# Struktureret tekst

## Opbygning af programmer

Samlet set har vi valgt, at lave fire forskellige programmer. Dette har vi valgt at gøre, for at holde det overskueligt, så vi nemt kan finde et bestemt stykke kode. Vi har lavet de fire programmer, som vil blive beskrevet senere i dette afsnit.

* Reset
* TCP
* Draw
* EmergencyStop

Vi har benyttet B&R’s hjælpeprogram til at finde informationer omkring struktureret tekst. Dette har været en god hjælp, hvis der er noget man har været i tvivl om.

Liste over globale variabler hvis der bliver henvist til en variabel i teksten og den ikke står som en lokal variabel står den her.

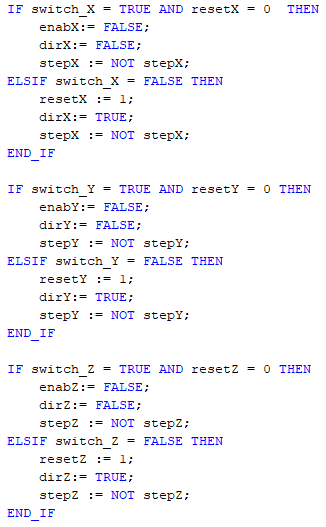
|  |  |  |  |
| --- | --- | --- | --- |
| **Navn** | **Type** | **Navn** | **Type** |
| dirX | BOOL | quit | BOOL |
| dirY | BOOL | sharpen | UINT |
| dirZ | BOOL | status | USINT |
| enabX | BOOL | testEnab | BOOL |
| enabY | BOOL | z\_move\_down | BOOL |
| enabZ | BOOL | z\_move\_up | BOOL |
| input | USINT [0..65535] | y\_move\_right | BOOL |
| posX | UDINT | y\_move\_left | BOOL |
| posY | UDINT | x\_move\_right | BOOL |
| posZ | UDINT | x\_move\_left | BOOL |
| reset | BOOL | newLineX | UDINT |
| stepX | BOOL | ValueZ | UINT |
| stepY | BOOL | penLength | REAL |
| stepZ | BOOL | placeInArray | UDINT |
| switch\_X | BOOL | placement | UINT |
| switch\_Y | BOOL; | newLineHelper | UDINT |
| switch\_Z | BOOL |  |  |

Reset-program

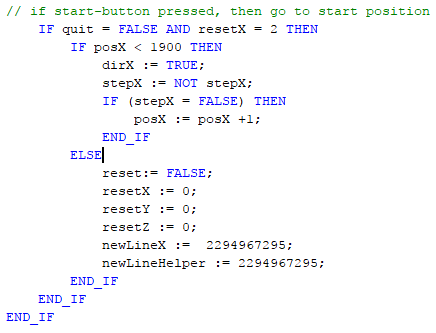
Dette program er lavet til at sætte robotten i den samme startposition, hver gang vi enten starter robotten eller af en eller anden grund ønsker at ‘genstarte’ programmet. Ud over at sætte robotten i startposition findes der også en række ‘testfunktioner’, som bruges til at køre med robotten på de forskellige akser. Reset-programmet benytter de knapper der er blevet monteret på de tre akser, som vi kalder x, y og z.

I dette program benytter vi en lille række lokale variable, som kan ses i nedenstående tabel.

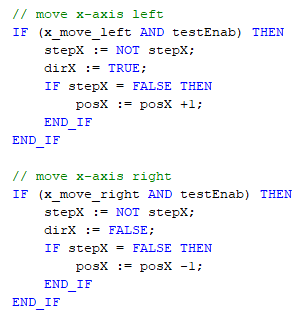
|  |  |
| --- | --- |
| **Navn** | **Datatype** |
| resetX | USINT |
| resetY | USINT |
| resetZ | USINT |
| timer | TON |

