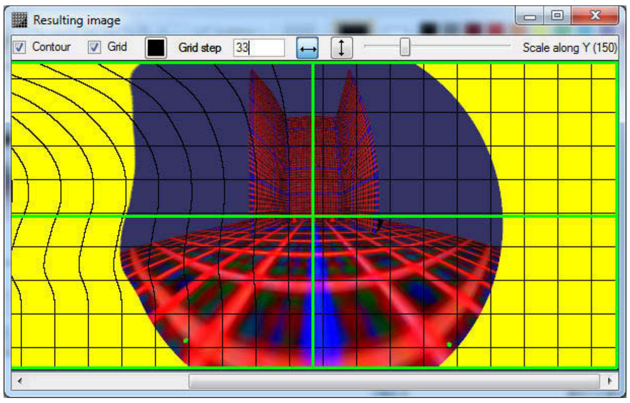


Fig.3. Direct image correction.

But this feature elaboration was rather complex because automatic creation of scale functions has various reasonable solutions. The image scale in horizontal and vertical directions can be considered independently due to selected correction algorithm representation, but scale in each direction in general case depends on two functions. In horizontal direction, for example it is ScaleX(y) and ScaleX(x). It is hard to find reasonable and transparent for a user proportion between these two functions of given image pixel moving. Moreover the new X position of pixel is determined by integral of ScaleX(x) from zero (Y axis) till its initial X coordinate (see details of algorithm below). So the movement of pixel can be achieved using different ScaleX(x) function. Only corresponding integral is essential. Due to this reason the direct image correction in our system affects on ScaleX(y) and ScaleY(x) functions only. Even with this restriction the task is not trivial as it is described in algorithms details below.

4. CORRECTION ALGORITHM DETAILS

4.1 Smoothing algorithm

The third order polynomial was used for smooth interpolation of scale coefficients between points of definition. To provide really smooth interpolation it is desirable that extreme points (maximal/minimal scales) in linear interpolation are to be extreme points of spline interpolation. In another words the interpolation should be monotony between node points. This requirement is provided in described below algorithm by special definition of derivations in knot points.

The segment of the original polyline between all the pairs of "knots" $(x_i,\ y_i)$ and $(x_{i+1},\ y_{i+1})$ is smoothed by the third order polynomial qi(x), where

$$q_i(x_{i+1}) = q_{i+1}(x_{i+1}) = y_{i+1}$$
$$q'_i(x_{i+1}) = q'_{i+1}(x_{i+1})$$

The third order polynomial q(x) for which

$$q(x_1) = y_1$$
$$q(x_2) = y_2$$
$$q'(x_1) = k_1$$
$$q'(x_2) = k_2$$

can be written in symmetrical form

$$q = (1\text{-}t) \cdot y_1 + t \cdot y_2 + t \cdot (1\text{-}t) \cdot (a\,(1\text{-}t) + b\,t)$$

where

$t = (x - x_1) / (x_2 - x_1)$

$a = k1 \cdot (x_2 - x_1) - (y_2 - y_1)$

$b = - k2 \cdot (x_2 - x_1) + (y_2 - y_1)$

The derivatives in the knots are defined as followed:

1. For the first knot:

$$k_1 = (y_2 - y_1) / (x_2 - x_1) \qquad (1)$$

2. For the last knot:

$$k_n = (y_n - y_{n-1}) / (x_n - x_{n-1})$$

For intermediate knots:

$$k_i = 0 \text{ if } y_i \le y_{i-1} \text{ and } y_i \le y_{i+1} \qquad (2.1)$$
$$ki = 0 \text{ if } y_i \ge y_{i-1} \text{ and } y_i \ge y_{i+1} \qquad (2.2)$$
$$ki = (y_{i+1} - y_{i-1}) / (x_{i+1} - x_{i-1}) \text{ for all other cases.} \qquad (2.3)$$

The derivations definition 2.1 and 2.2 provides that extreme points of linear interpolation remain extreme one for smooth interpolation also.

4.2 Output image smoothing

The scale tables described above describe transformation of any point from original image to the corrected one, but both images are discretized ones, i.e. consist of pixels. So we have some freedom how pixels of output image will be constructed from original ones. The described below algorithm was constructed to provide relatively smooth image and minimize moiré and aliasing effects.

In general case the original image pixel will change its form in output image. It will be scaled depending on its X and Y coordinates. Scale of any pixel which don't have its own value in the table is calculated as a linear or smooth (see p. 4.1) interpolation between the nearest lower and higher values.

