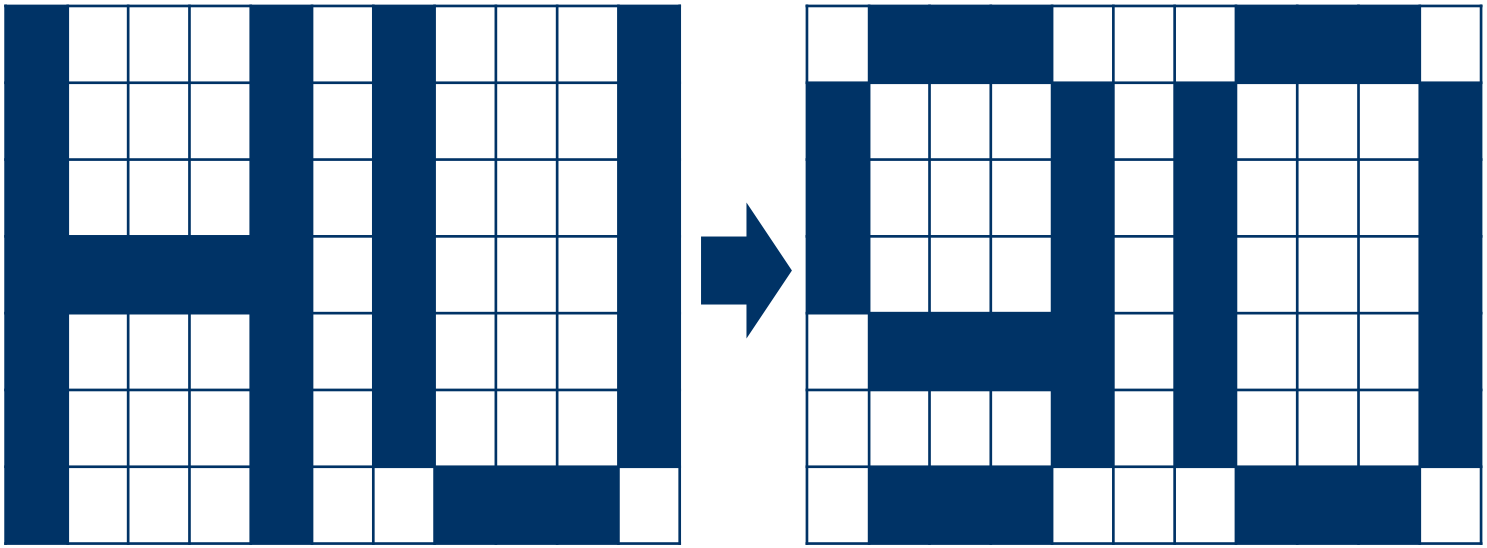
EVD1 – Vision operators

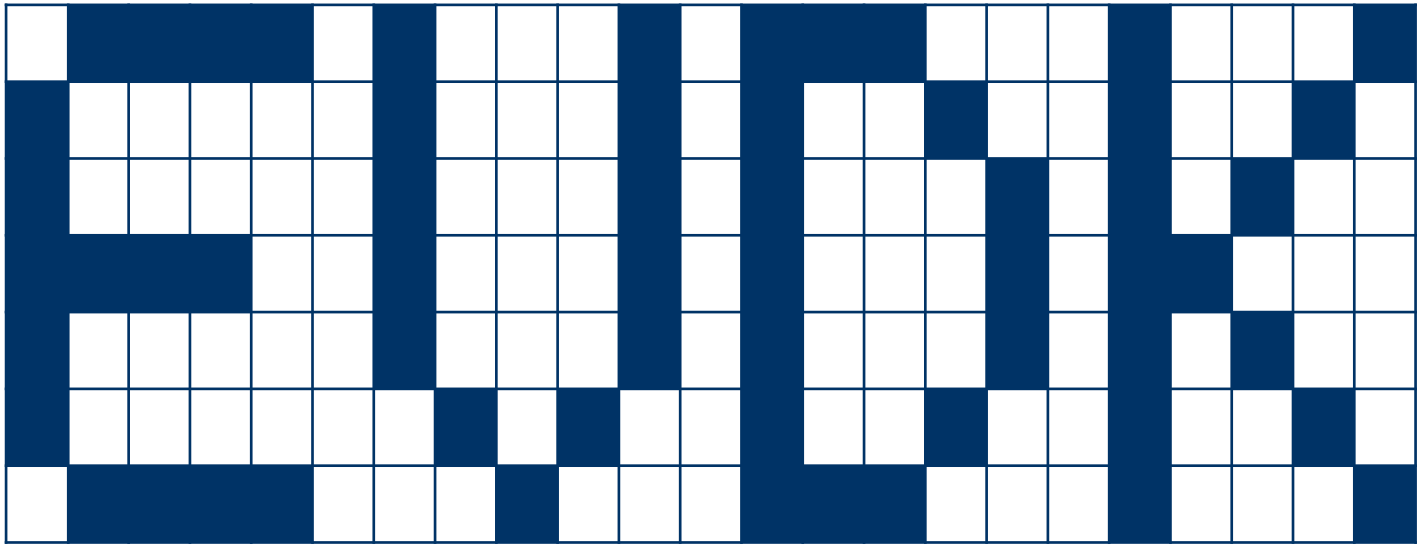
Door:



Source *Destination*

Hogeschool van Arnhem en Nijmegen

EVD1 – Vision operators



Doel: **bieden van een ontwikkelomgeving voor vision operatoren waarmee we**
- makkelijk kunnen debuggen,
- thuis kunnen werken,
- kunnen testen op een embedded target,

