Document Image Enhancement with Perspective Transformation using Python

Indra Bayu Muktyas[1], Nerru Pranuta Murnaka[2], Samsul Arifin[3], Wikky Fawwaz Al Maki[4]
{indrabayu.muktyas@stkipsurya.ac.id[1], nerru.pranuta@stkipsurya.ac.id[2], samsul.arifin@binus.edu[3], wikkyfawwaz@telkomuniversity.ac.id[4]}

[1,2]STKIP Surya, Tangerang, Indonesia, [3]Binus University, Jakarta, Indonesia, [4]Telkom University, Bandung, Indonesia

Abstract. Many people need soft files of their important documents. One of the technological sophistications that are currently developing is to use a smartphone camera to photograph a document. However, the results from the photo are not like the results with a precision-shaped scanner. Based on this, we developed a program that can improve the quality of a distorted (imprecise) document image to become a scanned-like rectangle image. In addition, the purpose of this study is to find out some of the mathematical theories behind it. The method we use is a literature study on line gradients, perspective transformations, and central tendency. Then we apply it to the python program. The results of this study are gradients can be used to determine the boundaries of the rectangular area of the document image. The perspective transformation can be used to change a shape from an arbitrary quadrangle to a rectangle. Central tendency (mean, median, and mode) can be used to fill in the gaps in the perspective transformed image.

Keywords: scanline, gradient line, perspective transformation, measure of central tendency, python

1 Introduction

Image of the obtained document from the camera no like the picture scan results. Picture not truly seen from the top. Documents in the form of rectangles are long often seen, as side four anything that doesn't have precision. Repairing picture documents to form rectangles long can be done from a transformation perspective. Pictures that can be transformed by transformation perspective must be in the form of a flat image. Some research about repair distortion images with a transformation perspective was conducted by [1], [2]. In research, the transformation perspective is still written in general, with less detail on how the process the math.

Transformation perspective is also used in other things, one of them is on data augmentation for object detection [3]. With a transformation perspective, some pictures could be put together and become a panoramic image. This has been done in [4].

In this paper, improved picture documents will be written with angle look draft mathematics basics, including line gradient transform perspective, and measures of central tendency. The gradient is used to find out the dot, dot, dot corner from the picture document. Next will be used in the scanline method to detect special only areas rectangular from the document. Every point in the area then changed Becomes shaped rectangle long with a transformation perspective.

ICEHHA 2022, October 21-22, Ruteng, Indonesia
Copyright © 2023 EAI
DOI 10.4108/eai.21-10-2022.2329633

There are some holes in the picture results of the transformation. It is filled by utilizing theory measures of central tendency, namely mean, median, and mode.

2 Method and Materials

In this section, we will study about scanline, perspective transform, find and fill the blank pixels, and their implementation in python.

2.1 Scanline method

Document image can be seen as quadrangle. We know the coordinate of points A, B, C, and D. We can see in the Figure 1.