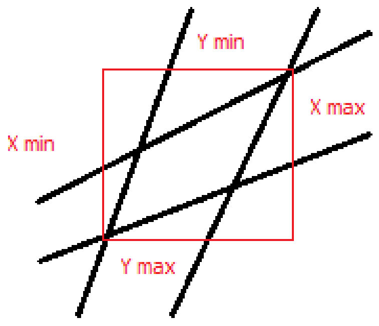


Fig.4. Scaling a pixel

Calculated coordinates of each pixel vertex after scaling and its bounding box drawn by red color are shown on Fig.4. We consider this bonding box as scaled original pixel which color should be put to the all pixels of new (output) image, which it covers as it is shown on Fig. 5.

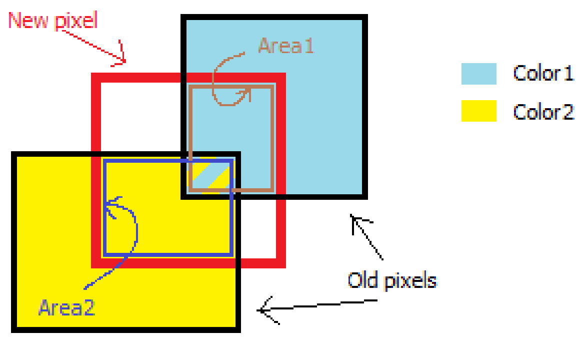
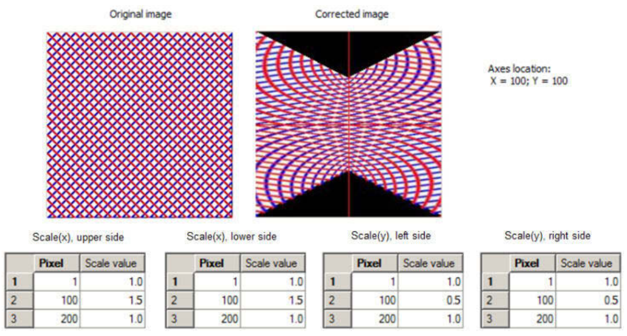


Fig.5. Color setting for a new image pixel.

It should be taken into account that scaled pixels (initial, "old" pixels on the Fig.5) can have overlapping due to extension by bounding box described above. Finally, the color of output is calculated as the weighted sum of initial pixels colors with weight of intersection area of old pixel with new one.

The result of image scaling for some test example is shown on Fig.6.



Scale(x), upper side

	Pixel	Scale value
1	1	1.0
2	100	1.5
3	200	1.0

Scale(x), lower side

	Pixel	Scale value
1	1	1.0
2	100	1.5
3	200	1.0

Scale(y), left side

	Pixel	Scale value
1	1	1.0
2	100	0.5
3	200	1.0

Scale(y), right side

	Pixel	Scale value
1	1	1.0
2	100	0.5
3	200	1.0

Fig. 6. Result of image scaling.

5. DIRECT IMAGE CORRECTION ALGORITHM

The procedure assumes that the user will move any pixels from one position (start position) to another one (end position). And the algorithm should calculate such scale transformations so that original pixel (correspondent pixel of non-scaled picture) will be moved to the selected end position. This task includes two sub-tasks:

- determination of initial pixel position by its start position on the current scaled image;
- calculation of new scale table described in p.2. (or correction of the current one) which moves pixel from initial position to the end one.

5.1 Calculation of initial pixel

The new (transformed) pixel coordinates (xn, yn) are calculated from the original (x_o, y_o) ones by the following formulae:

$$x_n(x_0, y_0) = x_a + (x_0 - x_a)\cdot S_{xy}(y_0)\cdot \int_{x_a}^{x_0} S_{xx}(x)dx \quad (3)$$

$$y_n(x_0, y_0) = y_a + (y_0 - y_a)\cdot S_{yx}(x_0)\cdot \int_{y_a}^{y_0} S_{yy}(y)dy \quad (4)$$

Here:

- x_o,y_o are original pixel coordinates;
- x_n,y_n are transformed (scaled)pixel coordinates;
- S_{xy}, S_{xx}, S_{yx}, S_{yy} are scale functions defined in p.2;
- x_a – **Y** axis position
- y_a – **X** axis position

5.1.1 Area subdivision and scales description.

The scale functions definition in table form described in p.2 subdivides the initial pixel image into rectangular cells. The horizontal bounds of these cells will be y-lines passed throw the pixels where scale functions $S_{xy}(y)$ (left and right) and $S_{yy}(y)$ are defined. The vertical bounds of these cells will be x-lines passed throw the pixels where scale functions $S_{yx}(x)$ (upper and low) and $S_{xx}(x)$ are defined.