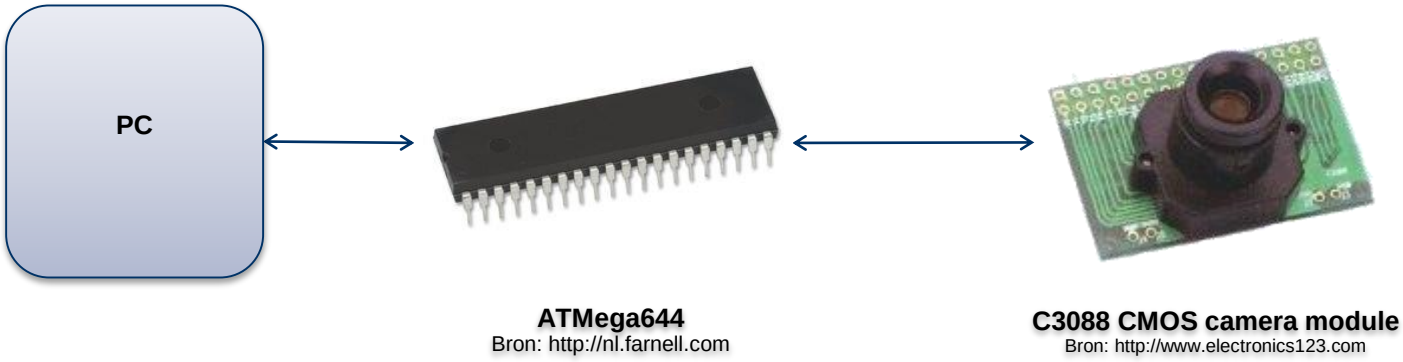
- uitdagende opdrachten voor EVD project realiseren!

HAN

EVD1 – Vision operators

Historie

EVD1 2009-2010



ATMega644
Bron: http://nl.farnell.com

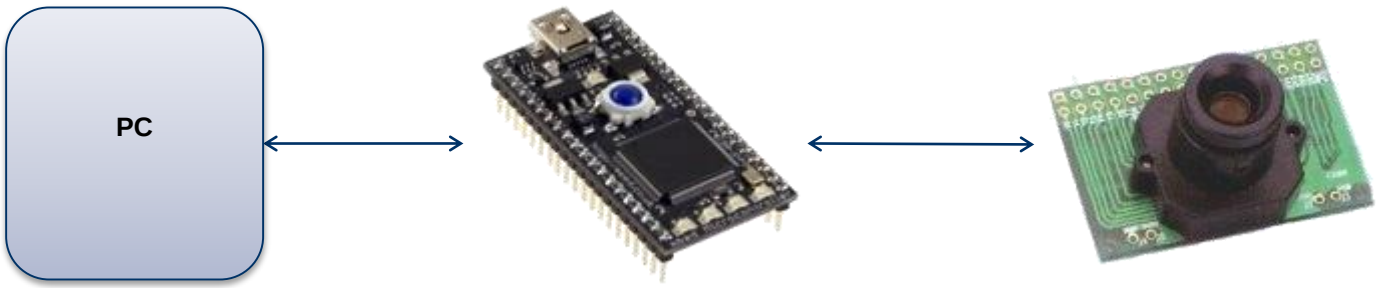
C3088 CMOS camera module
Bron: http://www.electronics123.com

Features:
- ~ 1 frame per minuut data transfer
- vision operators ‘on-the-fly’
- 4 kB RAM (picture 50x50 pixels)

EVD1 – Vision operators

Historie

EVD1 2010-2011



MBED
Bron: http://mbed.org/nxp/lpc1768

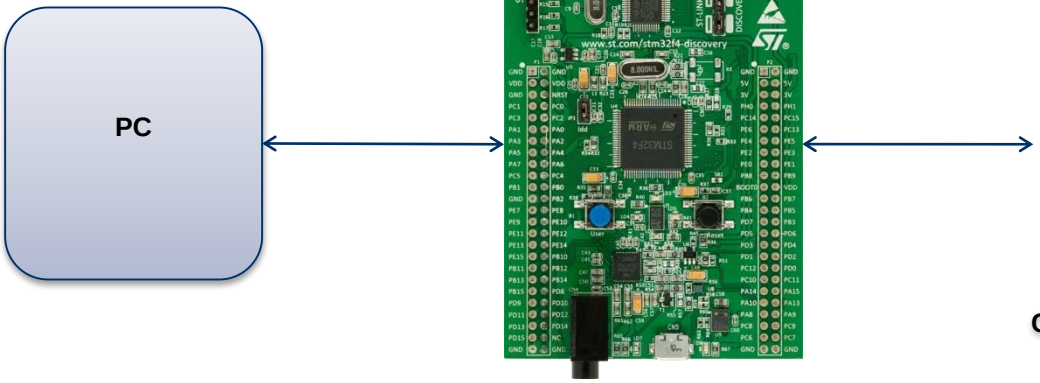
C3088 CMOS camera module
Bron: http://www.electronics123.com

Features:
- ~ 1 fps data transfer
- ~ 3 fps (standalone, incl. vision operators)
- 32 kB RAM (picture 176x144 pixels)

EVD1 – Vision operators

Historie

EVD1 2011-2012



STM32F4Discovery
Bron: http://www.st.com

C3088 CMOS camera module
Bron: http://www.electronics123.com

Features:
- 30 fps data transfer
- 7 fps (standalone, incl. vision operators)
- 192 kB RAM (multiple pics. 176x144 pixels)

