

Largest inscribed rectangle or square

1 Introduction

The function `LargestRectangle` will find the largest inscribed rectangle of any orientation and in any arbitrary image. This means the rectangle may be tilted and some obstacles may be inside the image as shown in Fig. 8.

Also included are following functions:

- **LargestRectangleParFor**, operates in parallel processing mode (you need the parallel toolbox), delivers the same results as `LargestRectangle` in a shorter time.
- **LargestSquare** is the same as `LargestRectangle`, but the rectangle sides are equal.
- **LargestSquareParFor**, as `LargestRectangleParFor`, but for squares.

2 Instruction

function LRout=LargestRectangle (image,varargin)

Function to find the largest inscribed rectangle in an arbitrary shape with multiple holes.

INPUT:

The input has a minimum of one entry and maximum of 6 entries in following order:

image: Image, RGB, grey or BW. By preference BW.

E.g.: `image=imread(C:\MyImage.tif);`

RotationStep: Default: 5°

Rotation Step in degrees. In order to find tilted rectangles, the image is rotated.

Range: $0 < \text{RotationStep} \leq 90$.

If $\text{RotationStep} > (\text{LastAngle} - \text{FirstAngle})$ then the image is rotated once.

iterate: Default: 1 (with iteration)

If 0 then no iteration, if 1 then iteration.

No iteration if $(\text{FirstAngle} == \text{LastAngle})$

Iterate refines the rotation steps with the goal to find the largest possible rectangle,

In some cases, it might lock on another smaller local maximum.

FirstAngle: Default: 0°, first angle for detection of rectangle with any orientation.

First angle: $0 \leq \text{FirstAngle} < 90$

LastAngle: Default: 89.9999°, last angle for detection of rectangle with any orientation.

Last angle: $\text{FirstAngle} \leq \text{LastAngle} \leq 90$

Graphic: Default: 1 (Plot graphic), see Fig. 8.

OUTPUT

LRout:

1st row: Area of the largest rectangle in px, Rotation angle in degrees counterclockwise

2nd row: x,y of top corner of the largest rectangle

3rd row: x,y of right corner of the largest rectangle

4th row: x,y of bottom corner of the largest rectangle

5th row: x,y of left corner of the largest rectangle

EXAMPLES:

`LRout=LargestRectangle(myImage)` Run with default values

`LRout=LargestRectangle(myImage,0.1,0)` Run with small rotation steps,
no iteration.

`LRout=LargestRectangle(myImage,0,0,0,0)` For axis parallel rectangle

`LRout=LargestRectangle(myImage,5,0,10,20,0),`

Example for rotation step=5, no iteration, starting at 10°, ending at 20°, no graphic.

REMARK:

Any hole (black pixel), even only one pixel wide, will not be inside the largest rectangle

3 Limits

If the image contains only 1x1 px areas, LargestRectangleParFor or LargestSquareParFor will cause an error.

4 Coordinate system

For image processing Matlab uses a coordinate system as shown in Fig. 1:

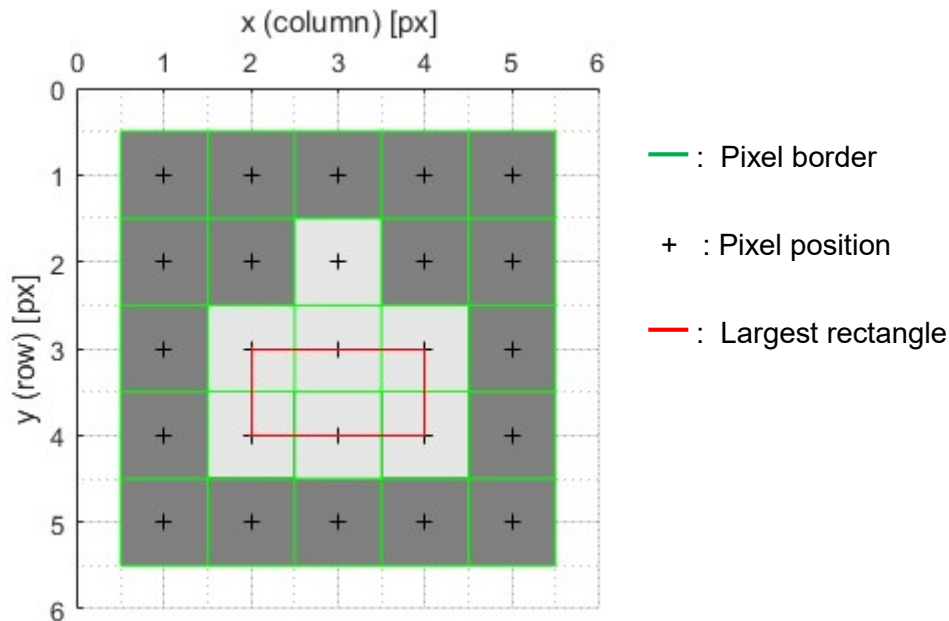


Fig. 1: Coordinate system

The origin of the coordinate system is outside the image at the top left corner.

The first pixel of the image is at position $x,y=1,1$.

The area of the largest rectangle is 6 px. Please observe that the sides of the red rectangle are only 1x2 px. But the surface is $(1+1) \times (2+1)$ px.

5 Method

The image is rotated in order to detect arbitrary oriented rectangles.

Each right white pixel that is next to a left black pixel is searched for the maximum sized rectangle. If its size is larger than the previous largest rectangle, it is the new largest rectangle LR. If all left border pixels are checked, the image is rotated by the rotation step given by the user. Again all left border pixels are checked for the largest rectangle. After all rotations up to nearly 90° are done, the largest rectangle and its orientation is the result.

6 Iteration

Since we do not know how much the rectangle is tilted, we need to refine the angles of rotation.

To refine the result by nested intervals will not work, since the function $LR(\text{angle})$ is a discontinuous function with many local peaks around the absolute maximum value.

Therefore, we look in rotation ranges by continuously decreasing the rotation step.

Very rough explanation of iteration

On one side we want to get the largest rectangle on the other side we want to minimize the time consumed by the iteration steps. Therefore, many exceptions, that are not mentioned here, are part of the code.

With iteration on, the code forces to have initially at least 10 results for rectangle sizes at different angles.

Now iteration starts. The surroundings of the 7 largest rectangles are checked for even larger rectangles. The rotation step is divided by 2 and again the 7 largest peaks are checked. This repeated until a rotation step of 0.5° is reached. Now only the surrounding of the largest peak is checked until we reach 0.01° .

6.1 Example

We want to find the largest rectangle within the image of Fig. 2.

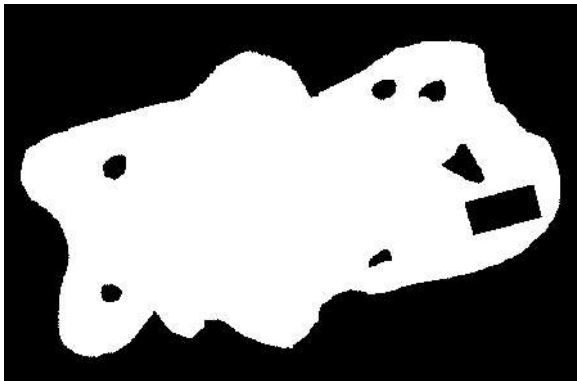


Fig. 2 Image to be analyzed

If we would rotate this image by very small rotation steps, we get the function of the largest rectangles versus the corresponding angles, see Fig. 3. In this case we have 4 major peaks at a), b), c) and d). The largest rectangle is at c).

