

TECHNISCH RAPPORT

Sudoku vision system



Auteurs: Roy Franck, Ivo Ebbers, Thom Hubers

Datum: 13-11-2019

Embedded Vision Design

Inleiding

Van de hogeschool van Arnhem en Nijmegen heeft een groep studenten de opdracht gekregen om een vision systeem te realiseren dat de stappen doorloopt van het machine vision proces. Deze stappen zijn: **acquisitie , beeldverbetering , segmentatie kenmerk bepaling en classificatie.** Dit algoritme moet geïmplementeerd worden op een embedded systeem , welke verbonden is met een camera en een display .

Dit document gaat over de technische realisatie van het sudoku vision systeem. Dit systeem maakt een beeld van een sudoku spel , herkent cijfers in het sudoku spel en zet de resultaten van de herkenning op een display.

Het document is als volgt ingedeeld. Eerst word het functioneel document besproken. Daarna het technische document. En de laatste twee hoofdstukken gaan over de realisatie en het testen van het sudoku vision systeem

Inhoud

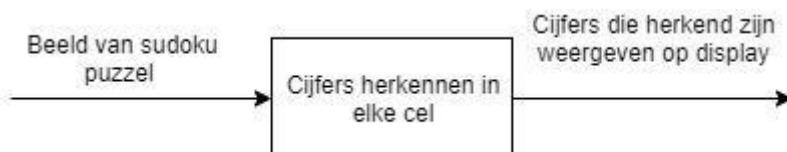
Inleiding	2
1.Functioneel ontwerp	4
1.1 Globale IPO	4
1.2 Functionele specificatie	5
1.3 Platform	5
1.4 User interface	5
1.5 Gedetailleerd IPO schema	5
3.Technisch ontwerp	6
3.2 vison proces	6
3.3 acquisitie	6
3.4 Beeldverbetering	7
3.4.1 Gemiddelde filter	7
3.4.2 Warpen	7
3.5 segmentatie	8
3.5.1 grijstinten	8
3.5.2 threshold	8
3.5 kenmerkbepaling	9
3.5.1 Labelen	9
3.7 classificatie	9
3.8 software architectuur	9
3.8 hardware en hardware architectuur	9
3.8.1 hardware architectuur	9
3.8.2 hardware	10
4.Realisatie	12
4.1 Acquisitie Opstelling	13
4.2 Beeldverwerking	13
5.Testresultaten	14
5.1 belichting	14
5.2 herkennen cijfers	15
6.Conclusies en aanbevelingen	16
Referenties	16
Bijlage	16
A Functionele specificaties	16
B Algoritme operators	17

1.Functioneel ontwerp

In dit hoofdstuk zal het functioneel ontwerp worden besproken. Het hoofddoel van het sudoku project is om een vision algoritme te ontwikkelen dat de cijfers kan detecteren en herkennen in de cellen van een sudoku puzzel. Met behulp van deze algoritme kan in een vervolg stap een sudoku puzzel worden opgelost.

1.1 Globale IPO

In figuur 1 is het globale input-proces-output schema van het van het sudoku vision systeem weergegeven. Het IPO schema geeft het systeem op het hoogste niveau weer.



Figuur 1 - Globale IPO schema sudoku vision systeem

De input is een beeld van een sudoku puzzel dat genomen wordt met een digitale camera. Het beeld wordt vervolgens verstuurd naar een microcontroller. Deze microcontroller verwerkt het beeld met behulp van een computer vision algoritme. Deze algoritme zal cijfers uit elke cel van het sukoku puzzel detecteren en herkennen. Daarna zal het systeem de cijfers in een matrix zetten en weergeven op een display.

In een vervolg project kan een deze matrix worden opgelost met een van de sudoku vele oplosmethodieken die in de literatuur/internet te vinden zijn. Het oplossen van de sudoku puzzel valt buiten dit vision project en zal dus niet worden geïmplementeerd in het systeem.

1.2 Functionele specificatie

De functionele specificaties zijn te vinden in de bijlage A.

1.3 Platform

Het Sudoku solver systeem zal op zichzelf moeten kunnen werken op een stm32F746NGH6 microcontroller. Deze microcontroller zal door een power adapter worden gevoed dat verbonden is aan het lichtnet.

1.4 User interface

Er is geen user interface aanwezig, waarbij de gebruiker het systeem bestuurt. Het beeld zal automatisch verwerkt worden en de resultaten (herkenning van de cijfers in de sudoku cellen) zullen worden weergegeven op een display. Voor onderhoud kan gebruikt worden van de debugger in de softwarepakketten Qt of Stm32CubeMX. Hiervoor moet de microcontroller worden verbonden op een pc via een usb kabel.

1.5 Gedetailleerd IPO schema

In figuur 2 is het globale input-proces-output schema van het van het sudoku vision systeem weergegeven. Aan de ingang zijde is de voeding te zien dat het systeem voedt. Daarnaast het beeld van de sudoku puzzel dat door de camera wordt gemaakt. Aan de uitgangzijde worden de cijfers die herkent zijn worden weergegeven op een display. De informatie naar gebruiker is wat de gebruiker te zien krijgt, namelijk de matrix met getallen die herkend zijn.



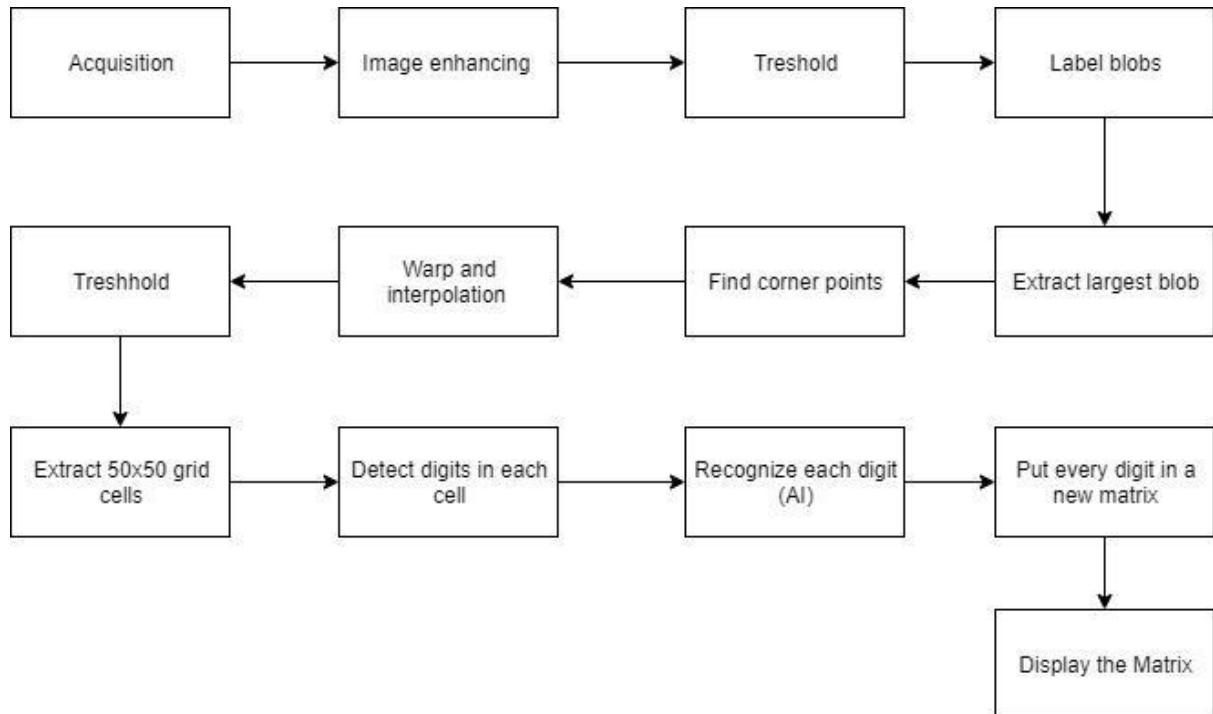
Figuur 2 – Gedetailleerde IPO schema

3.Techничк ontwerp

In dit hoofdstuk wordt het technisch ontwerp besproken van het sodoku vision systeem.

