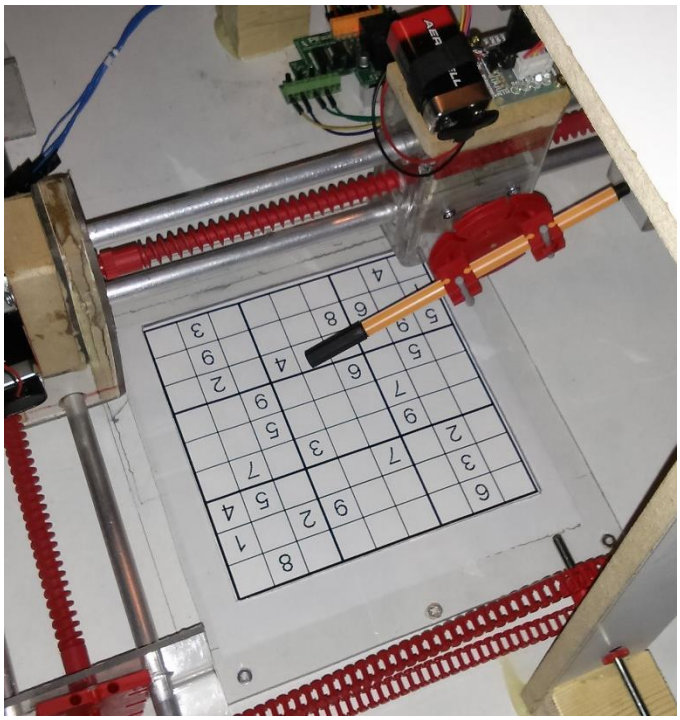


DE SUDOKUPLOTTER

Het ontwerp en bouwproces van een automaat die een sudoku kan lezen, oplossen én invullen.

Door Jos Feenstra & Joren Vrancken

11/12/2014

Wolfert Lyceum

INHOUDSOPGAVE

INTRODUCTIE

Een apparaat die een sudoku oplost. Het praktisch nut hiervan valt te betwijfelen, maar het maken van een complex apparaat als deze vormt wel een uitdagend project waarin we beide onze beste eigenschappen kunnen toepassen. Het begon met een idee om werktuigbouwkunde te combineren met programmeren. Op deze manier kon Jos een mechanisch apparaat ontwerpen en bouwen wat goed aansloot op zijn voorlopige studiekeuze Werktuigbouwkunde. Joren kon op zijn beurt een complex programma ontwikkelen wat weer goed paste bij zijn waarschijnlijke vervolgopleiding, Technische Informatica. Na enkele brainstorm sessies aan het begin van ons 5e schooljaar kwam Jos op het idee om een automaat te maken die een Rubiks Kubus kon oplossen. Joren had hier echter bezwaar tegen, aangezien het project dan lastig uitvoerbaar zou worden. Een paar weken daarna kreeg Joren het wel voor elkaar een algoritme te schrijven dat een sudoku op kon oplossen, en na dit met Jos te hebben besproken leek het ons beide een ideaal idee: Een automaat die een sudoku leest, oplost én invult.

VOORBEREIDING

Bij het opstellen van de onderzoeksvragen hadden wij het voorlopige idee een schrijfmachine te maken. Later leek dit apparaat van onze schetsen een zogenaamde plotter te heten. Een plotter is een printer die een pen gebruikt in plaats van een stempelsysteem, waardoor deze in staat is scherpe lijnen te tekenen.

Wat belangrijk is om te melden, is dat ons ontwerp een duidelijke input, verwerking en output heeft, zoals een automaat. Deze onderdelen staan ook redelijk los van elkaar, en kunnen apart behandeld worden. Daarom hebben wij gekozen om het gehele apparaat (mechanica, motoren, frame voor camera, Raspberry Pi en toebehoren) een automaat te noemen, en de deelvragen ook onder de drie onderdelen ervan te laten vallen. Ons project is echter ook een plotter, waardoor we met het woord plotter ook naar de automaat zullen refereren.

HOOFDVRAAG:

"Hoe laat je een machine automatisch een sudoku puzzel lezen, oplossen en invullen?"

De hoofdvraag is een "hoe"-vraag, aangezien wij niet de eerste zullen zijn die een machine maken om een sudoku op te lossen. We weten dus al dat het mogelijk is, dus de vraag blijft hoe we het gaan maken. Ook zijn deze hoofd en deelvragen in de loop van ons profielwerkstuk met opzet niet aangepast. Daarom wordt de plotter hier nog simpelweg "machine" genoemd.

DEELVRAGEN:

- **Input: lezen**
  - Hoe ontvangt de machine informatie over de sudoku?
  - Hoe herkent de machine de informatie die hij nodig heeft?
  - Hoe wordt deze informatie overgezet in een vorm die bruikbaar is?
- **Verwerking: oplossen**
  - Hoe lost de machine alle mogelijke sudoku's systematisch op?
- **Output: invullen**
  - Hoe wordt de verkregen informatie toegepast bij het oplossen?
  - Hoe laat je een machine een sudoku fysiek invullen?

DE EISEN

De hoofd- en deelvragen hebben wij uiteindelijk vertaald in eisen voor de drie onderdelen. Het idee was dat als wij aan deze eisen zouden kunnen voldoen, we gemakkelijk de hoofd- en deelvragen kunnen beantwoorden. Het bood ook goeie richtlijnen voor het uiteindelijke ontwerp waar wij destijds nog mee bezig waren.

INPUT:

- Onze automaat zal een aantal knoppen nodig hebben (minimaal 2), om de processen aan te sturen.
- De sudoku die opgelost moet worden moet een standaard formaat hebben. Dit vel moet goed op z'n plek blijven.
- Er moet een foto gemaakt worden van de sudoku. Deze foto zal doorgestuurd worden naar de on-board computer.
- De foto moet goed belicht en van redelijke kwaliteit zijn.
- De foto moet de gehele sudoku fotograferen en de sudoku zelf moet duidelijk zijn.
- Met behulp van een door ons geschreven programma moeten de getallen met bijbehorende locatie worden herkend en opgeslagen.

VERWEKING:

- De gevonden sudoku moet opgelost worden met behulp van een zelfgeschreven programma.
- Alle mogelijke oplossingen van een sudoku moeten gevonden worden.
- Alle mogelijke sudoku's moeten opgelost worden.
- Dit moet kunnen in een redelijke tijdsperiode.

OUTPUT:

- De getallen met bijbehorende posities die gevonden zijn bij de verwerking moeten verschijnen op de onopgeloste sudoku van de input.
- Om dit te realiseren zal de automaat over een mechaniek beschikken om een viltstift over een X en een Y coördinaat te bewegen, met behulp van 2 motoren.
- precisie is essentieel voor de motoren. Ze zullen elke keer weer exact hetzelfde resultaat moeten leveren, met minimale afwijkingen.
- deze viltstift zal met behulp van pneumatiek op het vel gedrukt worden. [1]
- ook moet het ontwerp beschikken over een motor die de 'diskdrive' zal laten bewegen, om een nieuw sudoku-vel erop te laten bevestigen.[2]
- ten slotte zijn er nog een aantal led lampjes nodig om bepaalde processen aan te tonen die de computer uitvoert

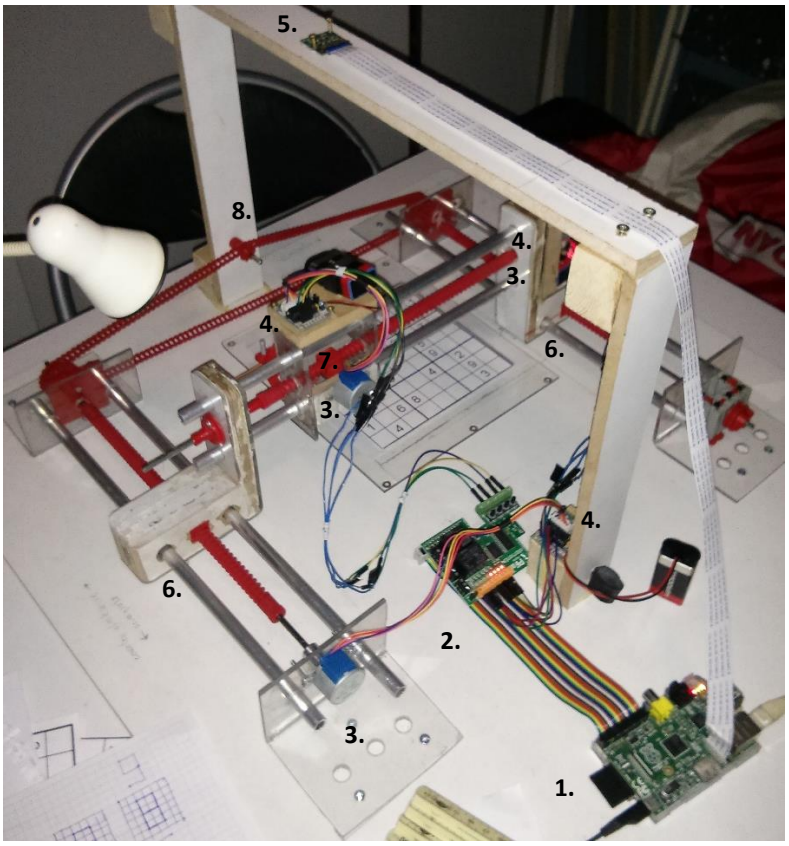
---

1 Deze eis hebben wij later geschrapt, aangezien wij een ontwerp bedachten waarin het pneumatisch systeem vervangen kon worden door een enkele motor, waardoor de pen veel makkelijker aangestuurd kan worden.

2 Deze eis hebben wij ook ingetrokken, toen we erachter kwamen dat een extra motor voor dit systeem veel problemen met zich meebracht, en deze functie eigenlijk geen extra waarde toe zou voegen aan onze automaat.

HET RESULTAAT

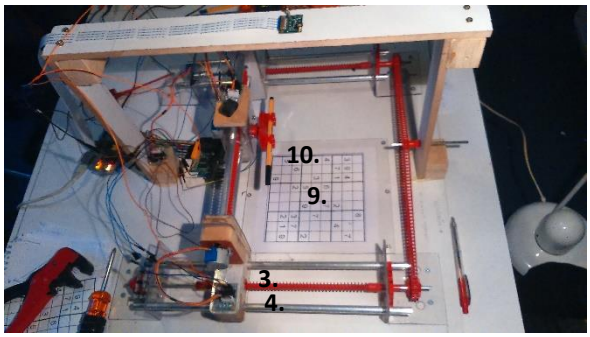
Uiteindelijk is dit (afb.1) de automaat die wij gebouwd hebben. De plotter is in staat een pen horizontaal, verticaal en op en neer te bewegen met behulp van 3 stappenmotoren. Ook bevat de plotter een camera om wat er op de plotter aanwezig is, in dit geval een sudoku, vast te leggen. De motoren en camera zijn bevestigd aan een Raspberry Pi, een soort miniatuurcomputer. De automaat is volledig “van scratch” af aan opgebouwd. Daarmee bedoelen we dat, behalve apparatuur als de Raspberry Pi, alles zelf is ontworpen en gemaakt. In de volgende paragrafen zullen specifieke details van de automaat nader uitgelegd worden.



Afbeelding 1a: het achteraanzicht van de volledige plotter

Legenda

1. Raspberry Pi
2. Piface: General input/output device
3. Stappenmotor
4. ULN2003 chip
5. Camera
6. A – Module
7. B – Module
8. Ketting
9. Frame met ingeschoven sudoku
10. Draaischijf met pen.



Afbeelding 1b: Zijaanzicht van de plotter

DE ONDERDELEN

De bouwmaterialen van onze plotter zijn zorgvuldig uitgekozen. Zo hebben we bijvoorbeeld plexiglas gebruikt aangezien het eenvoudig te buigen is. Bovendien zorgt de doorzichtigheid ervoor dat technische aspecten, zoals de draaischijf van de pen, goed zichtbaar zijn. Verder bestaat de basis van de plotter uit hout, aangezien het in het algemeen zeer geschikt bouwmateriaal is, en enkele aluminium buizen. De meeste bevestigingen zijn gemaakt met constructielijm, tweecomponentenlijm en schroeven.

Afbeelding 2: Diverse Fischertechnik onderdelen.

Voor de mechanische aspecten van de plotter hadden we echter veel verfijnde onderdelen nodig om de vereiste precisie mogelijk te maken. Dit hebben wij gerealiseerd doormiddel van *Fischertechnik*, wat beschreven kan worden als LEGO voor technische doeleinden (afb. 2). Het bevat onder andere tandwielen, kettingen, wormwielen en onderdelen om een framewerk mee te bouwen. Fischertechnik kwam vooral goed van pas bij de drie wormwielen, de ketting en de draaischijf voor de pen.

DE ELEKTRONICA

De elektronica bestaat uit een Raspberry Pi met een besturingsmodule, in totaal 3 stappenmotoren (afb. 3) en een camera. Afgezien van traagheid bij het moeilijkere rekenwerk, is de Raspberry gelijk aan een normale computer en dus in staat code op te slaan en toe te passen. Ook de Raspberry heeft een specifieke takenlijst:

- Motoren aansturen.
- Camera gebruiken en foto opslaan.
- Code opslaan en toepassen.
- Code aansturen door middel van knoppen.
- Met led lampjes aangeven welk proces er gaande is.

Afbeelding 3: Een stappenmotor met bijbehorende ULN2003 chip.

