Embedded Vision Design

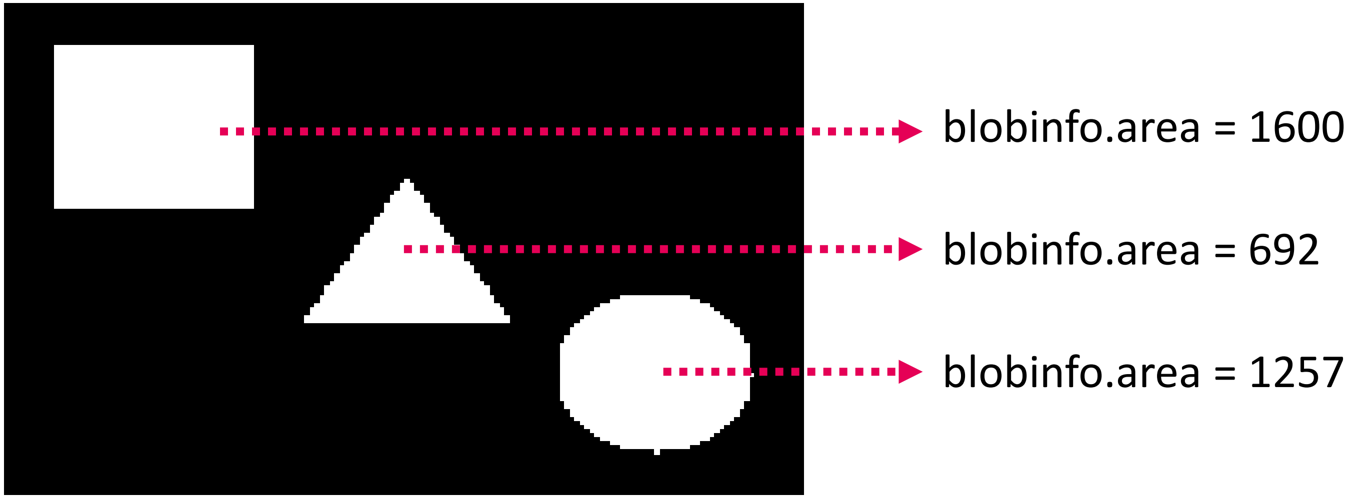
EVD1 - Week 6

Mensuration

By Hugo Arends

Mensuration

- Measurement of features associated with objects
- Used for object classification or data analysis
- All measured values will be collected in a blob information structure



Mensuration

- Measurement of features associated with objects
- Used for object classification or data analysis
- All measured values will be collected in a blob information structure

- Centroid
- Area
- Perimeter
- Circularity
- Hu invariant moments

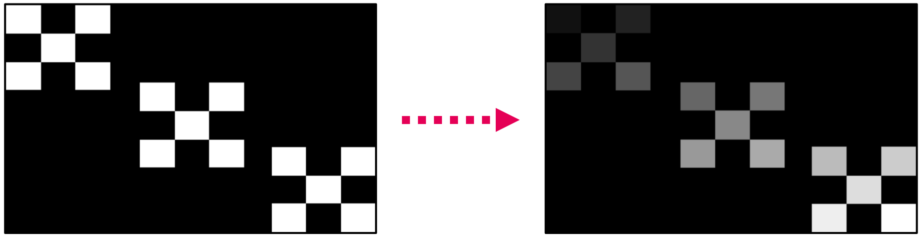
- Label

```
/// Defines features that can be measured of BLOBs
typedef struct
{
    point_t centroid;     ///< The centroid coordinate of the BLOB
    uint32_t area;        ///< The BLOB area in number of pixels
    float perimeter;      ///< The perimeter of the BLOB
    float circularity;    ///< The circularity of the BLOB
    float hu_moments[4]; ///< The first four Hu moments of the BLOB

}blobinfo_t;
```

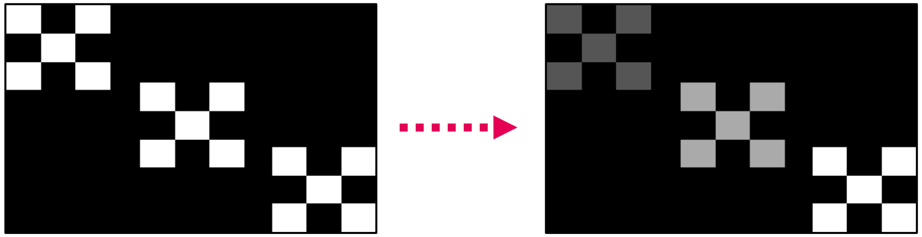
Label

- Counts and labels all BLOBs
- A BLOB is a Binary Linked Object and its pixels are either 4-connected or 8-connected
- Labelling is performed in ascending order from left-top to right-bottom

Label - examples



4-connected : 15 blobs



8-connected : 3 blobs

Label – Iterative algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	
			1	1	1	1	

Source

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					2	2	
			2	2	2	2	

Destination

Label – Iterative algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	
			1	1	1	1	

Label – Iterative algorithm

						1	
					2	3	
		4			5	6	
	7	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order

Label – Iterative algorithm

						1	
					2	**3**	
		4			**5**	**6**	
	7	**8**	**9**	**10**	**11**	**12**	
					13	**14**	
			15	**16**	**17**	**18**	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	3	
		4			5	6	
	7	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			5	6	
	7	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			5	6	
	7	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	6	
	7	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	7	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	8	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	9	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	4	10	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	4	1	11	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	4	1	1	12	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	4	1	1	1	
					13	14	
			15	16	17	18	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	4	1	1	1	
					13	13	
			15	13	13	13	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		4			1	1	
	4	4	1	1	1	1	
					13	13	
			13	13	13	13	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		1			1	1	
	4	1	1	1	1	1	
					13	13	
			13	13	13	13	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					13	13	
			13	13	13	13	

- Assign unique number in ascending order
- While changes
    - Loop entire image and assign lowest neighbor value

Label – Iterative algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					2	2	
			2	2	2	2	

- Assign unique number in ascending order
- While changes
  - Loop entire image and assign lowest neighbor value
- Assign labels in ascending order

Label – Iterative algorithm

Advantage

- Easy implementation

Disadvantage

- Cannot use uint8 images, because this would allow only 255 pixels
- Uses several iterations through the entire image, depending on the object
- Slow

Label – Iterative algorithm

uint32_t **labelIterative**(const image_t ***src**, image_t ***dst,** const eConnected **connected**);

See file **EVDK_Operators\mensuration.c**

```
convertToUint8(cam, src);
thresholdOtsu(src, src, BRIGHTNESS_DARK);
removeBorderBlobsTwoPass(src, thr, CONNECTED_EIGHT, 128);

int32_t blobs = labelIterative(thr, lbl, CONNECTED_EIGHT);
printf("blobs: %d\n", blobs);

scaleFast(src, src);

if(blobs > 0)
{
    scaleFast(lbl, lbl);
    cv::imshow(“lbl", cv_lbl);
}
```

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	
			1	1	1	1	

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	
			1	1	1	1	

- How to handle the edge pixels?

Label – Two-pass algorithm

						1	
					1	**1**	
		1			**1**	**1**	
	1	**1**	**1**	**1**	**1**	**1**	
					1	**1**	
			1	**1**	**1**	**1**	

- How to handle the edge pixels?
- An out-of-bounds check for every pixel would be a waist of time

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	
			1	1	1	1	

- How to handle the edge pixels?
- An out-of-bounds check for every pixel would be a waist of time
- We could handle the corner and border pixels separately

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	

- How to handle the edge pixels?
- An out-of-bounds check for every pixel would be a waist of time
- We could handle the corner and border pixels separately
- We can make our lives easier and discard the borders ☺

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	

Equivalence LUT					

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: **mark this pixel with a new value**

Equivalence LUT					
1					
1					

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1					
1					

HAN_UNIVERSITY OF APPLIED SCIENCES

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1					
1					

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	1	1	1	1	1	1	
					1	1	

Equivalence LUT					
1	2				
1	2				

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: **mark this pixel with a new value**
    - Yes: mark this pixel with the same value

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	1	1	1	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1	2				
1	2				

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	1	1	1	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1	2				
1	2				

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	1	1	1	1	1	
					1	1	

Equivalence LUT					
1	2				
1	2				

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	1	1	1	1	
					1	1	

Equivalence LUT					
1	2				
1	2				

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					1	1	

Equivalence LUT					
1	2				
1	2				

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1	2				
1	2				

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: mark this pixel with the same value
      and
      Update the LUT to the lowest neighbor value

Equivalence LUT					
1	2				
1	1				

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: mark this pixel with the same value
      and
      Update the LUT to the lowest neighbor value
      and
      Search the LUT for other equivalence updates

Equivalence LUT					
1	2				
1	1				

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					1	1	

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Equivalence LUT					
1	2				
1	1				

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					1	1	

Equivalence LUT					
1	2				
1	1				

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					3	1	

Equivalence LUT					
1	2	3			
1	1	3			

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: **mark this pixel with a new value**
    - Yes: mark this pixel with the same value

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					3	3	

Equivalence LUT					
1	2	3			
1	1	3			

- Loop entire image
  - Is it an object pixel?
  - Is a neighbor already marked?
    - No: mark this pixel with a new value
    - Yes: **mark this pixel with the same value**

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					3	3	

Equivalence LUT					
1	2	3			
1	1	3			

- Although the values in the LUT are from low-to-high, they are not necessarily in ascending order

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					3	3	

- Although the values in the LUT are from low-to-high, they are not necessarily in ascending order
- Assign LUT labels in ascending order

Equivalence LUT					
1	2	3			
1	1	2			

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	2	1	1	1	
					3	3	

- Loop entire image
- Use lookup table to assign correct label

Equivalence LUT					
1	2	3			
1	1	2			

Label – Two-pass algorithm

						1	
					1	1	
		1			1	1	
	1	1	1	1	1	1	
					2	2	

- Loop entire image
- Use lookup table to assign correct label

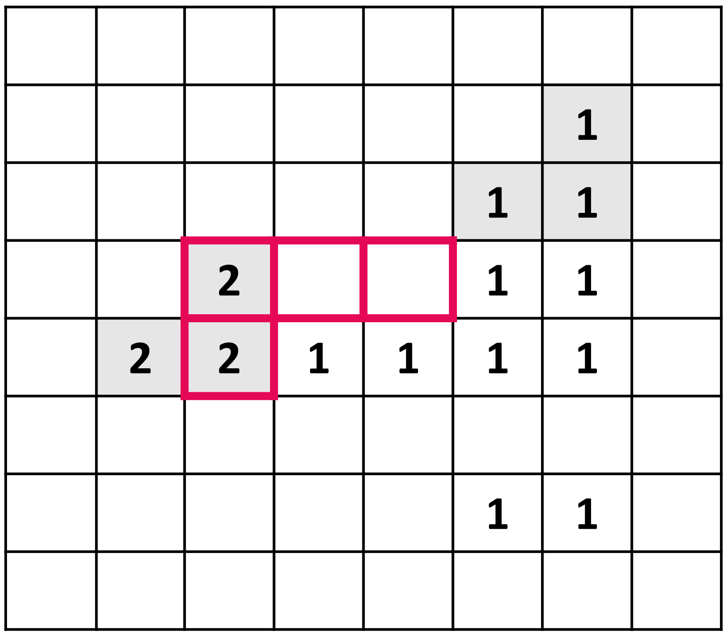
Equivalence LUT					
1	2	3			
1	1	2			

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	1	1	1	1	
					1	1	

Performance considerations

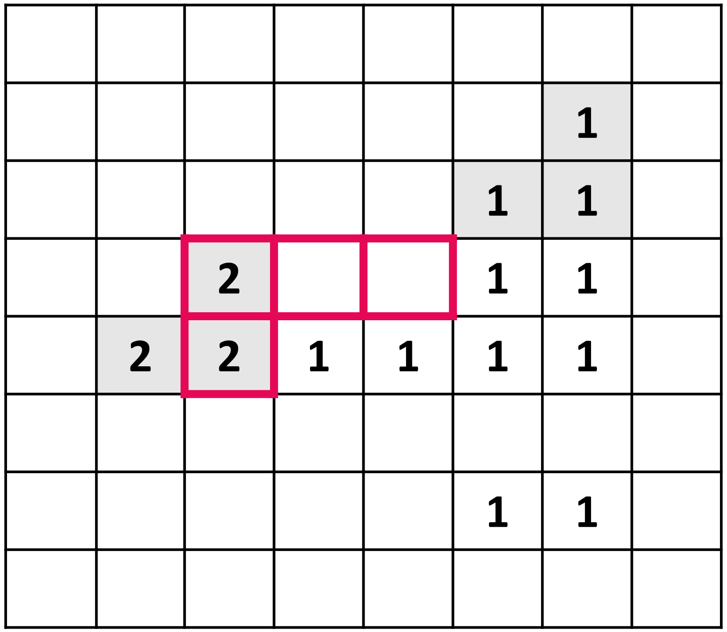
Label – Two-pass algorithm



Performance considerations

- Consider updated pixels only!

Label – Two-pass algorithm



Performance considerations

- Consider updated pixels only!
- Smaller LUT means:
  - Less lookup operations, which is good for performance
  - Less maximum number of labels that can be found

Label – Two-pass algorithm

						1	
					1	1	
		2			1	1	
	2	2	1	1	1	1	
					1	1	

Performance considerations

- Consider updated pixels only!
- Smaller LUT means:
  - Less lookup operations, which is good for performance
  - Less maximum number of labels that can be found
- If the number of expected labels is well below 255, uint8 images are feasible.

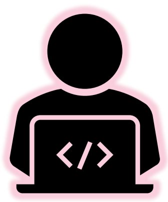
Label – Two-pass algorithm

uint32_t **labelTwoPass**(const image_t ***src**, image_t ***dst,**
const eConnected **connected,**
const uint32_t **lutSize**);

See file **EVDK_Operators\mensuration.c**

```
convertToUint8(cam, src);
thresholdOtsu(src, src, BRIGHTNESS_DARK);
removeBorderBlobsTwoPass(src, thr, CONNECTED_EIGHT, 128);

int32_t blobs = labelTwoPass(thr, lbl, CONNECTED_FOUR, 128);
printf("blobs: %d\n", blobs);

scaleFast(src, src);
if(blobs > 0)
{
    scaleFast(lbl, lbl);
    cv::imshow(“lbl", cv_lbl);
}
```

EVD1 – Assignment



Study guide

Week 6

1 Mensuration – labelTwoPass()

HAN_UNIVERSITY OF APPLIED SCIENCES

Centroid

- Measures the geographic centre of an object
- Is expressed in image coordinates
- Also known as the first central moment
- The centroid (x_c, y_c) is defined as

$$x_c = \frac{1}{A}\sum_{i=1}^{A} x \quad and \quad y_c = \frac{1}{A}\sum_{i=1}^{A} y$$

where

x: $x\ coordinate\ of\ the\ i_{th}\ object\ pixel$

y: $y\ coordinate\ of\ the\ i_{th}\ object\ pixel$

A: $object\ area$

HAN_UNIVERSITY OF APPLIED SCIENCES

Centroid - example

Image

	0	1	2	3	4	5	6	7
0								
1								
2				1				
3			1	1	1			
4		1	1	1	1	1		
5			1	1	1			
6				1				
7								

$$x_c = \frac{1}{A}\sum_{i=1}^{A} x$$

$$A = 13$$

All x values:

$$\begin{array}{ccccc} & & 3 & & \\ & 2 & 3 & 4 & \\ 1 & 2 & 3 & 4 & 5 \\ & 2 & 3 & 4 & \\ & & 3 & & \end{array}$$

$$x_c = \frac{1}{13} \times 39 = 3$$

$$y_c = \frac{1}{A}\sum_{i=1}^{A} y$$

$$A = 13$$

All y values:

$$\begin{array}{ccccc} & & 2 & & \\ & 3 & 3 & 3 & \\ 4 & 4 & 4 & 4 & 4 \\ & 5 & 5 & 5 & \\ & & 6 & & \end{array}$$

$$y_c = \frac{1}{13} \times 52 = 4$$

Centroid - algorithm

void **centroid**(const image_t ***img**, blobinfo_t ***blobinfo**,
const uint32_t **blobnr**);

See file **EVDK_Operators\mensuration.c**

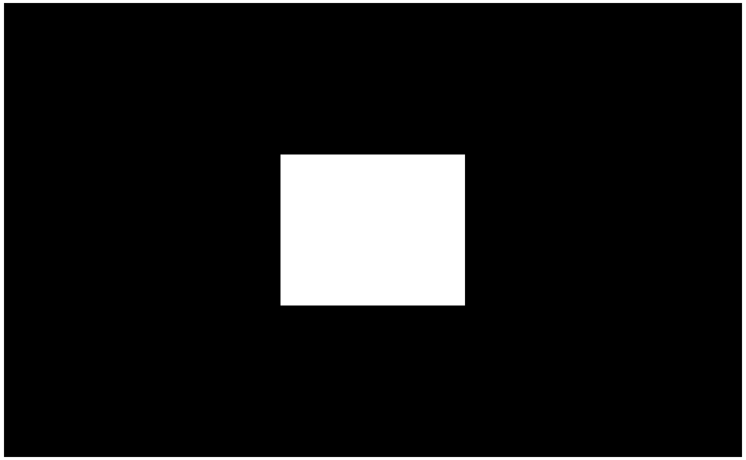
```
int32_t blobs = labelTwoPass(thr, lbl, CONNECTED_FOUR, 128);
printf("blobs: %d\n", blobs);

// Iterate the blobs
for(uint32_t blob=1; blob <= blobs; ++blob)
{
    blobinfo_t blobinfo;

    // Get the centroid of the blob
    centroid(lbl, &blobinfo, blob);

    // ...
}
```

Area

- Counts the number of pixels of an object

Area - example

$$blob\ area = 1600\ pixels$$



HAN_UNIVERSITY OF APPLIED SCIENCES

Area - algorithm

void **area**(const image_t ***img**, blobinfo_t ***blobinfo**,
const uint32_t **blobnr**);

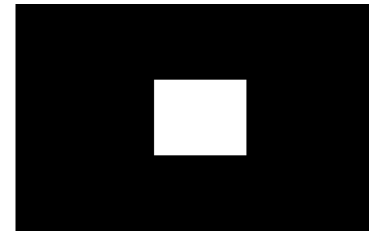
See file **EVDK_Operators\mensuration.c**