Størstedelen af programmet er indkapslet i et if-statement, som spørger, om reset = true og om testEnab = false. Dette sørger for, at programmet kører, når knappen reset er trykket ind, og dermed true. Herefter køres der igennem koden fra top til bund. Hvis disse betingelser er opfyldte begynder den på reset-sekvensen, som kører alle akserne tilbage til udgangspunktet. Dette gøres igen ved et if-statement, som spørger om en knap er trykket ind eller ej. Hvis knappen ikke er trykket ind sættes retningen af motoren mod den ønskede position hvorefter den kører med motoren indtil knappen bliver trykket ind. Umiddelbart efter spørges der om knappen er trykket ind og kører herefter ud igen, indtil der ikke længere trykkes på knappen. Dette kan ses på Billede 1. Dette gentages for hver af de tre akser, så vi ender ud med et udgangspunkt for resten af programmerne, så vi altid ved, at vi starter i samme position, hver gang vi kører programmet. Herefter sættes posX, posY og posY til nul, så de kan benyttes senere i koden. Dette gøres ved endnu et if-statement, som spørger om alle akserne er blevet resat, efterfulgt af en lille pause på 100ms, hvorefter værdierne sættes.

Billede 1: Udklip af reset-program

Derudover er der et stykke kode, som kører robotten hen i den bestemte startposition, som sættes i gang, når der bliver trykket på knappen ‘Start’. Som det kan ses på Billede 2 er der opsat et if-statement, som spørger om quit = false(startknap) og om resetX = 2. Dette sørger for, at der ikke kan trykkes start, før akserne er blevet resat. Herefter køres robotten i position ved hjælp af et if-statement på x-aksen. Der køres 1900 ‘steps’ ud af x-aksen.

Billede 2: Udklip af start-knap

Til sidst i dette program har vi seks if-statements, som hver gør det muligt at vi kan styre de forskellige akser. Disse har vi brugt i testfasen, hvor vi evt. skulle finde ud af, hvor robotten skulle starte, hvor blyantspidseren er placeret osv. Vi benytter et boolesk udtryk

Billede 3: Testknapper til styring af robot

TCP

TCP-programmet er vores modtage-program/TCPserver, der over en internet eller LAN forbindelse kan modtage et array fra en anden enhed. Dette array bliver sendt som en streng, men bliver lavet om så det modtages som et array (input) med 8 bit på hver plads. Selve koden er genbrug fra en opgave B&R har lavet med os. Dette betyder også, at selvom der er en del af koden der kan sende data, benyttes det ikke. Når strengen bliver modtaget af PLC’en lægges der et U, D, N eller Q ind på hver plads i arrayet, men da det er et USINT array lægges disse ind som decimaltal. Karaktererne i strengen bliver til nedenstående ascii kode.

Tallene bliver selvfølgelig sat ind i arrayet fra plads 0. Dette gør TCP’en for de pakker der modtages. Der er så det problem, at som programmet virker i version 5 kan der kun modtages 1 pakke på 64kb eller mindre, hvilket betyder hvis vores billedefil bliver større end 64kb vil den starte på pakke nummer 2, som den så lægger ind i arrayet fra plads 0 igen derfor begrænses arrayet til 65536 hvilket er en plads mere ned hvad en TCP pakke skal kunne håndtere. Dette gøres for at være sikker på, at det er hele pakke 1 der kommer med.

Af de variabler der bruges i denne del af programmet kan der ses en liste nedenfor:

|  |  |
| --- | --- |
| **Navn** | **Datatype** |
| tcp1 | TcpOpen |
| tcp2 | TcpServer |
| tcp3 | TcpRecv |
| tcp4 | TcpSend |
| tcp5 | TcpClose |
| state | UINT |

TcpOpen, TcpServer, TcpRecv, TcpSend og TcoClose er alle funktionsblokke fra biblioteket AsTCP.

Til sidst i programmet lukkes der for serveren for at der senere kan modtages ny data, uden at PLC’en skal genstartes. Dette kan gøres da serveren åbnes igen med det samme, men venter på at modtage ny data inden den lukker serveren igen.

