HAN AEA - Embedded Vision & Machine Learning

EVD1 - Week 3

Image Fundamentals
Nonlinear Filters
Spatial Filters

By Hugo Arends

HAN_UNIVERSITY OF APPLIED SCIENCES

Image Fundamentals

- Functions for creating and deleting images
- Functions for converting images
- Functions for reading and writing pixels
- Basic image processing operators

- Discrete convolution
- Discrete correlation

Discrete convolution

- Results in a filtering operation
- Applies a filter mask to an image by convolving the filter mask with the original image
- Two-dimensional discrete convolution is defined as

$$p_{dst}(x,y) = \sum_{i=-n/2}^{i=n/2} \sum_{j=-m/2}^{j=m/2} p_{src}(x-i, y-j) \cdot p_{mask}(i + n/2, j + m/2)$$

where
$m \times n$: $mask\ size$
$p_{dst}(x,y)$: $the\ calculated\ result\ pixel\ in\ the\ destination\ image\ at\ (x,y)$
$p_{src}(x-i, y-j)$: $a\ pixel\ value\ in\ the\ src\ image\ within\ the\ mask$
$p_{mask}(i + n/2, j + m/2)$: $a\ pixel\ value\ in\ the\ mask\ image$

Discrete convolution

- Results in a filtering operation
- Applies a filter mask to an image by convolving the filter mask with the original image
- Two-dimensional discrete convolution is defined as

$$p_{dst}(x,y) = \sum_{i=-n/2}^{i=n/2} \sum_{j=-m/2}^{j=m/2} \underbrace{p_{src}(x-i, y-j)}_{\substack{\text{All pixels in the} \\ m \times n \\ \text{neighbourhood} \\ \text{of the src pixel at} \\ (x,y)}} \cdot p_{mask}(i + n/2, j + m/2)$$

HAN_UNIVERSITY OF APPLIED SCIENCES

Discrete convolution

- Results in a filtering operation
- Applies a filter mask to an image by convolving the filter mask with the original image
- Two-dimensional discrete convolution is defined as

$$p_{dst}(x,y) = \sum_{i=-n/2}^{i=n/2} \sum_{j=-m/2}^{j=m/2} p_{src}(x-i, y-j) \cdot \underbrace{p_{mask}(i + n/2, j + m/2)}_{\text{All pixels in the } m \times n \text{ mask}}$$

HAN_UNIVERSITY OF APPLIED SCIENCES

Discrete convolution

- The corresponding pixels in the src image and mask are **flipped** in both horizontal and vertical direction.

Mask

a	b	c
d	e	f
g	h	i

Flipped mask

i	h	g
f	e	d
c	b	a

- If the mask is not symmetrical in both horizontal and vertical direction, the mask should be flipped before performing a convolution.