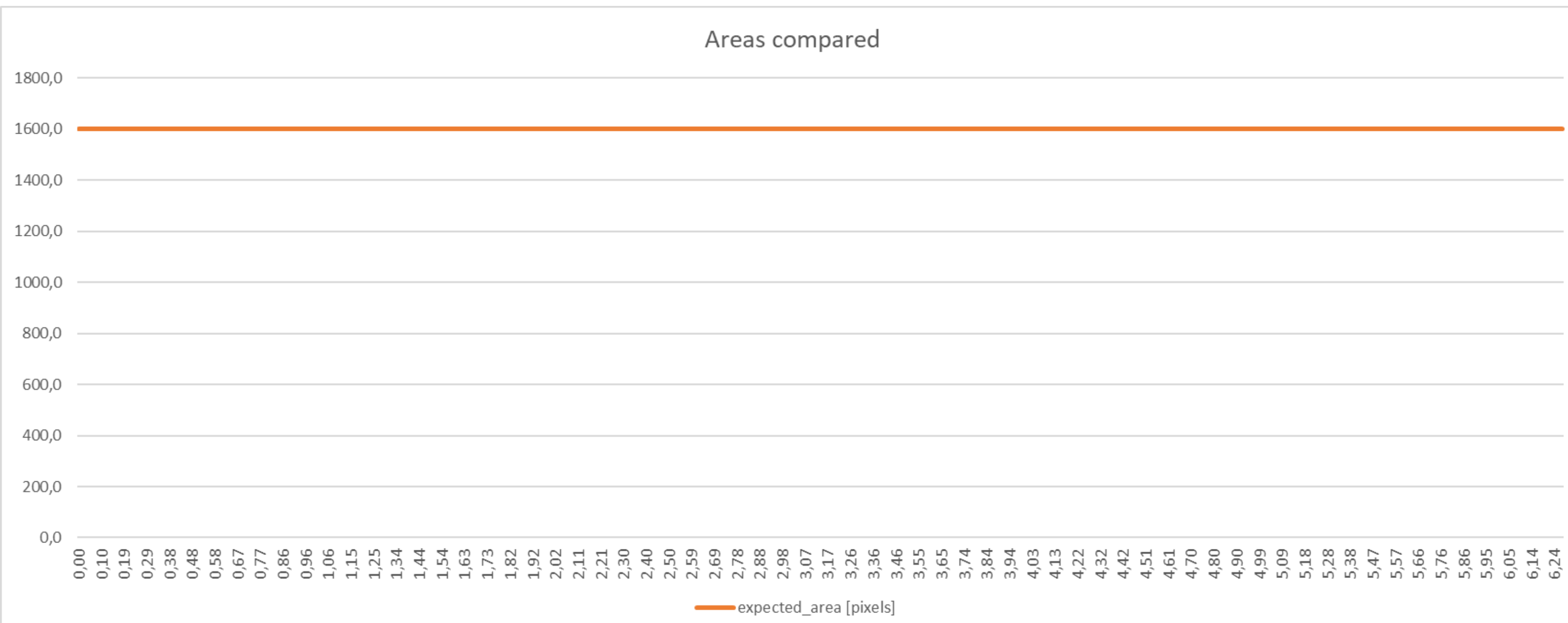
```c
// Iterate the blobs
for(uint32_t blob=1; blob <= blobs; ++blob)
{
    blobinfo_t blobinfo;

    // Get the blob size
    area(lbl, &blobinfo, blob);

    // Filter the blob based on its size
    if(blobinfo.area > 100)
    {
        // ...
    }
}
```

Area - Square

With $a = 40$

- $\color{orange}{expected\ area} = a^2 = 1600\ pixels$

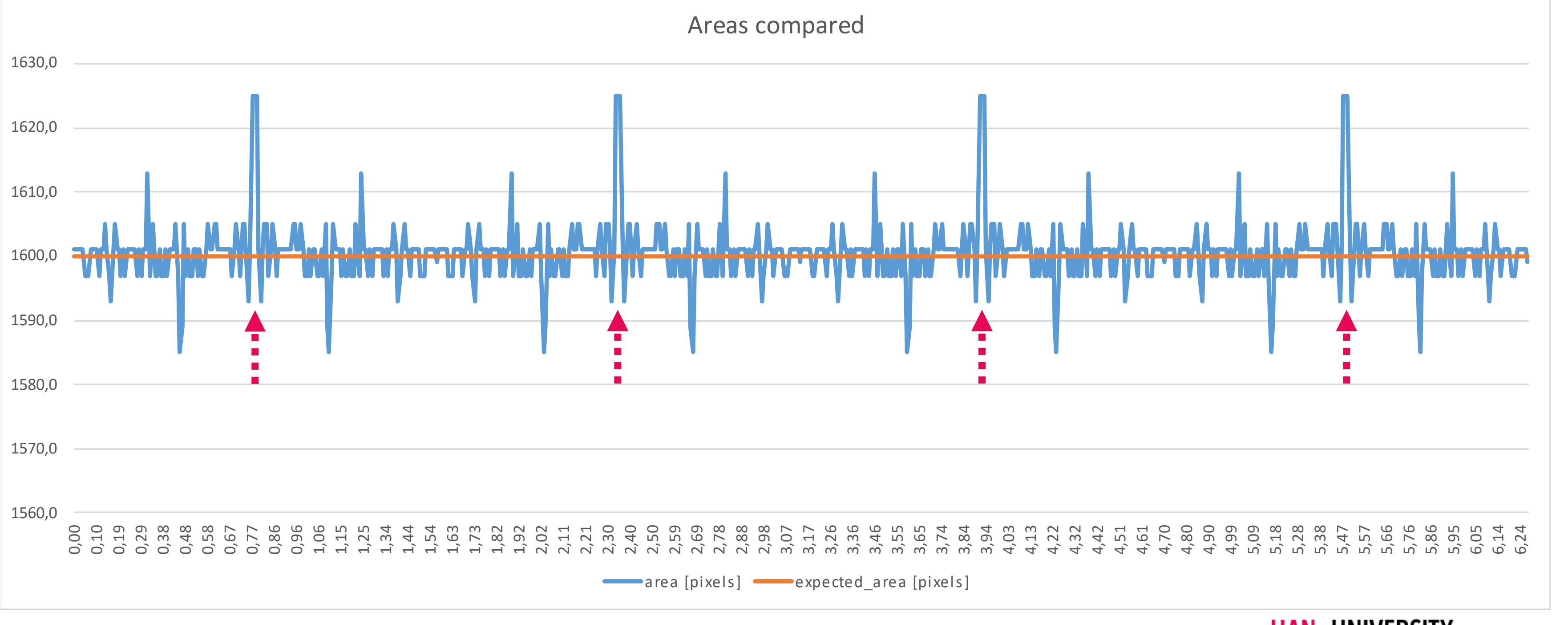




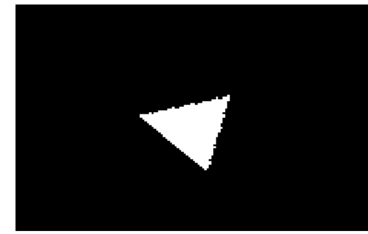
Area - Square

With $a = 40$

- $expected\ area = a^2 = 1600\ pixels$
- $measured\ area$

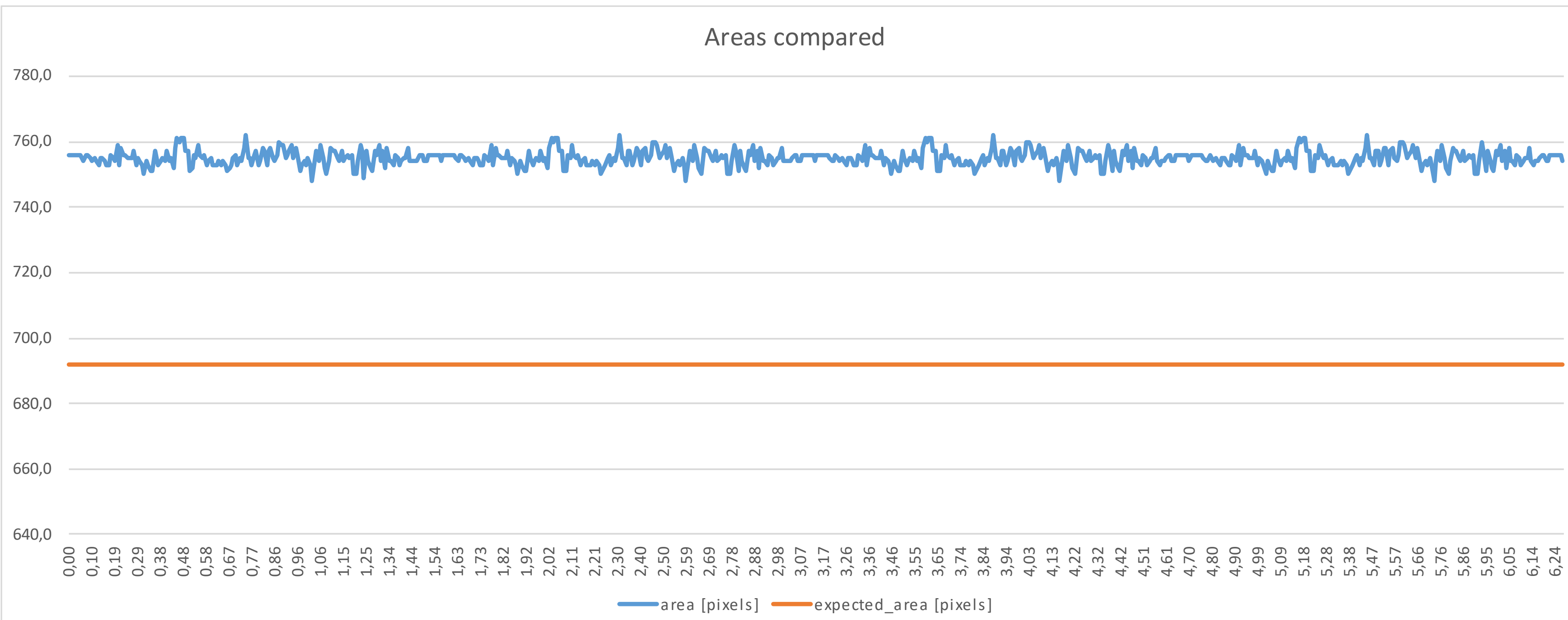


Area - Equilateral triangle



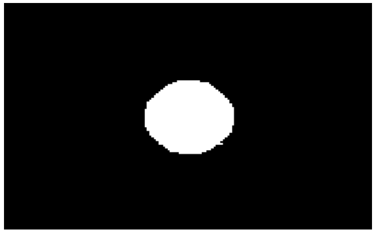
With $a = 40$

- $expected\ area = \frac{1}{4}a^2\sqrt{3} = 692\ pixels$
- $measured\ area$

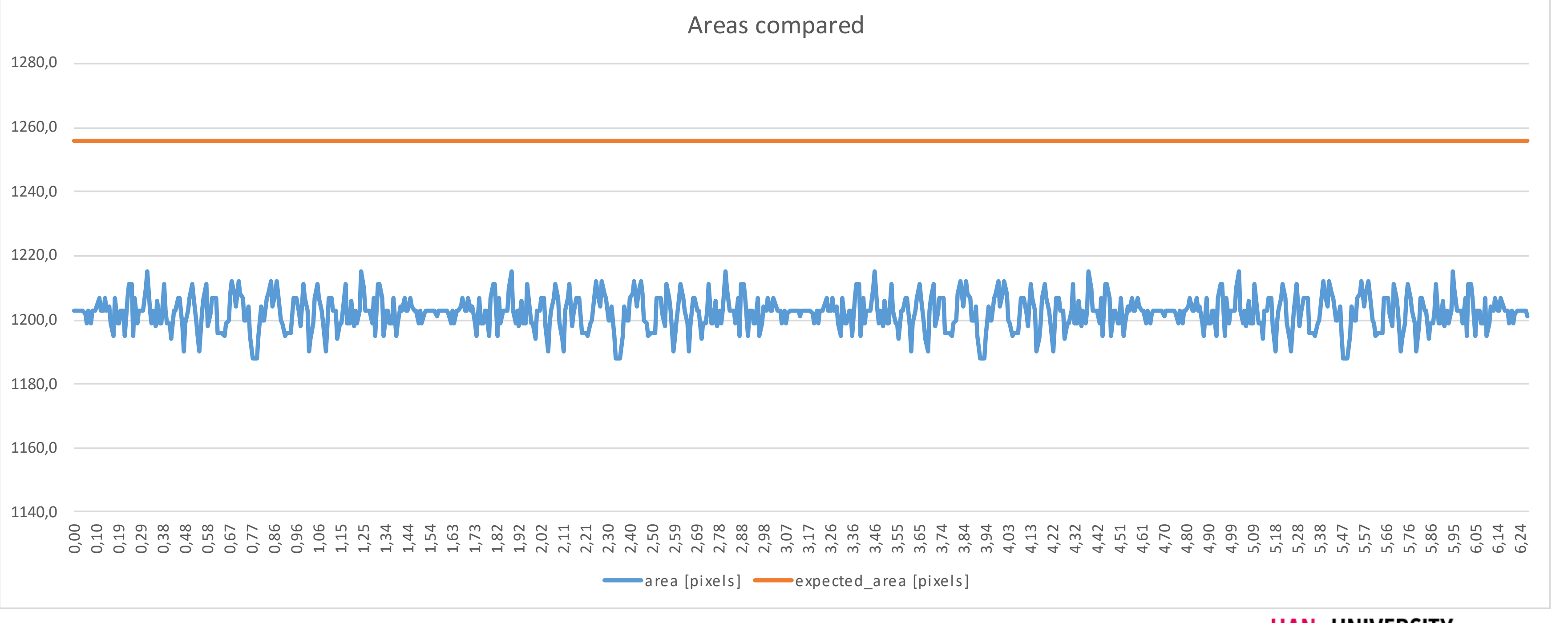


Area - Circle

With $a = 40$



- $expected\ area = \pi r^2 = 1256\, pixels$
- $measured\ area$



HAN_UNIVERSITY OF APPLIED SCIENCES

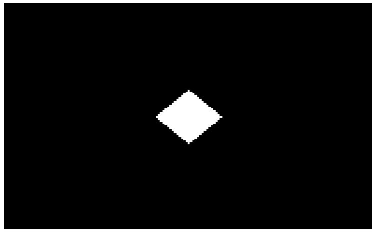
Area

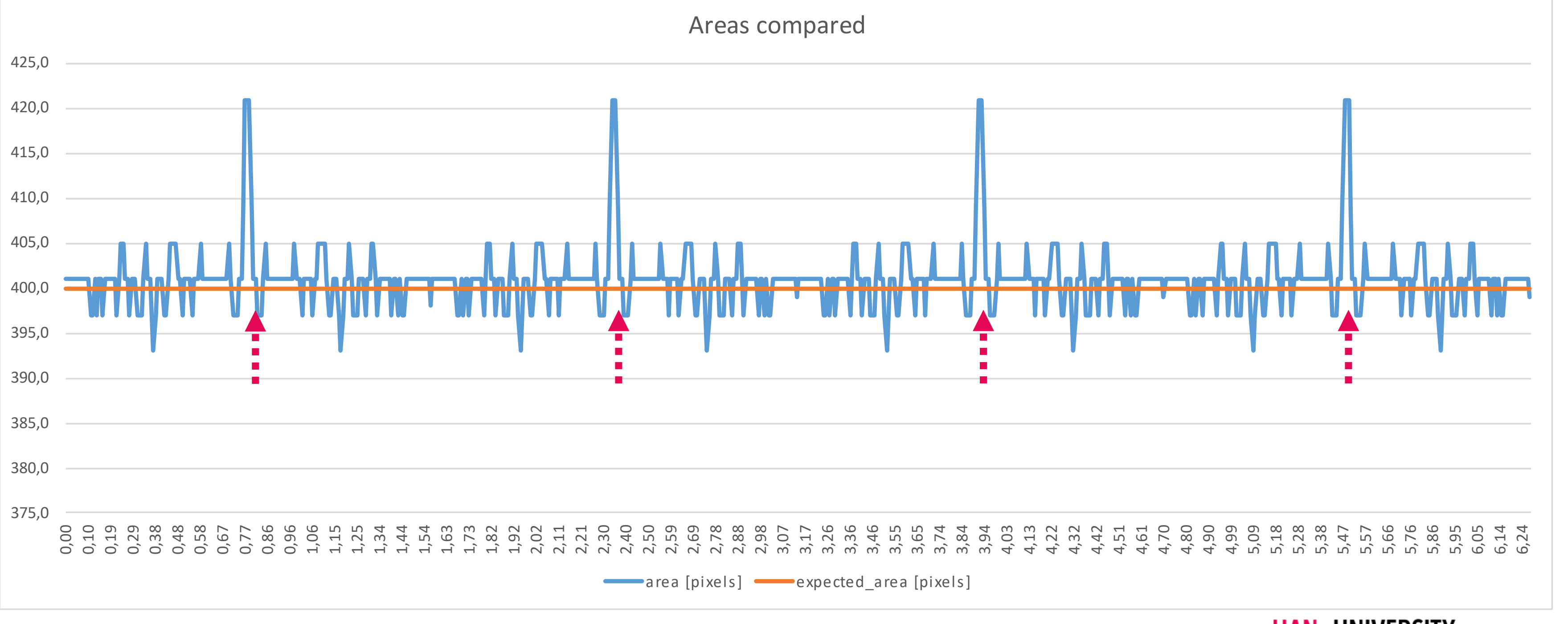
And how is this for smaller objects?

