Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2023 Åpen

Forfattere:
Erlend Meyer Sørflaten [256445]

Miriam Stensland [256804]

Daniel Tandstad Andersen [253666]

Oppgave av: Didrik Efjestad Fjereide

Veileder: Tormod Drengstig

Norsk tittel: Utvikling av maskinkontroller til CNC-fres

Engelsk tittel: Development of Machine Controller for CNC Milling

Studiepoeng: 20

Emneord: UiS, Elektroteknikk, CNC-fres, G-Kode, Motordriver, Maskinkontroller, Subtraktiv Produksjon	Sidetall: 73 Vedlegg/Annet: 40 Stavanger 15. mai 2023

Sammendrag

Motivasjonen for oppgaven var å lage en maskinkontroller til en CNC-maskin for å kunne frese ut deler ved bruk av datamaskinstøttet design.

Arbeidet i bacheloroppgaven har gått ut på å lage en maskinkontroller til en CNC-maskin. Rapporten innebærer fremgangsmåte på hvordan en maskinkontroller lages. Det innebærer design, kobling av ledninger og komponenter, endebrytere, stegmotorer og brukergrensesnitt. Vi sammenligner maskinkontrolleren og brukergrensesnitt med en åpen kilde-programvare GRBL. Den benyttes som maskinkontroller med UGS som brukergrensesnitt.

CNC-maskinen er satt sammen av blant annet 3D-printede deler, stegmotorer med drivere og elektronisk utstyr som nødstopp, aluminiumsprofiler, gjengestenger og plattform. De elektriske forbindelsene er organisert i et elektrisk panel for å holde det ryddig og oversiktlig. Maskinen styres med vår egen maskinkontroller og eget brukergrensesnitt. Ved å kjøre brukergrensesnittet fra Raspberry Pi med Arduino Uno via seriekommunikasjon, kan vi oversette G-kode til motorkommandoer. G-koden produseres i Fusion 360.

Resultatet er en forenklet maskinkontroller inspirert av GRBL. Maskinkontrolleren kan bevege et verktøy fritt innenfor et arbeidsområde. Resultatet er at vi kan bruke maskinkontrolleren til å tegne alle typer 2D-figurer som sirkler, diagonaler og firkanter på papir. Her er en link til en video, hvor vi sammenligner resultatet våres med GRBL og UGS.

Link til introduksjonsvideo: YouTube - UiS B.Sc: Utvikling av CNC Fres
Link til filer brukt i oppgaven: Google Drive - CNC Fres

Forord

Forfatterene av rapporten er Miriam Stensland, Erlend Meyer Sørflaten og Daniel Tanstad Andersen. Vi studerer Bachelor i Automatisering og elektronikkdesign ved instituttet for Data- og Elektroteknikk (IDE) på Universitet i Stavanger (UiS).

Vi har jobbet sammen for å utvikle en maskinkontroller og bygge en CNC-maskin. Det innebærer montering av maskinen, design og installasjon av el-tavle, utvikling og programmering av maskinkontrolleren i Arduino IDE [3], brukergrensesnitt i Python [17] og seriekommunikasjon.

I forbindelse med oppgaven, arbeidet og resultatet vil vi gi en takk til:

- **Tormod Drengstig**
  for god oppfølging, inspill, utstyr og veiledning.
- **Didrik Efjestad Fjereide**
  for oppgaven, god oppfølging, design, veiledning, printing av deler til CNC-maskinen og annet laboratorieutstyr.
- **Romuald Karol Bernacki**
  for lån av laboratorieutstyr, deler til maskin og teknisk hjelp.

Innhold

Forkortelser

AMASS Adaptive multi-axis smoothing system
CAD Computer-aided design
CAM Computer-aided manufacturing
CCW Counterclockwise
CNC Computerized numerical control
CPU Central processing input
CW Clockwise
EMI Elektromagnetisk interferens
G-kode Geometrisk kode
GUI Graphical user interface
GND Ground
IDE Integrated development environment
ISR Interupt Service Routine
NC Numerical control
PCB Printed circuit board
PSU Power Supply Unit
RPM Revolutions per minute
USART Universal synchronous/asynchronous receiver/transmitter
USB Universal Serial bus
UGS Universal G-code Sender
VCC Voltage Common Collector
V DC Voltage Direct Current

Kapittel 1

Introduksjon

I denne oppgaven tar vi for oss design, bygging og testing av en egenbygget CNC-maskin, samt programmering av maskinkontroller. Vi tar med de nødvendige komponentene for å sette sammen maskinen, samt tilhørende program som trengs for å kjøre maskinen. Det omfatter blant annet stegmotorer med drivere, maskinvare og maskinkontroller. I tillegg innebærer det en evaluering av det ferdige resultatet sammenlignet med eksisterende programvare GRBL. Gjennom oppgaven får vi en forståelse av design, bygging, testing og andre utfordringer involvert i å lage en CNC-maskin fra bunnen av.

Link til introduksjonsvideo:
YouTube - UiS B.Sc: Utvikling av CNC Fres
Filer brukt i oppgaven er tilgjengelig i linken under eller som vedlegg:
Google Drive - CNC Fres

Introduksjonen er inspirert av CHAT GPT, se figur H.1.

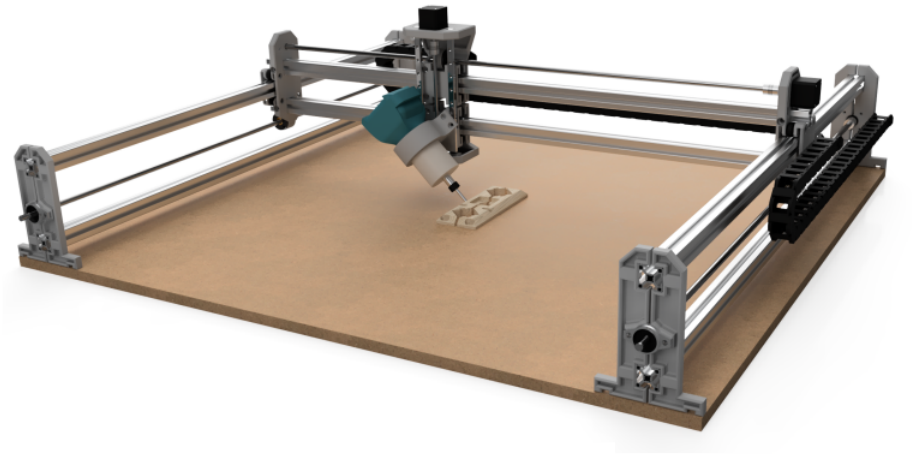
1.1 Oppgavebeskrivelse

Denne bacheloroppaven, med tittelen «Utvikling av maskinkontroller for CNC fres», er utført av Daniel Tandstad Andersen, Erlend Meyer Sørflaten og Miriam Stensland, og omfatter følgende punkter hentet fra oppgavebeskrivelsen:

- Teoretisk forståelse: *Utforske teorien bak CNC-maskinen og dens operasjonelle prinsipper.*
- Montering av maskinen: *Gjennomgang av komponenter og monteringsprosessen for CNC-maskinen.*
- Design og installasjon av el-tavle: *Utforme og installere el-tavle samt generere el-tavle skjema.*
- Programmering av maskinkontrolleren: *Utvikle en maskinkontroller ved bruk av Python og C++ for oversetting av G-kode til motorkommandoer, styring av stegmotorer, innhenting av sensordata, sikkerhetsfunksjoner og brukergrensesnitt. I tillegg testing og implementering av allerede eksisterende maskinkontroller GRBL med UGS som brukergrensesnitt.*

Prosjektet kombinerer teori, praktisk arbeid, programmering og installasjon for å utvikle en maskinkontroller til en CNC-maskin. Målet er å oppnå bedre forståelse for maskinens drift og utforske mulighetene med en maskinkontroller. Resultatene vil bidra til en dypere forståelse i utvikling av maskinkontroller-teknologi og dens anvendelse.

Prosjektoppgaven er gitt av Didrik Efjestad Fjereide og Tormod Drengstig.



Figur 1.1: Figur av designet CNC-maskin, hentet fra oppgavebeskrivelsen.

1.2 Om prosjektgruppen

Vi er en gruppe på tre studenter innenfor ingeniørveien Automatisering og elektronikkdesign. Vi startet på studieprogrammet i 2020 og fullfører bachelor våren 2023.

1.3 Rapportens struktur

Oversikt over kapitlene:

- Kapittel 1: Introduksjon
- Kapittel 2: Beskrivelse av CNC-maskinen og utstyr
- Kapittel 3: Porgramvare
- Kapittel 4: GRBL og UGS
- Kapittel 5: Resultat og sammenligning
- Kapittel 6: Konklusjon

1.4 Om CNC-maskiner

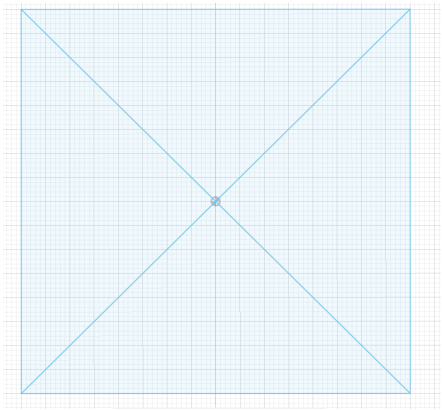
En verktøymaskin er en betegnelse på en maskin som bearbeider materialer. Eksempler på verktøymaskiner er dreiebenk, fresemaskin og slipemaskin. CNC-maskin brukes blant annet til å skjære ut og produsere deler av ulike materialer som for eksempel metall, tre eller plast.

CNC-styring er forkortelse på *computerized numerical control*-styring, som på norsk er «*datamaskinbasert numerisk styring*». Styringen dreier seg om å bruke datamaskin til å automatisere maskineringsprosesser. Hensikten med å automatisere arbeidsoppgaver er blant annet å erstatte eller avlaste menneskelig arbeidskraft. I tillegg er resultatet mer nøyaktig, repeterbar og gjøres raskere enn menneskelig arbeidskraft.

Prosessen innebærer å bruke motorer til å bevege et verktøy som freser ut deler. CNC-styring opererer et maskinverktøy ved hjelp av kodede instruksjoner i maskinkontrollsystemet. Instruksjonene er kombinasjoner av tall, bokstaver og symboler som gis i en sekvensiell og logisk rekkefølge, se kode (1.2a) og figur (1.2b) for hvordan tegningen ser ut. Hva instruksjonene betyr kommer vi til senere i rapporten. Instruksjonene er samlet i et CNC-program (.nc fil).

```
G92
G90
M17
G0 Z20 F2000
G0 X0 Y0 F5000
G0 Z0 F2000
G0 X40 Y0 F5000
G0 X40 Y-40
G0 X0 Y-40
G0 X0 Y0
G0 X40 Y-40
G0 Z20 F2000
G0 X40 Y0
G0 Z0 F2000
G0 X0 Y-40
G0 Z20 F2000
G0 X0 Y0 F4000
Z0 F2000
```

(a) G-kode av kvadrat (80 mm x 80 mm) med to diagonaler i kryss.



(b) Kvadrat med diagonaler i Fusion 360.

Figur 1.2: Eksempel på G-kode produsert i Fusion 360.

CNC-maskiner bruker intern mikroprosessor med minne-registre som lagrer rutiner som operatøren kan endre. Instruksjonene er lagret som programvare-instruksjoner som betyr at de logiske funksjonene kan endres av operatøren. Dette gjør CNC-maskinen allsidig.

Selve CNC-maskinen er sammensatt og består av datamaskin, aktuatorer og sensorer som samkjører om å følge en forhåndslaget vei i geometrisk kode (G-kode). G-koden inneholder instruksjoner om fart, bevegelseslengde og hvilken type bevegelse som skal gjøres.

1.4.1 Historie om CNC-styring

Hullkort og hullbånd oppsto på 1700-tallet originalt til bruk for automatisk veving. De ble brukt til å registrere data på ved å slå hull på kortet eller båndet. Informasjonen ble overført fra kortene eller båndene til datamaskinens minne ved elektromekanisk eller fotoelektrisk avlesning. Metodene ble

mye brukt på 1960- og 70-tallet for blant annet å lese inn informasjon på datamaskiner, i form av *numerical control*-styring, forkortet til NC-styring. Disse metodene var langsomme ettersom det kun var 500-1500 tegn som ble lest hvert sekund.

Etterhvert ble det fra 60-tallet av vanlig å bruke CNC-styring istedenfor NC-styring. Det ble mer populært på 70-tallet, og enda mer på 80-tallet ved introduksjon av rimelige mikrodatamaskiner. Etterhvert ble hullkort og hullbånd helt erstattet av CNC-styring. I dag er innlesnings-hastigheten opptil flere millioner tegn i sekundet for blant annet DVD-er. Hullkort og hullbånd er derfor i dag erstattet av magnetiske og optiske datalagringsmedia, som er innebygd elektronisk lagring i datamaskinen [9].

CNC-styring er i dag svært betydelig i produksjonssektoren og for metallarbeid i industrien.

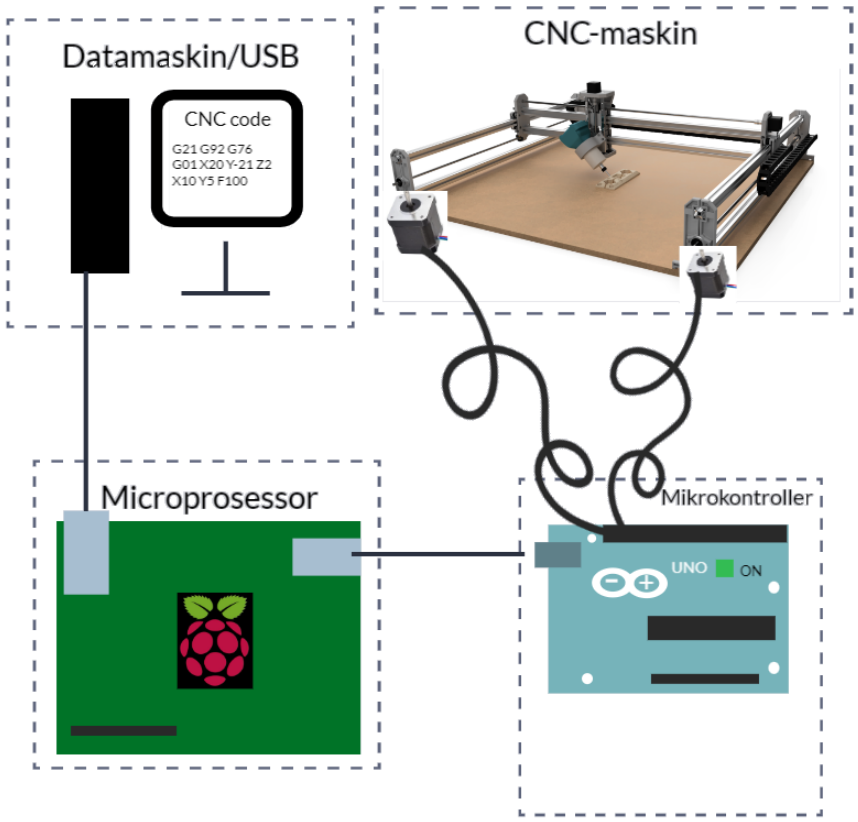
1.5 Problemstilling

Dette prosjektet omhandler CNC-maskinen vist i figur (1.1). CNC-maskinen har fire frihetsgrader: lineærbevegelse i X-, Y- og Z-retning (translasjon), samt rotasjonsakse om Y-aksen.

Arbeidsområdet er 1000x100x100 mm og kan justeres. Målet med oppgaven er at bruker designer et produkt i et *computer aided*-program, forkortet CAD, som for eksempel *Autodesk Fusion 360*. CNC-maskinen skal deretter skjære bort materiale slik at det ferdige arbeidsstykket er igjen.

For å gjennomføre oppgaven trengs det en maskinkontroller for CNC-maskinen. Arbeidet består av kontrollering av stegmotorer, programmering, koblinger, el-tavle, endebrytere, generere G-kode og oversetting av G-kode til motorkommandoer.

Figur (1.3) beskriver i enkle detaljer hvordan stegmotorene styres av mikrokontrolleren Arduino Uno, som kommuniserer med datamaskinen Raspberry Pi.



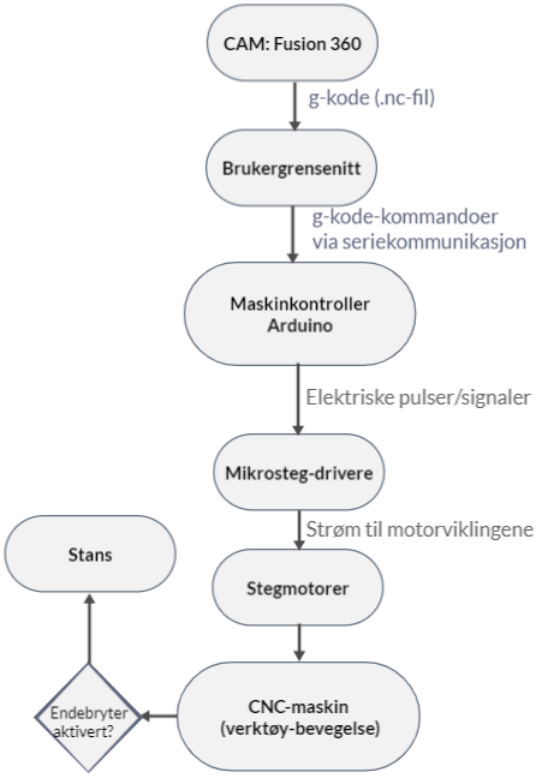
Figur 1.3: Overordnet blokkskjema av systemet. Det er trådløs kommunikasjon mellom datamaskinen og Raspberry Pi.

For å oppnå et godt resultat av CNC-styringen er det viktig å ta hensyn til verktøy-informasjon, material-spesifikasjoner (datablad), maskinsekvensen, arbeidsskisse og utregninger. I tillegg er det viktig å ha en god fremgangsmåte for programmeringen.

Det er også viktig å ta høyde for materialets størrelse, type, fasong, kvalitet, tilstand og hardhet. Målet er å bruke informasjonen for å etablere en effektive og produktiv måte å frese ut et produkt på. For denne CNC-maskinen er det foreløpig ikke benyttet freseverktøy eller material, men det er brukt penn og papir.

Figur (1.4) viser flytskjema over hvordan oppgaven er løst, fra brukerens

produktdesign (G-kode) til CNC-maskinens resultat (produkt).



Figur 1.4: Flytskjema over prosessen.

1.6 Tilgjengelig programvare

I forbindelsen med å lage maskinkontrolleren brukes en tilgjengelig programvare GRBL som inspirasjon. GRBL er et åpent kildeprogram laget og utviklet over flere år, og brukes som inspirasjonen til utvikling av maskinkontrolleren. I tillegg er programmeringen av eget brukergrensesnitt inspirert av et ferdig brukergrensesnitt *Universal G-Code Sender*, forkortet UGS.

Kapittel 2

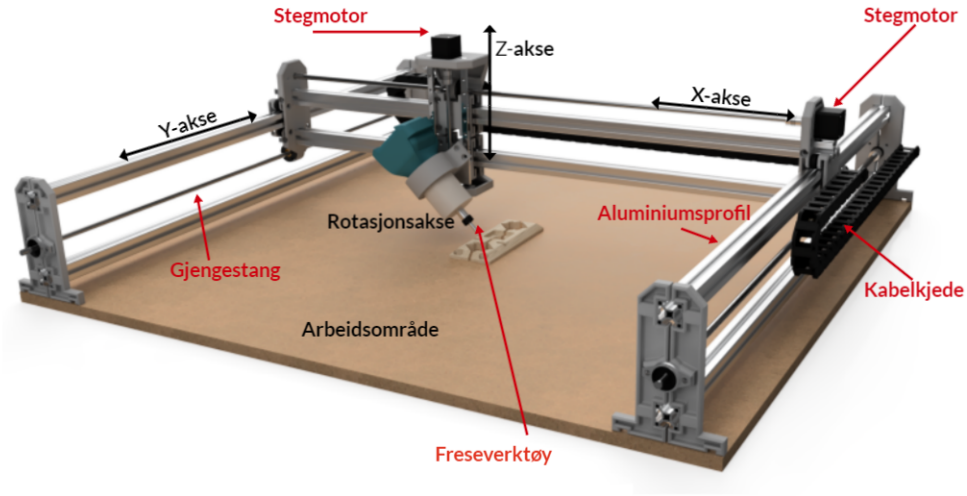
Beskrivelse av CNC-maskinen og utstyr

Dette kapittelet gir en beskrivelse av utstyr, komponenter i CNC-maskinen. Formålet med kapittelet er å klargjøre hvordan teori og komponenter henger sammen for å skape en fungerende CNC-maskin. Vi ønsker at leseren skal få en grundig forståelse av det utstyret som brukes og hvordan det fungerer for å produsere deler.

2.1 Maskinens struktur

Fra oppgavebeskrivelsen har vi fått et designutkast av maskinen, se figur (2.1). Figuren gir en oversikt over sentrale deler til CNC-maskinen. Det endelige resultatet av CNC-maskinen er vist i figur (2.2) og inneholder flere komponenter som er lagt til underveis i arbeidet.

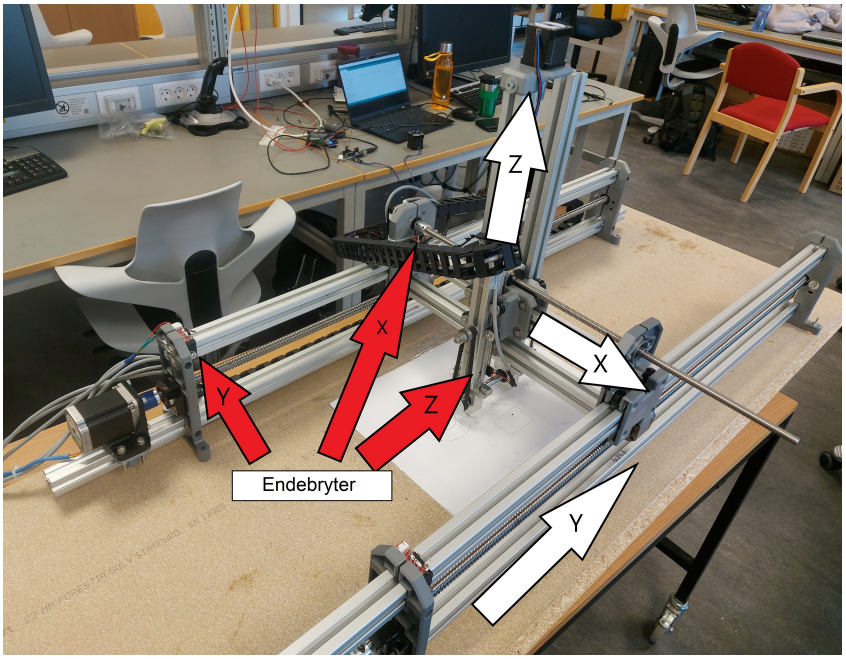
2.1 Maskinens struktur



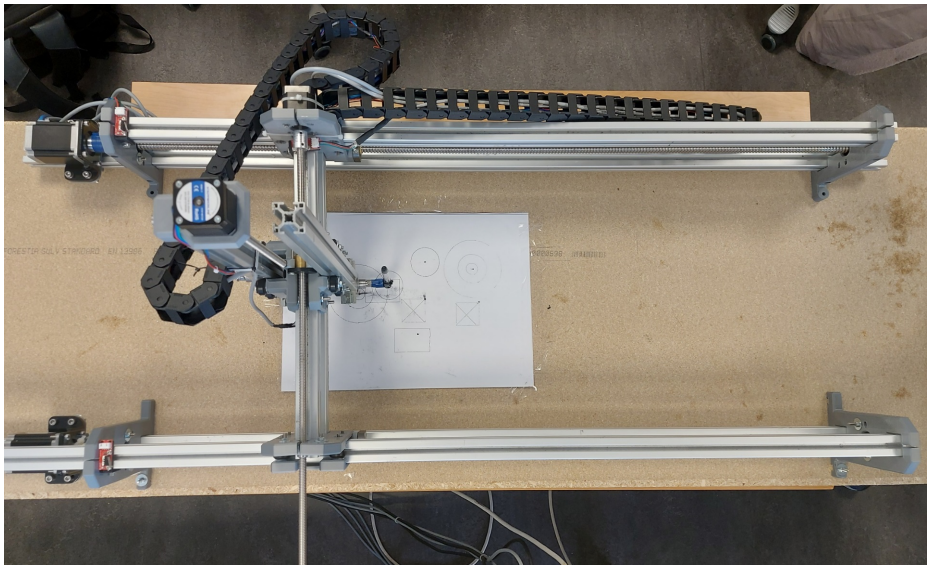
Figur 2.1: CNC-maskinen med noen sentrale komponenter. Det er to synlige stegmotorer på bildet, men det er totalt fire: to på Y-aksen, en på X-aksen og en på Z-aksen.

CNC-maskinen sin hovedstruktur er satt sammen av aluminiumsprofiler, gjengestenger og 3D-printede støttedeler festet på en sponplate. Vi opplevde problemer med ustabilitet og anbefaler derfor å finne en mer stabil måte å montere maskinen på. En forbedringsmulighet er å erstatte de 3D-printede plastdelene med aluminiumsprofiler for bedre stabilitet. Ved å implementere forbedringen blir CNC-maskinen mer robust og stabil. Forbedringen bidrar til å fjerne den uønskede vibrasjonen og dermed forbedre nøyaktigheten under kjøring av maskinen.

Figurene (2.2) og (2.3) viser resultatet av CNC-maskinen.



Figur 2.2: CNC-maskinen som ble bygget i oppgaven, inkludert endebrytere.

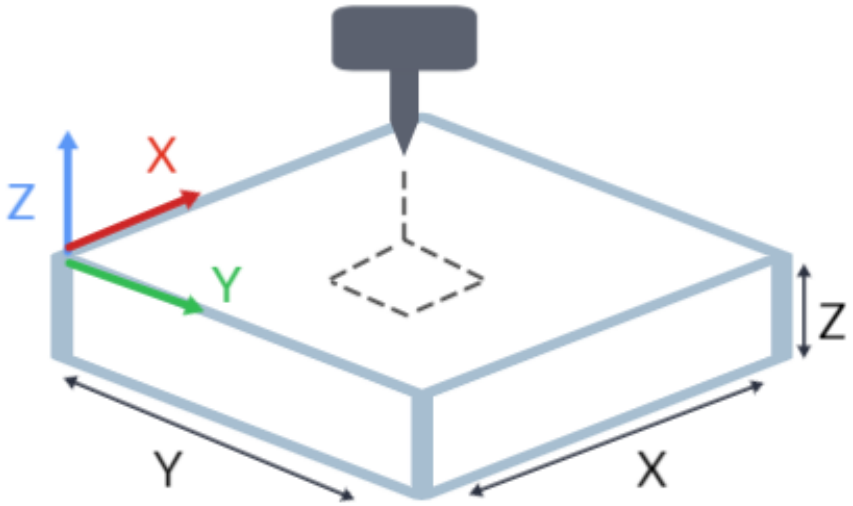


Figur 2.3: CNC-maskinen sett ovenfra.

Planen var å ha et arbeidsområde på 1000x1000x100 mm (lengde, bredde, høyde). Vi benyttet en sponplate som underlag. Arbeidsområdet ble justert etter sponplaten. Lengden og bredden på arbeidsområdet ble 1000x400 mm. Høyden ble av praktiske grunner justert til 60 mm. Hele arbeidsområdet er derfor 1000x400x60 mm.

2.2 Akser og koordinatsystem

CNC-maksinen beveger verktøyet langs X-, Y- og Z-aksene, se figur (2.4). Dersom verktøyet ikke beveger seg nøyaktig langs aksene, fører det til feilproduksjon. Kunnskap og forståelse om akser og koordinatsystemet er viktig for å utvikle maskinkontrolleren (programmet) som styrer stegmotorene. Koordinatsystemet fungerer som et hjelpemiddel for å vite hvor verktøyholderen er.