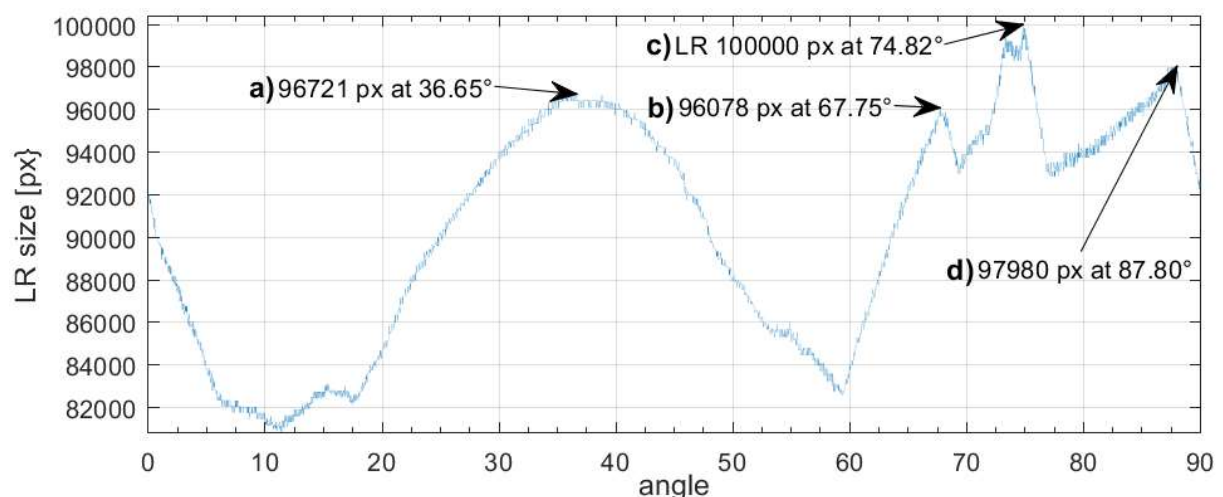


Fig. 3 Largest rectangle versus angle for the image of Fig. 2 rotated in small steps

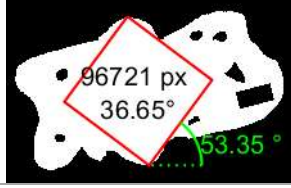
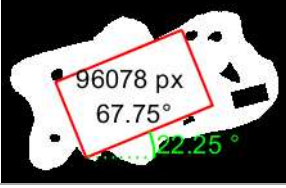
			
Fig. 4 a) 96721 px at 36.65°	Fig. 5 b) 96078 px at 67.75°	Fig. 6 c) LR, 100000 px at 74.82°	Fig. 7 d) 97980 px at 87.80°

Fig. 4 to Fig. 7: Rectangles at different peaks.

Let us assume we set a rotation step of 6°. The 7 largest rectangles before iteration are at 30,36,42,66,72,78 and 84°. The rotation step is divided by 2 (gives 3°) and the procedure is repeated until we reach a rotation step of 0.5°.

Now we concentrate only on the largest peak. A threshold (limit) is defined by the size of the actual largest rectangle:

$$\text{limit} = LR \left(1 - 0.05 * 0.85^{\log_2 \left(\frac{\text{RotationStep}}{15} \right)} \right) \quad (1)$$

Around the actual maximum the angles are searched that have a rectangle value above the limit (97524 px).

The iteration continues by setting the rotation step to one-half of the previous step until a rotation step of 0.01° is reached.

The final result for LR is 100000 px at 74.82° , see Fig. 8.

Again, this only a rough explanation, since more constrains are applied.

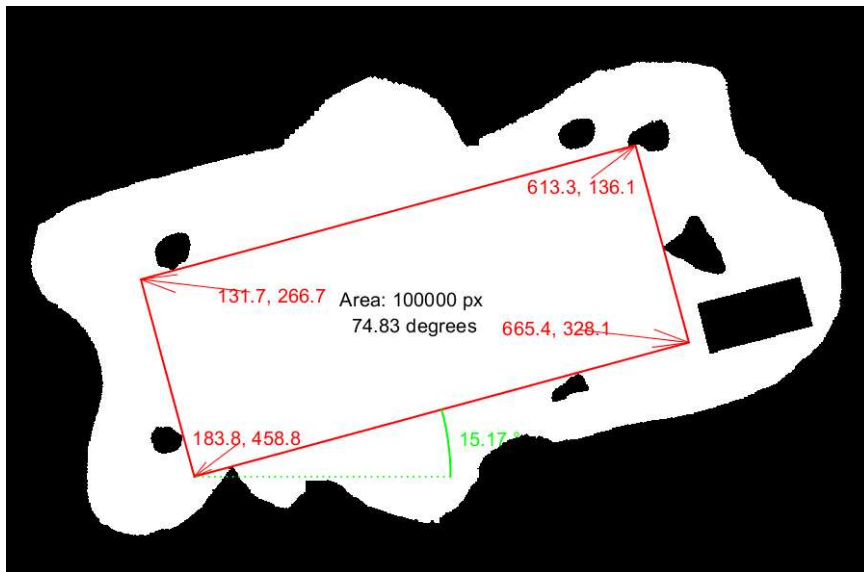
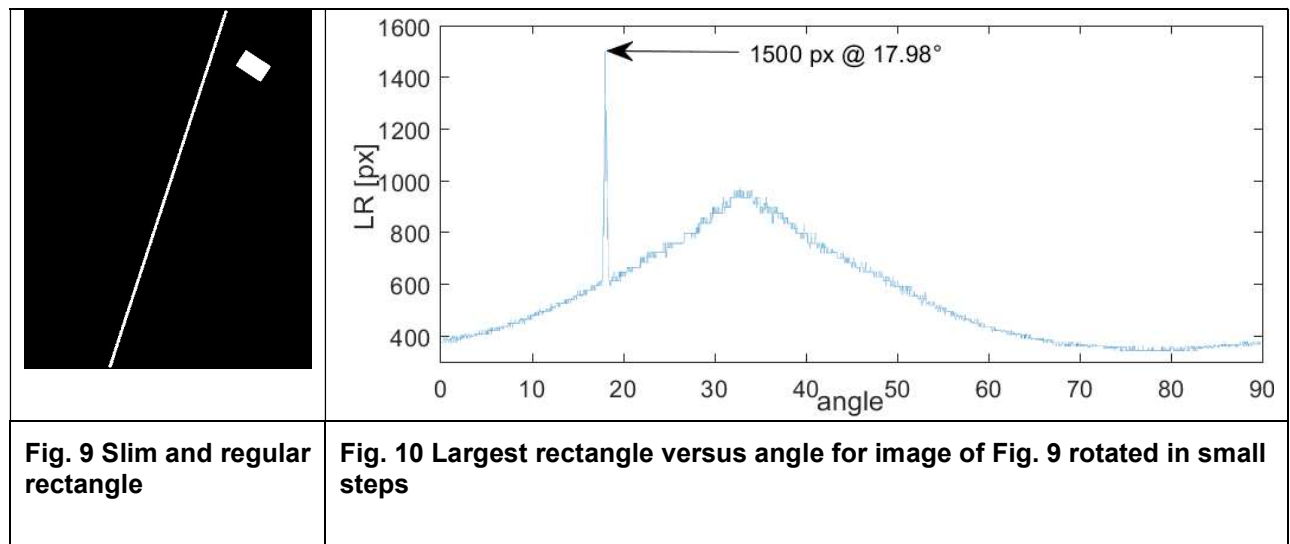


Fig. 8 Correct largest rectangle for the image acc. Fig. 2

6.2 Second experiment

Here we have a slim rectangle that contains 1500 px and a regular rectangle that has an area of 960 px.

It is difficult to detect the slim rectangle that is tilted by 18°. You can find it either by luck (setting the rotation step to hit 18°: e.g. 1°, 2°, 3°, 6°, 9° or 18°) or by setting a rotation step smaller than 0.015° or with iteration to 0.365°.



Therefore, if you expect a slim rectangle, you should apply small rotation steps.

6.3 Conclusion

Iteration refines the result, but does not assure to find the largest rectangle.

7 Accuracy

7.1 Errors due to rotation pixelation

The accuracy mainly depends on the rotation. Due to the digital image you may lose (or sometimes gain) a pixel on one or two sides of the rectangle. That is explained in detail in chapter 11.1 “Rotating errors”.

7.2 Errors due to rotation resolution

Another source for errors is the resolution of the rotation steps.

In Fig. 11 we see how much the error depends on the rotation offset and the shape of the rectangle. E.g. for a rectangle with equal sides, a square, lowest line in the graphic, we get a maximal area error of 3.4% if we miss the correct orientation by 1° .