Ook de stappenmotoren zijn uitgekozen met een goede rede. Normale DC-motoren zijn alleen te programmeren door ze een X-aantal seconde stroom te geven. Ook hebben ze een kleine “wind up time”, en rollen ze een klein stukje door. DC-motoren zijn daarom niet te vertrouwen voor het nauwkeurige werk van het schrijven van een getal.

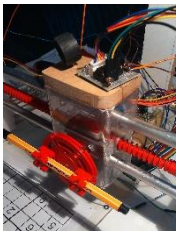
De motoren die wij gebruiken kunnen extreem precies geprogrammeerd worden, door bijvoorbeeld te zeggen “draai 180 graden met de klok mee”. De motor zal dan nooit meer kunnen uitwijken dan 0,703 graden van dit punt. Ook starten en stoppen deze motoren erg nauwkeurig. Als we in dit zelfde voorbeeld de motor weer 180 graden terug laten draaien zal de motor op precies hetzelfde punt eindigen. Samengevat: Stappenmotoren zijn precies en consistent.

DE MECHANICA

Dan nu het mechanisme van de plotter. Het is aan te raden om de foto en legenda bij het begin van dit hoofdstuk te raadplegen bij het lezen van deze paragraaf. Zoals eerder uitgelegd zijn de taken van de plotter:

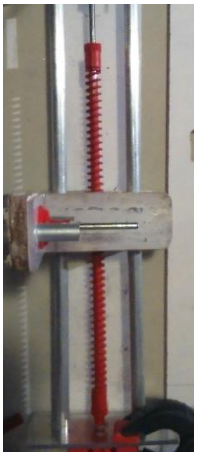
- De pen van links naar rechts en van rechts naar links te bewegen, in andere woorden: De ‘’X’’ van de pen aanpassen.
- De pen van voor naar achter en van achter naar voor te bewegen, in andere woorden: De ‘’Y’’ van de pen aanpassen.
- De pen op het papier te drukken en van het papier af te halen, oftewel de hoogte of “Z” van de pen aanpassen.

Om deze drie bewegingen aan te sturen zijn er dus drie respectievelijke motoren nodig, de X- , Y- en Z-motor. Het mechanisme voor de Z-motor is een relatief eenvoudig systeem. Deze motor is aangesloten op een schijf, waar vervolgens de pen op is bevestigd (zie afb. 4). Door nu de schijf links- of rechtsom te draaien kan de pen op en neer bewogen worden. Een belangrijk detail van dit Z-mechanisme is dat de schijf niet is vastgelijmd, maar vastgeklemd zit op de as van de Z-motor. Hierdoor is het mogelijk om de motor door te laten draaien als de pen al op het papier gedrukt zit, wat resulteert in een lichte druk die de pen uitoefent op het papier. Deze druk is essentieel voor de plotter om met duidelijke, rechte lijnen te schrijven.



Afbeelding 4: De B-module op de rails.

Nu de pen op het papier kan worden gedrukt moet de plotter in staat zijn de X en Y van de pen aan te passen. Deze verplaatsing moet behaald worden uit de rotatie van de X- en Y-motoren. Om dit voor elkaar te krijgen, hebben wij gebruik gemaakt van Fischertechnik wormwielen in samenwerking met aluminium buizen. De wormwielen hebben een bijpassende “mal”, waar de schroef precies doorheen past. Door deze mal vast te zetten op een blok dat over de buizen kan glijden (zie afb. 4 en 5), zal het blok heen en weer te bewegen wanneer de schroef gedraaid wordt.



Afbeelding 5: Een A-module op de rails.

Deze techniek is in totaal drie maal toegepast op de plotter (zie afb 1). De Z-motor met de draaischijf vormt samen met een mal voor de schroef en gaten voor de buizen de zogenaamde B-module. Door deze module zijn vervolgens twee buizen en een schroef geschoven, wat als een soort rails beschouwd kan worden. De rails is op zijn beurt aan weerskanten bevestigd aan twee ‘’connectieblokken’’, die wij A-modules hebben genoemd. De X-motor bevindt zich op een van de twee blokken, en is aangesloten op de schroef, wat ervoor zorgt dat de X-motor de B-module naar links/rechts kan bewegen.

Dit gehele apparaat (de B-module, de rails, de X-motor en de twee A-modules) bevindt zich weer op 2 railsstukken die bevestigd zijn op de grondplaat. Ieder railsstuk bestaat elk ook weer uit twee buizen en een schroef. Door een ketting tussen de twee wormwielen te laten lopen, en één van de twee wormwielen aan te sluiten op de Y-motor, kan nu het hele schrijfapparaat naar voren of achteren verplaatst worden bij het draaien van de Y-motor.

De ketting en tweede schroef voor de Y-beweging is noodzakelijk om de beweging goed te laten verlopen. Bij het testen van een prototype zonder deze onderdelen namen wij waar dat de A-module zonder schroef dan met achterstand meegetrokken werd door de A-module met schroef. Dit kwam doordat de “trekkracht” die de schroef uitoefent niet gelijk verdeeld was over het gehele schrijfapparaat. Er moest dus een tweede schroef komen met of een ketting, of een extra motor. Dit laatste hebben wij niet gekozen, omdat de Raspberry Pi moeite kreeg met het aansturen van een vierde motor.

DE AUTOMAAT IN ACTIE

Allereest moet de onopgeloste sudoku worden ingevuld op een vel met standaard afmetingen, aangezien het schrijfsysteem hierop gebaseerd is. Sudoku’s uit bijvoorbeeld de krant kunnen opgelost worden, maar om ze ook te laten invullen moeten ze aan de standaard maten voldoen. Nadat de sudoku op een standaardlocatie is bevestigd, kan het oplosproces worden gestart met het drukken van een knop op de Pi. Dit proces werkt als volgt:

Ten eerste zal er een foto van de sudoku worden gemaakt. De sudoku moet goed belicht zijn om alle getallen te kunnen lezen, dus hij moet bij deze stap worden beschenen door een lamp. De foto wordt vervolgens doorgestuurd naar de Raspberry Pi. Hier zal het complete verwerkingsproces ervoor zorgen dat de foto uiteindelijk wordt omgezet in een stappenplan voor de motoren. De tussenstappen hiervan zullen in een volgend hoofdstuk worden uitgelegd.

Dit stappenplan stuurt de mechanica van de vorige paragraaf aan. De pen zal zich langs elk vakje bewegen en het gewenste getal invullen indien nodig. De getallen die de plotter schrijft lijken op digitale getallen, omdat het simpele, blokvormige getallen moesten worden die geschreven konden worden zonder de pen op te tillen. Als het schrijfproces klaar is zal de pen terugkeren naar een standaardpositie, waarna de opgeloste sudoku weer losgemaakt kan worden.

De foto maken, herkennen en oplossen van de sudoku duurt maar enkele seconden. Het invullen van de sudoku duurt echter wel een kwartier tot halfuur. Dit komt door het feit dat de stappenmotoren relatief langzamer zijn dan bijvoorbeeld DC-motoren. Stappenmotoren hebben precisie en consistentie als uitgangspunt, en dus niet snelheid. Eén sudoku vakje overbruggen duurt daarom ongeveer 14 seconde, dus het is begrijpelijk dat het schrijven zo lang duurt.

VERWERKING: DE CODE

De programmeerkant van het verhaal bestaat voornamelijk uit drie delen:

- Een foto maken en de sudoku hierin herkennen via Optical Character Recognition.
- De sudoku oplossen door Donald Knuth's Algorithm X toe te passen.
- De sudoku invullen op papier door de Piface Digital (een general-purpose input output chip voor de Raspberry Pi) te gebruiken.

Dit zijn drie redelijk losstaande onderwerpen, met weinig overlap; van beeldbewerking tot motors aansturen met code.

Alle code is geschreven in Python 2.7. Er is bewust voor Python gekozen, omdat deze taal een paar voordelen heeft. Alle code moet lopen op de Raspberry Pi. Python is een van de best ondersteunde talen op de Raspberry Pi. Ook is Python relatief eenvoudig en zeer flexibel, wat een groot voordeel is aangezien dit project uit verschillende aspecten bestaat. De keuze om versie 2.7 te gebruiken is ook bewust geweest. Deze versie is op dit moment nog de best ontwikkelde versie, hierdoor wordt de code op de meest efficiënte en snelle manier gedraaid. In tests bleek dat bijvoorbeeld de code in Python 3.4 5 seconden langzamer is. Dat 2.7 de meest doorontwikkelde versie is, is al helemaal het geval op Linux. In Linux wordt bijna alle Python geprogrammeerd in Python 2.7.

HERKENNEN: OPTICAL CHARACTER RECOGNITION

OCR is een moeilijk proces. Het is ook het wankelste punt in de code, omdat de code geen controle heeft over het belangrijkste gedeelte van OCR: het beeld (of in het geval van de plotter, de foto). Met software kan er extra contrast en scherpte toegevoegd worden, maar als een foto onderbelicht is kan software weinig doen. Gelukkig betekent moeilijk niet onmogelijk. Het is namelijk zeer goed mogelijk met genoeg debuggen en testen (weten wij nu uit ervaring).

Wij hebben het proces van herkennen opgedeeld in acht stappen.

1. EEN FOTO MAKEN

Alle code wordt uitgevoerd op de Raspberry Pi. Hiervoor bestaat een cameramodule. We hebben dus de Raspberry Pi Camera aangesloten en zo kunnen wij een foto maken. De camera kan eenvoudig aangeroepen worden via de command line. Dit gebeurt ook in de code met het volgende commando:

```
sudo raspistill -roi .4,0.05,1,1 -cfx 128:128 -w 600 -h 450 -q 100 -ex night -e jpg -sh 100 -co 100 -o image.jpg
```

Dit commando maakt een foto, bewerkt deze en slaat deze op.

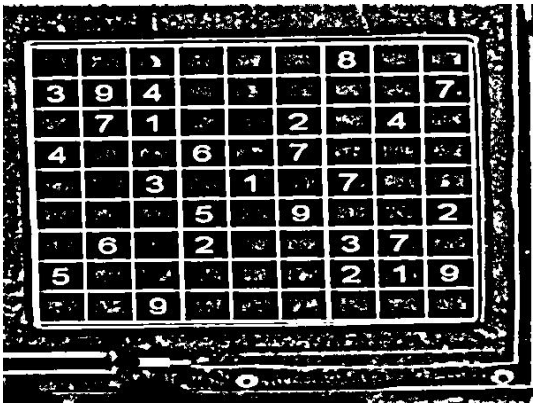
- `sudo raspistill`: raspistill is het daadwerkelijke programma dat aan de camera gekoppeld zit. Dit programma is gemaakt door de makers van de Rasperry Pi. Sudo zorgt ervoor dat het met genoeg rechten wordt uitgevoerd.
- `roi .4,0.05,1,1:`
  de ROI (region of interest) flag wordt gebruikt om alleen het gedeelte van de foto te pakken waar we geïnteresseerd in zijn, de sudoku.
- `cfx 128:128:`
  Zorgt ervoor dat de foto zwart-wit wordt.
- `w 600 -h 450:`
  De cameramodule maakt detailfoto's met heel hoge resolutie, dit hebben we niet nodig. Het OCRproces gaat zelfs een stuk sneller met een lagere resolutie.
- `ex night:`
  De ex flag zet de exposure optie. Hiermee bepaal je hoe licht je foto wordt, met de night optie wordt de foto lichter.
- `co 100 -sh 100 -q 100:`
  Deze flags zorgen ervoor dat het contrast, de scherpte en de kwaliteit maximaal zijn.
- `e jpg -o image.jpg:`
  We slaan de foto op als JPEG bestand genaamd “image.jpg”.

2. DE VERKREGEN FOTO BEWERKEN

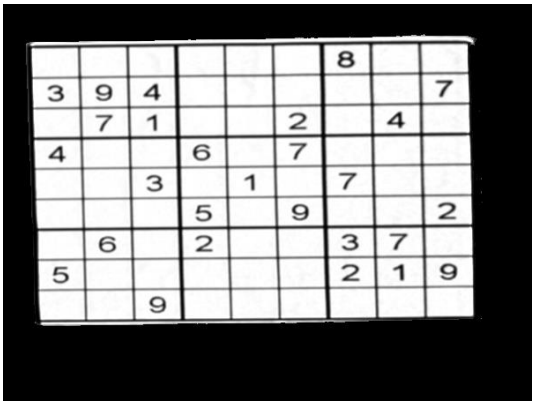
Nu de foto is gemaakt lezen we deze in, in het programma. We voeren na het importeren een gaussian blur uit om de foto wat gladder te maken. Daarna zetten we de foto om naar grayscale.

3. DE SUDOKU PUZZEL VINDEN

Eerst maken we een threshold image van de foto. Hierbij kijken we naar iedere pixel, ligt deze boven een bepaalde waarde dan wordt hij wit, zo niet dan zwart. In deze threshold versie van de foto zoekt het programma naar alle contouren en neemt aan dat de grootste de sudoku puzzel is.



Afbeelding 6: de threshold image



Afbeelding 7: de gevonden puzzel

4,5. VERTICALE EN HORIZONTALE LIJNEN VINDEN IN DE PUZZEL

Stap 4 en 5 zijn eigenlijk hetzelfde alleen gaat stap 4 over de X-as en stap 5 over de Y-as. Als eerste creëren we een langwerpige rechthoek (staand voor verticale lijnen en liggend voor horizontale lijnen). We bewaren alles wat op deze rechthoek lijkt en verwijderen alles wat niet op de rechthoek lijkt. Om de goede lijnen eruit te filteren, zoeken we weer alle contouren. We behouden de contouren met de juiste afmetingen.