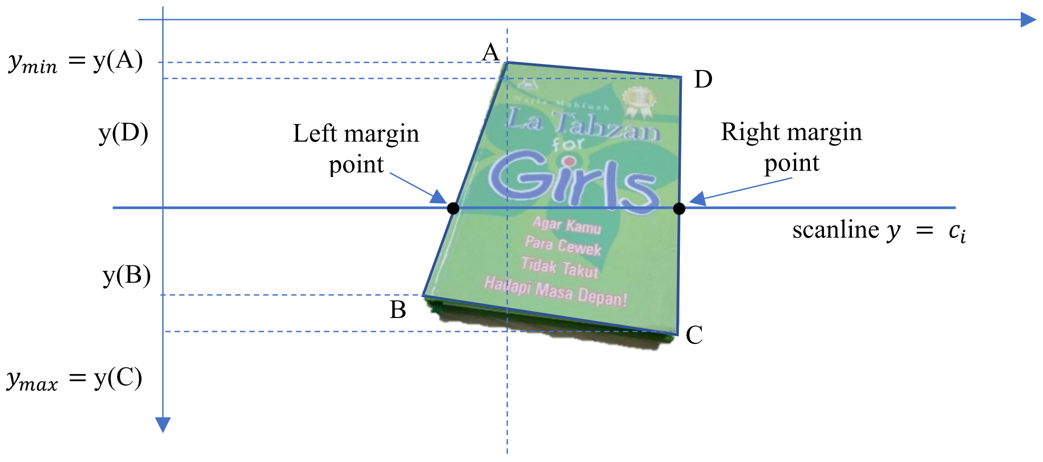


Fig. 1. Region of picture document

To determine the location from pixels only in area of quadrangle, we can use the scanline [5]. We start scanning from the y value at the top pixel (y_{min}), to the y value at the bottom pixel (y_{max}). Because the shape of the quadrangle is arbitrary, then we first order the vertices of A, B, C, and D based on their y value. Create a dividing line, that is dotted line, the point next to left from the topmost point and the point next to right or equal to the topmost point. The amount of possibility from the arrangement of the dots below top point is $2^3 = 8$. Several possibilities from quadrangle could see in Figure 2.

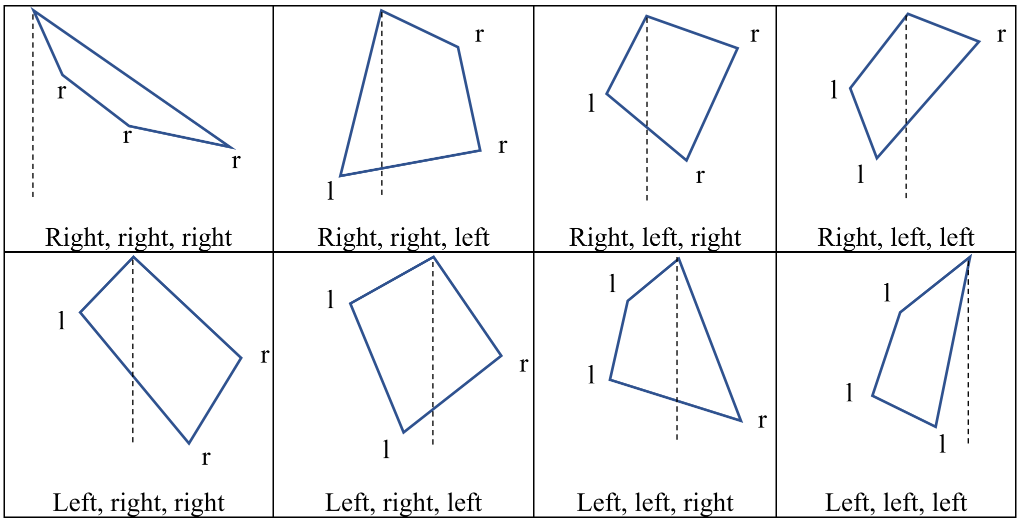
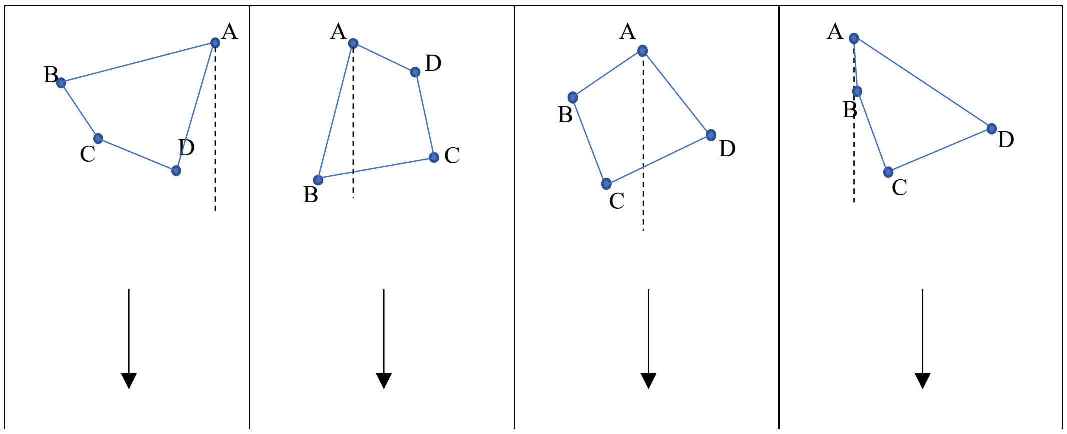


Fig. 2. Possibility arrangement of the points below the top point.