Serveren bruger ip-adressen som PLC’en er blevet tildelt i konfigurationen. I dette setup bruges der en router så der er mulighed for at tilslutte flere enheder. Dette betyder så også, at der bliver brugt en ip ud fra routerens ip-range, i dette tilfælde bruges 192.168.0.103. der ud over bruges en port, som bliver defineret i programmet under variabel tcp1. Her bruges port 12345. Der kunne bruges en hel række andre porte, dog ikke porte der allerede er i brug som 5900 som bruges af visu-delen(Interfacet).

DRAW

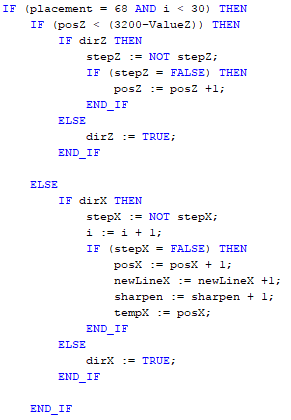
Draw programmet er den del der læser arrayet, og ud fra hvad der sker i arrayet ”input" sørger for at robotten bevæger sig.

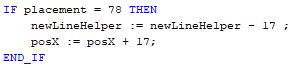
I programmet bruger vi en række lokale variabler, som kan ses neden for.

|  |  |
| --- | --- |
| **Navn** | **Datatype** |
| Activator | BOOL |
| tempX | UDINT |
| Timer | TON |
| sharpenderLen | UINT |
| i | UDINT |

Først i koden testes der, om robotten er blevet nulstillet ”reset”, og om der er blevet sat en længde på blyanten ”penLength”. Den tester også om variablen ”testEnab” er falsk, hvis alle disse er sande i forhold til det if-statement de ligger i, vil den gå ind og teste, om der er nogle af knapperne ude i akserne der er blevet trykket ned. Hvis dette er tilfældet stoppes robotten, da dette er et tegn på at der er en akse der er kørt for langt i en retning den ikke skulle køre i. Herefter sættes variablen ”status” da denne viser, i visuen, at programmet køre.

Efter alt dette bliver der testet om blyanten skal spidses på variablen ”sharpen”. Hvis variablen er nået til 30000 vil den flytte blyanten over og spidse denne. Dette gør den ved først at flytte blyanten op af z-aksen. Hver gang den tegner sætter den variablen ”tempX” som bruges for at robotten kan huske, hvor den er noget til på papiret. Dette udnyttes i den næste del, hvor robotten bevæger sig hen af X-aksen mod blyantspidseren. Da blyantspidseren er placeret under nulpunktet ved robotten også, hvor den skal køre hen af x-aksen for at spidse blyanten. Nå den så er nået til nulpunktet bliver blyanten sænket ned i spidserne. Hvor langt blyanten bliver sænket ned, kommer an på, hvor lang blyanten er sat i ”penLength”. Her bruges formlen som giver antallet af steps den skal gå ned, plus en værdi der er sat til 1500 steps fra 0 på z-aksen. Denne værdi er sat ud fra at det er den længst mulige blyant der er vurderet til at kunne blive spidset. Når blyanten har været nede i spidseren i 6 sekunder, hvilket er bestemt af variablen ”timer” vil den blive taget op af spidseren og bevæge sig tilbage til der, hvor den stoppede.

Hvis variablen ”sharpen” ikke er nået til 30000, vil den begynde at tegne. Her bruger den så arrayet ”input”. For at holde styr på hvor den er nået til i arrayet bruges variablen ”placeInArray” hvor den så lægger, hvad der er på den plads i arrayet, ind i variablen ”placement”. ”placeInArray” ændres hver gang variablen ”i” når 30, hvilket den gør at når der er taget 15 steps på enten x- eller y-aksen hvilket sker mens den tegner. Dette gøres for at kunne bestemme størrelsen af de pixels der tegnes. Grunden til hver pixel kun er 15 step store selvom ”i” er 30, er fordi stepper motorerne reagerer på et højt signal, altså når variablen for step bliver boolsk lav, på grund af transistorende som invertere signalet. I programmet bruges ”i” kun sammen med variablerne ”stepX” og ”stepY” da der tegnes i 2 dimensioner.