6 ALLE KRUISPUNTEN VINDEN

Dit is relatief eenvoudig als je de horizontale lijnen en de verticale lijnen hebt. Het programma vergelijkt alle lijnen met elkaar en de overeenkomende punten zijn kruispunten.

7 PUNTEN OPTIMALISEREN

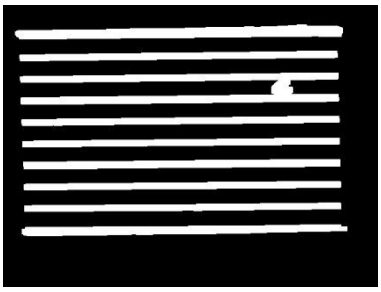
In het plaatje met alleen de kruispunten vindt het programma vervolgens alle contouren. Op deze manier vindt het de daadwerkelijke coördinaten van de kruispunten.

8 HET HERKENNEN VAN AFZONDERLIJKE NUMMERS

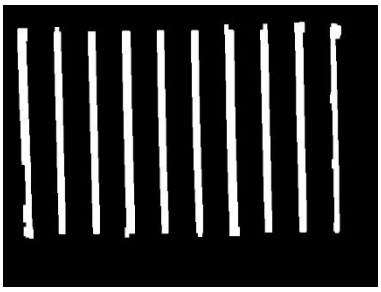
Nu kan het herkennen van de cijfers beginnen. Nu we de coördinaten van alle hoekpunten (de kruispunten) hebben, kunnen we ook alle hokjes afzonderlijk uit de originele foto snijden. Als we de afzonderlijke hokjes hebben, kunnen we dus de afzonderlijke cijfers herkennen. Het programma knipt zeven pixels aan alle kanten af, zodat de rand niet meer op de vakjes staat. Dan krijg je uiteindelijk vergelijkbare vakjes zoals hieronder of leeg.

Het daadwerkelijke herkennen gaat via een zogeheten “library”. Dit is code die vooraf door iemand anders is geschreven, zodat een andere programmeur deze niet meer hoeft te schrijven. Deze library genaamd Python-Tesseract, gebruikt weer een ander programma genaamd Tesseract. Tesseract is een OCR programma geschreven door een development team binnen HP en later overgenomen en verbeterd door programmeurs bij Google.

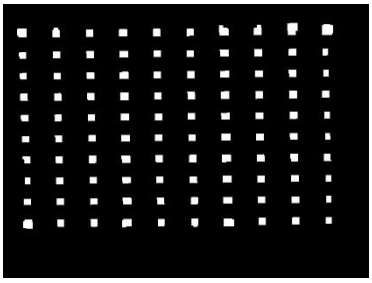
Als alles is herkend, wordt het resultaat doorgestuurd naar het tweede gedeelte van het programma, de oplosser.



Afbeelding 8: horizontale lijnen gevonden uit de sudoku



Afbeelding 9: de verticale lijnen



Afbeelding 10: de kruispunten gevonden door de vorige twee afbeeldingen te kruisen

OPLOSSEN: ALGORITME X

Er zijn verschillende algoritmes om sudoku's op te lossen. Een van de meest gebruikte is algoritme X bedacht door Donald Knuth. Dit is een recursief algoritme dat elke oplossing vindt voor het exacte overdekking(exact cover) probleem. Het exact cover probleem gaat als volgt:

Stel je hebt zes verzamelingen van nummers. Deze zes verzamelingen noem je A t/m F. Deze zes verzamelingen bevatten een aantal nummers uit de nummerreeks 1 /tm 7. Bijvoorbeeld:

A = 1, 2, 3 B = 1, 2, 4, 7 C = 4, 6, 7 D = 3, 5, 6 E = 3, 6 F = 7

Dit laten we zien door middel van het volgende schema (matrix):

X	**1**	**2**	**3**	**4**	**5**	**6**	**7**
A	1	1	1	0	0	0	0
B	1	1	0	1	0	0	1
C	0	0	0	1	0	1	1
D	0	0	1	0	1	1	0
E	0	0	1	0	0	1	0
F	0	0	0	0	0	0	1

Nu is het idee van exact cover dat je alle verzamelingen vindt die samen precies 1 t/m 7 bezitten. In dit voorbeeld zijn dat B, D. Algoritme X vindt deze oplossingen systematisch. Dit gaat via zes recursieve stappen. Als we beginnen is de deeloplossing leeg.

STAP 1: Als de matrix leeg is, is de deel oplossing, de oplossing. Stop het algoritme.

STAP 2: Kies de kolom met de minste 1'en

STAP 3: Kies een rij uit de gekozen kolom van stap 2 die een 1 bevat.

STAP 4: Voeg de gekozen rij toe aan de deel oplossing.

STAP 5: Voor iedere vakje in de rij gekozen in stap 3, verwijder alle kolommen en alle rijen waarin een 1 voorkomt.

STAP 6: Ga naar stap 1 met de verkleinde matrix.

Voorbeeld:

Ronde 1:

Stap 1: De matrix is niet leeg, dus het algoritme gaat door.
Stap 2: Het laagste aantal 1'en in alle kolommen is 1, namelijk kolom 5.
Stap 3: Omdat kolom 5 maar 1 rij bevat met een 1, moet die gekozen worden. Dit is rij D.
Stap 4: D is nu deel van de tijdelijke oplossing.
Stap 5: Rij D heeft een 1 in kolommen 3, 5 en 6. Kolom 3 heeft een 1 in rijen A, D en E. Kolom 5 heeft een 1 in rijen D. En als laatste heeft kolom 6 een 1 in rijen C, D, E. Dus rijen A, C, D en E worden samen met kolommen 3, 5 en 6 verwijderd.
Stap 6: Nu hebben we de volgende matrix over. Hiermee gaan we verder.

Ronde 2:

Stap 1: De matrix is niet leeg, dus het algoritme gaat door.
Stap 2: Het laagste aantal 1'en in een kolom is een. Kolom 1,2 en 4 hebben allemaal een 1 in zich. We kiezen de eerste dat is kolom 1.
Stap 3: De enige rij met een 1 in zich is B, dus rij B wordt gekozen.
Stap 4: Rij B gaat in de tijdelijke oplossing.
Stap 5: Rij B heeft een 1 in kolom 1, 2, 4 en 7. Rij 1,2 en 4 hebben allemaal alleen maar een 1 in rij B. Kolom 7 heeft twee 1'en, in rij B en F. Rijen 1, 2, 4 en 7 worden dus verwijderd samen met kolommen B en F
Stap 6: De matrix is helemaal leeg.

Ronde 3:

Stap 1: Omdat de matrix nu leeg is, kunnen we niet meer verder en is de tijdelijke oplossing de definitieve oplossing. Dat klopt, de tijdelijke oplossing is B en D, en deze zijn samen 1 t/m 7.

Ronde 2

Stap 2:

X	1	2	4	7
B	1	1	1	1
F	0	0	0	1

Stap 3:

X	1	2	4	7
B	1	1	1	1
F	0	0	0	1

Stap 5:

X	1	2	4	7
B	1	1	1	1
F	0	0	0	1

Ronde 1:

Stap 2:

X	1	2	3	4	5	6	7
A	1	1	1	0	0	0	0
B	1	1	0	1	0	0	1
C	0	0	0	1	0	1	1
D	0	0	1	0	1	1	0
E	0	0	1	0	0	1	0
F	0	0	0	0	0	0	1

Stap 3:

X	1	2	3	4	5	6	7
A	1	1	1	0	0	0	0
B	1	1	0	1	0	0	1
C	0	0	0	1	0	1	1
D	0	0	1	0	1	1	0
E	0	0	1	0	0	1	0
F	0	0	0	0	0	0	1

Stap 5:

X	1	2	3	4	5	6	7
A	1	1	1	0	0	0	0
B	1	1	0	1	0	0	1
C	0	0	0	1	0	1	1
D	0	0	1	0	1	1	0
E	0	0	1	0	0	1	0
F	0	0	0	0	0	0	1

Stap 6:

X	1	2	4	7
B	1	1	1	1
F	0	0	0	1

ALGORITME X MET SUDOKU'S

Op het eerste gezicht lijkt dit algoritme niet van toepassing te zijn op sudoku's, maar het exact cover probleem kan in bijna alle puzzels gevonden worden, dus kunnen algoritme die het exact cover probleem oplossen gebruikt worden om bijna alle puzzels op te lossen.

Als je als verzamelingen alle mogelijke combinaties van nummers en vakje in een sudoku neemt en als nummerreeks een aantal voorwaarden waaraan een sudoku oplossing moet voldoen, is een sudoku een exact cover probleem.

De verzamelingen worden dan alle nummers in de vorm van (Rij van nummer, Kolom van nummer, nummer zelf). Dus als in een vakje in rij 5 en kolom 8 een 3 staat, staat het als verzameling (5, 8, 3). Omdat de ingevulde vakjes al vaststaan, wordt er dus alleen (5, 8, 3) toegevoegd aan alle verzamelingen. Als het vakje leeg is, worden er 9 verzamelingen toegevoegd (Rij van nummer, Kolom van nummer, 1) t/m (Rij van nummer, Kolom van nummer, 9). Dat zijn alle verzamelingen.

Als nummerreeks (of kolommen in de matrix) neem je voorwaarden. Iedere sudoku-oplossing moet aan een aantal voorwaarde voldoen:

- In iedere rij moeten de nummers 1 t/m 9 staan.
- In iedere kolom moeten de nummers 1 t/m 9 staan.
- In iedere 3x3 vakken blok moeten de nummers 1 t/m 9 staan.
- Ieder vakje moet gevuld zijn.

We krijgen dus een lange lijst van voorwaarden. Deze lijst bestaat uit (rij, 1, 1) t/m (rij, 9, 9), (kolom, 1, 1) t/m (kolom, 9, 9), (blok, 1, 1, 1) t/m (blok, 3, 3, 9) en (vakje, 1, 1) t/m (vakje, 9, 9). We rekenen voor iedere rij in de verzameling uit aan welke voorwaarde deze precies voldoet. Dus het voorbeeld van hierboven, (5, 8, 3), voldoet aan de volgende voorwaarde:

- een 3 in rij 5: (rij, 5, 3)
- een 3 in kolom 8: (kolom, 8, 3)
- een 3 in cel blok (2, 3): (blok, 2, 3, 3)
- vakje (5, 8) gevuld: (vakje, 5, 8)

In de matrix zou (5, 8, 3) bij deze voorwaarden dus een 1 krijgen.

Als we dat voor iedere rij in de verzameling doen krijgen we een nieuwe matrix. We hebben nu een sudoku in de vorm van het exact cover probleem. Hier kunnen we algoritme x op los laten en snel alle oplossingen van een sudoku vinden.

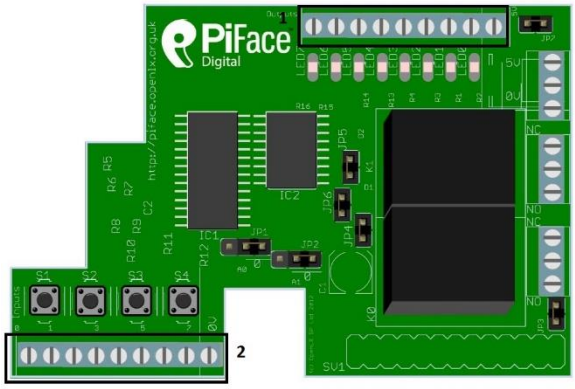
SCHRIJVEN: PIFACE DIGITAL

Dit zou de makkelijkste van de drie delen geweest moeten zijn, maar het bleek uiteindelijk het moeilijkste. Dit komt vooral omdat dit gedeelte voor een groot deel uit hardware bestaat. Hardware debuggen is ingewikkeld en tijdrovend.

Er moesten in totaal drie stappenmotoren bestuurd worden met de PiFace. Hiernaast is het boven aanzicht van de PiFace te zien. Omdat de stappenmotoren vier aansluitingen per stuk hebben moeten er twaalf outputs gebruikt worden. Omdat er maar acht outputs zijn, moeten er dus nog vier bijkomen.

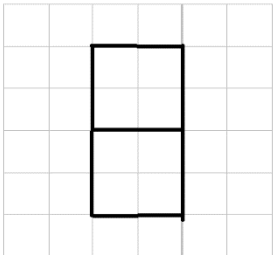
Dit is mogelijk met een kleine hack. Door een paar lijnen code toe te voegen kan namelijk de IO richting van de input pins handmatig ingesteld worden. In andere woorden: de inputs worden outputs.

De code zelf is relatief simpel, deze bestaat uit de Piface initialiseren met een specifieke library, 3 motorobjecten aanmaken en dan op het goede moment de goede motor aansturen.



Afbeelding 11: de PiFace. Het bevat output pins (1), input pins (2), acht led lampjes en vier drukknoppen.

HET OPLOSSEN



Afbeelding 12: een acht op een 6x6 rooster. Op vergelijkbare wijze kunnen de getalen 1 t/m 9 geschreven worden.

De wiskunde achter de sudokuplotter werkt als volgt: We verdelen ieder vakje van de sudoku in 6x6 ‘subvakjes’ om zo een groot raster van 54 bij 54 (9 * 6) vakjes te creëren. Daarna hebben we berekend hoeveel rotaties de stappenmotor moest doen om één zo’n subvakje te passeren. Op deze manier hebben we een groot assenstelsel waarover we de pen kunnen besturen met de stappenmotoren. De verdeling in 6x6 subvakjes is bewust gekozen. Met 6x6 kan de motor namelijk goed digitale cijfers in de vakjes zetten. In afbeelding 12 is een acht weergegeven in een 6x6 raster, de dik gedrukte lijnen zijn plekken waar de pen een lijn moet trekken.