No need to flip

0	-1	0
-1	5	-1
0	-1	0

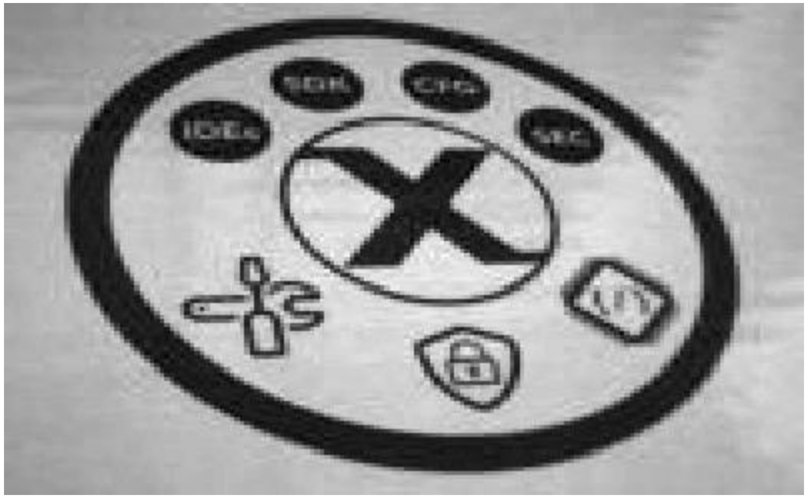
Must be flipped

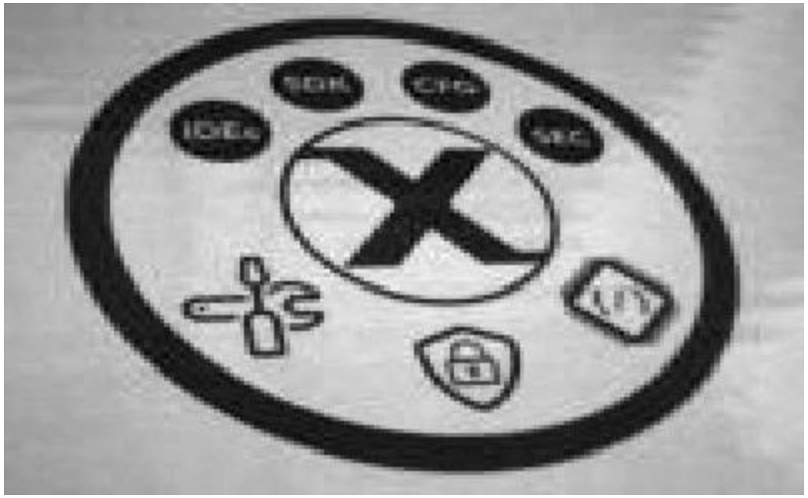
0	0	1
0	1	0
0	1	0

Discrete convolution - example

Identity

0	0	0
0	1	0
0	0	0

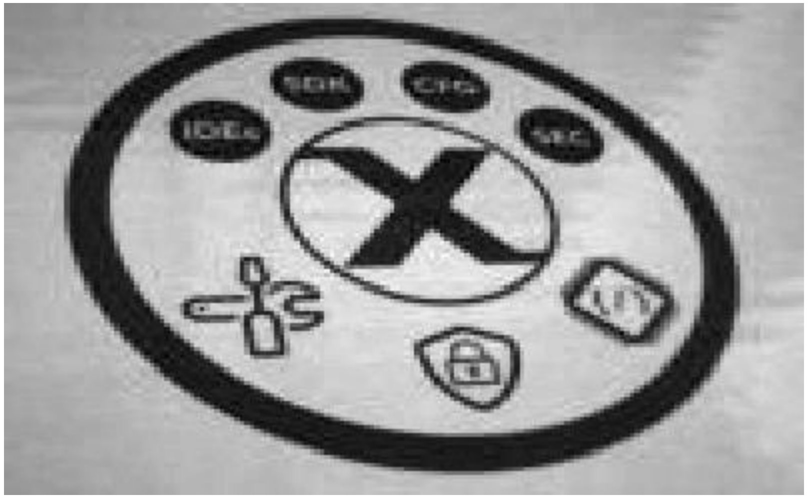


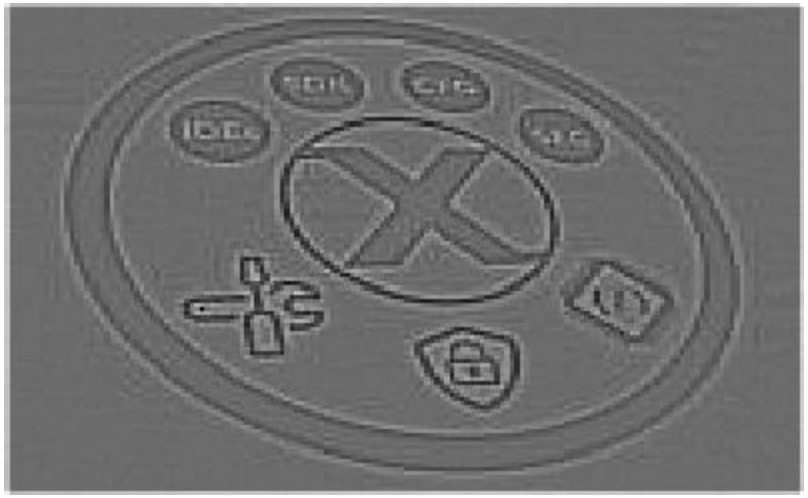


Discrete convolution - example

Edge

-1	-1	-1
-1	8	-1
-1	-1	-1

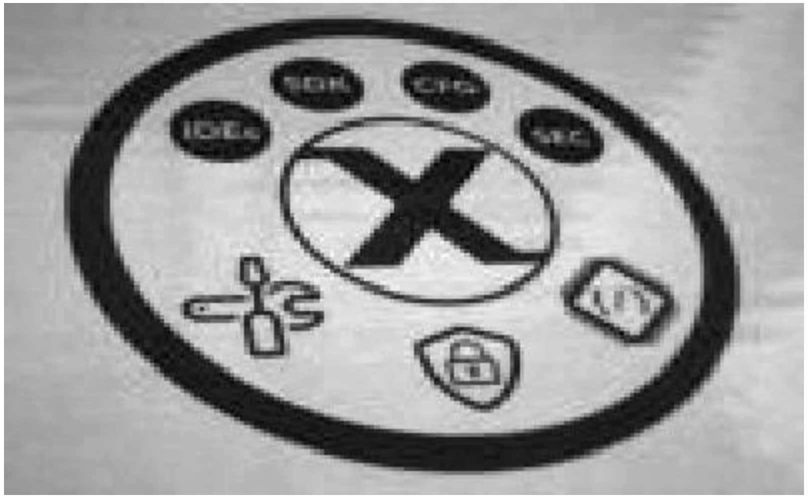


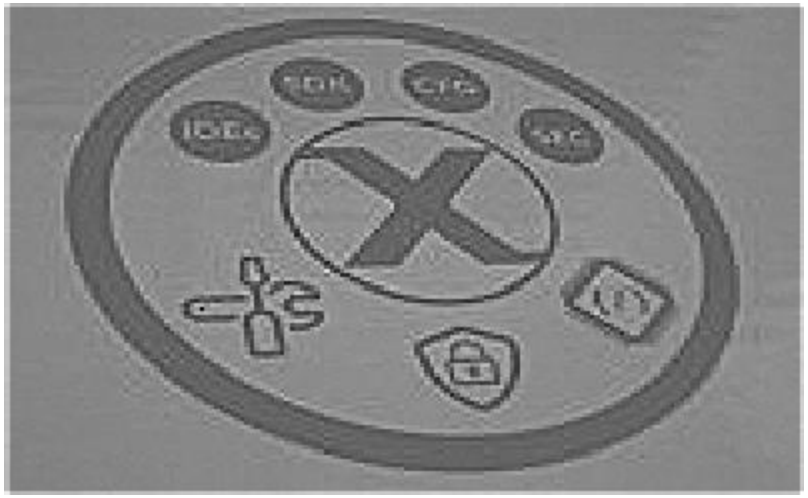


Discrete convolution - example

Sharpen

0	-1	0
-1	5	-1
0	-1	0





Discrete convolution - implementation

```
void convolve(  const image_t *src, image_t *dst,
                const image_t *msk);
```

See file **EVDK_Operators\image_fundamentals.c**

Discrete convolution - implementation

```
void convolve(const image_t *src, image_t *dst, const image_t *msk)
{
    // Loop all pixels
    for(int32_t y=0; y<src->rows; y++)
    {
        for(int32_t x=0; x<src->cols; x++)
        {
```


HAN_UNIVERSITY OF APPLIED SCIENCES

Discrete convolution - implementation

```
int32_t val = 0;
int32_t dr = (msk->rows/2);
int32_t dc = (msk->cols/2);

// Apply the kernel only for pixels within the image
for(int32_t j=-dr; j<=dr; j++)
{
    for(int32_t i=-dc; i<=dc; i++)
    {
        if((x-i) >= 0 &&
           (y-j) >= 0 &&
           (x-i) <  src->cols &&
           (y-j) <  src->rows)
        {
            val += getInt16Pixel(src,x-i,y-j) * getInt16Pixel(msk,i+dc,j+dr);
        }
    }
}
```

Discrete convolution - implementation

```
            // Clip the result
            if(val>INT16_PIXEL_MAX)
                 val=INT16_PIXEL_MAX;

            if(val<INT16_PIXEL_MIN)
                 val=INT16_PIXEL_MIN;

            // Store the result
            setInt16Pixel(dst, x, y, val);
        }
    }
}
```

Discrete convolution - implementation

```
            // Clip the result
            if(val>INT16_PIXEL_MAX)
                 val=INT16_PIXEL_MAX;

            if(val<INT16_PIXEL_MIN)
                 val=INT16_PIXEL_MIN;

            // Store the result
            setInt16Pixel(dst, x, y, val);
        }
    }
}
```

Optimize most (-O3)

UVC connected

~35 ms

Discrete convolution – improve performance

- Assume a 3x3 mask and ignore border pixels

```
// Apply the kernel only for pixels within the image
for(int32_t j=-dr; j<=dr; j++)
{
    for(int32_t i=-dc; i<=dc; i++)
    {
        if((x-i) >= 0 &&
           (y-j) >= 0 &&
           (x-i) <  src->cols &&
           (y-j) <  src->rows)
        {
            val += getInt16Pixel(src,x-i,y-j) * getInt16Pixel(msk,i+dc,j+dr);
        }
    }
}
```

No loops needed with variable delta r and delta c (loop unrolling)

Discrete convolution – improve performance

- Assume a 3x3 mask and ignore border pixels

```
// Apply the kernel only for pixels within the image
for(int32_t j=-dr; j<=dr; j++)
{
    for(int32_t i=-dc; i<=dc; i++)
    {
        if((x-i) >= 0 &&
           (y-j) >= 0 &&
           (x-i) <  src->cols &&
           (y-j) <  src->rows)
        {
            val += getInt16Pixel(src,x-i,y-j) * getInt16Pixel(msk,i+dc,j+dr);
        }
    }
}
```

No loops needed with variable delta r and delta c (loop unrolling)