After that, each set our defined gradient of the line through the topmost point (i.e. A) with each point other. The gradient of the line through two points A and B are [6]

$$m = \frac{y(B) - y(A)}{x(B) - x(A)} \tag{1}$$

Together with the topmost point, the points next to the left will produce a gradient marked negative, while the dots next to right will produce a positive gradient. The gradient value from each set is then sorted by decreased (big to small). After the point is sorted by descending order based on gradient, combine the second set to set left more first, then set right. An illustration could be seen in Figure 3.



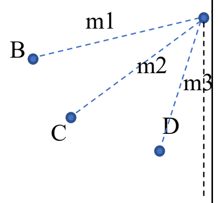
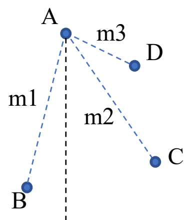
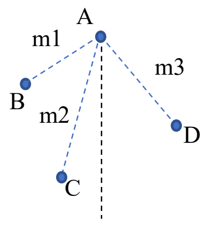
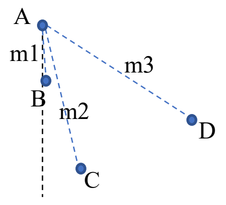
 $[m1 > m2 > m3] + [\ \ ]$ [B, C, D] + [ ] = [B, C, D]	 $[m1] + [m2 > m3]$ [B] + [C, D] = [B, C, D]	 $[m1 > m2] + [m3]$ [B, C] + [D] = [B, C, D]	 $[\ \ ] + [m1 > m2 > m3]$ [ ] + [B, C, D] = [B, C, D]

Fig. 3. Formation combined set order

The points corresponding to the sides left are determined by taking the points in the set combined from the adjacent left, while the points corresponding to the sides right are determined by taking the points in the set combined from the adjacent right. Next these points will become limit left and limit right on the scanline. For example, if the set combined is [B, C, D] then the side left starts from the far left point, i.e. B, while the side right started from the rightmost point, which is D. We draw a line $y = c_i$for $y(A) \leq c_i \leq y(C)$. Next, we determine the left and right boundaries right. The left border is score *x* from point cut Among side left, i.e. the line through points A and B and the line $y = c_i$. The line that passes through points A and B can be found by

$$\frac{y - y(A)}{y(B) - y(A)} = \frac{x - x(A)}{x(B) - x(A)} \tag{2}$$

We substitute $y = c_i$ into the equation, and we get

$$\frac{c_i - y(A)}{y(B) - y(A)} = \frac{x_{kiri} - x(A)}{x(B) - x(A)} \tag{3}$$

$$x_{left} = \frac{\big(c_i - y(A)\big) \cdot \big(x(B) - x(A)\big)}{y(B) - y(A)} + x(A) \tag{4}$$

In the same way, the limit right is

$$x_{right} = \frac{\big(c_i - y(D)\big) \cdot \big(x(C) - x(D)\big)}{y(C) - y(D)} + x(D) \tag{5}$$

More goes on, actually limit left is determined by the intersection side left and line $y = c_i$. When $c_i \leq y(B)$, the line segment used is AB, while for $c_i > y(B)$, the line segment is used in BC. Likewise for the right border. At this time $c_i \leq y(D)$, the line segment used is AD, while $c_i > y(D)$ the line segment used is DC. After the limit left and limit right are known, next stay our map the pixels from (x_{left}, c_i) until (x_{right}, c_i). And that is done from $y = y(A)$ until $y =$

$y(C)$. Because the scanline method sort the corner point based on y, then no matter the order of 4 corner points, the area of quadrangle will by appropriate detected.

2.2. Perspective transformation

It deals with homogeneous coordinates. At homogeneous coordinates, the point is symbolized by a tuple of 3 numbers, namely (x_h, y_h, w_h). Every point in coordinates cartesian could be changed to homogeneous coordinates in the way $(x_c, y_c) = (x_h/w_h, y_h/w_h)$. As for changing the point in coordinates cartesian to in homogeneous coordinates in the way $(x_h, y_h, w_h) = (x_c, y_c, 1)$.