3.2 vison proces

In figuur 3 is het gehele vision proces te zien voor het sudoku project. Het begint met de acquisitie van het beeld. De acquisitie vertelt hoe het beeld van de echte wereld



Figuur 3 - Sudoku vision proces diagram

3.3 acquisitie

De acquisitie opstelling bestaat uit een lichtbron, camera lens en camera sensor(zie figuur 3). De lichtbron bestaat uit een LED ring array dat om de camera is gemonteerd. Door de puzzel te belichten kan er een hoog contrast worden gecreëerd tussen de achtergrond en de belangrijke kenmerken (raster , cijfers.). Daarnaast wordt er diffusie papier voor de LED verlichting geplaatst om schaduwvorming te voorkomen t.g.v. oneffenheden(kreukels) van het sudoku papier.

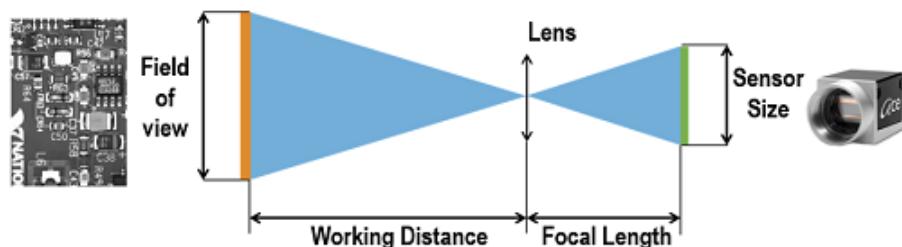


Figuur 4 - acquisitie diagram

Uit deskresearch is gebleken dat de OV9655 camera een geschikte kandidaat is voor het sudoku project. Deze camera is namelijk relatief goedkoop, makkelijk verkrijgbaar en voldoet aan de minimale eis voor de resolutie(640 x 320). De camera lens heeft een grootte van $\frac{1}{4}$ inch(3.6mm x 2.7mm) en een focus lengte van 4.1mm (B, 2012) en heeft een CMOS camera sensor. De werk afstand tussen de camera en de sudoku puzzel kan worden berekend met de twee onderstaande formules:

$$\text{Magnification} = (\text{Sensor Size of the camera}) / (\text{Field of View})$$

$$\text{Working Distance} = \text{Focal Length} \left(\frac{1 + \text{Magnification}}{\text{Working Distance}} \right)$$



Figuur 5 - Verband tussen werkafstand, focuslengte sensor grootte en FOV

Vullen we de gegevens in voor de sensor grootte (in een richting) en de focuslengte dan is de werkafstand ongeveer 155mm, zie hieronder de berekening .

$$\text{Magnification} = \frac{2.7}{100} = 0,027$$

$$\text{Working Distance} = 4.1 \left(\frac{1 + 0,027}{0,027} \right) = 155\text{mm}$$

3.4 Beeldverbetering

Het doel van beeldverbetering is om het beeld gereed te maken voor segmentatie. Er zijn twee beeldverbetering die het soduku systeem doorvoert, namelijk het filteren van ruis en warpen van het beeld.

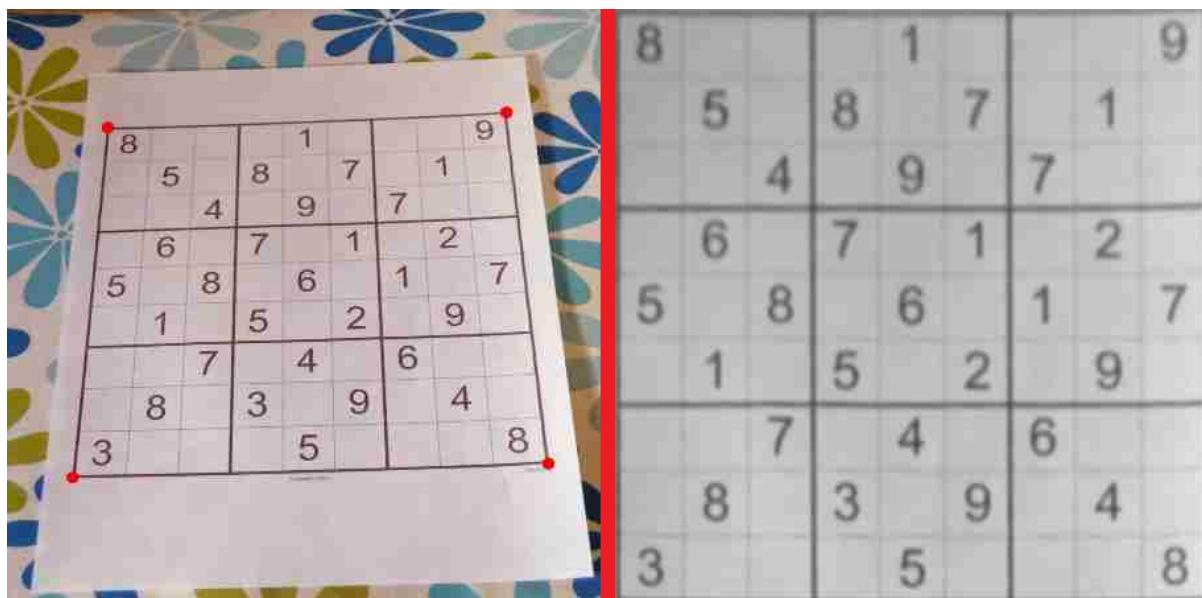
3.4.1 Gemiddelde filter

Voor beeld verbetering moet eerst worden nagegaan van welke type ruis de camera kan verwachten. De meest voorkomende ruis in een camera is gaussische ruis. Deze ruis wordt vaak veroorzaakt door slechte belichting , ruis van de elektronica of hoge tempratuur verschillen. De gaussische ruis veroorzaakt variaties van intensiteit tussen naburige pixels in een plaatje. Om de gaussische ruis te verminderen is een gemiddelde filter het meest geschikt om te gebruiken. Deze filter gaat elke pixel na en vervangt deze pixel met de gemiddelde waarde van de naburige pixels. Hierdoor worden de hoeveelheid variaties tussen naburige pixels verminderd.