Area - Square

With $a = 20$

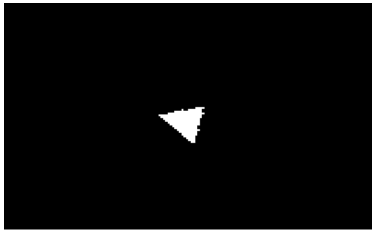
- $expected\ area = a^2 = 400\ pixels$
- $measured\ area$



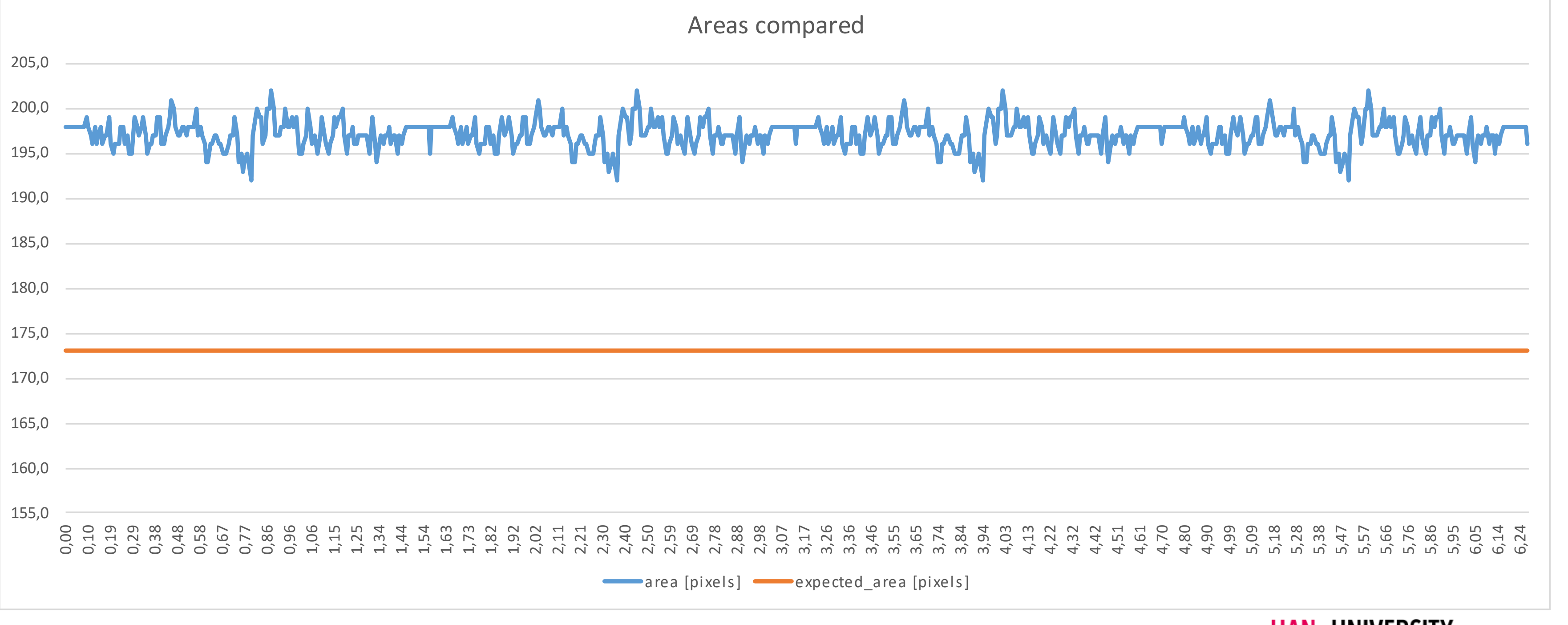


Area - Equilateral triangle

With $a = 20$



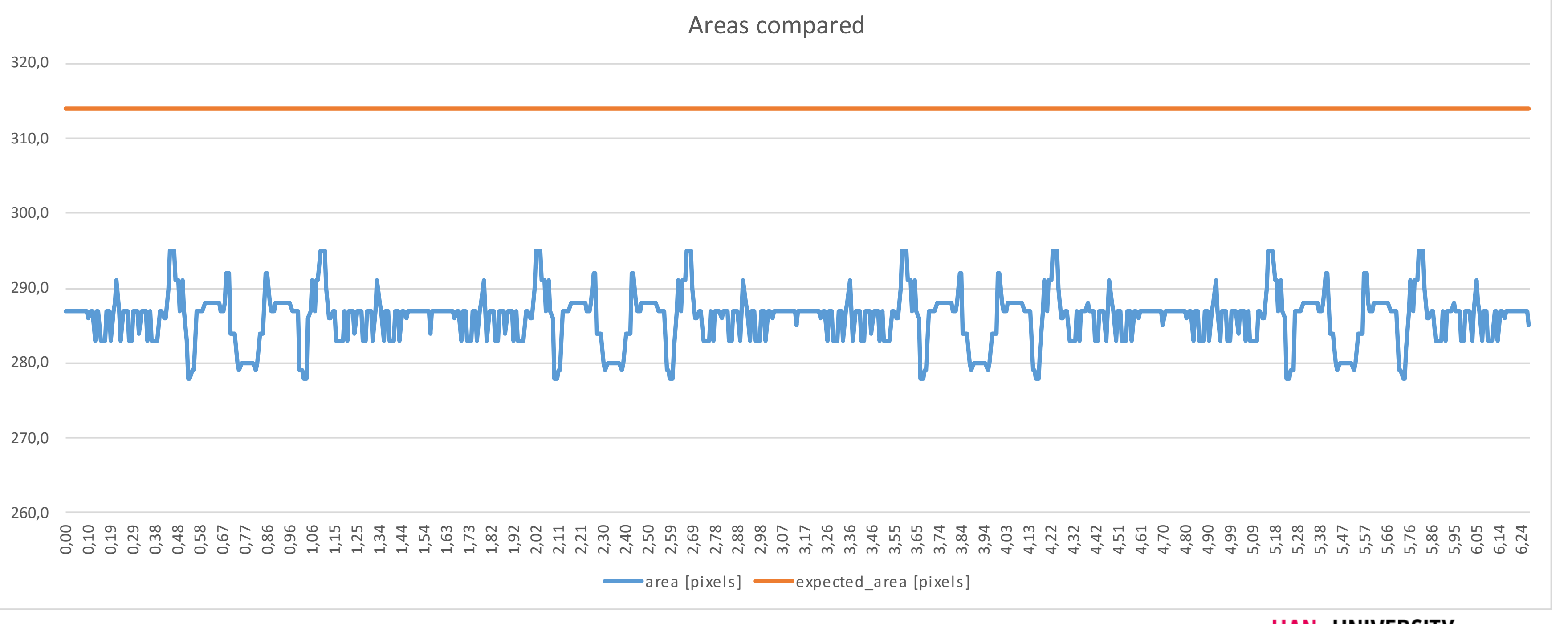
- $expected\ area = \frac{1}{4}a^2\sqrt{3} = 173\ pixels$
- $measured\ area$



Area - Circle

With $a = 20$

- $expected\ area = \pi r^2 = 314\ pixels$
- $measured\ area$



Area - Conclusion

In conclusion

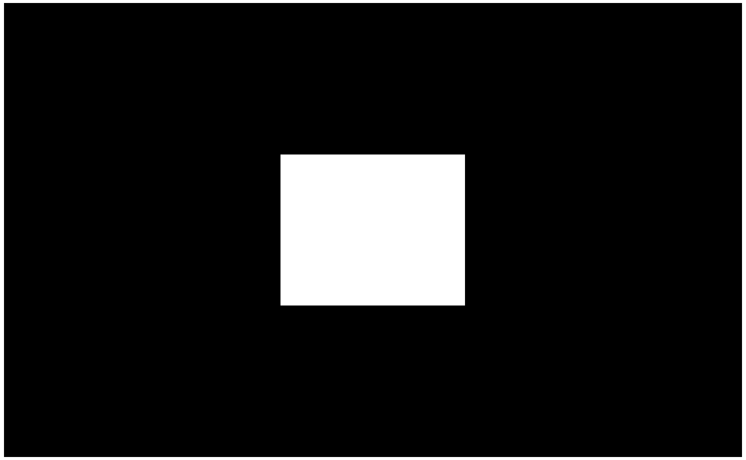
Shape	Area measured when rotated 2π radians, especially for small objects
Square	Is as expected (except when rotated $\frac{1}{2}\pi$ radians)
Equilateral triangle	**Larger** than expected
Circle	**Smaller** than expected

Perimeter

- Calculates the perimeter of an object

Perimeter - example

$$blob\ perimeter = 160\ pixels$$



Perimeter - example

void **perimeter**(const image_t ***img**, blobinfo_t ***blobinfo**,
const uint32_t **blobnr**);

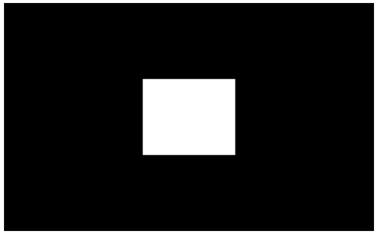
See file **EVDK_Operators\mensuration.c**

```
// Iterate the blobs
for(uint32_t blob=1; blob <= blobs; ++blob)
{
    blobinfo_t blobinfo;

    // Get the blob perimeter
    perimeter(lbl, &blobinfo, blob);

    // ...
}
```

Perimeter alternative a

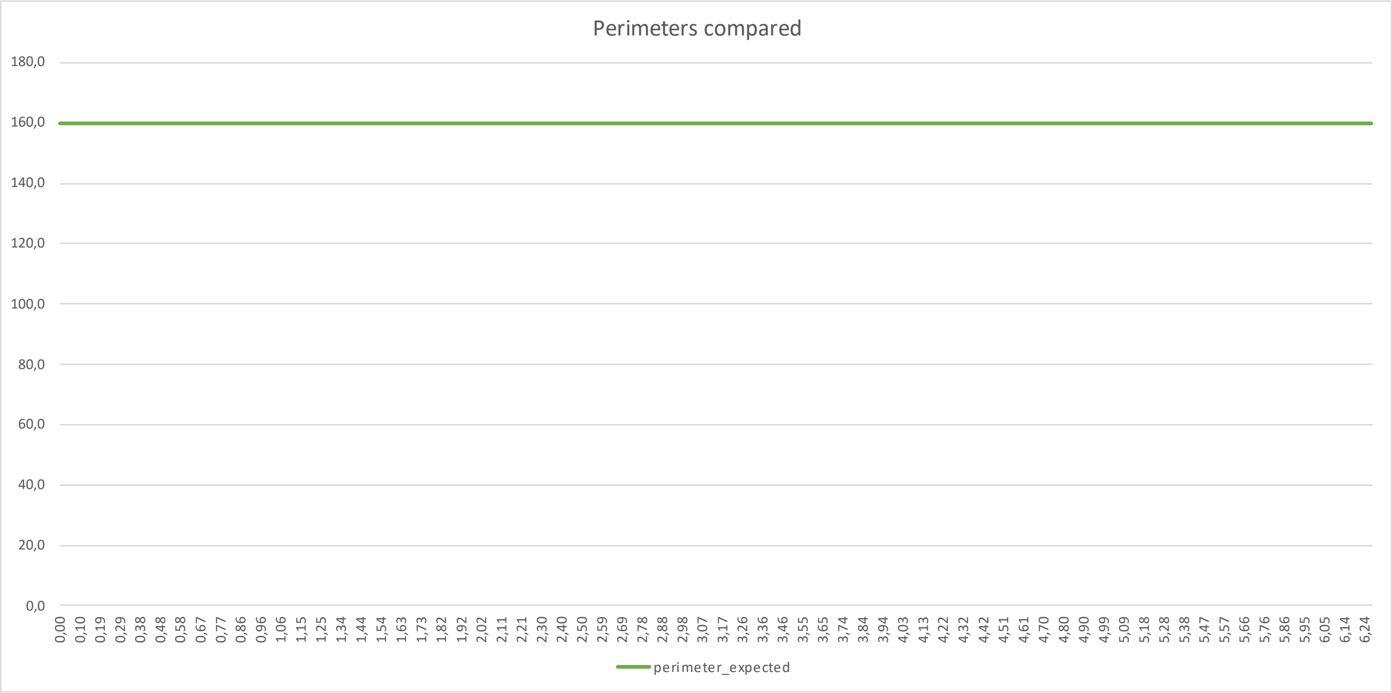
Count all edge pixels

Perimeter alternative a - Square



With $a = 40$

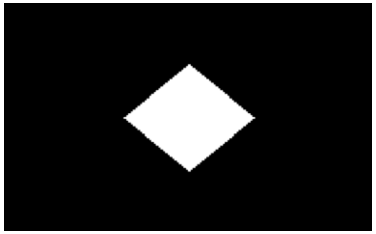
- $\textcolor{green}{expected\ perimeter} = 4a = 160\ pixels$

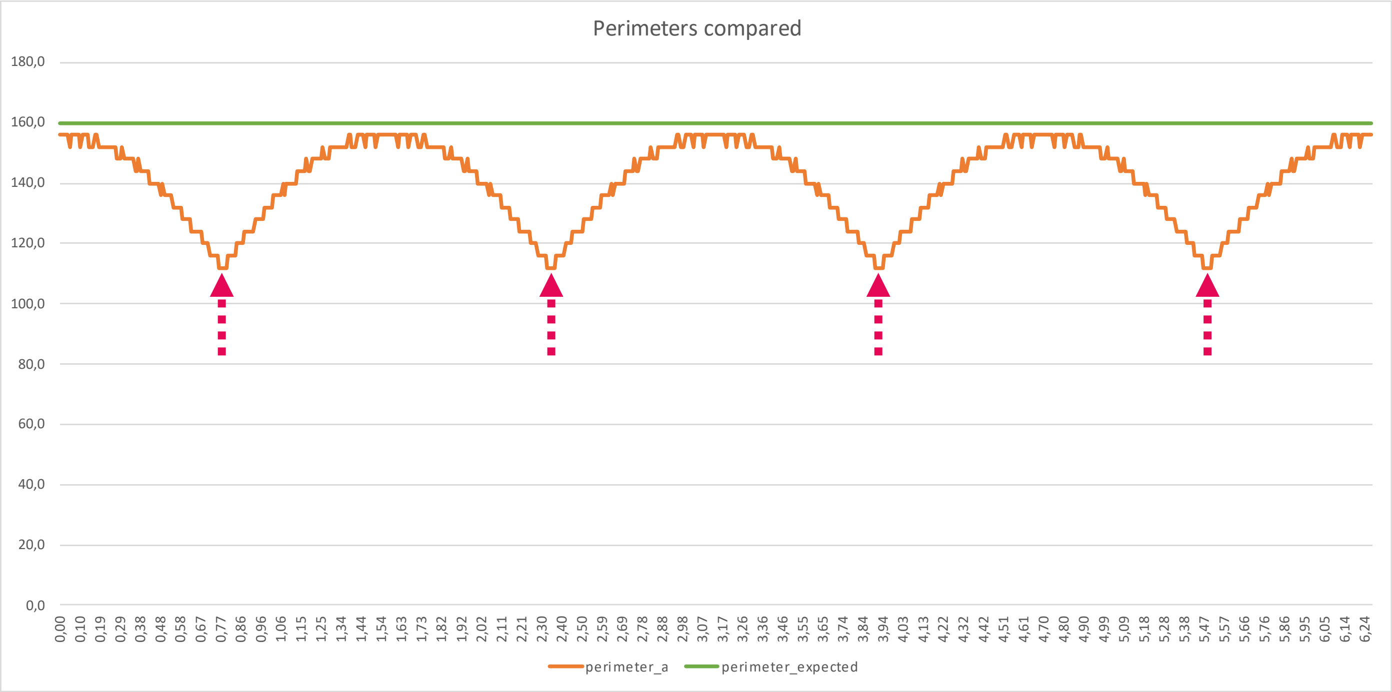


Perimeter alternative a - Square

With $a = 40$

- $expected\ perimeter = 4a = 160\ pixels$
- $measured\ perimeter$



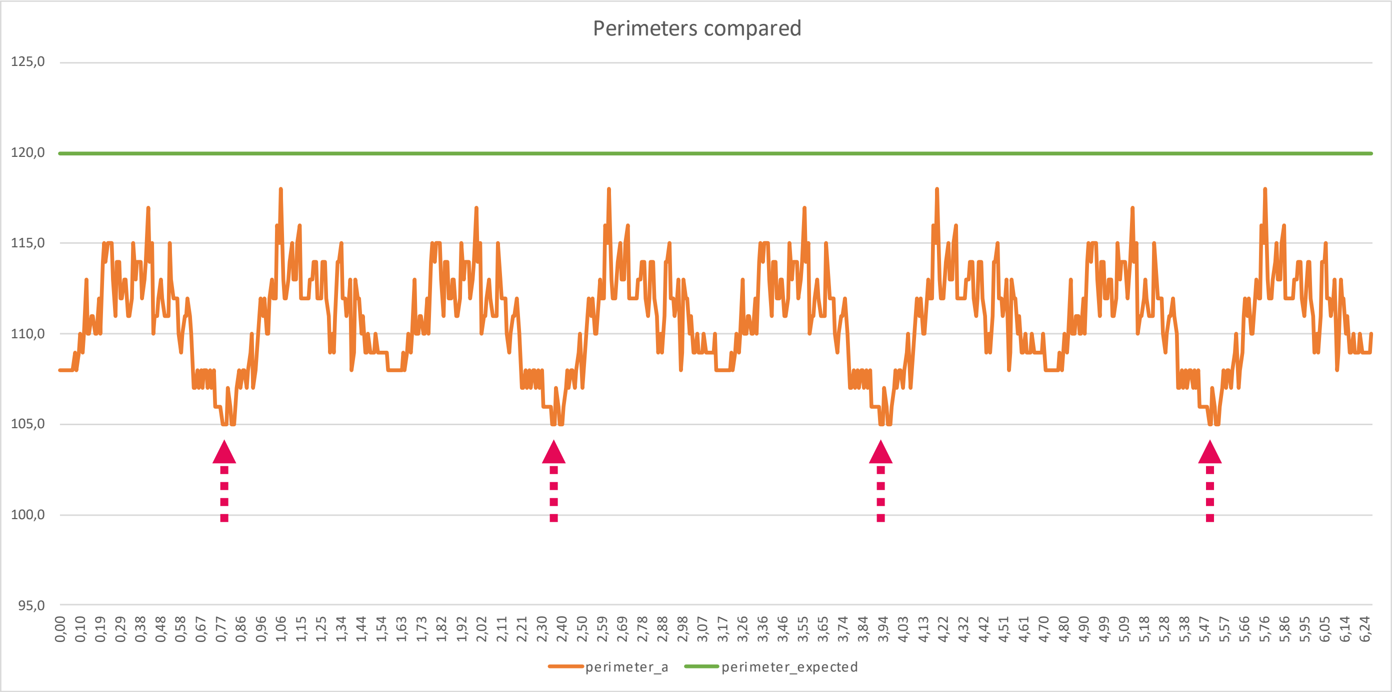


HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative a - Equilateral triangle