The mapping process is done using a matrix transformation-shaped perspective

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \tag{6}$$

Need to be noticed that entry $M_{33} = 1$. The points in the Cartesian coordinates (x, y) are converted into homogeneous coordinates $(x, y, 1)$. By matrix multiplication, we get

$$\begin{bmatrix} x't \\ y't \\ t \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{7}$$

clear that

$$x' = \frac{ax + by + c}{gx + hy + 1}, \qquad y' = \frac{dx + ey + f}{gx + hy + 1} \tag{8}$$

With a little calculation algebra obtained

$$\begin{aligned} x' &= ax + by + c - gxx' - hyx' \\ y' &= dx + ey + f - gxy' - hyy' \end{aligned} \tag{9}$$

To search score $a, b, c, d, e, f, g,$ and h, it takes four pairs of origins (x_i, y_i) and maps (x_i', y_i') where $i = 1,2,3,4$. We get the matrix equation

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \\ y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_1y_1' & -y_1y_1' \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} \tag{10}$$

So, we know the value of a, ..., h by

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_1y_1' & -y_1y_1' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_1y_1' & -y_1y_1' \end{bmatrix}^{-1} \cdot \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \\ y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} \tag{11}$$

In detail, the determination matrix perspective could be seen in Algorithm 1.

Algorithm 1. Determination matrix perspective

```
input: 4 dots source, 4 dots destination
A = []
b = []
for i in {0.. len(source)}:
  xs , ys = source [ i ]
  xt , yt = destination [ i ]
  A.append ([ xs , ys , 1, 0, 0, 0, - xt * xs , - xt * ys ])
  A.append ([ 0, 0, 0, xs , ys , 1, - yt * xs , - yt * ys ])
b += [ xt , yt ]
h = solve( A, b)
h.append ([1])
Output : M = h.reshape((3x3))
```

2.3 Algorithm push one by one

How to map a dot, a dot, a dot is to multiply matrix transformation perspective with position point that. If done one by one, then the time required will be longer. This is caused by mapping conducted pixel by pixel, from the point with the highest *y*, to the point with the lowest *y*. Then in each line, done from the point with *x* on the boundary left to *x* at the limit right. Suppose, the number of pixels in quadrangle area is *n*, then multiplication matrix $M \cdot [x_i, y_j, 1]^T$ done *n* times. Matrix map is a matrix of 3 × 1, that is $map = [x't, y't, t]^T$. From here, we can get the *x*' and y' coordinates by $x' = \frac{x't}{t} = \frac{map[1]}{map[3]}$ and $y = \frac{y't}{t} = \frac{map[2]}{map[3]}$ where $map[i]$ is the *i*-th line from matrix *map* with *i* = 1, 2, 3. Overall, push algorithm could be seen in Algorithm 2.

Algorithm 2. Algorithm push 1 (map pixels one by one)

```
Input: g = matrix of document photo
sources = 4 points quadrangle in g

h = max(distance( sources[ 0], sources[1]),
distance(sources[2], sources[3]))
w = max(distance( sources[ 0], sources[3]),
distance(sources[1], sources[2]))

# find the orientation of the paper (portrait or landscape)
if h > w: # portrait
w = h / sqrt( 2)
else: # landscape
```

```
w = h * sqrt( 2)
dest = [[0,0], [ 0,h ], [ w,h ], [w,0]]
m_transformed = [ 0]* (size=(h, w, number_of_layer (g)))

# to detect the "hole"
m_transformed [:,:, 0] = [-1]
M = perspective_ matrix ( sources, dest ) # use Algorithm 1

sources_sort_by_y = sort sources by y value, ascending order
A = top point of sources_sort_by_y

left_gradient = [gradient( A,P )], P={points in the left of A}
right_gradient = [ gradient ( A,Q )], Q={points in the right
of A}
sort points in P and Q based on left_gradient and
right_gradient descending order
ordered_points = [ left_points ] U [ right_points ]

id_left = 0; id_right = 2
point1_left = A; point1_right = A
point2_left = ordered_points [ id_left ]; point2_right =
ordered_points [ id_right ]

for y in ( ymin_sources , ymax_sources ):
  if y > y( ordered_points [ id_left ])
    id_left += 1
    point1_left = point2_left
    point2_left = ordered_points [ id_left ]

  if y > y( ordered_points [ id_right ])
    id_right -= 1
    point1_right = point2_right
    point2_right = ordered_points [ id_right ]

# use equations (4) and (5)
  x_left = intersect of y and line(point1_left, point2_left)
  x_right = intersect of y and line(point1_right,
point2_right)

# transform every pixel value of quadrangle in g
  for x in ( x_left , x_right ):
    [ x't , y't , t] $^T$ = M x [x, y, 1] $^T$
    x' = x't /t
    y' = y't /t
    if x' and y' in the destination paper dimension
      m_ transformed [ y', x'] = g[y, x]
Outputs: m_transformed
```