3.4.2 Warpen

Het warpen is nodig omdat het soduku papier niet altijd even recht voor de camera staat. Als het papier niet recht staat voor de camera dan kan het lastig worden om sommige cijfers te herkennen. Warpen is een geometrische operatie en bestaat uit een mapping functie en een interpolatie methode. De mapping functie specificeert waar de pixels van het orginele beeld naartoe worden gewarpt. En de interpolatie methode specificeert hoe de intensiteit van de pixels wordt berekend in het nieuwe beeld (princeton, n.d.).

Een mapping functie is nodig om de pixels van het orginele plaatje naar een nieuw plaatje te warpen om zodoende het soduku spel recht voor het beeld te kunnen krijgen. De mapping functie voor het dit project is perspectieve transformatie. Er is geprobeerd om affine transformatie te gebruiken maar deze gaf niet het gewenste resultaat. De perspectieve transformatie neemt de pixel coördinaten(x,y) van de hoekpunten van het grootste object en transformeert deze naar een nieuwe coördinaat (x',y') in een nieuw plaatje, zie figuur 6.



Figuur 6 - Voorbeeld perspectieve transformatie sodoku puzzel (Jhawar, 2019)'

Er zijn meedere algoritmes beschikbaar voor interpolatie , de belangrijkste zijn:

- Adaptive

- Non adaptive(bilinear , bicubic , nearest neighbour etc.)

Welke interpolatie methode het meest geschikt is hangt af van de kwaliteit en beeldverwerkingstijd. Er is gekozen om een bicubic interpolatie toe te passen omdat deze de beste verwerkingstijd/kwaliteit verhouding geeft ten opzichte van de andere interpolatie methodes.

3.5 segmentatie

3.5.1 grijstinten

Het eerste stap van de segmentatie is het beeld omzetten in grijstinten. Kleur is namelijk niet nodig en zorgt voor onnodige complexiteit van het algoritme.

3.5.2 threshold

De tweede stap voor segmentatie is om een threshold methode toe te passen. Het doel van een threshold is om de pixels donker(0) of licht (1) te maken. Door een threshold uit te voeren wordt een beeld binair gemaakt. Hierdoor worden de letters en raster gesegmenteerd van de achtergrond. Er zijn verschillende methodes om de threshold uit te voeren. Voor dit project is adaptieve threshold geïmplementeerd. Lichtintensiteit kan namelijk variëren en dit is onwenselijk. Door adaptieve threshold toe te passen kunnen regionale contrast verschijnen worden verminderd. De adaptieve threshold (R.Fisher, 2003) werkt als volgt:

- 1) Pas een niet-lineaire gemiddelde filter toe op het plaatje
- 2) Trek source(src) plaatje van de destination image(dst, gefilterd plaatje) af
- 3) Threshold het destination plaatje
- 4) Inverteer het destination plaatje

Er wordt twee keer threshold toegepast in het sudoku algoritme. De eerste keer voor het vinden van het rooster en de tweede keer voor het uit elkaar halen van het rooster voor cijfer detectie.

3.5 kenmerkbepaling

3.5.1 Labelen

Voor de kenmerkbepaling moeten eerst de belangrijkste blob in de sodoku puzzel worden gelabeld. Dit wordt gedaan door een waarde(1,2,3 etc) toe te kennen aan de blob. Voor het sodoku beeld is de grootste blob de buitenste vierkant van het spel. De overige niet blobs worden verwijderd. Het labelen van de grootste blob is nodig om het sodoku spel te kunnen warpen.

3.7 classificatie

Het kNN(k-Nearest Neighbors) algoritme wordt gebruikt voor het herkennen van getallen. Dit algoritme is één van de simpelste machine learning-algoritmes die er zijn voor “supervised learning”. Supervised learning-algoritmen worden aan de hand van gelabelde voorbeelden getraind . Het idee achter het kNN-algoritme is het zoeken naar het best overeenkomende patroon in vergelijking met de test data.

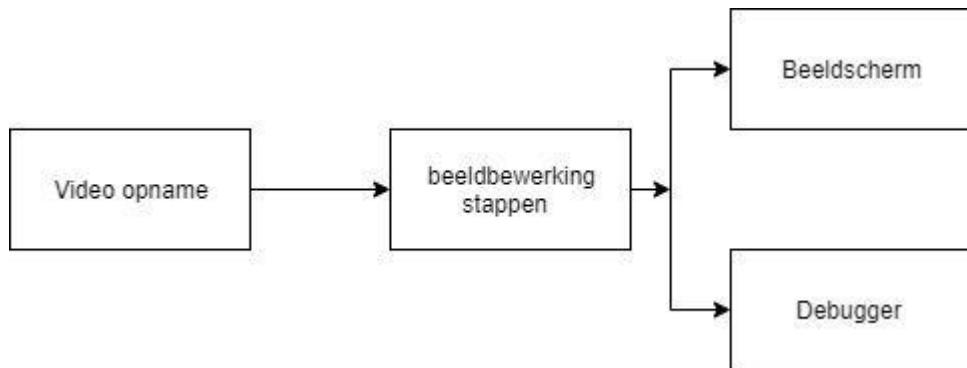
Het proces voor de methode omvat de volgende stappen:

- 1) Trainen met classificaties(handgeschreven cijfers reeks)
- 2) Herkennen van getallen aan de hand van de classificaties

De reden waarom deze classificatie methode is gekozen is omdat het een simpele maar toch erg krachtige algoritme is voor classificatie. Daarnaast heeft deze algoritme een relatief lage verwerkingstijd ten opzichte van andere classificatie algoritmes (SRIVASTAVA, 2018).

3.8 software architectuur

De software architectuur bestaat uit een video opname als input ,beeldbewerkingstapping in het vision proces , beeldscherm en debugger als output. De debugger is nodig om fouten in het systeem op te sporen.



Figuur 7 - Architectuur schema voor de software

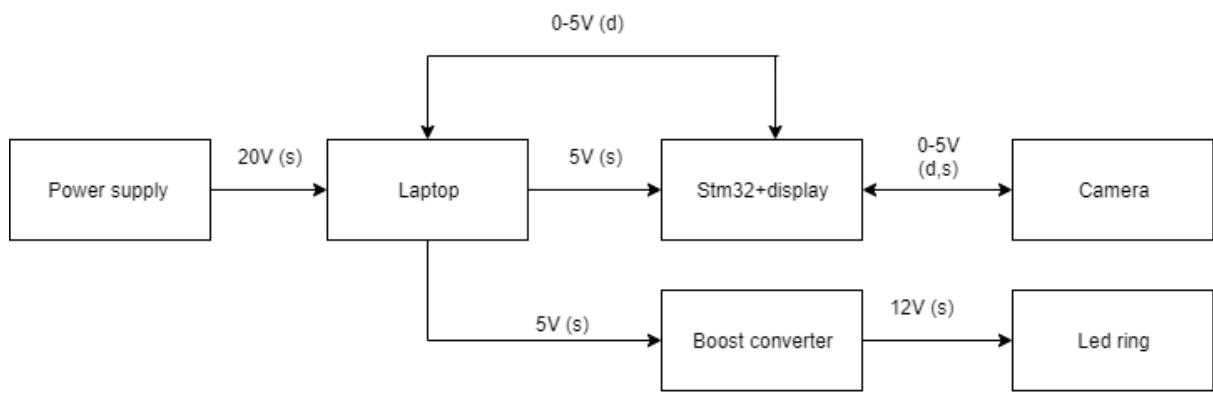
3.8 hardware en hardware architectuur

In deze paragraaf worden de hardware en hardware architectuur besproken. De hardware van het sudoku systeem bestaat uit een microcontroller, camera , een LED ring en een DC-DC converter. Hieronder worden de individuele hardware apparatuur besproken en hun keuzes toegelicht.