EVD1 – Vision operators

Historie

EVD1 2013-2014

Silica Xynergy-M4
Bron: http://www.silica.com

CMOS 1.3 Megapixel camera-module
Bron: http://www.element14.com

Features:
- ? fps data transfer
- ? fps (standalone, incl. vision operators)
- 64 M x 16 bit

EVD1 – Vision operators

Toekomst

EVD1 2015-2016

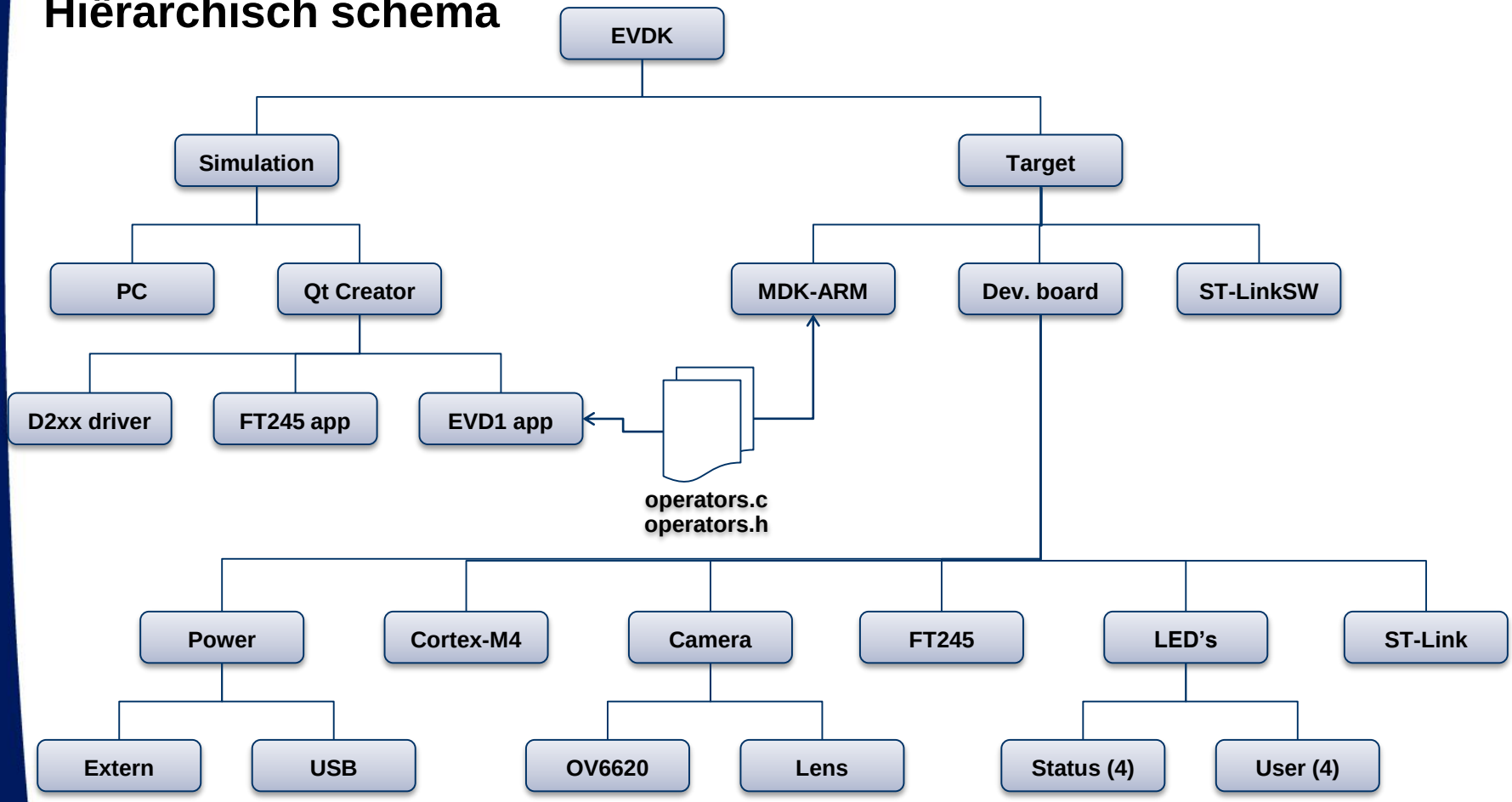
CMOS 5 Megapixel camera-module
Bron: http://www.terasic.com.tw

DE1-SoC Board
Bron: http://www.altera.com

Features:
- ? fps data transfer
- ? fps (standalone, incl. vision operators)
- Altera's Cyclone® V FPGA
- Dual-core ARM Cortex-A9 (HPS)
- 1GB DDR3 SDRAM (FPGA), 64 MB SDRAM (HPS)