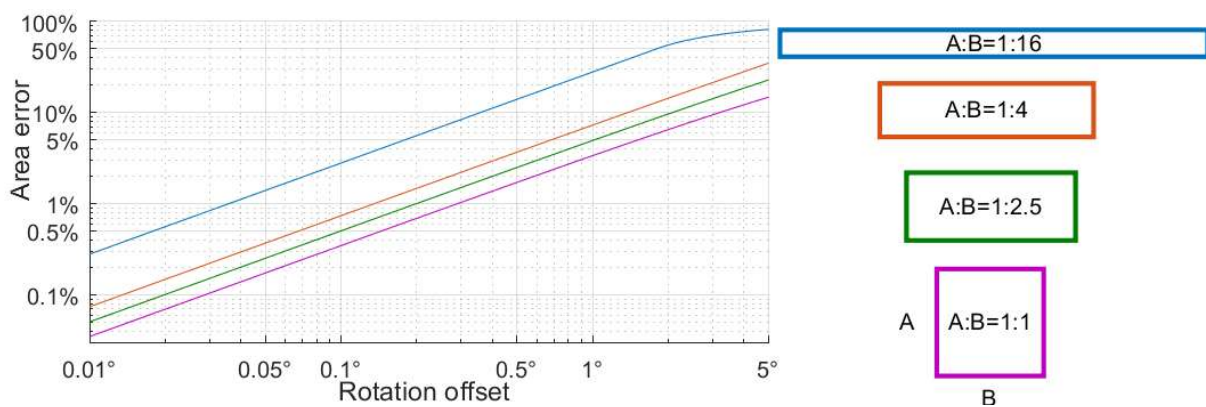


Fig. 11 Error in area due to rotation offset and shape of rectangle

For a rectangle, where the long side is 16 times larger than the short side, the error with 1° offset is 28%., see Fig. 12 and top line in Fig. 11.

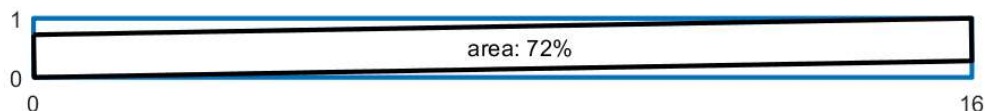


Fig. 12 Max. inscribed rectangle at 1° in a 1:16 outer rectangle

With rotation steps the error is smaller.

In Fig. 13 we evaluated the maximal error you can get with a 200x500 px rectangle (upper curve). The lower curve is for the image according Fig. 28. For all rotation steps with any orientation of the image we search the worst constellation and built the curve with it.

The largest rotation offset for a geometric rectangle is then one half rotation step. E.g. for a 1:2.5 rectangle we get at an offset of 5° an error of 23%, see Fig. 11. With a rotation step of 10° and also a 1:2.5 rectangle we get the same error, see Fig. 13.

You would get the upper curve in Fig. 13 if you have only a 200x500 px rectangle in your image and no digitalization effects due to rotation.

For largest rectangles inside an arbitrary image the situation is different. On one hand the errors may be less in real images since the boundaries may be loose and other rectangles may be found, see lower curve in Fig. 13.

More details and how to calculate inscribed geometric rectangles is explained in chapter 11.2.

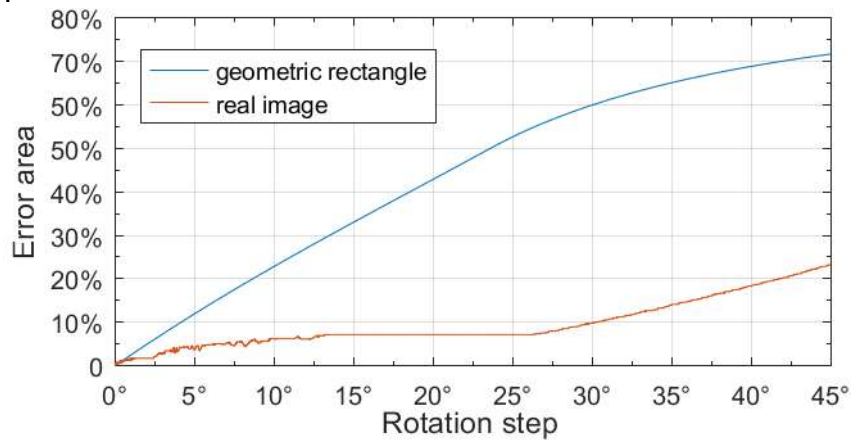


Fig. 13 Error curve of a 200x500 px rectangle and image of Fig. 28 with a 200x500 px rectangle as max. area rectangle.

7.3 Conclusion

Due to digital rotation you get pixelation errors.
Small rotation steps are essential with slim rectangles.

Remark:

With the BW-image in double precision, the rotation is more accurate and you get slightly (~1%) better results. The processing time is then 3 times higher. Remove in 'LargestRectangle.m' at line 60 the word 'logical'.

8 About imrotate

The rotation of a digital image is never correct.

8.1 Rotation with different border sizes

The `imrotate` function of Matlab produces unexpected results as following example describes.

As first image we have a white rectangle of 3x51 px surrounded by a black border of 1 px. The second image has the same white rectangle surrounded by a black border of 2 px. The rotation center is in both cases at the same place in the white rectangle. If we rotate both images by the same angle, we get different results, see Fig. 14.

The detected largest inscribed rectangles are in this case different (153 px vs. 102 px). Approximately 60% of the rotations with 9001 different angles deliver different LRs. But the averages of the LR errors (17%) are nearly the same.

The test is done with Matlab R2016a. If you use another version, you may get other results.

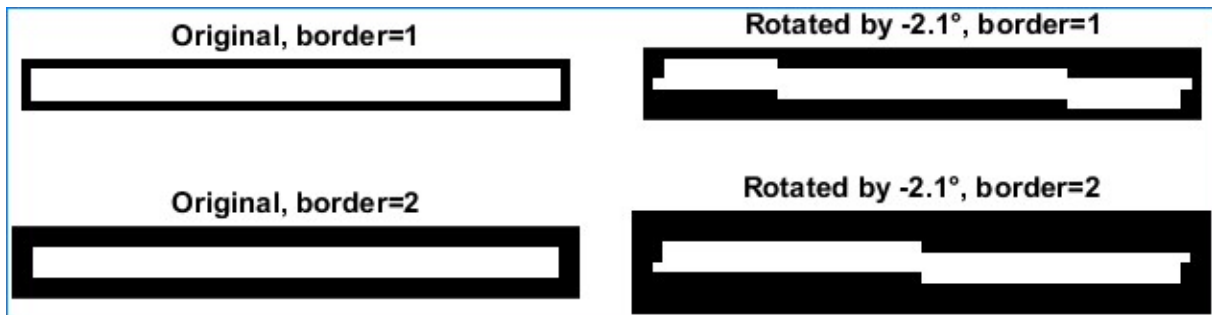


Fig. 14 Rotation of the same white rectangle with different border width

The problem is not associated with small borders. Fig. 15 shows the result with larger borders. Also here the rotation result is different.