2.4 Algorithm push index (all at once)

We can form a matrix index *id* with size of $3 \times n$, which contains the index or location of the pixels in the quadrangle area, arranged in a row to the right. Suppose(x_1, y_1) is the position of first pixels and $(x_r, y_s)$is the position of the last pixel in the area of the rectangle, then

$$id = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_r \\ y_1 & y_2 & y_3 & \cdots & y_s \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}$$

Then, the matrix multiplication of $map = M \cdot id$ only done 1 time. It should be noted that *the map* is a matrix with size $3 \times n$, that is

$$map = \begin{bmatrix} x'_1 t_1 & x'_2 t_2 & x'_3 t_3 & \cdots & x'_r t_n \\ y'_1 t_1 & y'_2 t_2 & y'_3 t_3 & \cdots & y'_s t_n \\ t_1 & t_2 & t_3 & \cdots & t_n \end{bmatrix}$$

x' and *y*' are obtained with $x'_i = \frac{x'_i t_j}{t_j} = \frac{map[1]}{map[3]}$and $y'_i = \frac{y'_i t_j}{t_j} = \frac{map[2]}{map[3]}$. The practice (in python) is to collect id_x', i.e. the first row of matrix *map*, collect id_y', i.e. the second row of the *map* matrix, then reindex pictures based on id_x' and id_y'. The array $map[id\_x', id\_y']$ will produce the desired sequence. This will save a lot of time in the transformation process. For more details, it is presented in detail in Algorithm 3.

Algorithm 3. Algorithm push 2 (use index on matrix multiplication)

```
Input: g = matrix of document photo
sources = 4 points quadrangle in g

h = max(distance( sources[ 0], sources[1]),
distance(sources[2], sources[3]))
w = max(distance( sources[ 0], sources[3]),
distance(sources[1], sources[2]))

# find the orientation of the paper (portrait or landscape)
if h > w: # portrait
    w = h / sqrt( 2)
else: # landscape
    w = h * sqrt( 2)
dest = [[0,0], [ 0,h ], [ w,h ], [w,0]]
m_transformed = [ 0]* (size=( h,w,number_of_layer_of (g)))

# to detect the "hole"
m_transformed [:,:, 0] = [-1]
M = perspective_ matrix ( sources, dest ) # use Algorithm 1

sources_sort_by_y = sort sources by y value, ascending order
A = top point of sources_sort_by_y

left_gradient = [gradient( A,P )], P={points in the left of A}
right_gradient = [ gradient ( A,Q )], Q={points in the right
of A}
sort points in P and Q based on left_gradient and
right_gradient descending order
ordered_points = [ left_points ] U [ right_points ]

id_left = 0; id_right = 2
point1_left = A; point1_right = A
point2_left = ordered_points [ id_left ]; point2_right =
ordered_points [ id_right ]
index = [ 0]* (size=(3,0))
for y in ( ymin_sources , ymax_sources ):
  if y > y( ordered_points [ id_left ])
    id_left += 1
point1_left = point2_left
point2_left = ordered_points [ id_left ]

  if y > y( ordered_points [ id_right ])
    id_right -= 1
point1_right = point2_right
point2_right = ordered_points [ id_right ]

# use equations (4) and (5)
  x_left = intersect of y and line( point1_left, point2_left)
  x_right = intersect of y and line( point1_right,
point2_right)

  index.append ([ x_left , y, 1] T .. [ x_right , y, 1] T )
map = M x index
id_x ' = map[0]/ map[ 2]
id_y ' = map[1]/ map[ 2]
id_x = first row of index
id_y = second row of index
m_ transformed [ id_y ', id_x '] = g[ id_y , id_x ]
Outputs: m_transformed
```

2.5 Another way detect area instead of scanline method

Make a rectangular boundary of quadrangle from the picture, that is, by determining value of (x_{min}, y_{min}) and (x_{max}, y_{max}). Map each point on the rectangle. If the map result is outside of the desired area, then we not used it, if the result is in the desired area, map that pixel.