Er moet ook een pad komen voor hoe de pen zo effectief mogelijk alle 81 vakjes op de sudoku gaat passeren. Dit hebben we als volgt gerealiseerd:

>	>	>	>	>	>	>	>	Stop
^	<	<	<	<	<	<	<	<
>	>	>	>	>	>	>	>	^
^	<	<	<	<	<	<	<	<
>	>	>	>	>	>	>	>	^
^	<	<	<	<	<	<	<	<
>	>	>	>	>	>	>	>	^
^	<	<	<	<	<	<	<	<
Start	>	>	>	>	>	>	>	^

Afbeelding 13: het pad van de pen.

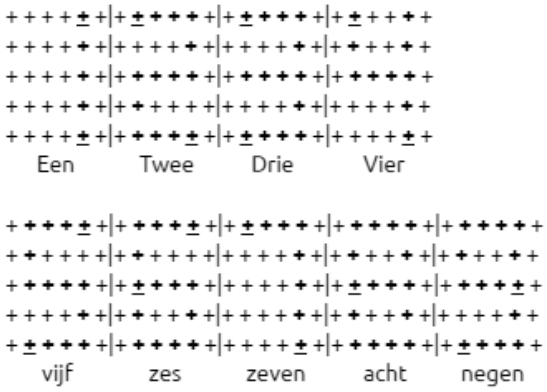
De pen begint links onderaan(vakje (1,9)) de sudoku. Hij zal eerst alle getallen van rij 9 invullen van links naar rechts en vervolgens 1 vakje omhoog gaan. Vanaf daar zal de pen de getallen van rechts naar links invullen, om daarna weer een vakje omhoog te gaan, enzovoort (zie afb. 13).

Om van vakje naar vakje te gaan rekent het programma steeds de coördinaten van het beginpunt en het eindpunt, om zo te weten hoeveel stappen in welke richting gezet moeten worden. Dit gaat met de simpele formules:

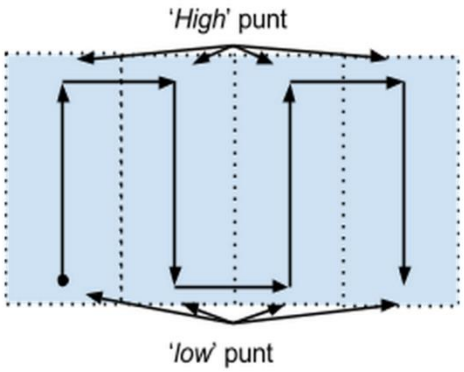
```
subvakjes verplaatsen op de Yas = Yeind punt - Y start punt

subvakjes verplaatsen op de Xas = X eind punt - X start punt
```

Naast de snelste route over de hele sudoku nemen, neemt het programma ook de snelste route tussen 2 nummers. Ieder nummer heeft namelijk een ‘high’-punt en een ‘low’-punt, die respectievelijk hoog en laag zijn. Afbeelding 14 hieronder geeft de coördinaten van alle getallen. De onderstreepte punten zijn de beginpunten. Het programma zal altijd beginnen met het schrijven van een cijfer bij één van deze twee punten. Daarna zal het programma het volgende cijfer bij high beginnen als het eindpunt high was en bij low beginnen als het eindpunt low was. Dus op de volgende manier (afb. 15):



Afbeelding 15: het schrijfpatroon van de getallen 1 t/m 9



Afbeelding 14: het schrijfpatroon de pen over 4 hokjes. Hierin geven de verticale lijnen aan wanneer een cijfer geschreven wordt, de horizontale lijnen wanneer de pen naar een nieuw hokje gaat en de rechthoeken de hokjes van een sudoku.

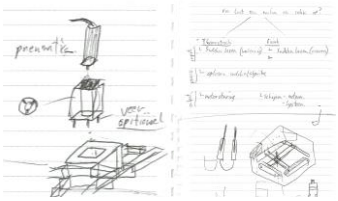
HET ONTWERP- EN BOUWPROCES VAN DE PLOTTER

Het ontwerpen en bouwen van de plotter heeft het grootste deel van onze tijd in beslag genomen. Om een plotter te bouwen die een uiterst precies en vooral consistent resultaat kan leveren moesten we zorgvuldig te werk gaan. Concentratie en overzicht houden op het gehele ontwerp waren daarom van groot belang.

DE ORIENTATIEWEKEN VOOR DE ZOMERVAKANTIE

We spraken af om de week voor de vakantie al goed gebruik te maken van de tijd die we hadden gekregen. Het afgelopen jaar hadden we al veel gebrainstormd over een goed onderwerp, wat ons nu dus de ruimte gaf om direct aan de slag te gaan. Er was tijdens het brainstormen al een conceptontwerp ontstaan (afb. 16), en we besloten hier een prototype van LEGO van te maken. Het bevatte twee rupsbanden die elk een frame kon laten bewegen. De twee frames kruisen bij een blok waarin de pen zich bevindt. De X en Y konden dus bepaald worden bij het draaien van de rupsbanden. Bij dit ontwerp was er met opzet niet naar vergelijkbare projecten op internet gekeken, om zo te zien hoe ver ons geheel eigen idee zou komen, maar ver kwam het niet. Het zou inderdaad in theorie moeten werken, maar bij het prototype kwamen er al gauw drie duidelijke problemen naar boven:

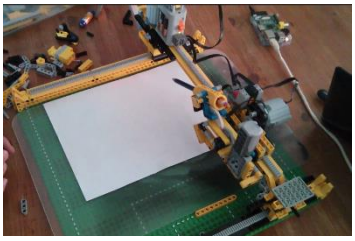
- Het eerder uitgelegde probleem van een trek-/duwkracht die niet gelijk verdeeld is over het gehele schrijfapparaat. Hierdoor verloopt de verplaatsing niet of ongelijk.
- Het probleem dat rupsbanden niet nauwkeurig genoeg zijn voor het schrijven van een getal.
- De plotter was in het algemeen vrij instabiel dankzij dunne en wankele elementen



Afbeelding 16: schetsen van LEGO prototype 1

Hoewel we dit rupsbandontwerp en deze problemen hadden kunnen oplossen en uitwerken, besloten we om een geheel nieuw ontwerp te maken, en wederom te testen met LEGO. Ditmaal kozen we ervoor om wel het internet te raadplegen. Wel spraken we af om ons enkel en alleen door foto's te laten inspireren, zodat ons project niet verder aangetast zou worden. De foto's zijn te vinden bij de bronnen aan het eind van het verslag.

Na het zien van een aantal voorbeelden hadden we een nieuw ontwerp gemaakt wat meer lijkt op het eindresultaat (zie afb. 17). Deze plotter is in het algemeen een stuk stabieler. We hadden zelfs een set op afstand bestuurbare LEGO motoren gebruikt, waardoor de plotter al enigszins automatisch is. De rails bestaat bij deze plotter uit een lange rij tanden waar een tandwiel overheen rolt. Deze techniek is een stuk nauwkeuriger dan rupsbanden, maar nog steeds niet precies genoeg voor cijfers. Om het probleem van de ongelijke verplaatsing op te lossen bevatte het nieuwe prototype een as onder de X-rails. Door deze as wordt de y-rotatie tussen twee stukken rails verdeeld, waardoor de Y-verplaatsing nu gelijk verloopt.



Afbeelding 17: LEGO prototype 2

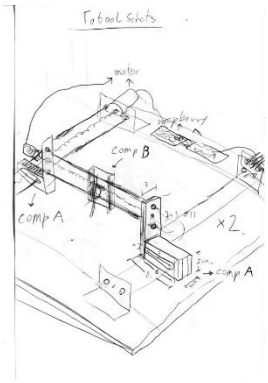
Het enige probleem wat dit ontwerp nog met zich meebracht is dat LEGO niet bedoeld is om nauwkeurig mee te werken. Er zit ruimte tussen enkele tandwielen en de motoren zijn ook ongeschikt. Het ontwerp was destijds dus bruikbaar, de LEGO plotter zelf niet.

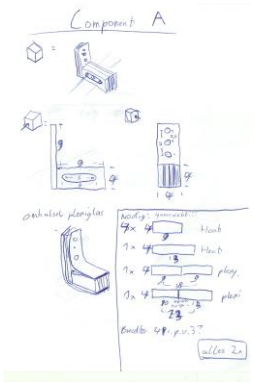
DE WERKWEEK

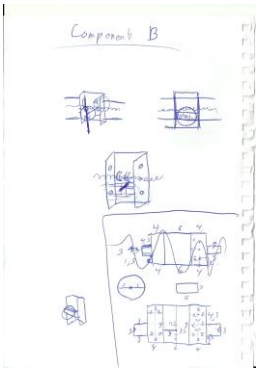
Toen de werkweek daar was hadden we nog geen definitief ontwerp. Het ontwerpen van de plotter was een lastige opgave, aangezien er met heel veel zaken tegelijk rekening gehouden moest worden. Ook wisten we nog niet welke materialen beschikbaar en vooral betaalbaar waren. We besloten dus om de Karwei een bezoekje te brengen om inspiratie op te doen voor bruikbare materialen.

We kwamen terug met twee aluminiumbuizen van 2 meter, een plaat mdf, een plaat plexiglas en twee draadeindes. Een draadeinde was ons voorlopige idee voor een schroef. Bij het kopen van de materialen hadden we al goede ideeën voor in ieder geval een aantal onderdelen van de plotter. We besloten deze onderdelen alvast te maken, en zo gaande weg de rest van plotter te realiseren.

Dinsdag was het gehele ontwerp af (zie afb 18a). Afgezien van details zoals de camera of het frame voor de sudoku zelf, is het eindresultaat goed terug te zien op deze tekeningen. Hieruit hebben we vervolgens de afmetingen bepaald en een soort bouwtekening gemaakt voor de verschillende componenten (zie afb. 18b & 18c). De rest van de week bestond uit deze onderdelen maken en oplossingen vinden voor alle problemen waar we mee te maken kregen. Er ging bijvoorbeeld bij de vele gaten die geboord moesten worden vaak wat mis wat resulteerde in een scheve rails. Ook moesten de gaten goed gevijld worden zodat de componenten soepel konden bewegen.







Afbeelding 18a: de schets van de volledige plotter.

Afbeelding 18b: de schets en benodigdheden voor module (hier nog component) A

Afbeelding 18c: de schets en benodigdheden voor module (hier nog component) B

DE HERFSTVAKANTIE

Na de werkweek was het framewerk van de plotter zo goed als af. Er moest echter nog veel gedaan worden om de plotter daadwerkelijk aan te sturen. De belangrijkste punten waren:

- De stappenmotoren moesten worden aangesloten op de plotter en wormwielen.
- Er moest nog een frame over de plotter heen voor de positie voor de camera.
- De plotter had nog een standaard locatie nodig waar de sudoku in vastgezet kon worden.
- Er moesten Wormwielen gekocht worden en in de plotter worden verwerkt.

Na het zien van de snelheid (of eerder de traagheid) van de stappenmotoren hadden we het idee van het draadeinde geschrapt. Met draadeindes zou onze plotter er uren over moeten doen om de oplossing op te schrijven. We vervingen dit idee met de Fischertechnik wormwielen die wij toen ontdekt hadden. De levering hiervan duurde echter wel een aantal weken, dus we moesten het project voorlopig stilzetten. Bovendien hadden we ook een toetsweek om voor te leren.

DE LAATSTE WEKEN

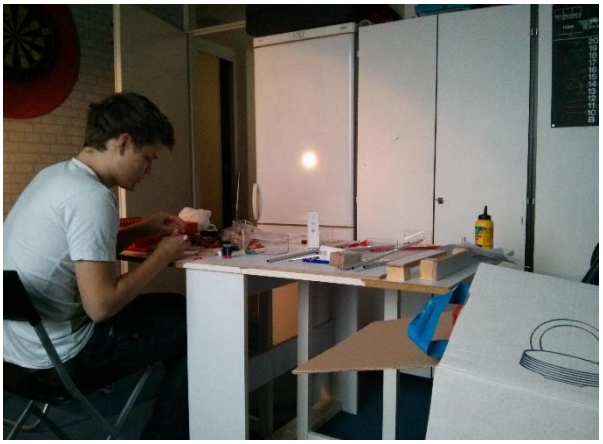
En dan nu de laatste paar weken. Deze periode duurt van de zondag na de toetsweek tot 11 december, en heeft zich voornamelijk afgespeeld bij Jos thuis. We hebben in deze weken de complete plotter afgemaakt. De punten van de vorige paragraaf zijn opgelost, en veel fouten met beide de mechanica en elektronica zijn opgelost. De assen met de wormwielen daarop zijn meerdere malen verplaatst om de optimale ligging ervan te bepalen. Ook onderdelen als de ketting en de motoren moesten vaak bijgesteld worden omdat ze niet optimaal bewogen.

Er zijn ook in dit stadium veel nieuwe onderdelen toegevoegd aan de plotter. De vele mogelijkheden van Fischertechnik hebben dit project gered, aangezien we het nooit op tijd af konden hebben zonder deze perfecte bouwstukken. De draaischijf voor de pen, de ketting tussen 3 tandwielen en de wormwielen waren hierdoor snel gerealiseerd en het eindproduct kwam eindelijk in zicht.