With $a = 40$

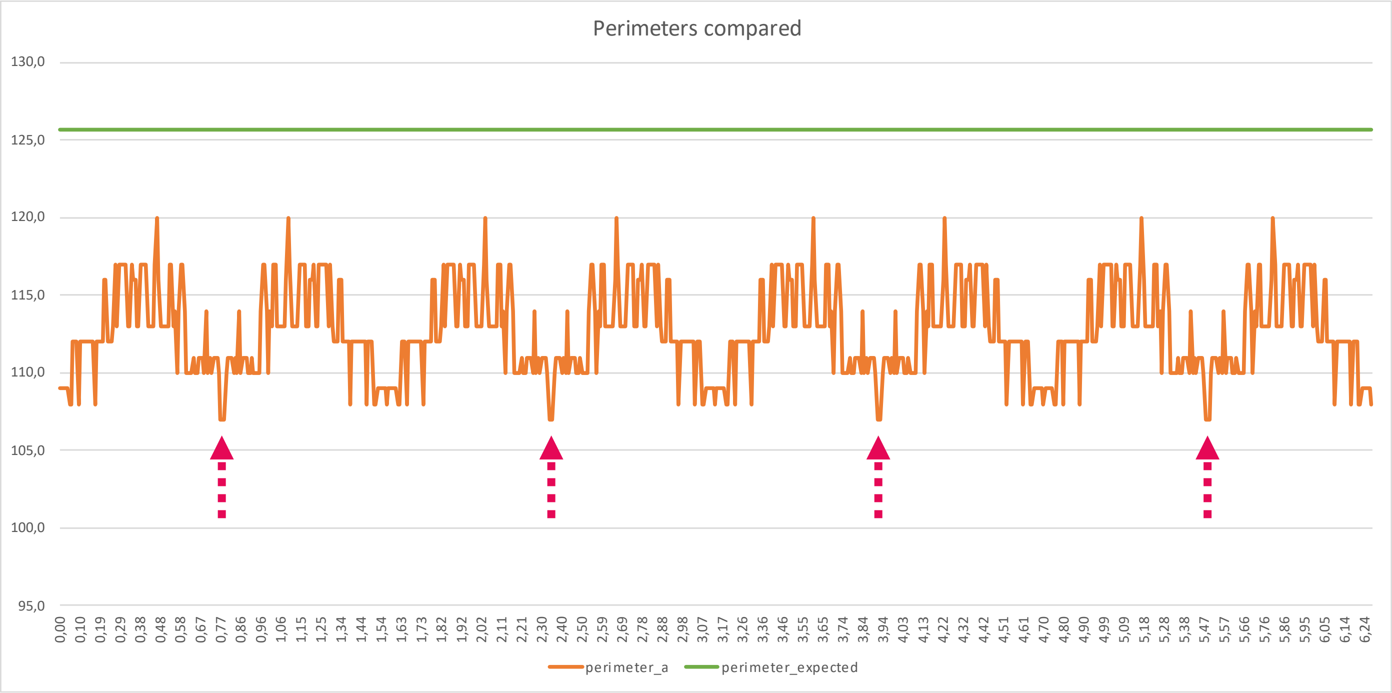
- $expected\ perimeter = 3a = 120\ pixels$
- $measured\ perimeter$



Perimeter alternative a - Circle

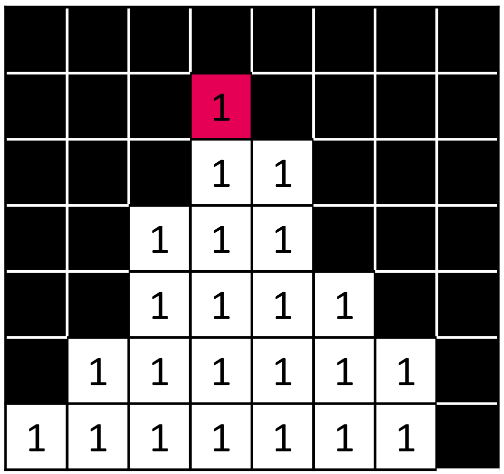
With $a = 40$

- $expected\ perimeter = 2\pi r = 126\ pixels$
- $measured\ perimeter$



Perimeter alternative b

Let’s consider the contribution of each edge pixel more carefully

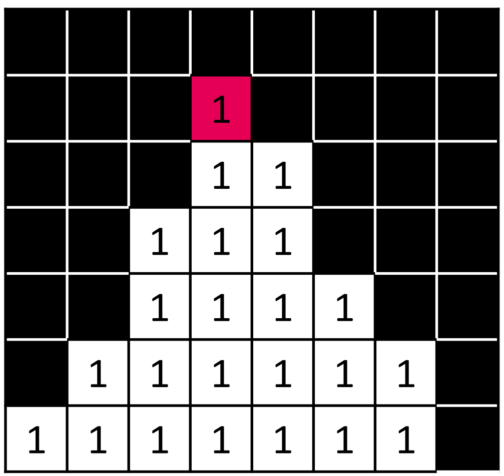


3-edges connected to
the background

$$p = p + 3$$

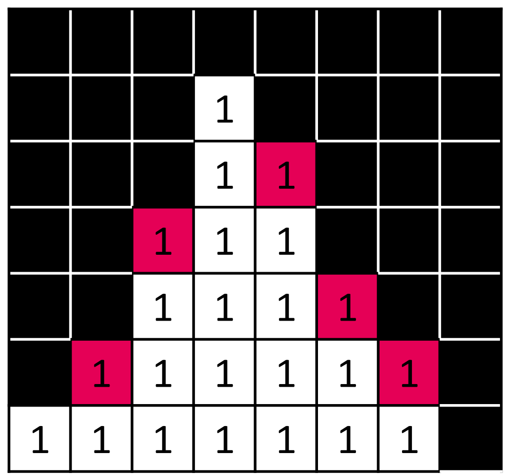
Perimeter alternative b

Let’s consider the contribution of each edge pixel more carefully



3-edges connected to the background

$$p = p + 3$$

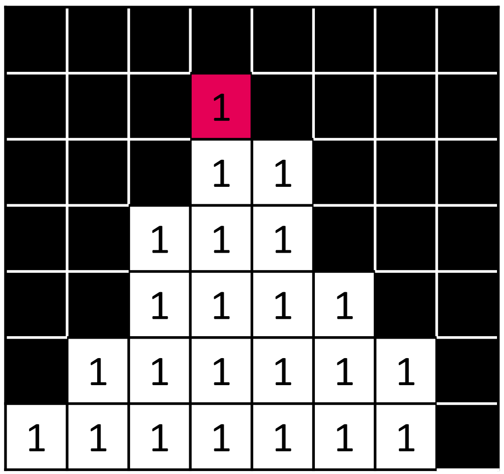


2-edges connected to the background

$$p = p + 2$$

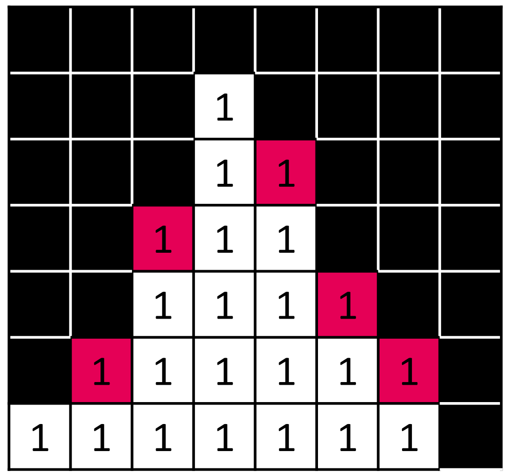
Perimeter alternative b

Let's consider the contribution of each edge pixel more carefully



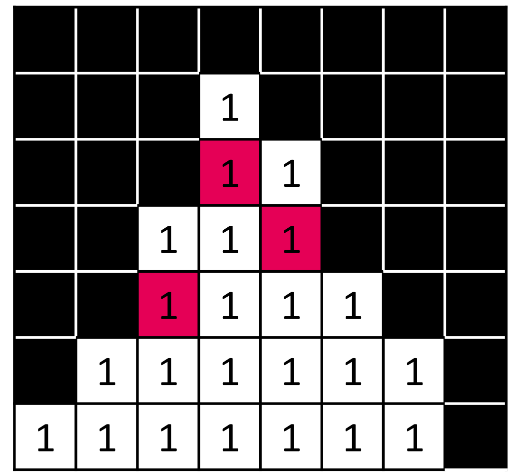
3-edges connected to the background

$$p = p + 3$$



2-edges connected to the background

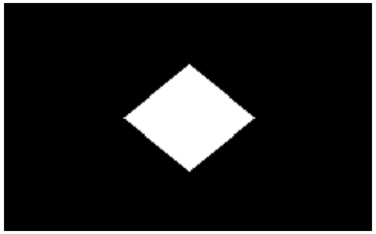
$$p = p + 2$$



1-edge connected to the background

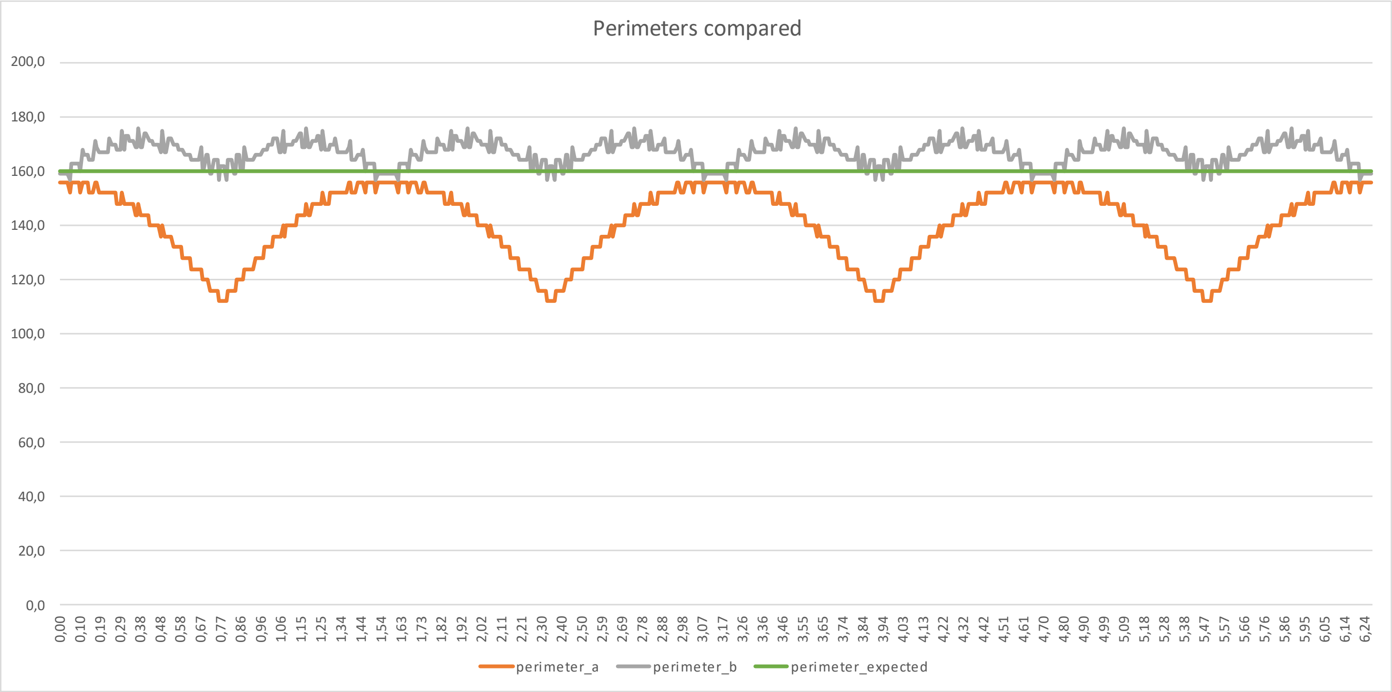
$$p = p + 1$$

Perimeter alternative b - Square



With $a = 40$

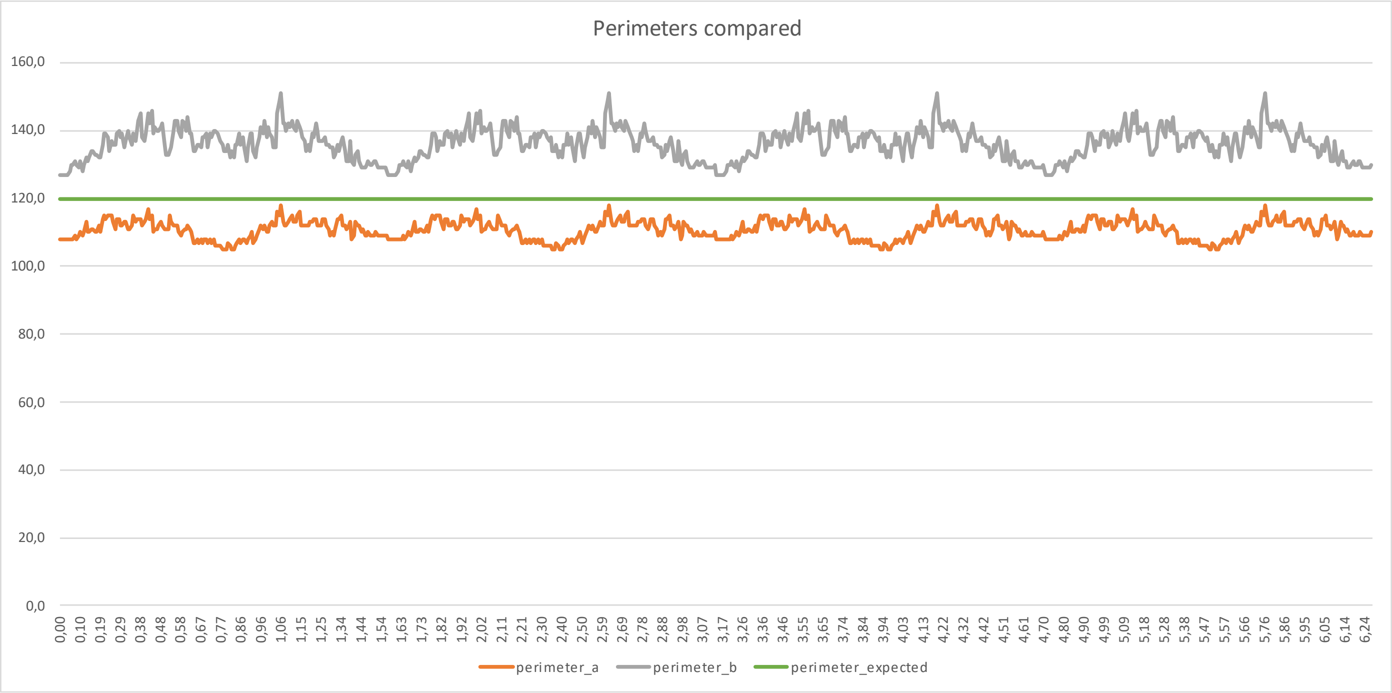
- $\color{green}{expected\ perimeter} = 4a = 160\ pixels$
- $measured\ perimeter$



Perimeter alternative b - Equilateral triangle

With $a = 40$

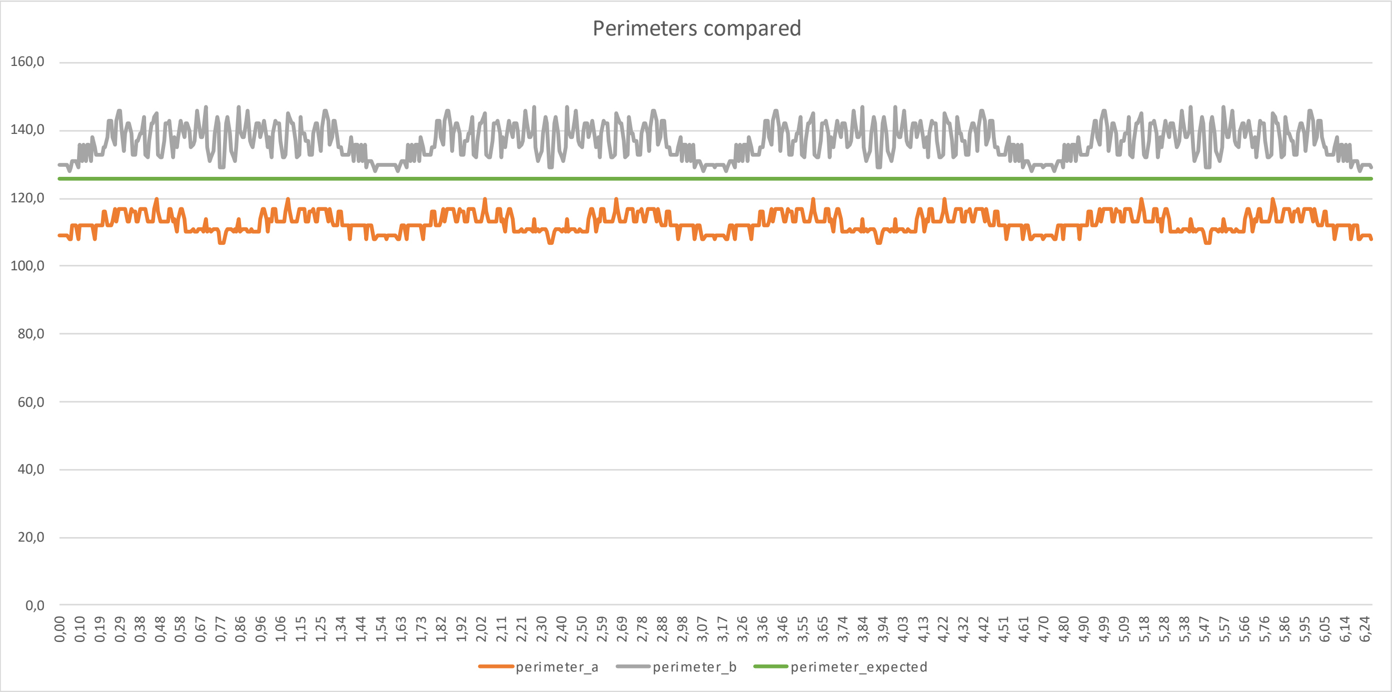
- $\textit{expected perimeter} = 3a = 120\ pixels$
- $\textit{measured perimeter}$



Perimeter alternative b - Circle

With $a = 40$

- $expected\ perimeter = 2\pi r = 126\ pixels$
- $measured\ perimeter$



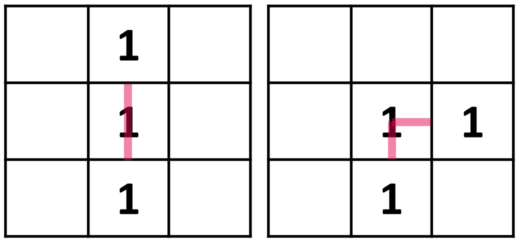
HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative c

Let’s consider the contribution of each edge pixel as described by the following source

Benkrid, K., Crookes, D., & Benkrid, A. (2000, September). Design and FPGA implementation of a perimeter estimator. In *Proceedings of the Irish Machine Vision and Image Processing Conference* (pp. 51-57).