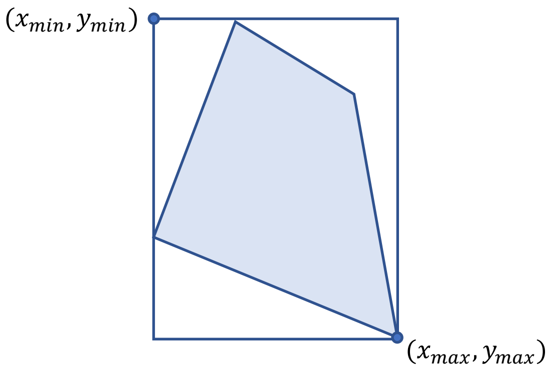


Fig. 4. Rectangular boundary of quadrangle

2.6 Holes detection from picture results by perspective transformation

One way to detect the position of the blank pixels is to set value -1 in one of the matrix layers transformations early. In that way, after every point is mapped, the points that are still blank on matrix transformation will value -1 on one of the layers. Next to determine any points with holes, we only searching for value -1 on the layer that we have specified earlier. This happens because the normal value of pixels only in the integer value from 0 to 255. Figure 5 ilustrate this.

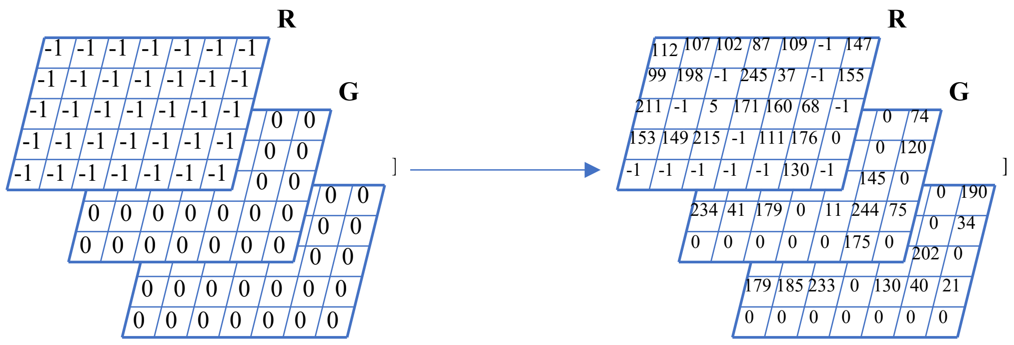


Fig. 5. An example detection hole is to put -1 on one of the matrix layers' transformations beginning

2.7 Interpolation score hollow pixels

After hollow pixels are detected, next we conducted filling in the holes. The holes will be fill in according to the value surrounding pixels. It can be done using measure of central tendency. In mathematics, there are three ways, namely the mean, median, and mode [7]. Entry value around the holes (empty entries) can be seen in Figure 6.

$M(y-1,x-1)$	$M(y-1,x)$	$M(y-1,x+1)$
$M(y,x-1)$	$\mathbf{-1}$	$M(y,x+1)$
$M(y+1,x-1)$	$M(y+1,x)$	$M(y+1,x+1)$

Fig. 6. Illustration from a coordinate neighbor from “hole” pixels

Take note that the hole is not always in the middle. If the position is vacant on the edge, then our defined score pixels = mean, median, or mode of score pixels not empty all around the course. In practice, if $x-1<0$ then $x=0$. If $x+1>$ width then $x=$ width. If $y-1<0$ then $y=0$. If $y+1>$ height then $y=$ height. We only use the active pixels, that is not “hole” pixel. Table 1 illustrates some of the conditions of a “hole” pixel.

Table 1 . Example determination value of “hole” pixels by viewing value surrounding pixels that are not “hole”. The gray color is the hole that will fill the value, -1 means position the is the hole

