

TEGNEROBOTTEN

Udvikling af en 3-akset tegnerobot,
som kan gengive et digitalt
billede med blyant på papir

Projektgruppe 8

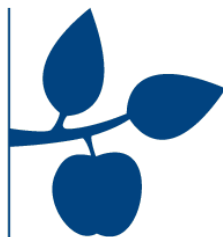
August Büchert - aubry20

Emil Dagelykke - emdag19

Nikolas Kristensen - nikkr20

Thomas Therkelsen - ththe20

Victoria Jørgensen - vijoe20



SYDDANSK | UNIVERSITET

Diplomingeniør Robotteknologi

TEK MMMI

Syddansk Universitet

11/12/2020

Abstract

First, this report will introduce the theory fundamental to creating and understanding a 3-axis drawing robot. In addition, it shall be presented how one might program a PLC to draw pictures that are processed and instructionalized from a Java program through a socket using TCP/IP. Furthermore, grayscaleing will be unraveled as the main course of action to convert an image into coordinates, which can then be adapted to be interpreted by the PLC to compose breathtaking drawings, utilizing pencil and paper.

Thereafter, the programming language of Java, along with the basic knowledge surrounding it, shall be explained, and accordingly the details about how Java stands imperative to the attainment of the projects purpose will be distinguished. Furthermore, the analysis of the concept TCP/IP implying the communication between different individual systems capable of connecting to the internet will thus including a description of the approach in setting up a socket. Moreover, in depth explanations on PLC and the term structured text will be provided, preeminently how they were of vital importance to enhance the final solution to this project.

Finally, the report further contains a chapter unveiling logic gates regarding basic theory, utilization of a statemachine and its involvement in a functional drawing robot. Possible improvements as well as thoughts on choices made throughout the process of effectuating a solution may be shared, debunked and evaluated. The entire project shall certainly culminate in a summarized conclusion on the coalescence of Java, PLC and TCP/IP, the three main segments embodying the objective of this project, moreover a final conclusion on the project course itself and viability hereof.

Forord

Heraf fremgår det i prioriteret rækkefølge hvad det enkelte gruppemedlem primært har ydet og bidraget med til projektet, i løbet af projektførløbet.

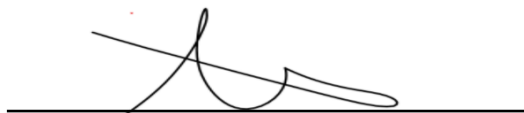
August og Thomas har primært beskæftiget sig med, at udvikle Java & PLC samt at skrive rapport. Emil og Nikolas har opsat & klargjort PLC samt skrevet rapport. - Mens Victoria har beskæftiget sig med, at udvikle Java & opsætning- og skrivning af rapport.

Indhold i Zip-filen **Gruppe8.zip**:

Java - Som indeholder Java koden og Assets

PLC - Som indeholder PLC koden

August Büchert



Emil Dagelykke



Thomas Therkelsen



Nikolas Kristensen



Victoria Jørgensen



Indholdsfortegnelse

1	Indledning	5
2	Teori	6
2.1	Hardware	6
2.2	Java [1]	8
2.2.1	Indlæsning af digitalt billede	8
2.2.2	Pixelanalyse	9
2.2.3	Grayscale	9
2.2.4	Generering af PLC protocol-streng:	10
2.3	TCP/IP & Socket	11
2.4	Logiske kredsløb	12
2.5	PLC	14
2.5.1	Struktureret tekst [3]	15
2.5.2	Input/output	15
2.5.3	Koordinatbehandling	16
2.5.4	Statemachine [5]	17
3	System test	18
3.1	Hardware tests	19
3.2	Cycle time test	20
3.3	Software tests	20
4	Diskussion	20
4.1	Billede kvalitet	21
4.2	Tegne i begge retninger	21

4.3	Blyantspidser	22
4.4	Langsom testning	22
5	Konklusion	23
6	Appendix	24
	Referenceliste	27

1 Indledning

Den fjerde industrielle revolution, som mange i vor tid vælger at betegne det, er under konstant udvikling, og det observeres hvordan flere af industriens, såvel som hverdagens processer, automatiseres.

En af de mange fordele ved automation er naturligvis, at de fejl, som mennesket foretager ved den manuelle proces, elimineres. Med dette i mente, designes og konstrueres robotter til alverdens forskellige formål i dag. Selv en opgave så simpel, som det, at tegne en papirstegning, kan med fordel automatiseres, for at undgå de væsentligste fejl og dermed skabe en præcis gengivelse af et givent motiv.

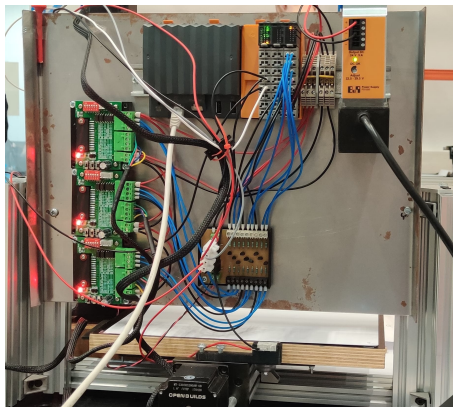
Netop det, er hvad følgende rapport beskæftiger sig med: At programmere en 3-akset tegnerobot, som kan scanne et vilkårligt digitalt billede, analysere hver pixel, for derefter at gengive billedet med blyant på papir. I første del af rapporten beskrives først og fremmest robottens fysiske elementer, og derudover softwarens 3 delelementer - Java-, socket- og PLC-programmerne - der er relevante at forstå i forbindelse med robottens funktion.

Herefter følger en systemtest, hvor tegnerobotten afprøves og til sidst en sammenfattende konklusion og diskussion, hvor der opsamles på rapportens væsentligste pointer og resultater, fordele/ulemper, og forslag til hvordan enkelte elementer eller problemer kunne optimeres.

2 Teori

2.1 Hardware

På figur 1 ses bagsiden af robotten, hvis mekaniske og elektriske elementer på forhånd var samlet og monteret til brug i projektet. Den fysiske del af robotten består først og fremmest af en PLC lavet af B&R, med modelnummer X20CP1382.