No need to check image boundaries for every pixel

Discrete convolution – improve performance

- Assume a 3x3 mask and ignore border pixels
- Avoid using getters and setters

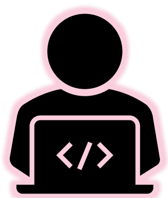
```
// Apply the kernel only for pixels within the image
for(int32_t j=-dr; j<=dr; j++)
{
    for(int32_t i=-dc; i<=dc; i++)
    {
        if((x-i) >= 0 &&
           (y-j) >= 0 &&
           (x-i) <  src->cols &&
           (y-j) <  src->rows)
        {
            val += getInt16Pixel(src,x-i,y-j) * getInt16Pixel(msk,i+dc,j+dr);
        }
    }
}
```

No loops needed with variable delta r and delta c (loop unrolling)

No need to check image boundaries for every pixel

Use pointers instead

EVD1 – Assignment



Study guide

Week 3

2 Image fundamentals – convolveFast()

Discrete correlation

- Compares two images mathematically
- The result is a two-dimensional expression of equivalence
- The mask image is often referred to as the template
- The correlation is then called template matching
- Two-dimensional discrete correlation is defined as

$$p_{dst}(x,y) = \sum_{i=-n/2}^{i=n/2} \sum_{j=-m/2}^{j=m/2} p_{src}(x+i, y+j) \cdot p_{mask}(i + n/2, j + m/2)$$

where

$m \times n$: $mask\ size$

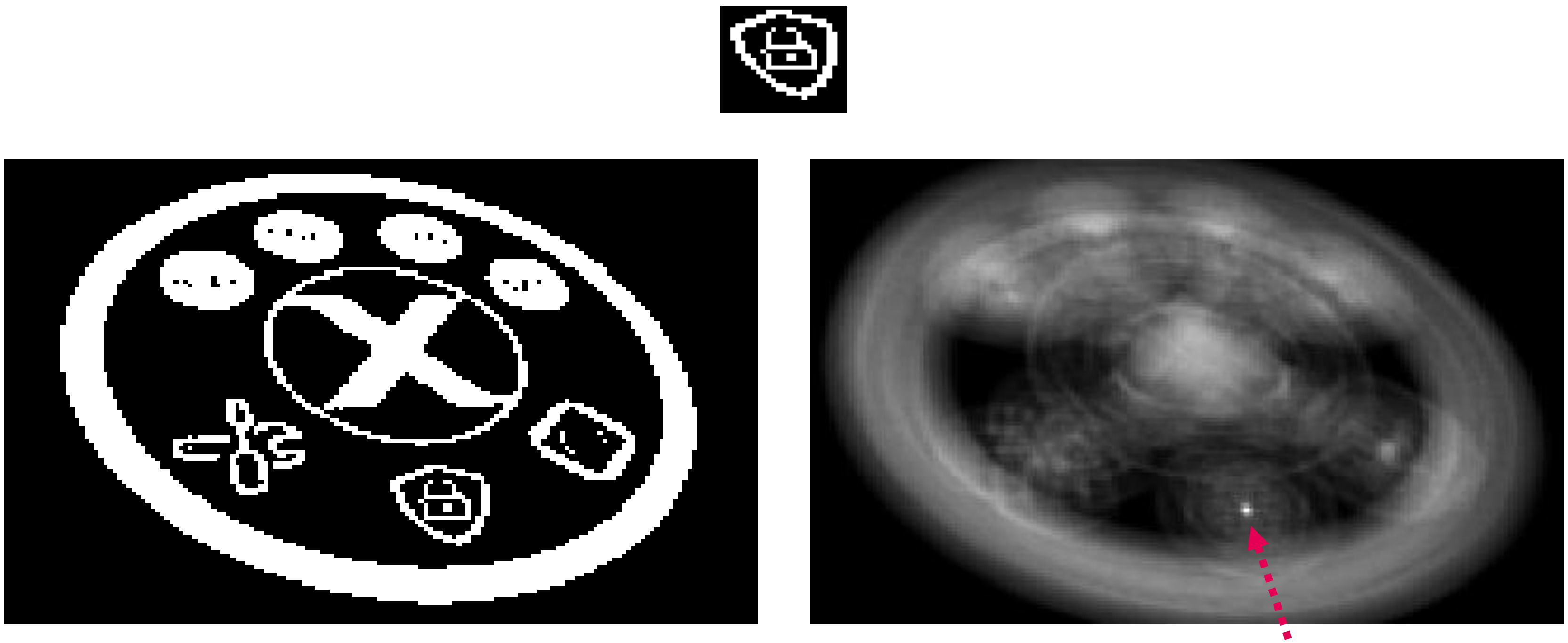
$p_{dst}(x,y)$: $the\ calculated\ result\ \ pixel\ in\ the\ destination\ image\ at\ (x,y)$

$p_{src}(x+i, y+j)$: $a\ pixel\ value\ in\ the\ src\ image\ within\ the\ mask$

$p_{mask}(i + n/2, j + m/2)$: $a\ pixel\ value\ in\ the\ mask\ image$

Discrete correlation – example

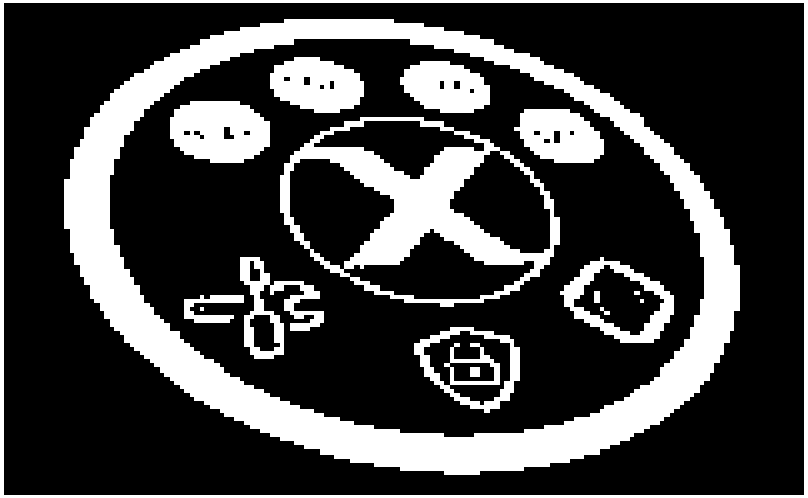
Binary template matching

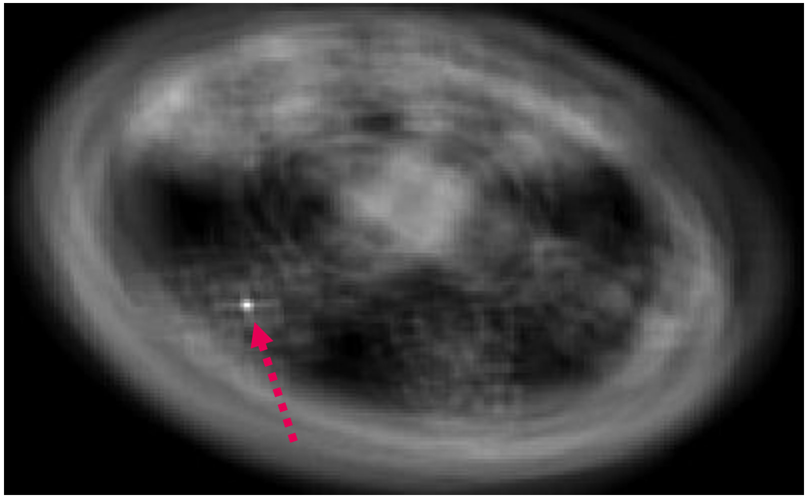


HAN_UNIVERSITY OF APPLIED SCIENCES

Discrete correlation – example

Binary template matching





HAN_UNIVERSITY OF APPLIED SCIENCES

Discrete correlation – implementation

void **correlate**(const image_t ***src**, image_t ***dst**,
const image_t ***msk**);