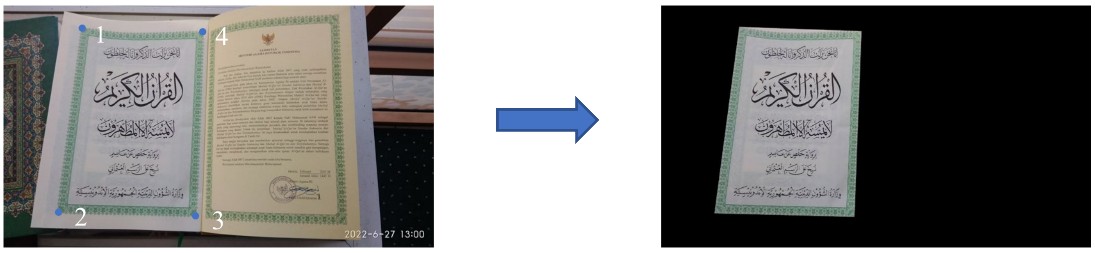
hole position	$\begin{array}{\|c\|c\|}\hline -1 & 123 \\ \hline 20 & 0 \\ \hline\end{array}$ $px_{active} = \{0, 20, 123\}$	$\begin{array}{\|c\|c\|c\|}\hline 123 & -1 & -1 \\ \hline 8 & 123 & -1 \\ \hline\end{array}$ $px_{active} = \{8, 123, 123\}$	$\begin{array}{\|c\|c\|}\hline 6 & -1 \\ \hline 205 & -1 \\ \hline\end{array}$ $px_{active} = \{6,205\}$
average	$px = \frac{0 + 20 + 123}{3} \approx 48$	$px = \frac{123 + 8 + 123}{3} \approx 85$	$px = \frac{6 + 205}{2} \approx 106$
median	$px = 20$	$px = 123$	$px = \frac{6 + 205}{2} \approx 106$
mode	$px = 0$	$px = 123$	$px = 6$

3 Results and Discussion

In this session, we will discuss the results we get from this research. We will start with an example of each process.

3.1 Scanline

Figure 7 illustrates the order fourth point corner and the result. On the picture adjacent top, the order starts at the point adjacent left up, then turn opposite direction clockwise, while picture adjacent bottom, the order point corner starting next door left down, then unidirectional clockwork. Second picture the by appropriate detected by the scanline method. This can be seen from the resulting image on the side right of each image.



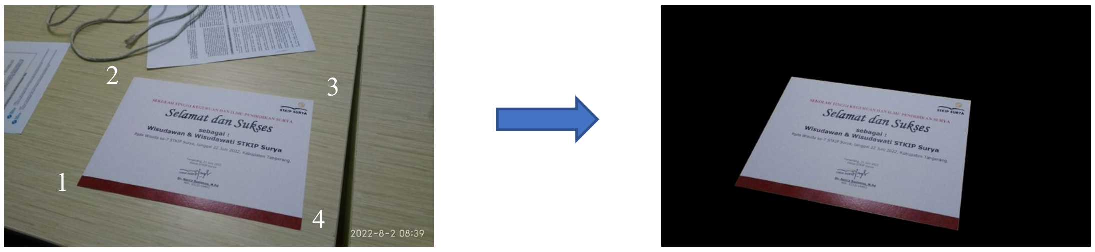
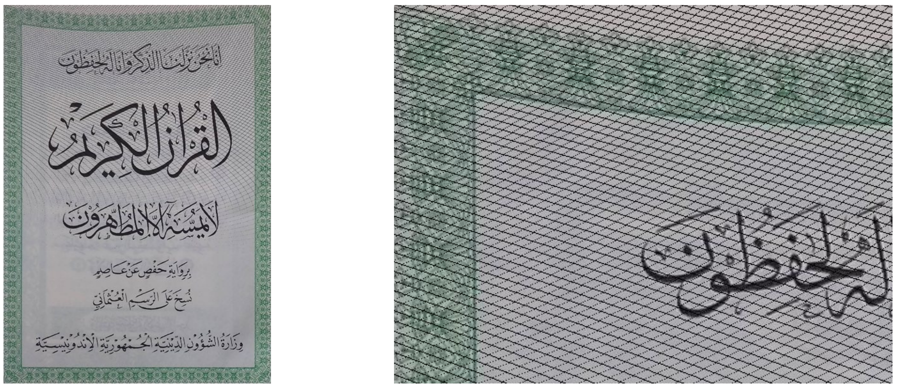


Fig. 7. The scanline method worked access only in the area bounded by 4 points corner side four however order from the point to the corner

3.2 Mapping the quadrangle to rectangle using perspective transformation

After mapping all the points in the area side four to a rectangle, it will be the "hollow" points in the rectangle, blank pixels. These pixels are filled by doing interpolation from surrounding points. We can see the detail in the Figure 8.



(a) (b)

FIGURE 8. (a) picture results transformation perspective from Figure 7. **(b)** enlarged image (a). Seen many "holes" in the picture

3.3 Fill in the blank pixel using mean, median, and mode

Difference results in charging holes with mean, median, and mode can be seen in Figure 9.