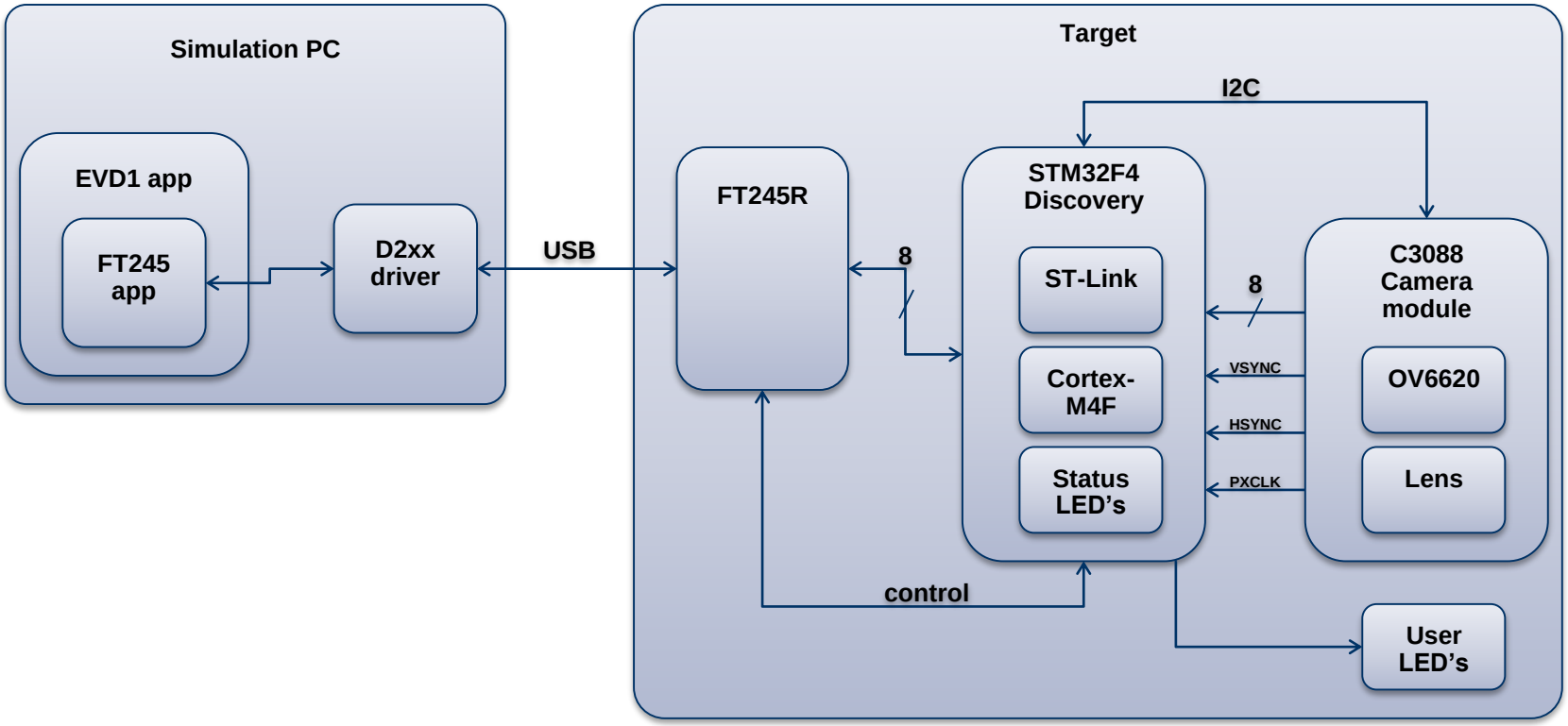
EVD1 – Vision operators

Hiërarchisch schema



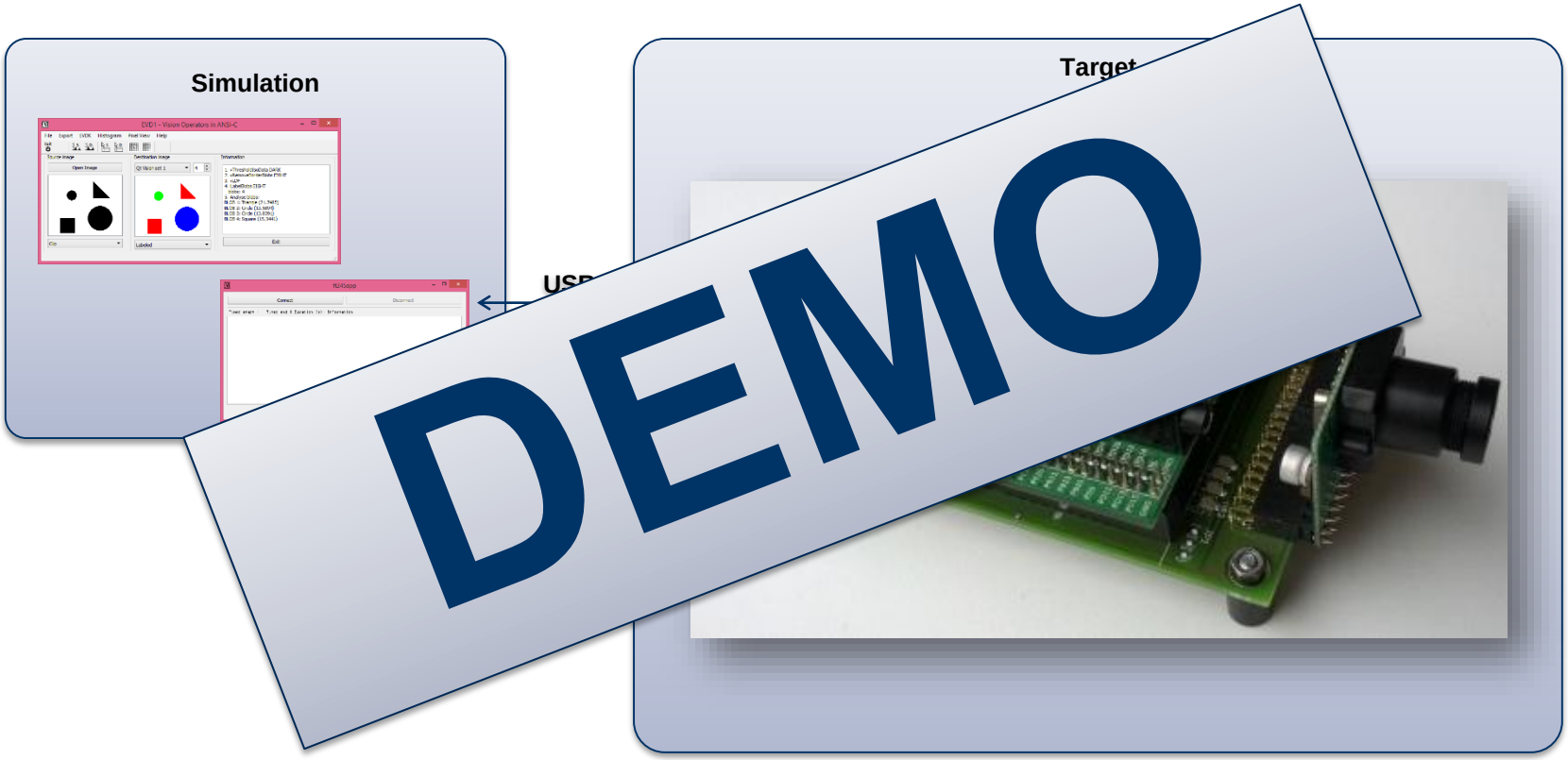
EVD1 – Vision operators

Architectuurschema