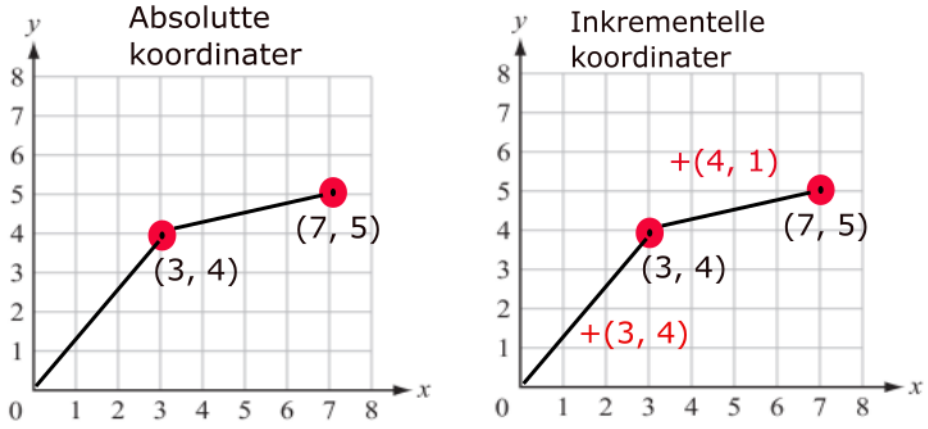


Figur 2.4: Verktøyet beveger seg langs X, Y og Z-aksene.

Utviklingen av CNC-maskinen inneholder generelle matematiske prinsipper, hvor de mest relevante er trigonometri, geomteri og lineær algebra. Ved programmeringen av maskinkontrolleren representerer punktene i koordinatsystemet steder på arbeidsområdet. Informasjonen som finnes i en 2D- eller 3D-figur oversettes til G-kode ved bruk av koordinater og funksjoner. G-kode blir forklart senere.

2.3 Referansepunkter og maskinens nullpunkt

Det finnes generelt to prinsipper for å definere referansepunkt for CNC-maskinen: absolutte eller inkrementelle koordinater. Inkrementell koordinater beskriver en posisjon relativt til nåværende posisjon. Det betyr at maskinens nåværende posisjon brukes for å angi neste posisjon, se figur (2.5).



Figur 2.5: Absolutte og inkrementelle koordinater.

Dersom verktøyet beveger seg fra punktet origo (0, 0) til (3, 4) og deretter (7, 5) kan det beregnes på en av følgende måter:

- Absolutte koordinater: Tar utgangspunkt i et fast referansepunkt origo og legger inn koordinatet som verktøyet skal gå til.

- Inkrementelle koordinater: Referansepunkt endres til nåværende posisjon. Det vil si at (3, 4) vil være referansepunkt for bevegelse til (7, 5), og derfor må det legges til +4 på X-aksen og +1 på Y-aksen for å nå dette punktet: (3, 5) + (4, 1) = (7, 5).

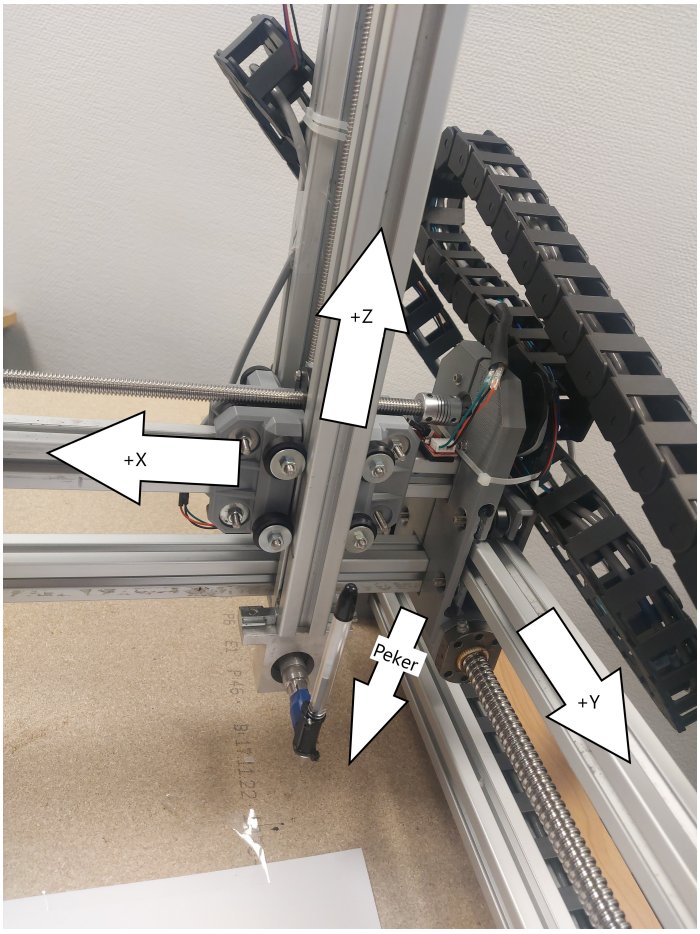
I programmet til maskinkontrolleren er det mulig å bytte mellom inkrementelle og absolutte koordinater ved hjelp av kommandoene «G90» og «G91». Se tabell (2.1) for oversikt over G-kode kommandoene.

Tabell 2.1: Disse G-kode kommandoene benyttes i programmet.

G-kode	Beskrivelse
G0	Rask posisjonering
G1	Lineær bevegelse med spesifikk fart (F)
G2	Bevegelse i bue med spesifikk fart (F)
G3	Bevegelse i bue med spesifikk fart (F)
G90	Absolutt distanse modus
G91	Inkrementell distanse modus
G92	Sett referansepunkt
M17	Aktivere motorer
M18	Deaktivere motorer
M100	Hjelp
M114	Posisjon til verktøyet
M120	Hjemsyklus

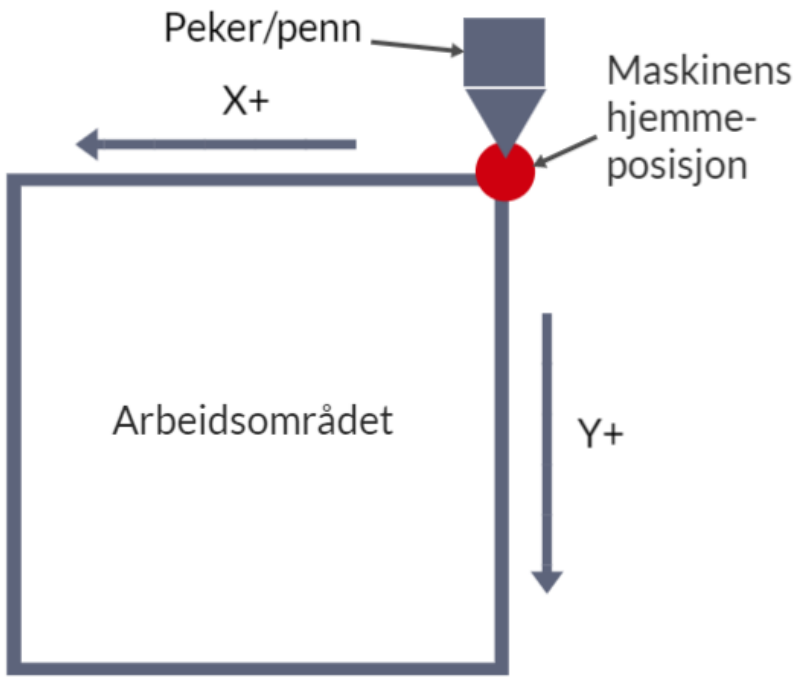
I dette prosjektet benyttes en penn som verktøy, også kalt peker. Tuppen på pekeren er definert som arbeidspunktet, også kalt verktøykoordinatet. Bevegelsen av verktøyet foregår på de positive aksene av koordinatsystemet. Når CNC-maskinen starter er verktøykoordinatet et logisk nullpunkt. Deretter beveger CNC-maskinen verktøyet med referanse til dette logiske nullpunktet.

I figur (2.6) vises translasjonene X, Y og Z. Stegmotorene roterer gjengestengene slik at verktøyet beveger seg langs X-, Y- og Z-aksene (3-akse fresing). Bevegelsene er lineære hvor de kombinert sammen kan generere ulineære bevegelser i form av for eksempel sirkler. En mer avansert form for fresing er fem-akse fresing som lager mer komplekse og avanserte bevegelser. Det er da mulig å bevege maskinen i mer enn tre frihetsgrader.



Figur 2.6: Positive X-, Y- og Z-koordinater. Tuppen av pekeren er vendt nedover mot arbeidsområdet.

Maskinens hjemmeposisjon er punktet hvor X, Y og Z er null (origo). Posisjonen er oppe i høyre hjørnet og kalles også for nullposisjon, se figur (2.7). Hensikten med hjemmeposisjonen er å sikre at verktøyet ikke beveges utenfor arbeidsområdet. Vi vet lengden på aksene, og ved å starte i hjemmeposisjonen har programmet kontroll over hvor verktøyet befinner seg. Det sørger også for at bevegelsen ikke går lengre enn mulig. Hjemmeposisjonen er også hvor automatisk verktøy-forandring skjer. I denne maskinens tilfelle er det kun ett verktøy, og et eventuelt bytte må skje manuelt.



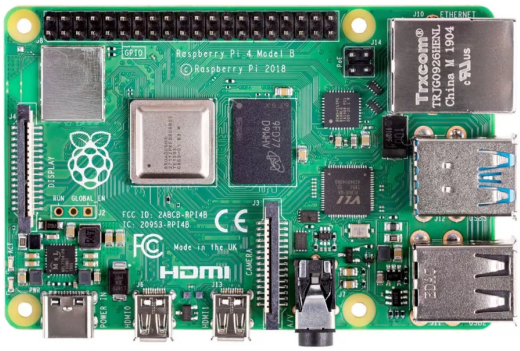
Figur 2.7: Oversikt over arbeidsområdet i 2D (X, Y) med maskinens hjemmeposisjon. Hjemmeposisjonen er den røde prikken og er felles endepunkt for aksene. Dette er posisjonen til verktøyet etter hjemsyklusen er ferdig.

Hjemmeposisjonen er en del av maskinens fysiske oppbygning, og kalles derfor for et «fast» punkt. Maskinen returnerer til denne posisjonen automatisk for å kalibrere posisjonen til verktøyet. Vi benytter oss av hjemsyklus (eng: *homing*) ved hjelp av fysiske endebrytere. Etter hjemsyklus er utført er verktøyet ved hjemmeposisjonen.

Logisk nullpunkt er et justerbart referansepunkt. De justerbare referansepunktene kalles for «fleksible» eller «flytende» punkt. Etter at referansepunktet er satt, kjøres G-kode fra dette punktet. Logisk nullpunkt er justerbart, men maskinens nullposisjon er fast.

2.4 Raspberry Pi

Raspberry Pi er en ettkortsdatamaskin. Det betyr at alle hovedkomponentene som prosessor, minne og tilkoblingsmuligheter er samlet på ett enkelt kretskort. Vi benytter Raspberry Pi 4 Model B for å trådløst styre maskinkontrolleren. Via brukergrensesnittet i Raspberry Pi brukes seriekommunikasjon (USB) med Arduino Uno. Arduino sender signaler til mikrostegdriverene som styrer stegmotorene.

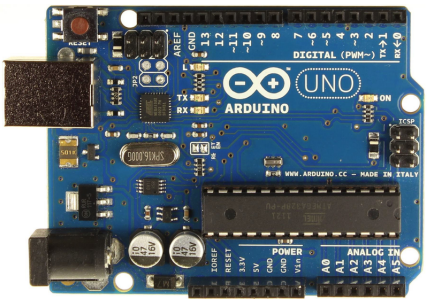


Figur 2.8: Raspberry Pi 4 Model B [18].

Det er mulig å bruke pinnene på Raspberry Pi til å styre stegmotorene. Det er ikke like effektivt som Arduino. Raspberry Pi er egnet til applikasjoner som krever mer kompleks databehandling, mens Arduino er spesialisert på sanntidskjøring. Sanntidskjøring er avgjørende i dette prosjektet, da det kreves rask respons på sanntidsdata fra sensorer.

2.5 Arduino Uno

Arduino Uno er en åpen maskinvare mikrokontroller-plattform. Den består av et kretskort med innebygd mikrokontroller og de nødvendige maskinvarekomponentene for å bruke mikrokontrolleren. Arduino er en plattform for enkle mikrokontroller oppgaver.



Figur 2.9: Arduino Uno [5].

I denne oppgaven benyttes Arduino til å styre stegmotorene ved å lage en maskinkontroller vi har utviklet selv. I tillegg tester vi også ut med å installere en ferdiglagd G-kode tolk «GRBL» på kortet. GRBL er laget for å være på en 8-biters Atmel ATMEGA328 mikrokontroller, som er tilgjengelig på Arduino Uno. Arduino Uno er egnet til å styre stegmotorene til CNC-maskinen. Med Arduino Uno kan man programmere maskinen til å utføre komplekse oppgaver.

2.6 Stegmotor

Stegmotorene som brukes i prosjektet er bipolar synkron børsteløs likestrømsmotorer, som presist styres til ønsket posisjon med liten marginfeil. En stegmotor skiller seg fra andre elektriske motorer ved at den ikke roterer kontinuerlig, men heller beveger seg i diskrete trinn.

Stegmotorer blir ofte brukt i systemer som krever presis posisjonering og bevegelseskontroll. Presisjonen avhenger av applikasjon og krav til nøyaktighet. Stegmotor velges når kostnad er en viktig faktor og ved lav til moderat hastighet. På grunn av kostnad er stegmotor valgt.

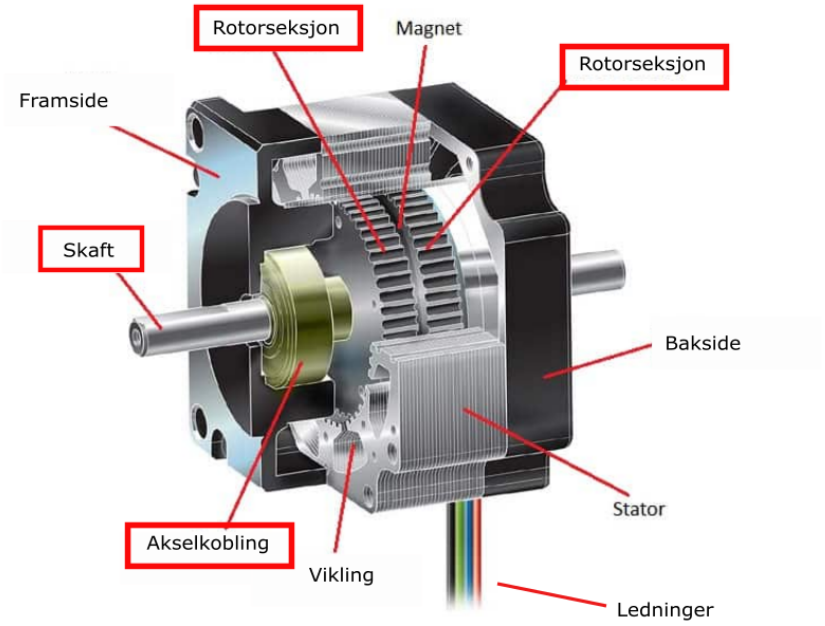
Alternativet til en stegmotor er servomotor. De er bedre ved høy ytelse og når høy presisjon er nødvendig. Servomotorer har raskere respons og bedre akselerasjonsegenskaper enn stegmotorer. Stegmotorer er ikke 100% effektive ettersom noe av den elektriske kraften blir omgjort til varmeenergi. Ved lange driftsperioder kan varmen påvirke ytelsen til stegmotoren. Om varmegenereringen utgjør et problem eller ikke avhenger av omgivelsene der

motoren opererer. Servomotorer er designet med bedre varmehåndtering og er derfor bedre egnet til lang kontinuerlig drift.

Et forbedringsforslag er derfor å oppgradere til servomotorer dersom CNC-maskinen skal drives over lengre tid. I tillegg dersom CNC-maskinen skal utvikles videre og det stilles høyere krav til presisjon.

2.6.1 Hvordan fungerer en stegmotor?

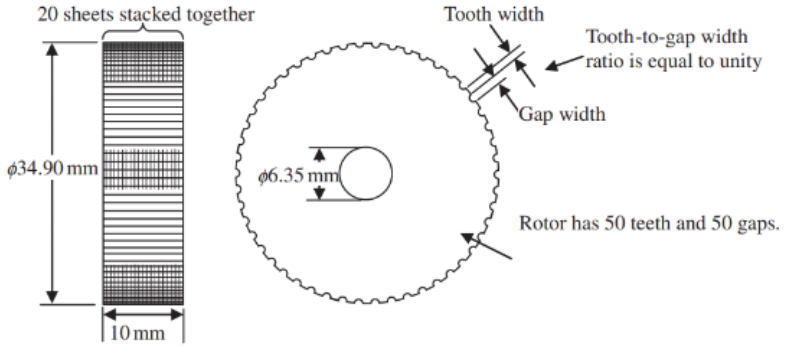
En stegmotor er en elektrisk motor som går i trinn eller «steg». Når motoren mottar elektriske signal fra en styringsenhet roterer rotoren. Motoren er bygget opp av flere komponenter, se figur (2.10).



Figur 2.10: Bildet viser oppbyggingen til en stegmotor, og er hentet fra ISL Products [12]. Figuren inneholder mange komponenter, men de viktigste for oppgaven er rotor, skaft og akselkobling.

Komponentene som er viktige å ha kunnskap om for å styre stegmotorene er rotor, akselkobling og skaft. En gjengestang kobles til rotoren, og ved hjelp av akselkobling kan stangen rotere med rotoren. Rotasjonen av gjengestangen gjør at verktøyet beveger seg i arbeidsområdet.

Rotoren omformer elektrisk kraft til mekanisk kraft. Rotoren befinner seg i midten av stegmotoren og består av tenner (steg), se figur (2.11). Den roterer alt etter hvilket ledningspar det sendes strøm gjennom. Det betyr i praksis at magnetfeltet skifter polaritet slik at rotorretningen snur.



Figur 2.11: Figuren er hentet fra boken «Creating Precision Robots» [14]. Bildet viser hvordan en 50-tenners rotor er satt sammen av 20 deler med permanentmagnetiske ark.

Rotoren er plassert inni statoren. Det er den magnetiske kraften mellom stator og rotor som produserer et dreiemoment på rotoren. Når strøm veksles på å sendes gjennom ledningsparene i statorviklingene, vil magnetiseringen i stator endre seg, og rotoren vil bevege seg. Dersom strømmen sendes sekvensielt i ledningsparene vil stegmotoren rotere.

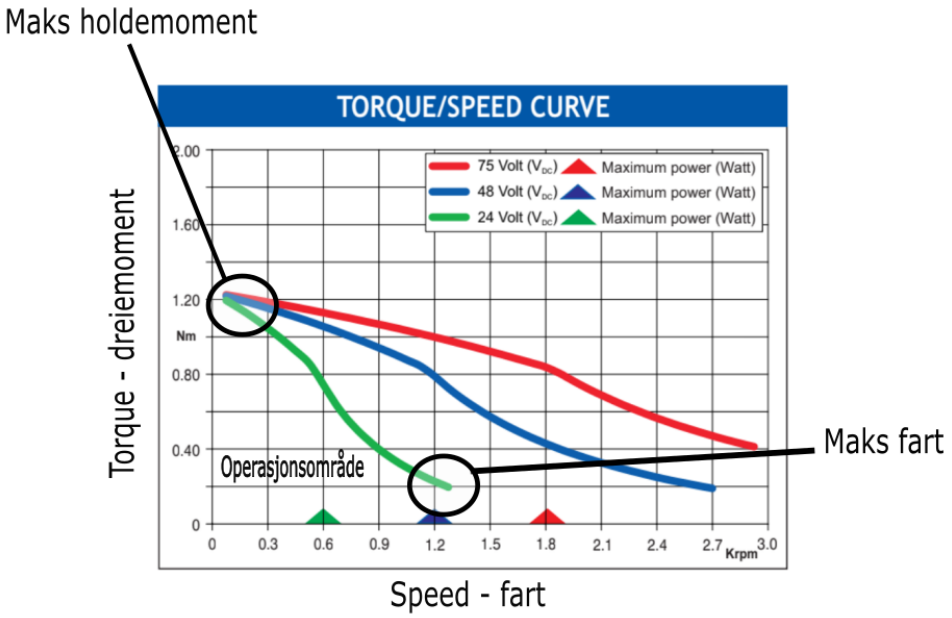
2.6.2 Parametere til en stegmotor

For CNC-maskinen brukes det ulike steg-motorer for X-, Y- og Z-aksene. Det er fordi det stilles ulike krav til moment, hastighet og nøyaktighet. Når stegmotor velges, ser man på spenning, strøm og maks holdemoment. Parametrene er oppgitt i motorens datablad.

- **Steg/vinkel**: Hvor mange grader ett steg tilsvarer. Vanligvis har en stegmotor 200 steg per rotasjon, altså 1.8° for hvert steg. Dette tilsvarer en full rotasjon på 360°. Antall steg per rotasjon kan justeres ved hjelp av en mikrosteg-driver. Det gir høyere presisjon og økt kompleksitet. Steg/vinkel er viktig for å regne ut hvor mange millimeter verktøyet beveger seg i arbeidsområdet (mm/steg).
- **Strømforbruk**: Avhenger av motorens design og konstruksjon. Et høyt strømforbruk indikerer at motoren har et høyt holdemoment (eng: *holding torque*).
- **Holdemoment**: Mengden dreiemoment som kreves for å rotere et fullt steg når viklingene i en motor er aktivert, men rotor er stasjonær. Dreiemoment er en måleenhet for kraften som trengs for å rotere en gjenstand. Høyt holdemoment gjør at motoren beveger seg mer detaljert og nøyaktig ved å overvinne tregheten til lasten. Dette gjør at motoren kan holde en tyngre last og motsette seg eksterne krefter. Det resulterer i presis posisjonering som er viktig for CNC-maskiner.

En ulempe med stegmotoren er dreiemomentets begrensning ved høy fart. Maks dreiemoment oppnås ved lav hastighet, som er nyttig når tung last, som for eksempel verktøy, skal håndteres. Det er ikke kritisk for denne CNC-maskinen ettersom vi bruker en penn for testing.

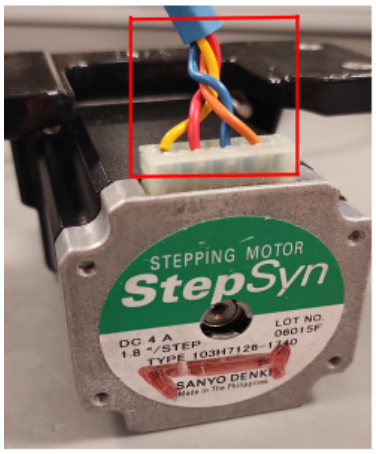
Sammenhengen mellom dreiemoment og hastighet hentes fra databladet til stegmotoren, og presenteres i en dreiemoment/farts-kurve, se figur (2.12). Kurven er en referanse på hva stegmotoren kan oppnå, men parametrene varierer og avhenger av kombinasjonen av motor og mikrosteg-driver.



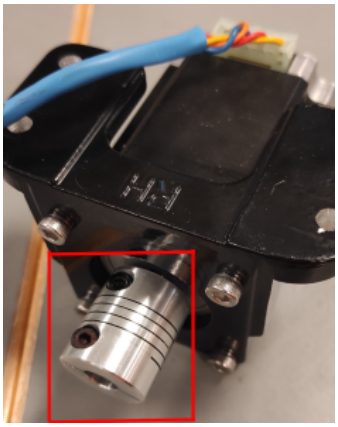
Figur 2.12: Grafen er hentet fra databladet til Y-stegmotor [19]. Den grønne grafen er relevant siden vi bruker 24 V.

2.6.3 Sammenligning av parametrene

Y-stegmotor er vist i figurene nedenfor. Figur (2.13) viser motorens fire ledere og figur (2.14) viser skaftet som er festet på rotor.



Figur 2.13: Y-stegmotor med ledningene.

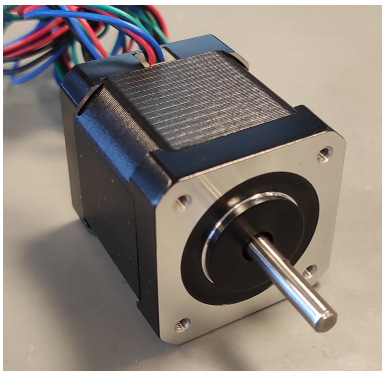


Figur 2.14: Skaftet mellom stegmotoren og gjengetestang.

I følge databladet har Y-stegmotor 1.8° per steg, med usikkerhet på 0.09° per steg [19]. Det betyr at 200 steg tilsvarer en rotasjon på 360°. Motorens maks strømtrekk er 4 A. I databladet er det oppgitt hvilke ledere som er koblet sammen. Oransje og blå, rød og gul. De fire ledningene omtales som de to ledningsparene, og er sentrale i korrekt styring av stegmotoren. Dette forklares senere i oppgaven.

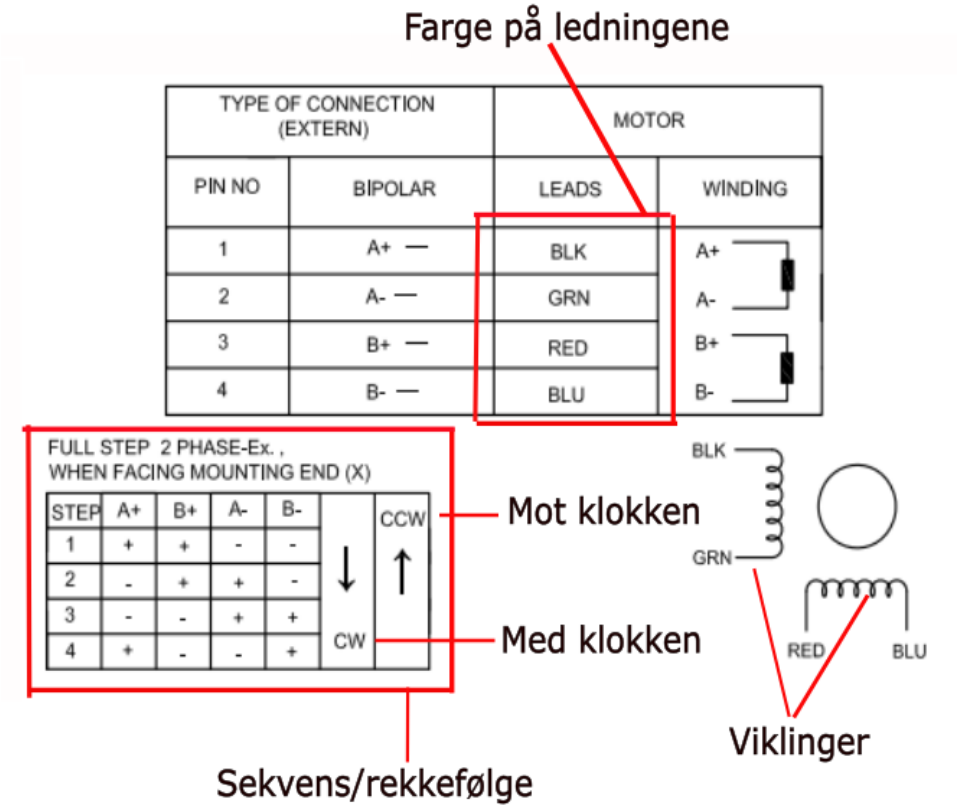
X- og Z-stegmotor vises i figurene nedenfor.

Figur 2.15: Ledningene til X- og Z-stegmotor.



Figur 2.16: Skaftet mellom stegmotoren og gjengetestang.

Figur (2.15) viser baksiden av motoren og de fire ledningene. Figur (2.16) viser skaftet til motoren, som er en del mindre enn skaftet til Y-stegmotoren. I følge databladet har X- og Z-stegmotoren en steglengde 1.8° og 200 steg per rotasjon [20]. Usikkerheten er 5% av steglengden, som tilsvarer: 1.8° · 0.05 = 0.09°. Motorens maks strømtrekk er 2 A. I figur (2.17) vises viklingene og hvordan de henger sammen.



Figur 2.17: Informasjon om X- og Z-stegmotor er hentet fra databladet [20]. Viklingene viser hvilke ledninger som er koblet sammen, svart og grønn, rød og blå. I tillegg viser databladet hvilken sekvense ledningene må drives i for å gjøre et fullt steg. Ved å endre på rekkefølgen skiftes det mellom to retninger: rotering med (CW) eller mot klokken (CCW).

I tabell (2.2) er det oversikt over parametrene.

Tabell 2.2: Oversikt over sentrale parametre for stegmotorene.