See file **EVDK_Operators\image_fundamentals.c**

Discrete correlation – implementation

```
val += getInt16Pixel(src,x+i,y+j) * getInt16Pixel(msk,i+dc,j+dr);
```

Optimize most (-O3)

UVC connected

25x25 mask

> 1000 ms

Discrete correlation – improve performance

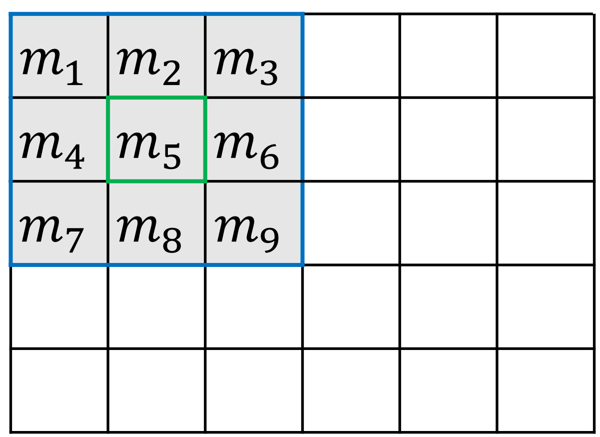
- For template matching it does not make sense to assume a 3x3 mask, because the size of a mask is at least tens by tens pixels
- Although the border pixels can be skipped, if the mask size increases, so are the number of skipped pixels
- In other words, a template matcher with a reasonable sized mask takes too long to execute on the microcontroller

Nonlinear filters

- Operate on an image by computing a given nonlinear function over a local window
- The local window can vary in size, is often a square
- Replace one specified pixel within the local window with the computed value, often the centre pixel (hence window size is an odd value)
- Are not solely used for filtering, e.g. binary erosion, binary dilation, and edge detection

Image

$window\ size\ (n) = 3$

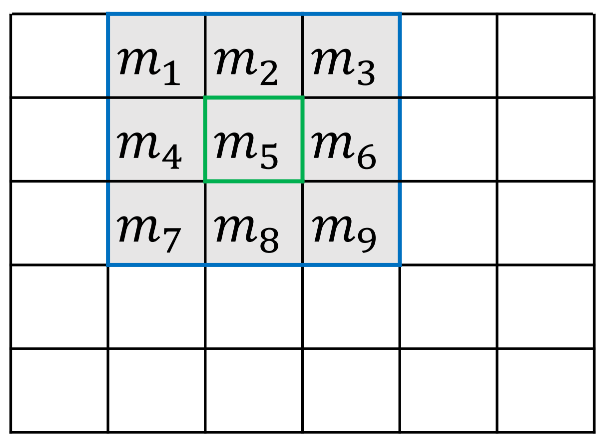


Nonlinear filters

- Operate on an image by computing a given nonlinear function over a local window
- The local window can vary in size, is often a square
- Replace one specified pixel within the local window with the computed value, often the centre pixel (hence window size is an odd value)
- Are not solely used for filtering, e.g. binary erosion, binary dilation, and edge detection

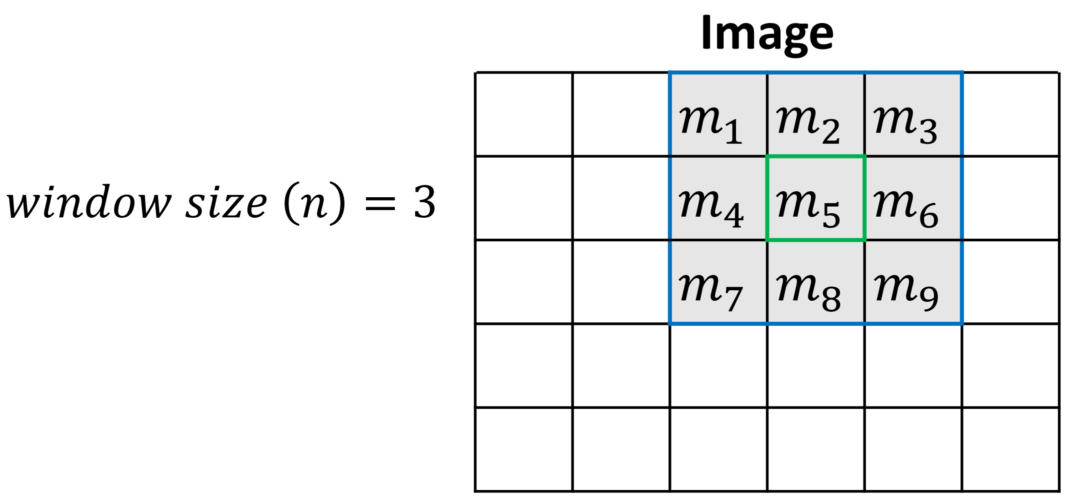
Image

$window\ size\ (n) = 3$



Nonlinear filters

- Operate on an image by computing a given nonlinear function over a local window
- The local window can vary in size, is often a square
- Replace one specified pixel within the local window with the computed value, often the centre pixel (hence window size is an odd value)
- Are not solely used for filtering, e.g. binary erosion, binary dilation, and edge detection

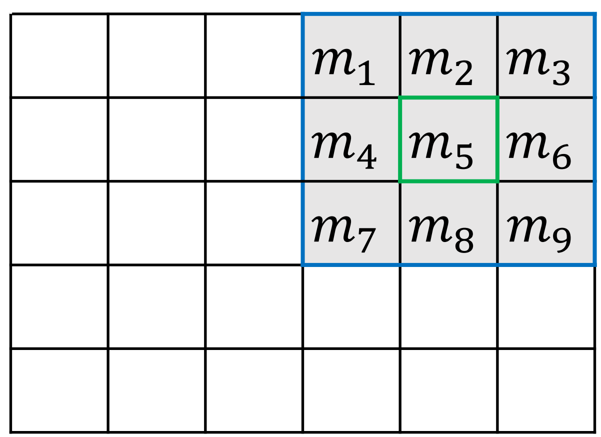


Nonlinear filters

- Operate on an image by computing a given nonlinear function over a local window
- The local window can vary in size, is often a square
- Replace one specified pixel within the local window with the computed value, often the centre pixel (hence window size is an odd value)
- Are not solely used for filtering, e.g. binary erosion, binary dilation, and edge detection