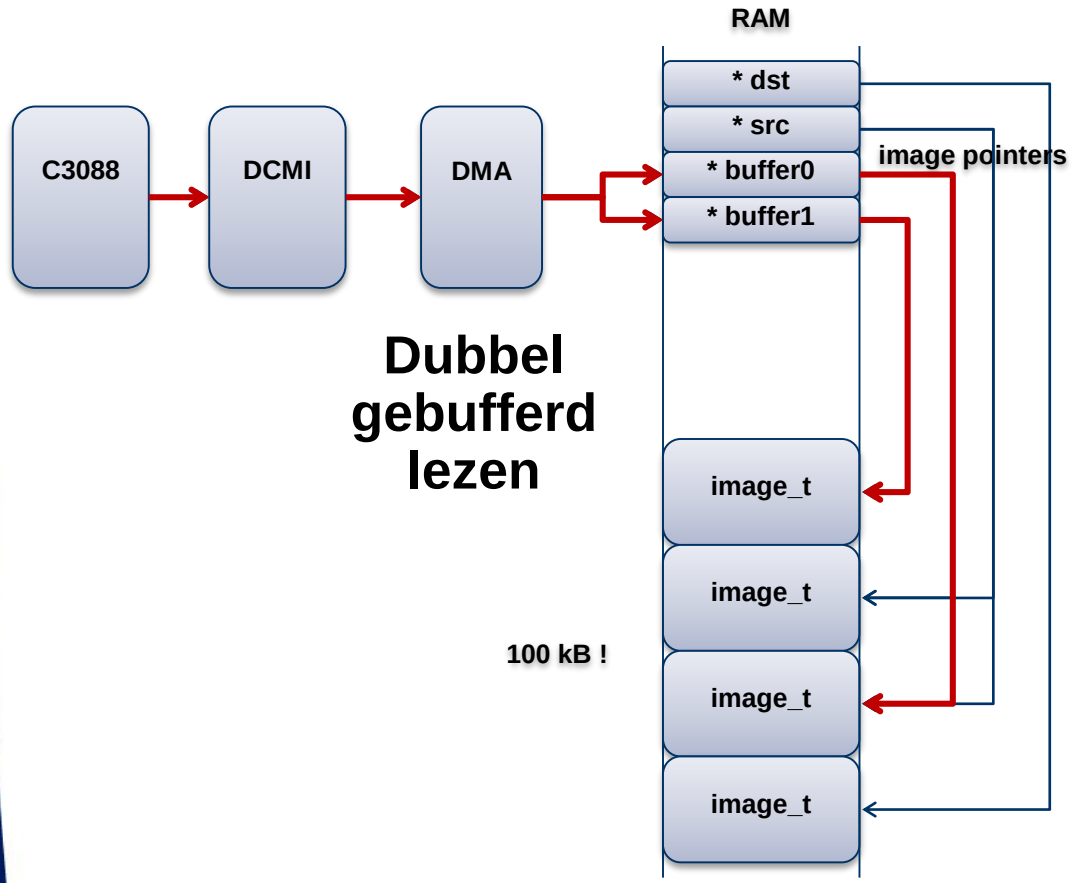
EVD1 – Vision operators

Realisatie



EVD1 – Vision operators

Software architectuur target

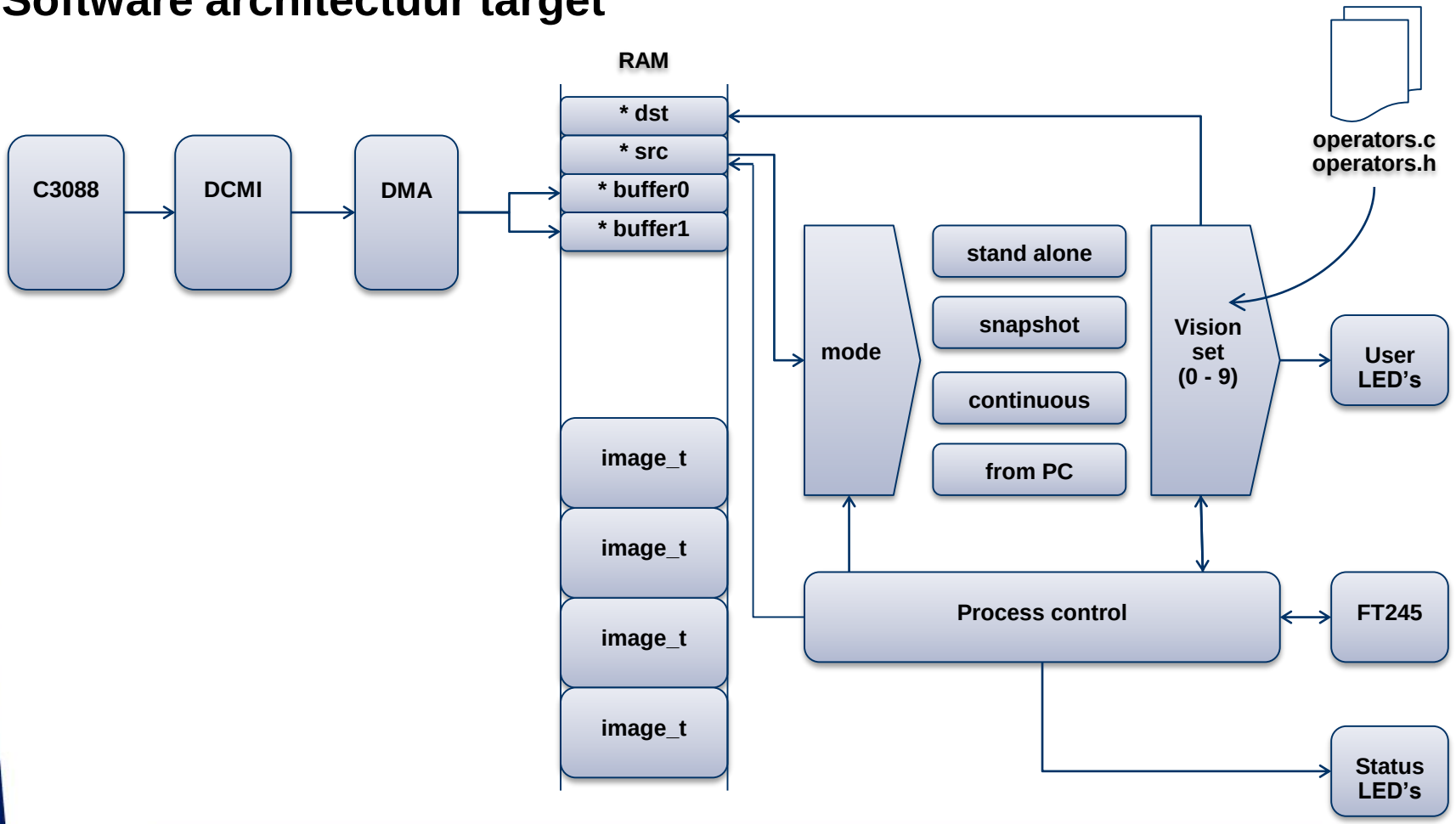


Snapshot nemen is verplaatsen van pointers…

```
if(DMAchannel0 != active)
{
  src = buffer0;
  buffer0 = src;
}
else
{
  src = buffer1;
  buffer1 = src;
}
```