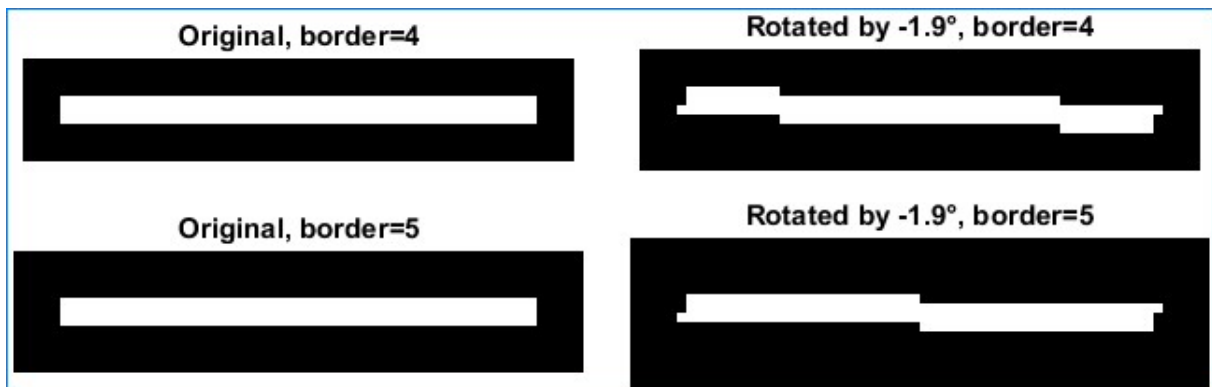


Fig. 15 Rotation of the same white rectangle with larger border width

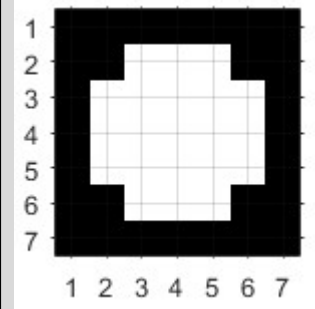
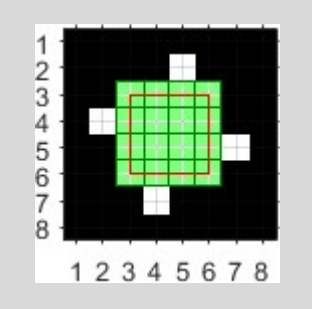
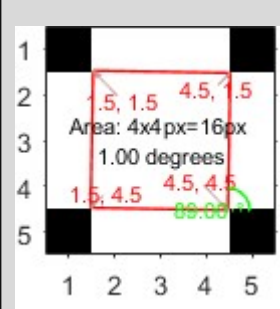
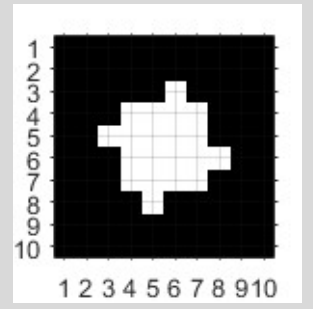
8.2 Rotation of small images

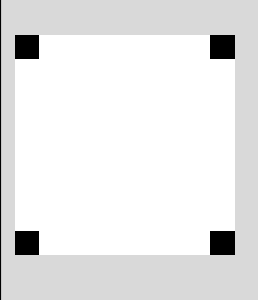
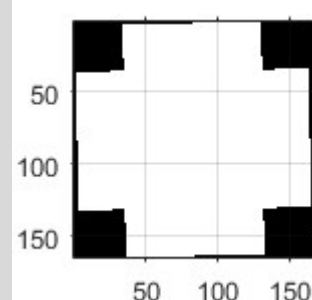
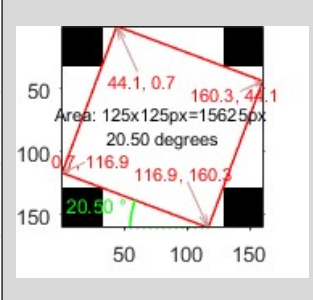
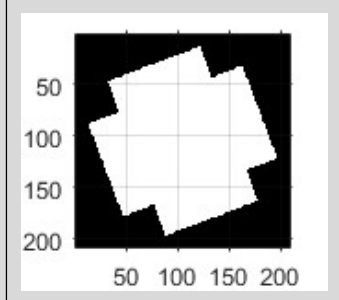
While rotating small images, you may get unexpected large relative errors. As an example see Fig. 16 to **Fig. 23**. Fig. 16 is a 5x5 px image with 1 px black corners. With a rotation of 1°, **Fig. 17**, we get the first largest square, **Fig. 18**, that has 16 px. We would also get the 16 px with any further rotation between 1° to 89°, e.g. 20,5°, see **Fig. 19**. **Fig. 17** and **Fig. 19**

are exactly the same except an added black border in **Fig. 19**. Obviously, the result with 1° rotation does not look good.

In order to get better result we can magnify the image e.g. 32 times, **Fig. 20**. Here we get the best result, **Fig. 22**, at a rotation of 20.5° , **Fig. 23**. The area is 15625 px. Surprisingly, if we calculate this area down from the 160×160 px image to the 5×5 px image, the area is only 15.3 px ($15625/32^2$).

That is explained by **Fig. 17**. Here we see the pixels of the rectangle (green) are fully inside the white area of the rotated image. The red rectangle is on the center of the border pixels. If we rotate this rectangle back to the original image the green pixels will cover the black pixels of **Fig. 18**. Therefore the result is not accurate. The result of the enlarged image, **Fig. 22**, is very, but also not totally, accurate.

			
Fig. 16 Original, 5x5 px, with 1px border	Fig. 17 Rotated by 1°	Fig. 18 Result	Fig. 19 Rotated by 20.5°

			
Fig. 20 Original, 160x160 px, no border, dimension of black squares are 32x32 px	Fig. 21 Rotated by 1°	Fig. 22 Result	Fig. 23 Rotated by 20.5°

8.3 Rotation with different interpolations

Imrotate offers 3 standard interpolation methods as shown in the first column of Table 1.

In this experiment we evaluate how the different interpolation behave when rotating the images of Fig. 24 and Fig. 25 The images are rotated 9001 times with 0.01° steps.



	 Fig. 24 size: 365 x 775		 Fig. 25 size: 1221 x 4320 px	
Interpolation	Total rotation time	Average absolute error of LR	Total rotation time	Average absolute error of LR
Nearest-neighbor	67 s (100%)	0.46 %	298 s (100%)	0.29 %
Bilinear	69 s (103%)	0.32 %	346 s (115%)	0.15 %
Bicubic	90 s (134%)	0.36 %	740.s.(247%)	0.20 %

Table 1 Rotation with different interpolations

As you can see from the table, the bilinear interpolation is the best method in this context. Bilinear has the lowest errors. The bilinear interpolation takes a little more time than nearest-neighbor method.

Bicubic smooth the image too much and removes or adds more pixels.

8.4 Conclusion

Bilinear is the best interpolation for imrotate in this context.

9 Process time

Table 2 shows the process time for two images. Of course the process time is depending on the PC you use.

The first result row is with the default values, but without graphic output. The second result line is with many rotations. In all cases the expected largest rectangle is found.

The largest process time (39% up to 61%) is used for the rotation.

Using iteration, the gain with parallel processing is roughly 1.5 times with 6 workers. The gain in speed is that low, because during iteration the parfor loop is restarted several times.

With small rotation steps we gain 3 to 4 times in speed with parallel processing.



	 Fig. 26: 554 x 844 px Amount of border pixel: 2989				 Fig. 27: 1221 x 4320 px Amount of border pixel: 40864			
Rotation steps	Num. rotations	Time total	Time rotation	Time total parallel	Num. rotations	Time total	Time rotation	Time total parallel
5° with iteration	120	1.3 s	61%	1 s	66	4.1 s	39%	2.6 s
0.02° no iteration	4500	48 s	54%	12 s	4500	318 s	40%	111s

Table 2 Process times for two images, linear and parallel processing

The process time is mainly depending on

- Image size (influences rotation time)
- Amount of border pixel (influences the time to find the largest rectangle)

9.1 Conclusion

The process time is, depending on your processor, in the lower seconds range.

If time is crucial, you may change the final rotation step for iteration from 0.01° to 0.1° . (Variable: MinRotationStep inside the code) The max. error for the tested images (Fig. 28, Fig. 29 and Fig. 32) is then around 1%. The gain in speed is around 40%.

If you accept a larger error, you may set the rotation step to 9° and MinRotationStep=1. The process time will be reduced to approx. 10%. The max. error in the tested images is then 3%.

Remark:

With the BW-image in double precision, the rotation is more accurate and you get slightly ($\sim 1\%$) better results. The processing time is then 3 times higher. Remove in 'LargestRectangle.m' at line 60 the word 'logical'.

10 Test images

Table 3 is a list of some test images used for this evaluation and their properties.

Description of the columns:

Column 1	Image and its name
Column 2	<ol style="list-style-type: none">1. The first number is determined by the width of the peak out of the graphics from column 6. Then the largest rectangle is found without iteration.2. The number in parenthesis is with Iteration ON. For the test the rotation steps are increased by 0.001° from 0.001° to 89.999°. "Any" means, that any rotation step will result in full success. It is unlikely, but still possible, that special constellations (start angle and rotation step) lead to not optimal results.3. The third number is the max. rotation angle for 1% error, without iteration.4. The 4th number is the max. rotation angle for 5% error, without iteration.
Column 3	Number of false detection out of 9999 rotation steps between 0.001° and 9.999° . Iteration is ON.
Column 4	For the total number of rotations you have to add $90/\text{RotationStep}$.
Column 5	The x-axis is for the image rotation angle and the y-axis is for the largest rectangle. E.g. for the image of Fig. 28 the axis parallel (0° rotation) largest rectangle is 94068 px. Since the graphs are vector graphics, you may want to zoom in.