3.8.1 hardware architectuur

In figuur 8 is de hardware architectuur te zien van het sudoku vision systeem. Hierin is te zien hoe de hardware onderdelen zijn verbonden en welke spanningen/data elke onderdeel kan verwachten. De verbinding boven de laptop en stm32 is de debugger kabel.

Belangrijke componenten zijn de voeding, laptop ,stm32 kit met display , boos converter voor het voeden van de led ring, en camera.



d= data stream ; s=supply

Figuur 8 – architectuur sudoku vision systeem

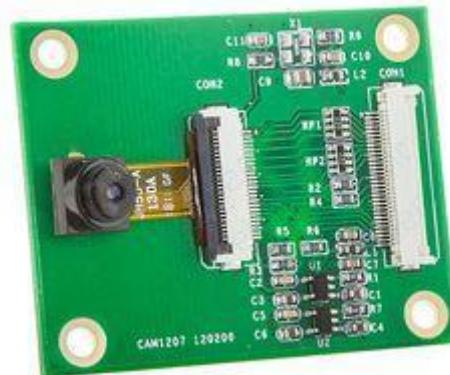
3.8.2 hardware

Camera module

Er is gekozen voor een stm32F4DIS-CAM camera module. In deze module zit een OV9655 camera met een CMOS SXGA beeld sensor. Daarnaast zijn de volgende specificaties van toepassing:

- Maximale resolutie : 1280x1024 (SXGA)
- Framerate: 15-30 fps
- CMOS sensor
- Lens grootte: 1/4 inch
- Energieverbruik: +-90mW

De keuze voor deze camera is omdat de camera de gewenst resolutie(hoger dan 640x320) heeft en kan worden aangesloten aan de stm32 bordje.

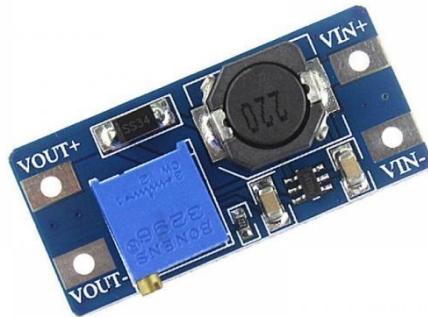




Figuur 10 – LED ring

Boost converter

De ledring moet gevoed worden uit de 5V van de usb aansluiten voor de laptop. Om de spanning te verhogen kan een boost converter of een spannings regelaar worden gebruikt. Echter een spannings regelaar verliest relatief veel energie bij grote spanningsvallen(5 naar 12v). Vandaar dat besloten is om een buckconverter te gaan gebruiken i.p.v. een spanningsregelaar. Deze heeft namelijk een efficiency van ongeveer 80% ongeacht de spanningsvallen .



Figuur 11 – Boost converter

Stm32

Voor dit project is gekozen om een stm32 microcontroller te gaan gebruiken waarop het vision algoritme wordt geïmplementeerd. De reden hiervoor is dat de microcontroller beschikt over een display dat erop is gemonteerd. Daarnaast is deze kit beschikbaar op school en hoeft niet op nieuw worden aangeschaft.



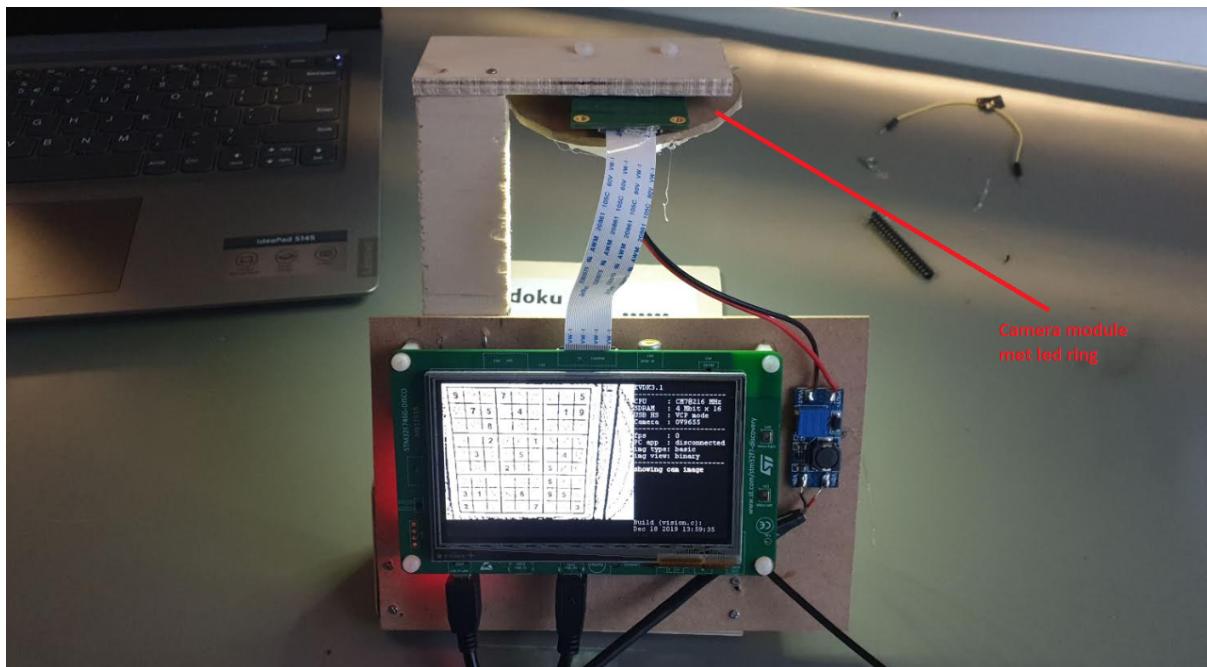
Figuur 12– STM32 kit met display

4. Realisatie

De realisatie omschrijft hoe de resultaten van de vison processen zijn bereikt. Eerst wordt verteld over de acquisitie opstelling. Vervolgens worden de implementatie van het beeldverwerkingsproces besproken.

4.1 Acquisitie Opstelling

Zoals eerder vermeld bestaat de acquisitie opstelling uit een de belichting en camera. Op deze camera is de led verlichting gemonteerd. De constructie dat in figuur 12 is de constructie te zien die ervoor zorgt dat de afstand tussen het sudoku papier en camera ongeveer 155 mm is. Zodoende kan het gehele beeld van de sudoku worden weergegeven op het scherm.



Figuur 13– Acquisitie opstelling voor het sudoku vision systeem

4.2 Beeldverwerking

Allereerst zijn de individuele operatoren gecreëerd om het hoofdalgoritme te kunnen programmeren. De volgende operatoren zijn geïmplementeerd in het hoofd algoritme:

- **Non linear average filter** voor ruistfiltering
- **Substract** voor adaptive threshold
- **Threshold** voor adaptive threhsold
- **Invert** voor adaptive threshold
- **Remove border blobs** voor het verwijderen van het blobs aan de randen van het plaatje
- **Label blobs** voor het labelen van het grootste blob
- **Blob info** voor het vinden van het grootste blob aan de hand van de maximale blob grootte en aantal pixels
- **Get basic pixel** voor het verkrijgen van de pixelwaarde in het plaatje
- **Set basic pixel** voor het definiëren van een pixelwaarde
- **Interpolatie** voor het interpoleren van het beeld na het warpen

In de bijlage B zijn de individuele algoritmes van de operatoren te vinden.