Figur 1: Bagsiden af robotten

PLC'en består af [4]:

- Intel x86 400MHz-kompatibel CPU, m/ integreret I/O processor
- Onboard Ethernet, POWERLINK m/ poll response og USB
- 1 slot til modulær interface udvidelse (I/O slots)
- 30 digitale I/O og to integrerede analog inputs
- 2GB flash drev onboard
- 256MB DDR3 SDRAM
- Batteridrevet real-time ur

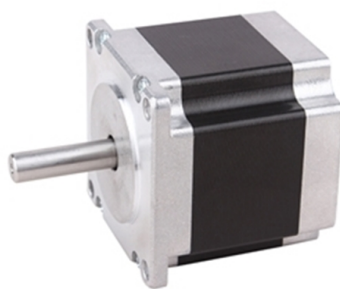


Figur 2: X20CP1382

De 30 I/O porte er fordelt ud over tre I/O moduler kaldt hhv. X1a, X2a og X3a, og det ene analoge input kan bruges til PT1000 modstandstemperaturmålinger. Udover dette, består robotten af tre Nema 23 (Model nr.: MT-2303HS280AW-0B) stepper-motorer, som vha. fjederkoblinger til gevindstænger bevæger blyanten i hhv. x- og z- retningerne - og en stepper motor, der bevæger selve pladen, som udgør y-aksen i robottens plan. Dette minder mest af alt om en CNC maskine, kombineret med en 3D printer eller en laserskærer.

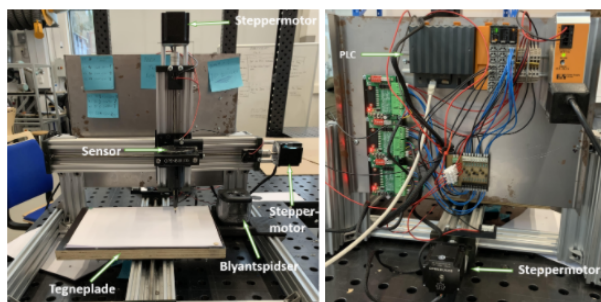
Step motorens specifikationer er [2]:

- Stepvinkel på $1,8^\circ$ med en præcision på $\pm 5\%$,
og 200 steps/revolution
- Holding Torque på 1,23Nm
- Spænding på 3.2V
- Strøm/Fase på 2.8A, 4 faser
- Fungerer i omgivelser mellem -20°C og $\sim +50^\circ\text{C}$



Figur 3: Nema 23 — MT-2303HS280AW-0B

Blyanten er monteret i et tool unit med en shuttle, som indeholder en jumboblyant med blødt bly, samt en fjeder for at kunne regulere blyanttrykket på papiret. Der er monteret tre normally open microswitches (også kendt som en miniature snap-action switch), som sender et svagt strømsignal igennem ledningen når de bliver aktiveret, dette kan bruges til at kalibrere tegnerobotten. De tre primære switches er placeret således, at når robotten køres ud i hjørnerne af x- og y-akserne, og hele vejen op på z-aksen - så aktiveres de. Derudover er der en fjerde, som er placeret lige over blyanten, som gør, at når den får kontakt med papiret under den, bliver blyanten trykket op, og rammer dermed microswitchen. Dette kan bruges til, at informere PLC'en om at blyanten er på papiret. Den elektriske blyantspidser er placeret ude i højre side af x-aksen (relativt til robottens front). Dertil er der også monteret en justerbar strømkilde. Alt dette var blevet udvalgt og sammensat forud for projektets start.



Figur 4: Fysisk opstilling

2.2 Java [1]

Java er et objektorienteret tredjegerations-programmeringssprog, som benytter klasser til at danne en skabelon for objekter samt constructors til at skabe et nyt objekt af klassen og metoder til at bestemme objektets adfærd og kunnen. Java er endvidere kendetegnet ved at være et sprog, hvis programmer kan afvikles på nærmest enhver enhed, og er baseret på grundpincipperne vedrørende sekvens, selektion og iteration.

Al programmering i Java bygger på 3 simple delelementer:

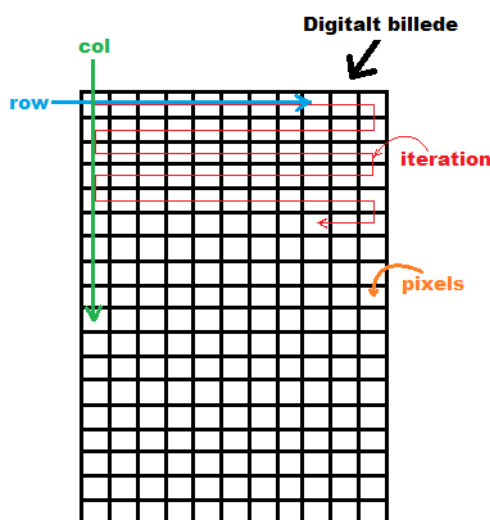
- **Boolean expression** som returnerer hvorvidt, et bestemt boolsk udsagn er TRUE eller FALSE
- **If/Else statements** som udfører et bestemt stykke kode afhængig af hvorvidt, et eller flere boolske udtryk er sande
- **Loops** som gentagende udfører et bestemt stykke kode, herunder **for/for-each loops** som udfører koden et forudbestemt antal gange, og **while/do-while loops** som afhænger af et boolsk udtryk, dvs. udfører et stykke kode, så længe det boolske udtryk er sandt.

2.2.1 Indlæsning af digitalt billede

For at kunne importere billedet, der skal behandles i Java og derefter tegnes af robotten vha. Automation Studio på PLC-siden, benyttes Picture klassen, som er blevet udleveret som en del af projektet sammen med en række andre klasser i starten af projektforsløbet. Picture klassen har en række forskellige måder at indlæse og behandle billeder på, heriblandt fire forskellige constructors som alle tager forskellige input-parametre for at indlæse et billede. I dette tilfælde bruges en constructor, der tager billedets navn og fil-sti som input - her benyttes dog en fil-position, der er relativ til Java-kodens position for at muliggøre eksekvering af kode på tværs af forskellige IDE'er (integrated development environment) og PC'er.