Description of the images;

Fig. 28	Simple image with one dominant peak.
Fig. 29	Complex image with one peak at 74.8° and a flat peak of similar size at 36.7° . The flat peak is hit very often.
Fig. 30	Same as Fig. 29, but the flat peak is only 0.3% smaller than the largest peak. Therefore, it is unlikely to find the largest (narrow) peak by iteration. Nevertheless, the error is than only 0.3%.
Fig. 31	The narrow largest peak will not be detected with rotation steps larger than 0.36° and iteration ON.
Fig. 32	This is a large image with many border points and takes long to be evaluated.


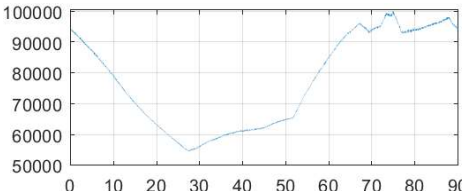
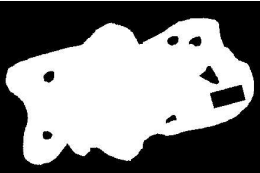
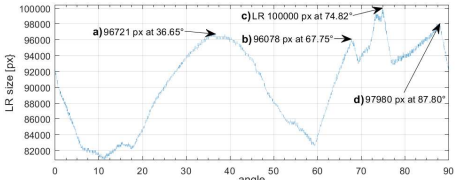
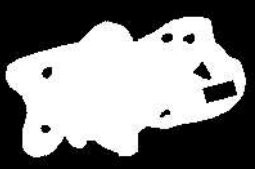
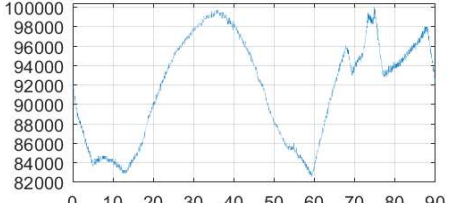
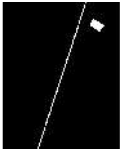
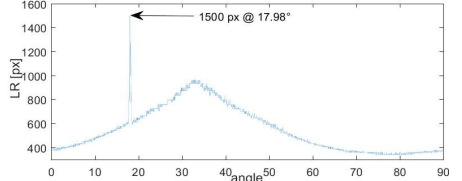

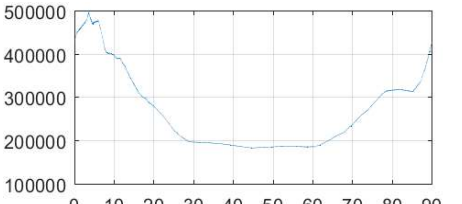
Image name	Max. rotation step for guarantied success, ...	Number of correct detections out of 9999 with iteration	Number iterations	Plot of largest rectangle versus rotation angle
 <p>Fig. 28: 554 x 844 px imTest100000px @74,83°.tif</p>	<p>0.011° (any)</p> <p>1% 0.292° 5% 5.389°</p>	All (100%)	<p>Mean: 93 Max: 134 Min: 28</p>	
 <p>Fig. 29: 554 x 844 px imTest100000px @74,83°b.tif</p>	<p>0.011° (any)</p> <p>1% 0.292° 5% 9.665°</p>	All (100%)	<p>Mean: 96 Max: 140 Min: 28</p>	
 <p>Fig. 30: 554 x 844 px imTest100000px @74,83°d.tif</p>	<p>0.011° (0.067°)</p> <p>1% 2.316° 5% 19.021°</p>	2289 (23%)	<p>Mean: 107 Max: 1150 Min: 40</p>	
 <p>Fig. 31: 480 x 383 px imTest2Needle1500px @17,99°.tif</p>	<p>0.015° (0.365°)</p> <p>1% 0.015° 5% 0.015°</p>	1623 (16%)	<p>Mean: 100 Max: 162 Min: 18</p>	
 <p>Fig. 32: 1221 x 4320 px Buffalo496836px @3,50°.tif</p>	<p>0.010° (any)</p> <p>1%: 0.231° 5% 1.752°</p>	All (100%)	<p>Mean: 55 Max: 99 Min: 20</p>	

Table 3 Some test images and their properties

11 Appendix on accuracy

11.1 Rotating errors

Rotating a digital image, especially a binary image, is never accurate.

In order to evaluate the accuracy, we take an image, rotate it a small step, take that image as image to test and determine the largest rectangle. In the best case the calculated size of the largest rectangle should be independent of the rotation.

11.1.1 Example with an arbitrary image

The largest inscribed rectangle of the image in Fig. 33 has a nominal size of 200x500 px.

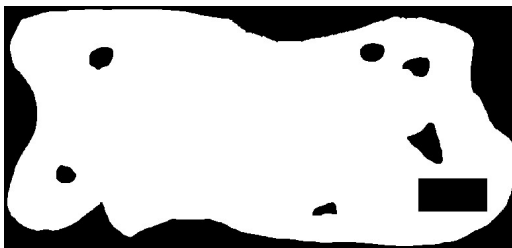


Fig. 33 Arbitrary image with holes, 365 x 775 px

This image is rotated in steps of 0.01° . For example, we rotate this image by -10° and look what LR we get by rotating it back by 10° .

Fig. 34 shows the function LR(angle). The largest absolute errors are less than 0.7% with an average of 0.32%. The expected LR size is 100000 px.

loss [px] on one small side	loss [px] on one long side	rectangle size [px]	error [%] (absolute)	occurrence [%]
-1	0	201 x 500 = 100500	0.5	0.1
-1	1	201 x 499 = 100299	0.3	0.02
0	-1	200 x 501 = 100200	0.2	0.6
0	0	200 x 500 = 100000	0.0	28.8
0	1	200 x 499 = 99800	0.2	18.9
1	-1	199 x 501 = 99699	0.3	0.4
1	0	199 x 500 = 99500	0.5	28.0
1	1	199 x 499 = 99301	0.7	20.3

Table 4 Summary of errors for rotated arbitrary image

Due to rotation inaccuracy, we lose or gain many times 1 px on a short or long side of the rectangle, see 3rd column of Table 4.

The histogram, Fig. 35, that only a few rectangles (0.7%) are larger than expected, 20% are correct and the remaining are smaller than expected.