Parametre	Y-akse	X-akse	Z-akse
Steg/vinkel (°)	1.8	1.8	1.8
Maks strømtrekk (A)	4.0	2.0	2.0
Spenning (V)	24	24	24
Maks holdemoment (Ncm)	165	59	59

Høyt strømtekk assosieres med høyt holdemoment. Y-stegmotor har høyere strømtrekk og høyere holdemoment enn X- og Z-stegmotor. Det er en fordel når rotoren er stasjonær, ettersom mindre dreiemoment (eller momentkraft) kreves for å overvinne tregheten til lasten. Derfor kan Y-stegmotor være bedre egnet som Z-stegmotor dersom lasten (verktøyet) er for tung for Z-stegmotor i gravitasjonsfeltet. Dersom det er et problem når CNC-maskinen utvikles videre og et tyngre verktøy benyttes, er det en forbedringsmulighet å bytte ut disse motorene med motorer som har høyere holdemoment.

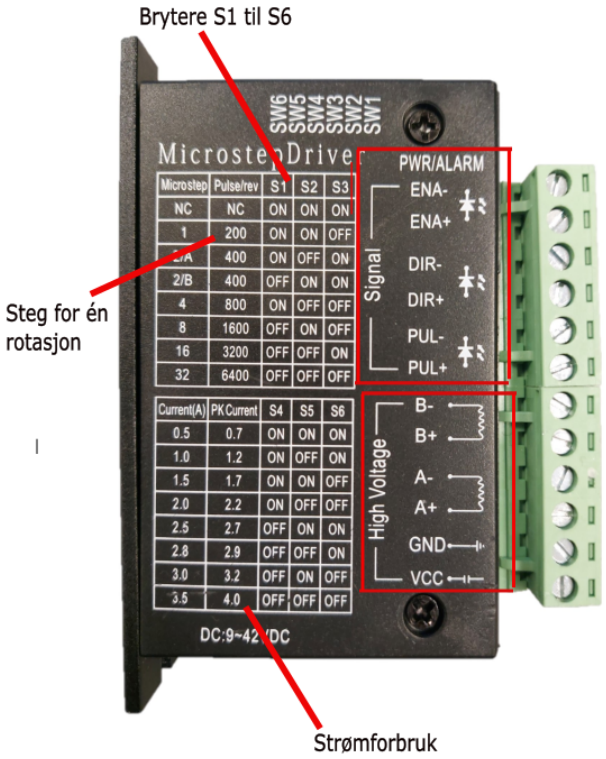
2.6.4 Styring av stegmotor med mikrostegdriver

Stegmotoren styres ved hjelp av en mikrosteg-driver. Driveren regulerer strømmen i spolene ved hjelp av to kontrollsignaler. Alternativet er å bruke driverkretser som halvbro- eler helbrodriverkretser. Det gir ikke like høy presisjon som en mikrosteg-driver og er bedre egnet for enklere applikasjoner. Mikrosteg-driver brukes for å oppnå mer nøyaktighet med å øke antall steg per rotasjon. Det er spesielt viktig for CNC-maskinen da det gir mulighet for et mer detaljert arbeid.

Mikrosteg-driveren sender elektriske signaler til motorviklingene som fører til rotasjon med eller mot klokken. Feil sekvens fører til at rotoren henger fast på ett punkt i stedet for å bevege seg. Mikrosteg-driveren håndterer dette ved å sende signal i rett sekvens til lederne. Farten til motoren endres ved å justere perioden til pulssignalene (instruksjonene) til mikrosteg-driveren [10].

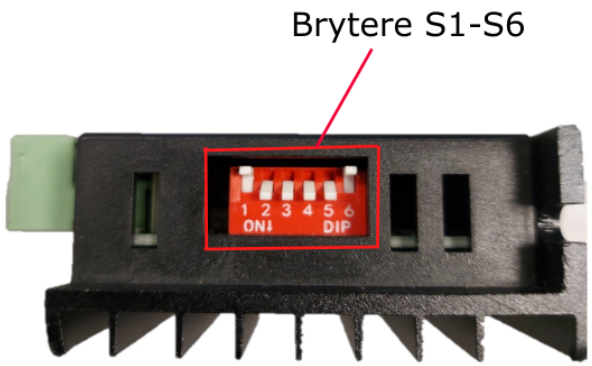
For å styre driveren benyttes Raspberry Pi og signalene fra Arduino. Signalene kobles til signal-delen på driveren. Mikrosteg-driveren er koblet til kraftforsyning med VCC og GND, mens motorens viklinger (ledningspar)

er koblet til A+, A-, B+ og B-, se figur (2.18). Inngangsspenningen er 9∼42 V DC. Styresignalene til mikrosteg-driveren er koblet til pinner på Arduino.



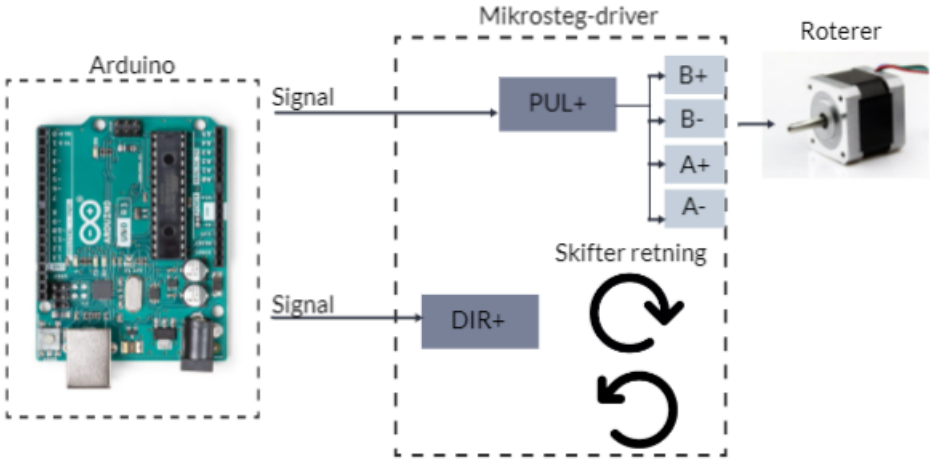
Figur 2.18: Viklingene til stegmotoren kobles til B-, B+, A- og A+. Et fullt steg til en motor med 200 steg, kan maks deles opp i 32 mikrosteg. Dette tilsvarer 6400 mikrosteg for en full rotasjon (200·32 = 6400). Figuren viser også hvilken innstilling bryterne S1-S6 skal ha for å oppnå ønsket antall mikrosteg og strømtrekk. Signalene kobles til pinner på Arduino.

Bryterpanelet for bryterne S1-S6 er vist i figur (2.19).



Figur 2.19: Brytere på mikrosteg-driveren. De har to innstillinger: av eller på. De kombineres etter tabellen på mikrosteg-driveren.

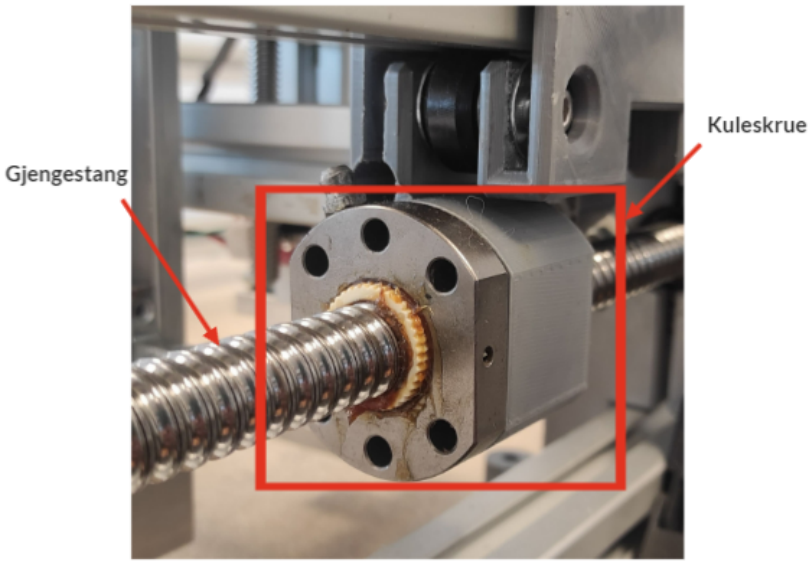
For å styre stegene brukes signal-inngangene. «PUL+» mottar elektriske puls-signal fra Arduino og mirkosteg-driveren kontrollerer sekvensen til lederene slik at rotoren beveger seg med eller mot klokken. Når driveren mottar puls på «DIR+» skifter retningen. «PUL-», «DIR-» og «ENA-» parallelkobles til *ground* på Arduino.



Figur 2.20: Puls-signalet (PUL+) får rotoren til å rotere, mens retnings-signalet (DIR+) endrer retningen.

2.6.5 Gjengestang og kuleskrue

Gjengestangen er koblet til skaftet på stegmotoren. Skaftet er koblet til rotoren. Dette får stangen til å rotere og deretter kuleskruen til å bevege seg. Når kuleskruen beveger seg beveger også verktøyet seg i arbeidsområdet, se figur (2.21).



Figur 2.21: Kuleskruen er festet til gjengestangen.

Gjengestangen festet til Y-stegmotor har en stigning på 5 mm. Det er da mulig å regne steg/mm, lengde a. Ved bruk av mikrosteg-driveren er det 400 steg/revolusjon. Utregning:

$$\mathrm{a} = \frac{400\ [steg]}{5\ [mm]} = 80\ [steg/mm] \tag{2.1}$$

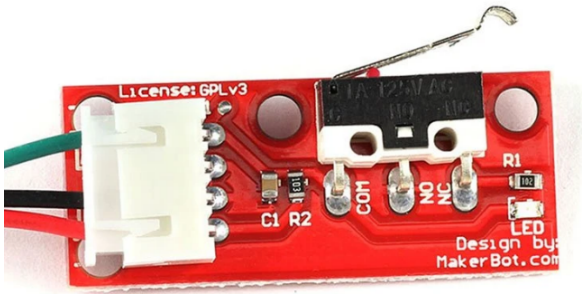
Gjengestangen festet til X- og Z-stegmotor har en stigning på 2 mm. Utregning for steg/mm, lengde b, er da:

$$\mathrm{b} = \frac{400\ [steg]}{2\ [mm]} = 200\ [steg/mm] \tag{2.2}$$

Lengdene a og b er nødvendige for maskinkontroller-programmet for å bevege pennen ønsket lengde.

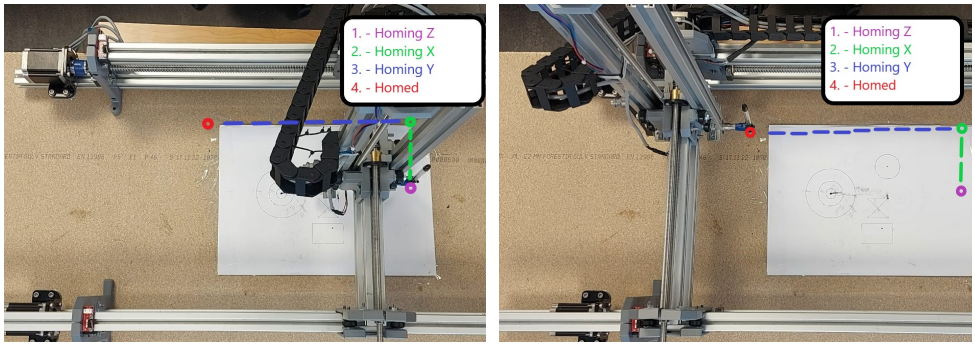
2.7 Endebryter

En viktig sikkerhetsfunksjon for CNC-maskinen er endebryterene. CNC-maskinen har tre mekaniske endestoppbrytere, se figur (2.22). Endebryterene plasseres på begynnelsen av hver akse (hjemmeposisjonen). Dette punktet brukes under hjemsyklusen.



Figur 2.22: Endebryteren som benyttets i prosjektet er designet av Makerbot [2] og kjøpes blant annet hos AliExpress [1].

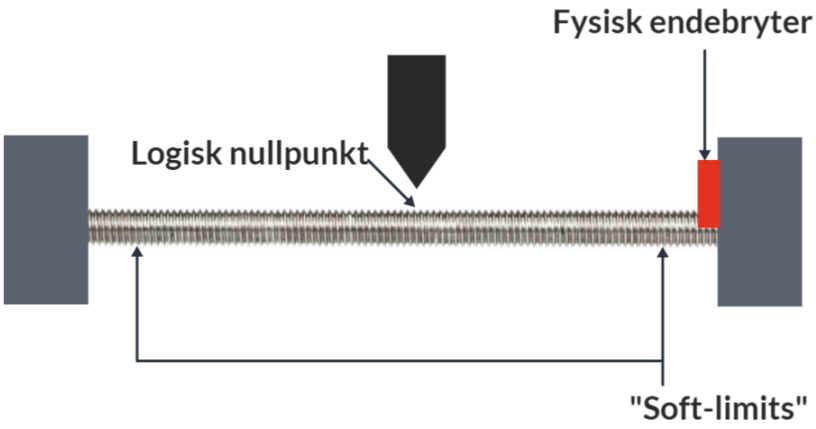
Hjemsyklus krever minst én endebryter for hver akse. Det er mulig å ha to endebrytere per akse for å hindre at motorene beveges utenfor grensene, men det er ikke nødvendig. For å spare tid, penger og kabler holder det å bruke én endebryter for hver akse og heller definere «programvare-grenser» (eng: *soft-limits*) i tillegg. Det er viktig å utføre hjemsyklus av maskinen ved oppstart for å kalibrere maskinen og få korrekt posisjon til verktøyet, se figur (2.23). Det hindrer kollisjon og sørger for at maskinkontroller-programmet vet hvor verktøyet befinner seg til en hver tid.



(a) Før hjemsyklus **(b)** Etter hjemsyklus

Figur 2.23: Hjemsyklus ved hjelp av endebryterene.
Z-akse utføres først, deretter X-akse og så Y-akse.

På engelsk kalles endebryterene for *hard-limits* som på norsk er «maskinvare-grenser». I tillegg til de fysiske endebryterene kan vi legge til «programvare-grenser» (eng: *soft-limits*). Maskinkontrolleren bruker programvare-grensene får å stanse kjøringen. Ved å sette koordinatene til programvare-grensene med en sikkerhetsdistanse på noen cm fra enden av aksen, hindres verktøyet i å gå for langt, se figur (2.24). Hvis verktøyet forsetter mot enden av aksen og forbi sikkerhetsdistansen så har det skjedd noe feil og brukeren har da tid til å bruke nødstopp. Vi har ikke lagt til slike programvare-grenser i maskinkontrolleren, men GRBL bruker det.



Figur 2.24: Figuren viser hvor *soft-limits* og endebryter (*hard-limit*) er plassert. Logisk nullpunkt (startpunkt) plasseres innenfor området, definert av *soft-limits*.

2.7.1 Kobling av endebryter

Endebryteren er en normalt lukket bryter og har tre tilkoblinger: jord (GND), inngangsspenning (+Vcc) og signalet (S). Koblingsskjema er i figurene (C.1) og (C.2) i vedlegg C.

2.7.2 Normalt åpen eller lukket kobling av endebryter

Bryteren i en elektrisk krets er enten åpen eller lukket. Vi bruker normalt lukket ettersom det fungerer som en innebygd sikkerhetsfunksjon. Hvis en ledning eller bryter er defekt blir kretsen åpnet/brutt og motorene stanser. Dersom bryteren hadde vært normalt åpen og en skade på ledningen oppstår, oppdages i verste tilfelle ikke dette før en skade allerede er gjort. Normalt lukket bryter reduserer risiko for uønskede hendelser.

2.7.3 Elektromagnetisk interferens

Ved implementering av endebryterene oppdages det støy. Frekvensen til støyet måles med oscilloskop. Støy oppstår mellom ledningene til bryterne og motorene, og fører til at endebryterne oppfører seg som berørt. Dette skyldes at ledningene tar opp elektromagnetisk interferens, forkortet EMI, fra motorene. Dette påvirker S-signalet på 5 V DC som sendes fra endebryterene. Dette løses med lavpass-filter og tvinnede ledninger.

Ettersom målet er så lav støy som mulig setter vi ønsket knekkfrekvens lavest mulig. Deretter tilpasses filteret etter hvilken motstand- og kondensatorverdi som er tilgjengelig. Vi velger en knekkfrekvens mellom 300 og 400 Hz og bruker formel for knekkfrekvens (2.3) til å finne verdiene til kondensatoren og motstanden .

$$\text{Knekkfrekvens: } f_c = \frac{1}{2\pi RC}\ [Hz] \tag{2.3}$$

Ved å velge en kondensator med standardverdi på 100 nF finner vi verdien til motstanden etterpå. Først med knekkfrekvens på 300 Hz og deretter 400

Hz. Deretter velges en tilgjengelig motstandsverdi mellom de to resultatene.

$$300\ [Hz] = \frac{1}{2\pi \cdot 100n \cdot R}$$

$$R = \frac{1}{2\pi \cdot 100n \cdot 300}\ [\Omega]$$

$$R = \underline{5305\ [\Omega]} \tag{2.4}$$

$$400\ [Hz] = \frac{1}{2\pi \cdot 100n \cdot R}$$

$$R = \frac{1}{2\pi \cdot 100n \cdot 400}\ [\Omega]$$

$$R = \underline{3979\ [\Omega]} \tag{2.5}$$

Vi velger fra standardtabell en motstand på 4700 Ω. Knekkfrekvensen er da 383.63 Hz (2.6).

$$\text{Knekkfrekvens: } f_c = \frac{1}{2\pi RC}\ [Hz]$$

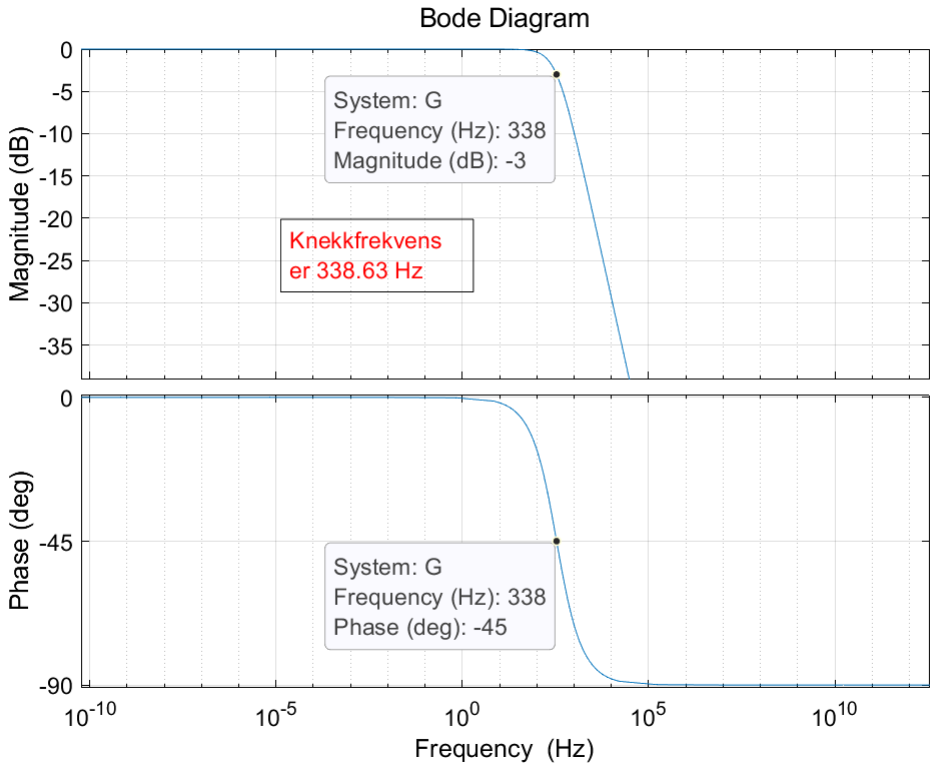
$$f_c = \frac{1}{2\pi \cdot 4700 \cdot 100n}\ [Hz]$$

$$f_c = \underline{\underline{383.63\ [Hz]}} \tag{2.6}$$

Verdiene brukes i et *matlab script* og bruker funksjonen *bode* for å få opp frekvensresponsen, se kode (2.1) og resultatet i figur (2.25).

Kode 2.1: *Matlab script* som gir frekvensrespons til filteret.

```
% Bode plot RC filter
R = 4700; % ohm
C = 100e-9; % Farat
T = R*C;
options = bodeoptions;
options.FreqUnits = 'Hz';
options.Title.FontSize = 16;
s = tf('s');
G = 1/(1+T*s);
bode(G, options)
grid on;
```

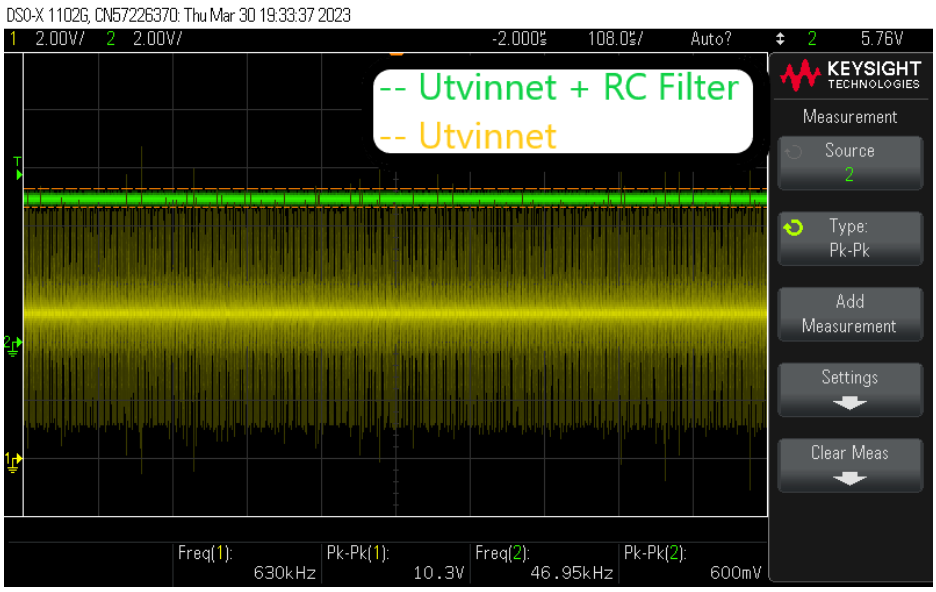


Figur 2.25: *Bode plot* som viser *Magnitude (dB)* og Fase (grader) mot Frekvens (Hz). Knekkfrekvensen er 338 Hz ved -3dB og -45 grader.

I tillegg til filteret tvinnes ledningene for å redusere støyet. Det gjøres for å utnytte prinsippet om magnetisk feltkansellering. Når to ledninger er ved siden av hverandre oppstår det et magnetisk felt som forstyrrer signalene gjennom ledningene. Dersom ledningene tvinnes om hverandre, genereres

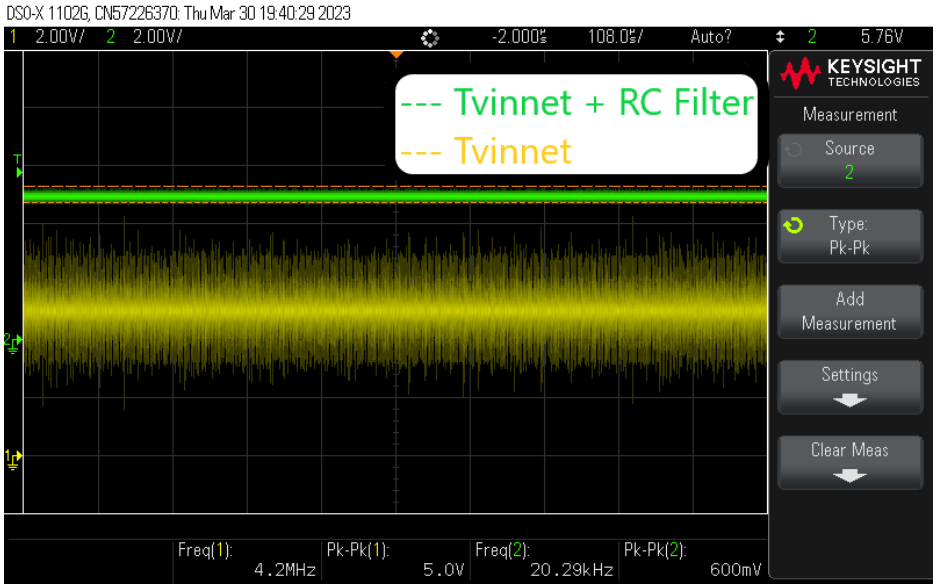
det et magnetisk felt som går i motsatt retning. De magnetiske feltene kansellerer hverandre ut. Det fører til redusert EMI ved å forbedre den elektromagnetiske kompatibiliteten mellom ledningene som ligger ved siden av hverandre.

I figur (2.26) vises signalet uten tvinnede ledninger og resultatet av signalet uten tvinnede ledninger og lavpass-filter.



Figur 2.26: Oscilloskopmåling av signalet ved kjøring av maskinen. Gult signal er før støyreduksjon og grønt signal er etter støyreduksjon. Gult signal er målt uten bruk av lavpass-filter og uten tvinnede ledninger, mens grønt signal er målt med lavpass-filter og uten tvinnede ledninger. Maksimal estimert *peak to peak* støy på gult signal er 10.3 V, mens estimert *peak to peak* på grønt signal er 600 mV. Høye frekvenser er filtrert bort.

I figur (2.27) vises resultatet med tvinnede ledninger og lavpass-filter.



Figur 2.27: Oscilloskopmåling ved kjøring av maskinen. Gult signal er før støyreduksjon og grønt signal er etter støyreduksjon. Gult signal er med tvinnede ledninger. Grønt signal er med lavpass-filter og tvinnede ledninger. Estimert *peak to peak* på gult signal er 5.0 V. Estimert *peak to peak* på grønt signal er 600 mV. Resultatet viser at lavpass-filteret og tvinnede ledninger fungerer, og at de høye frekvensene er filtrert bort.

Med tvinnede ledninger uten filter er støyen redusert fra *peak to peak* 10.3 V til 5.0 V. Støyen er fremdeles veldig høy ettersom vi ønsker lavest mulig støy. Det er derfor nødvendig å bruke lavpass-filter. Resultatet med lavpass-filter viser at støyen er redusert fra *peak to peak* 10.3 til 600 mV. Resultatet er 600 mV med eller uten tvinnede ledninger fordi lavpass-filteret fjerner mer støy enn det tvinnede ledninger gjør alene.

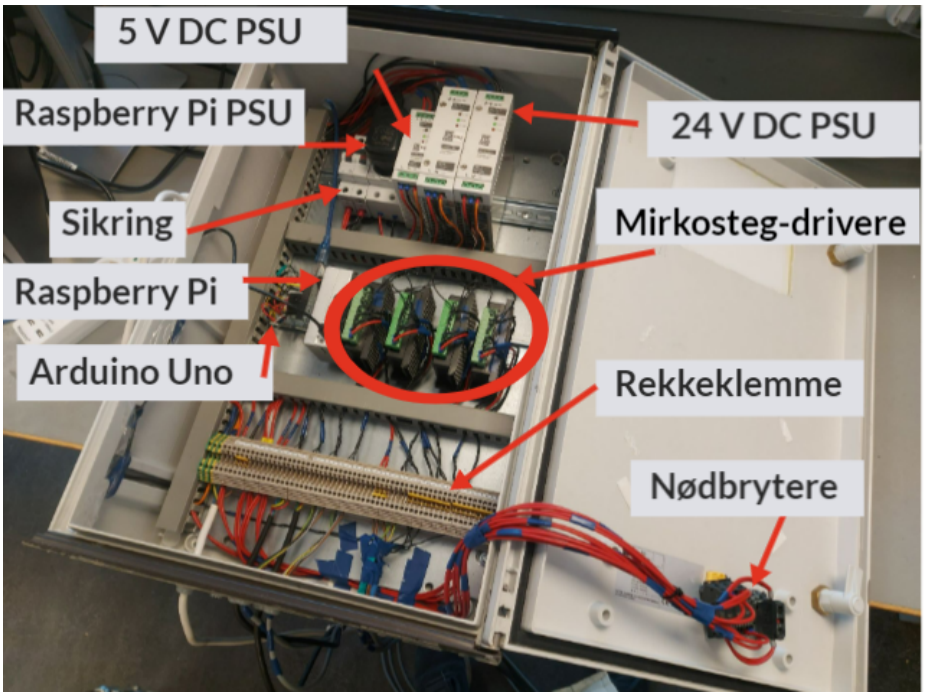
Det er mulig å gjøre mer for å redusere støy, men som ikke er nødvendig for driften av denne CNC-maskinen. Følgende alternativer er hentet fra «OpenBuilds» sin CNC-maskin [15]:

- **Isolere ledningene til stegmotorene**
  Ledningene til endebryterene og stegmotorene deler kabelkjede, som er en av grunnene til at støy oppstår under kjøring av maskinen. For å fjerne interferensen kan ledningene tvinnes og isoleres med ledningsskjold.