Vervolgens is het hoofdalgoritme geprogrammeerd. Dit algoritme doorloopt de volgende stappen:

- 1) Converteer het plaatje na grijsinten
- 2) Maak een destination image
- 3) Pas adaptive threshold toe voor segmentatie
- 4) Label de blobs
- 5) Vind het grootste blob (het sudoku spel)
- 6) Verwijder overige blobs
- 7) Bepaal de hoekpunten voor het warpen
- 8) Warp het beeld
- 9) Pas adaptive threshhold toe aan het nieuw gewarpte beeld
- 10) Haal de individuele cellen met cijfers uit het beeld voor herkenning
- 11) Pas het Knn algortime toe voor cijfer herkenning

In het hoofdstuk testen is het resultaat van het algoritme te zien en in bijlage C het hoofdalgoritme.

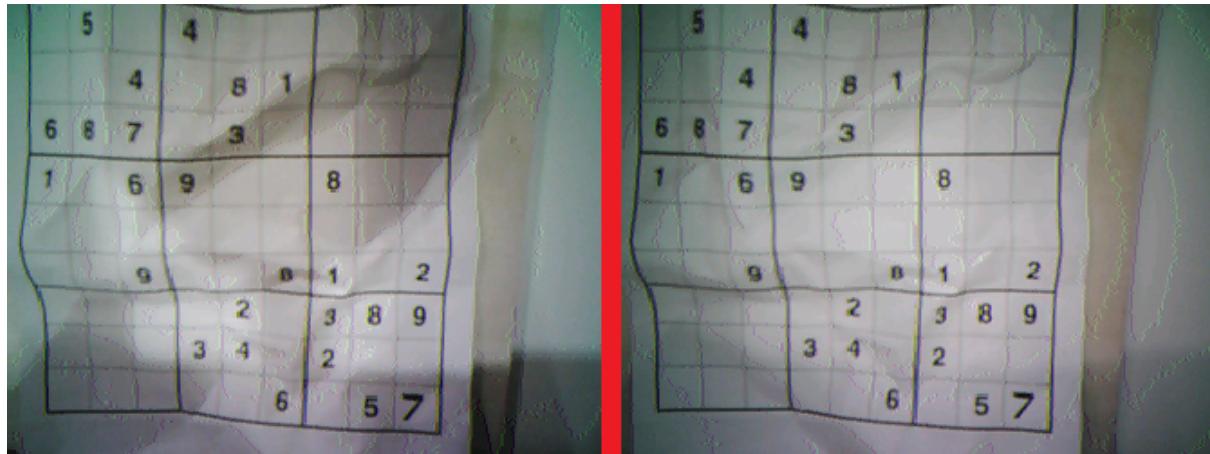
5.Testresultaten

In dit hoofdstuk worden de testresultaten besproken , die zijn uitgevoerd om de werking te verifiëren. Voor de testopstelling is de constructie dat in figuur 13 is te zien gebruikt.

5.1 belichting

Voor het testen van de belichting is het papier kruikelig gemaakt, om zodoende schaduwvorming te simuleren door oneffen oppervlakte in het sudoku papier. Door het led verlichting eerst uit te zetten in het resultaat in het linker afbeelding in figuur 14 en 15 verkregen. Vervolgens , door de led ring verlichting aan te zetten is het rechter in figuur 14 en 15 verkregen. Wat opvalt is dat schaduwvorming aanzienlijk is afgenumen. Deze afnamen is het gevolg van het licht dat verstroot op het oppervlakte van het papier straalt. Hierdoor wordt de schaduwzijde van de oneffenheid ook belicht . Wat in figuur 15 ook opvalt zijn de zwarte ringen die te zien zijn in de binaire afbeelding. Dit

is niet ten gevolg van schaduwvorming maar de slechte kwaliteit van de lens. Geconcludeerd kan worden dat het beeld kwaliteit van de sudoku aanzienlijk verbeterd als er difused licht wordt gebruikt op het sudoku papier.



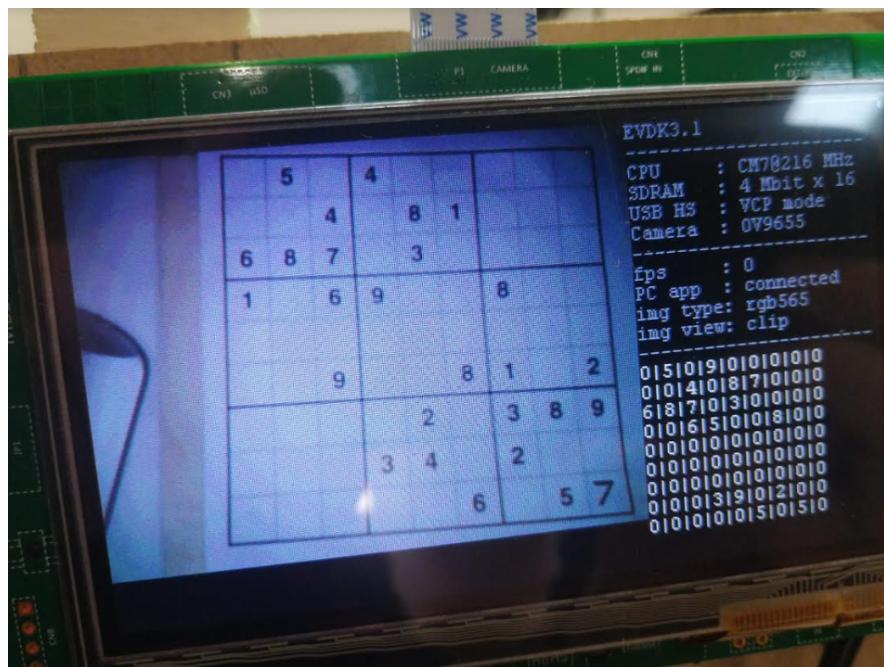
Figuur 14– normale afbeelding van sudoku met(rechts) en zonder belichting (links)



Figuur 15– threshold afbeelding van sudoku met(rechts) en zonder belichting (links)

5.2 herkennen cijfers

De testresultaat van het herkennen van de cijfers is in figuur 3 te zien. Voor het testen is het gehele herkenning algoritme gebruikt. Daarnaast is er een matrix gemaakt waarop de cijfers te zien zijn op de juiste locatie in het raster. Echter niet alle cijfers worden correct herkend. Dit is vooral te zien bij het getal 4. Deze wordt als 9 herkend. Een verklaring hiervoor kan zijn is de ruis in de camera. Door de ruis kunnen sommige cijfers waarschijnlijk toch niet helemaal goed gedetecteerd kunnen worden. De resultaten van de benchmark tonen aan de verwerkingssnelheid van het gehele algoritme ongeveer 0.1 frames/sec is.