De laatste paar dagen zijn besteed aan de optimale positie voor de camera en sudoku bepalen, een framewerk voor deze beide elementen maken en een groot aantal testruns. Bij deze testruns kan gedacht worden aan:

- Testen hoeveel druk er op de pen moet worden uitgeoefend.
- Testen of de Y beweging gelijk verloopt (geen oneffenheden in ketting en tandwielen).
- Het maximale vierkant waarin de plotter kan schrijven bepalen.
- Testen hoeveel rotaties er nodig zijn om een sudoku vakje te overbruggen.
- Testen of de pen soepel genoeg over het papier beweegt om probleemloos te schrijven.

Bij het eindresultaat zijn er nog kleine oneffenheden aanwezig. Sommige lijnen zijn niet zo scherp als dat een plotterlijn zou moeten zijn. De grootste reden hiervoor is dat de plotter op bepaalde plekken nog vrij wankel is, wat komt door het vele vijlen. Sommige gaten voor de aluminiumbuizen zijn in het midden niet precies rond, waardoor het gehele schrijfapparaat lichtelijk op en neer kan bewegen. Voor de rest schrijft de plotter de getallen wel netjes binnen de vakken, en de cijfers zijn eenvoudig herkenbaar.



Afbeelding 19: de "kluskamer" bij Jos thuis

HET SCHRIJFPROCES VAN DE CODE

HET PROGRAMMEREN VAN HET OCR GEDEELTE

De problemen met de OCR code lag vooral in het vinden van de juiste beeldbewerkingen zodat de meeste sudoku's worden herkend. Vooral contrast en exposure speelden hierbij een rol.

Daarna was het herkennen van de cijfers een groot struikelblok. OCR is zo complex dat het veel tijd zou kosten om het herkennen zelf te schrijven. Dus moesten we bestaande software gebruiken. OCR software is een ver gevorderd concept, maar wordt vooral ontwikkeld door grote bedrijven en overheden en is dus niet toegankelijk voor de particulier. Open-source OCR software is niet in overmaat te vinden. Door de tijd en vaardigheden die nodig zijn om het te maken, zijn er weinig mensen die het maken en daarna Open-source maken.

Tesseract, gemaakt door Google en HP, is de beste optie voor open-source OCR software, omdat Tesseract het hoogste herkensucces heeft. Maar Tesseract is verre van perfect. Daarom geven we het maar één cijfer per keer om te herkennen, dit maakt het veel makkelijker dan een hele tekst. Na lang testen met de verschillende opties die Tesseract heeft, hebben we een opzet gevonden die redelijk betrouwbaar werkt.

HET PROGRAMMEREN VAN HET OPLOSSEN

Dit was de makkelijkste code om te schrijven, in de zin dat het een klassiek programmeer probleem is. Alleen wiskunde en logica, niks meer, niks minder. Dit wil echter niet zeggen dat het makkelijk is qua complexiteit. Er zijn veel versies van de code geweest voor dit programma. Iedere versie net iets beter, net iets sneller. De eerste echt werkende versie loste een moeilijke sudoku op binnen één minuut op een moderne computer. De laatste versie lost een moeilijke sudoku op binnen 0.3 seconden. Het schrijven gaf niet heel veel problemen, het was heel veel Wikipedia lezen en het voorbeeld van Wikipedia programmeren. Toen we dat werkend hadden, heeft de vader van Joren Vrancken, Jos Vrancken, uitgelegd hoe we het konden gebruiken om ook sudoku's op te lossen. En vanaf daar is het steeds verbeterd.

HET PROGRAMMEREN VAN HET SCHRIJFGEDEELTE

Zoals eerder al opgemerkt, is de code relatief eenvoudig. De code bestaat er dan ook vooral uit wanneer welke motor aangestuurd moet worden. De problemen die we tegenkwamen waren vooral hardware gerelateerd. Een groot probleem was stroomtoevoer; batterijen gingen te snel leeg en een parallel schakeling met één voeding bracht veel problemen met zich mee. Uiteindelijk leek één aparte voeding voor iedere motor te werken.

Qua code was het grootste probleem het vinden van de juiste manier om de motors aan te sturen. Er zijn drie goede Python libraries voor de Piface geschreven: Pifacecommon, Pifacedigitalio en WiringPi. De eerste twee zijn geschreven door de makers van Piface maar WiringPi is door een derde geschreven. WiringPi werkt (daarom waarschijnlijk) ook slecht. Pifacedigitalio is een versimpelede versie van Pifacecommon. Maar versimpeling brengt vermindering van functionaliteit. Dus we gebruikten Pifacecommon, één kleine maar complexe library.

TOEPASSINGEN

Een praktische toepassing voor een automaat dat een sudoku oplost is er natuurlijk niet. Ook is het invullen van de sudoku een vrij nutteloze functie aangezien het alleen kan op een sudoku van standaard formaat. Bovendien is dit proces ook uiterst traag. Het was vanaf het begin dan ook niet onze bedoeling om een apparaat te maken wat de wereld zou verbeteren. In plaats daarvan wilde we vaak toegepaste technieken, zoals een X&Y beweging wat terug te vinden is in printers, of karakterherkenning terug te vinden bij camera's die tekst moeten herkennen, zelf opnieuw uitvinden. Voor ons was dit ontwikkelingsproces belangrijker dan een duidelijk praktisch nut.

Natuurlijk kan een systeem als deze doorontwikkeld worden tot een apparaat dat wellicht wel nuttig is. Een plotter die automatisch een formulier leest, begrijpt en daarna invult is bijvoorbeeld goed mogelijk. Voor de rest zijn de individuele technieken toegepast in de automaat al veel in gebruik. Een 3d-printer is waarschijnlijk de nieuwste toepassing van camera-apparatuur combineren met een X-, Y-, Z-beweging. 3d-printers zijn nog steeds in een ontwikkelingsfase, maar nu al ontdekken mensen er vele toepassingen voor. Kunstenaars kunnen met deze techniek bijvoorbeeld complexe beelden en sieraden realiseren, maar er worden ook al huizen met een 3d printer gemaakt (zie afb. 20).

Afbeelding 20: Schaalmodel van een 3d-printer van een huis. Merk op hoe de mechanica van de X, Y-beweging lijkt op onze plotter.

CONCLUSIE EN REFLECTIE

Een concreet antwoord op de vraag ''Hoe laat je een machine automatisch een sudoku puzzel lezen, oplossen en invullen?'' kan helaas niet in één zin gegeven worden. In principe geeft het hoofdstuk "Het resultaat" het volledige antwoord op de hoofdvraag en op de deelvragen "Hoe ontvangt de machine informatie over de sudoku?" en "Hoe laat je een machine een sudoku fysiek invullen?". Verder worden de overige deelvragen uitgebreid behandeld in het hoofdstuk "verwerking: De code. "

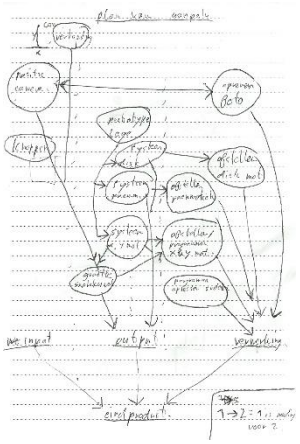
We zijn in het algemeen zeer tevreden met het resultaat. Toen we begonnen aan dit werkstuk hadden velen twijfels over de haalbaarheid van de plotter. Ook wij wisten tijdens de werkweek echt nog niet of het zou gaan lukken. Het feit dat onze plotter een volledige sudoku kan invullen met getallen die prima herkenbaar zijn is dus al een overwinning op zichzelf. De automaat heeft nog enkele mankementen die moeilijk weg te werken zijn, zoals een trillende pen, maar al met al zijn we gelukkig met wat het geworden is.

Het proces naar dit eindresultaat toe werd echter op een gegeven moment zeer lastig. De vele problemen waar we mee te maken kregen aan het eind begon ons geduld aan te tasten. Met elke fout die opgelost werd ontstonden drie nieuwe problemen. Het ene moment hadden de motoren last van haperingen, dan werden ze weer oververhit, daarna merkten we weer op dat een wormwiel scheef gemonteerd was en dan raakte de ketting weer verstrikt. We hebben dus ontzettend vaak de plotter uit elkaar gehaald, een klein detail aangepast en vervolgens weer het hele gevaarte opnieuw opgebouwd, en deze problem-solving kostten ons heel veel tijd.

Wellicht is dit ook wel de belangrijkste les die we van het bouwen van de plotter hebben geleerd: ontwerpen en bouwen is lastig, maar daadwerkelijk iets laten werken is vele malen moeilijker.

PERSOONLIJKE REFLECTIE ~ JOS

Toen we aan het profielwerkstuk begonnen had ik maar één echte eis aan het hele project: Dat we er beide trots op terug konden kijken. En dat kan ik nu bevestigen. Dit is oprecht het grootste, leukste, en meest uitdagende project van de afgelopen 6 jaar.



Afbeelding 21: "Plan van aanpak"

Dit project diende als een "meesterproef" van wat we de afgelopen 6 jaar hebben opgestoken, maar voor mij en Joren had het nog een functie: een voorproef op de jaren als student aan een technische universiteit. Bouwkunde aan de TU-Delft is hoogstwaarschijnlijk de vervolgstudie die ik ga kiezen. Hoewel de plotter niet direct aansluit op deze voorlopige keuze, weet ik zeker dat de kennis die ik heb opgedaan van het hele project me nog goed van pas zal komen.

Dit project dwong ons om erg creatief en vindingrijk na te denken. Voor talloze kleine problemen moesten we oplossingen verzinnen, en deze oplossingen ontstonden meestal door "out of the box" te denken. Dit vond ik uiteindelijk ook wel iets moois hebben. Als je op school even niet verder komt, raadpleeg je meestal een boek of een leraar. Bij dit project moesten we echter, op enkele hulpbronnen na (vaders, internet), zelf alle problemen op zien te lossen. Zelf iets (opnieuw) bedenken, ontwerpen of oplossen is iets waar "echte" intelligentie een grote rol speelt, en we hebben dan ook ontzettend veel geleerd van het gehele proces.

Toen we er halverwege achter kwamen hoe moeilijk dit project uiteindelijk zou worden bleven we gedisciplineerd doorwerken. Op een gegeven moment wilde ik het project zo graag afzien, dat ik elk moment van de dag aan de plotter zat te werken of erover na zat te denken. Een grootschalig project als deze, met personen (in dit geval 2) die ieder hun eigen kwaliteiten hebben is ook iets wat we vaker terug zullen gaan zien op de TU, en onze plotter gaf hier dus een goede preview van

De samenwerking met Joren kan perfect genoemd worden. We waren beide even gedreven aan het werk, en we vulde elkaar goed aan zonder de ander in de weg te zitten. Deze geweldige samenwerking is waarschijnlijk ook de rede waardoor de plotter daadwerkelijk gelukt is.

Waar ik het meest trots op ben van het hele project is dat het eindresultaat bijna volledig overeen komt met de schetsen die ik ver van tevoren had gemaakt. Als ik volgend jaar bouwkunde ga studeren aan de TU-Delft moet ik ook bij schetsen en ontwerpen rekening houden met veel verschillende aspecten. Bij het ontwerpen van een huis bijvoorbeeld moet je al weten of een kamer groot of klein aan gaat voelen. Dit “inlevingsvermogen” bij ontwerpen heb ik dus met dit project ook goed geoefend.

Het leukste van ons profielwerkstuk moet nog komen. Leraren, leerlingen, ouders, vrienden en kennissen: werkelijk iedereen aan wie ik het idee van een sudokuplotter aan het begin vertelde zette er grote vraagtekens achter. Op dit moment hebben vele de plotter nog niet in actie gezien, dus ik kijk uit naar alle verbaasde gezichten. Mocht jij één van deze personen zijn, wil ik je graag het volgende mededelen: “Zie je nou wel, het is ons gewoon gelukt!”

PERSOONLIJKE REFLECTIE ~ JOREN

In het Engels bestaat de term “*moonshot*”, een ogenschijnlijk onmogelijk project. Dit PWS was voor ons een moonshot. Toen we eraan begonnen wisten we dat er een redelijke kans bestond dat het niet zou lukken. Maar we wisten ook dat als we er veel tijd en moeite in zouden steken het zeker mogelijk was. Dat hebben wij gedaan en we kunnen nu trots zeggen: het is gelukt!

Het is echt van probleem naar probleem werken geweest, maar ik denk dat dit project daarom zo realistisch is, in de zin dat het veel lijkt op een project bij een universiteit of bij een bedrijf.

We moesten vaak snel een nieuwe oplossing verzinnen om nog redelijk op schema te blijven.

Dat ieder zijn eigen ding deed is een van de grootste redenen waarom dit project is gelukt. Jos deed het mechanische gedeelte en ik het programmeer gedeelte. We zaten elkaar niet in de weg, integendeel we vulde elkaar perfect aan. En het belangrijkste: geen van ons had dit project zonder de ander kunnen doen.

Voor mijzelf (qua programmeren), was het een hele uitdaging. Complexe problemen die met code opgelost moesten worden (vooral OCR) en dat drie maal. Moeilijk maar leuk en zeer uitdagend.

Ik heb zelf veel geleerd niet alleen op het gebied van dingen als beeldbewerking en programmeren, maar ook op het gebied van samenwerken en project managment.

Het is goed om zelfs je beste eigenschappen af en toe eens goed uit te dagen.