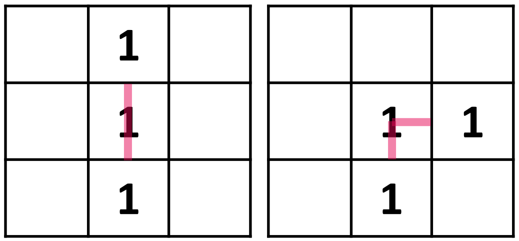
Perimeter alternative c



Horizontal and vertical links

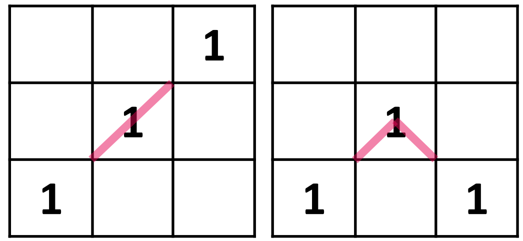
$$p = p + 1$$

Perimeter alternative c



Horizontal and vertical links

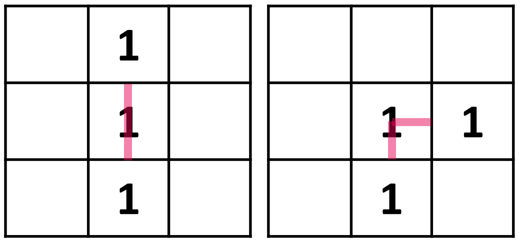
$$p = p + 1$$



Diagonal links

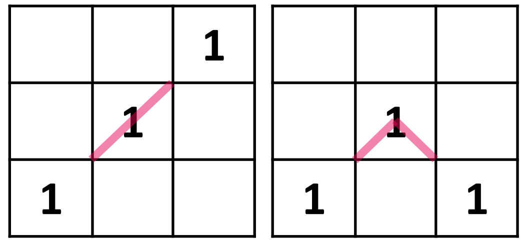
$$p = p + \sqrt{2}$$

Perimeter alternative c



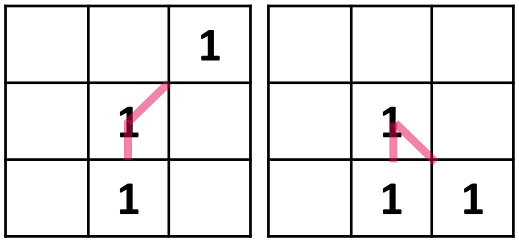
Horizontal and vertical links

$$p = p + 1$$



Diagonal links

$$p = p + \sqrt{2}$$

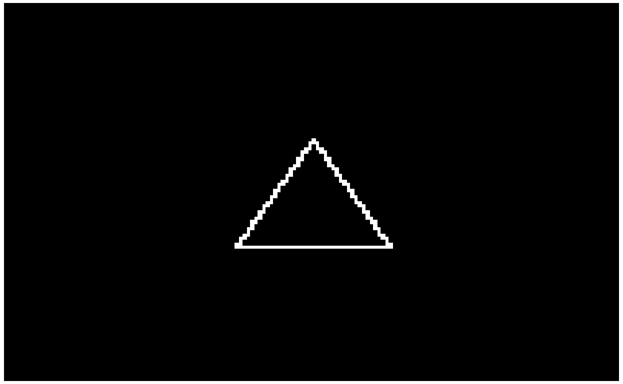


Horizontal and vertical link combined with diagonal link

$$p = p + \frac{1}{2} + \frac{\sqrt{2}}{2}$$

Perimeter alternative c

One way for calculating the contribution is by considering edge pixels only and calculate the contribution by a convolution with a mask



10	**2**	**10**
2	**1**	**2**
10	**2**	**10**

$$result_{p(x,y)} = \sum_{i=-1}^{i=1} \sum_{j=-1}^{j=1} mask(i,j) \quad if\ p(x+i, y+j)\ is\ an\ edge\ pixel$$

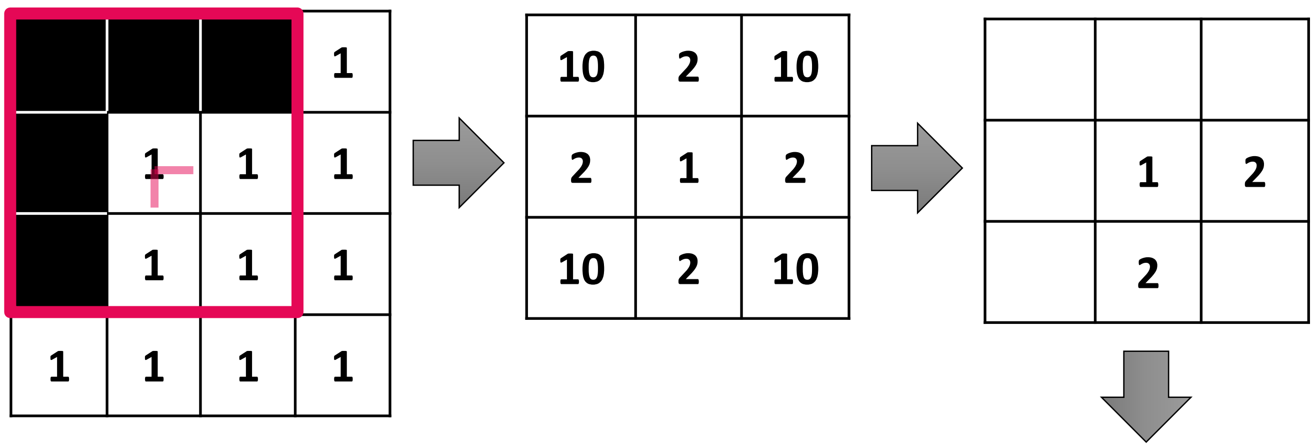
Perimeter alternative c

One way for calculating the contribution is by considering edge pixels only and calculate the contribution by a convolution with a mask

$Result$ **for pixel at** (x, y)	**Option**	**Perimeter increment**
5 or 15 or 7 or 25 or 27 or 17	1	1
21 or 33	2	$\sqrt{2}$
13 or 23	3	$\frac{1}{2} + \frac{\sqrt{2}}{2}$
Anything else	-	0

Perimeter alternative c

Detailed example

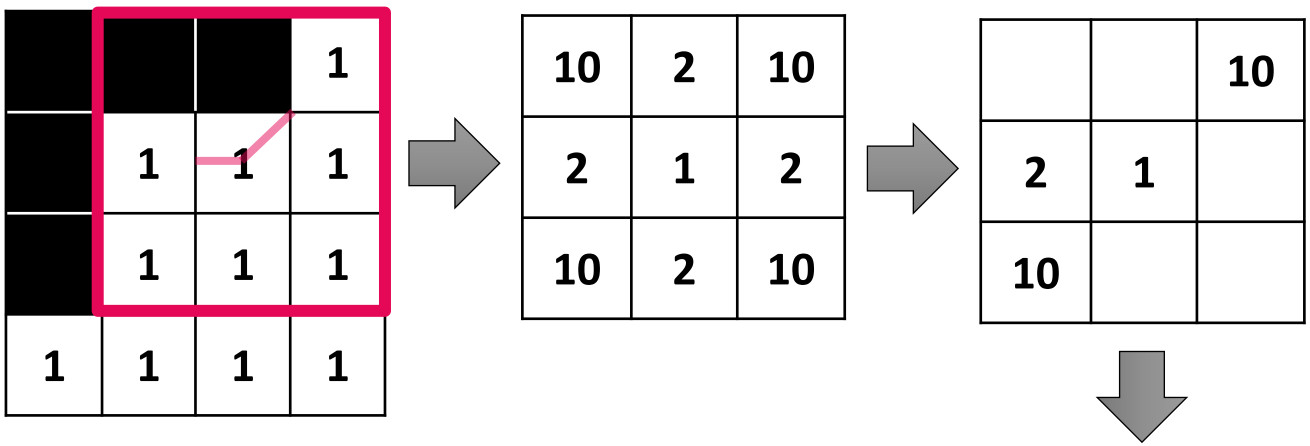


$$2 + 1 + 2 = 5$$

$$perimeter \mathrel{+}= 1$$

Perimeter alternative c

Detailed example

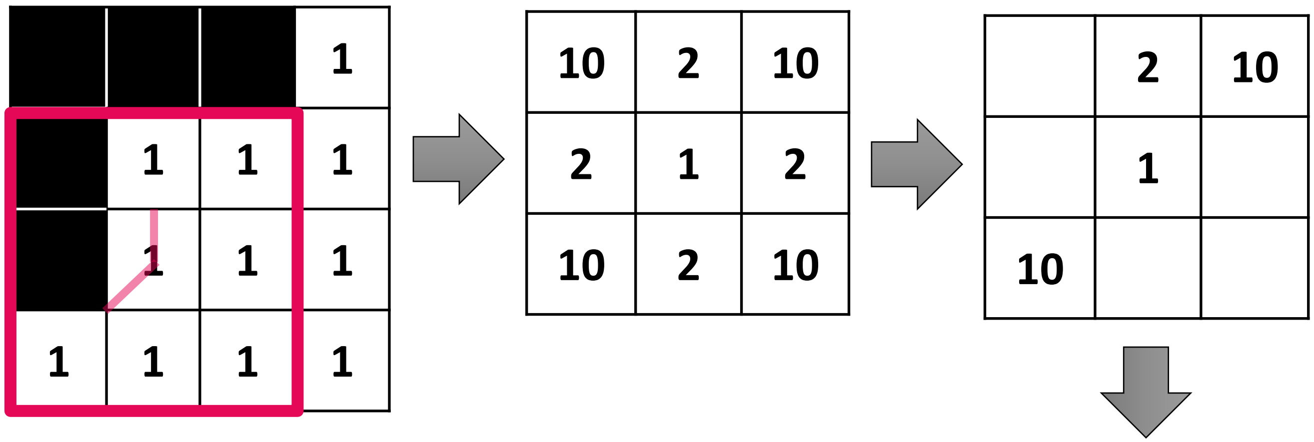


$$10 + 2 + 1 + 10 = 23$$

$$perimeter \mathrel{+}= \tfrac{1}{2} + \tfrac{1}{2}\sqrt{2}$$

Perimeter alternative c

Detailed example

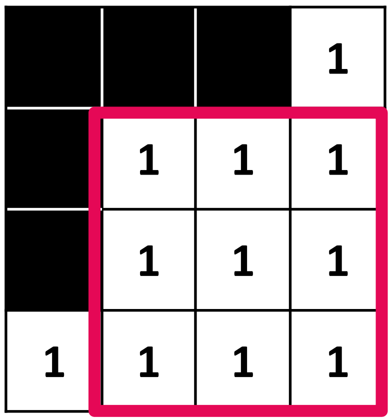


$$2 + 10 + 1 + 10 = 23$$

$$perimeter \mathrel{+}= \tfrac{1}{2} + \tfrac{1}{2}\sqrt{2}$$

Perimeter alternative c

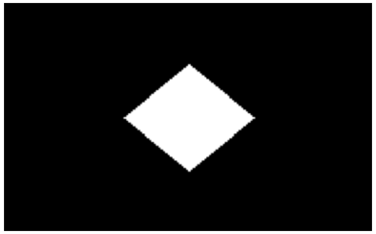
Detailed example



Not an edge pixel

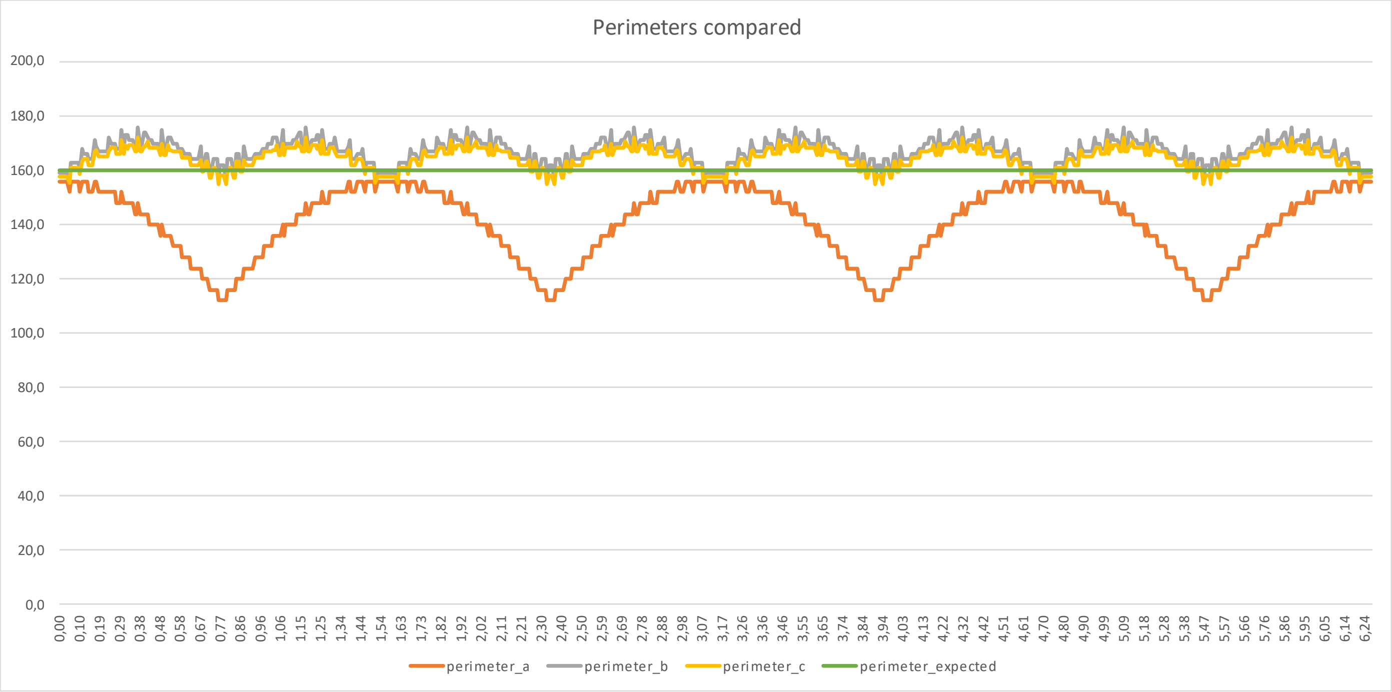
$$perimeter \mathrel{+}= 0$$

Perimeter alternative c - Square



With $a = 40$

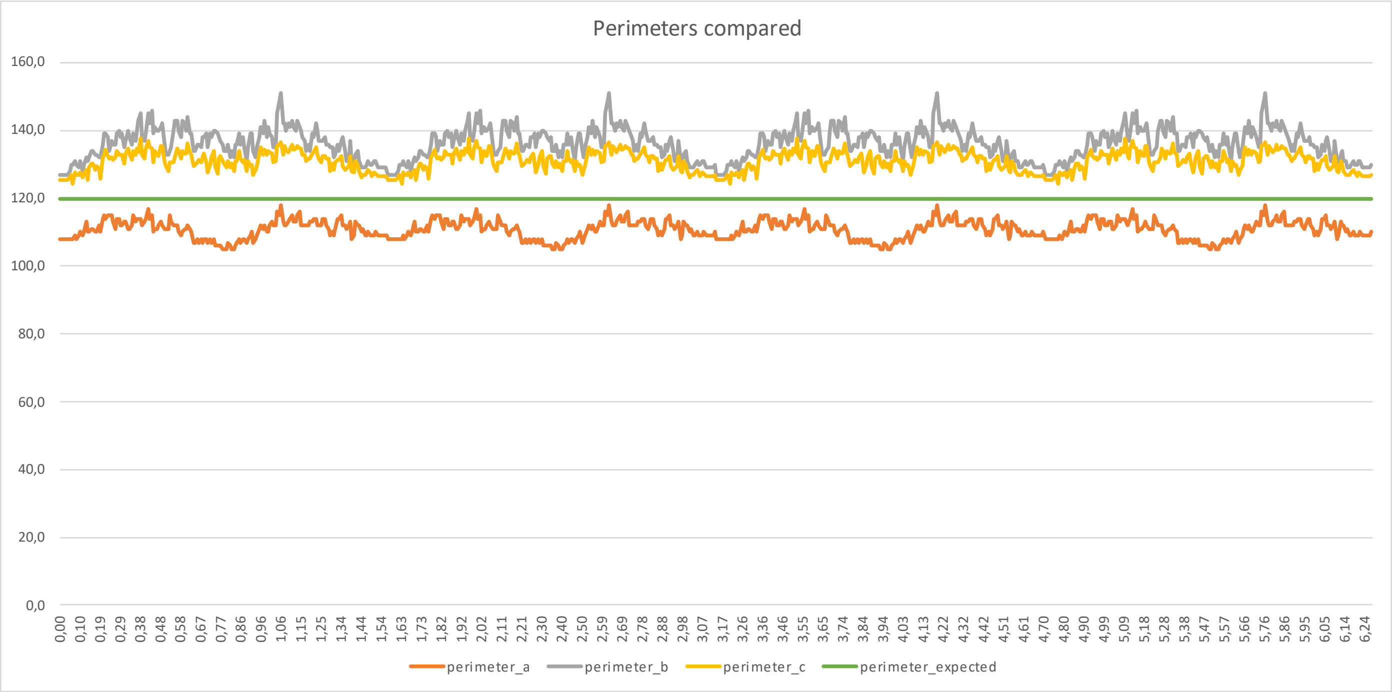
- $expected\ perimeter = 4a = 160\ pixels$
- $measured\ perimeter$



Perimeter alternative c - Equilateral triangle

With $a = 40$

- $expected\ perimeter = 3a = 120\ pixels$
- $measured\ perimeter$

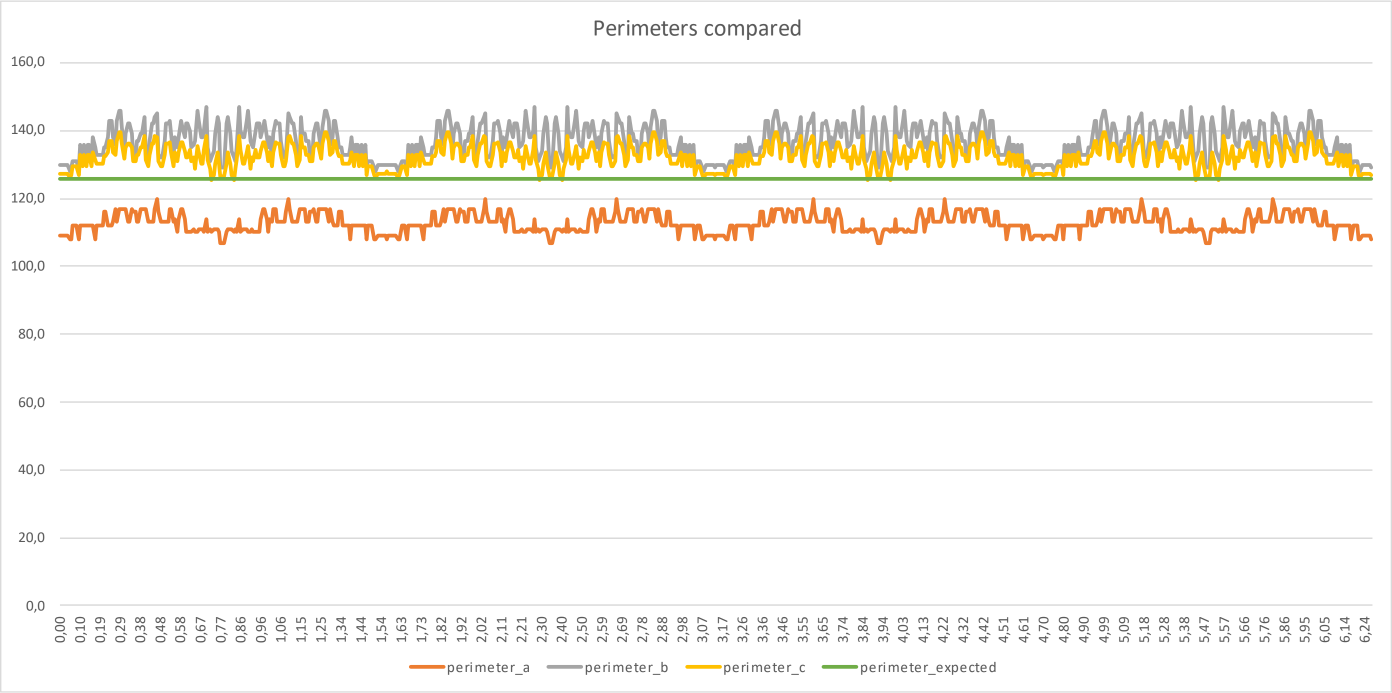


HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative c - Circle

With $a = 40$

- $expected\ perimeter = 2\pi r = 126\ pixels$
- $measured\ perimeter$

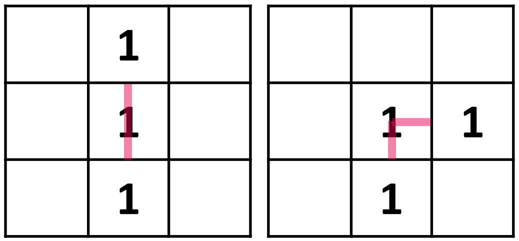


HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative d

Are there better alternatives for these contributions?