Figuur 16– testresultaat herkenning cijfers

6. Conclusies en aanbevelingen

Allereerst is het functioneel ontwerp besproken. Welke verteld wat het systeem zou moet een kunnen doen. Hierbij is de het globale input -output proces besproken, de functionele specificaties (zit in de bijlage), het platform (stm32) en de userinterface.

Vervolgens is het technisch ontwerp besproken . Deze verteld over hoe het sudoku systeem werkt, oftewel de wijze waarop het sudoku systeem werkt. In het technisch ontwerp zit het systeem diagram van het sysdoku systeem, de stappen die zijn doorlopen (acquisitie , beeldverbetering , segmentatie , kenmerk bepaling en classificatie) , architectuur schema's(voor de hardware en software) en informatie over welke hardware elementen het in het systeem zitten.

Hoofdstuk realisatie omschrijft hoe de resultaten van de vison processen zijn bereikt. Eerst is verteld over de acquisitie opstelling. Vervolgens de implementatie van het beeldverwerkingsproces.

Het laatste hoofdstuk ging over de testresultaten. Hierbij zijn de belichting getest en het gehele sudoku vison algoritme. De resultaat van het testen van de belichting is dat er een significant verschil is tussen het onbelichte en belichte sudoku spel. Dit betekend dat schaduwvorming in het soduku papier aanzienlijk verminderd word met de directe diffusie verlichting.

Voor het testen is het gehele getal herkennings algoritme gebruikt. Daarnaast is er een matrix gemaakt waarop de cijfers te zien zijn op de juiste locatie in het raster. Echter niet alle cijfers worden correct herkend. Dit is vooral te zien bij het getal 4. Deze wordt als 9 herkend. Een verklaring hiervoor kan zijn is de ruis in de camera. Door de ruis kunnen sommige cijfers waarschijnlijk toch niet helemaal goed gedetecteerd kunnen worden. De resultaten van de benchmark tonen aan de verwerkingssnelheid van het gehele algoritme ongeveer 0.1 frames/sec is.

Het is aanbevolen om een andere camera te gebruiken. Hierdoor kunnen de kringen die ontstaan in het beeld verminderd worden en cijfers beter worden gedetecteerd.

Referenties

Bijlage

A Functionele specificaties

De functionele eisen voor het sudoku project zijn in tabel 1 weergegeven. Deze eisen gaan voornamelijk over wat het sudoku solver systeem zou moeten doen en wat het kan. De prioriteit van de eisen worden m.b.v. de “MoSCoW methode geclasseerd en is als volgt gedefinieerd:

M - must haves: Deze eisen moeten in het eindresultaat terugkomen;

S – should haves: Deze eisen zijn zeer gewenst voor het eindresultaat;

C – could haves: Deze eisen zullen allen aan bod komen als er genoeg tijd beschikbaar is;

W-won’t haves: Deze eisen zullen waarschijnlijk niet aan bod komen , maar kunnen wel interessant zijn voor een project in de toekomst.

Code	Functionele eis	Prioriteit	Stakeholder
F1	Het systeem moet aan alle vijf stappen van beeldverwerking doorlopen: acquisitie, verbeteren, segmentatie, kenmerkextractie en classificatie	M	Opdrachtgever
F2	Het systeem moet een sudokuspel van 100x100mm detecteren en herkennen.	M	Gebruiker/projectleden
F2.1	Het systeem moet de buitenste rand(1mm dikte) van het sudoku raster kunnen detecteren.	M	projectleden
F2.2	Het systeem moet voor elke cel een hand geschreven of geprinte cijfer met een lijndikte van 1mm herkennen	S	projectleden
F3	Het systeem moet (hand geschreven) cijfers uit het sudokuspel identificeren aan de hand van databases van verschillende gefotografeerde cijfers.	M	projectleden
F4	Het systeem moet het sudoku spel kunnen oplossen met behulp van de brute force algoritme.	S	Gebruiker/projectleden

F5	Het systeem moet de oplossing visueel kunnen weergeven.	M	Gebruiker/projectleider
----	---	---	-------------------------

Tabel 1 - Functionele eisen

B Algoritme operators

De code wordt apart gestuurd via de mail.

C Hoofd algoritme

```
/*
*****
* File      vision.c
* Author    H. Arends
* Version   V1.0
* Date     25 October 2019
* Brief    Implementation of vision processing operators
*
*          Lines 12 to 19 can be used for displaying information on the
LCD.
*
*          Use the function LTDC_ShowString() for this purpose.
*          Tip: use the function sprintf() for composing a string with
variables.
*
*          An example can be found in this file.
*
*          line  0 |EVDKm.n           |
*          line  1 |-----|
*          line  2 |CPU    :           |
*          line  3 |SDRAM  :           |
*          line  4 |USB HS:           |
*          line  5 |Camera:           |
*          line  6 |-----|
*          line  7 |fps    :           |
*          line  8 |PC app  :           |
*          line  9 |img type:          |
*          line 10 |img view:          |
*          line 11 |-----|
*          line 12 |           | Available for use
*          line 13 |           | Available for use
*          line 14 |           | Available for use
*          line 15 |           | Available for use
*          line 16 |           | Available for use
*          line 17 |           | Available for use
*          line 18 |           | Available for use
*          line 19 |           | Available for use
*          line 20 |Build (vision.c): |
*          line 21 |__DATE__ __TIME__|
```

```
* Permission is hereby granted, free of charge, to any person
obtaining a
 * copy of this software and associated documentation files (the
"Software"),
 * to deal in the Software without restriction, including without
limitation
 * the rights to use, copy, modify, merge, publish, distribute,
sublicense,
 * and/or sell copies of the Software, and to permit persons to whom
the
 * Software is furnished to do so, subject to the following conditions:
*
 * The above copyright notice and this permission notice shall be
included in
 * all copies or substantial portions of the Software.
*
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS
 * IN THE SOFTWARE.
```

```
*****
*****
*/
/* Includes
-----*/
#include "vision.h"
#include "ai.h"
#include "benchmark.h"
#include "ltdc.h"
#include "main.h"
#include "mem_manager.h"
#include "pc_interface.h"
#include "stdio.h"

// -----
// Defines
// -----
-----
```

```

// -----
-----
// Global variables
// -----
-----
static image_t *src;
static image_t *tmp;
static image_t *temp;
static image_t *dst;
int arraySudokuMatrix[9][9];
// 
// The define AI_ENABLED is enabled/disabled in Vision/config.h
// 
#if defined AI_ENABLED

AI_ALIGNED(4)
static ai_float nn_in_data[AI_NETWORK_IN_1_SIZE] = {0};

AI_ALIGNED(4)
static ai_float nn_out_data[AI_NETWORK_OUT_1_SIZE] = {0};

#endif

// -----
-----
// Local function implementation
// -----
-----

// This function is executed a single time at startup.
void vision_setup(void)
{
    // Show build information on the LCD
    char str[48] = "";
    //LTDC_ShowString(20, "Build (vision.c):", 100);
    //sprintf(str, "%s %s", __DATE__, __TIME__);
    //LTDC_ShowString(21, str, 100);

    // Allocate memory for the images
#if defined AI_ENABLED
    src = newRGB565Image(EVDK_IMAGE_WIDTH, EVDK_IMAGE_HEIGHT);
#else
    src = newBasicImage(EVDK_IMAGE_WIDTH, EVDK_IMAGE_HEIGHT);
#endif

    tmp = newBasicImage(EVDK_IMAGE_WIDTH, EVDK_IMAGE_HEIGHT);
    dst = newBasicImage(EVDK_IMAGE_WIDTH, EVDK_IMAGE_HEIGHT);
    temp = newBasicImage(450, 450);
    // Check if there is enough memory for all images
    if(dst == NULL)