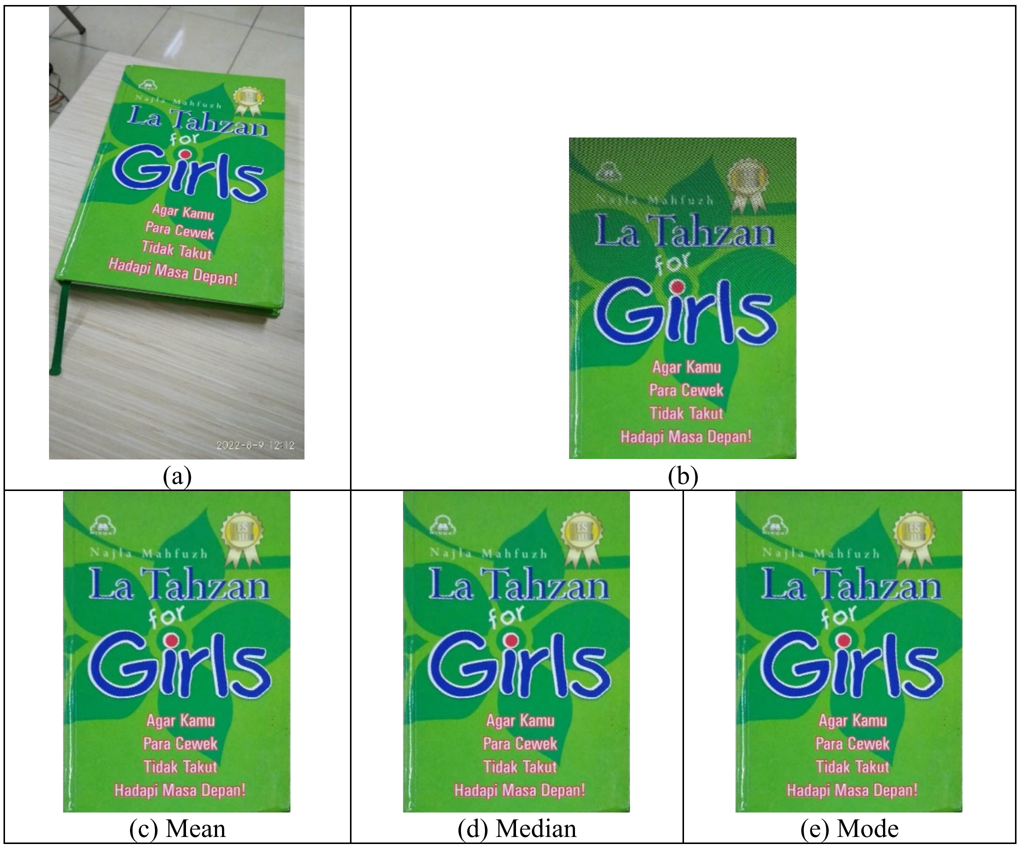


FIGURE 9. (a) picture of the original document from the camera, (b) pictures document after conducted transformation perspective however still perforated, (c, d, e) picture document after filling the hole using the mean, median, and mode method.

4 Conclusion

Distorted document images could be fixed by using a transformation perspective. In the process, the application forms materials simple math used here, like gradient, equation of line, multiplication matrix, solution system linear equations, and statistical data centering (mean, median, mode).

Acknowledgments. We would like to acknowledge the support of Ristekdikti for the grant.

References

[1] N. Basu and S. K. Bandyopadhyay, "Automatic perspective rectification of documents photographed with a camera," *IJAR*, vol. 2, no. 3, pp. 705–710, 2016.

[2] A. Geetha Kiran and S. Murali, "Automatic rectification of perspective distortion from a single image using plane homography," *J. Comput. Sci. Appl*, vol. 3, no. 5, pp. 47–58, 2013.

[3] K. Wang, B. Fang, J. Qian, S. Yang, X. Zhou, and J. Zhou, "Perspective transformation

data augmentation for object detection," *IEEE Access*, vol. 8, pp. 4935–4943, 2019.

[4] Z. Zhang and L.-W. He, "Whiteboard scanning and image enhancement," *Digit. Signal Process.*, vol. 17, no. 2, pp. 414–432, 2007.

[5] K. Kallio, "Scanline edge-flag algorithm for antialiasing," 2007.

[6] D. E. Varberg, E. J. Purcell, and S. E. Rigdon, *Calculus*. Pearson Educación, 2007.

[7] A. S. Pratikno, A. A. Prastiwi, and S. Ramahwati, "Ukuran Pemusatan Rata-rata," 2022.