BRONNEN

Allereerst wil ik onze twee grootste bronnen van informatie bedanken: onze vaders Anno Feenstra en Jos Vrancken. Op het moment dat we ze het meest nodig hadden hebben ze ons uit de brand geholpen. Hierdoor zijn vele vele problemen in beide de ontwerp- en uitvoerfase opgelost. Spullen die ze gebruikten als vroegere student konden na jaren ook weer is uit de kast gehaald worden, en het was daarom ook leuk om dit beetje nostalgie naar boven te zien komen.

Ook de moeders hebben een grote bijdrage geleverd met het nakijken van dit verslag en proefpersoon zijn voor de wat lastigere hoofdstukken.

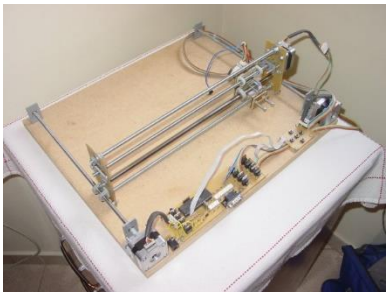
INSPIRATIE MECHANICA PLOTTER

Het ontwerp van de plotter is lichtelijk gebaseerd op de volgende drie foto's:

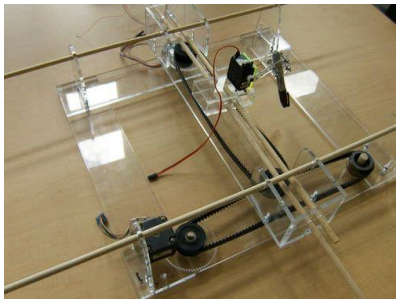
Homemade Plotter (CNC Machine): Afbeelding 22
http://myhobbyprojects.com/drupal-7.16/sites/all/themes/bluemasters/images/DSC02132.JPG

XY-Plotter: Afbeelding 23
http://www.electronics-lab.com/blog/wp-content/uploads/2011/05/XY-Plotter.jpg

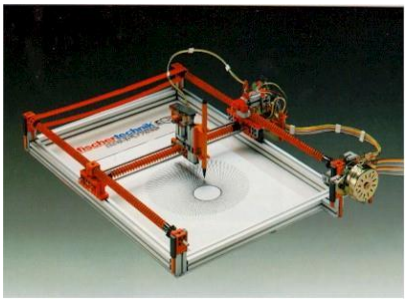
Fischer Technic BASIC Plotter (Date +/- 1985): afbeelding 24
http://www.luberth.com/plotter/fischer.jpg



Afbeelding 23:



Afbeelding 22



Afbeelding 24:

BRONNEN FOTO'S:

Fischertechnik:
http://de.wikipedia.org/wiki/Fischertechnik

3D-printer:
http://www.nutech.nl/gadgets/3677336/grote-3d-printer-kan-huis-bouwen-in-24-uur.html

De overige foto's hebben wij zelf getekend of gemaakt.

BRONNEN CODE

Bronnen - OCR:

http://www.aishack.in/2010/08/sudoku-grabber-with-opencv/

https://code.google.com/p/tesseract-ocr/

http://sudokugrab.blogspot.nl/2009/07/how-does-it-all-work.html

https://github.com/amnon/Sudoku-OCR

http://opencvpython.blogspot.nl/2012/06/sudoku-solver-part-1.html

http://stackoverflow.com/questions/10196_198/how-to-remove-convexity-defects-in-a-sudoku-square

http://stackoverflow.com/questions/9413216/simple-digit-recognition-ocr-in-opencv-python

http://www.iesensor.com/blog/2013/01/24/python-opencv-to-solve-sodoku-via-ocr/

https://code.google.com/p/python-tesseract/wiki/HowToCompilePythonTesseract

http://stackoverflow.com/questions/563198/how-do-you-detect-where-two-line-segments-intersect

Bronnen – Algoritme X:

http://en.wikipedia.org/wiki/Knuth's_Algorithm_X

Bronnen - Piface

https://projects.drogon.net

https://github.com/piface/pifacedigitalio

https://github.com/piface/pifacecommon

LOGBOEK

week voor vakantie (ong. 30 uur aan PWS gewerkt)

dag 1 ~ Jos

Joren en ik spraken af om de week voor de vakantie al goed gebruik te maken van de tijd die we hadden gekregen. Het afgelopen jaar hebben we al veel gebrainstormd over een goed onderwerp, wat ons dus nu de ruimte gaf om direct aan de slag te gaan. De 1e dag die school had weggelegd voor ons profielwerkstuk was daarom ook voldoende om alle verplichtte opdrachten af te werken. In onze taakverdeling hebben we o.a. afgesproken dat ik alles rond de mechanica regel, en dat Joren de code in orde maakt. Voor de overige 2 dagen besloten we om alvast een start te maken met ons werkstuk. Wij beide konden gelijk van start: Joren was al vrij ver met zijn programma, maar hij kon er nog veel aan verbeteren en optimaliseren. Ik aan de andere kant vond dat we een prototype van de schrijf mechaniek nodig hadden. Ik koos om dit te maken van LEGO, om vrij eenvoudig het ontwerp te kunnen veranderen.

dag 2 ~ Jos

Van mr. Scheinowitz kregen wij toestemming om thuis te mogen werken, aangezien er niets nuttigs voor ons op school is, en dat een grote bak lego op de fiets meenemen ook niet al te handig is.

LEGO prototype ~ Jos

Op school had ik enkele schetsen gemaakt van de mechanica die ik in gedachte had (zie foto).

toen ik dit in het echt gemaakt had, bleek het echter “clunky” en “sketchy” te werken. Het bewoog uiterst moeilijk en viel half uit elkaar. Ik besloot wat meer inspiratie op de toen op het internet. Daar kwam ik erachter dat de machine die ik probeer te maken een “plotter” heet, wat in principe een schrijfmachine is. Ik vond een aantal foto’s van Fischertechnik plotters, wat me goeie ideeën gaf voor een nieuw ontwerp: twee rijen tanden die ver uit elkaar staan, waar het complete apparaat verticaal op beweegt, met daartussen een rij tanden waarop de pen horizontaal beweegt (zie foto). De rest van de dag heb ik een begin gemaakt aan deze nieuwe plotter, en extra lego onderdelen besteld die ik hiervoor nodig had.

dag 3

Het nieuwe ontwerp was af. De resultaten waren vele malen beter dan het eerste ontwerp. Ik had ervoor gekozen om dit keer een as te gebruiken bij de verticale verplaatsing. Door deze as kan de rotatie van de motor worden gebruikt aan beide kanten van het apparaat, waardoor het apparaat gelijk beweegt over de track. hierdoor beweegt het uiterst soepel. Er waren nog wel 2 problemen:

1. Het mechanisme dat het potlood op en neer beweegt werkt veel te snel. Zo snel dat het potlood terug naar boven schiet zodra ik de motor niet meer laat bewegen. Dit wil ik langzamer laten bewegen m.b.v. een schroef.

2. de horizontale verplaatsing verloopt, dankzij een slechte keuze van tandwielen, uiterst langzaam. dit moet ik ook verbouwen.

Ik heb besloten deze problemen na de vakantie op te lossen

04-09-2014 (ong. 5 uur gewerkt)

LEGO prototype

bij deze heb ik de eerder genoemde fouten verbeterd, en het prototype goed werkend gekregen. De LEGO plotter heeft 3 motoren die op afstand bestuurbaar zijn. Hoewel het niet precies genoeg is om kleine letters te maken, kan het wel lange, strakke lijnen maken, waardoor grote letters mogelijk zijn. Om cijfers te kunnen schrijven, moeten we dus nog nauwkeurigere motors gebruiken.

het is mij gelukt om “hoi” te schrijven op een a4 blaadje.

Code optimalisering ~ *joren*

Ik heb vandaag de code van het OCR gedeelte van mijn programma opgeschoond. Met hulp van het programma pylint heb ik mijn code volgens de richtlijnen herschreven. Daarnaast waren er wat complicaties met de camera module van de Raspberry Pi, nader onderzoek volgt.

PWS week (ong, 35 uur gewerkt.)

22-09-2014
Het was lastig om gelijk te beginnen met ontwerpen en bouwen,. De belangrijkste reden hiervoor was dat we nog niet wisten wat geschikte materialen waren voor onze plotter. De besloten dus om eerst de Karwei te raadplegen om inspiratie op te doen voor bruikbare materialen. we kwamen terug op school aan met 2 aluminium buizen van 2 meter, 122x61 gedrukt hout plaat en 50x50 plexiglas. We zijn begonnen met bouwen. Jos heeft het nieuwe ontwerp afgemaakt. In de middag zijn we naar de Quartel gegaan om de laatste benodigdheden te kopen, bijvoorbeeld een schroefas. Deze bleek helaas erg duur te zijn waardoor we nu een voorlopig systeem maken met een draadeinde.

26-09-2014
De ontworpen onderdelen bouwen heeft de hele week in beslag genomen. Achteraf kan ik nu zeggen dat we beter thuis hadden kunnen klussen. De apparatuur op school is voldoende voor de knutselprojecten van de onderbouw, maar verschrikkelijk onhandig bij projecten zoals deze. De gevraagde precisie hebben wij uiteindelijk toch one way or another gerealiseerd. De A en B componenten zijn bij deze bijna af, De buizen zijn (ongeveer) op maat gezaagd en de vier connectiestukken voor de Y-rails zijn ook af.

DRAFT PROFIELWERKSTUK ~ JOS (ONG. 4 UUR GEWERKT)

15/10/2014
Er moet helaas nog veel gedaan worden:

- de stappenmotoren moeten worden aangesloten
- er moet een geschikt draadeinde gekocht worden en in het apparaat worden verwerkt.
- de afwijkingen van aluminiumbuizen zijn nu weggevijlt, maar ze moeten nog wel goed bevestigd worden.
- er moet nog een frame over de plotter heen voor een positie voor de camera
- de raspberry pie moet nog een vaste plek krijgen naast de plotter
- we moeten nog een code schrijven en ontwikkelen om de verkregen getallen in te vullen op de sudoku.

Vandaag zijn ook de fischertechnikonderdelen besproken en besteld. het draadeinde was een vreselijk idee en na het zien van een paar onderdelen die mijn vader nog thuis had liggen was ik volledig overtuigd. Op dit punt ben ik ook overtuigd dat we de plotter beter helemaal van Fischertechnik hadden kunnen maken, aangezien dit ons zeeën van tijd had bespaard.

LAASTE WEKEN ~ JOS

9/11/2014 - 12/8/2014 (ong. 90-110 uur gewerkt)
De weken voor de presentatie, of beter gezegd, de afgelopen maand, was/waren erg druk. Ik en Joren hebben bij mij thuis 4 tot 5 dagen per week afgesproken, en elke dag kwam de plotter weer een stapje verder. Er zijn dus veel avonduren in ons project gestoken.

9/11/2014
Vandaag heb ik alleen met mijn vader de constructie verder afgewerkt. De grondplaat is op maat gezaagd, de vier draagstukken van de Y-rails zijn op de plaat bevestigd en de schroeven zijn aan de onderkant afgewerkt met een Dremel. Verder is er met behulp van Fischertechnik een nieuw ophangsel bedacht voor de pen, en er is gebrainstormd over hoe de plotter daadwerkelijk gerealiseerd kan worden.

16/11/2014
De mechaniek heeft een enorme update gekregen deze week. De wormwielen van Fischertechnik zijn ten eerste in elkaar gezet. De mallen voor de wormwielen zijn bevestigd in de B- en beide A-componenten. De gaten hiervoor en de gaten voor de alluminiumbuizen (in totaal 12 gaten) zijn ook met een pneumatische boor en een rasp bewerkt zodat het geheel nu soepel beweegt. Ook heeft mijn vader bij de kwartel drie connectieonderdelen gehaald die het mogelijk maken om een motor vast de maken aan een wormwiel as.

23/11/2014
Week 2 is besteed aan het perfectioneren van de mechaniek. Wederom zijn er een groot aantal handelingen verricht:

- de motoren zijn bevestigd aan de A-module, de B-module en één van de vier connectiestukken.
- de motoren zijn bevestigd aan de wormwielen en de draaischijf voor de pen.
- Met behulp van Fischertechnik zijn er onderdelen gemaakt om de wormwielen op te laten rusten, en deze zijn ook bevestigd

- De buizen zijn bevestigd met behulp van ductape. Het zorgt voor een verdikking zodat de buizen in de gaten blijven steken. Hierdoor kunnen ze nog losgemaakt worden indien er iets vervangen moest worden.
- De ketting is aangebracht om de rotatie van de eerste as door te geven aan de tweede

Ook hebben we het frame voor de camera gemaakt. In een later stadium moet de ideale locatie voor de camera bepaald worden, aangezien we nog niet weten waar de sudoku precies komt te liggen.

De elektronica is in deze week ook toegevoegd aan de plotter. De motoren konden nu al aangestuurd worden door de Raspberry Pi.

30/11/2014

In de derde week stond fouten oplossen centraal. Het grootste probleem had te maken met de manier hoe de wormwielen waren bevestigd aan de motoren. De as die door de wormwielen heen gaat is namelijk te klein voor de afstand tussen de twee connectiestukken. We moesten dus “verlengstukjes” gebruiken, zes stuks in totaal, om de assen vast te maken aan andere assen, en hier ontstonden problemen. De connectiestukjes waren namelijk niet strak genoeg, waardoor de assen vaak losschoten of de rotatie niet goed over gaven aan de volgende as. Probleem is uiteindelijk opgelost dor een grote aluminium as te kopen met de zelfde diameter als de assen gebruikt bij de Fischertechnik, en deze op maat de snijden met een drempel.