Let the bounds of i-th rectangle are x_{mini}, x_{maxi}, y_{mini} and y_{maxi} Inside this rectangle we can represent integrals in (3) and (4) as:

$$\int_{x_a}^{x_0} S_{xx}(x)dx = \int_{x_a}^{x_{mini}} S_{xx}(x)dx + \int_{x_{mini}}^{x_0} S_{xx}(x)dx \quad (5)$$

$$\int_{y_a}^{y_0} S_{yy}(y)dy = \int_{y_a}^{y_{mini}} S_{yy}(y)dy + \int_{y_{mini}}^{y_0} S_{yy}(y)dy \quad (6)$$

Taking into account that the $S_{xy}(y_0)$ is constant for given y_0 and the $S_{yx}(x_0)$) is constant for given x_0 the scale transformation of initial pixel (x_0, y_0) to the scaled one (x_n, y_n) has the two following properties:

Property 1.

If y_o = const and $x_{o1} < x_{o2}$ then $x_{n1} < x_{n2}$. If y_o = const and $x_{o1} > x_{o2}$ then $x_{n1} > x_{n2}$. So in this case $x_n(x_o)$ is monotone increasing function. If follows from (3) and (5).

Property 2.

If x_o = const and $y_{o1} < y_{o2}$ then $y_{n1} < y_{n2}$. If x_o = const and $y_{o1} > y_{o2}$ then $y_{n1} > y_{n2}$. So in this case $y_n(y_o)$ is monotone increasing function. If follows from (4) and (6).

From properties 1 and 2 follow that the maximal and minimal values of scaled coorfinates x_n and y_n will be achieved on the scaled boundaries of original rectangular boundaries.

Moreover the maximum and minimum of x_n will be achieved on the rectangle vertices. If we set $x_o = x_{mini}$ then from (3) and (5) we have:

$$x_n(x_{mini}, yo) = x_a + K_1 \cdot S_{xy}(y_o) \qquad (7)$$

where

$$K_1 = (x_{mini} - x_a) \cdot \int_{x_a}^{x_{mini}} S_{xx}(x)dx,$$

and the $S_{xy}(y_0)$ is monotone function of y_0 inside the given cell (see p.4.1). So the minimum and maximum of (7) will be achieved on the segment ends. The same statement is true for $x_0 = x_{maxi}$ and finally for y_n.

Taking into account all these properties of scale transformation we implement the following algorithm for calculation initial (x_0, y_0) pixel from (x_n, y_n) scaled one:

5.1.2 Initial pixel finding algorithm.

1. Create the list of bounding boxes of scaled rectangular cells described above. Transformed original rectangular cells will be curved one in general case (see Fig. 3, for example) and so bounding boxes of scaled rectangular cells will be overlapped. Due to this reason the given scaled pixel can belong to the several bounding boxes simultaneously.

2. For each cell from list try to determine initial pixel by the following way:

3. If the given bouning box does not include (x_n,y_n) then go to the next one.

4. Calculate scaled (x_{n1},$_{yn1}$) pixel for center of original cell. If the distance from (x_{n1},$_{yn1}$) pixel to the (x_n,y_n) one is lesser than 1 then original pixel is found, (x_o,y_o) is center given original cell.

5. If width and height of original box is lesser than 1 then original pixel can not be found in the given cell. Go to the next bounding box.

6. Divide given original box on two ones by division of width or height in half. For each half calculate box of scaled cell. For each half cells execute pp. 3-6.

In the result we have calculated the initial pixel position.

5.2 Scale table correction

As it was pointed above the correction will not touch the $S_{xx}(x)$ and $S_{yy}(y)$ functions and so the correction will be done for $S_{xy}(y)$ and $S_{yx}(x)$ only. Corrections for $S_{xy}(y)$ and $S_{yx}(x)$ can be applied independently. So consider the $S_{yx}(x)$ only. Firstly consider only the linear interpolation between node points. One scale segment for $S_{yx}(x)$ is represented on the Fig. 7.

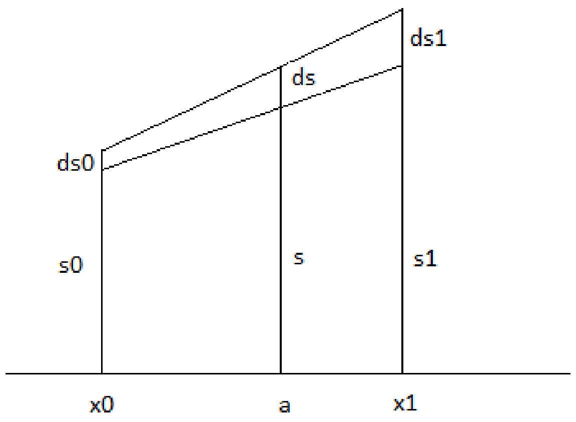


Fig. 7. Segment of scale function.