```

```

    {
        LTDC_ShowString(12, "Not enough memory", 255);
        while(1)
        {;}
    }
}

// This function is executed each time an image has been acquired.
// cam points to an image of type RGB565.
void vision(image_t *cam)
{

    benchmark_t benchmark;

    // Show the original image
    benchmark_start(&benchmark, "pc_interface_showimg()");
    pc_interface_showimg(cam, "cam");
    benchmark_stop(&benchmark);
    pc_interface_send_benchmark(&benchmark);

    // Show the original image
    //pc_interface_showimg(img, "source image");
    benchmark_start(&benchmark, "copy()");
    copy(cam, src);
    benchmark_stop(&benchmark);
    pc_interface_send_benchmark(&benchmark);

    benchmark_start(&benchmark, "convertImage()");
    tmp->type = IMGTYPE_BASIC;
    convertImage(src, tmp);
    benchmark_stop(&benchmark);
    pc_interface_send_benchmark(&benchmark);
    // Make sure we are using a basic image for further processing

    pc_interface_showimg(tmp, "before otsu");
    thresholdOtsu(tmp,dst,DARK);
    pc_interface_showimg(dst, "otsu");
    //fillHoles(tmp,dst,EIGHT);
    removeBorderBlobs(dst,tmp,EIGHT);
    pc_interface_showimg(tmp, "removedblobs");




    //      //pc_interface_showimg(src, "basic image");
    //
    //      // Create a destination image
    //      image_t *dst = newBasicImage(src->cols, src->rows);
    //
    //      int32_t i, j;
    //
    //      //adaptive threshold
    //      nonlinearFilter(tmp,dst,AVERAGE,7);
    //      subtract(tmp, dst);
}

```

```

// threshold(dst, dst, 0, 1);
// invert(dst,dst);
// //pc_interface_showimg(dst, "thresh");
//
// //label blobs
// int32_t blobCount = (int32_t)labelBlobs(dst,dst,EIGHT);
// //pc_interface_showimg(dst, "label blobs");
//
// //find biggest blob
// uint32_t biggestBlob;
// uint64_t maxSize = 0;
// for(i = 1; i <= blobCount; i++)
// {
//     blobinfo_t info = {0,0,0,0};
//     blobAnalyse(dst,i,&info);
//     if((uint32_t)info.nof_pixels > maxSize){
//         biggestBlob = i;
//         maxSize = (uint32_t)info.nof_pixels;
//     }
// }
// //remove all non biggest blobs
// for(i = dst->rows -1; i > -1; i--)
// {
//     // Loop all pixels in the image (cols)
//     for(j = dst->cols -1; j > -1; j--)
//     {
//         if(getBasicPixel(dst,j,i) != biggestBlob)
//         {
//             setBasicPixel(dst,j,i,0);
//         }
//     }
// }
// //set biggest blob to 1
// for(i = dst->rows -1; i > -1; i--)
// {
//     // Loop all pixels in the image (cols)
//     for(j = dst->cols -1; j > -1; j--)
//     {
//         if(getBasicPixel(dst,j,i) != 0)
//         {
//             setBasicPixel(dst,j,i,1);
//         }
//     }
// }
// //pc_interface_showimg(dst, "biggest blob");
//
// //label blobs again in case of more than 256 blobs
// blobCount = (int32_t)labelBlobs(dst,dst,EIGHT);
// //pc_interface_showimg(dst, "label blobs2");
//
// //
// //find biggest blob
// maxSize = 0;
// for(i = 1; i <= blobCount; i++)

```

```

//      {
//          blobinfo_t info = {0,0,0,0};
//          blobAnalyse(dst,i,&info);
//          if((uint32_t)info.nof_pixels > maxSize){
//              biggestBlob = i;
//              maxSize = (uint32_t)info.nof_pixels;
//          }
//      }
//      //remove all non biggest blobs
//      for(i = dst->rows -1; i > -1; i--)
//      {
//          // Loop all pixels in the image (cols)
//          for(j = dst->cols -1; j > -1; j--)
//          {
//              if(getBasicPixel(dst,j,i) != biggestBlob)
//              {
//                  setBasicPixel(dst,j,i,0);
//              }
//          }
//      }
//      //pc_interface_showimg(dst, "biggest blob2");
//      uint32_t topLeftX,topLeftY;
//      uint32_t bottomLeftX, bottomLeftY;
//      uint32_t topRightX, topRightY;
//      uint32_t bottomRightX, bottomRightY;
//
//      //decide in which corner the point belongs
//      //int32_t sum = 0; //minimum sum = top left, maximum sum =
bottom right
//      int32_t diff; //maximum diff (x - y) = top right, minimum
diff = bottom left
//
//      int32_t maxDiff = 0;
//      int32_t minDiff = dst->rows + dst->cols;
//      int32_t maxSum = 0;
//      int32_t minSum = dst->rows + dst->cols;
//
//
//      // RB -> LT
//      // Loop all pixels in the image (rows)
//      for(i = dst->rows - 1; i > -1; i--)
//      {
//          // Loop all pixels in the image (cols)
//          for(j = dst->cols - 1; j > -1; j--)
//          {
//              if(getBasicPixel(dst,j,i) == biggestBlob)
//              {
//                  sum = j + i;
//                  diff = j - i;
//              }
//              //top right
//              if (diff >= maxDiff)
//              {
//                  maxDiff = diff;
//              }
//          }
//      }

```

```

//
//                                topRightX = j;
//                                topRightY = i;
//
//                                }
//                                //bottom left
//                                if (diff <= minDiff)
//                                {
//                                    minDiff = diff;
//                                    bottomLeftX = j;
//                                    bottomLeftY = i;
//                                }
//                                //bottom right
//                                if (sum >= maxSum)
//                                {
//                                    maxSum = sum;
//                                    bottomRightX = j;
//                                    bottomRightY = i;
//                                }
//                                //top left
//                                if (sum <= minSum)
//                                {
//                                    minSum = sum;
//                                    topLeftX = j;
//                                    topLeftY = i;
//                                }
//                            }
//                        }
//                    }
//                    image_t *warp = newBasicImage(1000, 1000);
//                    image_t *cutout = newBasicImage(450, 450);
//
//                    erase(warp);
//                    setSelectedToValue(warp,warp,0,255);
//
//                    // Warp image
//                    uint32_t pointsSrc[4][2] =
{{topLeftX,topLeftY},{topRightX,topRightY},{bottomLeftX,bottomLeftY},{bottomRightX,bottomRightY}};
//                    uint32_t pointsDst[4][2] =
{{0,0},{450,0},{0,450},{450,450}};
//
//                    copy = copyRectMinMax(src,minX,minY,);
//                    showImg(copy,"cut");
//                    contrastStretch(tmp,tmp,0,254);
//                    pc_interface_showimg(tmp, "stretched");
//                    warpBasic(tmp,warp,pointsSrc,pointsDst);
//                    pc_interface_showimg(warp, "warped");
//
//                    copyRect(warp,cutout,0,0,450,450);
//                    pc_interface_showimg(cutout, "cutout");
//                    interpolate(cutout);
//                    pc_interface_showimg(cutout, "interpolated");
//
//                    temp->type = IMGTYPES_BASIC;