Ook was de ketting niet strak genoeg. Dit is opgelost met een klein tandwiel dat zo bevestigd is dat de ketting een soort driehoek vormt. Ook zijn sommige Fischertechnik onderdelen vervangen door simpelere stukken, aangezien sommige onderdelen overbodig waren.

De locatie voor de sudoku is ook bepaald en daarmee konden we nu de camera bevestigen. We ontdekte dat belichting erg belangrijk is bij deze stap.

7/12/2014

En dan nu de laatste week. Hoewel de plotter nu in theorie zou moeten werken, ontdekte we veel problemen, waaronder hapering bij de motoren. Het bleek te gaan om de stroomtoevoer. De motoren halen energie uit een 9 volt batterij, voor elke motor één. Aan het begin dachten we dat het simpelweg ging om lege batterijen, en dit was gedeeltelijk ook waar. Het was pas later in deze week dat we ontdekte dat de motoren in staat zijn een 9 volt batterij in leeg te trekken in enkele minuten. Een sudoku oplossen zou dan dus 3 9v batterijen kosten per oplossing, dus dat zou het een dure hobby maken. Er moest dus een alternatief komen.

Gelukkig had de vader van Joren een grote lading aan adapters paraat liggen die wij mochten gebruiken. Allereerst had Joren de motoren parallel aangesloten op één adapter, maar dit werd de plotter bijna fataal. De motoren werden zo heet dat we ze niet meer aan konden raken, en in paniek werd elk draadje van de plotter losgetrokken. Gelukkig heeft dit geen ernstige schade achtergelaten aan de plotter. We probeerde het opnieuw met een adapter van 5 volt, maar ook dit zorgde voor oververhitting. Ook merkte we op dat de plotter rare verschijnselen vertoonde, zoals led lampjes die bleven branden op momenten dat dat niet moest.

Uiteindelijk hebben we dit probleem, met hulp van Jorens vader, opgelost met drie 5 volt adapters, één voor elke motor. Door de motoren parallel aan te sluiten ontstonden er ongewenste stroomkringen die de motoren uiteindelijk teveel energie gaven, en bij dezen was dit probleem opgelost.

Verder hebben we het schrijven van de plotter gekalibreerd en een soort fotolijstje voor de sudoku gemaakt. Joren had de code voor het schijven van de pen vrij snel afgemaakt, en het werkte bijna meteen.

12/8/2014

Na een laatste paar foutjes in Jorens code opgelost te hebben hadden wij nu de plotter eindelijk af. Ruw geschat hebben we ±177 uur aan ons profielwerkstuk gewerkt. Daar zijn de uren besteed aan het schrijven van dit verslag niet bij opgeteld.

Woordenaantal zonder Apprendix: 10430.

APPENDIX

BRON CODE: BESTAND: COMB.PY

```
## import all the neccessary libraries, os and the custom ones
import os
import algx
import finder
import motor_control

## Take the photo using raspistill.
print "Taking photo"
os.system("sudo raspistill -roi .4,0.05,1,1 -cfx 128:128 -w 600 -h 450 -q 100 -ex
night -e jpg -sh 100 -co 100 -o image.jpg")


## Find the sudoku in the image using OCR.
sudoku = finder.OCR("image.jpg")

## Print out the found sudoku.
for row in sudoku:
    string = ""
    for digit in row:
        string += str(digit) + " "
    print string

## Do the necessary preprocessing before solving the sudoku.
Set, keys, values = algx.sudo2set(sudoku)
matrix = {}

for i in xrange(len(keys)):
    key = keys[i]
    matrix[key]=[]
    for value in values:
        if value in Set[key]:
            matrix[key].append(1)
        else:
            matrix[key].append(0)

## Solve the sudoku.
print "solving"
solutions = algx.exactcover(matrix)

## Print out the solved sudoku.
sudostring = ""
for i in sorted(solutions[0]):
    sudostring += str(i[2]) + " "

solved_sudoku = [sudostring[i:i+18] for i in xrange(0, 81*2, 18)]
for i in solved_sudoku:
    print i
```

```
### Select first solution and convert it into needed steps

solution = finder.split_len( sorted( solutions[0] )[::-1], 9)

## Filter the points that need to be filled and sort them the right way.

steps = []
for row in range(9):
    if row%2 == 0:
        for digit in solution[row][::-1]:
            row = digit[0]
            column = digit[1]
            if sudoku[row][column] == "x":
                steps.append( digit)

    else:
        for digit in solution[row]:
            row = digit[0]
            column = digit[1]
            if sudoku[row][column] == "x":
                steps.append(digit)

## 1 cell takes 1850 rotations of 1 motor, so 1/6 of 1 cell = 308.33
## To correct small defects, we rounded 308.33 down to 300
x = 300

## calculate and set step to the first digit start
print motor_control.steps_calc( None, steps[0])[0]
motor_control.set_step( motor_control.steps_calc( None, steps[0])[0], x)

## Move the pen down so it hits the paper.
motor_control.pen_down(90)

for step_i in xrange( len( steps ) ):
## Print out the number and the cell the solver is going to write.
    print steps[step_i]

## Write the number
    motor_control.write_number( steps[step_i], x )

## If the number is not the 81st number the program needs to go to the next number.
## The program calculates the path it needs to take.
    if step_i < len(steps) - 1 :
        print motor_control.steps_calc( steps[step_i] , steps[ step_i+1] )
        motor_control.set_step( motor_control.steps_calc( steps[step_i] ,
steps[ step_i+1] ), x )

## Move the pen up because the sudoku is solved.
motor_control.pen_up(90)
```

BESTAND: ALGX.PY

```
from time import time
def block(cell):
    """Calculate the block coordinates of a given cell, and format those into a block
contains X tuple."""
    value=[0,0,cell[2]]
    for i in xrange(2):
        if cell[i] < 3:
            value[i] = 1
        if cell[i] >= 3 and cell[i] < 6:
            value[i] = 2
        if cell[i] >= 6:
            value[i] = 3
    return ("block",value[0],value[1],value[2])

def dictcopy(dic):
    """Deepcopy a dict object, with lists as values."""
    keys = list(dic.keys())
    values = [list(i) for i in dic.values()]
    return dict(zip(keys,values))

def sudo2set(sudoku):
    """Create a dict object with as keys (row,column,allpossible values) an as values
a list
    containing the following: row contains value, column contains value, block
contains value
    and cell is filled. The function also returns a list of all unique keys and
values"""

    Set = {}
    keys = []
    values = []
    for row in xrange(9):
        for column in xrange(9):
            value = sudoku[row][column]
            if value == "x":
                for newvalue in xrange(1,10):
                    key = (row,column,newvalue)
                    value = [("row", row, newvalue),
                             ("column", column, newvalue),
                             block(key),
                             ("cell", row, column)]
                    Set[key] = value
                    keys.append(key)
                    values += value

            else:
                key = (row, column, value)
                value = [("row", row, value),
                         ("column", column, value),
                         block(key),
```

```
                        ("cell", row, column)]
                Set[key] = value
                keys.append(key)
                values += value
    return Set, keys, set(values)

def printsolution(solution,solutioncount):
    """Print out a solution."""
    print "Solution: " + str(solutioncount)
    print "-"*25

    sudostring = ""
    for i in sorted(solution):
        sudostring += str(i[2]) + " "

    solved_sudoku = [sudostring[i:i+18] for i in xrange(0, 81*2, 18)]
    for i in solved_sudoku:
        print i

    print "-"*25

def exactcover(matrix):
##Setting variables
##sets containts [ [ rows, main matrix, partial solution ] ]
    all_solutions = []
    sets = [[[],matrix,[]]]
    tempsets = []
    solved = False
    while solved == False:
##If there are no more sets, the exact cover has been found, thus all the solutions
have been found.
        if sets == []:
            solved = True

        for set in sets:
            matrix = set[1]
##Step 1) If the matrix A is empty, the problem is solved; terminate successfully.
            if matrix == {}:
                solution = set[2]
                all_solutions.append(solution)
                continue
##Step 2) Otherwise choose a column c (deterministically)
##Count the 1's in every column.
            columncount = [i.count(1) for i in zip(*matrix.values())]

##Calculate the smallest number of 1's in any column.
            minimum = min(columncount)

##If the minimum is 0, there is no exact cover possible, thus terminate
unsuccessfully.
            if minimum == 0:
                continue
```

```
##Take (one of) the columns with the fewest 1's.
            first_min_column_index = columncount.index(minimum)

##Step 3) Choose a row r such that Ar,c = 1 (detministically)
            rows = []
            for key in matrix:
                if matrix[key][first_min_column_index] == 1:
                    rows.append(key)
##Step 5) For each column j such that Ar,j = 1,
##              for each row i such that Ai,j = 1,
##                      delete row i from matrix A;
##              delete column j from matrix A.
            for row in rows:

##The matrix in a lot of sets is the same, so first (custom) deepcopy the matrix.
                matrix = dictcopy(set[1])

##For every column in row with the value 1, append to deletecolumns.
                deletecolumns = []
                for i in xrange(len(matrix[row])):
                    if matrix[row][i] == 1:
                        deletecolumns.append(i)
##For every row with the value 1 in a column in deletecolumn, append to deleterows.
                deleterows = []
                for key in matrix:
                    for column in deletecolumns:
                        if matrix[key][column] == 1:
                            deleterows.append(key)
                            break
##Remove the deleterows.
                for deleterow in deleterows:
                    del matrix[deleterow]

##Remove the deletecolumns from the remaining rows. The list is reversed so that the
indexes don't change.
                for column in deletecolumns[::-1]:
                    for key in matrix:
                        del matrix[key][column]

##Step 4) Include r in the partial solution.
##Put the potential sets in a temporary list.
                tempsets.append([[],matrix,set[2]+[row]])
##Remove the empty and unsuccessful sets, by only putting the potential sets into the
new list.
        sets = [i for i in tempsets]
        tempsets = []
    return all_solutions
```