2.2.2 Pixelanalyse

Der anvendes et dobbelt for-loop til at iterere gennem alle pixels i det digitale billede, dvs. et loop inden i et andet loop. Det ydre loop itererer gennem billedets første kolonne (yderst til venstre) af pixels, og ved at implementere et indre for-loop for hvert element i kolonnen itereres der gennem billedets vandrette række af pixels. Med andre ord itererer det ydre loop gennem pixels i y-retning, og det indre gennem pixels i x-retning. For hver anden række itereres baglæns for at optimere blyantens bevægelser, ved at undgå at robotten skal køre blyanten hen til position $x = 0$ for hver række. for hver pixel der findes, returneres RGB-værdien til gråskalering af billedet. (Se appendix A: Java image processing main)



Figur 5: Pixel-iteration

2.2.3 Grayscaleing

I luminance klassen er der en metode, som sætter et givent farve objekts variable R,G,B (Red, Green, Blue, hhv.) til alle at være en nuance af sort (dvs. et sted mellem sort og hvid). Der er givet en luminance klasse, som består af meget lidt kode, den metode som bruges, laver et givent farve objekt om til den tilsvarende gråtone. Det gøres ved først at udregne den givne pixels luminans vha. formlen $((\text{Red value} \times 299) + (\text{Green value} \times 587) + (\text{Blue value} \times 114)) / 1000$, og så sætte både R,G og B til luminansen. Denne værdi rangerer i intervallet 0-255, hvor 0 repræsenterer sort og 255 repræsenterer hvid.(Se appendix B: Java convert image metode)

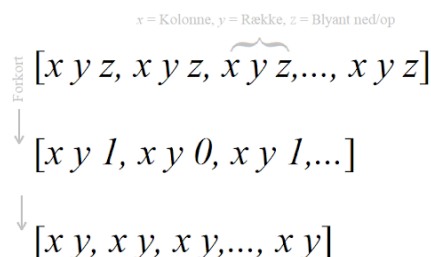
Afhængig af hvor mange konturer der ønskes på det tegnede billede, fastsættes en værdi i grayscale intervallet, som afgør hvorvidt den enkelte pixel skal fremstå mørk eller lys

(Om der skal tegnes eller ikke). Er den fastsatte værdi meget tæt på 0, vil kun billedets mørkeste konturer fremgå på det tegnede billede, og jo mere den fastsatte værdi forøges, desto lysere konturer vil endvidere fremgå på det tegnede billede. Herefter kan alle pixels med grayscale-værdi tilsvarende, eller højere end den fastsatte værdi tilskrives værdien 1 (for "tegn"), og pixels med grayscale-værdi lavere end den fastsatte værdi, tilskrives værdien 0 (for "tegn ikke").

2.2.4 Generering af PLC protocol-streng:

Målet er herefter at omdanne informationen om det digitale billede, til noget som med struktureret tekst kan bearbejdes og udføres af PLC'en. Der tilføjes row og col variable for hhv. rækker og kolonner (x og y) idet der itereres gennem billedets pixels, og disse indsættes i én lang streng sammen med værdien der angiver hvorvidt, der skal tegnes eller ej (z). (Se appendix C: Java process image metode)

Strengen kortes yderligere ved blot at returnere de koordinater for hvilken "tegn/ikke tegn"-værdien skifter.



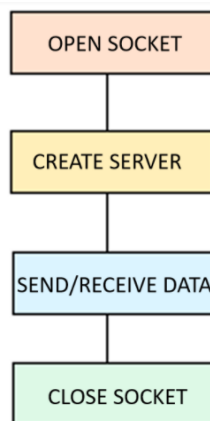
Figur 6: Forkortelse af streng-protocol

Eksempel på koordinater:

x1	y1	z1	x2	y2	z2
100	10	1	115	10	0
80	11	1	125	11	0
60	12	1	145	12	0
40	13	1	155	13	0

2.3 TCP/IP & Socket

TCP (Transmission Control Protocol) og IP (Internet Protocol) udgør kerneprotokollerne på nutidens internet, og er med tiden blevet den industrielle standard for sammenkobling af flere enheder på netværker. Hvis man splitter TCP/IP op i to protokoller, er der først og fremmest TCP-delen, som er den del, der tager sig af selve behandlingen af data. En TCP klargør data til at blive sendt, ved at tage mængder af data og kompilere dem til datapakker, som så muliggør at en TCP på den anden side af eksempelvis en socket, kan udpakke dataen igen. Dette er lidt som at zippe en fil, for at kunne sende den hurtigere da den fylder mindre, hvorefter en bruger på en anden computer unzipper den når de skal tilgå den. Den anden del er IP-protokollen, som muliggør, og sørger for, at datapakkerne bliver sendt fra din TCP-klient til den rigtige modtagende TCP-klient. TCP benytter portnumre som muliggør etableringen af datastrømme fra og til samme netværkshost. Ved etablering af disse datastrømme, (afsendelse og modtagelse af forskellige data), oprettes såkaldte sockets, på de to eller flere netværksenheder hvorimellem der ønskes kommunikation (en datastrøm). Sockets er et stykke software, som netop udnytter portnumre og IP-adresser til at oprette en server, hvortil flere enheder kan forbindes via TCP. Socket-kode på en netværksenhed er opbygget således at der først åbnes en socket, dernæst oprettes en server med portnummer og IP-adresse, der sendes/modtages data og slutteligt lukkes socketen.



Figur 7: pixel-iteration

Denne teori anvendes i projektet til at oprette en datastrøm mellem PC'en, hvorpå Java-softwaren bearbejdes, og PLC'en som styrer selve tegnerobottens hardware. Der programmeres altså sockets i Java på PC'en og i structured text på PLC'en. Port's og sockets er det man bruger til at fortælle en remote host, hvilke funktioner/processer fra local host den skal udføre. Hvis man skal forklare dette koncept med vores tegnerobot, bruger vi en socket til, at fortælle PLC'en hvor den skal tegne. Denne information kommer fra Automation Studio, som selv har fået koordinaterne fra vores Java kode via en socket. Et ID-nummer kan variere fra platform til platform, og er derfor ikke unikt eller har en fast

form som for eksempel en IP-adresse har. Det er dog yderst vigtigt når man snakker om socket, det er nummeret der er tildelt processen når man starter en socket. Det er værd at have i mente, at ID kan blive ændret hver gang man starter socketen. Det socketen gør er, at sætte det til noget unikt som kunne passe til den proces, man gerne vil lave. Den husker dette, så man ikke hver gang skal fortælle begge parter det nye ID.