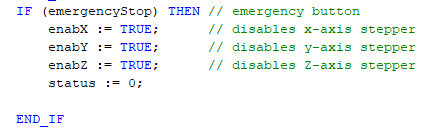
For at tegne pixels bruges ”U” og ”D” som i ascii er 85 og 68. ”U” betyder at der ikke skal tegnes, mens ”D” betyder at der skal tegnes. Så det første der sker er, at der bliver testet om blyanten er nede eller oppe. Hvis højden ikke stemmer overens med hvad den plads i array den er noget til siger den skal være, altså ”U” eller ”D” vil den først bevæge sig af z-aksen, enten op eller ned, hvorefter den vil bevæge sig af x-aksen for en pixel altså 15 steps.

Y-aksen bevæger sig kun når ”placement” er blevet ascii for ”N”, altså 78, hvilket betyder, at der skal laves en ny linje. Den starter med at z-aksen køre op til 2900 minus variablen ”ValueZ” (ValueZ er sat ud fra penLength igennem ligningen denne virker desværre ikke med alle længder så der skal findes en anden konstant.) efter blyanten er rykket op bevæger robotten sig tilbage af x-aksen ud fra variablerne ”newLineX” og ”newLineHelper” disse 2 variabler sørger for, at hver linje starter det samme sted. Dette gøres bl.a. ved at trække 17 steps fra ”newLineHelper”, hver gang der laves en ny linje da motorerne springer ca. 17 steps over ved hver ny linje derfor lægges der 17 til variablen ”posX” igen for at kompensere for motorerne. På grund af alt dette starter både ”newLineX” og ”newLineHelper” meget højt, så der ikke laves et overflow.

Nå den så er flyttet tilbage af x-aksen rykker den y-aksen 1 pixel altså 15 steps ved hjælp af ”i”.

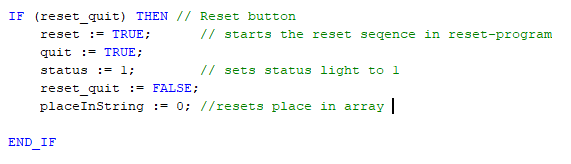
Emergency-program

Vi har valgt at lave et dedikeret program til robottens ‘Emergency Stop’. Vi har gjort dette, da vi vil være sikker på, at motorerne stopper, når man trykker på knappen. Ved denne løsning køres programmet hvert 100ms og dermed får vi en hurtig reaktion når der trykkes på knappen. Som det kan ses på billede 4, er programmet ret simpelt. Det eneste der sker er, at et if-statement spørger efter variablen ‘emergencyStop’, og hvis denne er true, så sættes enabX, enabY og enabZ til true, hvilket stopper stepper-motorerne.



Billede 4: Nødstop

Derudover findes der et stykke kode der styrer knappen ‘reset’. Hvis denne bliver trykket sættes variablen ‘reset’ til true. Dette gør, at programmet ‘Reset’ starter med at køre, og sætter robotten tilbage til startpositionen. Derudover sættes en række andre variable, som er nødvendige for at kunne starte reset-programmet.



### HMI - Human Machine Interface

Vi har valgt at lave et grafisk interface til at styre tegnerobotten. Vi har lavet dette fordi det giver et rigtigt godt overblik over de forskellige variable og på den måde kan vi følge med i, hvad robotten gør. Vi har valgt at lave tre forskellige sider, med hver deres funktion. På hver af disse sider findes der et ’Common layer’. Dette lægges som et lag ovenpå de enkelte sider. Her har vi placeret de vigtigste ting, såsom ’nødstop’, ’reset’ og ’start’. Derudover er der også en inputboks, hvori man indtaster den længe der stikker ud af blyantsholderen. Dette benyttes i koden i programmet ’Draw’. Til sidst findes der tre knapper, som fører hen til de sider, der findes i interfacet (Kan ses i Bilag 1).