BESTAND: FINDER.PY

```
"""Finds sudoku puzzles in images."""
import time
import cv2
import numpy as np
import sys
import tesseract
API = tesseract.TessBaseAPI()

def ocr_singledigit(image):
    """Recognize a singe digit on a small image"""
    API.Init(".", "eng", tesseract.OEM_DEFAULT)
    API.SetVariable("tessedit_char_whitelist", "123456789")
    API.SetPageSegMode(6)
    tesseract.SetCvImage(image, API)
    CHAR = API.GetUTF8Text()
    CHAR = CHAR.replace(" ", "").strip()

    if len(CHAR) == 0:
        return "x"
    return int(CHAR)

def split_len(item, length):
    """Split a item into a list, split on length"""
    return [item[i:i+length] for i in range(0, len(item), length)]

def getcorners(C, points):
    """Calculate the corner points from point coordinates"""
    x = C[0]
    y = C[1]
    top_left_corner_index = 10 * y + x
    down_right_corner_index = top_left_corner_index + 11

    top_left_corner = points[top_left_corner_index]
    down_right_corner = points[down_right_corner_index]

    return (top_left_corner[0], down_right_corner[0], top_left_corner[1],
down_right_corner[1])

def OCR(IMAGE_FILE):
    """Recogonize a sudoku in a photo."""

## Debug mode init
    DEBUG_MODE = False
    if DEBUG_MODE:
        import os
        DEBUG_FOLDER = "debug/"
        if not os.path.exists(DEBUG_FOLDER):
            os.mkdir(DEBUG_FOLDER)

    print "[+]Using file: " + IMAGE_FILE
```

```
## Image PreProcessing
    print "[1]Image preprocessing"

## Read the image, apply gaussianblur and grayscale.
    IMG = cv2.imread(IMAGE_FILE)
    IMG = cv2.GaussianBlur(IMG, (5, 5), 0)
    GRAY = cv2.cvtColor(IMG, cv2.COLOR_BGR2GRAY)
    MASK = np.zeros((GRAY.shape), np.uint8)

## Create a mask for horizontal and vertical line recognizing.
    MASK_HOR = np.zeros((GRAY.shape), np.uint8)
    MASK_VER = np.zeros((GRAY.shape), np.uint8)

## Creating some copies and ellipse.
    KERNEL1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))

    CLOSE = cv2.morphologyEx(GRAY, cv2.MORPH_CLOSE, KERNEL1)
    DIV = np.float32(GRAY)/(CLOSE)
    RES = np.uint8(cv2.normalize(DIV, DIV, 0, 255, cv2.NORM_MINMAX))
    RES2 = cv2.adaptiveThreshold(RES, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, 0, 151,
70)

## Finding Sudoku Square

    print "[2]Finding Square"

## Applying the threshold.
    THRESH = cv2.adaptiveThreshold(RES, 255, 0, 1, 19, 2)

    if DEBUG_MODE:
        cv2.imwrite(DEBUG_FOLDER + "thresh.jpg", THRESH)

## Finding contours and selecting the largest one.
    CONTOUR, HIER = cv2.findContours(THRESH, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

    MAX_AREA = 0
    for CNT in CONTOUR:
        AREA = cv2.contourArea(CNT)
        if AREA > 1000 and AREA > MAX_AREA:
            MAX_AREA = AREA
            BEST_CNT = CNT

## Drawing the contours.
    cv2.drawContours(MASK, [BEST_CNT], 0, 255, -1)
    cv2.drawContours(MASK, [BEST_CNT], 0, 0, 2)

    RES = cv2.bitwise_and(RES, MASK)
    if DEBUG_MODE:
        cv2.imwrite(DEBUG_FOLDER + "puzzle.jpg", RES)

## Finding Vertical lines.
    print "[3]Finding V lines"
```

```
## Creating a rectangle.
    KERNELX = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 10))

## Sobel is a edge detection algorithm. We detect the edges and then optimize the
output.
    DX = cv2.Sobel(RES, cv2.CV_64F, 1, 0)
    DX = cv2.convertScaleAbs(DX)
    cv2.normalize(DX, DX, 0, 255, cv2.NORM_MINMAX)
    RET, CLOSEX = cv2.threshold(DX, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    CLOSEX = cv2.morphologyEx(CLOSEX, cv2.MORPH_DILATE, KERNELX)

## Find and draw the contours with the right measurments.
    CONTOUR, HIER = cv2.findContours(CLOSEX, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for CNT in CONTOUR:
        x, y, w, h = cv2.boundingRect(CNT)
        if h/w > 5:
            cv2.drawContours(CLOSEX, [CNT], 0, 255, -1)
        else:
            cv2.drawContours(CLOSEX, [CNT], 0, 0, -1)

    CLOSEX = cv2.morphologyEx(CLOSEX, cv2.MORPH_DILATE, None, iterations = 2)

    if DEBUG_MODE:
        cv2.imwrite(DEBUG_FOLDER + "vlines.jpg", CLOSEX)

## Finding Horizontal Lines
    print("[4]Finding H Lines")

## Creating a rectangle.
    KERNELY = cv2.getStructuringElement(cv2.MORPH_RECT, (10, 2))

## Sobel is a edge detection algorithm. We detect the edges and then optimize the
output.
    DY = cv2.Sobel(RES, cv2.CV_64F, 0, 1)
    DY = cv2.convertScaleAbs(DY)
    cv2.normalize(DY, DY, 0, 255, cv2.NORM_MINMAX)
    RET, CLOSEY = cv2.threshold(DY, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    CLOSEY = cv2.morphologyEx(CLOSEY, cv2.MORPH_DILATE, KERNELY)

## Find and draw the contours with the right measurments.
    CONTOUR, HIER = cv2.findContours(CLOSEY, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    for CNT in CONTOUR:
        x, y, w, h = cv2.boundingRect(CNT)
        if w/h > 5:
            cv2.drawContours(CLOSEY, [CNT], 0, 255, -1)
        else:
            cv2.drawContours(CLOSEY, [CNT], 0, 0, -1)

    CLOSEY = cv2.morphologyEx(CLOSEY, cv2.MORPH_DILATE, None, iterations = 2)
```

```
    if DEBUG_MODE:
        cv2.imwrite(DEBUG_FOLDER + "hlines.jpg", CLOSEY)

## Finding Grid POINTS
    print("[5]Finding POINTS")
    RES = cv2.bitwise_and(CLOSEX, CLOSEY)

    if DEBUG_MODE:
        cv2.imwrite(DEBUG_FOLDER + "POINTS.jpg", RES)

## Correcting the defects
    print("[6]Correcting defects")
    CONTOUR, HIER = cv2.findContours(RES, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    POINTS = []
    for CNT in CONTOUR:
        mom = cv2.moments(CNT)
        (x, y) = int(mom['m10']/mom['m00']), int(mom['m01']/mom['m00'])
        POINTS.append((x, y))

    if len(POINTS) != 100:
        print "[!]Centroids: " + str(len(POINTS))
        print "[!]Exiting"
        sys.exit()

## Sort the points.
    C_POINTS = [sorted(i, key = lambda x: x[1]) for i in split_len(sorted(POINTS),
10)]

    R_POINTS = [ list(i) for i in zip(*C_POINTS)]
    R_POINTS = [x for sublist in R_POINTS for x in sublist]

##OCR Stage
    print "[7]OCR Stage: "
    RESULT = [[0]*9 for i in range(9)]

## Loop through the

    for row in range(9):
        print "\t[+]Row: " + str(row + 1)
        for column in range(9):

## Calculate the cornerpoints of the cell
## Top left corner = x1:y1 | down right corner = x2:y2
            x1, x2, y1, y2 = getcorners((row, column), R_POINTS)

## Create a new image with a single digit in it.
            crop = RES2[y1 + 7 : y2 - 7 , x1 + 7: x2 - 7]

            digit = cv2.cv.CreateImageHeader((crop.shape[1], crop.shape[0]),
cv2.cv.IPL_DEPTH_8U, 1)
            cv2.cv.SetData(digit, crop.tostring(), crop.dtype.itemsize*crop.shape[1])
```

```
            if DEBUG_MODE:
                print row, column
                cv2.imwrite(DEBUG_FOLDER + str(row) + str(column)+".jpg", crop)

## Recognize the digits
            RESULT[column][row] = ocr_singledigit(digit)

    return RESULT

## The test section, if the finder is execute on it's own.
if __name__ == "__main__":
    TIMER_START = time.time()
    RESULT = OCR(sys.argv[1])

## Print out the sudoku puzzle
    print
    print "[+]Solution:"
    for row in RESULT:
        string = ""
        for digit in row:
            string += str(digit) + " "
        print string

    print
    print "Total time: " + str(round(time.time() - TIMER_START, 2)) + " seconds"
```

BESTAND: MOTOR_CONTROL.PY

```
import pifacecommon.mcp23s17
import time

## Init the mcp23s17 chip on the Piface.
mcp = pifacecommon.mcp23s17.MCP23S17()

mcp.iocon.value = (
            pifacecommon.mcp23s17.BANK_OFF |
            pifacecommon.mcp23s17.INT_MIRROR_OFF |
            pifacecommon.mcp23s17.SEQOP_OFF |
            pifacecommon.mcp23s17.DISSLW_OFF |
            pifacecommon.mcp23s17.HAEN_ON |
            pifacecommon.mcp23s17.ODR_OFF |
            pifacecommon.mcp23s17.INTPOL_LOW
        )

mcp.gpioa.value = 0x00
mcp.iodira.value = 0x00

## This is where the magic happens, 0xAA means that the inputs 0, 2, 4 and 6 are
outputs.
```

```
## Because 0xAA means that only pins 1 3 5 7 are inputs.
mcp.iodirb.value = 0xAA

### Main class for motors.
class motor():
    """Motor class that is used for init and rotation of the motors."""
    def __init__(self, pin_range, GPIOX, mcp):
        self.rotate_pins = [pifacecommon.mcp23s17.MCP23S17RegisterBit(i, GPIOX, mcp)
for i in pin_range]

    def clockwise_rotate(self, rotations):
        """Rotate the motor clockwise."""
        wait = 3/1000
        for i in range(int(rotations)):

            self.rotate_pins[3].set_high()
            time.sleep(wait)
            self.rotate_pins[0].set_low()
            time.sleep(wait)
            self.rotate_pins[2].set_high()
            time.sleep(wait)
            self.rotate_pins[3].set_low()
            time.sleep(wait)
            self.rotate_pins[1].set_high()
            time.sleep(wait)
            self.rotate_pins[2].set_low()
            time.sleep(wait)
            self.rotate_pins[0].set_high()
            time.sleep(wait)
            self.rotate_pins[1].set_low()
            time.sleep(wait)
        self.rotate_pins[0].set_low()

    def counter_clockwise_rotate(self, rotations):
        """Rotate the motor counter clockwise."""
        wait = 3/1000
        for i in range(int(rotations)):

            self.rotate_pins[0].set_high()
            time.sleep(wait)
            self.rotate_pins[3].set_low()
            time.sleep(wait)
            self.rotate_pins[1].set_high()
            time.sleep(wait)
            self.rotate_pins[0].set_low()
            time.sleep(wait)
            self.rotate_pins[2].set_high()
            time.sleep(wait)
            self.rotate_pins[1].set_low()
            time.sleep(wait)
            self.rotate_pins[3].set_high()
            time.sleep(wait)
            self.rotate_pins[2].set_low()
```

```
        self.rotate_pins[3].set_low()


def steps_calc(point_A, point_B):
    "Calculate the path between point A and point B."
    ### Each digit gets 2 points, 1 high and 1 low. This way the program can write
from low to high to low.
    ### This is more efficient.
    ### If the (column) is even it means the writing ended on the high point.
    sPoint,ePoint = None, None
    if point_A:
        ## Calc start point
        yA,xA,nA = point_A
        sPoint = [ xA * 6, (8 - yA) * 6 ]

        ## High point
        if xA % 2 == 0:
            if nA in (1,5,6):
                sPoint[0] += 4
                sPoint[1] += 5

            elif nA in (2,3,4,7):
                sPoint[0] += 2
                sPoint[1] += 5

            elif nA == 8:
                sPoint[0] += 2
                sPoint[1] += 3

            else: ## 9
                sPoint[0] += 4
                sPoint[1] += 3

        ## Low Point
        else:
            if nA in (1,2,4,7):
                sPoint[0] += 4
                sPoint[1] += 1

            elif nA in (3,5,9):
                sPoint[0] += 2
                sPoint[1] += 1

            elif nA in (6,8):
                sPoint[0] += 2
                sPoint[1] += 3


## Calc end point
    if point_B:
        yB,xB,nB = point_B
        ePoint = [ xB * 6, (8 - yB) * 6 ]
```

```
        ## Low point
        if xB % 2 == 0:
            if nB in (1,2,4,7):
                ePoint[0] += 4
                ePoint[1] += 1

            elif nB in (3,5,9):
                ePoint[0] += 2
                ePoint[1] += 1

            elif nB in (6,8):
                ePoint[0] += 2
                ePoint[1] += 3

        ## High Point
        else:
            if nB in (1,5,6):
                ePoint[0] += 4
                ePoint[1] += 5

            elif nB in (2,3,4,7):
                ePoint[0] += 2
                ePoint[1] += 5

            elif nB == 8:
                ePoint[0] += 2
                ePoint[1] += 3

            else: #9
                ePoint[0] += 4
                ePoint[1] += 3

    if point_A and point_B:
        return (ePoint[0] - sPoint[0], ePoint[1] - sPoint[1])

    else:
        return (ePoint,sPoint)



def write_number(point, x):
    """Write a number."""

    y_co, x_co, n = point
    pen_down(30)

## start point is low
    if x_co % 2 == 0:
        if n == 1:
            up(4 * x)

        if n == 2:
            left(2 * x)
```

```
    up(2 * x)
    right(2 * x)
    up(2 * x)
    left(2 * x)

if n == 3:
    right(2 * x)
    up(2 * x)
    left(2 * x)
    right(2 * x)
    up(2 * x)
    left(2 * x)

if n == 4:
    up(4 * x)
    down(2 * x)
    left(2 * x)
    up(2 * x)

if n == 5:
    right(2 * x)
    up(2 * x)
    left(2 * x)
    up(2 * x)
    right(2 * x)

if n == 6:
    down(2 * x)
    right(2 * x)
    up(2 * x)
    left(2 * x)
    up(2 * x)
    right(2 * x)

if n == 7:
    up(4 * x)
    left(2 * x)

if n == 8:
    down(2 * x)
    right(2 * x)
    up(2 * x)
    left(2 * x)
    up(2 * x)
    right(2 * x)
    down(2 * x)
  left(2 * x)

if n == 9:
    right(2 * x)
    up(2 * x)
    left(2 * x)
    up(2 * x)
```

```
            right(2 * x)
            down(2 * x)

## start point is high
    else:
        if n == 1:
            down(4 * x)

        if n == 2:
            right(2 * x)
            down(2 * x)
            left(2 * x)
            down(2 * x)
            right(2 * x)

        if n == 3:
            right(2 * x)
            down(2 * x)
            left(2 * x)
            right(2 * x)
            down(2 * x)
            left(2 * x)

        if n == 4:
            down(2 * x)
            right(2 * x)
            up(2 * x)
            down(4 * x)

        if n == 5:
            left(2 * x)
            down(2 * x)
            right(2 * x)
            down(2 * x)
            left(2 * x)

        if n == 6:
            left(2 * x)
            down(2 * x)
            right(2 * x)
            down(2 * x)
            left(2 * x)
            up(2 * x)

        if n == 7:
            right(2 * x)
            down(4 * x)

        if n == 8:
            down(2 * x)
            right(2 * x)
            up(2 * x)
            left(2 * x)
```

```
            up(2 * x)
            right(2 * x)
            down(2 * x)
            left(2 * x)

        if n == 9:
            up(2 * x)
            left(2 * x)
            down(2 * x)
            right(2 * x)
            down(2 * x)
            left(2 * x)
    pen_up(30)

def set_step(translation, x):
    "Do steps in a certain direction."
    xTr, yTr = translation[0], translation[1]

    if xTr < 0:
        left( abs(xTr) * x)
    if xTr > 0:
        right(xTr * x)

    if yTr < 0:
        down(abs(yTr) * x)
    if yTr > 0:
        up(yTr * x)

## Here we init the motors using the Piface input and output pins.
## Output pins 0 - 3
motor1 = motor(range(0,4), pifacecommon.mcp23s17.GPIOA, mcp)

## Output pins 4 - 7
motor2 = motor(range(4,8), pifacecommon.mcp23s17.GPIOA, mcp)

## Input pins 0, 2, 4, 6
motor3 = motor(range(0,8,2), pifacecommon.mcp23s17.GPIOB, mcp)

## To make it a bit easier for us, we put the rotation direction in a variable
corresponding.
down = motor1.counter_clockwise_rotate
up = motor1.clockwise_rotate

right = motor2.clockwise_rotate
left = motor2.counter_clockwise_rotate

pen_down = motor3.clockwise_rotate
pen_up = motor3.counter_clockwise_rotate
```