2.4 Logiske kredsløb

Inden for logiske kredsløb snakker man om to tilstande, enten er noget falsk eller sandt, med andre ord, tændt eller slukket. Hvis der er tændt, kommer der strøm igennem, og de objekter der kræver strøm tændes, indtil de får en besked om, at der nu er slukket, for så stopper motoren, eller hvad man nu har af output. Man kan også komme ud for at der sker noget når programmet opfanger en logisk ændring altså at man går fra tændt til slukket eller omvendt. Dette kaldes for RISING eller FALLING, det der sker i tegnerobotten. Den stepper både i stadiet RISING men også FALLING, dette er det man kalder for en operation. Den specifikke operation her er en OR-operation. Når man snakker logiske kredsløb, snakker man om forskellige operationer, såsom AND, OR og NOT.

AND Gate

AND Gate er som udgangspunkt LOW, medmindre begge inputs A og B er HIGH, hvilket giver et output på LOW



Figur 8: AND Gate

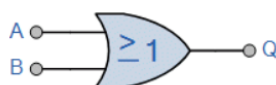
Et eksempel fra Automation Studio er når vi skal kalibrere, skal x og y stå mellem 2 værdier hver, og denne kan kun blive True hvis begge er imellem de beskrevne værdier.

```
IF (xCounter >= 3280 AND xCounter <= 17680 AND yCounter >= 1364 AND yCounter <= 13364) THEN
```

Figur 9: AND eksempel

OR Gate

OR Gate er LOW når alle dens inputs er LOW, og er HIGH når bare en eller begge af dens inputs bliver HIGH.



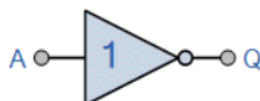
Figur 10: OR Gate

Et eksempel her kunne være

```
ELSIF (zCounter = 500 OR zStop = TRUE) THEN
```

Figur 11: OR eksempel

NOT Gate (Inverter) Ved en NOT gate, gælder det således at dens output Q som standard er HIGH, hvis input A er LOW, og når input A bliver HIGH, ændre output Q sig til LOW - Så den bliver inverteret.



Figur 12: NOT Gate

Signal edge

Signal edges er brugt i elektronik til at detektere en logisk ændring i et digitalt signal.

Rising edge

En rising edge, også kendt som positive edge detekterer er når et digitalt signal ændrer sig fra logisk LOW til logisk HIGH.

Falling edge

En falling edge, også kendt som negative edge detekterer når er et digitalt signal ændrer sig fra logisk HIGH til logisk LOW.

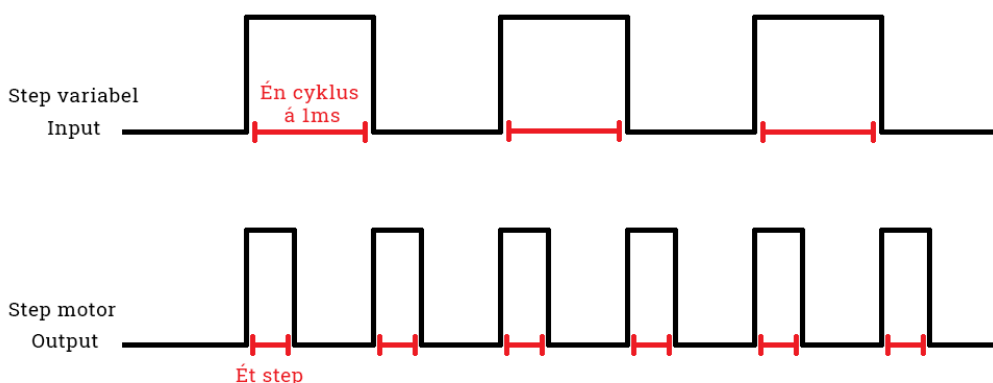
Eksempel i praksis

Et eksempel på brug af både rising og falling edge, er vores setup med stepper motorerne. PLC'en er forbundet til tre ST330-V3 driver boards som sender signal igennem til NEMA 23 stepper motorerne. Det muliggør kontrol af stepper motorerne, ved at man inverterer værdien af en inputvariabel som er forbundet til driverboardet.

```
DRAW:
  IF (killswitch <> TRUE) OR (counter < instSize) THEN
    IF (xStop <> TRUE) THEN
      IF (xArray[counter] * scale > xCounter) THEN
        xDirection := TRUE;
        xStep := NOT xStep;
        xCounter := xCounter + 1;
      ELSIF (xArray[counter] * scale < xCounter) THEN
        xDirection := FALSE;
        xStep := NOT xStep;
        xCounter := xCounter - 1;
      ELSIF (xArray[counter] * scale = xCounter) THEN
        ROBOSTATE := PENCILDOWN;
      END_IF
    END_IF
  END_IF
```

Figur 13: Draw i ST

Logisk set betyder det, at hvis man har step variabelen som input og stepper motorens bevægelse som output, så vil der komme et output på både Rising- og Falling edge.



Figur 14: Rising OR Falling edge

2.5 PLC

Programmable Logic Controller, oftest mere kendt som PLC, er bredt kendt i den industrielle verden og er set som en form for industriel standard ift. kontrol og automation af maskiner indenfor produktion. Dette skyldes primært en række egenskaber, som adskiller den fra alternative controllers til samme formål. En især karakteristisk fordel ved selve PLC'en, som også viser hvorfor den er så udbredt i industrien er, at den i forhold til andre microcontrollers af ellers samme kaliber, er bygget utroligt stabilt som gør den yderst holdbar i det industrielle miljø. Ydermere er dens konstruktion forstærket til at kunne modstå temperaturer fra -25°C til 60°C , stærke vibrationer, elektrisk støj og en vis fysisk belastning ved eksempelvis flytning, tab eller bump. Samtidig kan PLC'en køre live simulationer med identiske omstændigheder til virkelighedens, hvorved programmer kan testes forud i forskellige tænkelige situationer for at undgå fejl. Man kan næsten forstille sig en fysisk PLC som en legoklods, hvis der ønskes ekstra RAM, kan en ekstra PLC kobles op ved siden af og så er RAM'en fordoblet. Især RAM er en mangel hos PLC'en, da den primært afhænger af lagring på ROM, dette medfører, at den ikke kan omprogrammeres imens den kører det pågældende program. Dog besidder PLC'en, sammenlignet med andre microcontrollers og CPU'er, nogle afgørende fordele, der underbygger dens solide fremtræden i industrien.