Here the movement pixel along x = **a** coordinate is considered. The point **a** belongs to the (x0,x1) segment. Let us denote the scale values and its variation at the ends of the segment as s0, s1, ds0 and ds1 correspondingly. The scale and scale variation in the point **a** denote as **s** and **ds**. We want to determine correspondent changes ds0 and ds1 which provide ds changing in the point **a** in the linear interpolation case.

For simplicity, we will use the dimensionless coordinates.

Let x0 = 0 and x1 = 1, a1 = (a - x0) / (x1 - x0).

Due to the linear interpolation we have:

$$s0 \cdot (1-a) + s1 \cdot a = s$$
$$(s0+ds0) \cdot (1-a) + (s1+ds1) \cdot a = s+ds$$

So

$$ds0 \cdot (1 - a) + ds1 \cdot a = ds \qquad (8)$$

We should determine ds0 and ds1 via ds and a. Let us will find solution in form:

$$ds0 = ds \cdot f_0(a); \quad ds1 = ds \cdot f_1(a). \qquad (9)$$

From (8) and (9) we have

$$f_0(a) \cdot (1-a) + f_1(a) \cdot a = 1; \qquad (10)$$

From (9) and (10) we have the following boundary conditions:

$$f_0(0) = 1; \quad f_1(1) = 1.$$
$$f_0(1) = 0; \quad f_1(0) = 0.$$

We also naturally suppose functions symmetry:

$$f_0(a) = f_1(1-a) \qquad (11)$$

From (10) and (11) also followed that

$$f_0(0.5) = f_1(0.5) = 1$$

So finally we have the following conditions

$$\begin{cases} f_0(0) = 1; & f_0(0.5) = 1; & f_0(1) = 0; \\ f_1(0) = 0; & f_1(0.5) = 1; & f_1(1) = 1; \end{cases} \qquad (12)$$

There are many functions which satisfy to these conditions. For example we can determine $f_0(a)$ on [0.5, 1.0] as any function decreasing from 1.0 to 0.0. In this case $f_1(a)$ will be determined on [0, 0.5] by (11). Then we can determine $f_0(a)$ on [0, 0.5] by substituting $f_1(a)$ in equation (10) and solving it for $f_0(a)$. We conider the following two variants of solution:

5.2.1 1st variant of solution.

Let

$$f_0(a) = 2 \cdot (1 - a) \text{ on } [0.5, 1] \qquad (13)$$

and so from (11)

$$f_1(a) = 2 \cdot a \text{ on } [0, 0.5] \qquad (14)$$

From (10) and (13) we have

$f1(a) = (1 - 2 \cdot (1 - a)^2) / a$ on [0.5, 1]

and from (10) and (14) we have

$f_0(a) = (1 - 2 \cdot a^2) / (1 - a)$ on [0, 0.5]

And finally:

$$f_0(a) = \begin{cases} (1 - 2 \cdot a^2)/(1 - a), & on\ [0, 0.5] \\ 2 \cdot (1 - a), & on\ [0.5, 1] \end{cases}$$

$$f_1(a) = \begin{cases} 2 \cdot a, & on\ [0, 0.5] \\ (1 - 2 \cdot (1 - a)^2)/\ a, & on\ [0.5, 1] \end{cases}$$

Both functions has the same maximum ~1.172 in points $1 - \sqrt{0.5}$ and $\sqrt{0.5}$ appropriately.

5.2.2 2nd variant of solution.

Let us will find the solution in the parabola form $f_0(a) = k2 \cdot a^2 + k1 \cdot a + k0$ on [0, 1]. According (12) this parabola should pass through the points (0, 1), (0.5, 1) and (1.0). These restrictions completely define parabola coefficients:

$$f_0(a) = -2 \cdot a^2 + a + 1 \quad on \quad [0, 1]$$

From (11)

$$f_1(a) = -2 \cdot a^2 + 3 \cdot a \quad \text{on} \quad [0, 1]$$

$f_0(a)$ decreases from 1.0 to 0.0 on [0.5, 1.0]

It is easy to check that f_0 and f_1 satisfy the (10) equation.

Both functions has the same maximum 1.125 in the points 0.25 and 0.75 appropriately.