Image

$window\ size\ (n) = 3$

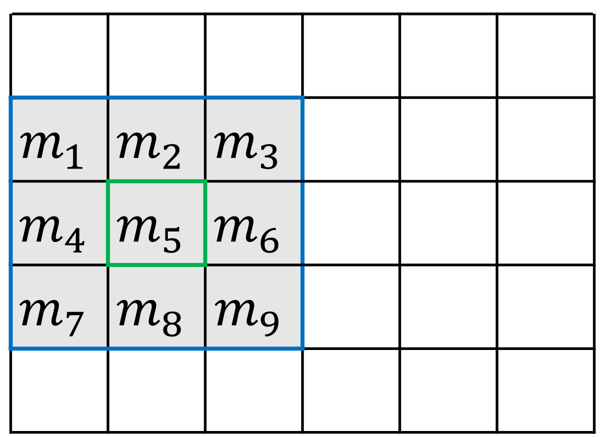


Nonlinear filters

- Operate on an image by computing a given nonlinear function over a local window
- The local window can vary in size, is often a square
- Replace one specified pixel within the local window with the computed value, often the centre pixel (hence window size is an odd value)
- Are not solely used for filtering, e.g. binary erosion, binary dilation, and edge detection

Image

$window\ size\ (n) = 3$

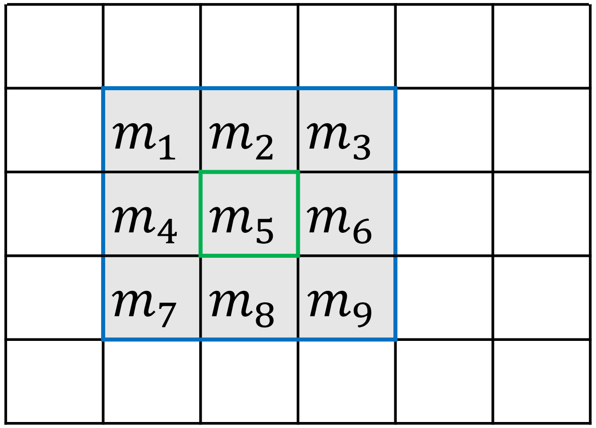


Nonlinear filters

- Operate on an image by computing a given nonlinear function over a local window
- The local window can vary in size, is often a square
- Replace one specified pixel within the local window with the computed value, often the centre pixel (hence window size is an odd value)
- Are not solely used for filtering, e.g. binary erosion, binary dilation, and edge detection

Image

$$window\ size\ (n) = 3$$



Nonlinear filters – implementation

Implementation is very similar to convolution

Handles border pixels by just taking the valid pixels into account in the filter specific operation

Nonlinear filters - implementation

```
void mean(const image_t *src, image_t *dst, const uint8_t n)
{
    // Loop all pixels
    for(int32_t y=0; y<src->rows; y++)
    {
        for(int32_t x=0; x<src->cols; x++)
        {
```


HAN_UNIVERSITY OF APPLIED SCIENCES

Nonlinear filters - implementation

```
// Initialize filter specific variables
int32_t cnt = 0;
int32_t sum = 0;

// Apply the kernel only for pixels within the image
for(int32_t j=-n/2; j<=n/2; j++)
{
    for(int32_t i=-n/2; i<=n/2; i++)
    {
        if((x+i) >= 0 &&
           (y+j) >= 0 &&
           (x+i) <  src->cols &&
           (y+j) <  src->rows)
        {
            // Count the number of pixels in the calculation
            cnt++;

            // Get pixel and perform filter specific calculation
            sum += getUint8Pixel(src,x+i,y+j);
        }
    }
}
```

Nonlinear filters - implementation

```
            // Calculate and store the result
            setUint8Pixel(dst,x,y,(uint8_pixel_t)((float)sum/(float)cnt + 0.5f));
        }
    }
}
```


HAN_UNIVERSITY OF APPLIED SCIENCES

Nonlinear filters - arithmetic mean

- Calculates the arithmetic mean of the pixels within the window
- The arithmetic mean is defined as

$$p_{dst}(x,y) = \frac{1}{n^2} \sum_{i=-n/2}^{i=n/2} \sum_{j=-n/2}^{j=n/2} p_{src}(x+i, y+j)$$

where
$p_{dst}(x,y)$: $the\ calculated\ result\ \ pixel\ in\ the\ destination\ image\ at\ (x,y)$
$p_{src}(x+i,y+j)$: $a\ pixel\ value\ in\ the\ src\ image\ within\ the\ window$
n: $the\ window\ size$

Nonlinear filters - arithmetic mean

void **mean**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);

See file **EVDK_Operators\nonlinear_filters.c**

Nonlinear filters - arithmetic mean

void **mean**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);

See file **EVDK_Operators\nonlinear_filters.c**

```
// ----------------------------------------------------------------------------
// Image processing pipeline
// ----------------------------------------------------------------------------

convertBgr888ToUint8(img, src);

mean(src, dst, 3);
```

Nonlinear filters - arithmetic mean

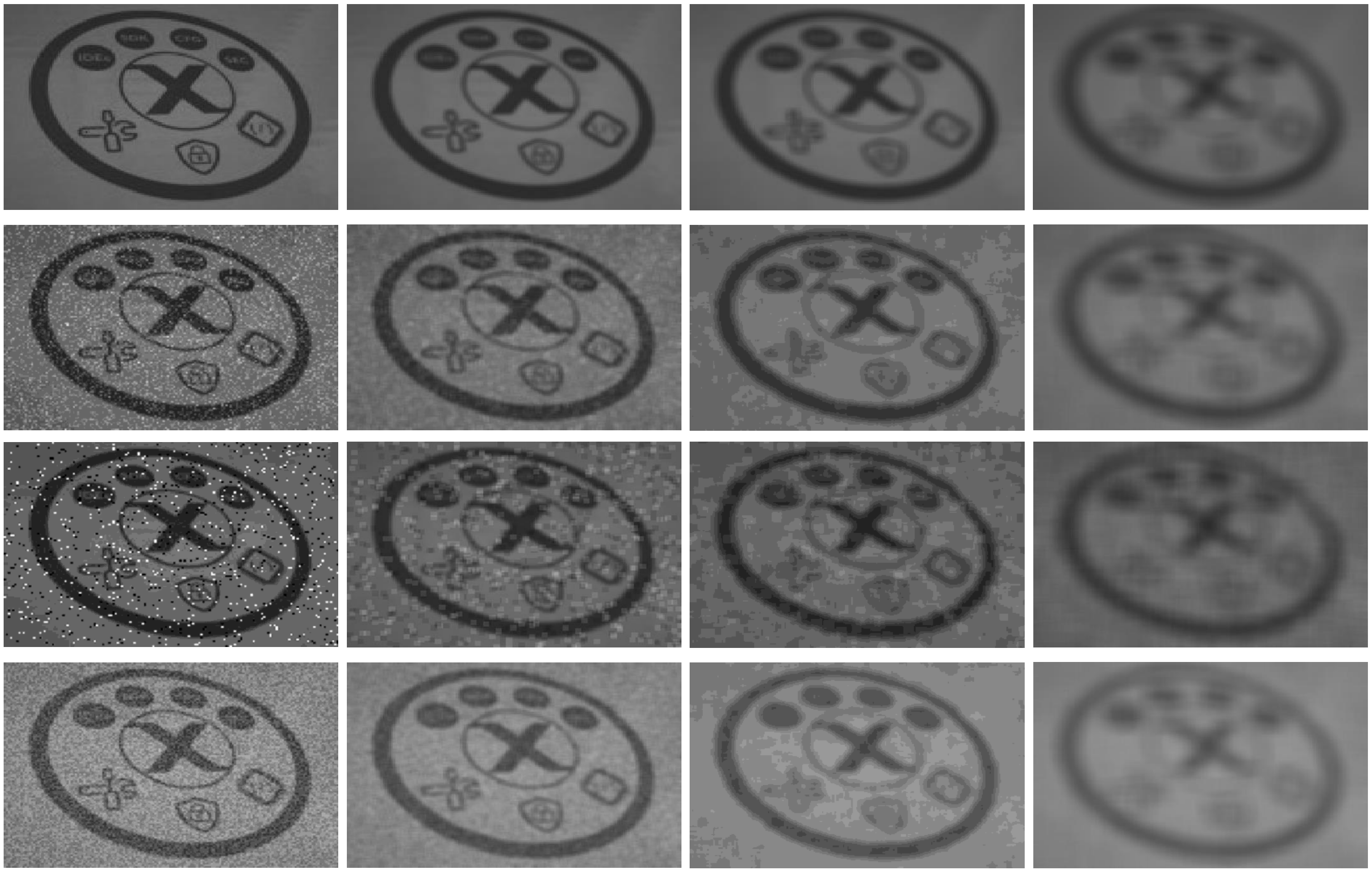
void **mean**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);