- **Ferittkjerner**
  Ferittkjerner kan kobles på ledningene til stegmotorene og fjerne mer av det høy-frekvente støyet [23].

2.8 El-tavle

El-tavlen er utstyrt med strømbrytere og sikringer for å beskytte det elektriske systemet mot overbelastning og kortslutninger. Hvis det oppstår en feil i kretsen utløses kretsbryteren i sikringen og hindrer skade på komponentene i systemet. Koblingsskjemaet ligger i vedlegg C, figur (C.2). I figur (2.28) vises komponentene i el-tavlen.



Figur 2.28: El-tavlen til CNC-maskinen.

Kapittel 3

Programvare

Dette kapittelet forklarer utførelsen av maskinkontroll-programmet. Programmet lastes opp på Arduino og har som oppgave å tolke G-koden for å styre stegmotorene.

3.1 G-kode

G-kode, også kjent som «RS-274», står for «geometrisk kode». Koden gir instruksjoner til maskinkontrolleren som deretter forteller stegmotorene hvilken posisjon, hastighet, akselerasjon og bevegelse som skal følges.

Følgende liste gir informasjon om hva G-kode instruksjonene betyr:

- **G**: Kommando.
- **M**: Brukes for forskjellige funksjonskoder som frese og spindel-kontroll.
- **X**, **Y** og **Z**: Brukes til å gi absolutte eller inkrementelle posisjonskommandoer.
- **I**, **J** og **K**: Angir sentrum til en bue i henhold til X-, Y- og Z-aksene.

Tabell (D.2) i vedlegg D gir oversikt over G-kode instruksjonene og hva de betyr. Her er eksempeler på de mest vanlige G-kode kommandoene:

- **G90** - Absolutt koordinat
  Beveger verktøyet til spesifisert X-, Y- eller Z-koordinat, se figur (2.5).

- **G91** - Inkrementell koordinat
  Beveger verktøyet til nåværende koordinat pluss oppgitt X-, Y- eller Z-koordinat, se figur (2.5).

- **G0 / G00** - Hurtig posisjonering
  Eksempel:
  G90 *% Absolutt koordinat*
  G0 X100 Y100 *% Bevegelse med maks hastighet*
  Denne kommandoen brukes til å flytte verktøyet raskt. I eksempelet flytter maskinen verktøyet til absolutt posisjon X = 100 mm og Y = 100 mm fra fra origo med maks hastighet (*feedrate*).

- **G1 / G01** - Lineær interpolasjon
  Eksempel:
  G90 *% Absolutt koordinat*
  G1 Z10 F300 *% Bevegelse med hastighet 300 mm/min*
  Denne kommandoen brukes til å flytte verktøyet. I eksempelet flytter maskinen verktøyet til absolutt posisjon Z = 10 mm fra origo med hastighet på 300 mm/min.

- **G2 / G02** - Sirkulær interpolasjon med klokken
  Mer om dette senere.

- **G3 / G03** - Sirkulær interpolasjon mot klokken
  Mer om dette senere.

3.2 CAM-Design

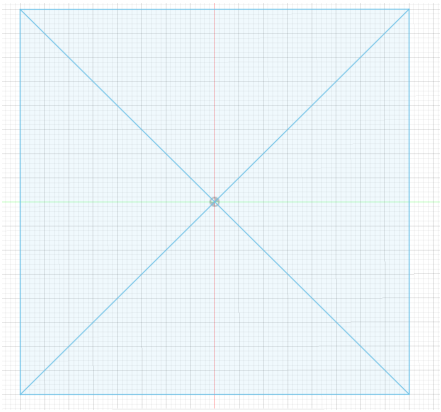
Programvaren vi bruker til å lage G-kode med er «Fusion 360». Programvaren har en «CAM-modul» som gir oss muligheten til å designe 3D-modeller. I modulen velges verktøy, materialer, maskinbane og fresemetode. Fra 3D-modellen genereres det G-kode som inneholder de valgte innstillingene organisert etter linjenummer.

For å lage et kvadrat med diagonaler produseres det en skisse i Fusion 360. Deretter eksporteres skissen som en G-kode-fil med GRBL-standarden.

Filen inneholder en sekvens med kommandoer som maskinkontrolleren leser og utfører, se kode (3.1) og figur (3.1a) for skissen.

Kode (3.1) G-kode instruksjoner for et kvadrat med diagonaler.

```
1  G92
2  G90
3  M17
4  G0 Z20 F2000
5  G0 X0 Y0 F5000
6  G0 Z0 F2000
7  G0 X40 Y0 F5000
8  G0 X40 Y-40
9  G0 X0 Y-40
10 G0 X0 Y0
11 G0 X40 Y-40
12 G0 Z20 F2000
13 G0 X40 Y0
14 G0 Z0 F2000
15 G0 X0 Y-40
16 G0 Z20 F2000
17 G0 X0 Y0 F4000
18 Z0 F2000
```



(a) Skisse: kvadrat med diagonaler.

Figur 3.1: Eksempel på G-kode og skisse laget i Fusion 360.

Programmet i koden (3.1) inneholder G-kode instruksjoner for å lage et kvadrat med diagonaler. Koden er på 18 linjer og er lagret i en .nc-fil. Koden inneholder X-, Y- og Z-instruksjoner som forteller hvor verktøyet skal beveges i arbeidsområdet. F-instruksjonene er *feedrate*. Tabell (D.2) i vedlegg D inneholder en oversikt over hva G- og M- instruksjonene betyr.

3.3 Brukergrensesnitt

Det visuelle brukergrensesnittet, forkortet GUI (eng: *Graphical User Interface*), er laget i Python 3.10 [17]. Brukergrensesnittet kjøres enten med Windows 10 eller trådløst via Raspberry Pi 4B [18]. Vi benytter to brukergrensesnitt: vårt eget til maskinkontrolleren, se figur (3.3), og UGS. UGS fortelles det mer om senere.

Følgende Python-moduler er benyttet i programmeringen av brukergrensesnittet:

- **PySimpleGUI** til å lage det visuelle brukergrensesnittet [16].
- **MatPlotLib** til å visualisere posisjonen til verktøyet i 2D og 3D [11].
- **Serial** til seriekommunikasjon mellom GUI og maskinkontroller [13].

Kode (3.2) tilsvarer kommunikasjonen mellom brukergrensesnittet og maskinkontrolleren (Arduino). Maskinkontrolleren bruker *Universal Serial Asynchronous Receiver Transmitter*, forkortet USART. USART er en seriell kommunikasjonsprotokoll med en bestemt overføringshastighet (eng: *baudrate*) [8].

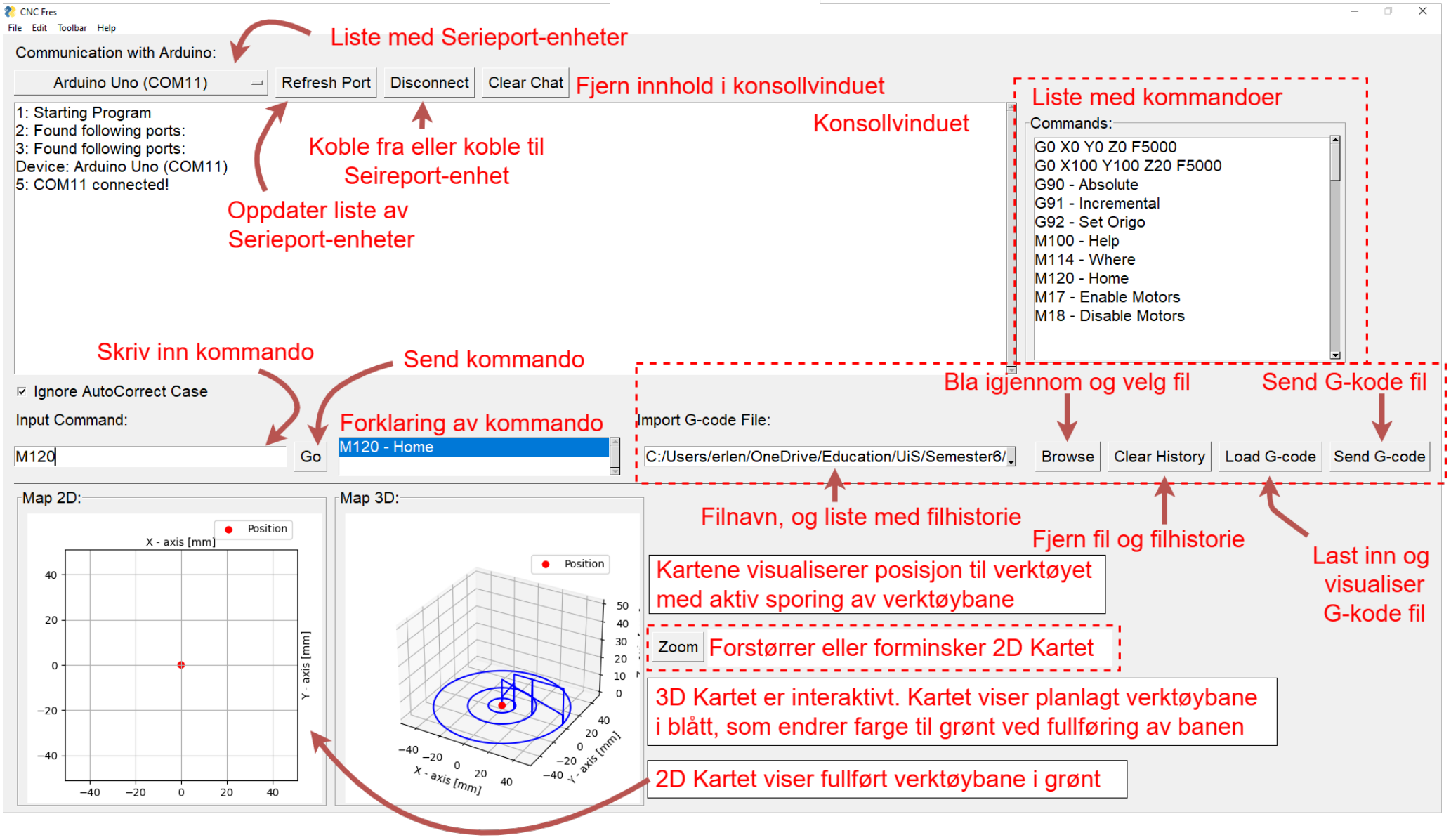
Kode 3.2: Seriekommunikasjon mellom brukergrensesnittet og maskinkontrolleren.

(a) Koden i Python.

```
1 import serial
2 ID = "COM10"
3 baud = 115200
4 s = serial.Serial(ID, baud)
```

(b) Koden i C++.

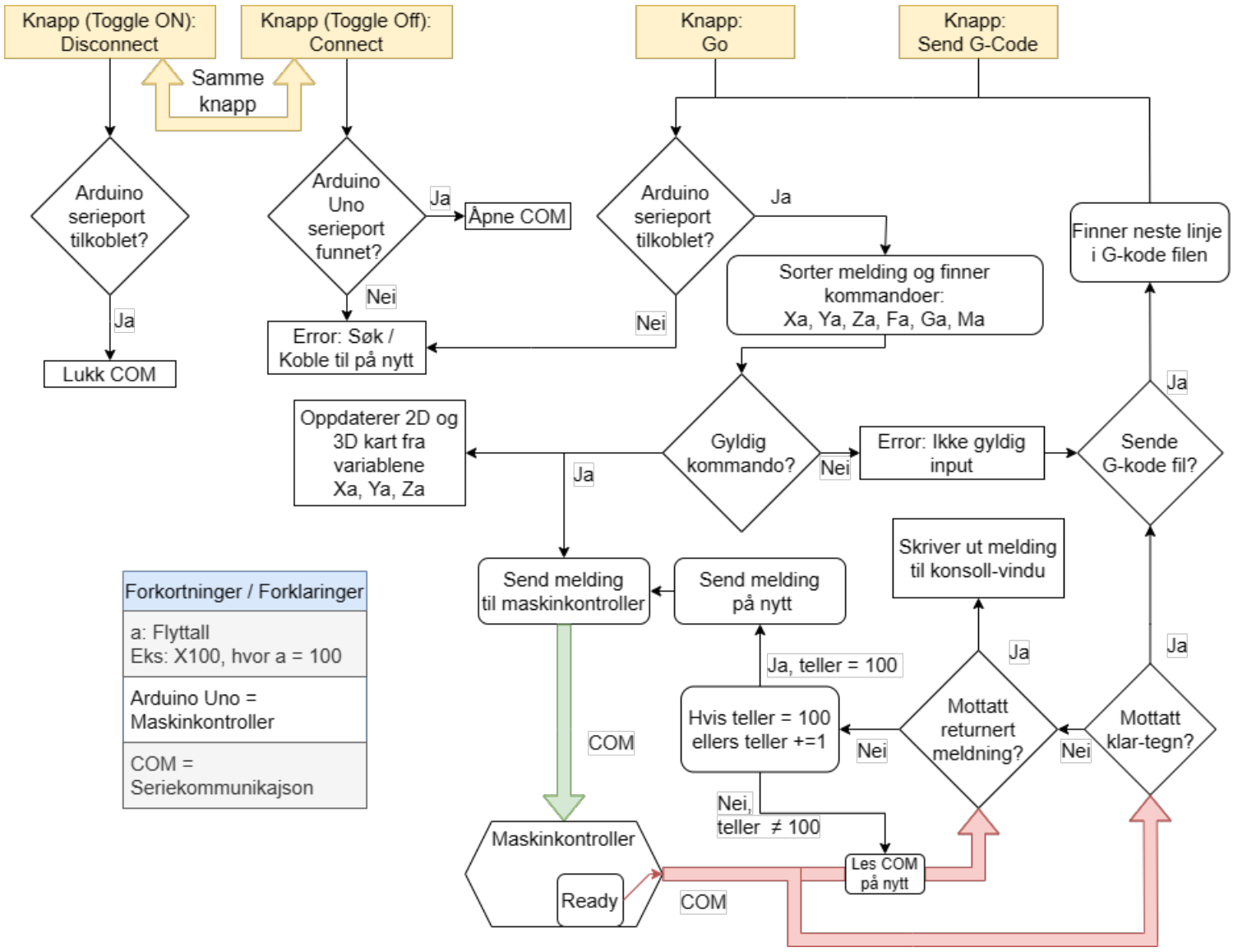
```
1
2 void setup() {
3   Serial.begin(115200);
4 }
```



Figur 3.3: Brukergrensesnittet.

3.3 Brukergrensesnitt

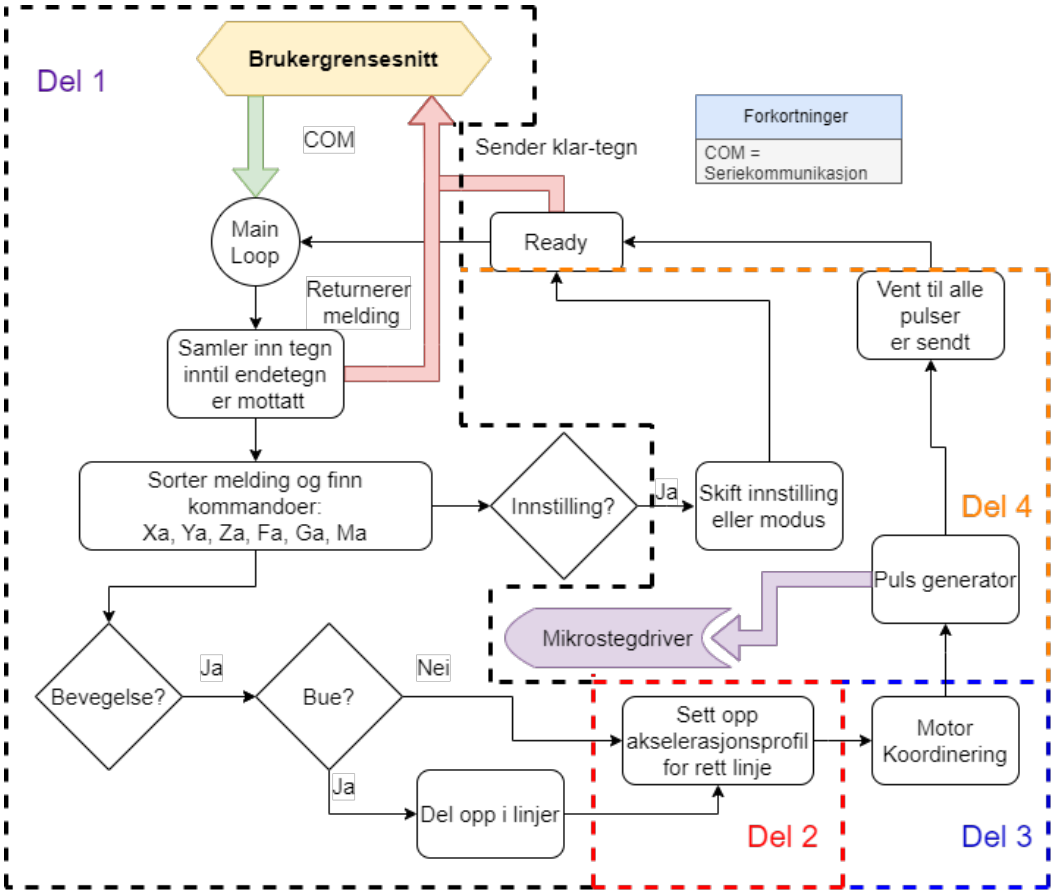
Figur (3.4) viser flytskjema over funksjonene til brukergrensesnittet.



Figur 3.4: Flytskjema over brukergrensesnittet.

3.4 Maskinkontrolleren

Første del av dette kapittelet tar for seg hvorfor akselerasjon er viktig og hvordan det implementeres. Deretter hvordan maskinkontrolleren styrer stegmotorene. Figur (3.5) viser flytskjema av maskinkontrolleren.



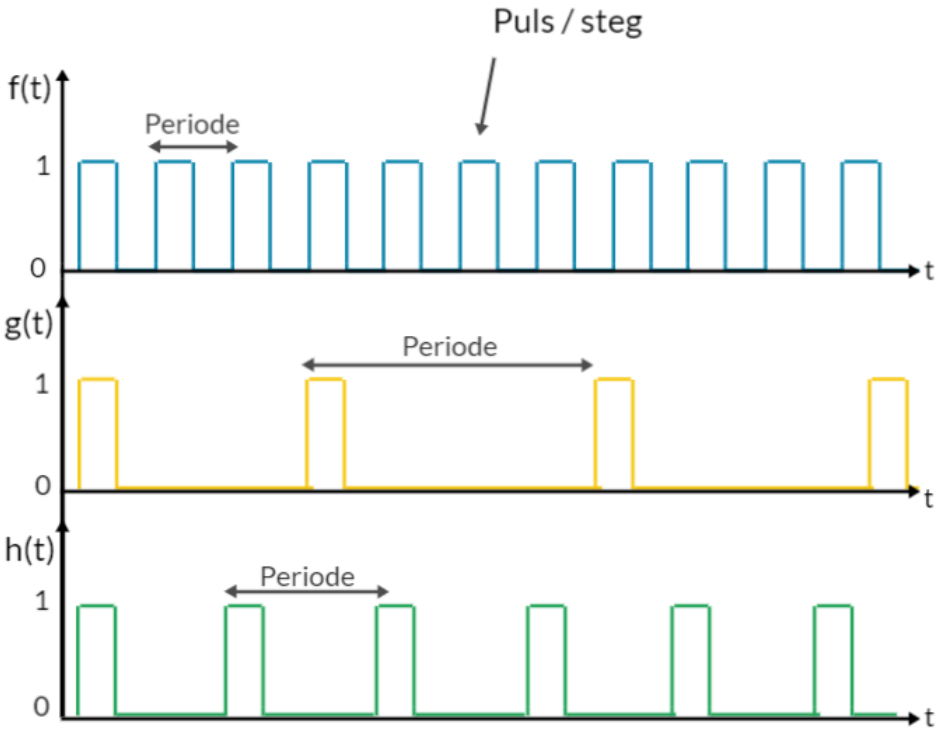
Figur 3.5: Flytskjema av maskinkontrolleren.

3.4.1 Akselerasjon

Dette delkapittelet tar for seg hva som skjer i del 2 i figur (3.5): sette opp akselerasjonsprofil for en rett linje.

For å sørge for at stegmotorene kjører rett og med ønsket hastighet, sender vi pulssignaler som akselererer motorene. Når vi sender pulssignaler som er raskere enn det motoren håndterer kan motoren hoppe over steg eller i verste tilfelle stoppe opp. For å hindre at disse problemene oppstår benyttes akselerasjon ved hjelp av pulssignal.

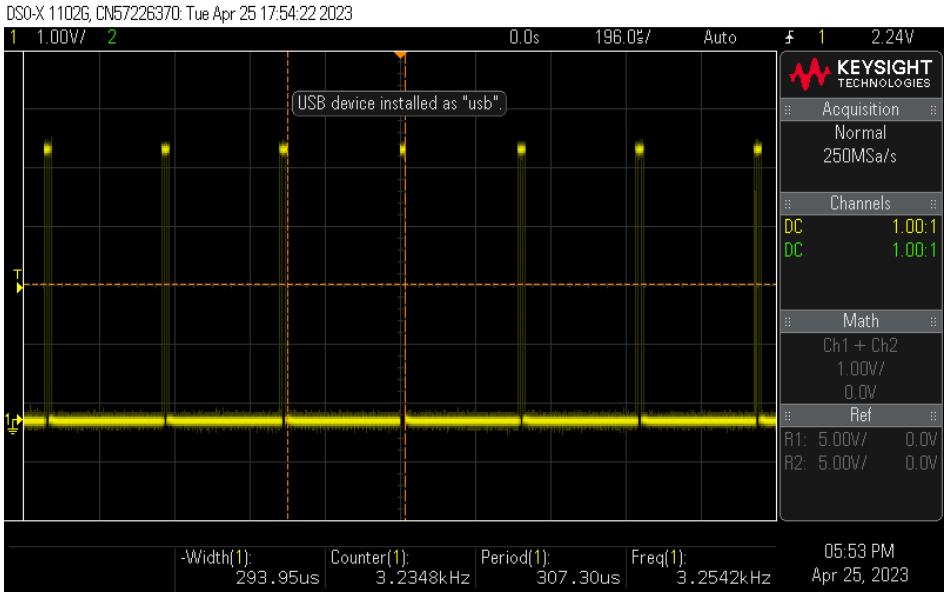
Signalene sendes til mikrosteg-driveren i form av et firkant-signal. Ved å endre perioden til firkant-signalet endres hastigheten til stegmotoren. Figur (3.6) viser hvordan firkant-signalet ser ut med ulik periode.



Figur 3.6: Firkant-signalene f, g og h har ulik periode mellom pulsene, som indikerer ulik hastighet. Disse signalene sendes til mikrosteg-driverene som håndterer styringen av stegmotorene.

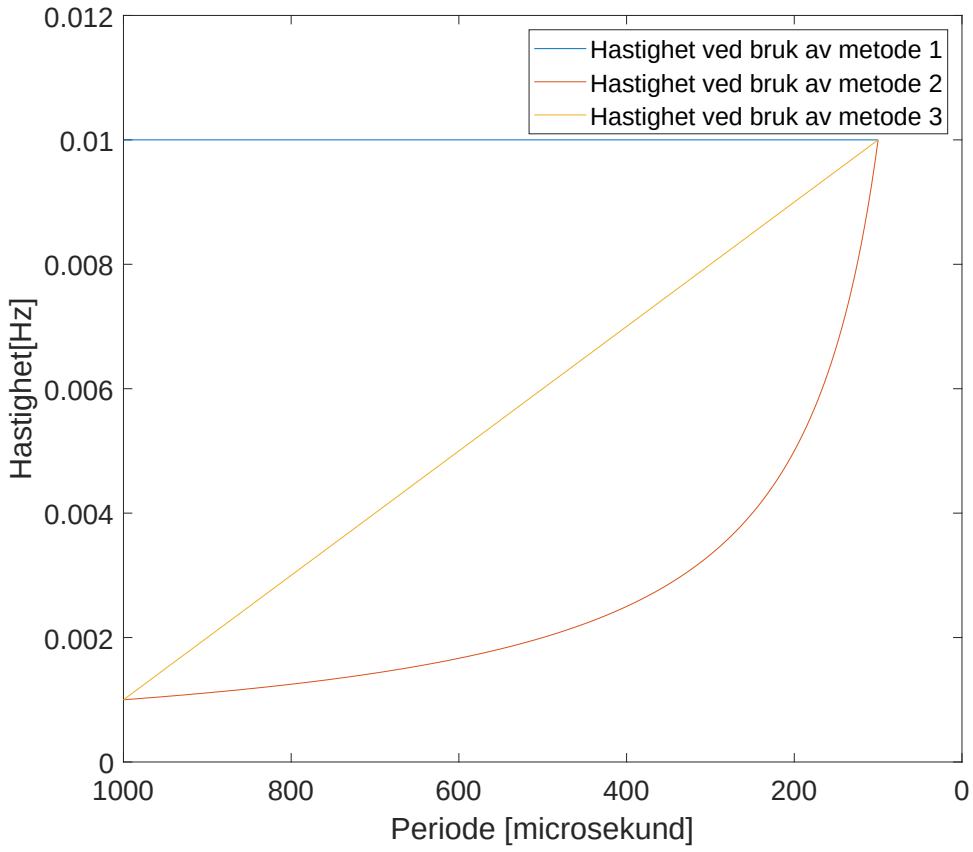
I kapittelet om mikrosteg-driver (2.6.4) forklares det hvordan en mikrosteg-driver styrer stegmotorene. Kode (B.1) i vedlegg B styrer stegmotorene.

Firkant-signal genereres ved å endre på tiden mellom høy og lav flanke. Dette gir mulighet til å sette en ønsket hastighet til stegmotorene. Når pulssignalet har en fast periode mellom hver puls betyr det at det ikke er akselerasjon. Dette kaller vi for «metode 1». Ingen akselerasjon fører til at motorene ikke klarer å holde følge med signalet fra maskinkontrolleren. Fartsendringen er for rask for stegmotoren. Vi målte den laveste signalperioden som motoren klarte å følge med på til å være omtrent 307 µs, se figur (3.7).



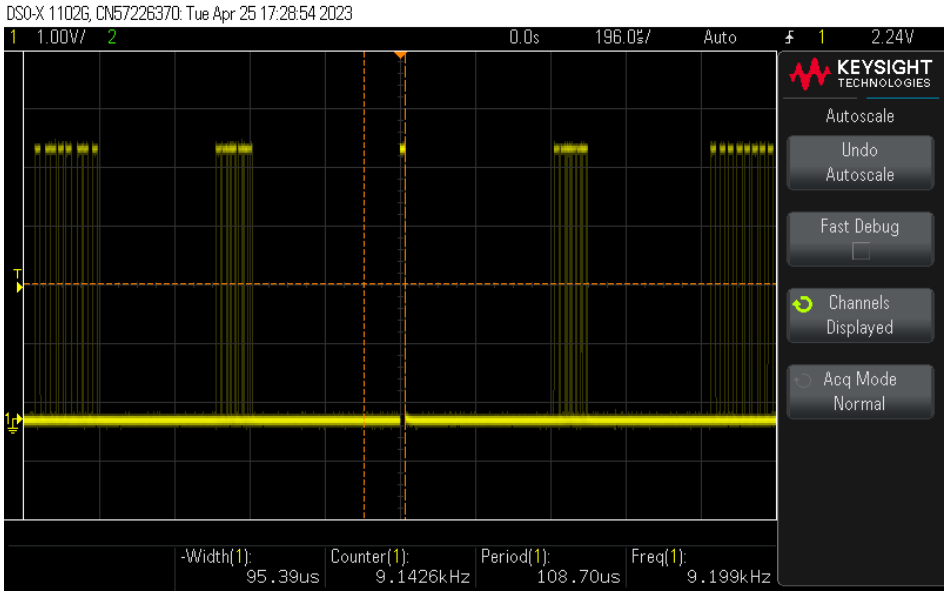
Figur 3.7: Pulssignalet produsert med kode (B.1) og er målt periode til metode 1.

I neste forsøk akselererer vi stegmotorene ved å endre perioden med ett fast antall sekunder. Dette kaller vi for «metode 2». Responsene til metodene 1 og 2 vises i figur (3.8) sammen med en lineær respons vi kaller for «metode 3». Metode 3 fortelles det mer om senere. Grafen viser hvordan metode 2 øker hastigheten med de største mengdene når perioden nærmer seg omtrent 100 µs (eksponentiell vekst). Dette gir de samme problemene som ved metode 1.