Perimeter alternative d

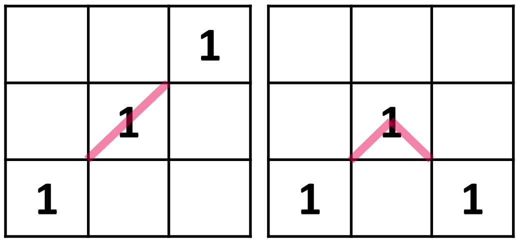


Horizontal and vertical links

$$p = p + 1$$

No alternative

Perimeter alternative d

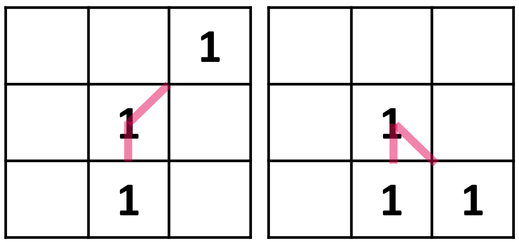


Diagonal links

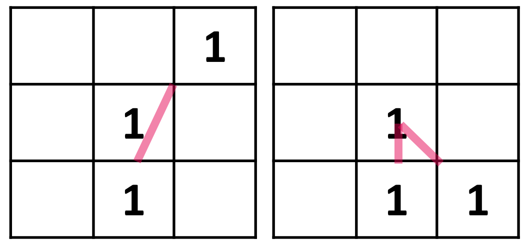
$$p = p + \sqrt{2}$$

No alternative

Perimeter alternative d



Horizontal and vertical link combined with diagonal link



Horizontal and vertical link combined with diagonal link

$$p = p + \frac{1}{2} + \frac{\sqrt{2}}{2} \longrightarrow p = p + \frac{1}{2}\sqrt{5}$$

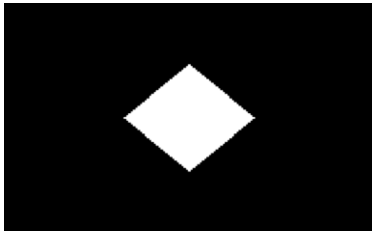
Although not correct for all cases, it will be correct for most cases!

Perimeter alternative d

Updated contributions

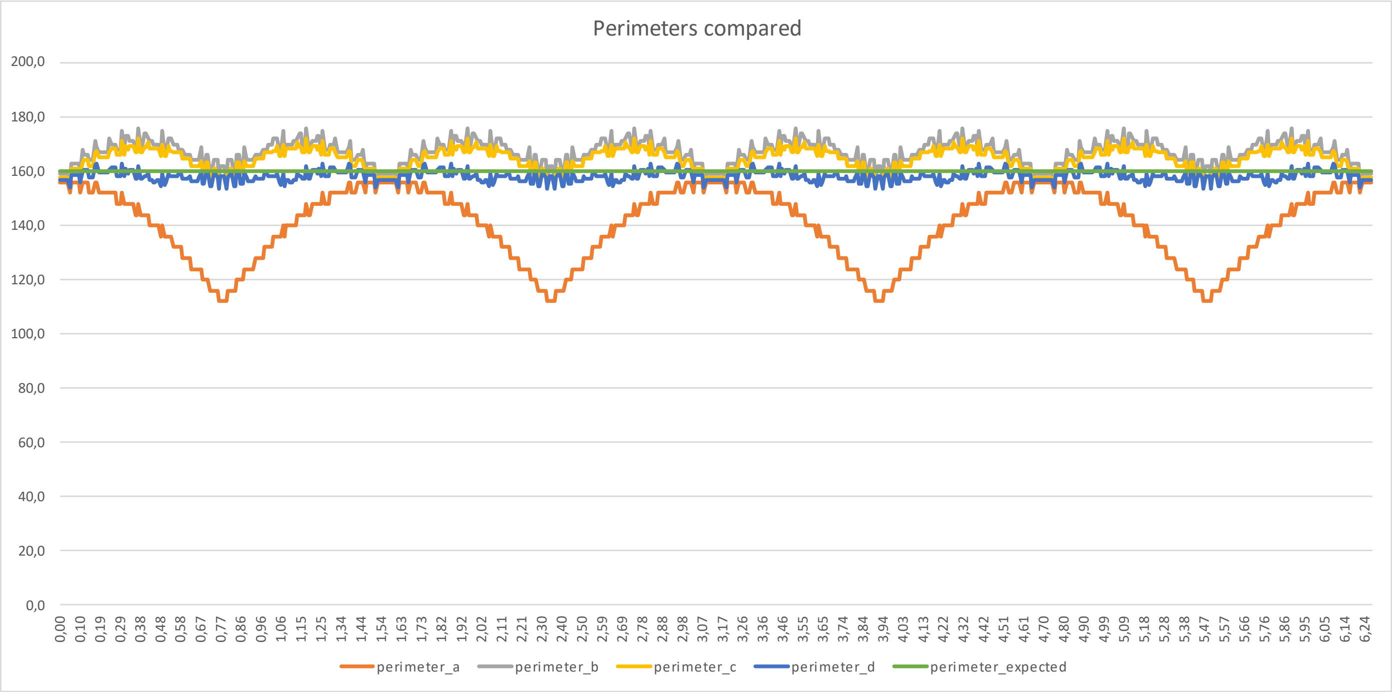
$Result$ **for pixel at** (x, y)	**Option**	**Perimeter increment alternative c**	**Perimeter increment alternative d**
5 or 15 or 7 or 25 or 27 or 17	1	1	1
21 or 33	2	$\sqrt{2}$	$\sqrt{2}$
13 or 23	3	$\frac{1}{2}+\frac{\sqrt{2}}{2}$	$\frac{\sqrt{5}}{2}$
Anything else	-	0	0

Perimeter alternative d - Square



With $a = 40$

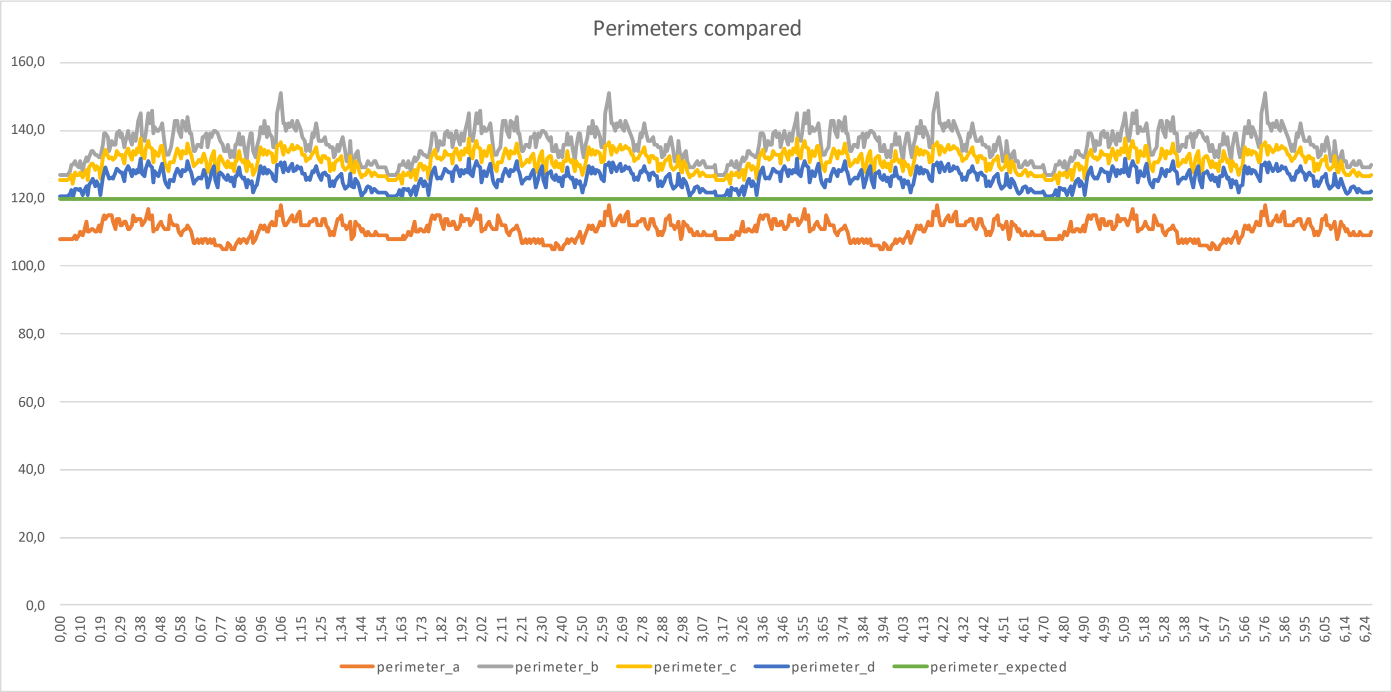
- $expected\ perimeter = 4a = 160\ pixels$
- $measured\ perimeter$



Perimeter alternative d - Equilateral triangle

With $a = 40$

- $\textit{expected perimeter} = 3a = 120\ \textit{pixels}$
- $\textit{measured perimeter}$

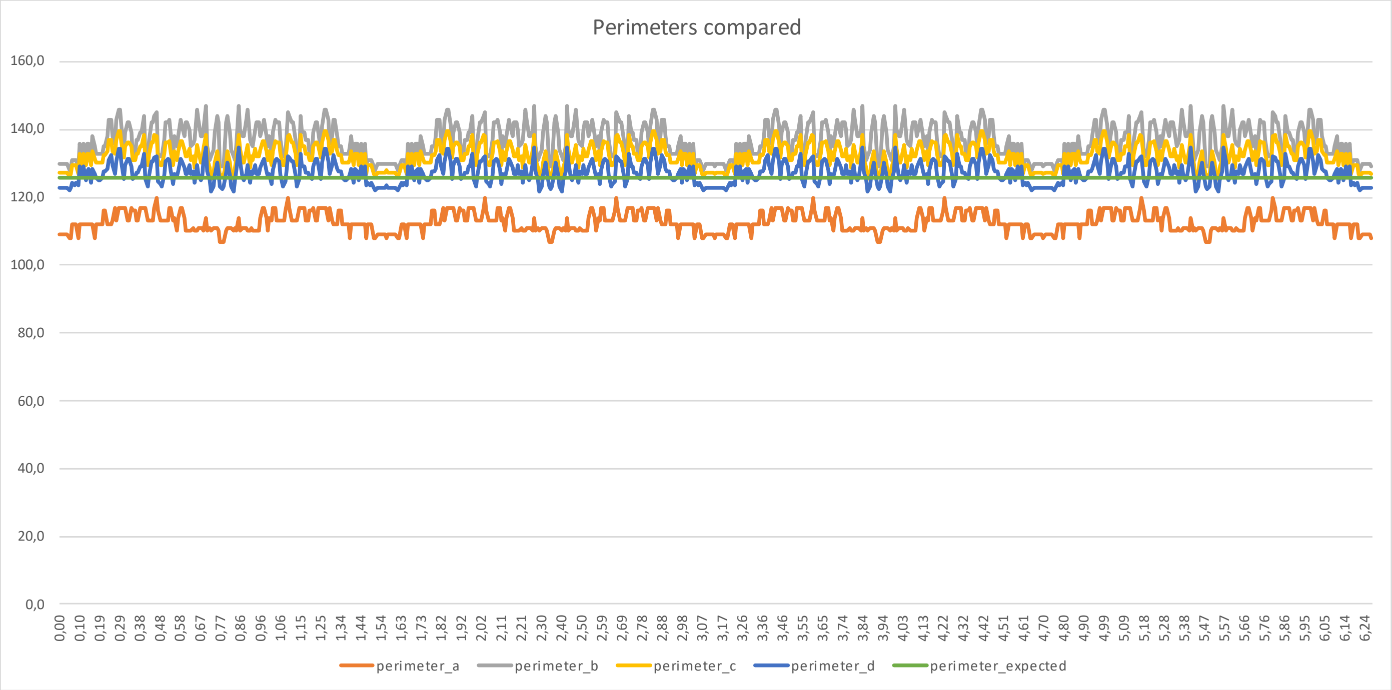


HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative d - Circle

With $a = 40$

- $expected\ perimeter = 2\pi r = 126\ pixels$
- $measured\ perimeter$

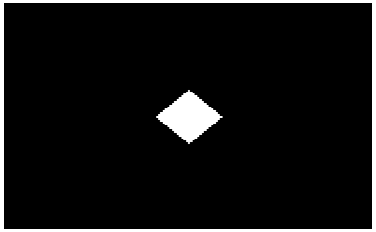


HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative d

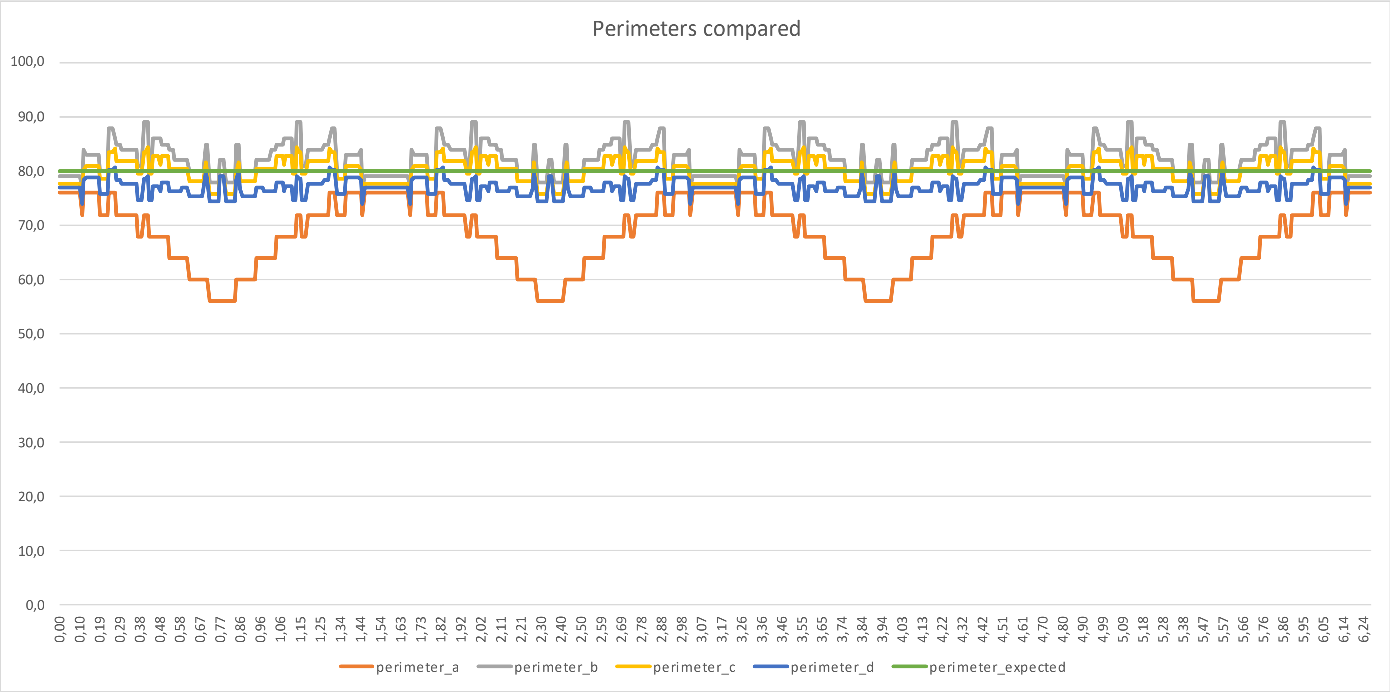
Does it hold for smaller objects?