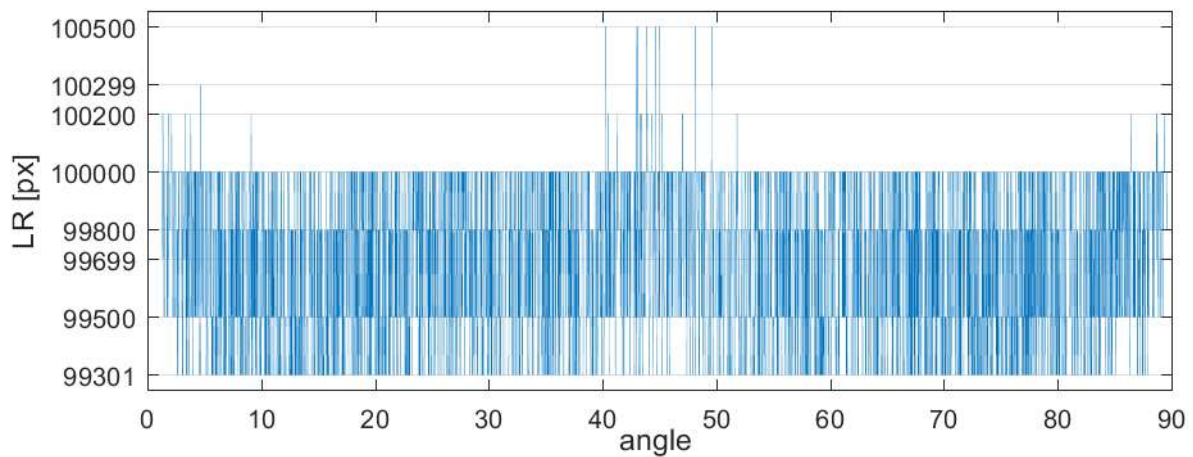


Fig. 34 Largest rectangle versus angle for the rotated arbitrary image

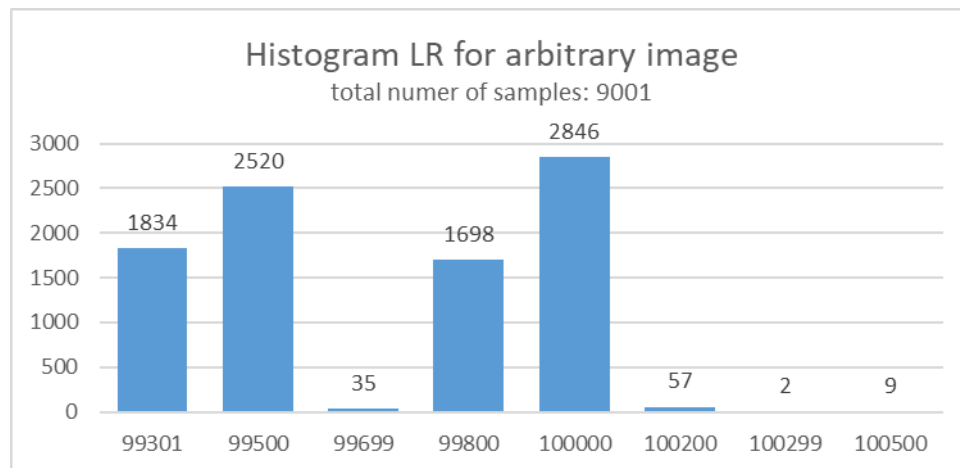


Fig. 35 Histogram of LR(angle) with 9001 rotation angles for arbitrary image

11.1.2 Example with the same arbitrary image but twice the size

For this experiment the image of Fig. 33 is resized by two in respect to the edges. The image is rotated in 0.01° steps and the results for the LRs are divided by 4 and rounded. The average of the absolute errors is 0.22% compared to 0.32% with the original image. The calculation time increased 2.5 times.

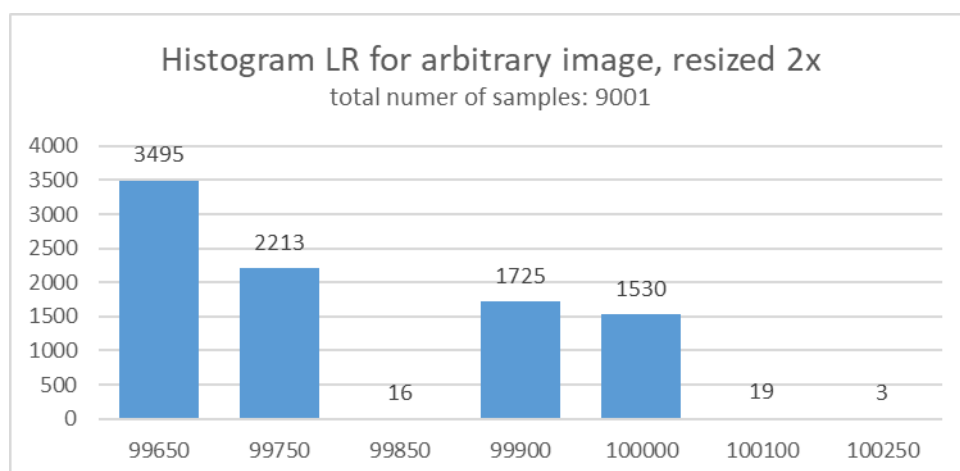


Fig. 36 Histogram for the same image as for Fig. 35 , but twice the size

11.1.3 Example with a regular rectangle

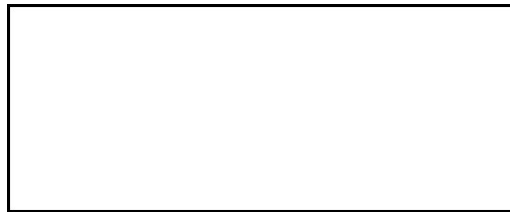


Fig. 37 Regular 200x500 px rectangle

The rectangle with a size of 200x500 px is rotated in 0.01° steps.

Fig. 38 shows the function LR(angle) and Fig. 39 the amount of the obtained different LR.

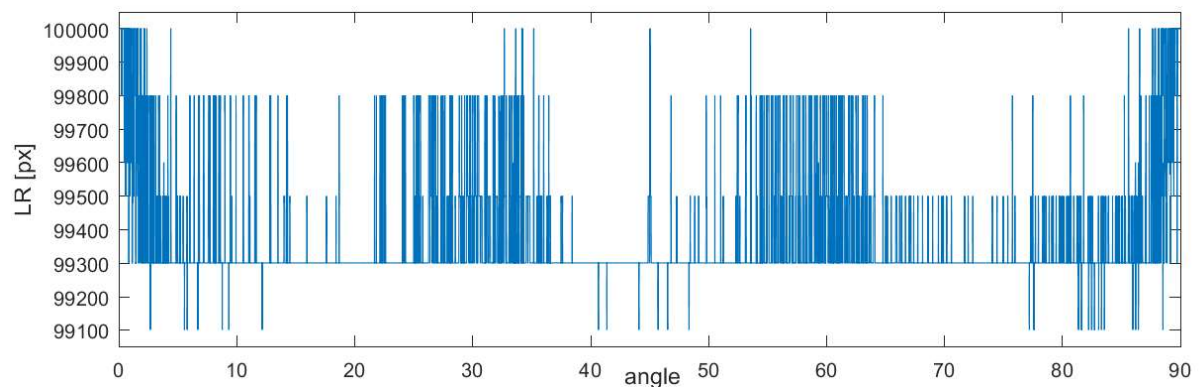


Fig. 38 Largest rectangle versus angle for the rotated 200x500 px rectangle

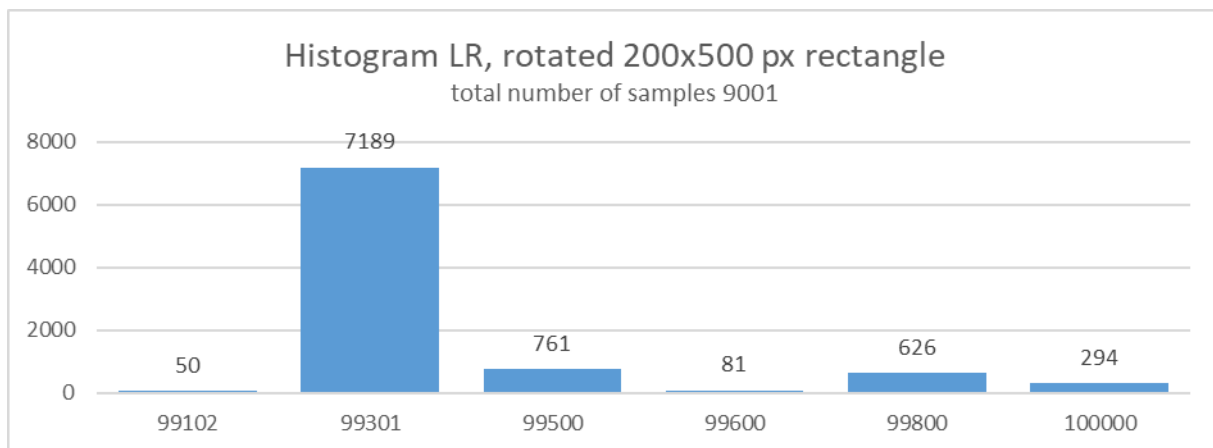


Fig. 39 Histogram of LR(angle) with 9001 rotation angles for regular rectangle

The largest errors are 0.9% with an average of 0.6%, see Table 5.

loss [px] small side	loss [px] long side	rectangle size [px]	error [%]	occurrence [%]
0	0	200 x 500 = 100000	0.0	3.3 %
0	1	200 x 499 = 99800	0.2 %	7.0 %
0	2	200 x 498 = 99600	0.4 %	0.9 %
1	0	199 x 500 = 99500	0.5 %	8.5 %
1	1	199 x 499 = 99301	0.7 %	79.9 %
1	2	199 x 498 = 99102	0.9 %	0.6 %

Table 5 Summary of errors for rotated regular rectangle

Many times we lose 1 px

- on two sides of the rectangle, see histogram, Fig. 39, bin 99301
- on a small side of the rectangle , Fig. 39, bin 99800
- on a long sides of the rectangle , Fig. 39, bin 99500

We lose sometimes also 1 px on the small side and 2 px on the long side of the rectangle, Fig. 39, bin 99102.