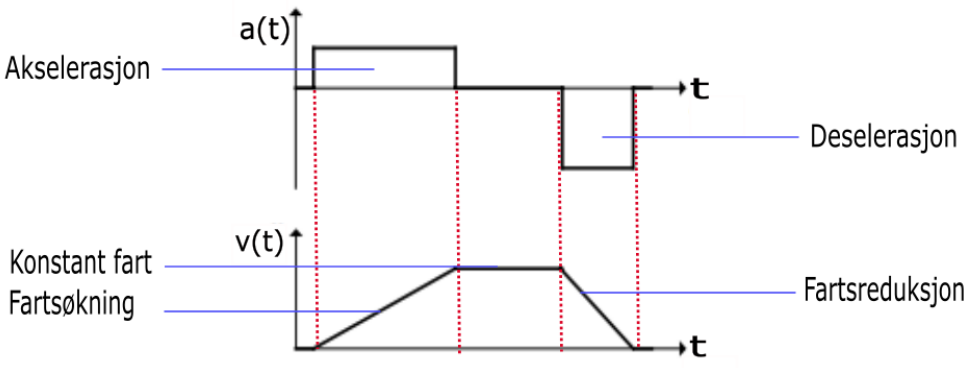
Figur 3.8: Simulert hastighetsprofil ved bruk av kodene (B.1), (B.2) og (B.3) for metodene 1, 2 og 3. Metode 1 har ingen stigningstall. Metode 2 øker eksponentielt. Metode 3 har en lineær fartsøkning.

Laveste periode stegmotoren klarte å holde følge med ved bruk av Metode 2 er omtrent 109 µs, se figur (3.8).



Figur 3.9: Pulssignalet produsert med kode (B.2) og er målt periode til metode 2.

Metode 1 og 2 gir grunnlaget for den endelige metoden: metode 3. I en artikkel fra 2006 om lineær hastighetskontroll for stegmotorer forklares det hvordan akselerasjon kan oppnås [7]. I figur (3.10) vises det en fart- og akselerasjonsgraf.



Figur 3.10: Grafene viser sammenheng mellom akselerasjon og fart [7].

Målet er å oppnå en lignende akselerasjons-profil for stegmotorene. Vi bruker formel (3.1) fra artikkelen.

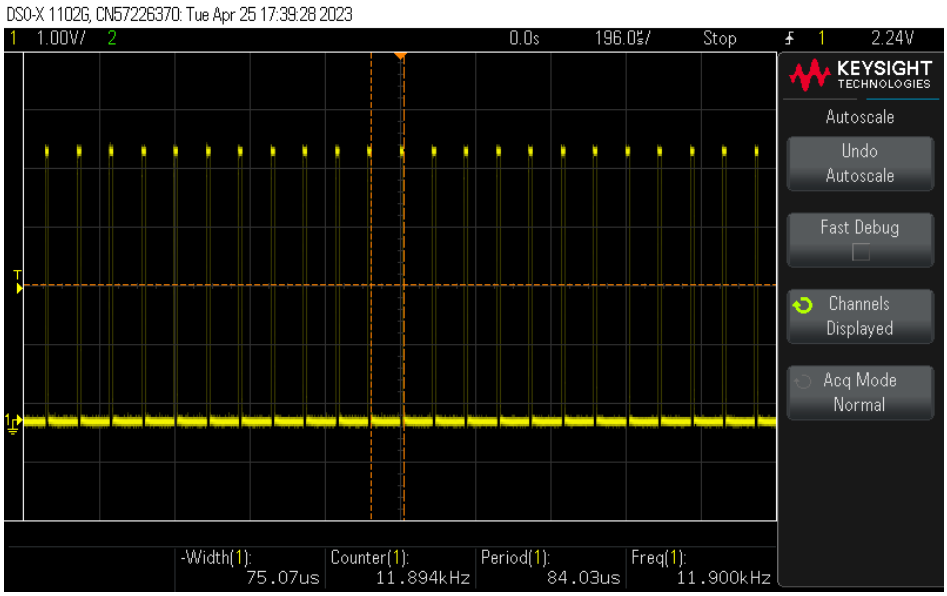
$\boldsymbol{n}$: antall pulser produsert.
$\boldsymbol{c_n}$: forsinkelsen for n-te steget.

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n-1} \tag{3.1}$$

Kode 3.2: Implementering av formel (3.1) i maksinkontrolleren.

```
55    if ( rampUpStepCount == 0 ) { // akselerasjonsfasen
56      n++;
57      d = d - (2 * d) / (4 * n + 1); // likning for cn i fra AVR
        datablad, her er cn skiftet om til d for delay for å
        enklere kunne følge koden.
58      if ( d ≤ maxSpeed ) { // Sjekker om vi har nådd max
        hastighet
```

Ved implementering av formel (3.1) holder motorene følge med en signalperiode på omtrent 84 µs, se figur (3.11).



Figur 3.11: Pulssignalet produsert med kode (B.3) og er målt periode til metode 3.

3.4.2 Koordinert kontroll av flere motorer

Dette delkapittelet tar for seg hva som skjer i del 3 i flytskjemaet av maskinkontrolleren, se figur (3.5).

For å kontrollere flere motorer samtidig er det viktig å ta hensyn til antall steg. Vi må sørge for at motorene utfører bevegelsene sine og stanser samtidig. Det er nødvendig slik at motorene kan utføre koordinerte bevegelser, som for eksempel å lage diagonal bevegelse. X- og Y-stegmotor bruker ulik mengde steg for samme distanse. X-stegmotor bruker 200 steg/mm og Y-stegmotor bruker 80 steg/mm. Derfor må utførelsestid økes for motoren som bruker kortest tid.

Kode 3.3: Koden finner hvilken motor som bruker lengst tid og deretter hvor mye tregere de andre motorene må kjøre.

```
363 //------------------------------------------------------------
364 //-Justerer hastigheten slik at alle motorene starter og
        stopper på samme tid-//
365
366 void adjustSpeedScales() {
367   float maxTime = 0;
368   for (int i = 0; i < NUM_STEPPERS; i++) {
369     if ( ! ((1 << i) & remainingSteppersFlag) )
370       continue;
371     if ( steppers[i].estTimeForMove > maxTime )
372       maxTime = steppers[i].estTimeForMove;
373   }
374   if ( maxTime != 0 ) {
375     for (int i = 0; i < NUM_STEPPERS; i++) {
376       if ( ! ( (1 << i) & remainingSteppersFlag) )
377         continue;
378       steppers[i].speedScale = maxTime / ...
        steppers[i].estTimeForMove;
379       Serial.print("SpeedScaler: ");
380       Serial.print(steppers[i].speedScale);
381     }
382   }
383 }
384 //------------------------------------------------------------
```

Kode (3.3) brukes til å finne motoren som tar lengst tid. For å finne ut hvor mye lengre tid de andre motorene må ta utføres følgende utregninger:

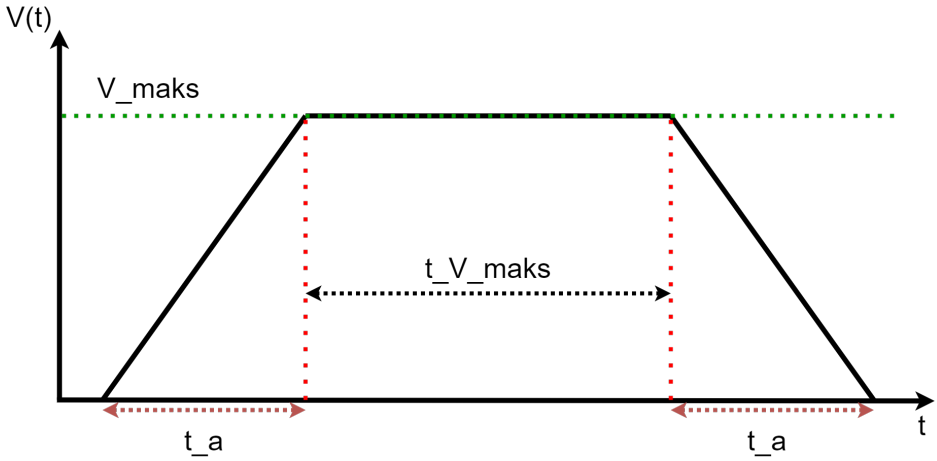
MaxTime: Estimert tid for motoren som tar lengst tid.
estTimeForMove: Estimert tid for de andre motorene.
speedScale: Forholdstall for hvor mange ganger de raskere motoren beveger seg.

$$speedScale = \frac{MaxTime}{estTimeForMove} \tag{3.2}$$

Eksempel: *MaxTime* er 2000 s og *estTimerForMove* er 1000 s, for X og Y motor. Da blir *speedScale*:

$$speedScale = \frac{MaxTime}{estTimeForMove} = \frac{2000\ s}{1000\ s} = 2$$

SpeedScale multipliseres med perioden til Y-stegmotor slik at den stopper samtidig som X-stegmotor. For å regne ut *speedScale*, trenger vi utførelsestiden til hver motor. For å regne utførelsestiden tregner vi å vite tid gitt i graf (3.12):



Figur 3.12: For å regne ut utførelsestiden trenger vi t_a, tiden motorene akselererer og $t_{V_{maks}}$, tiden brukt i maks hastighet.

Formel (3.3) brukes for å finne $t_{V_{maks}}$:

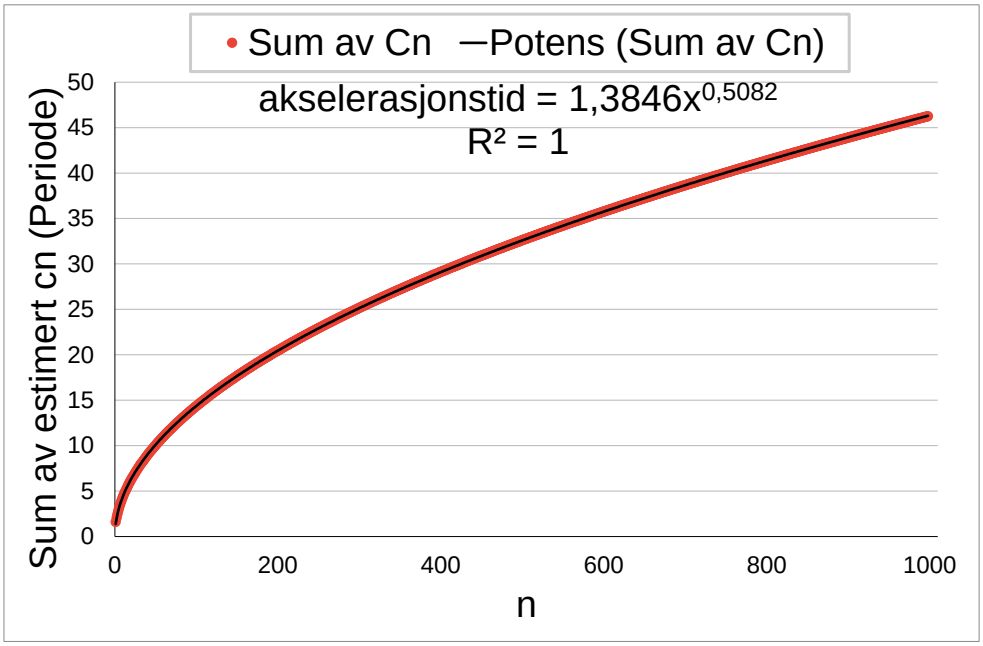
Steg$_{V_{maks}}$: stegene som blir utført i maks hastighet.
Periode$_{min}$: minste periode.

$$Steg_{V_{maks}} \cdot Periode_{min} = t_{V_{maks}} \tag{3.3}$$

For å finne t_a samler vi resultatene fra formel (3.1) og summerer dem:

$$\sum_{k=0}^{n} c_k = t_a \tag{3.4}$$

Vi plotter verdiene fra formel (3.4) og får resultatet i figur (3.13):



Figur 3.13: Grafen viser resultat fra formel (3.4) og likning for t_a ved hjelp av regresjon.

Likningen (3.5) implementeres i kode (3.4).

$$\mathrm{t}_a = 1,385 \cdot n^{0,508} \tag{3.5}$$

Kode 3.4: Kode brukt for å regne ut total akselerasjonstid.

```
218     return s.c0 * 1.385*pow(numSteps ,0.508);
```

Resultatet brukes til å regne ut *estTimeForMove* (3.6) som gir muligheten til å finne *speedScale* (3.2).

$$estTimeForMove = t_a \cdot 2 \ + \ t_{V_{maks}} \tag{3.6}$$

3.4.3 Utføring av g-kode

Dette delkapittelet tar for seg hva som skjer i del 1 i flytskjemaet av maskinkontrolleren, se figur (3.5).

Maskinkontrolleren mottar G-kode fra brukergrensesnittet. «\n» er stoppbit (eng: *stop-bit*) og skiller mellom linjene i G-kode filen. Det forteller maskinkontrolleren at én G-kode linje er motatt. Deretter sendes linjen videre til dekoding. Programmet leter etter G-kodeord som er bokstaver etterfulgt av tall (eks: G2, X10 og M20). Kode (3.5) finner «G0» eller «G1» etterfulgt av X-, Y-, Z- eller F-kommandoer.

Kode 3.5: Koden finner G0 eller G1 etterfulgt av X, Y, Z eller F kommandoer.

```
616
617 //===============
618
619
620 void processCommand() {
621
622 if(gittKommando[0]=='G'){
623   G = true;
624   M = false;
625 }else if(gittKommando[0]=='M'){
626   G = false;
627   M = true;
628 }
629
630 if(G==true){
631 cmd = parseNumber('G',cmd);
632 switch(cmd)
633 {
634   case 0: case 1:
635   {
636     feedrate(parseNumber('F',fr));
637     line(parseNumber('X',(mode_abs?px:0)),
638     parseNumber('Y',(mode_abs?py:0)),
639     parseNumber('Z',(mode_abs?pz:0)));
640     Serial.println(cmd);
641     break;
642
643   }
```

For eksempel sender vi inn G-kode linje «G1 X10 Z5 F2000». Kode (3.5) vil sjekke om første tegn i linjen er en «G» og deretter se om den finne «0» eller «1». Deretter letes det etter hjelpe-ord som i dette tilfellet er «F». I koden brukes to funksjoner for dette:

Feedrate(): Tar inn og setter ny matehastighet (variabel: **nfr**) i [mm/min]. For å bruke det som periode konverterer vi det til [µs/steg]. Vi bruker «a» og «b» som resultat fra (2.1) og (2.2). Nedenfor er det brukt «a».

$$\text{nfr } [mm/min] = \frac{\text{nfr } [mm/min] \cdot a \ [steg/mm]\cdot}{60000000} = \frac{\text{nfr } [steg/min]}{60000000} \tag{3.7}$$

$$\Rightarrow \frac{300000}{\text{nfr}} \ [\mu s/steg] = \text{periode for X og Z} \tag{3.8}$$

$$\Rightarrow \frac{750000}{\text{nfr}} \ [\mu s/steg] = \text{periode for Y} \tag{3.9}$$

parseNumber(): Brukes for å dele opp G-kodeord. Funksjonen tar inn to variabler: bokstaven som skal letes etter (her: F) og hva verdien er. Dersom et tall ikke blir funnet benyttes en standard verdi (eng: *default value*). Om «F» er med eller ikke letes det etter X, Y og Z. I eksemplet finner vi «X10» og «Z5» som sendes videre via «Line()» funksjonen.

Line(): Brukes for å planlegge antall pulser som trengs for antall millimeter bevegelse. For å konvertere brukes formlene (2.1) og (2.2). $steg/mm = pulser/mm$. Programmet sjekker også hvilken stegmotor som skal beveges. I eksemplet har vi fått inn ordre for X- og Z-stegmotor, så Y-stegmotor beveges ikke.

Etter at disse delene av programmet er utført vil pulsene bli sendt til akselerasjons planlegging (kap: 3.4.1) og motorkoordinering (kap: 3.4.2) før pulsene genereres.

3.4.4 Signal-generering

Dette delkapittelet tar for seg hva som skjer i del 4 i flytskjemaet av maskinkontrolleren, se figur (3.5).

Informasjonem om Atmel328P er funnet i dette databladet [6]. I koden for signal-generatoren benyttes en tidsavbryter (eng: *Time-Interrupt*). Vi bruker «TIMER 1» i CTC. Det betyr at TIMER 1 bruker en teller basert på klokke-hastigheten til CPU-en. Hver gang telleren blir full aktiveres «ISR()» som står for *Interupt-Service-Routine*. ISR() benyttes for å produsere pulssignalene som sendes til mikrosteg-driverene. Deretter re-settes telleren.

Pseudokoden under viser hvordan vi bruker funksjonen ISR() for å produsere pulser for mikrossteg-driverene, samt bestemme når vi akselererer, har konstant fart og deselererer.

Pseudokode 1 Koden viser hvordan puls-generatoren fungerer.

```
for i = 0, i < AntallMotorer, i++ do
    if MotorSteg == 0 then
        Motore er ferdig
    end if
    if MotorStegGjort < TotalMotorSteg then
        LagPuls
        MotorStegGjort++
        if MotorStegGjort ≤ TotalMotorSteg then
            MotorSteg = 0
        end if
    end if
    if Akselerer then
        n++
        regner cn fra formel (3.1)              ▷ Her brukes formel (3.1)
        if cn < minPeriode then
            cn = minPeriode
            StopAkselerer
        end if
        if NårIkkeTopHastighet then
            StoppAkselerer
        end if
    else if Deselerer then
        regner cn fra formel (3.1)              ▷ Her brukes formel 83.1)
        n − −
    end if
    JusterPeriode for koordinerte bevegelser
end for
```

Kapittel 4

GRBL og UGS

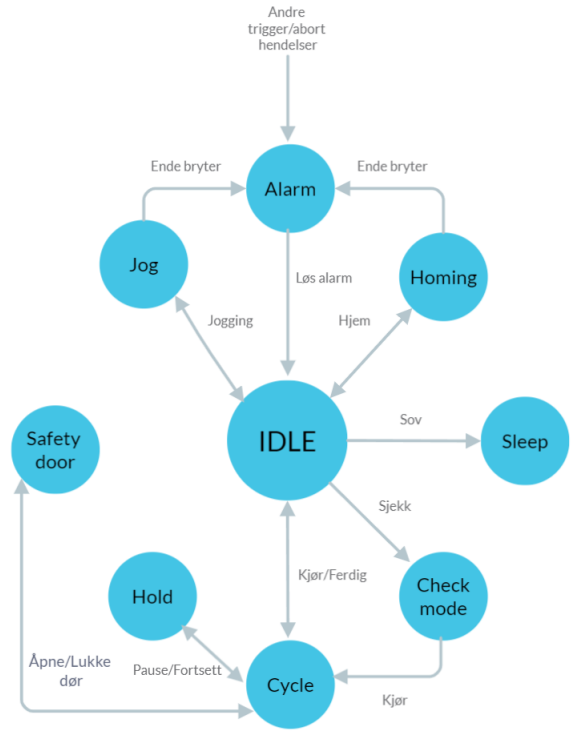
Universal Gcode Sender, forkortet UGS, er et brukergrensesnitt som brukes til å kontrollere maskinkontrollere som for eksempel GRBL. GRBL er en åpen kildekode-programvare hentet fra github-siden «gnea» [21]. GRBL innstalleres på Arduino og fungerer for CNC-maskiner. Vi bruker UGS til å sende G-kode-instruksjoner til GRBL. GRBL oversetter G-koden og sender instruksjonene videre til mikrosteg-driverene som igjen kontrollerer stegmotorene.

4.1 GRBL Struktur

GRBL sine systemer går innom tre forskjellige blokker: status-kontoller, oversetter og bakgrunns-prosesser. Status-kontroll er blokken som sender GRBL fra en status til den neste. Oversetter inneholder modulene som skal jobbe sammen om å oversette G-kode og sette opp en «plan» for hastighetskontroll. Bakgrunns-prosessene består av moduler som har spesifikke oppgaver. For eksempel filen *stepper.c* som generere pulssignal.

4.2 GRBL's Statuser

Diagrammet i figur (4.1) viser oversikt over statusene til GRBL.



Figur 4.1: Diagram som viser statusene til GRBL.

Status-kontroller består av tre deler:

- Initialisering, som er «Main» i programfilene.
- *Loopen*, som er «protocoll» i programfilene.
- Statusene, se figur (4.1).

De tre delene utgjør hoveddelen til GRBL og inneholder sanntids-kontroll og ordrer. Etter at GRBL er initialisert sender *Loopen* GRBL til diverse stasuser. *Loopen* ser etter alarmer, jogging eller overskriving for å sette «flagg». Flagg aktiverer diverse statuser. For eksempel når statusen settes til «Jog», utfører ikke GRBL de samme planleggings-kravene som settes for normal kjøring. Dette skjer slik at joggingen av maskinen holdes i sanntid.

4.3 Oversetter

GRBL behandler G-kode gjennom fire steg:

- **Steg 1**: Initialiserer *parser block struct*. Her lagrer GRBL dataene den får fra lest G-kode.
  - Sjekker om det er JOG-kommando eller ikke.
- **Steg 2**: Importerer alle G-kode ord i blokk-linjen som for eksempel «G0».
  - Finner G- og M-kommandoer.
  - Sjekker G-kode instruksjonene først og deretter M-koden. Etterpå sjekkes andre bokstav-kommandoer.
  - Sjekker om det er feil i ordene.
  - Sjekker hvilke G-kode kommandoer det er og om de henger sammen.
- **Steg 3**: Sjekker kommandoer/verdier fra forrige steg for feil.
  - Hvis en feil finnes, dumpes blokken.
- **Steg 4**: Utfør koden!
  - Oppdaterer tilstand og utfører blokk i henhold til rekkefølgen blokken skal utføres i.

4.3.1 Bue-produksjon

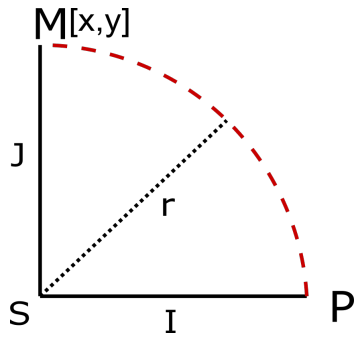
Den mest krevende bevegelsen å produsere for GRBL er buer. Arduino har begrenset minne og prosessor-kraft. Det betyr at mellomregninger og operasjoner ikke kan bruke for mange ressurser slik at *timeren* blir avbrutt. Blir *timeren* avbrutt fører det til at stegmotorene mister steg og i verste tilfelle utfører feil bevegelse. GRBL fikser problemet ved å bruke approksimasjoner som tar mindre tid og minne. GRBL får derfor utført de nødvendige utregningene mellom stegene og planlegger bevegelser.

Først dekodes G-koden og det sjekkes om buen er mulig å gjennomføre. Deretter forberedes variabler som brukes senere. Følgende to eksemper på G-kode gir resultatet i figur (4.2):

r: radius
S: sentrum (origo)
I og J: avvik fra sentrum
P: startpunkt
M: endepunkt

Eksempel 1:
G01 X10 Y0 F20 (startpunkt)
G03 X0 Y10 R10 (endepunkt og radius)

Eksempel 2:
G01 X10 Y0 F20 (startpunkt)
G03 X0 Y10 **I-10 J0**. (endepunkt og avviket fra buen til sentrum)



Figur 4.2: Eksempel 1 og 2 gir samme resultat. P er startpunkt og M er endepunkt. S er sentrum av sirkelen.

I eksempel 2 er $I = -10$ og $J = 0$. P er $(X, Y) = (10, 0)$. 10 er avviket fra S. Informasjonen forteller hvor S er i forhold til nåværende posisjon og hvor buen ender opp.

GRBL bruker likningene (4.1) og (4.2) for å finne I og J ved hjelp av r og Δ mellom P og M.

$$I = \frac{1}{2} \cdot \left(\Delta x - \frac{\Delta y \cdot \sqrt{4 \cdot r^2 - \Delta x^2 - \Delta y^2}}{\sqrt{\Delta x^2 + \Delta y^2}} \right) \tag{4.1}$$

$$J = \frac{1}{2} \cdot \left(\Delta y + \frac{\Delta x \cdot \sqrt{4 \cdot r^2 - \Delta x^2 - \Delta y^2}}{\sqrt{\Delta x^2 + \Delta y^2}} \right) \tag{4.2}$$

Dersom vi bruker eksempel 2 regnes r ut slik: $r = \sqrt{I^2 + J^2}$. Denne utregningen krever færre ressurser.

I GRBL er en bue bygget opp av mange korte linjer som kalles «segmenter». Formel (4.3) brukes for å dele opp buen. «bue-toleranse» bestemmes på forhånd og er vanligvis 0,002 mm. Bue-toleransen bestemmer lengden på segmentet. Lav toleranse gir en fin bue mens en høyere toleranse gir en grov bue.

$$\text{segmenter} = \frac{0.5 \cdot \theta \cdot r}{\sqrt{\text{bue-toleranse} \cdot (2 \cdot r - \text{bue-toleranse})}} \tag{4.3}$$

Vi bruker målene fra figur (4.2) i følgende eksempel: For å finne segmentene trenger vi vinkelen. GRBL setter opp en rekke variabler for dette:

- S (0, 0), sentrum
- Distansen fra S til P, ΔP (10 ,0)
- Distansen fra S til M, ΔM (0, 10)

$$\theta = tan^{-1}(\frac{10 \cdot 10 - 0 \cdot 0}{10 \cdot 0 + 0 \cdot 10}) = 1,57\ rad$$

1,57 rad brukes som vinkel i formel (4.3) med bue-toleranse på 0,002 mm:

$$\frac{0.5 \cdot 1,57 \cdot 10}{\sqrt{0,002 \cdot (2 \cdot 10 - 0,002)}} \approx 39 \text{ segmenter}$$

GRBL bruker vektorrotasjon med bruken av transformasjonsmatrisen, se formel (4.4). ΔM er den roterte vektoren, ΔP er original vektor og θ er den totale vinkelen ΔP roteres. For å lage buen utføres like mange vektorrotasjoner rundt S som det er segmenter. 39 segmenter tilsvarer 39 rotasjoner. Derfor skal ΔP roteres med $\theta/segmenter$.

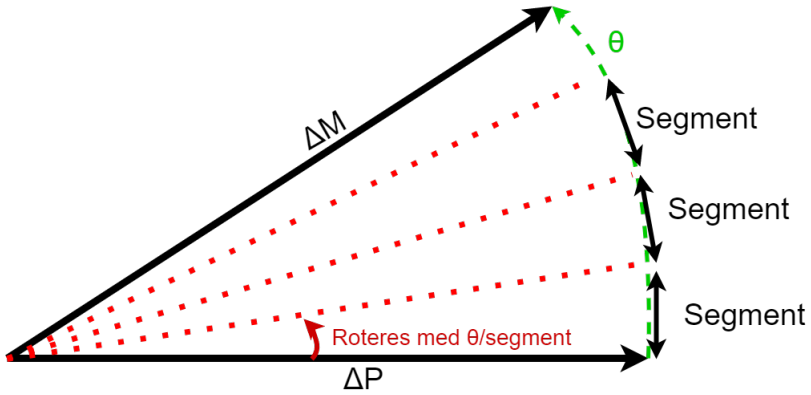
$$\Delta M = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \cdot \Delta P \tag{4.4}$$

Trigonometriske operasjoner er ressurskrevende og tar mellom 100 til 200 µs. GRBL bruker kun slike operasjoner for å fikse eventuelle avvik. Disse avvikene oppstår fordi GRBL bruker 2. og 3. ordens «taylor approksimasjon», se formel (4.5) og (4.6).

$$cos_{approx} = 1 - \frac{(\theta/segment)^2}{2} \tag{4.5}$$

$$sin_{approx} = (\theta/segment)^2 - \frac{\theta/segment)t^3}{6} \tag{4.6}$$

Figur (4.3) viser hvordan en bue genereres med flere segmenter.



Figur 4.3: Figuren viser hvordan ΔP roteres for å produsere en bue.

Ved å bruke formel (4.5) og (4.6) roteres vektoren med å regne ut formel (4.4). GRBL produsere deretter en linje til det gitte punktet og repetere prosessen til alle segmentene er fullførte.