Perimeter alternative d - Square



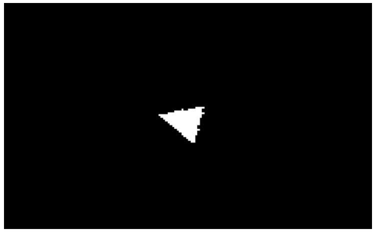
With $a = 20$

- $expected\ perimeter = 4a = 80\ pixels$
- $measured\ perimeter$



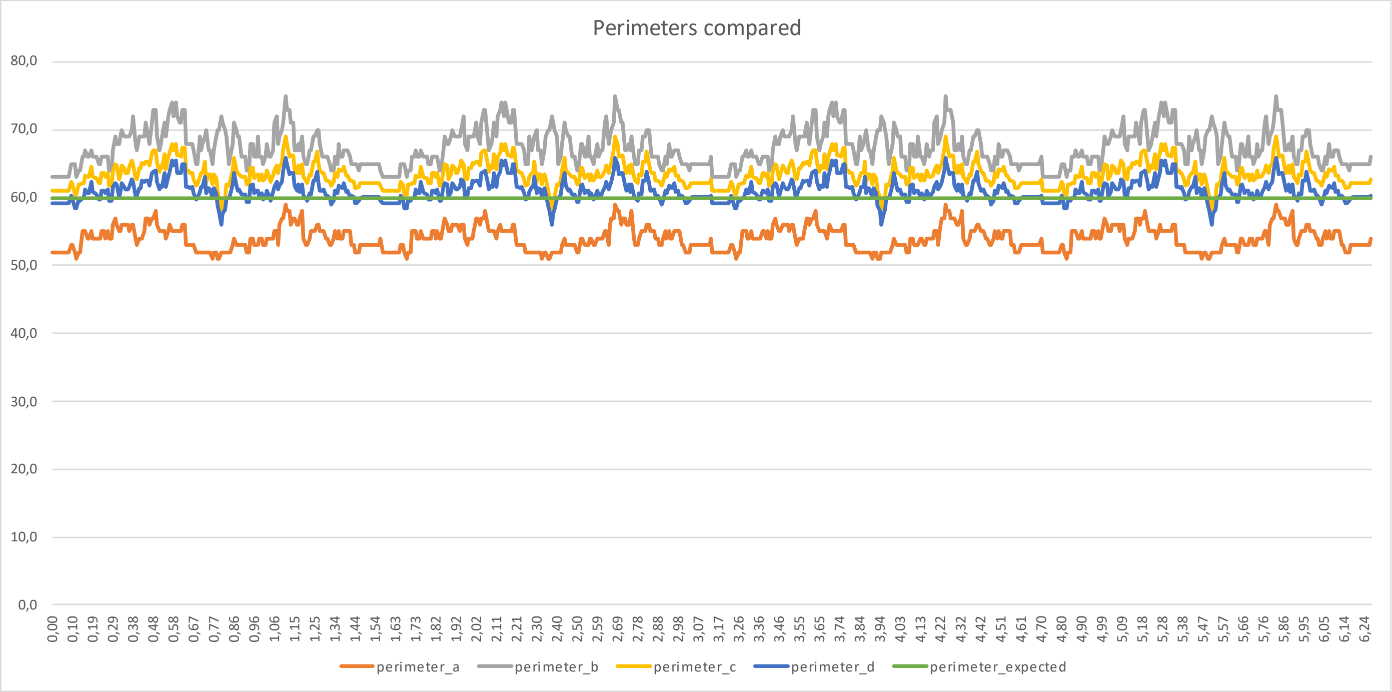
HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative d - Equilateral triangle



With $a = 20$

- $\color{green}{expected\ perimeter} = 3a = 60\ pixels$
- $\color{blue}{measured\ perimeter}$

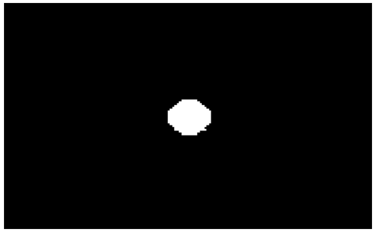


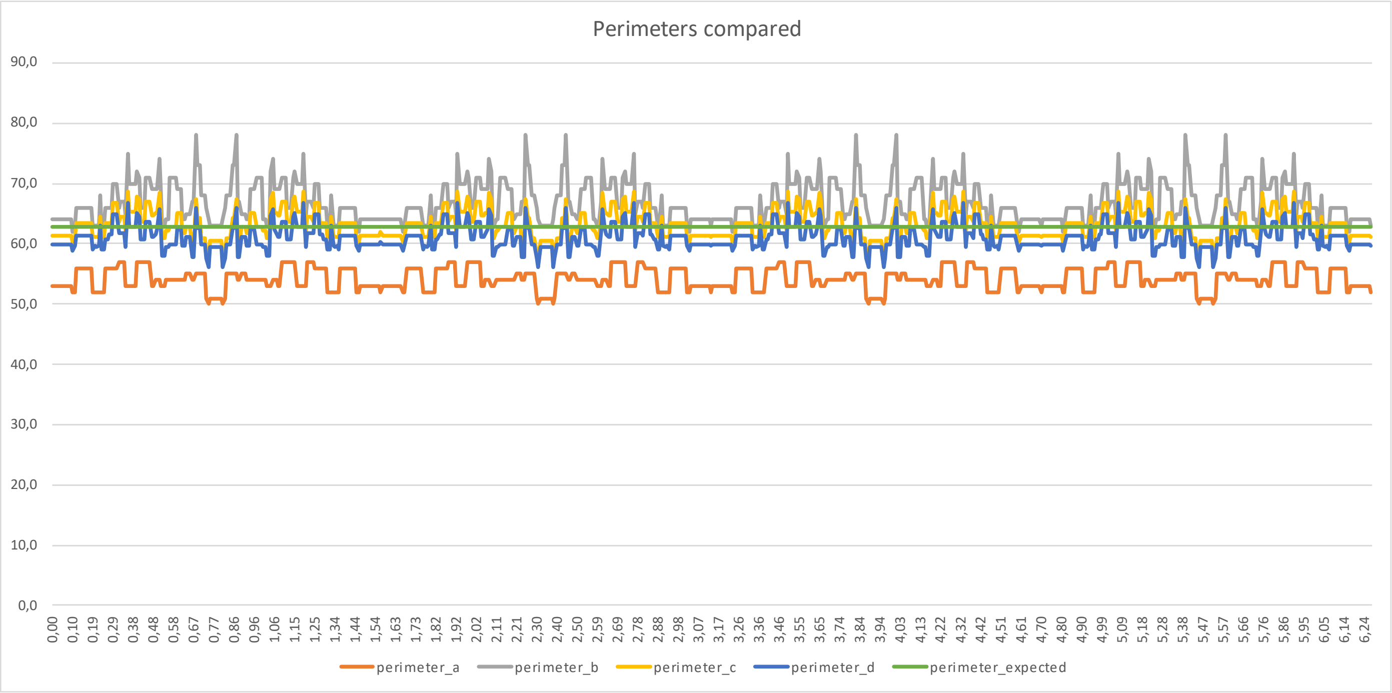
HAN_UNIVERSITY OF APPLIED SCIENCES

Perimeter alternative d - Circle

With $a = 20$

- $expected\ perimeter = 2\pi r = 63\ pixels$
- $measured\ perimeter$



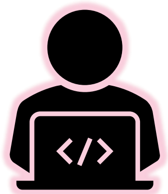


Perimeter alternative d - Conclusion

In conclusion

Shape	Perimeter measured when rotated 2π radians, especially for small objects
Square	**Shorter** than expected
Equilateral triangle	A **little longer** than expected
Circle	A **little shorter** than expected

EVD1 – Assignment



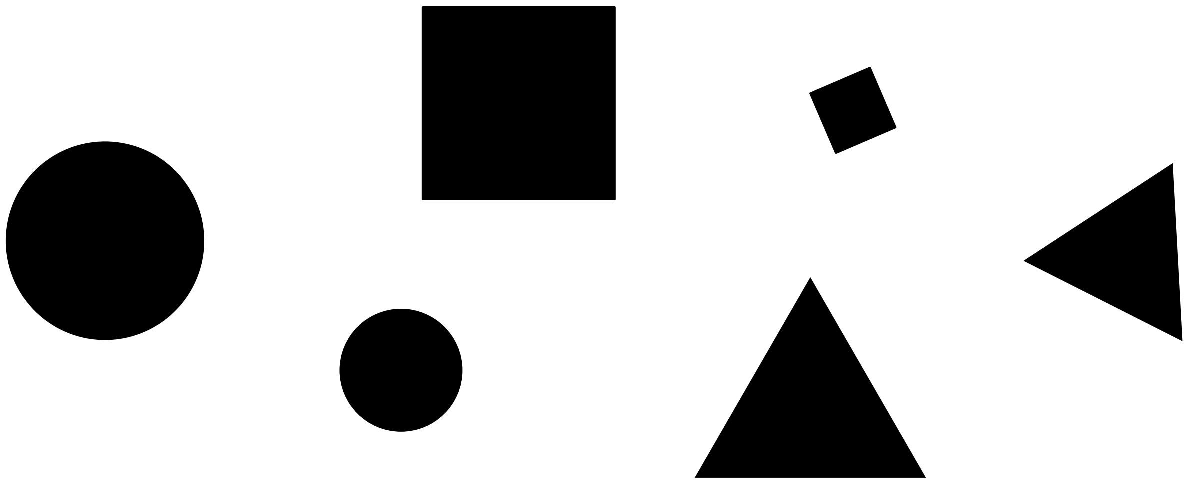
Study guide

Week 6

2 Mensuration – perimeter()

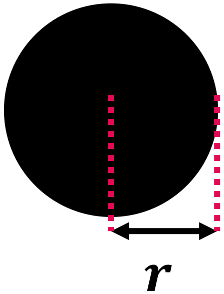
Circularity

- Circularity is a number representing an object’s roundness, no matter the:
  - Translation
  - Scaling
  - Rotation



HAN_UNIVERSITY OF APPLIED SCIENCES

Circularity - circle



- For a perfect circle we know that:

$$area = \pi r^2$$
$$perimeter = 2\pi r$$

- This means that:

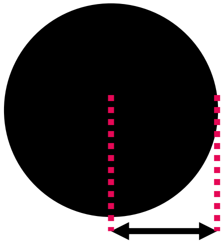
$$r = \frac{perimeter}{2\pi}$$

- Substitution gives:

$$area = \pi \left(\frac{perimeter}{2\pi}\right)^2$$

HAN_UNIVERSITY OF APPLIED SCIENCES

Circularity - circle



- Rearranging gives:

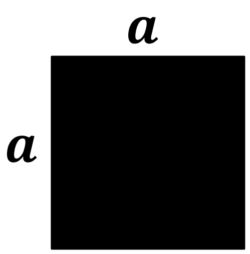
$$\frac{perimeter^2}{area} = 4\pi$$

- Conclusion: for a perfect circle, the ratio between the *perimeter squared* and the *area* is 4π

- Or, rearranging:

$$4\pi \times \frac{area}{perimeter^2} = \mathbf{1.0}$$

Circularity - square



- When dealing with a perfect square, what is the expected circularity?

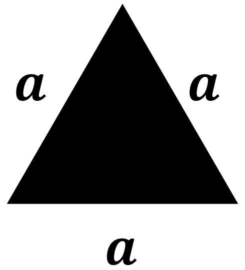
$$circularity = 4\pi \, \frac{area}{perimeter^2}$$

- Substituting:

$$circularity = 4\pi \, \frac{a \times a}{(4a)^2}$$

$$circularity = 4\pi \, \frac{a^2}{4^2 a^2} = \frac{\pi}{4} \approx \mathbf{0.7854}$$

Circularity - equilateral triangle



- And for a perfect equilateral triangle?

$$circularity = 4\pi \frac{area}{perimeter^2}$$

- Substituting gives:

$$circularity = 4\pi \frac{\frac{1}{4} \times a^2 \times \sqrt{3}}{(3a)^2}$$

- Rewriting:

$$circularity = 4\pi \frac{\frac{1}{4} \times a^2 \times \sqrt{3}}{3^2 a^2}$$

$$circularity = 4\pi \frac{\frac{1}{4} \times \sqrt{3}}{3^2} \approx \mathbf{0.6045}$$

Circularity

$$circularity = 4\pi \frac{area}{perimeter^2}$$

Shape	Theoretical circularity	Area observations, especially for small objects	Perimeter alternative d observations, especially for small objects	Practical circularity
Square	0.7854	Is as expected	**Shorter** than expected	$\gg 0.7854$
Equilateral triangle	0.6045	**Larger** than expected	A **little longer** than expected	$\gg 0.6045$
Circle	1.0	**Smaller** than expected	A **little shorter** than expected	> 1.0

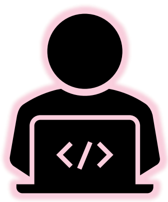
Circularity

void **circularity**(const image_t ***img**, blobinfo_t ***blobinfo**,
                                const uint32_t **blobnr**);

See file **EVDK_Operators\mensuration.c**

```
// Iterate the blobs
for(uint32_t blob=1; blob <= blobs; ++blob)
{
    blobinfo_t blobinfo = {0};

    // Get the circularity of the blob
    circularity(lbl, &blobinfo, blob);

    // ...
}
```

EVD1 – Assignment



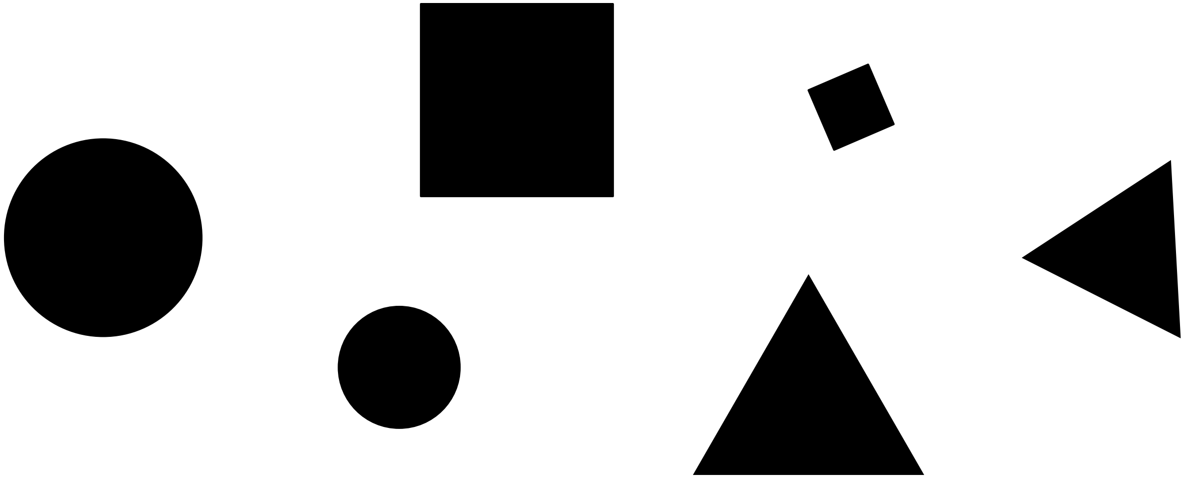
Study guide

Week 6

3 Mensuration – circularity()

Hu invariant moments

- Hu invariant moments are a set of numbers, calculated by using the normalized central moments, no matter the:
  - Translation
  - Scaling
  - Rotation



Hu invariant moments

- A set of 7 invariant moments is described by Hu
- The first two invariant moments are defined as

$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4(\eta_{11})^2$$

Hu invariant moments - definitions

Invariant Moment

$$\phi_1 = \eta_{20} + \eta_{02}$$

Normalized Central Moments

Hu invariant moments - definitions

Normalized Central Moment

$$\eta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}}$$

Central Moments

where

The (normalized) central moment's order

$$\gamma = \frac{p+q}{2} + 1 \ (with\ p+q = \ 2,3, \dots)$$

Hu invariant moments - definitions

Central Moment

$$\mu_{pq} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} (x - x_c)^p \cdot (y - y_c)^q \cdot f(x,y)$$

where

$$f(x,y) = \begin{cases} 1 & if\ (x,y) is\ part\ of\ the\ blob \\ 0 & otherwise \end{cases}$$

$$x_c{:}\ mean\ x\ value\ of\ all\ blob\ pixels\ \bar{x} = \frac{m_{10}}{m_{00}}$$

$$y_c{:}\ mean\ y\ value\ of\ all\ blob\ pixels\ \ \bar{y} = \frac{m_{01}}{m_{00}}$$

Moments

Hu invariant moments - definitions

Moment

$$m_{kl} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} x^k \cdot y^l \cdot f(x,y)$$

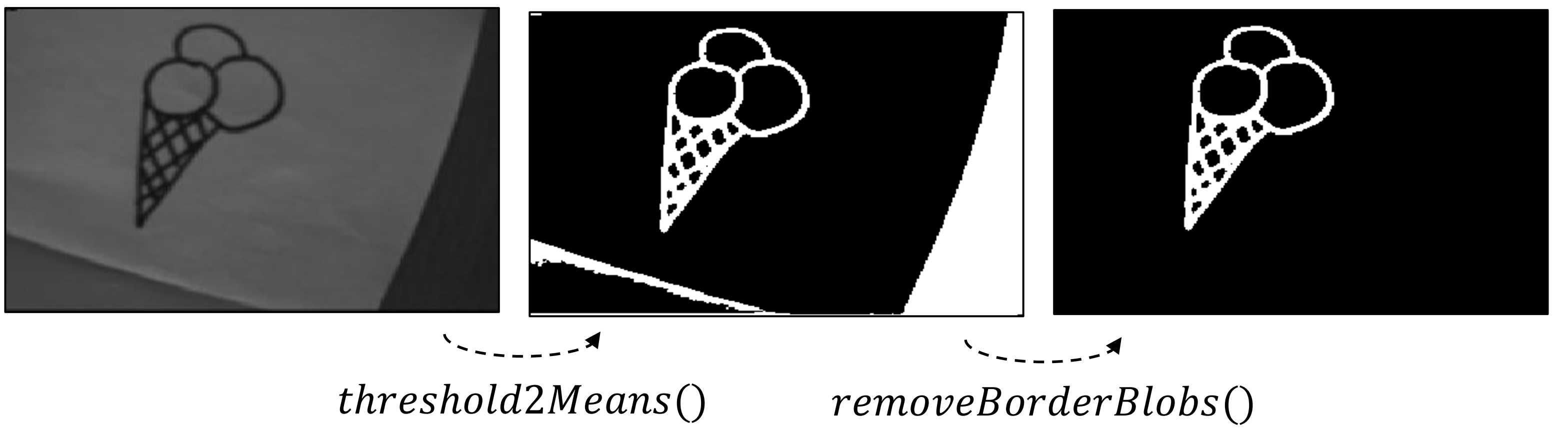
where

$$f(x,y) = \begin{cases} 1 & if\ (x,y) is\ part\ of\ the\ blob \\ 0 & otherwise \end{cases}$$

A <u>moment</u> is a measure of the distribution of pixels across an axis.

Images have 2D moments.

Example: $\boldsymbol{\phi_1 = \eta_{20} + \eta_{02}}$

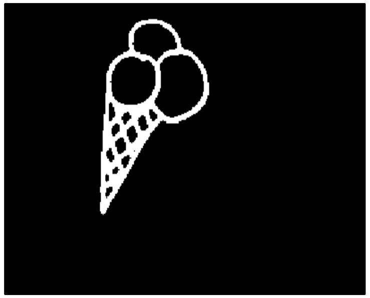


Steps to take:

1. Calculate moments $\boldsymbol{m_{00}}$, $\boldsymbol{m_{01}}$ and $\boldsymbol{m_{10}}$
2. Calculate central moments $\boldsymbol{\mu_{00}}$, $\boldsymbol{\mu_{20}}$ and $\boldsymbol{\mu_{02}}$
3. Calculate normalized central moments $\boldsymbol{\eta_{20}}$ and $\boldsymbol{\eta_{02}}$
4. Calculate invariant moment 1 $\boldsymbol{\phi_1}$

Step 1: moments $\boldsymbol{m_{00}}$, $\boldsymbol{m_{01}}$ and $\boldsymbol{m_{10}}$

Moment $\boldsymbol{m_{00}}$



$$m_{kl} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} x^k \cdot y^l \cdot f(x,y)$$

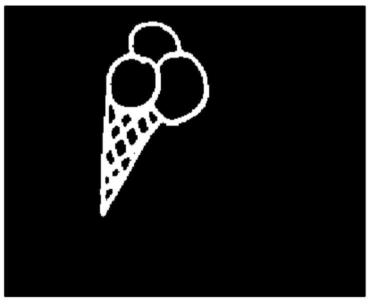
$$m_{00} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} 1 \cdot 1 \cdot f(x,y)$$

$$m_{00} = 1158$$

This is the sum of all pixels that are part of the blob, or the <u>area</u>.