2.5.1 Struktureret tekst [3]

Programmeringssproget anvendt til at kode PLC'en er Structured Text (ST). Dette sprog, som er udviklet og designet til programmering af PLC'er, og er udledt af et gammelt kodesprog kaldet Pascal, bygger på det funktionsorienterede programmeringsprincip, men tager (ligesom Java) stadig udgangspunkt i grundprincipperne sekvens, selektion og iteration. Det som gør ST særligt brugbart til at programmere PLC'er, (for bare at nævne ét eksempel) er muligheden for at programmere såkaldte statemachines, hvor PLC'en udører et bestemt program alt efter hvilken "state" den befinder sig i. Det bygger som navnet siger på tekst, det vil sige at i modsætning til ladderdiagrammer som er grafisk baseret, nemmere at forstå hvis man i forvejen kender til andre lignende programmeringssprog. Struktureret tekst fylder også mindre end dens modpart.

Opbygning

Struktureret tekst er opbygget ved at man har tre hovedprogrammer, som er hhv. INIT, CYCLIC og EXIT.

PROGRAM_INIT

INIT programmet køres netop én gang, når PLC'en starter op. Det er her hvor man bl.a. giver sine variable en startværdi.

PROGRAM_CYCLIC

CYCLIC programmet køres én gang pr. cycle, hvor cycle time og tiden mellem cycles er defineret på selve PLC'en. Her kan man med fordel sørge for at skrive et program som ikke tager længere tid at evaluere end det tager PLC'en at gennemføre en cycle. Det er for eksempel en rigtig dårlig ide at bruge løkker (WHILE, DO WHILE og FOR) i sin CYCLIC, medmindre man har helt styr på at løkken slutter inden PLC'en har gennemført en cycle, ellers går den pr. automatik i SERVICE MODE.

PROGRAM_EXIT

EXIT programmet køres netop én gang, når PLC'en slukker. Her kan man med fordel sørge for f.eks. at slukke diverse motorer, lamper, m.v.

Dertil har man forskellige lokale og globale variabler og typer, så det er muligt at kunne kombinere flere PLC programmer på tværs.

2.5.2 Input/output

x/y/z-Step: De 3 steppermotorer er forbundet til PLC'en via 3 boolske inputvariable, navngivet xStep, yStep og zStep. For hver gang at der sker en ændring i signalet på dette input, altså at Step går fra logisk lav til logisk høj eller omvendt, laver moteren 1 step

som output. (Rising edge OR Falling edge) For at få motorerne til kontinuerligt at steppe, implementeres derfor følgende kode i det cykliske PLC-program:

$$\text{Step} = \text{NOT Step};$$

Dvs. at Step-variablen negeres for hver cyklus, og dermed tager motoren 1 step pr. cyklus.

x/y/z-Direction: Til hver steppermotor er der forbundet yderligere en boolsk input-variabel, nemlig hhv. xDirection, yDirection og zDirection. Direction-variablen styrer motorens rotationsretning afhængig af om variablen er logisk høj eller logisk lav. Er Direction-variablen logisk høj, bliver outputtet at motoren stepper i højre rotationsretning, og er Direction logisk lav, steppes der i venstre rotationsretning.

x/y/z/b-Stop: De 3 microswitches som sidder sammen med motorerne, for enden af alle akserne, er forbundet til PLC'en via 3 boolske input navngivet xStop, yStop, zStop. Den sidste microswitch som sidder lige over blyanten og dedekterer hvorvidt blyanten rammer papirer, er ligeledes forbundet via inputvariablen, bStop. Denne inputvariabel i sig selv, har intet direkte output på hardwaren. Derimod har Stop-variablene i programmet direkte indflydelse på Step-variablene idet de afgør hvorvidt motorerne skal steppe eller ej. Når microswitchesne bliver trykket ind, går Stop variablen logisk høj, og afbryder "Step = NOT Step"-processen.

2.5.3 Koordinatbehandling

Når koordinaterne modtages fra Java-programmet på sin rå form i pixels (x, y) må de først bearbejdes til noget som kan bruges i PLC-programmet. Java programmet kører alle pixelene igennem, og sletter alle dem der ikke opfylder vores "threshold" så vi står tilbage med et sort hvidt billede hvor vi kender alle x,y koordinaterne på hver enkelt sorte/mørke pixel. Dernæst slettes alle de hvide koordinater mellem to "ændringer" en ændring er når vi går fra hvid til sort eller omvendt. Det vi står tilbage med er altså et start- og slut-koordinat på alle de sorte streger der er på hver linje. Fra Java siden sender vi disse streger som to koordinatsæt i en streng, som så kan blive oversat af PLC programmet. Oversættelse sker bla. ved hjælp af delimiters, så programmet kan se hvad der er det første x-koordinat og hvad der er det første y-koordinat osv. Herefter sættes de ind i forskellige arrays vi har et "xArray", "yArray" og "zArray", man kan diskutere om vi behøver et "zArray". Det skal lige nævnes at når de er blevet opdelt men inden de sættes

ind i arraysne bliver de lavet om til integers. Måden vi finder rundt i vores array er med en simpel "counter" som er ens i alle arrays så vi har den rigtige x-værdi til den rigtige y-værdi.

Ved at lave lidt matematik kan vi finde den skalar der bruges til at regne vores koordinater om til steps. Skalaren er bestemt ved et tegne en lige streg på 1 cm i robotopstillingen. Der tilføjes en countvariabel til et tomt program, som tæller antal steps på den motor som bruges til at tegne strengen. Herudfra kan antal steps pr. milimeter beregnes, og med en opslagsværdi for pixels pr. milimeter, kan steps pr. pixel slutteligt beregnes. (Se appendix E: Matematiske beregninger til koordinatbehandling). Det vi kommer frem til at at det er lidt over 25 steps pr. pixel, dette er så vores skalar som vi skal gange på vores x,y,z -værdier for at finde ud af hvor vi befinder os i forhold til den counter der holder øje med hvor vi befinder os på x- og y-aksen i forhold til (0,0). Herefter ganges koordinaterne med skalaren 25 steps/px.

2.5.4 Statemachine [5]

En Statemachine eller på dansk en tilstandsmaskine, er ikke som sådan en maskine men mere en tankegang som baserer sig på digital logik der bliver sat ind i et program. En statemachine er et hvilket som helst program der kan gemme en tilstand på et givent tidspunkt. Det betyder at den fx. kan sætte en robot i en State og bliver i den State indtil statens betingelser er opfyldt, hvor efter den bliver sendt videre til den næste State. Der kan godt være flere forskellige veje ud af en State, med dette menes at man sagtens kan bruge IF-statements så den State den bliver sendt videre til ikke nødvendigvis er den samme, som forrige gang den var i den givne State.