4.3.2 Planner

Planleggeren (filnavn: *Planner*) til GRBL finjusterer individuelle bevegelser (blokker i GRBL) til en stor sammenhengende bevegelse. Først går den gjennom alle blokkene for å fylle opp *plan-bufferen*. Den fyller opp bufferen med G-kode linjer som sendes fra brukergrensesnittet. Deretter går den baklengs gjennom alle blokkene og passer på at slutt- og startfart for hver blokk er kompatible. På denne måten sikrer GRBL at bevegelsene vil kjøre etter hverandre uten å stoppe. Dette gir mindre produksjonstid og kontinuerlig bevegelse.

4.4 Bakgrunns-prosesser

Bakgrunns-prosessene er systemer og rutiner med en spesifikk oppgave. I GRBL er dette filer med navn som henger sammen med oppgaven de har (feks: *Stepper*).

4.4.1 Bresenham's algoritme

Informasjonen brukt i dette avsnittet er funnet hos Wikipedia [24] og i kildekoden [21] til GRBL.

Dette er en algoritme for å tegne linjer som bruker forskjellig antall steg i X- og Y-aksen. For å bruke denne er det satt to krav:

1. Den ene aksen må ha flere steg enn den andre. Dvs $x_1 - x_0$ må være større eller midre enn $y_1 - y_0$.
2. Linjen må ha en stigning større enn eller mindre enn 1. Dersom stigningen er mindre enn 1 er X-aksen dominerende og i motsatt tilfelle

er Y-aksen dominerende.

Den dominerende aksen vil alltid ta et steg på linjen. Bresenham-algoritmen brukes for å finne ut om den ikke-dominerende aksen skal ta et steg eller ikke. Alt baseres rundt formelen for rette linjer:

$$y = mx + b,\ der\ m = \frac{\Delta y}{\Delta x}$$

Bresenham viser at vi ønsker å velge steg som har kortest distanse til linjen som skal tegnes. For å beholde utregningene med kun heltall blir differansen (D) regnet ut slikt:

$$\Delta y = y_1 - y_0$$

$$\Delta x = x_1 - x_0$$

$$D = (2 \cdot \Delta y) - \Delta x$$

Kode nedenfor viser en versjon av bresenham-algoritmen hvor X er dominerende akse og Y øker eller minsker med 1, eller holdes statisk.

Pseudokode 2 Bresenham-algoritmen

```
function plotLineLow(x0, y0, x1, y1)
    dx = x1 - x0
    dy = y1 - y0
    yi = 1
    if dy < 0 then
        yi = -1
        dy = -dy
    end if
    D = (2 * dy) - dx
    y = y0
    for x from x0 to x1 do
        plot(x, y)
        if D > 0 then
            y = y + yi
            D = D + (2 * (dy - dx))
        else
            D = D + 2*dy
        end if
    end for
end function
```

Pseudokode (2) er en imitasjon av hva som er implementert i GRBL. GRBLs variasjon inneholder flere deler slik at det fungerer for alle aksene.

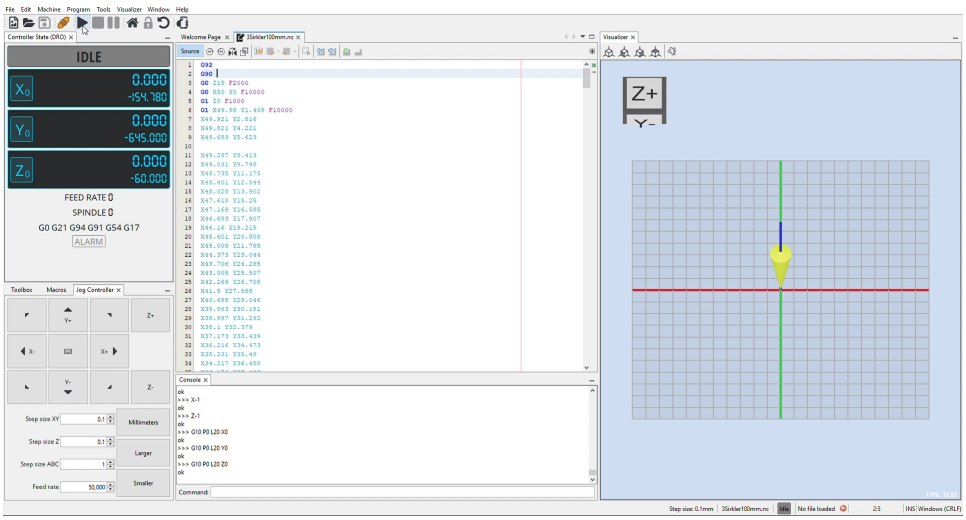
En utfordring med bresenham-algoritmen er multi-akse bevegelser. Ved den ikke-dominante aksen oppstår det ujevne bevegelser som skaper vibrasjoner i motorene. Dette fikses med GRBLs egen implementering av «adaptive multi-axis smoothing system», forkortet AMASS. AMASS forbedrer oppløsningen til bresenham-algoritmen ved lave frekvenser uten å vrenge signalet til algoritmen.

Til sammen dannes en avbruddsmetode som ikke utfører avansert mattematikk. Med bresenham-algoritmen bestemmes det hvilken motor som mottar pulser. Den første avbruddsmetoden blir deretter støttet opp av enda en avbruddsmetode som har som hovedoppgave å resette utgangs-portene til motorene.

4.5 Kommunikasjon og innstillinger

Universal Gcode Sender, forkortet UGS, er et brukergrensesnitt som gir mulighet for sending av G-kode-kommandoer til CNC-maskinen. GRBL er en åpen kildekode-programvare som er utviklet for å kjøre på en Arduino-basert mikrokontroller.

I UGS endres de nødvendige innstillingene: *hard-limits*, *soft-limits*, *homing*, *feedrate*, *acceleration* osv. For å endre alle innstillinger i GRBL, kan en liste for innstillinger bli lastet opp. Innstillingene som ble brukt for vårt oppsett av GRBL finnes som vedlegg E i tabell (E.1).



Figur 4.4: Brukergrensesnittet UGS.

Kapittel 5

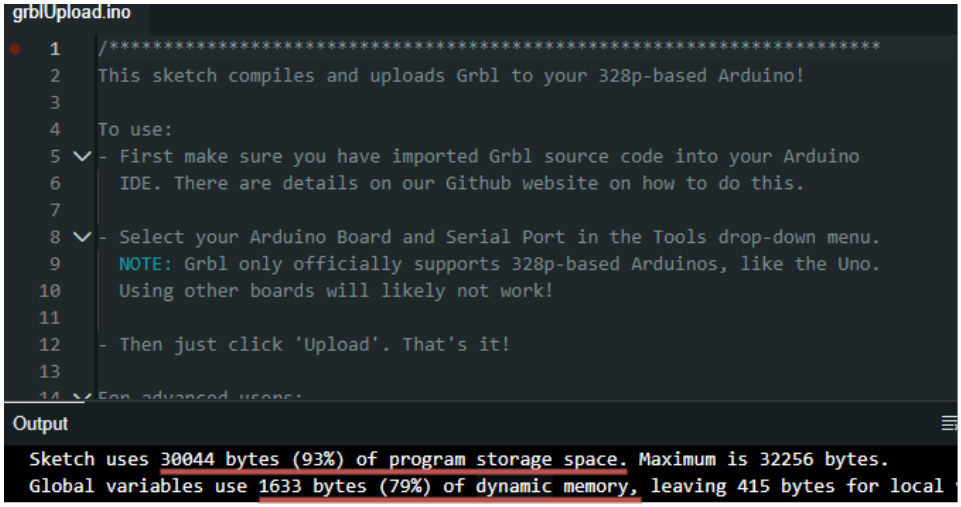
Resultat og sammenligning

Det å utvikle en maskinkontroller er en kompleks og tidskrevende prosess. Vi har valgt å sammenligne maskinkontrolleren og brukergrensesnittet med GRBLs maskinkontroller og UGS som brukergrensesnitt.

5.1 Fordeler og ulemper med maskinkontrollerene

Ettehvert som flere frihetsgrader og stegmotorer legges til blir kodingen mer avansert. GRBL er utviklet over syv år [21]. En maskinkontroller utviklet over fire måneder er ikke like avansert. Vi har løst oppgaven med bruk av tre akser.

En viktig fordel med GRBL er at det er en ferdiglagd maskinkontroller som er kompatibel med Fusion 360. Ulempen er at GRBL er vanskelig å modifisere og tar mye minne- og lagringsplass på Arduino, se figur (5.1). GRBL tar 93% av programlagringsplassen og 79% av dynamisk minne, mens vår maskinkontroller tar 43% av programlagringsplassen og 32% av dynamisk minne. GRBL må også ha nok plass til midlertidig G-kode som er grunnen til at det er vanskelig å legge til flere funksjoner i GRBL. Løsningen er eventuelt å oppgradere maskinvaren fra Ardiono Uno til Arudino Mega [4].



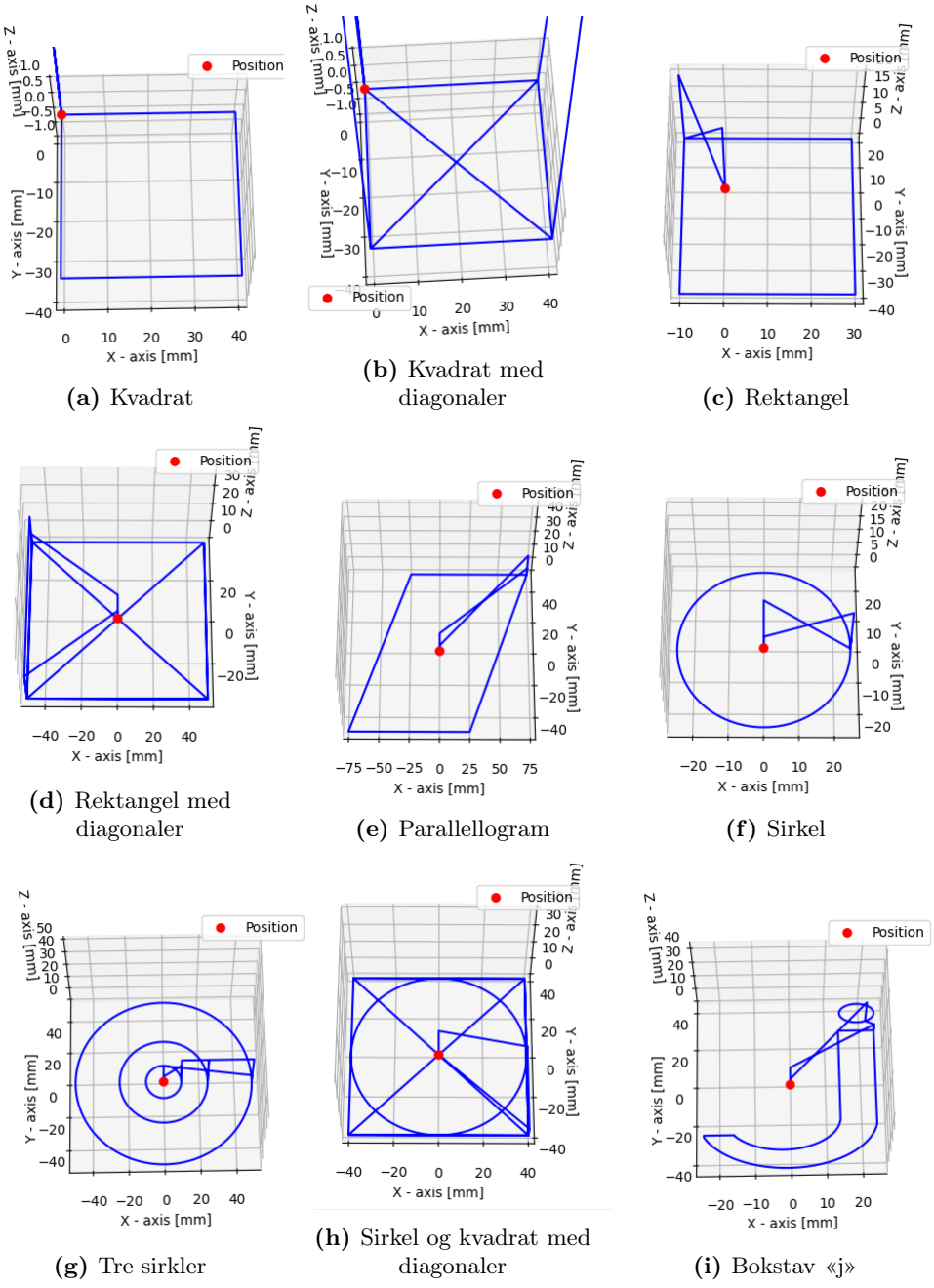
Figur 5.1: GRBL utnytter maskinvaren ettersom minnebruk og lagringsplassen nesten er fullt.

5.2 Sammenligning av resultatet

Vi sammenligner ved å teste vår maskinkontroller mot GRBL ved å måle utførelsestiden av samme G-kode fil. Vår maskinkontroller og GRBL har lik hastighet, forkortet F, med enheten millimeter per minutt [mm/min].

5.2 Sammenligning av resultatet

G-kode filer testes og figurene nedenfor visualiserer G-koden:



(a) Kvadrat

(b) Kvadrat med diagonaler

(c) Rektangel

(d) Rektangel med diagonaler

(e) Parallellogram

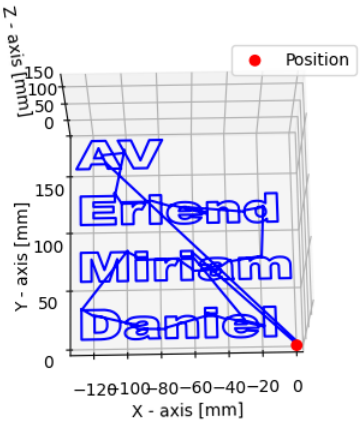
(f) Sirkel

(g) Tre sirkler

(h) Sirkel og kvadrat med diagonaler

(i) Bokstav «j»

Figur 5.2: Disse G-kode figurene blir testet på begge maskinkontrollerene.



Figur 5.3: G-kode «Navn» testet medmaskinkontrollerene.

Testene utføres ved å laste inn G-kode filen i brukergrensesnittene. Tiden starten når «Play» (▶) trykkes i UGS eller når «Send G-Code» trykkes i vårt GUI. Tiden stopper når maskinen er ferdig med kjøringen og tilbake til logisk nullpunkt. Resultatet er vist i tabell (5.1). G-kode filene er tilgjengelig i vedlegg D.

Tabell 5.1: G-kodene fra figur (5.2) og (5.3) testes med begge maskinkontrollerene. Resultatet er en sammenligning av utførelsestid i sekunder [s].

G-kode	GRBL, Tid [s]	Vår, Tid [s]
Kvadrat	26	26
Kvadrat med diagonaler	49	44
Rektangel	34	29
Rektangel med diagoneler	95	93
Parallellogram	49	62
Sirkel	21	149
Tre sirkler	63	441
Sirkel og kvadrat med diagonaler	98	217
Bokstav «j»	42	121
Navn	417	1425

Resultatet viser at begge maskinkontrollerene har omtrent lik utførelsestid ved produksjon av firkanter og diagonaler. Når buer og/eller sirkler produseres bruker vår maskinkontroller mer tid. Tiden kan forbedres ved å implementere en «planlegger», se delkapittel 4.3.2.

Ved kjøring av lange og kompliserte G-kode filer som «Navn» brukte vår maskinkontroller nesten 24 minutter. GRBL brukte omtrent 7 minutter. Dette er ikke godt nok for en bedrift. Maskinkontrolleren er ikke spesielt konkurransedyktig ved tidskrevende prosjekter.

5.3 Forslag til videre arbeid og utvikling

Det finnes flere måter å forbedre og videreutvikle maskinkontrolleren og CNC-maskinen på:

- Maskinvare: Ved å oppgradere fra Arduino Uno til Arduino Mega kan flere motorer og sensorer kobles til. I tillegg utvides minnekapasiteten.
- Planlegging: Vår maskinkontroller har lang utførelsestid. Videreutvikling er å planlegge programutførelsen flere linjer med G-kode i forveien, slik at maskinen ikke stopper etter hver linje. Planlegging fører til at maskinen får økt produktivitet og effektivitet.
- Bedre verktøy: Bytte ut pennen og pennholderen med en bedre og mer stabil penn og pennholder, slik at tegningene blir jevnere. Etterhvert bytte ut pennen med et freseverktøy.
- Bedre materialer: Holde sammen CNC-maskinens struktur med aluminium istedenfor plast. Det gjør maskinen mer stabil og slitesterk.
- Større arbeidsområde: Utviding av arbeidsområdet kan gjøre maskinen mer allsidig og øke potensialet.
- Tilpasningsevne: Legge til flere akser, sensorer og andre modifikasjoner. Det gjør det mulig å bearbeide flere typer materialer og utføre andre oppgaver.

Kapittel 6

Konklusjon

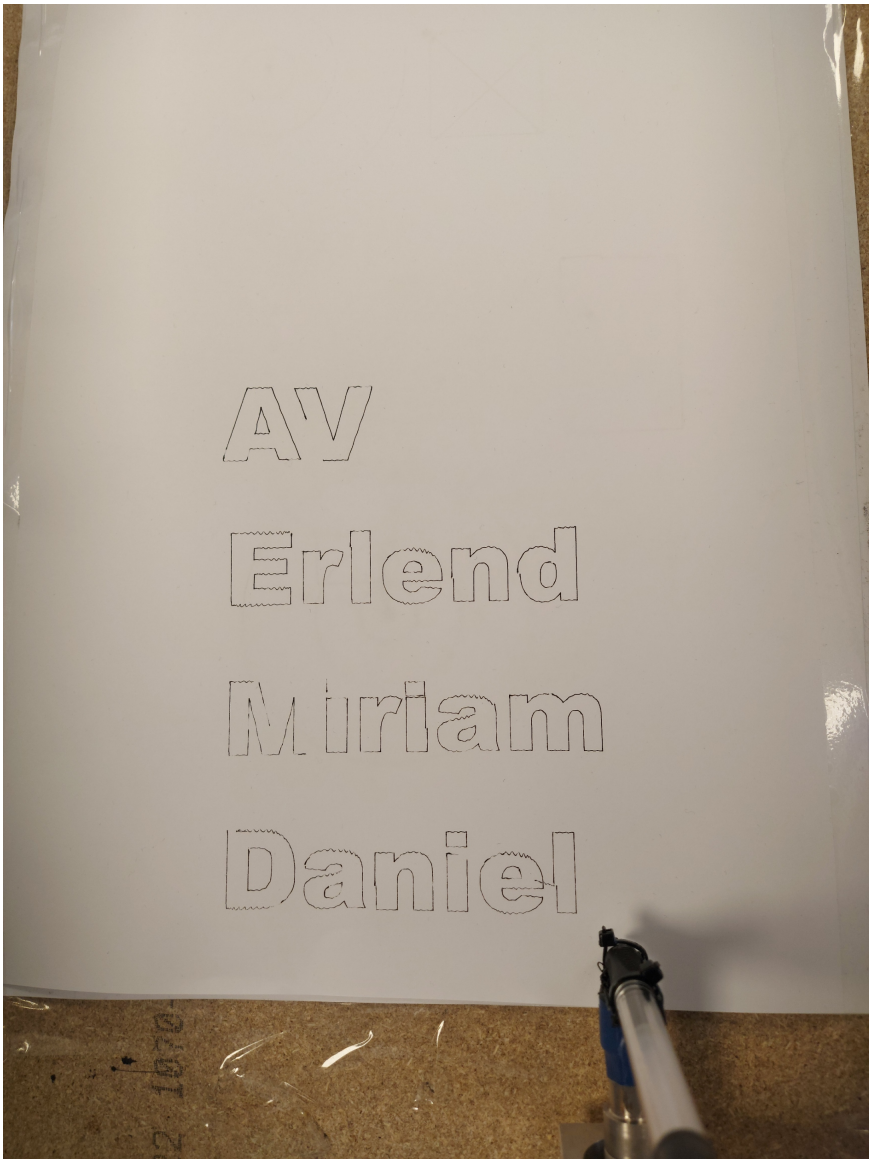
CNC-maskinen er en del av en avansert teknologi som har revolusjonert produksjonsindustrien. Ved å bruke datamaskiner med avansert programvare kan maskinen automatisere prosessen, redusere arbeidskostnader og produsere deler med høy presisjon.

I rapporten har vi designet, bygget og testet maskinkontroller og egenprodusert CNC-maskin. Vi har fått en dyp og god forståelse for hvordan en CNC-maskin fungerer og hva som må til for å sette sammen maskinen fra bunnen av. I tillegg har vi lært hvordan en egen maskinkontroller kan lages, og hvor krevende det blir jo mer komplisert CNC-maskinen fysisk blir.

Arbeidet viser at CNC-maskiner kan bygges selv med grunnleggende kunnskaper innenfor elektroteknikk, automatisering og programmering. Likevel er prosessen avansert og et godt resultat krever mye arbeid. Derfor er det brukt en ferdiglagd kontroller GRBL og et ferdig brukergrensesnitt UGS som fungerer utmerket til en selvlagd CNC-maskin.

I tillegg har vi foreslått videre utviklingsmuligheter til maskinen som flere frihetsgrader, oppgradering av maskinvare, bedre planlegging og automatisk verktøyskiftning.

Link til bonusvideo av kjøring av G-kodefilen «Navn»: YouTube - UiS B.Sc: CNC Fres Bonus!



Bibliografi

[1] Aliexpress: Mechanical endstop limit switch. Hentet 21 Februar 2023. URL: https://www.aliexpress.com/item/32980508075.html.

[2] Makerbot's hjemmeside. Hentet 21 Februar 2023. URL: https://www.makerbot.com/.

[3] Arduino. Arduino ide software. Hentet 14. mars 2023. URL: https://www.arduino.cc/en/software.

[4] Arduino. Arduino mega. Hentet 12. mai 2023. URL: https://store.arduino.cc/products/arduino-mega-2560-rev3#looxReviews.

[5] Arduino. Arduino uno. Hentet 13. mars 2023. URL: https://store.arduino.cc/products/arduino-uno-rev3#looxReviews.

[6] Atmel. Atmega328p datasheet. Hentet 5. mai 2023. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[7] Atmel. Avr446: Linear speed control of stepper motor, 2006. Hentet 13. april 2023. URL: https://ww1.microchip.com/downloads/en/Appnotes/doc8017.pdf.

[8] Electronicwings. Usart in arduino uno. Hentet 21. februar 2023. URL: https://www.electronicwings.com/arduino/usart-in-arduino-uno.

[9] Lars Hallstrøm Eriksen. Verktøymaskiner, Jan 2023. Hentet: 16.01.23. URL: https://snl.no/verktÃÿymaskiner.

[10] Allan R. Hambley. *16.5 Stepper motors and brushless DC motors*, page 847–848. Pearson, 7 edition, 2019. Electrical engineering: Principles and applications, NY NY.

[11] John D. Hunter and Michael Droettboom. matplotlib 3.7.1. Hentet 3. februar 2023. URL: https://pypi.org/project/matplotlib/.

[12] islproducts. Stepper motor fundamentals, 2022. Bilde hentet 23. mars 2023. URL: https://islproducts.com/design-note/stepper-motor-fundamentals/.

[13] Chris Liechti. Python serial port extension. Hentet 3. februar 2023. URL: https://pypi.org/project/pyserial/.

[14] Francis Nickols and Yueh Jaw Lin. *Kapittel 4: Theory III The Stepper Motor and Its Control*, pages 50–54. Elsevier Science & Technology, 2018. Creating Precision Robots: A Project-Based Approach to the Study og Mechatronics and Robotic.

[15] OpenBuilds. Electromagnetic interference (emi). Hentet 28. Mars 2023. URL: https://docs.openbuilds.com/doku.php?id=docs:blackbox-4x:faq-emi.

[16] PySimpleGUI. Pysimplegui 4.60.4. Hentet 3. februar 2023. URL: https://pypi.org/project/PySimpleGUI/.

[17] Python. Python nettside. Hentet 1. februar 2023. URL: https://www.python.org/.

[18] RaspberryPi. Raspberry pi 4 model b. Hentet 13. mars 2023. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[19] RS. Sanyo denki 103h7 series hybrid, single shaft stepper motor, 24 v, bipolar, 1.65nm torque, 1.8°, 6.35mm shaft, 2023. Nedlastet PDF datablad, hentet: 13. Januar 2023. URL: http://no.rs-online.com/web/p/stepper-motors/1366716?cm_mmc.

[20] STEPPERONLINE. Nema 17 bipolar 59ncm (84oz.in) 2a 42x48mm 4 wires w/ 1m cable & connector, 2023. Nedlastet PDF datablad, Hentet: 19. Januar 2023. URL: https://www.omc-stepperonline.com/nema-17-bipolar-59ncm-84oz-in-2a-42x48mm-4-wires-w-1m-cable-connector-17hs19-2004s1.

[21] Sungeun «Sonny» Jeon and Simen Svale Skogsrud. Grbl hjemmeside. Hentet 16. mars 2023. URL: https://github.com/gnea/grbl.

[22] Sungeun «Sonny» Jeon and Simen Svale Skogsrud. Grbl interface basics. Hentet 14. mars 2023. URL: https://github.com/gnea/grbl/wiki/Grbl-v1.1-Interface.

[23] w2aew. Basics of ferrite beads, 2023. Hentet 1. mai 2023. URL: https://www.youtube.com/watch?v=81C4IfONt3o.

[24] Wikipedia. Bresenham's line algorithm, 2023. Hentet 21. februar 2023. URL: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm.

Vedlegg A

Kode for brukergrensesnitt.

Koden nedenfor er skrevet i Python 3.10. Tilgjengelig her: 

NB! Filene kan bare hentes med Adobe Acrobat.