See file **EVDK_Operators\nonlinear_filters.c**

Optimize most (-O3)
UVC connected
3x3 mask: **56 ms**
5x5 mask: **136 ms**
11x11 mask: **594 ms**

HAN_UNIVERSITY OF APPLIED SCIENCES

Nonlinear filters - arithmetic mean examples



Original image, without (additional) noise

Gaussian noise
$m = 0$
$\sigma = 800$

Salt and pepper noise
$p = 0.05$

Uniform noise
$a = 10$
$b = 60$

Nonlinear filters

An overview of all nonlinear filters

void **harmonic**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);
void **maximum**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);
void **mean**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);
void **midpoint**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);
void **minimum**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);
void **range**(const image_t ***src**, image_t ***dst**, const uint8_t **n**);

Nonlinear filters – harmonic mean

- Is better at removing Gaussian type noise and preserves edges
- Removes positive outliers
- Harmonic mean filter is defined as

$$p_{dst}(x,y) = \begin{cases} 0 & \text{If any } p_{src}(x+i,y+j) = 0 \\ \dfrac{m}{\sum_{i=-n/2}^{i=n/2} \sum_{j=-n/2}^{j=n/2} \dfrac{1}{p_{src}(x+i,y+j)}} & otherwise \end{cases}$$

where
$p_{dst}(x,y)$: $the\ calculated\ result\ pixel\ in\ the\ destination\ image\ at\ (x,y)$
$p_{src}(x+i,y+j)$: $a\ pixel\ value\ in\ the\ src\ image\ within\ the\ window$
m: $the\ number\ of\ pixels\ included\ in\ the\ summation\ calculation$

Nonlinear filters – median

- Removes long tailed noise and salt and pepper type noise
- Has minimum bluring effect and preserves spatial details
- Can remove outlier noise from images that contain less than 50% of its pixels as outliers
- The median is defined as

$$p_{dst}(x,y) = \begin{cases} X[\frac{n}{2}] & if\ n\ is\ odd \\ \dfrac{X\left[\frac{n-1}{2}\right] + X[\frac{n}{2}]}{2} & if\ n\ is\ even \end{cases}$$

where

$p_{dst}(x,y)$: $the\ calculated\ result\ pixel\ in\ the\ destination\ image\ at\ (x,y)$

n: $the\ window\ size$

X: $sorted\ list\ of\ values\ from\ the\ source\ image\ in\ the\ window$

$$[window(x+i, y+j)]_{i=-n/2\ j=-n/2}^{i=n/2\ j=n/2}$$

Nonlinear filters – minimum

- Outputs the local minimum
- Is used to remove positive outlier noise
- Minimum filter is defined as

$$p_{dst}(x, y) = min[window(x + i, y + j)]_{i=-n/2\ j=-n/2}^{i=n/2\ j=n/2}$$

where
$p_{dst}(x, y)$: $the\ calculated\ result\ \ pixel\ in\ the\ destination\ image\ at\ (x, y)$
$window(x + i, y + j)$: $all\ values\ in\ the\ src\ image\ within\ the\ window$
n: $the\ window\ size$

Nonlinear filters – maximum

- Outputs the local maximum
- Is used to remove negative outlier noise
- Maximum filter is defined as

$$p_{dst}(x, y) = max[window(x + i, y + j)]_{i=-{}^{n}/_{2}\ j=-{}^{n}/_{2}}^{i={}^{n}/_{2}\ j={}^{n}/_{2}}$$

where
$p_{dst}(x, y)$: $the\ calculated\ result\ \ pixel\ in\ the\ destination\ image\ at\ (x, y)$
$window(x + i, y + j)$: $all\ values\ in\ the\ src\ image\ within\ the\ window$
n: $the\ window\ size$

HAN_UNIVERSITY OF APPLIED SCIENCES

Nonlinear filters – midpoint

- Outputs the average of the local minimum and maximum
- Used to remove short tailed noise, such as Gaussian and uniform type noise
- Midpoint filter is defined as

$$p_{dst}(x,y) = \frac{min[window(x+i,y+j)]_{i=-n/2\ j=-n/2}^{i=n/2\ j=n/2} + max[window(x+i,y+j)]_{i=-n/2\ j=-n/2}^{i=n/2\ j=n/2}}{2}$$

where
$p_{dst}(x,y)$: *the calculated result pixel in the destination image at* (x,y)
$window(x+i,y+j)$: *all values in the src image within the window*
n: *the window size*

HAN_UNIVERSITY OF APPLIED SCIENCES

Nonlinear filters – range

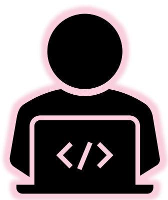
- Outputs the difference between the local maximum and minimum
- Range filter is defined as

$$p_{dst}(x,y) = \max[window(x+i,y+j)]_{i=-n/2\ j=-n/2}^{i=n/2\ j=n/2} - min[window(x+i,y+j)]_{i=-n/2\ j=-n/2}^{i=n/2\ j=n/2}$$

where
$p_{dst}(x,y)$: $the\ calculated\ result\ \ pixel\ in\ the\ destination\ image\ at\ (x,y)$
$window(x+i,y+j)$: $all\ values\ in\ the\ src\ image\ within\ the\ window$
n: $the\ window\ size$