Aktuelle state	Funktion	Næste state
INIT	Her sættes alle variabler til de ønskede værdier inden programmet startes.	READY
READY	Tjekker om Java er færdig med at sende koordinater.	CONVERT
CONVERT	Converterer den streng af koordinatsæt fra Java til individuelle INT som den derefter sætter ind i vores x,y,z Arrays.	RESET
RESET	Kører først med z-aksen til at den aktiver dens sensor og derefter samtidig med x,y indtil deres sensor aktiveres.	CALIBRATE

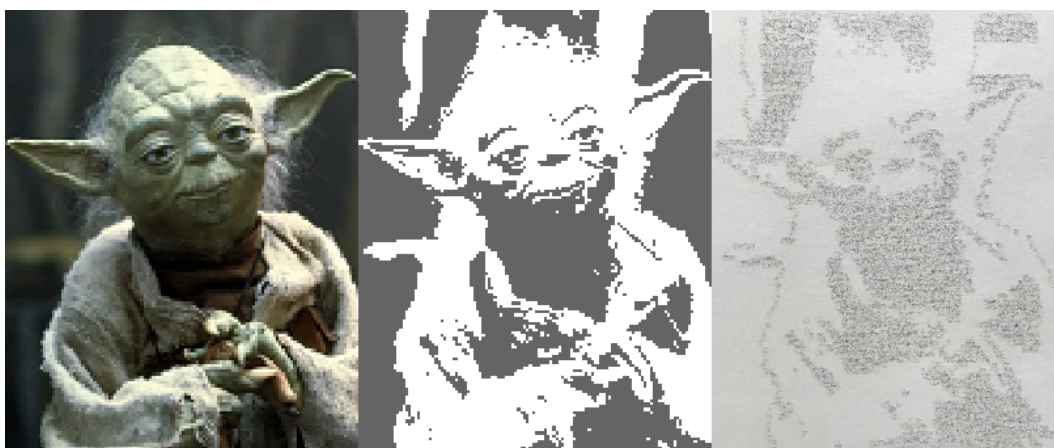
CALIBRATE	Tager blyanten fra den position den var i RESET og kører det hen til et punkt vi har valgt til at være (0,0).	DRAW
DRAW	Flytter blyanten (som ikke rører papiret) hen til koordinat.	PENCILDOWN
PENCILDOWN	Aktiverer steppermoteren i -z-retning indtil blyanten papiret.	DRAW2
DRAW2	Flytter blyanten (som tegner på papiret) hen til næste koordinat.	PENCILUP
PENCILUP	Aktiverer steppermoteren i z-retning indtil blyanten er et stykke over papiret.	CHECKLINE
CHECKLINE	Tjekker om y-værdien i det aktuelle koordinat er tilsvarende y-værdien i følgende koordinat, hvorved robotten skifter state til DRAW. Er y-værdierne forskellige skiftes til NEXTLINE.	DRAW V NEXTLINE
NEXTLINE	En countvariabel tæller antallet af tegnede rækker, og i denne state tjekkes hvorvidt countvariablen har nået 30. Er antallet af tegnede rækker over denne værdi, skiftes state til SHARPEN. Hvis dette ikke er tilfældet, går skiftes state til DRAW	DRAW. V SHARPEN
SHARPEN	Flytter blyanten hen til spidseren, ned i spidseren, aktiverer spidseren, inaktiverer spidseren og løfter blyanten op igen.	BACK
BACK	Denne state finder tilbage til de x,y-værdier som var aktuelle inden robotten gik i SHARPEN	DRAW
STOP	Når der ikke er flere koordinater at bearbejde, går Robot fra DRAW til STOP og flytter blyanten af papiret.	

3 System test

Vi kan uden at decideret teste noget, vurdere vores systems mangler og begrænsninger, både på hardware og software siden. Nogle ting kunne have været ændret for at løse enkelte problemer men så ville der være andre ting der ikke ville kunne udføres. Eksempler på dette ville være at vores system sender alle koordinaterne med det samme til PLC'en.

Dette betyder at vi kan pakke vores ting sammen og lade den køre over natten hvis vi ville, problemet med denne løsning er vi så er begrænset af hvor meget plads der er på PLC'en til at gemme disse koordinater men vi har kun en PLC til rådighed. Den kan gemme ca. 999 linjer koordinater, og da der er 2 koordinater pr. linje, kan vi altså kun lave billeder med 1998 koordinater.

På billederne nedenfor ses resultatet af vores systemtest. Yderst til venstre ses det digitale billede som vi har genskabt, midterst det processerede billede som er grayskalet efter en fastsat værdi, som passede bedst til netop dette billede, og yderst til højre ses så det billede som robotten har formået at tegne med blyant på papir.



Figur 15: Side by side sammenligning

3.1 Hardware tests

De tests vi foretog os på hardware siden, var at vi ville finde ud af hvordan vi fik den bedste opløsning med en blyant og hvor lav en cycle time vi kunne presse systemet ned på uden at der blev lavet fejl, desuden hvordan vi kunne løse forskellige problemer der var med hardwaren, det sidste er mere bare små forbedringer til robotten mere end der er system tests.

Så hvordan opnår man den bedst mulige opløsning fra en 3 akset printer med en tyk blyant. Når der snakkes om rettelser, blev der skuet over muligheder for skarphed, omkring hvordan man ville kunne formindske pixelstørrelsen eller hvor mange pixels der ville være mulighed for at kunne presse ind i hvert billede. Vi havde til tanke at dette kunne opnås, ved at spidse blyanten markant mere. Dog skulle det også tages til eftertanke, at en blyant som er spidset for skarpt, ville resultere i en høj risiko for overrivning af tegnepapiret. Derpå kom vi til den konklusion, at lade blyanten forblive relativt bred i bundareal, men dog lade den spidse mere end som den var i sin oprindelige form.