Kode A.1: Brukergrensesnitt

```
import PySimpleGUI as sg
import serial
import time
import matplotlib
from matplotlib.backends.backend_tkagg import ...
    FigureCanvasTkAgg
import matplotlib.pyplot as plt
from threading import Thread
# Imports from other files
import serial.tools.list_ports  # Finds Arduino Uno Port

# Runtime error:
# "Main thread is not in main loop
plt.switch_backend('agg')

# Windows:
# deviceID = "COM10"
# Rasbian:
# deviceID = "/dev/ttyACM0"
# Universal:
# deviceID = ["COM10"]
def check_ports():
    deviceID = []
    comValue = []
    ports = list(serial.tools.list_ports.comports())
    if not ports:
        deviceID = ["Null"]
    else:
        for p in ports:
            portInfo = f"Device: {p[1]}"
            print(portInfo)
            messages.append(portInfo)
            deviceID.append(p[1])
            comValue.append(p[0])
    return deviceID, comValue

```

```
37
38  # Baud Rate (how much data can you transfer per sec)
39  # Standard: 9600, highest: 115200
40  baudrate = 115200
41
42  # Sg theme
43  sg.theme('Default1')
44
45  # Variables:
46  cordX = [0.00, 0.00]
47  cordY = [0.00, 0.00]
48  cordZ = [0.00, 0.00]
49  pos = [0.00, 0.00, 0.00]
50  messages = []
51  # The list of choices that are going to be searched
52  choices = sorted(['G0 X0 Y0 Z0 F5000', 'G0 X100 Y100 Z20 ...
        F5000', 'M100 - Help', 'M120 - Home', 'M114 - Where',
53                    'M17 - Enable Motors', 'M18 - Disable ...
        Motors', 'G90 - Absolute', 'G91 - Incremental', 'G92 - ...
        Set Origo'])
54
55  # ========================
56
57
58  def sort_serial(msg):
59      # Sort output from arduino
60      printable = []
61      msg.replace("_", " ")
62      #print(msg)
63      cmdValueM = msg.find("M")
64      cmdValueG = msg.find("G")
65      doubleV = msg.find("<<")
66      if cmdValueM > 0 or cmdValueG > 0:
67          if doubleV > 0:
68              cmd = msg[:doubleV-1].replace("<","")
69          else:
70              cmd = ""
71      else:
72          if msg.find("<") > 0:
73              msg = msg.replace("<","")
74          cmd = ""
75
76      printable.append(cmd)
77      doubleV += 1
78      nymsg = msg[doubleV:]
79      for i in range(nymsg.count("|")):
80          msgValue = nymsg.find("|")
81          printable.append(nymsg[:msgValue])
82          msgValue += 1
83          nymsg = nymsg[msgValue:]
84      return printable
85
86
87  # ========================
88
89
90  def draw_figure(figure, canvas):
91      figure_canvas_agg = FigureCanvasTkAgg(figure, canvas)
92      figure_canvas_agg.draw()
93      figure_canvas_agg.get_tk_widget().pack(side='top', ...
        fill='both', expand=1)
94      return figure_canvas_agg
95
96
97  # ========================
98
99
100 def append_message(txt):
101     global messages
102     len_messages = str(len(messages) + 1)
103     txt = len_messages + ": " + txt
104     messages.append(txt)
105
106 # ========================
107
108
109 def setupSerial(baudRate, serialPortName):
110     global serialPort
111     try:
112         if serialPort.isOpen():
113             serialPort.close()
114             time.sleep(1)
115     except Exception as e:
116         print("Error: {}".format(e))
```

```
117     try:
118         serialPort = serial.Serial(port=serialPortName, ...
    baudrate=baudRate, timeout=0, rtscts=True)
119         while not serialPort.inWaiting():
120             break
121         print("Serial port " + serialPortName + " opened  ...
    Baudrate " + str(baudRate))
122         try:
123             if serialPort.isOpen():
124                 intro3 = "{} ...
    connected!".format(serialPort.port)
125                 serialPort.flushInput() # Remove startup text
    from Arduino
126                 print(intro3)
127                 append_message(intro3)
128                 time.sleep(2)
129             else:
130                 intro3 = "Connection failed."
131                 append_message(intro3)
132         except Exception as e:
133             print("Error: {}".format(e))
134             append_message(str(e))
135     except FileNotFoundError:
136         serialPort.close()
137         intro4 = "{} closed!".format(serialPort.port)
138         print(intro4)
139         append_message(intro4)
140     except Exception as e:
141         print("Error: {}".format(e))
142         append_message(str(e))
143
144
145 # ========================
146
147
148 def sendToArduino(stringToSend, window):
149     # this adds the start- and end-markers before sending
150     global serialPort
151     serialPort.flushInput()
152     stringWithMarkers = stringToSend + "\n"
153     # print(stringWithMarkers)
154
155     # serialPort.write(str.encode(wakeup))
156
157     serialPort.write(stringWithMarkers.encode('utf-8'))  #
    encode needed for Python3
158     serialPort.flushOutput()
159     print(stringWithMarkers.encode('utf-8'))
160     append_message("Sending to Arduino: " + stringToSend)
161     window['-OUTPUT-'].update('\n'.join(messages))
162
163     # Canvas updater
164     x, y, z = findCords(stringToSend)
165     if x != 0:
166         cordX.append(float(x))
167     else:
168         cordX.append(cordX[-1])
169     if y != 0:
170         cordY.append(float(y))
171     else:
172         cordY.append(cordY[-1])
173     if z != 0:
174         cordZ.append(float(z))
175     else:
176         cordZ.append(cordZ[-1])
177
178
179 # ==================
180
181
182 def arduino_listen(window):
183     global serialPort
184     serialPort.flushInput()
185     serialPort.flushOutput()
186     msg = ""
187     counterStop = 0
188     try:
189         while True:
190             try:
191                 serial_data = ...
    serialPort.read().decode("utf-8")
192                 msg_get = serial_data.strip()
```

```
193
194                     if msg_get.find('>') ≥ 0:
195                         print("REC: " + msg_get + "")
196                         messages.append("Arduino - Done")
197                         ...
    window['-OUTPUT-'].update('\n'.join(messages))
198                         break
199                     elif msg_get == "":
200                         counterStop +=1
201                     else:
202                         msg += msg_get
203                         print("MSG: " + msg_get + "")
204                     if window.is_closed() or counterStop == 1000:
205                         break
206                 except Exception as e:
207                     print("Mottat melding feilet: ")
208                     print(e)
209                     break
210             #print(msg)
211             newmsg = sort_serial(msg)
212             for newermsg in newmsg:
213                 messages.append(newermsg)
214             window['-OUTPUT-'].update('\n'.join(messages))
215         except NameError:
216             error1 = "Can't connect to Arduino: No port found"
217             print(error1)
218             append_message(error1)
219         except Exception as e:
220             error2 = "Can't connect to Arduino: Something else ...
    went wrong: {}".format(e)
221             print(error2)
222             append_message(error2)
223
224 # ==================
225
226
227 def sendToArduino2(stringToSend, window):
228     # this adds the start- and end-markers before sending
229     global serialPort
230     serialPort.flushInput()
231     stringWithMarkers = stringToSend + "\n"
232     serialPort.write(stringWithMarkers.encode('utf-8'))  #
    encode needed for Python3
233     serialPort.flushOutput()
234
235
236 # ==================
237
238
239 def arduino_listen2(window):
240     global serialPort
241     serialPort.flushInput()
242     serialPort.flushOutput()
243     msg = ""
244     counterStop = 0
245     try:
246         while True:
247             try:
248                 serial_data = ...
    serialPort.read().decode("utf-8")
249                 msg_get = serial_data.strip()
250
251                 if msg_get.find('>') ≥ 0:
252
253                     break
254                 elif msg_get == "":
255                     counterStop +=1
256                 else:
257                     msg += msg_get
258
259                 if window.is_closed() or counterStop == 1000:
260                     break
261             except Exception as e:
262                 pass
263
264
265         all_MSG2 = []
266         for newerermsg in msg:
267             all_MSG2.append(newererermsg)
268         newmsg = sort_serial(msg)
269         all_MSG = []
270         for newermsg in newmsg:
271             #messages.append(newermsg)
```

```
272             all_MSG.append(newermsg)
273         return all_MSG, all_MSG
274
275     except NameError:
276         pass
277         #error1 = "Can't connect to Arduino: No port found"
278
279
280     except Exception as e:
281         pass
282         #error2 = "Can't connect to Arduino: Something else
    went wrong: ".format(e)
283
284
285 # ==================
286
287
288 def showMsg(msg, window):
289     for i in msg:
290         append_message(i)
291     window['-OUTPUT-'].update('\n'.join(messages))
292
293 # ==================
294
295
296 def findCords(msg):
297     print(msg)
298     if msg.find("X") ≥ 0:
299         corX = ""
300         for i in msg[msg.find("X")+1:]:
301             if i.isnumeric() or i == "." or i == "-":
302                 corX += i
303             else:
304                 break
305     else:
306         corX = cordX[-1]
307     if msg.find("Y") ≥ 0:
308         corY = ""
309         for i in msg[msg.find("Y")+1:]:
310             if i.isnumeric() or i == "." or i == "-":
311                 corY += i
312             else:
313                 break
314     else:
315         corY = cordY[-1]
316     if msg.find("Z") ≥ 0:
317         corZ = ""
318         for i in msg[msg.find("Z")+1:]:
319             if i.isnumeric() or i == "." or i == "-":
320                 corZ += i
321             else:
322                 break
323     else:
324         corZ = cordZ[-1]
325     print("X{}, Y{}, Z{}".format(corX, corY, corZ))
326     return corX, corY, corZ
327
328
329 # ==================
330
331
332 def sendGcode(file, window):
333     global serialPort, cordX, cordY, cordZ
334     try:
335         f = open(file, "r")
336         append_message("Reading file: " + f.name)
337         serialPort.write("G92 G90 M17".encode("utf-8"))  #
    Reset current position and set it to 0,0,0
338         serialPort.flushInput()  # Flush startup text in
    serial input
339         time.sleep(1)  # Wait for grbl to nitialize
340         l_count = 0
341         cordX = [0.00, 0.00]
342         cordY = [0.00, 0.00]
343         cordZ = [0.00, 0.00]
344
345         for line in f:
346             serialPort.flushInput()  # Flush startup text in
    serial input
347             #time.sleep(0.1)
348             linje = line.strip()  # Strip all EOL characters
    for consistency
349             print('Sending: ' + linje)
350             repeat = True
351             while repeat:
352                 try:
```

```
353                     sendToArduino(linje, window)
354                     messages.append("Arduino - Recived and ...
    working: " + linje)
355     window['-OUTPUT-'].update('\n'.join(messages))
356                     repeat = False
357                 except Exception as e:
358                     if not serialPort.isOpen():
359                         repeat = False
360                     error2 = "Repeating send, Arduino ...
    didn't recive: {}".format(e)
361                     print(error2)
362                     append_message(error2)
363             try:
364                 while 1:
365                     grb_out = ...
    serialPort.read().decode("utf-8")
366                     grbl_out = grb_out.strip()  # Wait for
    grbl response with carriage return
367                     if grbl_out.find('>') >= 0:
368                         print("REC<" + str(l_count) + ": " ...
    + grbl_out + "")
369                         messages.append("Arduino - Done")
370     window['-OUTPUT-'].update('\n'.join(messages))
371                         break
372                     elif grbl_out == "":
373                         pass
374                     else:
375                         print("MSG: " + grbl_out + "")
376                 l_count += 1
377             except Exception as e:
378                 error2 = "Continuing, but didn't recived ...
    stopbit for Arduino: {}".format(e)
379                 print(error2)
380                 append_message(error2)
381         messages.append("Done sending: " + file)
382         window['-OUTPUT-'].update('\n'.join(messages))
383     except Exception as e:
384         error2 = "Can't upload Gcode: {}".format(e)
385         print(error2)
386         append_message(error2)
387
388 # ========================
389 # Load G-code visual
390
391
392 def append_x(string):
393     global xarr, yarr
394     pulsX = float(string[1:])
395     xarr.append(pulsX)
396
397
398 def append_y(string):
399     global xarr, yarr
400     pulsY = float(string[1:])
401     yarr.append(pulsY)
402
403
404 def append_z(string):
405     global xarr, yarr, zarr
406     pulsZ= float(string[1:])
407     zarr.append(pulsZ)
408
409
410 def animer_filen(fil):
411     gkodefil = open(fil, 'r')
412     linjer = gkodefil.readlines()
413     xyklar = False
414     xyzklar = False
415     zklar = False
416     xklar = False
417     yklar = False
418     z1 = False
419     z2 = False
420     teller = 0
421     teller1 = 0
422     teller2 = 0
423     teller3 = 0
424     for line in range(0, len(linjer)):
425         linje = linjer[line]
426         linjesplit = linje.split()
```

```
427             lengde = len(linjesplit)
428
429             if 'X' in linje:
430                 if 'Y' in linje:
431                     if 'Z' in linje:
432                         xyzklar = True
433                         xyklar = False
434                     if not 'Z' in linje:
435                         xyklar = True
436             else:
437                 xyklar = False
438                 xyzklar = False
439
440             if 'X' in linje and not 'Y' in linje and not 'Z' ...
    in linje:
441                 xklar = True
442             if 'Y' in linje and not 'X' in linje and not 'Z' ...
    in linje:
443                 yklar = True
444             if 'Z' in linje and not 'Y' in linje and not 'X' ...
    in linje:
445                 zklar = True
446
447             for i in range(0, lengde):
448                 string = linjesplit[i]
449                 if xklar or yklar or zklar:
450                     forrigex = xarr[-1]
451                     forrigey = yarr[-1]
452                     forrigez = zarr[-1]
453                     if string[0] == 'X':
454                         teller1 += 1
455                         append_x(string)
456                         yarr.append(forrigey)
457                         zarr.append(forrigez)
458                     if string[0] == 'Y':
459                         teller1 += 1
460                         append_y(string)
461                         xarr.append(forrigex)
462                         zarr.append(forrigez)
463                     if string[0] == 'Z':
464                         teller1 += 1
465                         append_z(string)
466                         xarr.append(forrigex)
467                         yarr.append(forrigey)
468                 elif xyzklar and not xyklar:  # Dersom alle i
    linjen: (ofte der de aldri alle på samme linje!)
469                     if string[0] == 'X':
470                         append_x(string)
471                     if string[0] == 'Y':
472                         append_y(string)
473                     if string[0] == 'Z':
474                         append_z(string)
475                 elif xyklar and (not xyzklar and not xklar and ...
    not yklar and not zklar):
476                     if string[0] == 'X':
477                         append_x(string)
478                         z1 = True
479                     if string[0] == 'Y':
480                         append_y(string)
481                         z2 = True
482                     if z1 and z2:
483                         teller3 += 1
484                         forrigez = zarr[-1]
485                         append_z(str(forrigez))
486                         z1 = False
487                         z2 = False
488                 elif string[0] == 'G':
489                     pass
490                     # print('G-kategori:', string)
491
492                 elif string[0] == 'Z':
493                     pass
494                     # print('Z-kategori:', string)
495                     # print('Flytt z-akse motoren lengde: ',
    string[1:])
496                 elif string[0] == 'F':
497                     pass
498                     # print('F-kategori:', string)
499                     # print('Farten er: ', string[1:])
```

```
500
501             xyklar = False
502             xyzklar = False
503             xklar = False
504             yklar = False
505             zklar = False
506         return xarr, yarr, zarr
507
508     # ========================
509
510
511     def main():
512         # Variables:
513         canvas_update = 100
514         input_width = 30
515         num_items_to_show = 2
516         lenThreads = 0
517
518         # Canvas
519         firstTime = True
520         zoom2DPlot = False
521
522
523         # Threads
524         threads = []  # Creates an array for threads
525
526         # Start Program Loop
527         intro1 = "Starting Program"
528         print(intro1)
529         append_message(intro1)
530
531         # Check Ports
532         intro2 = "Found following ports:"
533         print(intro2)
534         append_message(intro2)
535         deviceID, comValue = check_ports()
536         # ---- Menu Definition ----
537         menu_def = [
538             ['&File', ['&Open     Ctrl-O', '&Save       ...
        Ctrl-S', '&Properties', 'E&xit']],
539             ['&Edit', ['&Paste', ['Special', 'Normal', ], ...
        'Undo', 'Options::this_is_a_menu_key'], ],
540             ['&Toolbar', ['---', 'Command &1', 'Command &2',
541                           '---', 'Command &3', 'Command &4']],
542             ['&Help', ['&About...']]
543         ]
544
545         layout = [
546             [sg.Menu(menu_def, tearoff=True, font='_ 12', ...
        key='-MENUBAR-')],
547             [sg.Text('Communication with Arduino:')],
548             [sg.OptionMenu(deviceID, ...
        default_value=deviceID[0], s=(25, 2), key='-OPTION-'), ...
        sg.Button('Refresh Port'),
549              sg.Button("Connect", key='-BC-'), ...
        sg.Button('Clear Chat')],
550             [sg.Multiline(size=(110, 15), autoscroll=True, ...
        auto_refresh=True, key='-OUTPUT-'), ...
        sg.Column([[sg.Frame('Commands:',
551                                                                ...
        [[sg.Column([[sg.Listbox([choices[i] for i in ...
        range(len(choices))], key='-CMD-LIST-',size=(100, ...
        80))]],scrollable=True,
552                                                                ...
553                     vertical_scroll_only=True,                 ...
554                     size=(400, 300))]])]], pad=(0, 0)),],
555             [sg.CB('Ignore AutoCorrect Case', k='-IGNORE ...
        CASE-', default=True)],
556             [sg.Text('Input Command:'), sg.Text('Import G-code ...
        File:', pad=(670, 0))],
557             [sg.Input(size=(input_width, 1), ...
        enable_events=True, key='-IN-'), sg.Button('Go'),
558              sg.pin(sg.Col([[sg.Listbox(values=[], ...
        size=(input_width, num_items_to_show), enable_events=True,
                                           key='-BOX-', ...
        select_mode=sg.LISTBOX_SELECT_MODE_SINGLE, ...
        no_scrollbar=False)]],
559                            key='-BOX-CONTAINER-',
560                            visible=True)), ...
        sg.Combo(sorted(sg.user_settings_get_entry('-filenames-', ...
        [])),
```

```
561                 ...
        default_value=sg.user_settings_get_entry('-last ...
        filename-', ''), size=(40, 10), pad=(20, 10),
562                 key='-FILENAME-', enable_events=True, ...
        tooltip='Input file path'), sg.FileBrowse(),
563         sg.B('Clear History'), sg.Button('Load G-code', ...
        bind_return_key=True),
564         sg.Button('Send G-code')],
565         [sg.HorizontalSeparator()],
566         [sg.VStretch(),sg.Column([[sg.Frame('Map 2D:', ...
        [[sg.Column([[sg.Canvas(size=(1, 1), key='-CANVAS-', )]],
567                                                     ...
        size=(400, 400))]])]], pad=(0, 0)),
568         sg.Column([[sg.Frame('Map 3D:', ...
        [[sg.Column([[sg.Canvas(size=(1, 1), key='-CANVAS2-')]],
569                                                     ...
        size=(400, 400))]])]], pad=(0, 0)), sg.Button("Zoom")],
570         [sg.HorizontalSeparator()]]
571
572     window = sg.Window('CNC Fres', layout, size=(1030, ...
        770), return_keyboard_events=True,
573                        finalize=True, font=('Helvetica', ...
        16), grab_anywhere=True)
574     list_element: sg.Listbox = window.Element('-BOX-')
575     # Store listbox element for easier access and to get to
        docstrings
576     prediction_list, input_text, sel_item = [], "", 0
577     # Fullscreen
578     window.Maximize()
579
580     # Canvas
581     canvas_elem = window['-CANVAS-']
582     canvas = canvas_elem.TKCanvas
583     figsize = (4, 4)
584     fig, ax = plt.subplots(num=1, figsize=figsize, dpi=100)
585     ax.grid(True)
586     ax.set_xlim(-5, 105)
587     ax.set_ylim(-5, 105)
588     ax.set_xlabel("X - axis [mm]")
589     ax.set_ylabel("Y - axis [mm]")
590     fig_agg = draw_figure(fig, canvas)
591
592     # Canvas2
593     canvas_elem2 = window['-CANVAS2-']
594     canvas2 = canvas_elem2.TKCanvas
595     figsize = (4, 4)
596     #fig2, ax2 = plt.figure(figsize=figsize, dpi=100)
597     fig2 = plt.figure(num=2, figsize=figsize,)
598     ax2 = fig2.add_subplot(111, projection='3d')
599     ax2.grid(True)
600     ax2.set_xlim(-5, 1005)
601     ax2.set_ylim(-5, 1005)
602     ax2.set_zlim(-5, 70)
603     ax2.set_xlabel("X - axis [mm]")
604     ax2.set_ylabel("Y - axis [mm]")
605     ax2.set_zlabel("Y - axis [mm]")
606     #global xarr, yarr, zarr
607     fig_agg2 = draw_figure(fig2, canvas2)
608
609     # Element key binds:
610     window['-IN-'].bind("<Return>", "_Enter")  # Enter
611     window['-BOX-'].bind("<Return>", "_Enter")  # Enter
612     window['-CMD-LIST-'].bind("<Double-Button-1>", ...
        "_DoubleClick")  # Double
        click
613     while True:  # Event Loop
614         # Applying messages
615         window['-OUTPUT-'].update('\n'.join(messages))
616         # Starting events and timeout
617         event, values = window.read(timeout=10, ...
        timeout_key='-TIMEOUT-')
618
619         # Check events:
620         if event in (sg.WIN_CLOSED, 'Exit'):
621             # Closing window
622             break
623         if event == "Refresh Port":
624             print(intro2)
625             append_message(intro2)
```

```
626             deviceID, comValue = check_ports()
627             window['-OUTPUT-'].update('\n'.join(messages))
628             for i in range(len(deviceID)):
629                 window['-OPTION-'].update(deviceID[i])
630 
631         if event == "-BC-":
632             try:
633                 # Connect to Arduino / Can do this via button
634                 # print("Option")
635                 comDesc = values['-OPTION-']
636                 # print(comDesc)
637                 findComValue = deviceID.index(comDesc)
638                 # print(findComValue)
639                 comID = comValue[findComValue]
640                 if not serialPort.isOpen():
641                     try:
642                         thread_setupSerial = ...
    Thread(target=setupSerial, args=(baudrate, comID,))
643                         threads.append(thread_setupSerial)
644                         lenThreads = len(threads) - 1
645                         threads[lenThreads].start()
646                         time.sleep(0.2)
647                         if serialPort.isOpen():
648                             print("Connection worked")
649                             ...
    window["-BC-"].update("Disconnect")
650                     except Exception as e:
651                         errorConnect = "Connection failed: ...
    {}".format(e)
652                         print(errorConnect)
653                         append_message(errorConnect)
654                         ...
    window['-OUTPUT-'].update('\n'.join(messages))
655                 else:
656                     # Disconnect from arduino
657                     try:
658                         time.sleep(0.2)
659                         if serialPort.isOpen():
660                             serialPort.close()
661                             textDisconnect = ...
    "Disconnecting from port..."
662                             print(textDisconnect)
663                             append_message(textDisconnect)
664                             ...
    window['-OUTPUT-'].update('\n'.join(messages))
665                             window["-BC-"].update("Connect")
666                     except Exception as e:
667                         errorConnect = "Disconnetion ...
    failed: {}".format(e)
668                         print(errorConnect)
669                         append_message(errorConnect)
670                         ...
    window['-OUTPUT-'].update('\n'.join(messages))
671             except Exception as e:
672                 try:
673                     # Connect to Arduino / Can do this via
    button
674                     # print("Option")
675                     comDesc = values['-OPTION-']
676                     # print(comDesc)
677                     findComValue = deviceID.index(comDesc)
678                     # print(findComValue)
679                     comID = comValue[findComValue]
680                     try:
681                         thread_setupSerial = ...
    Thread(target=setupSerial, args=(baudrate, comID,))
682                         threads.append(thread_setupSerial)
683                         lenThreads = len(threads) - 1
684                         threads[lenThreads].start()
685                         time.sleep(0.2)
686                         if serialPort.isOpen():
687                             print("Connection worked")
688                             ...
    window["-BC-"].update("Disconnect")
689                     except Exception as e:
690                         errorConnect = "Connection failed: ...
    {}".format(e)
691                         print(errorConnect)
692                         append_message(errorConnect)
693                         ...
```

```
        window['-OUTPUT-'].update('\n'.join(messages))
694                 except Exception as e:
695                     errorConnect = "Connection failed: ...
    {}".format(e)
696                     print(errorConnect)
697                     append_message(errorConnect)
698 
    window['-OUTPUT-'].update('\n'.join(messages))
699 
700         if event in (sg.WIN_CLOSED, 'Go') or event == ...
    '-IN-' + '_Enter':
701             # Something in "Input Command"+ press enter or
    Gobutton is pressed
702             #global cordX, cordY, cordZ
703             send_text = str(values['-IN-'])
704             len_text = len(send_text)
705             print(len_text, ": Input is ", send_text)
706             try:
707                 if send_text[0] == "(" and send_text[-2:] ...
    == ",)":
708                     send_text = send_text[2:-3]
709                     print("input changed to ", send_text)
710                 if send_text[0] == "{" and send_text[-1:] ...
    == "}":
711                     send_text = send_text[2:-2]
712                     print("input changed to ", send_text)
713                 # Command Senter:
714                 if send_text == 'M100 - Help':
715                     msg_echo = 'M100'
716                     append_message(msg_echo)
717                     sendToArduino(msg_echo, window)
718                     arduino_listen(window)
719                 elif send_text == 'M120 - Home':
720                     msg_echo = 'M120'
721                     append_message(msg_echo)
722                     sendToArduino(msg_echo, window)
723                     arduino_listen(window)
724                 elif send_text == 'M114 - Where':
725                     msg_echo = 'M114'
726                     append_message(msg_echo)
727                     sendToArduino(msg_echo, window)
728                     arduino_listen(window)
729                 elif send_text == 'M17 - Enable Motors':
730                     msg_echo = 'M17'
731                     append_message(msg_echo)
732                     sendToArduino(msg_echo, window)
733                     arduino_listen(window)
734                 elif send_text == 'M18 - Disable Motors':
735                     msg_echo = 'M18'
736                     append_message(msg_echo)
737                     sendToArduino(msg_echo, window)
738                     arduino_listen(window)
739                 elif send_text == 'G90 - Absolute':
740                     msg_echo = 'G90'
741                     append_message(msg_echo)
742                     sendToArduino(msg_echo, window)
743                     arduino_listen(window)
744                 elif send_text == ' G91 - Incremental':
745                     msg_echo = 'G91'
746                     append_message(msg_echo)
747                     sendToArduino(msg_echo, window)
748                     arduino_listen(window)
749                 elif send_text == 'G92 - Set Origo':
750                     msg_echo = 'G92'
751                     append_message(msg_echo)
752                     sendToArduino(msg_echo, window)
753                     arduino_listen(window)
754                 else:
755                     # Sender til Arduino
756                     append_message(send_text)
757                     sendToArduino(send_text, window)
758                     arduino_listen(window)
759 
760             except Exception as e:
761                 print(e)
762                 send_text = 'Error: {}'.format(e)
763                 print(f"input changed to {send_text}")
764                 append_message(send_text)
```

```
765             window['-IN-'].update('')  # Empty Input Box
766             window['-BOX-'].update('')  # Empty Auto Correct
767             window['-OUTPUT-'].update('\n'.join(messages))
768         if event == 'Clear Chat':
769             messages.clear()
770             window['-OUTPUT-'].update('')
771         if event == 'Arduino COM':
772             setupSerial(baudrate, deviceID)
773         if event == sg.WINDOW_CLOSED:
774             break
775         elif event in (sg.WIN_CLOSED, 'Escape'):
776             window['-IN-'].update('')
777         elif event.startswith('Down') and ...
    len(prediction_list):
778             sel_item = (sel_item + 1) % len(prediction_list)
779             list_element.update(set_to_index=sel_item, ...
    scroll_to_index=sel_item)
780         elif event.startswith('Up') and len(prediction_list):
781             sel_item = (sel_item + (len(prediction_list) - ...
    1)) % len(prediction_list)
782             list_element.update(set_to_index=sel_item, ...
    scroll_to_index=sel_item)
783         elif event == '\r':
784             if len(values['-BOX-']) > 0:
785                 window['-IN-'].update(value=values['-BOX-'])
786         elif event == '-IN-':
787             text = values['-IN-'] if not values['-IGNORE ...
    CASE-'] else values['-IN-'].lower()
788             if text == input_text:
789                 continue
790             else:
791                 input_text = text
792             prediction_list = []
793             if text:
794                 if values['-IGNORE CASE-']:
795                     prediction_list = [item for item in ...
    choices if item.lower().startswith(text)]
796                 else:
797                     prediction_list = [item for item in ...
    choices if item.startswith(text)]
798
799             list_element.update(values=prediction_list)
800             sel_item = 0
801             list_element.update(set_to_index=sel_item)
802
803         elif event == '-BOX-' + '_Enter':
804             window['-IN-'].update(value=values['-BOX-'])
805             window['-IN-'].set_focus()
806         elif event == '-CMD-LIST-' + '_DoubleClick':
807             window['-IN-'].update(value=values['-CMD-LIST-'])
808             window['-IN-'].set_focus()
809         elif event == 'Zoom':
810             if zoom2DPlot:
811                 zoom2DPlot = False
812             else:
813                 zoom2DPlot = True
814         elif event == 'Send G-code':
815             if values['-FILENAME-']:
816                 # Changing / updating filename
817                 print("Filename changed to: " + ...
    str(values['-FILENAME-']))
818                 ...
    window['-FILENAME-'].set_tooltip(str(values['-FILENAME-']))
819                 # Remembering last filename
820                 sg.user_settings_set_entry('-filenames-', ...
    list(
821                     ...
    set(sg.user_settings_get_entry('-filenames-', []) + ...
    [values['-FILENAME-'], ])))
822                 sg.user_settings_set_entry('-last ...
    filename-', values['-FILENAME-'])
823                 # Getting new filename
824                 file = values['-FILENAME-']
825                 print("File loaded: " + file)
826                 append_message("File loaded: " + file)
827                 # sendGcode(file, window)
828                 thread_sendGcode = ...
    Thread(target=sendGcode, args=(file, window,))
829                 threads.append(thread_sendGcode)
```

```
                lenThreads = len(threads) - 1
                threads[lenThreads].start()

            else:
                txt = "No File found / added"
                print(txt)
                append_message(txt)

        elif event == '-FILENAME-':
            print("Filename changed to: " + str(values['-FILENAME-']))
            window['-FILENAME-'].set_tooltip(str(values['-FILENAME-']))
            messages.append("File changed to: " + str(values['-FILENAME-']))
            window['-OUTPUT-'].update('\n'.join(messages))

        elif event == 'Load G-code':
            global xarr, yarr, zarr
            if values['-FILENAME-']:
                # Endre/oppdatere filnavn
                print("Filename changed to: " + str(values['-FILENAME-']))
                window['-FILENAME-'].set_tooltip(str(values['-FILENAME-']))
                # Husk forrige filnavn
                sg.user_settings_set_entry('-filenames-', list(set(sg.user_settings_get_entry('-filenames-', []) + [values['-FILENAME-'], ])))
                sg.user_settings_set_entry('-last filename-', values['-FILENAME-'])

                # Hente det nye filnavnet:
                fil = values['-FILENAME-']
                append_message('File loaded: ' + fil)

                # Animer filen:
                try:
                    # Canvas 1
                    filpath = values['-FILENAME-']
                    xarr, yarr, zarr = [0.00], [0.00], [0.00]
                    xarr, yarr, zarr = animer_filen(filpath)
                    plt.figure(num=1)
                    ax.set_xlim(min(xarr) - 1, max(xarr) + 1)
                    ax.set_ylim(min(yarr) - 1, max(yarr) + 1)
                    zoom2DPlot = True


                    # Canvas 2
                    plt.figure(num=2)
                    ax2.axes.set_xlim3d(left=min(xarr) - 1, right=max(xarr) + 1)
                    ax2.axes.set_ylim3d(bottom=min(yarr) - 1, top=max(yarr) + 1)
                    ax2.axes.set_zlim3d(bottom=min(zarr) - 1, top=max(yarr) + 1)
                    plt.plot(xarr, yarr, zarr)
                    fig_agg2.draw()
                except Exception as e:
                    print(e)
                #plt.show()
                #window.disable()
            else:
                append_message('Ingen fil funnet / lagt til')

        elif event == 'Clear History':
            sg.user_settings_set_entry('-filenames-', [])
            sg.user_settings_set_entry('-last filename-', '')
            window['-FILENAME-'].update(values=[], value='')
        elif event == '-TIMEOUT-':
            # Canvas update
            if canvas_update == 100:
                # Canvas 1
                global cordX, cordY, cordZ
                # current position

                ax.cla()
                ax.grid(True)
                ax.set_xlim(-2, 503)
                ax.set_ylim(-3, 1003)
                try:
                    filpath = values['-FILENAME-']
```

```
                    xarr, yarr, zarr = [0.00], [0.00], [0.00]
                    xarr, yarr, zarr = animer_filen(filpath)
                    x1, y1, z1 = cordX[-1], cordY[-1], cordZ[-1]
                    if zoom2DPlot == True:
                        ax.set_xlim(min(xarr) - 1, max(xarr) + 1)
                        ax.set_ylim(min(yarr) - 1, max(yarr) + 1)
                except Exception as e:
                    x1, y1, z1 = [0.00], [0.00], [0.00]
                    print("her")
                    print(e)
                ax.set_xlabel("X - axis [mm]")
                ax.set_ylabel("Y - axis [mm]")
                ax.yaxis.set_label_position('right')
                ax.xaxis.set_label_position('top')
                try:
                    plt.figure(num=1)
                    for i in range(len(cordX)):
                        if i != len(cordX):
                            plt.plot(cordX[i:i+2], cordY[i:i+2], 'g', linestyle="--")
                        else:
                            break
                except Exception as e:
                    print(e)
                ax.scatter(x1, y1, c='red', s=50.0, label='Position', alpha=1, edgecolors='none')
                ax.legend()
                ax.legend(bbox_to_anchor=(1, 1.15))
                fig_agg.draw()

                # Canvas 2
                x, y, z = cordX[-1], cordY[-1], cordZ[-1]
                ax2.cla()
                ax2.grid(True)
                ax2.set_xlim(-0, 505)
                ax2.set_ylim(-0, 1005)
                ax2.set_zlim(-0, 70)
                ax2.set_xlabel("X - axis [mm]")
                ax2.set_ylabel("Y - axis [mm]")
                ax2.set_zlabel("Z - axis [mm]")
                try:
                    filpath = values['-FILENAME-']
                    xarr, yarr, zarr = [0.00], [0.00], [0.00]
                    xarr, yarr, zarr = animer_filen(filpath)
                    plt.figure(num=2)
                    ax2.axes.set_xlim3d(left=min(xarr)-1, right=max(xarr)+1)
                    ax2.axes.set_ylim3d(bottom=min(yarr)-1, top=max(yarr)+1)
                    ax2.axes.set_zlim3d(bottom=min(zarr)-1, top=max(yarr)+1)
                    plt.plot(xarr, yarr, zarr, 'b')
                    for i in range(len(cordX)):
                        if i != len(cordX):
                            plt.plot(cordX[i:i + 2], cordY[i:i + 2], 'g', linestyle="--")
                        else:
                            break
                except Exception as e:
                    pass
                    #print(e)
                ax2.scatter(x, y, z, c='Red', s=50.0, label='Position', alpha=1, edgecolors='none')
                ax2.legend()
                fig_agg2.draw()
                canvas_update = 0
            else:
                canvas_update = canvas_update + 1

        window['-OUTPUT-'].update('\n'.join(messages))
    window.close()


if __name__ == '__main__':
    main()
    print("Ending program")
```