HAN_UNIVERSITY OF APPLIED SCIENCES

EVD1 – Assignment



Study guide

Week 3

3 Nonlinear filters – meanFast()

4 Nonlinear filters – EXTRA

Spatial Filters

- Are basically discrete convolution filters
- The filter is known as a spatial mask
- Copy an image pixel-by-pixel while allowing for the effects of the pixel values in the local area (mask)
- Filter operations include multiplication, addition, and shifting
- Convolution is a time-consuming operation, so a 3x3 mask is typically used
- Typically uses odd masks and the centre of the mask for the pixel that is to be replaced, but this is by no means a requirement
- It can be implemented in parallel hardware for extremely fast execution

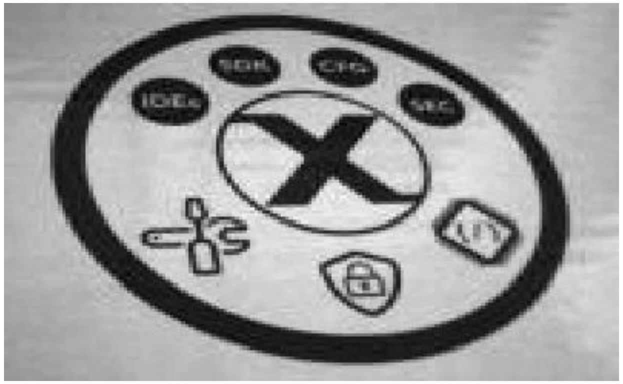
- Gaussian filters
- Laplacian filters
- Sobel filter

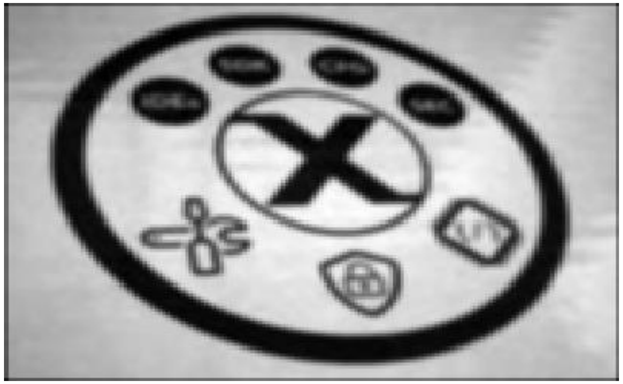
Gaussian

- Gaussian filters are masks formed from a two-dimensional Gaussian distribution
- Removes high frequency noise, but causes blurring

HAN_UNIVERSITY OF APPLIED SCIENCES

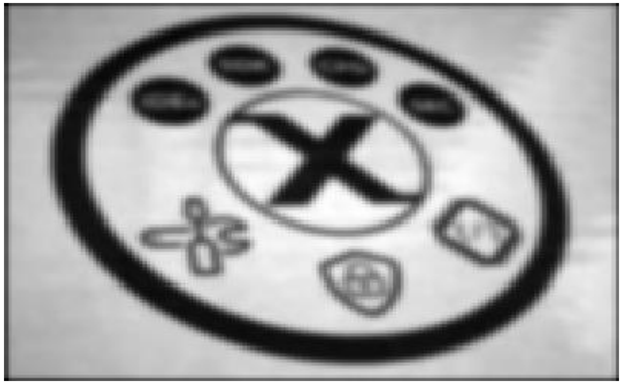
Gaussian - example





3x3

1	2	1
2	4	2
1	2	1



5x5

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

HAN_UNIVERSITY OF APPLIED SCIENCES

Gaussian - algorithm

void **gaussianFilter_3x3**(const image_t ***src**, image_t ***dst**);
void **gaussianFilter_5x5**(const image_t ***src**, image_t ***dst**);

See file **EVDK_Operators\spatial_filters.c**

HAN_UNIVERSITY OF APPLIED SCIENCES

Gaussian - algorithm

void **gaussianFilter_3x3**(const image_t ***src**, image_t ***dst**);
void **gaussianFilter_5x5**(const image_t ***src**, image_t ***dst**);

See file **EVDK_Operators\spatial_filters.c**

```
// PC app

// Create additional int16_pixel_t images, because calculations with such
// masks will not be in the uint8_pixel_t range
image_t *src_int16 = newInt16Image(IMG_WIDTH, IMG_HEIGHT);
image_t *dst_int16 = newInt16Image(IMG_WIDTH, IMG_HEIGHT);

cv::Mat cv_src_int16(IMG_HEIGHT, IMG_WIDTH, CV_16UC1, src_int16->data);
cv::Mat cv_dst_int16(IMG_HEIGHT, IMG_WIDTH, CV_16UC1, dst_int16->data);
```

Gaussian - algorithm

void **gaussianFilter_3x3**(const image_t ***src**, image_t ***dst**);
void **gaussianFilter_5x5**(const image_t ***src**, image_t ***dst**);

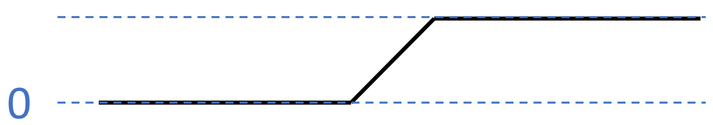
See file **EVDK_Operators\spatial_filters.c**

```
// ----------------------------------------------------------------------------
// Image processing pipeline
// ----------------------------------------------------------------------------
// Convert input image
convertBgr888ToUint8(img, src);
convertBgr888ToInt16(img, src_int16);

// Filter
gaussianFilter_3x3(src_int16, dst_int16);

// Scale both images to uint8_pixel_t for convenient visualisation
scale(src, src);
scaleInt16ToUint8(dst_int16, dst);
```

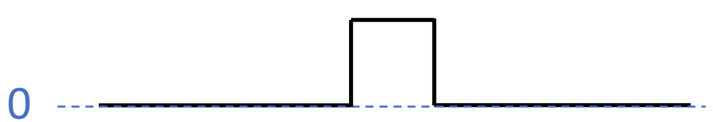
Laplacian

- Gives the second derivative in two directions
- Enhances changes and only changes

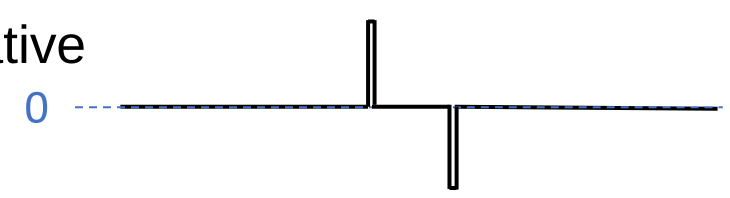
Edge



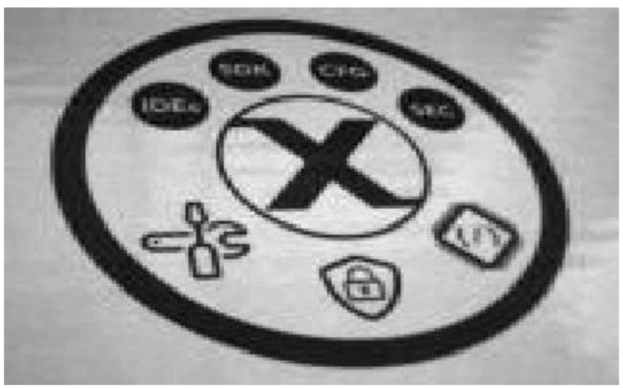
First derivative



Second derivative



Laplacian - example



3x3

0	−1	0
−1	4	−1
0	−1	0

5x5

-1	-1	-1	-1	-1
-1	-1	-1	-1	-1
-1	-1	24	-1	-1
-1	-1	-1	-1	-1
-1	-1	-1	-1	-1

Laplacian - algorithm

```
void laplacian_3x3(    const image_t *src, image_t *dst);
void laplacian_5x5(    const image_t *src, image_t *dst);
```