Step 1: moments $\boldsymbol{m_{00}}$, $\boldsymbol{m_{01}}$ and $\boldsymbol{m_{10}}$

Moment $\boldsymbol{m_{01}}$



$$m_{kl} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} x^k \cdot y^l \cdot f(x,y)$$

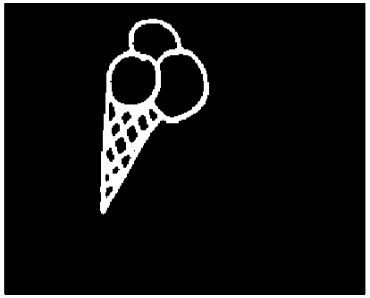
$$m_{01} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} 1 \cdot y^1 \cdot f(x,y)$$

$$m_{01} = 63159$$

This is the sum of all y-values of the pixels that are part of the blob.

Step 1: moments $\boldsymbol{m_{00}}$, $\boldsymbol{m_{01}}$ and $\boldsymbol{m_{10}}$

Moment $\boldsymbol{m_{10}}$



$$m_{kl} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} x^k \cdot y^l \cdot f(x,y)$$

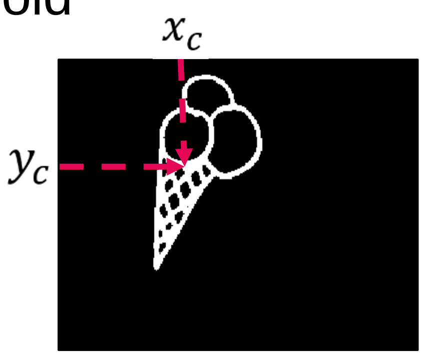
$$m_{10} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} x^1 \cdot 1 \cdot f(x,y)$$

$$m_{10} = 74269$$

This is the sum of all x-values of the pixels that are part of the blob.

Step 2: central moments $\boldsymbol{\mu_{00}}$, $\boldsymbol{\mu_{20}}$ and $\boldsymbol{\mu_{02}}$

Centroid

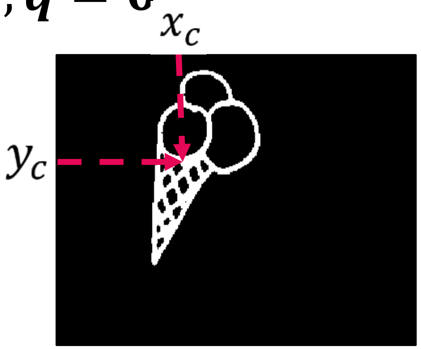


$$x_c = \frac{m_{10}}{m_{00}} = \frac{74269}{1158} = 64{,}1356$$

$$y_c = \frac{m_{01}}{m_{00}} = \frac{63159}{1158} = 54{,}5415$$

Step 2: central moments $\boldsymbol{\mu_{00}}$, $\boldsymbol{\mu_{20}}$ and $\boldsymbol{\mu_{02}}$

$\boldsymbol{p = 0, q = 0}$



$$\mu_{pq} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} (x - x_c)^p \cdot (y - y_c)^q \cdot f(x, y)$$

$$\mu_{00} = \sum_{y=0}^{y_{max}-1} \sum_{x=0}^{x_{max}-1} (x - 64{,}1356)^0 \cdot (y - 54{,}5415)^0 \cdot f(x, y)$$

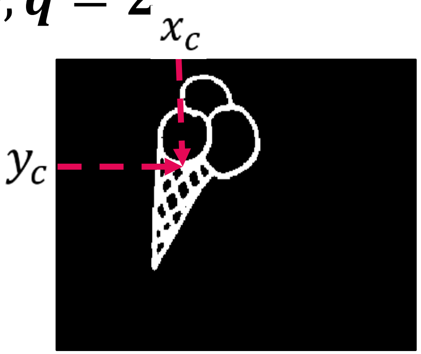
$$\mu_{00} = \sum_{y=0}^{y_{max}-1} \sum_{x=0}^{cx_{max}-1} 1 \cdot 1 \cdot f(x, y)$$

$$\mu_{00} = m_{00} = 1158$$

This is the sum of all pixel of the blob, or the <u>area</u>.

Step 2: central moments $\boldsymbol{\mu_{00}}$, $\boldsymbol{\mu_{20}}$ and $\boldsymbol{\mu_{02}}$

$\boldsymbol{p = 0, q = 2}$



$$\mu_{pq} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} (x - x_c)^p \cdot (y - y_c)^q \cdot f(x, y)$$

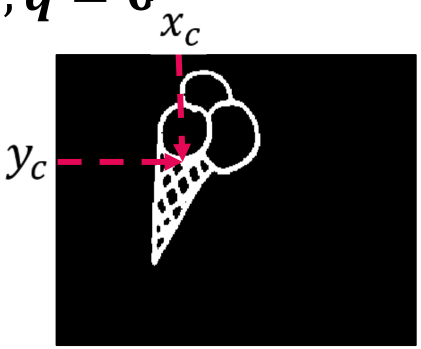
$$\mu_{02} = \sum_{y=0}^{y_{max}-1} \sum_{x=0}^{x_{max}-1} (x - 64{,}1356)^0 \cdot (y - 54{,}5415)^2 \cdot f(x, y)$$

$$\mu_{02} = 620666$$

HAN_UNIVERSITY
OF APPLIED SCIENCES

Step 2: central moments $\boldsymbol{\mu_{00}}$, $\boldsymbol{\mu_{20}}$ and $\boldsymbol{\mu_{02}}$

$\boldsymbol{p = 2, q = 0}$



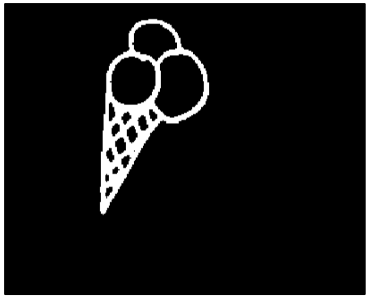
$$\mu_{pq} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} (x - x_c)^p \cdot (y - y_c)^q \cdot f(x,y)$$

$$\mu_{20} = \sum_{y=0}^{y_{max}-1} \sum_{x=0}^{x_{max}-1} (x - 64{,}1356)^2 \cdot (y - 54{,}5415)^0 \cdot f(x,y)$$

$$\mu_{20} = 231070$$

Step 3: normalized central moments $\boldsymbol{\eta_{20}}$ and $\boldsymbol{\eta_{02}}$

$\boldsymbol{p = 0, q = 2}$



$$\eta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}}$$

$$\gamma = \frac{p+q}{2} + 1 \; (with\, p + q = \; 2,3, \ldots)$$

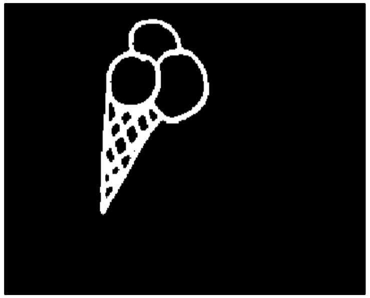
$$\gamma = \frac{0+2}{2} + 1 = 2$$

$$\eta_{02} = \frac{\mu_{02}}{(\mu_{00})^2}$$

$$\eta_{02} = \frac{620666}{(1158)^2} = 0.46285$$

Step 3: normalized central moments $\boldsymbol{\eta_{20}}$ and $\boldsymbol{\eta_{02}}$

$\boldsymbol{p = 2, q = 0}$



$$\eta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}}$$

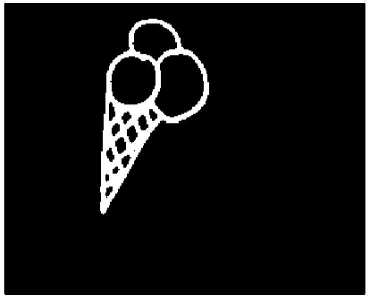
$$\gamma = \frac{p+q}{2} + 1 \ (with\ p + q = \ 2,3, \dots)$$

$$\gamma = \frac{2+0}{2} + 1 = 2$$

$$\eta_{20} = \frac{\mu_{20}}{(\mu_{00})^2}$$

$$\eta_{20} = \frac{231070}{(1158)^2} = 0.172316$$

Step 4: invariant moment 1 - $\boldsymbol{\phi_1}$

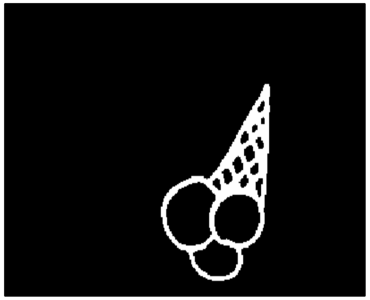


$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_1 = 0.170215 \; + \; 0.46098$$

$$\phi_1 = 0.631202$$

Step 4: invariant moment 1 - $\boldsymbol{\phi_1}$

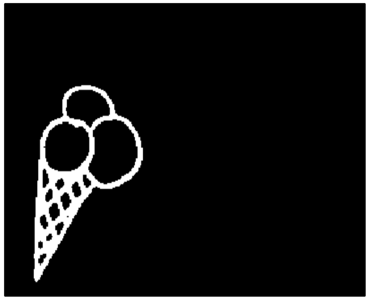


$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_1 = 0.167503 \ + \ 0.456256$$

$$\phi_1 = 0.623759$$

Step 4: invariant moment 1 - $\boldsymbol{\phi_1}$

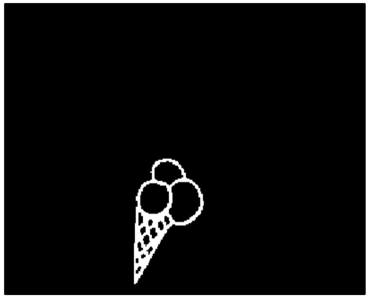


$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_1 = 0.170215 \; + \; 0.460987$$

$$\phi_1 = 0.631202$$

Step 4: invariant moment 1 - $\boldsymbol{\phi_1}$



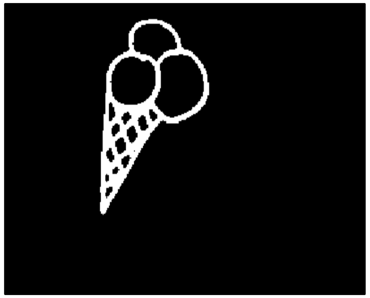
$$\phi_1 = \eta_{20} + \eta_{02}$$

$$\phi_1 = 0.166050 \ + \ 0.450186$$

$$\phi_1 = \ 0.61623$$

Special consideration

$\boldsymbol{p = 0, q = 0}$



$$\eta_{pq} = \frac{\mu_{pq}}{(\mu_{00})^{\gamma}}$$

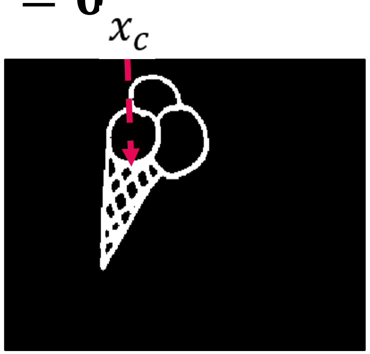
$$\gamma = \frac{p+q}{2} + 1 \ (with\ p+q = \ 2,3, \dots)$$

$$\gamma = \frac{0+0}{2} + 1 = 1$$

$$\eta_{00} = \frac{\mu_{00}}{(\mu_{00})^{1}} = 1.0$$

Special consideration

$\boldsymbol{p = 1, q = 0}$



$$\mu_{pq} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} (x - x_c)^p \cdot (y - y_c)^q \cdot f(x, y)$$

$$\mu_{10} = \sum_{y=0}^{y_{max}-1} \sum_{x=0}^{x_{max}-1} (x - x_c)^1 \cdot 1 \cdot f(x, y)$$

This is the sum of all distances to $\boldsymbol{x_c}$

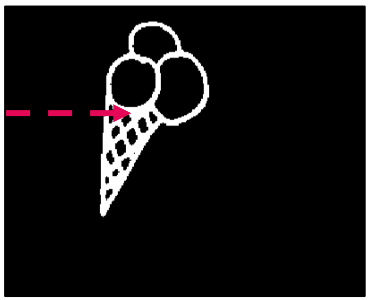
However, as x_c is the centre pixel, the sum of all pixels to the left must be equal to the sum of all pixels to the right

$$\begin{gathered}\mu_{10} = \{(60 - 64) + (61 - 64) \dots (67 - 64) + (68 - 64)\} \\ = \{(-4) + (-3) + \cdots + (3) + (4)\} \\ = 0\end{gathered}$$

Special consideration

$\boldsymbol{p = 0, q = 1}$

y_c



$$\mu_{pq} = \sum_{x=0}^{x_{max}-1} \sum_{y=0}^{y_{max}-1} (x - x_c)^p \cdot (y - y_c)^q \cdot f(x, y)$$

$$\mu_{10} = \sum_{y=0}^{y_{max}-1} \sum_{x=0}^{x_{max}-1} 1 \cdot (y - y_c)^1 \cdot f(x, y)$$

This is the sum of all distances to $\boldsymbol{y_c}$

However, as $\boldsymbol{y_c}$ is the centre pixel, the sum of all pixels above must be equal to the sum of all pixels below

$$\begin{aligned} \mu_{01} &= \{(53 - 55) + (54 - 55) \ldots (56 - 55) + (57 - 55)\} \\ &= \{(-2) + (-1) + \cdots + (1) + (2)\} \\ &= 0 \end{aligned}$$

HAN_UNIVERSITY
OF APPLIED SCIENCES

Hu invariant moments - examples

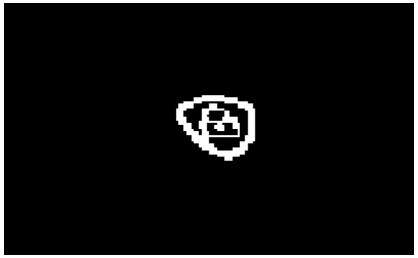
im1: 0.366
im2: 0.012
im3: 0.000
im4: 0.000

im1: 0.366
im2: 0.012
im3: 0.000
im4: 0.000

im1: 0.366
im2: 0.012
im3: 0.000
im4: 0.000

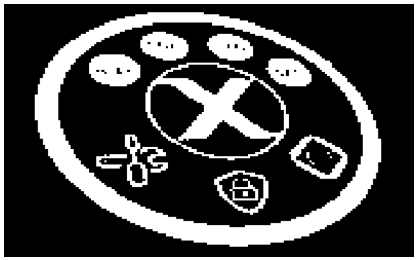
im1: 0.366
im2: 0.011
im3: 0.000
im4: 0.000

im1: 0.360
im2: 0.012
im3: 0.000
im4: 0.000

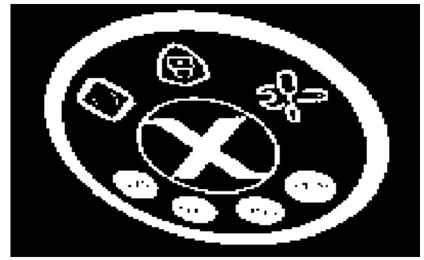


im1: 0.349
im2: 0.003
im3: 0.002
im4: 0.000

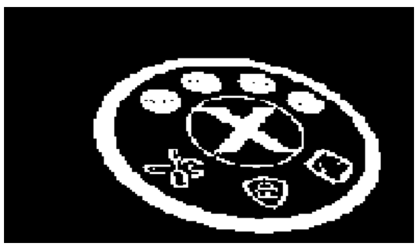
Hu invariant moments - examples



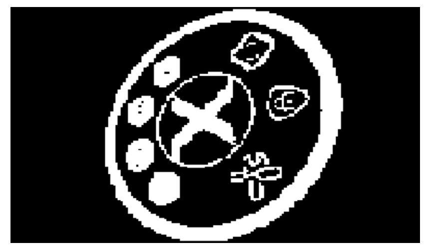
im1: 0.515
im2: 0.015
im3: 0.000
im4: 0.001



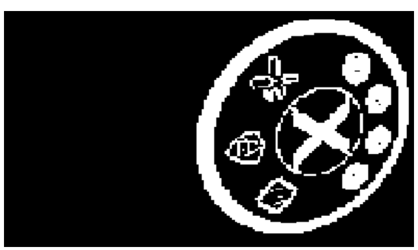
im1: 0.515
im2: 0.015
im3: 0.000
im4: 0.001



im1: 0.514
im2: 0.016
im3: 0.000
im4: 0.000



im1: 0.509
im2: 0.018
im3: 0.000
im4: 0.001



im1: 0.502
im2: 0.020
im3: 0.000
im4: 0.000

im1: 0.366
im2: 0.012
im3: 0.000
im4: 0.000

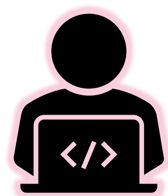
Hu invariant moments - algorithm

void **hu_moments**(const image_t ***img**, blobinfo_t ***blobinfo**,
const uint32_t **blobnr**);

See file **EVDK_Operators\mensuration.c**

```
// Iterate the blobs
for(uint32_t blob=1; blob <= blobs; ++blob)
{
    blobinfo_t blobinfo = {0};

    // Get the Hu invariant moments of the blob
    hu_moments(lbl, &blobinfo, blob);

    // ...
}
```

EVD1 – Assignment



Study guide

Week 6

4 Mensuration – hu_moments()

References

- Myler, H. R., & Weeks, A. R. (2009). *The pocket handbook of image processing algorithms in C*. Prentice Hall Press.

- Benkrid, K., Crookes, D., & Benkrid, A. (2000, September). Design and FPGA implementation of a perimeter estimator. In *Proceedings of the Irish Machine Vision and Image Processing Conference* (pp. 51-57).

- Gonzalez, R. (). 11.3.4 Moment Invariants. In *Digital Image Processing.* pp. 839-842. New Jersey: Pearson Prentice Hall.

- Haralick, R. M. (1981). Some neighborhood operators. In *Real-Time Parallel Computing: Imaging Analysis* (pp. 11-35). Boston, MA: Springer US.