3.2 Cycle time test

Dertil stod vi med et hardwareproblem, som blev ordnet med en software løsning. Det vi ville teste var i bund og grund hvor hurtigt vi kunne kører med vores stepper motor. Vi kører fra start af med en cycle time på 2 millisekunder, dette kunne robotten sagtens være med til, vores mål var at få den ned på 1 millisekund, hvis dette var muligt. Vi startede med at prøve med 1,5 millisekunder. Der kørte den fint dog var det som om den ikke blev modtaget i CHECKLINE-staten som skiftede linje. Vi er dog usikre hvorvidt dette er rigtigt, fordi det gik så hurtigt, at vores monitor i Automation studio ikke kunne følge med. Da vi reelt var usikre på om der overhovedet var fejl, satte vi cycle time ned til 1 millisekund. Her blev det tydeligt at vi havde overbelastet vores system. Der manglede både hele linjer og selv tegningen blev skæv. Så den optimale cycle time, for vores program, ligger altså mellem 1,5 millisekunder og 2 millisekunder, vi synes det ville være en dårlig prioritering at finde præcise sted, så vi fortsatte med 2 millisekunder som virkede perfekt indtil da. Som nævnt tidligere var dette et problem der kunne være løst fra vores program ved fx. at med vilje gøre billedet skævt i den anden side for at udligne, skævheden fra cycle time 1 millisekund.

3.3 Software tests

Også på software siden, blev der lagt vægt på at forkorte eksekveringstiden af både Java og Structured Text, for at få en hurtigst mulige tegnerobot. Første undersøgelse gik ud på at se hvor hurtigt kunne vi sende vores koordinater gennem socketen til PLC'en fra Java-programmet, uden at der skete fejl. Der var dog ikke andet end at prøve sig frem ligesom med cycle time testen, og præcis som der, endte vi heller ikke med et specifikt tal men nærmere et område hvori vi valgte et tal. Den sidste test vi kørte var at se hvor lav vores delay på vores converter kunne blive og her endte vi med et delay på 100 millisekund pr. koordinatsæt, det vil altså sige vi kan sende 10 koordinatsæt på 1 sekund.

4 Diskussion

Inden man går i gang med sådan en opgave - at programere en tegnerobot - er der mange beslutninger som skal tages. I dette projekt var mange beslutninger dog taget på forhånd, f.eks. præcis hvordan robotopstillingen så ud, hvor sensorer og motorer sad osv. Det var på forhånd også besluttet hvilke programmeingssprog vi skulle anvendes, og at kommunikationen skulle ske over en socket. Så hvad er der egentlig op til en selv? Jo vi startede med at se på hvordan vi reelt ville tegne: Skulle vi sætte en prik for hvert koordinat eller skulle vi lave edgedetection, (altså finde en outline på billedet og så kun tegne outline)? Man

kunne også tegne linjer. Som det fremgår af rapportens tidligere afsnit, valgte vi at tegne streger, hvor vi alstå benytter koordinatpar, som start og slut på en streg.

4.1 Billede kvalitet

Vi valgte fra start at vi ville lave vores tegnerobot så effektiv som muligt. Dette viste sig er være en sværere opgave end først anslået da vi rendte ind i flere problemer, før vi til sidst havde en løsning som kunne tegne et hvilken som helst motiv/billede. På Java-siden valgte vi at sende to sæt koordinater i en streng i stedet for at sende et koordinatsæt ad gangen. Dette gjorde selvfølgelig at det gik dobbelt så hurtigt, for som nævnt tidligere kan man ikke bare sende koordinaterne så hurtigt som man vil, da programmerne simpelthen ikke kan følge med. Der er nødt til at være et delay mellem to strenge. Vi besluttede os også for at vi ville sende alle koordinaterne på en gang. Denne løsning har én stor ulempe og én stor fordel. Kunne vi gøre det igen havde vi nok valgt at skrive programmerne sådan at vi sendte så mange koordinater som muligt, slettede dem og gentog processen. Ulempen er at vores tegninger ikke kunne blive særligt store måske 1/3 af papiret, og det havde selvfølgelig en indvirkning på detaljerne i vores tegning. Når ting bliver så små og det redskab vi har til at tegne med, er en tyk blyant bliver billedet pixeleret. Ligesom hvis man ser et billede på sin computer i meget dårlig opløsning eller, at man et sekund har dårligt internet og billedet ikke indlæses korrekt. Fordelen ved at sende alle koordinaterne er så at når de er sendte skal man ikke være tilsluttet længere end at man kan tage sine ting og smutte hjem hvis det var det man ville. Det havde nok været smartere at gøre det at man sendte nogle koordinater ad gangen, grunden til at vi valgte den anden løsning var at vi ikke vidste hvor lang tid det ville tage at tegne en tegning da vi startede med projektet. Vi så robotten lidt som en 3D printer og de kan sagtens bruge 6+ timer på at printe noget, med dette som vores eneste guide besluttede vi at det at få sin computer med hjem var mest optimalt. Vi har faktisk kapaciteten til at lave lidt større billeder, fordi vi sender vores z-koordinater med fra Java af, men bruger dem ikke i PLC koden. Når vi skal kører på z-aksen bruger vi den sensor der sidder på blyanten til at se hvornår vi rammer papiret. Hvis vi havde implementeret det, ville vores program sende 4 karakterer mindre pr. koordinat par. Grunden til at det er 4 og ikke 2 er fordi der så også bliver fjernet en delemeter som er det der bliver brugt til at convertere strengene. Det virker måske ikke som om det ville være supersvært at ændre, men der var ikke tid til det og vi prioriterede at kunne tegne, også selvom det måske ikke er den mest strømlinede løsning.

4.2 Tegne i begge retninger

Hvis man kigger på vores Java programmering kan man også se at der er en IF-statement som tjekker om noget kan divideres med 2. Det der sker her, er at vi prøver at efterligne

en rigtig printer og samtidig effektivisere robotten. For i stedet for at gå fra højre mod venstre og så løfte blyanten, bevæge sig tilbage til højre osv. Så gør vi det at vi tegner i begge retning ved at ændre det koordinat der kommer først, så hver andet koordinatsæt er invers.

4.3 Blyantspidser

Et andet sted vi kunne havde gået en anden vej end den vi gjorde, var omkring hvornår vi spidsede vores blyant, her valgte vi at den skulle gøre det når den havde tegnet 50 streger. Her kunne man også have valgt at bruge en timer men dette synes vi var en dårlig idé når man laver mange små streger, tager det længere tid end de længere streger som så samtidig slider mere på blyanten. Fordi når der er flere streger på en linje så skal blyanten først løftes rykkes og sænkes før man tegner igen. Derfor ville det løsning ikke tænke på hvornår blyanten at slidt men bare hvor lang tid den har kørt. Vores løsning med en MOD-operator holder øje med slitagen i stedet.