See file **EVDK_Operators\spatial_filters.c**

Sobel

- Edge detection algorithm with two results:
  1. Edge magnitude
  2. Edge direction
- Sobel magnitude and direction are defined as

$$M_{sobel}(x,y) = |G_H(x,y)| + |G_V(x,y)| \qquad \varphi_{sobel}(x,y) = tan^{-1}(\frac{G_V(x,y)}{G_H(x,y)})$$

where

$M_{sobel}(x,y)$: *the calculated magnitude in the destination image at* (x,y)
$\varphi_{sobel}(x,y)$: *the calculated direction in the destination image at* (x,y)
$G_H(x,y)$: *the pixel at* (x,y) *in the horizontal enhanced image*
$G_V(x,y)$: *the pixel at* (x,y) *in the vertical enhanced image*

Sobel

- The horizontal enhanced image G_H is obtained by a convolution with mask

3x3

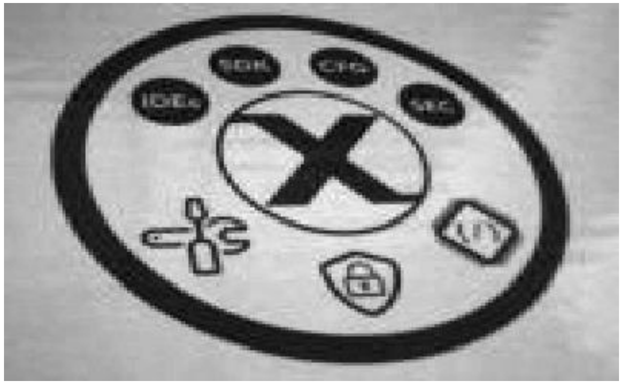
-1	-2	-1
0	0	0
1	2	1

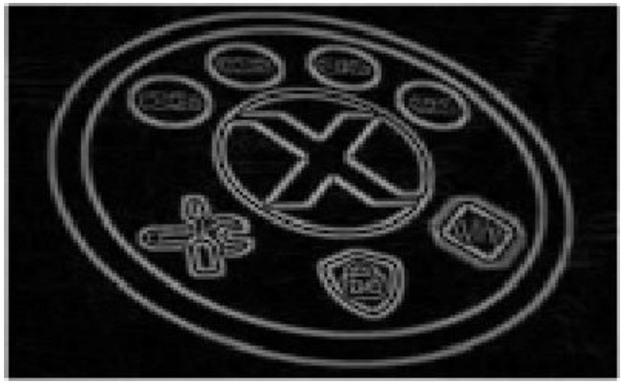
- The vertical enhanced image G_V is obtained by a convolution with mask

3x3

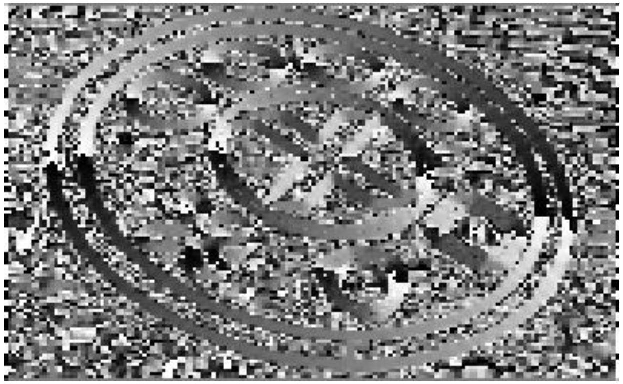
-1	0	1
-2	0	2
-1	0	1

Sobel - example





Magnitude



Direction between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$

Sobel - algorithm

void **sobel**(const image_t ***src**, image_t ***mag,** image_t ***dir**);

See file **EVDK_Operators\spatial_filters.c**

The dir image can be omitted (NULL). If not omitted, it must be of image type IMGTYPE_FLOAT.

Sobel - algorithm

void **sobel**(	const image_t ***src**, image_t ***mag,** image_t ***dir**);

See file **EVDK_Operators\spatial_filters.c**

The dir image can be omitted (NULL). If not omitted, it must be of image type IMGTYPE_FLOAT.

```
// EVDK example

// Create additional int16_pixel_t images, because calculations with such
// masks will not be in the uint8_pixel_t range
image_t *src_int16 = newInt16Image(IMG_WIDTH, IMG_HEIGHT);
image_t *dst_int16 = newInt16Image(IMG_WIDTH, IMG_HEIGHT);
```

Sobel - algorithm

void **sobel**(const image_t ***src**, image_t ***mag,** image_t ***dir**);

See file **EVDK_Operators\spatial_filters.c**

The dir image can be omitted (NULL). If not omitted, it must be of image type IMGTYPE_FLOAT.

```
// ----------------------------------------------------------------------
// Image processing pipeline
// ----------------------------------------------------------------------
// Convert uyvy_pixel_t camera image to int16_pixel_t image
convertUyvyToInt16(cam, src_int16);
```


HAN_UNIVERSITY OF APPLIED SCIENCES

Sobel - algorithm

void **sobel**(const image_t ***src**, image_t ***mag,** image_t ***dir**);

See file **EVDK_Operators\spatial_filters.c**

The dir image can be omitted (NULL). If not omitted, it must be of image type IMGTYPE_FLOAT.

```
// Copy timestamp
ms1 = ms;

// Detect edges using Sobel edge detect
sobel(src_int16, dst_int16, NULL);

// Copy timestamp
ms2 = ms;
```

Sobel - algorithm

void **sobel**(const image_t ***src**, image_t ***mag,** image_t ***dir**);

See file **EVDK_Operators\spatial_filters.c**

The dir image can be omitted (NULL). If not omitted, it must be of image type IMGTYPE_FLOAT.

```
// Scale uint8_pixel_t for convenient visualisation
scaleInt16ToUint8(dst_int16, dst);

// Convert uint8_pixel_t image to bgr888_pixel_t image for USB
convertToBgr888(dst, usb);
```

Sobel - algorithm

void **sobel**(const image_t ***src**, image_t ***mag,** image_t ***dir**);

See file **EVDK_Operators\spatial_filters.c**

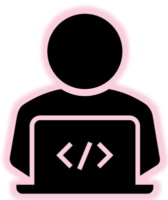
The dir image can be omitted (NULL). If not omitted, it must be of image type IMGTYPE_FLOAT.

Optimize most (-O3)

UVC connected

72 ms

EVD1 – Assignment



Study guide

Week 3

5 Spatial filters – sobelFast()

HAN_UNIVERSITY OF APPLIED SCIENCES

References

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