```

```

//      convertImage(cutout, temp);
//
//      //adaptive threshold
//      nonlinearFilter(cutout,temp,AVERAGE,7);
//      subtract(cutout, temp);
//      threshold(temp, temp, 0, 3);
//      //deleteImage(temp);
//      invert(temp,temp);
//      pc_interface_showimg(cutout, "warpthresh");
//
//      nonlinearFilter(temp,cutout,AVERAGE,3);
//      pc_interface_showimg(cutout, "MAXMIN");
//      image_t *smallcutout = newBasicImage(50,50);
//      image_t *smallercutout = newBasicImage(38,38);
//      uint32_t m,n;
//      deleteImage(temp);
//      //rows
//      for (m = 0; m < 450; m += 50)
//      {
//          //columns
//          for (n = 0; n < 450; n += 50)
//          {
//              copyRect(cutout,smallcutout,m,n,50,50);
//          }
//      }
copyRect(smallcutout,smallercutout,6,0,38,38);
//                                //deleteImage(temp);
//                                //pc_interface_showimg(smallcutout,
"smallcutout");
//-----AI zooi hier met
smallcutout-----
#endif defined AI_ENABLED

        // Make sure we are using a basic image for further processing
        // by creating a copy of the image taken by the camera
removeBorderBlobs(smallerCutout, smallerCutout, EIGHT);
//pc_interface_showimg(smallerCutout, "Border");
benchmark_start(&benchmark, "labelBlobs()");
register uint32_t blobs = labelBlobs(smallerCutout,
smallerCutout, EIGHT);
pc_interface_showimg(smallerCutout, "labels");
benchmark_stop(&benchmark);
pc_interface_send_benchmark(&benchmark);

        // Select the initial line for displaying information
uint32_t ltdc_line = 13;

        // In order to keep the application responsive:
        // do not handle images with no blobs or images with a lot of
noise
if(blobs > 0 && blobs < 15)
{
    // Loop all the blobs
    for(uint32_t blob_num=1; blob_num <= blobs; blob_num++)
    {

```

```

blobinfo_t blobinfo;

blobAnalyse(smallerCutout, blob_num, &blobinfo);

// Only process blobs that have sufficient pixels
if(blobinfo.nof_pixels > 50)
{
    benchmark_start(&benchmark, "prepare data
for ai");

        // Find the blob's centroid
        int32_t cc, rc;
        centroid(smallerCutout, blob_num, &cc,
&rc);
        pc_interface_showimg(smallerCutout,
"doorheen");
        // Show a red 28x28 border in the src
image if the border is within
        // the src image boundaries
        if(cc >= 14 && cc < (dst->cols-14) && rc
>= 14 && rc < (dst->rows-14))
        {
            //
            for(int k=cc-14; k<cc+14; ++k)
                //
            {
                //
                if(k>=0 && k<src->cols)
                    //
                {
                    //
                    if(rc-14 > 0)
                        //
setRGB565Pixel(src,k,rc-14,(rgb565_pixel_t)0xF800);
                        //
                        //
                    if(rc+14 < src->rows)
                        //
setRGB565Pixel(src,k,rc+14,(rgb565_pixel_t)0xF800);
                        //
                }
                //
            }
            //
        }
        //
        //
        for(int k=rc-14; k<rc+14; ++k)
            //
        {
            //
            if(k>=0 && k<src->rows)
                //
        }

```

```

        //
        if(cc-14 >= 0)
        //

setRGB565Pixel(src,cc-14,k,(rgb565_pixel_t)0xF800);
        //
        //
if(cc+14 < src->cols)
        //

setRGB565Pixel(src,cc+14,k,(rgb565_pixel_t)0xF800);
        //
}

// Change origin
cc -= 14;
rc -= 14;

// Copy 28x28 pixels data to 784
float array for neural network
for(int r=0; r<28; ++r)
{
    for(int c=0; c<28; ++c)
    {
        //nn_in_data[(r*28)
+ c] = getBasicPixel(tmp,(cc+c),(rc+r)) / 255.0f;

if(getBasicPixel(smallerCutout,(cc+c),(rc+r)) == blob_num)
{
    nn_in_data[(r*28) + c] = 1.0f;
}
else
{
    nn_in_data[(r*28) + c] = 0.0f;
}
}

benchmark_stop(&benchmark);

pc_interface_send_benchmark(&benchmark);

// Run the neural network
benchmark_start(&benchmark,
"aiRun()");

aiRun(nn_in_data, nn_out_data, 1);
benchmark_stop(&benchmark);

pc_interface_send_benchmark(&benchmark);

```

```

        float most_confident_val = 0.0f;
        int most_confident_idx = -1;

        // Verify neural network output
        for(int q=1; q<10; ++q)
        {
            if(nn_out_data[q] >
most_confident_val)
            {
                most_confident_val =
nn_out_data[q];
                most_confident_idx =
q;
            }
        }

        arraySudokuMatrix[m/50][n/50] =
most_confident_idx;

        // Visualize the result on the LTCD
and with a benchmark
        //
char str[16];
        //
sprintf(str, "Detected %d", most_confident_idx);
        //
LTDC_ShowString(ltdc_line++, str, 255);
        //
sprintf(str, "Value %.5f", most_confident_val);
        //
LTDC_ShowString(ltdc_line++, str, 255);
        //
benchmark_start(&benchmark, str);
        //
benchmark_stop(&benchmark);
        //
pc_interface_send_benchmark(&benchmark);

    }

}

else
{
    arraySudokuMatrix[m/50][n/50] = 0;
}

}

// Send (annotated) images to PC app

```

```

//      pc_interface_showimg(src, "src");
//      pc_interface_showimg(tmp, "tmp");
//      pc_interface_showimg(dst, "dst");

// Displaying an image comes at the cost of the number of frames
that can
// be handled. If no image is displayed, the original camera
image
// is shown. This doesn't take much computation speed, because
DMA
// is used for that purpose.
LTDC_ShowImage(src);
//LTDC_ShowString(12, "AI demo", 255);

#endif
//}

//}

uint32_t ltdc_line = 12;
while(ltdc_line <=21)
{
    LTDC_ShowString(ltdc_line++, ", 255);
}
//    ltdc_line = 12;
//    for(int r=0; r<9; ++r)
//    {
//        char str[20];
//        sprintf(str, "%d|%d|%d|%d|%d|%d|%d|%d|%d",
arraySudokuMatrix[0][r],arraySudokuMatrix[1][r],arraySudokuMatrix[2][r]
, arraySudokuMatrix[3][r],arraySudokuMatrix[4][r],arraySudokuMatrix[5][r]
, arraySudokuMatrix[6][r],arraySudokuMatrix[7][r],arraySudokuMatrix[8][r]);
    LTDC_ShowString(ltdc_line++, str, 255);
}
//      pc_interface_showimg(src, "src");
//      pc_interface_showimg(tmp, "tmp");
//      pc_interface_showimg(dst, "dst");

}

```