Vi har lavet siderne ’Information’ og ’Konfiguration’ til fremtidige tilføjelser. Disse er tiltænkt, som de beskriver, forskellige informationer om position, variables værdier, evt. billedet den er ved at tegne osv. Derudover var ideen, at man kunne sætte nogle indstillinger i konfigurationen, om evt. blyanten man benytter, tiden der skal spidses osv. Disse sider er endnu blanke, men kan ses i Bilag 2 og 3.

Til sidst er der en side dedikeret til testning. Her kan der ændres på forskellige variable, som det kan ses i bilag 4. Derudover er der knapper, som hver kan kører robotten i forskellige retninger. Til sidst er der indsat variable for positionerne på de forskellige akser. Denne side har været rigtig brugbar gennem størstedelen af forløbet.

### Forbedringer

Da dette programmeringssprog er nyt for os alle har vi nogle ting, vi ikke har haft mulighed og tid til at få lavet. Disse ting er alle løsninger der hver løser mindre problemer eller gør tegningen af billedet mere effektiv.

Den ting, som vi ser som den største forbedring vi kan lave er, at få robotten til at tegne hurtigere. Som det ser ud nu, så tegner robotten en linje, hvorefter den kører tilbage til start og tegner endnu en streg. Derfor ville det være en fordel at få robotten til at tegne billedet både på vej frem og tilbage. Dette kan gøres på flere forskellige måder. Enten ved at ændre java-programmet, og måden det er sat op på. Derudover kan det også gøres i struktureret tekst, dog tænker vi, at det bliver meget sværere.

En anden væsentlig ting ved vores program er, at alle de billeder vi tegner bliver spejlvendte. Dette kan dog ændres ret nemt ved at sætte robotten til at starte i det modsatte hjørne. Vi har valgt ikke at gøre det i programmet, da vi mener det ikke betyder så meget. Dette problem kommer først rigtigt frem, hvis man tegner billeder med tekst på. Her vil teksten altså blive spejlvendt i både den ene og den anden retning.

Derudover overvejede vi metoder der kunne laves forskellige nuancer af grå, som ville gøre de billeder vi tegnede mere detaljerede. Dette kunne gøres ved forskellige grader af spidsning af blyanten. Derudover kan det også gøres ved at tegne ovenpå billedet flere gange, så man får mørkere grå de ønskede steder. Vi valgte ikke at lave dette, da det ville blive meget komplekst og desuden ikke var et krav.

Vi har også tænkt en del på, hvordan vi kunne udregne længden af blyanten. Problemet opstår efter der er blevet tegnet et stykke tid. Det er tid til at spidse blyanten, hvorefter blyanten bliver kortere. Dette vil påvirke længden blyanten skal sænkes ned mod papiret, men også længden der skal køres ned for at spidse. Vi har forsøgt at udregne forholdet mellem længden af blyanten og antallet af ’steps’ der skal køres ned til papiret.

Til sidst er der nogle småting, som kunne optimeres i programmet. Vi ville gerne have nogle flere elementer i interfacet. Her skulle man gerne kunne ændre forskellige indstiller og kunne se forskellige variables værdier. Derudover resetter vi de tre akser på samme tid. Da vi gerne vil undgå, at blyanten kolliderer med blyantspidseren eller tegne streger på ved op i startpositionen ville vi gerne starte med at køre z-aksen helt i nul-position.

TCP programmet kunne forberedes ved at kunne modtage flere pakker vi har dog ikke haft mulighed for dette på grund af manglede viden og information omkring TCP bibloteket.