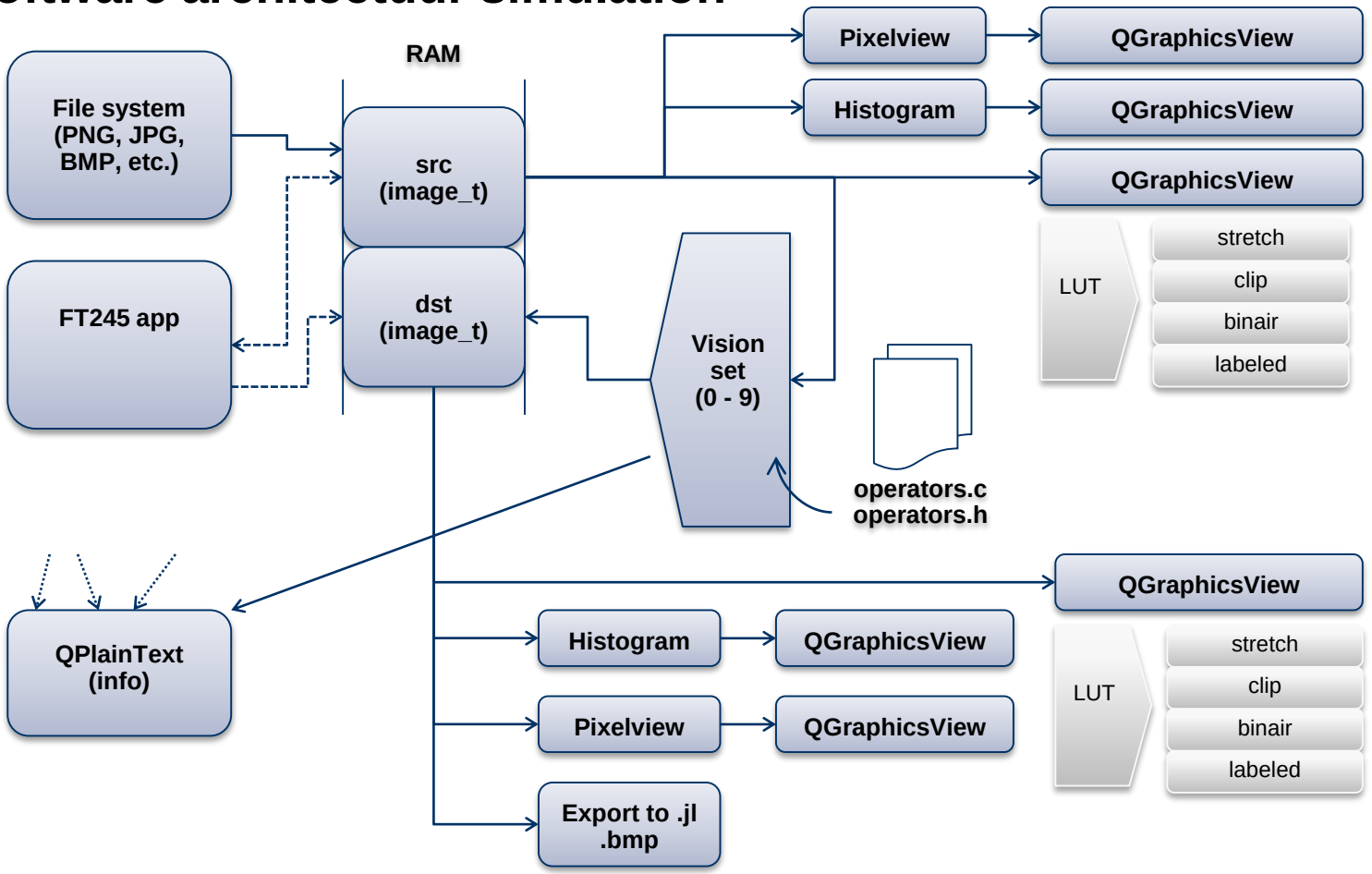
EVD1 – Vision operators

Software architectuur target



EVD1 – Vision operators

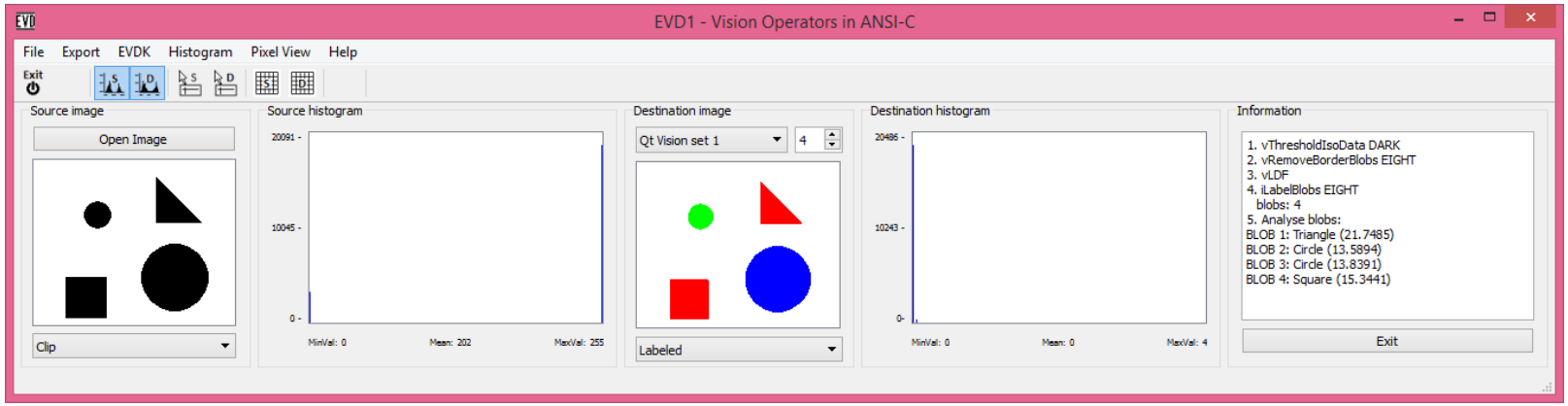
Software architectuur simulation



EVD1 – Vision operators

Doel van de opdracht: herkennen van 3 markers

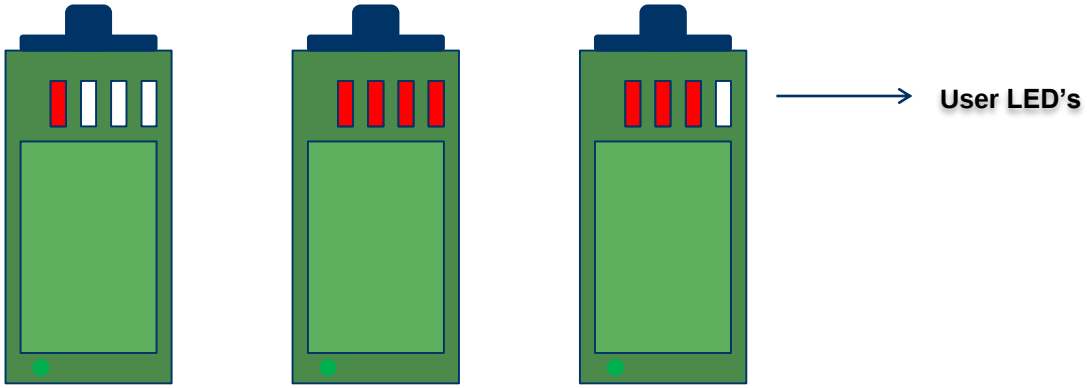
Bijvoorbeeld in simulatie:



EVD1 – Vision operators

Doel van de opdracht: herkennen van 3 markers

Bijvoorbeeld in target standalone mode:



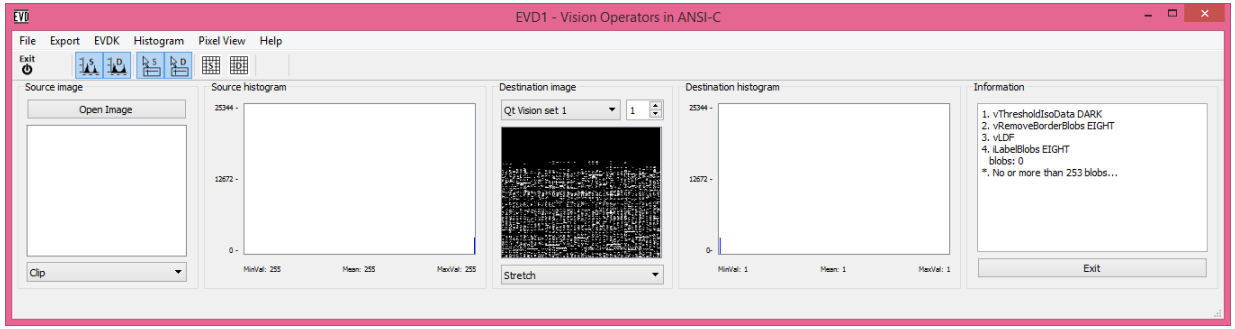
EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren *(zie ook file operators.h)*

```
void vContrastStretch(image_t *src, // must be a greyscale image
                      image_t *dst, // result is a greyscale image
                      uint16_t low,
                      uint16_t high);
```

Deze functie wordt ook gebruikt bij het 'streched' weergeven van images in de PC applicatie!

Zonder deze functie zie je:



EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren
(zie ook file operators.h)

```
void vThreshold(image_t *src,
                image_t *dst, // result is a binary image
                uint8_t low,
                uint8_t high);
```

Wordt veel gebruikt in andere functies!

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren *(zie ook file operators.h)*

```
void vRotate180(image_t *img);
```

De camera module op de target zit op de kop gemonteerd!

Wie implementeert het snelste algoritme? ☺

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren
(zie ook file operators.h)

```
void vContrastStretchFast(image_t *src,  // must be a greyscale image
                          image_t *dst); // result is a greyscale image
```

Veel gebruikte operator, we implementeren een snelle versie (stretched altijd van 0 t/m 255)

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren
(zie ook file operators.h)

```
void vErase(image_t *img);
```

Veel gebruikte operator, data wordt op 0 gezet

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren *(zie ook file operators.h)*

```
void vCopy(image_t *src,
           image_t *dst);

void vAdd(image_t *src,
          image_t *dst);

void vMultiply(image_t *src,
               image_t *dst);

void vSetSelectedToValue(image_t *src,
                         image_t *dst,
                         uint8_t selected,
                         uint8_t value);

void vSetBorders(image_t *src,
                 image_t *dst,
                 uint8_t value);
```

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren
(zie ook file operators.h)

```
void vBitSlicing(image_t *src,
                 image_t *dst,
                 uint8_t mask);

void vHistogram(image_t  *src,
                uint16_t *hist,
                uint32_t *sum);

void vThresholdIsoData(image_t *src,
                       image_t *dst,
                       eBrightness brightness); // DARK | BRIGHT

void vFillHoles(image_t *src, // must be a binary image
                image_t *dst,
                eConnected connected); // FOUR | EIGHT

void vRemoveBorderBlobs(image_t *src, // must be a binary image
                        image_t *dst,
                        eConnected connected); // FOUR | EIGHT
```

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren
(zie ook file operators.h)

```
void vBinaryEdgeDetect(image_t *src, // must be a binary image
                       image_t *dst);

uint16_t iLabelBlobs(image_t *src, // must be a binary image
                     image_t *dst,
                     eConnected connected);

void vBlobAnalyse(image_t *img,
                  const uint8_t blobnr,
                  blobinfo_t *pBlobInfo);

void vNonlinearFilter(image_t *src,
                      image_t *dst,
                      eFilterOperation fo,
                      uint8_t n);

double dNormalizedCentralMoments(image_t *img,
                                 uint8_t blobnr,
                                 int p,
                                 int q);
```

EVD1 – Vision operators

Om dit te bereiken implementeren we de operatoren
(zie ook file operators.h)

`En per student een unieke operator naar keuze in overleg met de docent!`

EVD1 – Vision operators

Belangrijk:

De operatoren voldoen aan de ANSI-C standaard.

De operatoren dienen zo geschreven te worden dat het destination image hetzelfde moet kunnen zijn als het source image (ivm beperkt RAM op target).

In de functies worden (dus) geen volledige local images gedeclareerd, hooguit een klein deel (bijvoorbeeld enkele rijen).

De functies worden zo geschreven dat de focus op performance ligt. Hoe sneller de operatoren uitgevoerd worden, des te beter.

EVD1 – Vision operators

En nu aan de slag…

Installeren
- Qt Creator
- Keil MDKARM uVision (lite)
- ST-Link utility software (eventueel)
- FTDI Chip D2xx driver

SVN Update van http://svn.han-ese.nl/opleiding/evdk
- Documentatie (planning, beoordeling, etc.)
- Lessen
- Test images
- Template project simulatie: Qt Creator
- Template project target: Keil MDKARM uVision

Logboek

EVD1 – Vision operators

En nu aan de slag…

Uitvoering
- Individueel (samenwerken mag, kopiëren niet!!)
- 2x3 lesuren per week
- verplichte aanwezigheid tijdens de lessen
- uitleg en ondersteuning mbt de operatoren in de les
- werken met een target ALLÉÉN tijdens de les

Beoordeling
- Adhv de ingeleverde source files
- Individueel assessment in de tentamenweek
- Bevraging over de implementatie en gebruik operatoren
- Zelfgeschreven logboek mag geraadpleegd worden