We also lose sometimes also 2 px on the long sides of the rectangle, Fig. 39, bin 99600.

If we would count all white pixels in the rotated rectangle the error would be much lower, since more than 50% rectangles have only an error of max ± 1 px, see Fig. 40. This method is not applicable to random images and is only a theoretical evaluation.

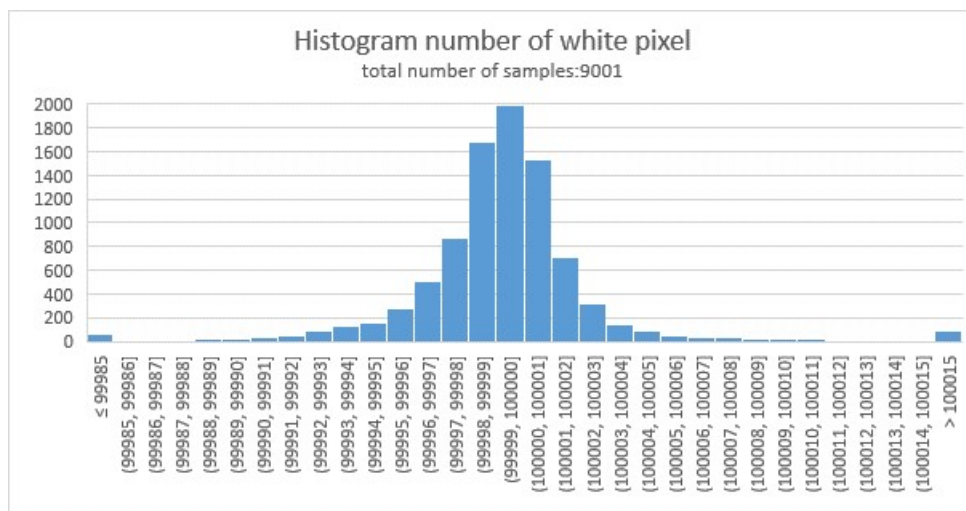


Fig. 40 Theoretical evaluation of a rectangle with rigid borders, number of white pixel during rotation

11.1.4 Extreme example with a very slim rectangle

Fig. 41 White rectangle 3x500 px with a surrounding border of 1 px

Rotating the slim rectangle of Fig. 41, results in maximal errors of 33%. About 55% of the measurements are correct with approx. 1500 px (1491 to 1593 px) for LR and the other half of the rotated images have only approx. 1000 px.

Fig. 42 shows the obtained largest rectangles when the slim rectangle is rotated by 0.01° steps. The values for LR are jumping between close to the correct value (1500 px) and close to 1000 px.

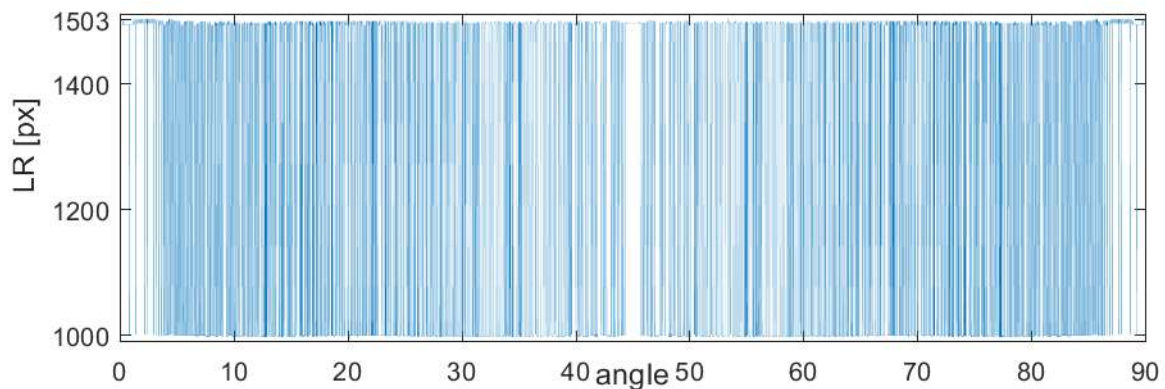


Fig. 42 Largest rectangle versus angle for rotated slim rectangle

11.1.5 Conclusion

Due to the rotation effects, you may lose one pixel on one or both sides of the rectangle. Seldom you may gain a pixel.

11.2 Maximal errors due to rotation deviation

In this section we derivate the formulas for the tilted maximum sized inner rectangle inside another rectangle. The observed range for φ is 0° to 90° .

11.2.1 Case 1: All corners of the inner rectangle touch the outside rectangle

Limits:

$A \leq B$ and $0 \leq \varphi \leq \frac{\arcsin(A/B)}{2}$ or $90^\circ - \frac{\arcsin(A/B)}{2} \leq \varphi \leq 90^\circ$ (for explanation see 11.2.3)

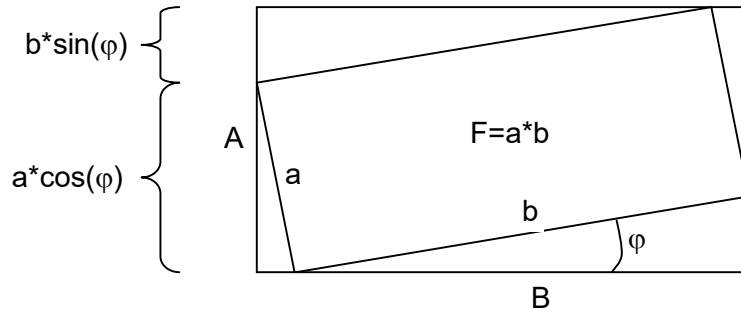


Fig. 43 Case1: all 4 corners of the inscribed rectangle touch the outer rectangle

Short side of outer rectangle: $A = a \cos(\varphi) + b \sin(\varphi)$ (2)

Long side of outer rectangle: $B = a \sin(\varphi) + b \cos(\varphi)$ (3)

Out of (2): $b = \frac{A - a \cos(\varphi)}{\sin(\varphi)}$ (4)

Out of (3): $b = \frac{B - a \sin(\varphi)}{\cos(\varphi)}$ (5)

(4)=(5): $\frac{A - a \cos(\varphi)}{\sin(\varphi)} = \frac{B - a \sin(\varphi)}{\cos(\varphi)}$ (6)

Out of (6): $a = \frac{A \cos(\varphi) - B \sin(\varphi)}{\cos(2\varphi)}$, with $\cos^2(\varphi) - \sin^2(\varphi) = \cos(2\varphi)$ (7)

(7) in (5): $b = \frac{B \cos(\varphi) - A \sin(\varphi)}{\cos(2\varphi)}$ (8)

Area: $F = ab = \frac{2AB - (A^2 + B^2) \sin(2\varphi)}{2 \cos^2(2\varphi)}$ (9)

11.2.2 Case 2: Two or three corners of the inscribed rectangle touch the outer rectangle

Limits: $B \geq A$ and $\frac{\arcsin(A/B)}{2} \leq \varphi \leq 90^\circ - \frac{\arcsin(A/B)}{2}$ (for explanation see 11.2.3)

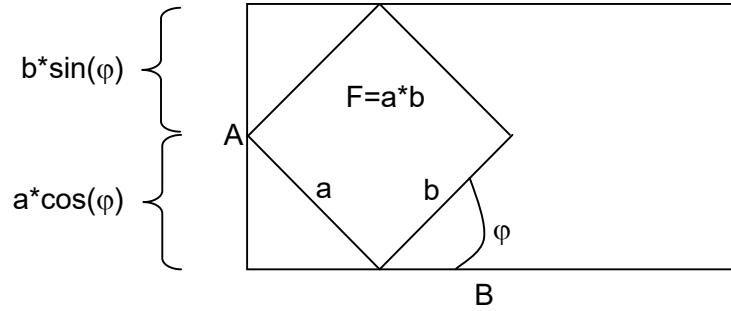


Fig. 44 Case2: only 2 or 3 corners of the inscribed rectangle touch the outer rectangle