4.4 Langsom testning

Nu har vi snakket meget om effektivitet men en af de ting som var mindst effektive var vi være gang vi ville teste noget nyt kode gik robotten i sin RESET-state og derefter CALIBRATE som nok tog 2-3 minutter hvis vi havde kommenteret dette ud kunne vi afteste tingene en del hurtigere, robotten ville godt nok ikke starte i (0,0) men hvis nu bare man skulle tjekke noget om hvordan den kørte frem og tilbage var dette egentlig underordnet hvor den tegnede tegningen henne. Havde testen bare gået ud på at tjekke om vore NEXTLINE virkede, havde det været ligegyldigt.

5 Konklusion

Socket og TCP/IP kommunikationen viste sig at have afgørende betydning for robottens funktion, da dette element er vitalt for robottens evne til at tegne den fysiske kopi af det digitale billede. PLC delen og Java delen er også yderst essentielle, men kunne anses som yderst optimeringsvenlige i form af, at forbedringer eller ændringer kunne implementeres, hvorved en effektivitetsforøgelse i eksempelvis Javaens pixelanalyse og tegningens kvalitet kunne opnås.

På trods af diverse udfordringer, så som valget af metoden til decideret at tegne, forhindringer som begrænsning i PLC'ens hukommelse og kommunikation mellem Java og PLC, er tegnerobotten med success bragt til live gennem programmering. Det blev med brug af Java muligt at uploade og indlæse informationer fra et digitalt billede og med billedeprocessering ved brug af pixelanalyse og gråskalering hvorefter x,y,z-koordinatsæt ved yderligere konvertering bliver udnyttet af tegnerobotten, til at tegne det endelige fysiske billede.

Hermed kan det endelig konkluderes, at der med dette projekt, successfuldt er udviklet en - om ikke uoptimal - så ihvertfald funktionel og brugbar metode til at processere og konvertere et digitalt billede til en fysisk tegning med udgangspunkt i programmering af en 3-akset CNC-type robot.

6 Appendix

A Java image processing main:

```
System.out.println("Starting image processing");
// For every column in every row, where every other row iterates backwards
for (int row = 0; row < imgpro.image.height(); row++) {
    if (row % 2 != 0) {
        for (int col = 0; col < imgpro.image.width(); col++) {
            imgpro.convertImage(col, row);
            imgpro.processImage(col, row);
        }
    } else {
        for (int col = imgpro.image.width() - 1; col >= 0; col--) {
            imgpro.convertImage(col, row);
            imgpro.processImage(col, row);
        }
    }
}
```

B Java convert image metode:

```
public void convertImage(int col, int row) {
    // Convert image to grayscale, then convert to black/white (no shading)
    image.set(col, row, Luminance.toGray(image.get(col, row)));
    grayVal = image.get(col, row).getRed();

    if (grayVal < threshold) {
        image.set(col, row, gray);
    } else {
        image.set(col, row, white);
    }

    // Update frame displaying image, and add current instruction to string
    image.show();
}
```

C Java process image metode:

```
public void processImage(int col, int row) {
    // If the intensity of gray is higher than the set limit
    // Pencil down (Draw), if not, Pencil up (Stop drawing)
    draw = (grayVal < threshold) ? 1 : 0;

    // Only send an instruction if draw is different from prevDraw
    // (Image goes from black to white, or vice versa)
    String s;
    if (draw != prevDraw) {
        if (draw == 1) {
            s = (col + 1) + " " + (row + 1) + " " + draw + ",";
        } else {
            s = (col + 1) + " " + (row + 1) + " " + draw + ",";
        }
        inst.add(s);
    }
    prevDraw = draw;
}
```

D Java exit program main:

```
int x = 1;
for (int i = 0; i < imgpro.inst.size(); i++) {
    if (i % 2 != 0) {
        System.out.print("Sending data [" + x + "] of [" + imgpro.inst.size()/2 + "] | ["
            + String.format("%.2f", (2*((float)x/(float)imgpro.inst.size()*100))) + "%] done | Data: ");
        imgpro.instructions = imgpro.instructions.concat(String.valueOf(imgpro.inst.get(i)));
        robcom.write(imgpro.instructions);
        System.out.println(imgpro.instructions);
        imgpro.instructions = "";
        x++;
    } else {
        imgpro.instructions = imgpro.instructions.concat(String.valueOf(imgpro.inst.get(i)));
    }
    Thread.sleep(250);
}
Thread.sleep(2500);
robcom.write("DONE");
```

E Matematiske beregninger til koordinatbehandling:

Der går 1000 steps til at tegne en streg på 1 cm med steppermotor, derfor:

$$1000 \text{ steps} = 1 \text{ cm} = 10 \text{ mm}$$

$$\frac{1000 \text{ steps}}{10 \text{ mm}} = 100 \text{ steps/mm}$$

Opslagsværdi på PPMM (pixels pr. mm) = 3,97 px/mm. Herefter beregnes skalaren i steps/px:

$$\frac{100 \text{ steps}}{3,97 \text{ px}} = 25,18892 \frac{\text{steps}}{\text{px}}$$

Afrundes til 25 steps/px.

Referenceliste

- [1] David J. Barnes Michael Kölling. *Objects First with Java: A Practical Introduction Using BlueJ*. Pearson, 2016. ISBN: 9781292159041.
- [2] MoTech Motor. *Nema 23 (57mm) - MT-2303HS280AW, Product Desc.* URL: <http://motechmotor.com/productDetail-0105-45.html>.
- [3] @PLCAcademy Peter. *Structured Text Tutorial to Expand Your PLC Programming Skills*. URL: <https://www.plcacademy.com/structured-text-tutorial/#what-is>.
- [4] B & R. *X20CP1382 Basic Information*. URL: https://www.br-automation.com/en-ca/products/plc-systems/x20-system/x20-cpus/x20cp1382/?fbclid=IwAR1KE3RSxJ8bx-x1kxzRg0pnw70cP8RTIFCmr6amVAm1mU1NR05uwiYF_c4.
- [5] technopedia. *State Machine*. URL: <https://www.techopedia.com/definition/16447/state-machine>.