The less the function maximum the less will be scale values variations in the nodes and so the resulted curves will be more smooth. So we selected parabola for $f_0()$ and $f_1()$ functions.

5.2.3 *Spline case.*

Described above in p.5.2.2 solution works in linear interpollation case only. In spline case the task become nonlinear one and can not be solved analitically. In this case the solution from p.5.2.2 is used as initial approximation. Let us linear interpollation for given ds gives (ds0, ds1) solution. Then the non linear solution ($ds0^n$, $ds1^n$) we will find by using the following equation:

$$ds1^n = K \cdot ds0^n, \text{ where } K = ds1 / ds0; \qquad (15)$$

Let us denote the function which calculate the scale variation in point **a** by using spline interpollation between scale table nodes as $F_{spl}(x)$, where x is scale variation in left segment point and the scale variation in the right sement point is defined by (15). Then $ds0^n$ can be found by solving the following non linear equation:

$$F_{spl}(ds0^n) = ds$$

We solve this equation by founding the solution inside segment. The one boundary is defined by linear interpollation approximation and for the second one is the maximal acceptable scale if $F_{spl}(ds0)$> ds and minimal one in oposite case.

6. RESULTS

Described interactive software was implemented as additional application (plugin) for CATIA [13, 14] CAD/PDM system. It can process both with images produced by real cameras and with ones simulated in CATIA by our products [14]. The plugin provides real time design of distortion correction algorithm in form of both scale tables and interactive resulting image modification. Corrected image is re-drawn during fraction of a second after parameters changing (Intel Core 2 Q9550 2.83Ghz).

The scale tables described above contains distortion correction information in a form acceptable for camera creation by the camera manufacturer. And implemented algorithms take into account parking system cameras manufacturing specific. In the result the implemented software provides effective and convenient tool for car park system designers to develop reasonable correction of camera images.

7. AKNOLEDGMENTS

This work was supported by the RFBR, grant10-01-00302, and by INTEGRA Inc. (Japan).

8. REFERENCES

[1] http://www.parkingcameras.com/store/home.php

[2] http://www.espow.com/wholesale-car-electronics-car-review-systems-rear-view-cameras.html

[3] http://www.thecarkitcompany.com.au/index.php/products/parking-sensors-a-reverse-cameras

[4] F. Devernay and O. Faugeras. Automatic calibration and removal of distortion from scenes of structured environments. SPIE Conference on investigative and trial image processingSanDiego, CA, 1995.

[5] H. Farid and A.C. Popescu. Blind removal of Lens Distortion. Journal of the Optical Society of America, 2001.

[6] J. Jedlička, M. Potůčková. Correction of Radial Distortion in Digital Images. Charles University in Prague Faculty of Science, http://dsp.vscht.cz/konference_matlab/MATLAB07/prispevky/jedlicka_potuckova/jedlicka_potuckova.pdf

[7] Janez Perˇs, Stanislav Kovaˇciˇc. Model-Based Radial Lens Distortion Correction Using Tilted Camera Assumption. Faculty of Electrical Engineering University of Ljubljana Nonparametric, http://vision.fe.uni-lj.si/docs/janezp/pers-wwk2002.pdf

[8] R. Swaminatha and S.K. Nayer. Non-metric calibration of wide angle lenses and poly-cameras. IEEE Conference on computer Vision and pattern recognition, pp 413, 1999.

[9] G. Taubin. Camera model for triangulation. Lecture notes EE-148, 3D Photography, Caltech, 2001.

[10] PTLens, http://epaperpress.com/ptlens/

[11] Photoshop, Correcting image distortion, http://helpx.adobe.com/photoshop/using/correcting-image-distortion-noise.html

[12] IRIS TUTORIAL, DSLR images distortion correction, http://www.astrosurf.com/buil/iris/tutorial19/doc42_us.htm

[13] CATIA - Virtual Design for Product Excellence, http://www.3ds.com/products/catia/welcome/

[14] Inspirer, Specter optical simulation system http://www.integra.jp/en/index.html

About the authors

Boris Kh. Barladyan, PhD, senior researcher, Keldysh Institute for Applied Mathematics RAS.

E-mail: obb@gin.keldysh.ru

Lev Z. Shapiro, PhD, senior researcher, Keldysh Institute for Applied Mathematics RAS.

E-mail: pls@gin.keldysh.ru

Ildar V. Valiev, researcher, Keldysh Institute for Applied Mathematics RAS.

Alexey G. Voloboy, PhD, senior researcher, Keldysh Institute for Applied Mathematics RAS.

E-mail: voloboy@gin.keldysh.ru