With (4):
$$F = ab = \frac{aA - a^2 \cos(\varphi)}{\sin(\varphi)} \quad (10)$$

$$\frac{dF}{da} = \frac{A - 2a \cos(\varphi)}{\sin(\varphi)} \quad (11)$$

F_{max} at $dF/da=0$:
$$a = \frac{A}{2 \cos(\varphi)} \quad (12)$$

Area:
$$F = \frac{A^2}{2 \sin(2\varphi)} \quad (13)$$

11.2.3 Limit for case 1 to case 2

With (4), (5) and (12) we get:
$$\varphi = \frac{\arcsin(A/B)}{2} \quad (14)$$

- The range for case 1 is:

$$0 \leq \varphi \leq \frac{\arcsin(A/B)}{2} \quad \text{and} \quad 90^\circ - \frac{\arcsin(A/B)}{2} \leq \varphi < 90^\circ \quad (15)$$

- The range for case 2 is:

$$\frac{\arcsin(A/B)}{2} \leq \varphi \leq 90^\circ - \frac{\arcsin(A/B)}{2} \quad (16)$$

11.2.4 Resulting graphic

In Fig. 45 you see an area plot of inscribed rectangles while the outer rectangles have different side length and a normalized area of 1. The slimmer the rectangle (lowest line in Fig. 45), the bigger the error with larger deviations in rotation angle.

With (13), (14) and $AB=1$ we get the area $F=0.5$, i.e. above $F=0.5$ we use equation (9), all 4 corners of the inscribed rectangle touch the outer rectangle, below $F=0.5$ we use equation (13). For a square as outer rectangle we always use equation (9).

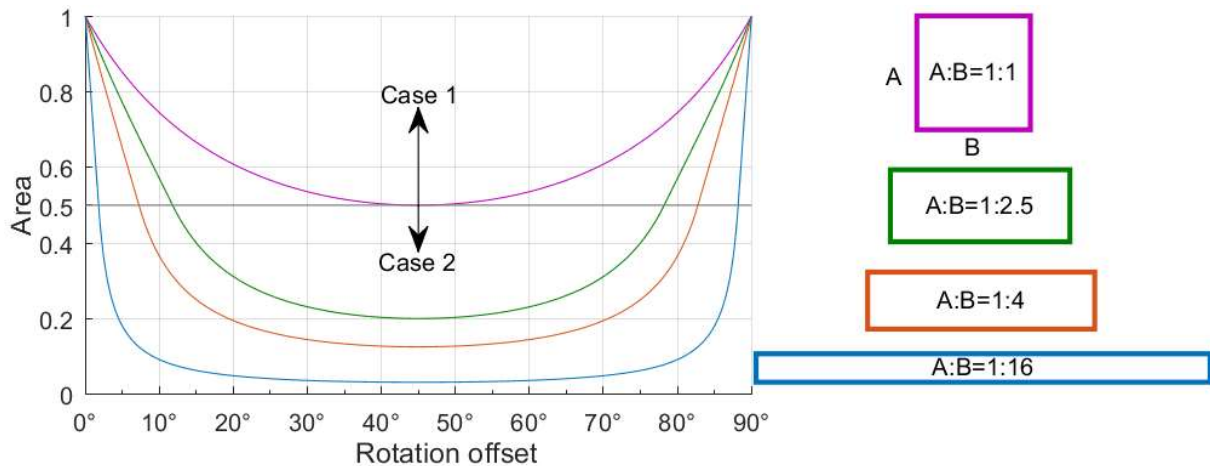


Fig. 45 Normalized area of inscribed max. area rectangle for different shapes of outer rectangle

11.2.5 Errors due to rotation offset

All curves in this chapter are normalized, i.e. the area values are divided by the largest area.

In Fig. 46 we see the calculated area curve according equation (9) and (13) for a 200 x 500 px geometric rectangle (in blue) and the same curve determined by the function 'LargestRectangle.m' (in red) for a 200x500px rectangle inside an image. The pixelation effect is low as you see in the detail figure Fig. 47. These curves show the error if you miss the correct angle. E.g. if you miss the correct angle by 45° you get an error of 80%, i.e. the found rectangle has instead of an expected area of 100000 px an area of 20000px.

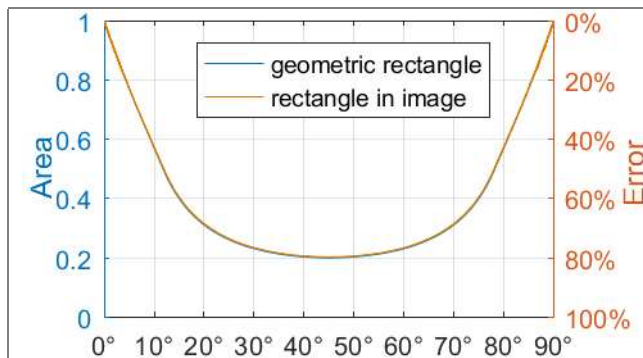


Fig. 46 Area error of 200x500 px rectangle due to rotation offset

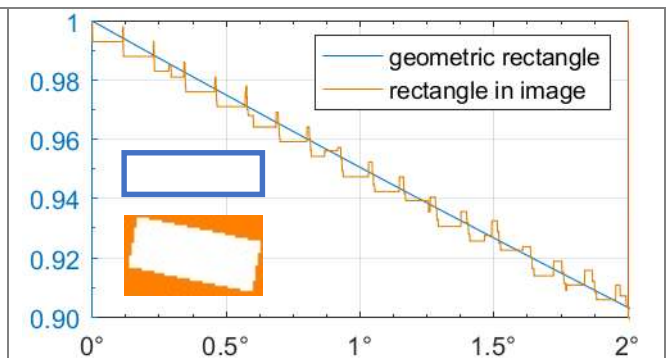


Fig. 47 Detail of Fig. 46

Now we know what the possible maximal error is when we miss the correct angle.

11.2.6 Errors due to rotation step

But we search the maximal rectangle by rotation steps. Therefore, we look for the maximal possible error with any rotation step.

Since we do not know the orientation of the maximal rectangle we have to check all possible image orientations.

Example:

We want to determine the worst area with 10° rotation steps for the image of Fig. 28.

- We determine the rectangle area with the function LargestRectangle at $0^\circ, 10^\circ, 20^\circ, \dots, 90^\circ$ and set the largest area as temporary worst result.
- Now we rotate the image by 0.001° and do so same as previous. If the largest rectangle area of this set is smaller than the previous result, we take this as the new result.
- We repeat this until the image is rotated to 90° and get the final lowest possible largest rectangle for 10° rotation steps

With all rotation steps from 0.001° to 45° we get for the real image the red curve as shown in Fig. 48.

The blue curve is for a geometric rectangle. The maximal area error is at a rotation offset of one half of the rotation angle.

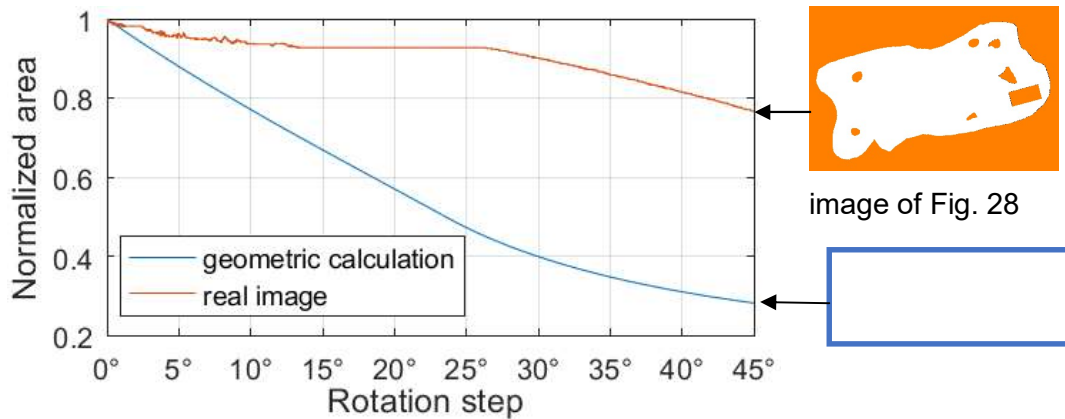


Fig. 48 Worst area vs. Rotation steps for 200x500 px geometric rectangle and a real image