Vedlegg B

Kode for maskinkontrolleren

Koden nedenfor er skrevet i Arduino IDE (modifisert C++). Kan hentes her. NB! Filen kan bare hentes med Adobe Acrobat.

Kode B.1: Skript som produserer firkant-puls. Gir mulighet for å utføre bevegelse av motor med konstant hastighet.

```
1  #define DIR_PIN          2
2  #define STEP_PIN         3
3  #define ENABLE_PIN       4
4
5  void setup() {
6    pinMode(DIR_PIN,    OUTPUT);
7    pinMode(STEP_PIN,   OUTPUT);
8    pinMode(ENABLE_PIN, OUTPUT);
9    digitalWrite(ENABLE_PIN, LOW);
10 }
11
12 void loop() {
13   // Her vil firkantpulsen bli produsert
14   digitalWrite(STEP_PIN, HIGH);
15 // Her blir bredden på pulsen satt.
16   delaymicroseconds(10);
17   digitalWrite(STEP_PIN, LOW);
18   // Her vil lengden på den lave perioden av pulsen bli satt.
19   delay(1);
20 }
```

Kode B.2: Denne koden viser hvordan vi implementerte simpel akkselerasjon.

```
1  #define DIR_PIN          2
2  #define STEP_PIN         3
3  #define ENABLE_PIN       4
4
5  void setup() {
6    pinMode(STEP_PIN,   OUTPUT);
7    pinMode(DIR_PIN,    OUTPUT);
8    pinMode(ENABLE_PIN, OUTPUT);
9  }
10
```

```
11 void simpleAccel(int steps) {
12
13   int lowSpeed = 2000; //Her blir laveste hastighet satt, også
       kjent som høyest periode mellom høy flanke.
14   int highSpeed = 100; // Her blir høyest hastighet satt, som
       da er laveste periode mellom høy flanke.
15   int change = 2; // Hvor mye perioden endres i mikrosekunder
16
17   // Setter opp akkselerasjons-rampe og deselerasjonsrampe
18   int rampUpStop = (lowSpeed - highSpeed) / change;
19   if ( rampUpStop > steps / 2 )
20     rampUpStop = steps / 2;
21   int rampDownStart = steps - rampUpStop;
22   int d = lowSpeed;
23
24   // Utfører steg ved å produsere firkant-pulsene og deretter
       sette ny periode for å øke/redusere hastigheten for
       motoren
25   for (int i = 0; i < steps; i++) {
26     digitalWrite(STEP_PIN, HIGH);
27     delayMicroseconds(10);
28     digitalWrite(STEP_PIN, LOW);
29     delayMicroseconds(d);
30     if ( i < rampUpStop )
31       d -= change;
32     else if ( i > rampDownStart )
33       d += change;
34   }
35 }
36
37 void loop() {
38   digitalWrite(DIR_PIN, LOW);
39   simpleAccel(2400);
40   digitalWrite(DIR_PIN, HIGH);
41   simpleAccel(2400);
42
43   while (true);
44
45 }
```

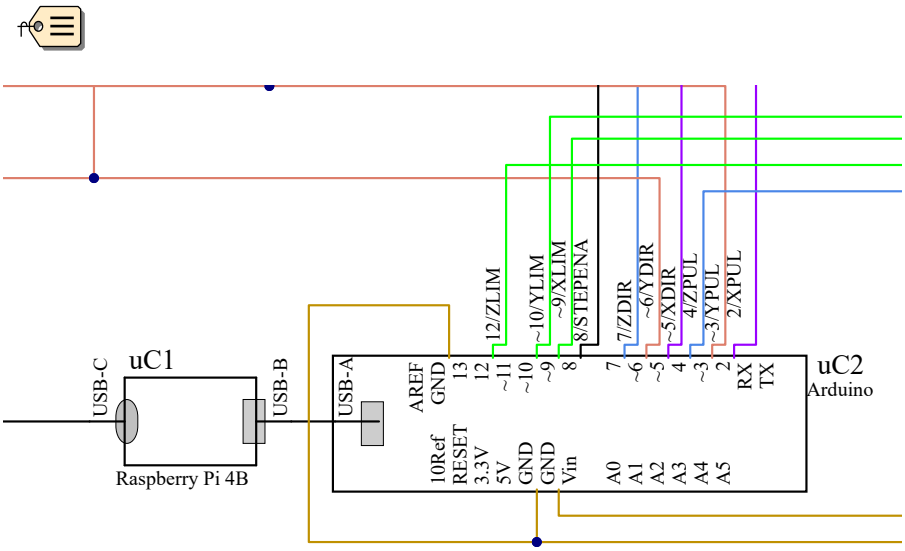
Kode B.3: I denne koden er det implementert utregninger for å oppnå konstant akselerasjon.

```
 1 #define DIR_PIN          2
 2 #define STEP_PIN         3
 3 #define ENABLE_PIN       4
 4
 5 #define STEP_HIGH        PORTD |=  0b00001000;
 6 #define STEP_LOW         PORTD &= ¬0b00001000;
 7
 8 #define TIMER1_INTERRUPTS_ON    TIMSK1 |=  (1 << OCIE1A);
 9 #define TIMER1_INTERRUPTS_OFF   TIMSK1 &= ¬(1 << OCIE1A);
10
11 unsigned int c0;
12
13 void setup() {
14   pinMode(STEP_PIN,   OUTPUT);
15   pinMode(DIR_PIN,    OUTPUT);
16   pinMode(ENABLE_PIN, OUTPUT);
17
18   noInterrupts();
19   TCCR1A = 0;
20   TCCR1B = 0;
21   TCNT1  = 0;
22   OCR1A = 1000;
23   TCCR1B |= (1 << WGM12);
24   TCCR1B |= ((1 << CS11) | (1 << CS10));
25   interrupts();
26
27   c0 = 1600; // Dette var lowSpeed i forrige kode, nå skiftet
       til c0 for å stemme overens med formler fra AVR sitt
       datablad. Det er start hastigheten til motoren.
28 }
29
30 volatile int dir = 0;
31 volatile unsigned int maxSpeed = 10; // Maks hastighet satt
```

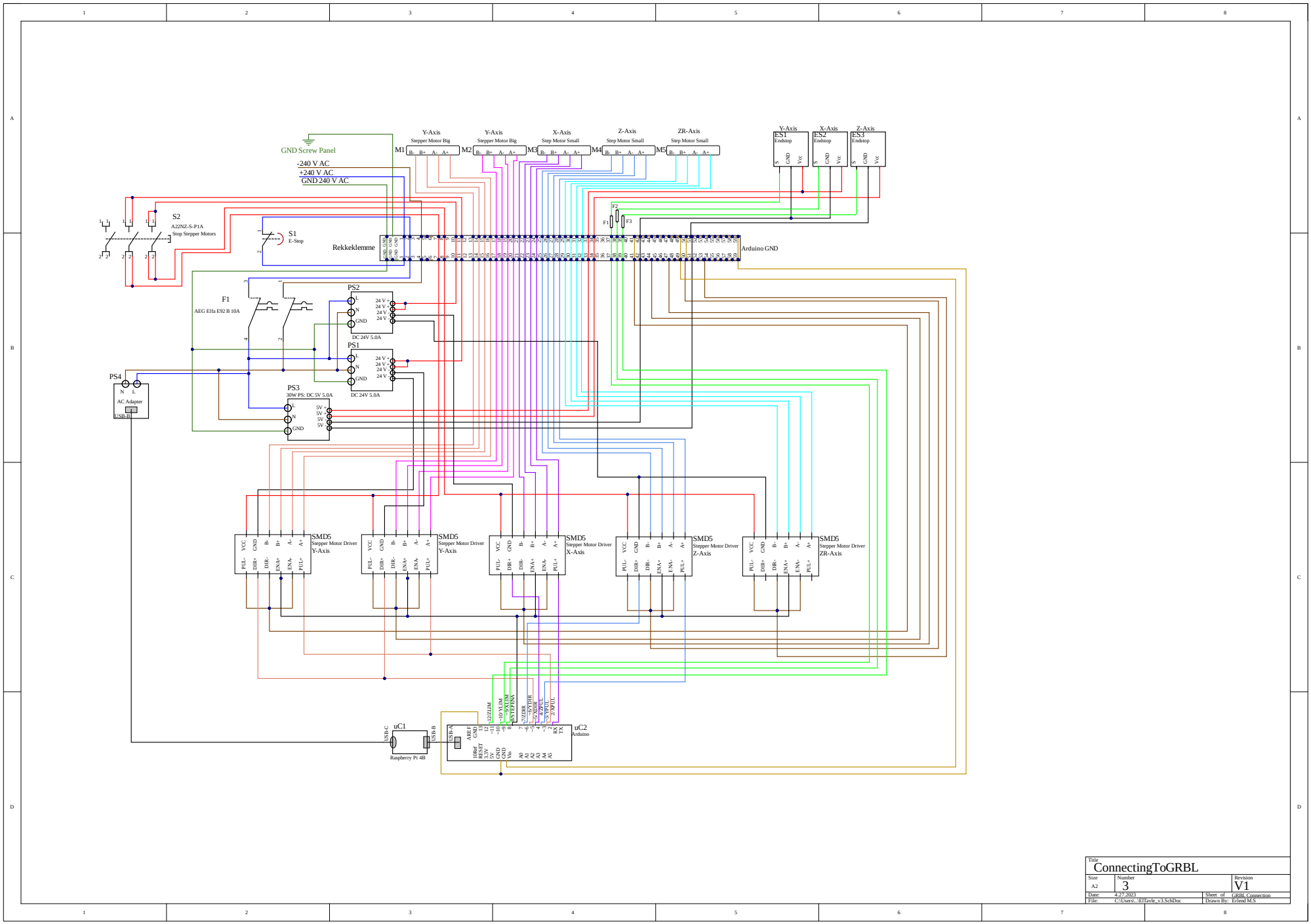
```
      som minste lav periode i microsekunder
32  volatile unsigned long n = 0;
33  volatile float d;
34  volatile unsigned long stepCount = 0;
35  volatile unsigned long rampUpStepCount = 0;
36  volatile unsigned long totalSteps = 0;
37  volatile int stepPosition = 0;
38
39  volatile bool movementDone = false;
40
41  ISR(TIMER1_COMPA_vect) // Denne blir kalt for å utføre
      stegene, dvs produsere pulsene som sendes videre
42  {
43    if ( stepCount < totalSteps ) {
44      STEP_HIGH
45      delayMicroseconds(10); // standard 10 micro sekunder slik
        at mikrosteg driver skal kunne detekte høy flanke
46      STEP_LOW
47      stepCount++;
48      stepPosition += dir;
49    }
50    else {
51      movementDone = true;
52      TIMER1_INTERRUPTS_OFF
53    }
54
55    if ( rampUpStepCount == 0 ) { // akselerasjonsfasen
56      n++;
57      d = d - (2 * d) / (4 * n + 1); // likning for cn i fra AVR
        datablad, her er cn skiftet om til d for delay for å
        enklere kunne følge koden.
58      if ( d ≤ maxSpeed ) { // Sjekker om vi har nådd max
        hastighet
59        d = maxSpeed;
60        rampUpStepCount = stepCount;
61      }
62      if ( stepCount ≥ totalSteps / 2 ) { // halveispunktet
63        rampUpStepCount = stepCount;
64      }
65    }
66    else if ( stepCount ≥ totalSteps - rampUpStepCount ) { //
        deakselerasjons fasen
67      n--;
68      d = (d * (4 * n + 1)) / (4 * n + 1 - 2);
69    }
70
71    OCR1A = d;
72  }
73
74  void moveNSteps(long steps) {
75
76    digitalWrite(DIR_PIN, steps < 0 ? HIGH : LOW);
77    dir = steps > 0 ? 1 : -1;
78    totalSteps = abs(steps);
79    d = c0;
80    OCR1A = d;
81    stepCount = 0;
82    n = 0;
83    rampUpStepCount = 0;
84    movementDone = false;
85
86    TIMER1_INTERRUPTS_ON
87  }
88
89  void moveToPosition(long p, bool wait = true) {
90    moveNSteps(p - stepPosition);
91    while ( wait && ! movementDone );
92  }
93
94  void loop() {
95
96    maxSpeed = 20;
97    moveToPosition( 0 );
98    moveToPosition(10000);
99
100   while (true);
101
102 }
```

Vedlegg C

El-tavle



Figur C.1: Koblingskjema til Arduino Uno.



Title: ConnectingToGRBL		
Size: A2	Number: 3	Revision: V1
Date: 4.27.2023	Sheet of: GRBL Connection	
File: C:\Users\...\ElTavle_v3.SchDoc	Drawn By: Erlend M.S	

Figur C.2: Koblingskjema av el-tavle.

Vedlegg D

G-kode

Tabell D.1: G-kode filer. Filene kan bare hentes med Adobe Acrobat

G-kode	**Fil**
Kvadrat	
Kvadrat med diagonaler	
Rektangel	
Rektangel med diagoneler	
Parallellogram	
Sirkel	
Tre sirkler	
Sirkel og kvadrat med diagonaler	
Bokstav «J»	
Navn	

Tabell D.2: Tabell for G-kode kommandoer. Sammenligner kommandoer mellom GRBL og vår maskinkontroller.

G-kode	Beskrivelse	GRBL	Vår
G0	Rask posisjonering	✓	✓
G1	Lineær bevegelse med spesifikk fart (F)	✓	✓
G2, G3	Bevegelse i bue med spesifikk fart	✓	✓
G4	Dveling (eng: dwell)	✓	
G17, G18, G19	Velg et plan for buene (?)	✓	
G20	Imperiske enheter	✓	
G21	Metriske enheter	✓	
G28	Gå til forhåndsdefinert posisjon	✓	
G28.2	Homing / Hjemmesyklus	✓	
G40, G43, G49	Akseptert, men ignorert	✓	
G45-G49	Akseptert, men ignorert	✓	
G90	Absolutt distanse modus	✓	✓
G91	Inkrementell distanse modus	✓	✓
G91.1	Bue radius modus	✓	
G92	Sett referansepunkt		✓
M0	Pause og stanse program - i karbidbevegelse vil dette tillate bruker å kjøre programmet igjen	✓	
M1	Sov (stopp) (eng: sleep, optional stop)	✓	
M3	Spindel på		
M2	Pause og stanse program	✓	
M5	Spindel av		
M6	Verktøy-bytte		
M7, M9	Akseptert, men ignorert	✓	
M17	Aktivere motorer		✓
M18	Deaktivere motorer		✓
M30	Pause og stanse program - i karbidbevegelse kan denne føre maskinen tilbake til maskinens opprinnelse	✓	
M100	Hjelp		✓
M114	Posisjon til verktøyet		✓
M120	Homing / Hjemsyklus		✓

Vedlegg E

GRBL Firmwaresettings

Innstillinger for GRBL, vist tabellen E.1 under, er tilgjengelig her [drive.google.com/...] eller på taggen. 

Tabell E.1: Liste med instillinger til GRBL. Verdiene i kollonen *Value* ble brukt i vårt oppsett av GRBL. Tabell hentet fra GRBL i github [22].

$x Setting	Value	Description	Unit
0	30	Step pulse time	microseconds
1	35	Step idle delay	milliseconds
2	0	Step pulse invert	mask
3	2	Step direction invert	mask
4	0	Invert step enable pin	boolean
5	0	Invert limit pins	boolean
6	0	Invert probe pin	boolean
10	1	Status report options	mask
11	0.010	Juntion deviation	millimeters
12	0.002	Arc tolerance	millimeters
13	0	Report in inches	boolean
20	1	Soft limits enable	boolean
21	1	Hard limits enable	boolean
22	1	Homing cycle enable	boolean
23	3	Homing diretion invert	mask
24	25.000	Homing locate feed rate	millimeters/minute
25	500.000	Homing search seek rate	millimeters/minute
26	244	Homing switch debounce delay	milliseconds
27	5.000	Homing switch pull-off distance	millimeters
30	1000	Maximum spindle speed	rounds per minute
31	1000	Minimum spindle speed	rounds per minute
32	0	Lase-mode enable	boolean
100	200.000	X-axis travel resolution	steps/millimeter
101	80.000	Y-axis travel resolution	steps/millimeter
102	200.000	Z-axis travel resolution	steps/millimeter
110	5000.000	X-axis maximum feedrate	millimeters/minute
111	15000.000	Y-axis maximum feedrate	millimeters/minute
112	300.000	Z-axis maximum feedrate	millimeters/minute
120	10.000	X-axis acceleration	millimeters/seconds^2
121	10.000	Y-axis acceleration	millimeters/seconds^2
122	10.000	Z-axis acceleration	millimeters/seconds^2
130	225.000	X-axis maximum travel (soft-limit)	millimeters
131	1000.000	Y-axis maximum travel (soft-limit)	millimeters
132	63.000	Z-axis maximum travel (soft-limit)	millimeters

Vedlegg F

Utstyrsliste

	Antall
El-tavle	1
Kraftforsyning 24V	2
Kraftforsyning 5V	1
Kraftforsyning (12V)	1
Mikrosteg driver	4
Motor y akse	2
Motor x akse	1
Motor z akse	1
Raspberry PI 4B	1
Arduino UNO	1
Gjengestang stor	2
Gjengestang liten	2
Grensebrytere	4
Nødbryter	2

Vedlegg G

Datablad

G.1 ATmega328P

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \times 100\%$$

Table 19-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f_{osc} = 1.0000MHz				f_{osc} = 1.8432MHz				f_{osc} = 2.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	–7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	–3.5%	7	0.0%	15	0.0%	8	–3.5%	16	2.1%
19.2k	2	8.5%	6	–7.0%	5	0.0%	11	0.0%	6	–7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	–3.5%
38.4k	1	–18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	–7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	–18.6%	1	–25.0%	2	0.0%	1	–18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max.(1)	62.5kbps		125kbps		115.2kbps		230.4kbps		125kbps		250kbps	

Note: 1. UBRRn = 0, error = 0.0%

Figur G.1: Tabell av overføringshastigheter med feil i %, hentet fra databladet ATmega328P [6]. ATmega328p er mikrokontrolleren som håndterer USB-seriell grensesnitt på Arduino Uno.

G.2 AVR446

2.3.1 Exact calculations of the inter-step delay

The first counter delay c_0 as well as succeeding counter delays c_n, are given by (see appendix for details):

$$c_0 = \frac{1}{t_t}\sqrt{\frac{2\alpha}{\dot{\omega}}} \qquad c_n = c_0\left(\sqrt{n+1} - \sqrt{n}\right)$$

The computational power of a microcontroller is limited, and calculating two square roots is time consuming. Therefore an approximation with less computational complexity is considered.

The counter value at the time n, using Taylor series approximation for the inter-step delay (see appendix for details) is given by:

$$c_n = c_{n-1} - \frac{2c_{n-1}}{4n+1}$$

This calculation is much faster than the double square root, but introduces an error of 0.44 at n=1. A way to compensate for this error is by multiplying c_0 with 0,676.

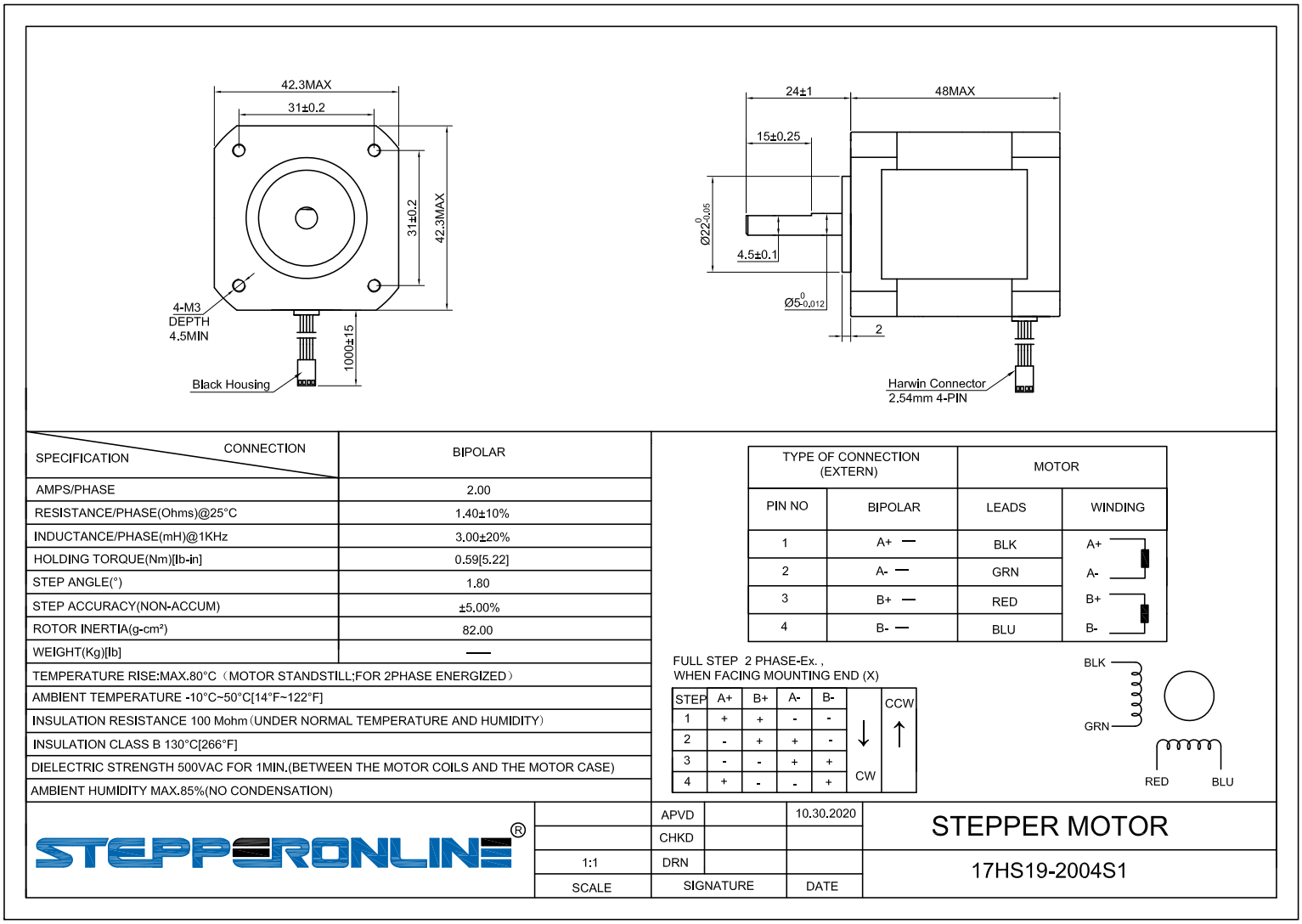
2.3.2 Change in acceleration

As shown in the appendix, the acceleration is given by c_0 and n. If a change in acceleration (or deceleration) is done, a new n must be calculated.

The time t_n and n as a function of the motor acceleration, speed and step angle are given by

Figur G.2: Formel brukt for utregning av akselerasjon i pulssignalene. Hentet fra databladet AVR446 [7]

G.3 Nema 17 Bipolar Step Motor



SPECIFICATION / CONNECTION	BIPOLAR
AMPS/PHASE	2.00
RESISTANCE/PHASE(Ohms)@25°C	1.40±10%
INDUCTANCE/PHASE(mH)@1KHz	3.00±20%
HOLDING TORQUE(Nm)[lb-in]	0.59[5.22]
STEP ANGLE(°)	1.80
STEP ACCURACY(NON-ACCUM)	±5.00%
ROTOR INERTIA(g-cm²)	82.00
WEIGHT(Kg)[lb]	—
TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）	
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]	
INSULATION RESISTANCE 100 Mohm（UNDER NORMAL TEMPERATURE AND HUMIDITY）	
INSULATION CLASS B 130°C[266°F]	
DIELECTRIC STRENGTH 500VAC FOR 1MIN.(BETWEEN THE MOTOR COILS AND THE MOTOR CASE)	
AMBIENT HUMIDITY MAX.85%(NO CONDENSATION)	

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A+	BLK	A+
2	A-	GRN	A-
3	B+	RED	B+
4	B-	BLU	B-

FULL STEP 2 PHASE-Ex.,
WHEN FACING MOUNTING END (X)

STEP	A+	B+	A-	B-
1	+	+	-	-
2	-	+	+	-
3	-	-	+	+
4	+	-	-	+

CW ↓ CCW ↑

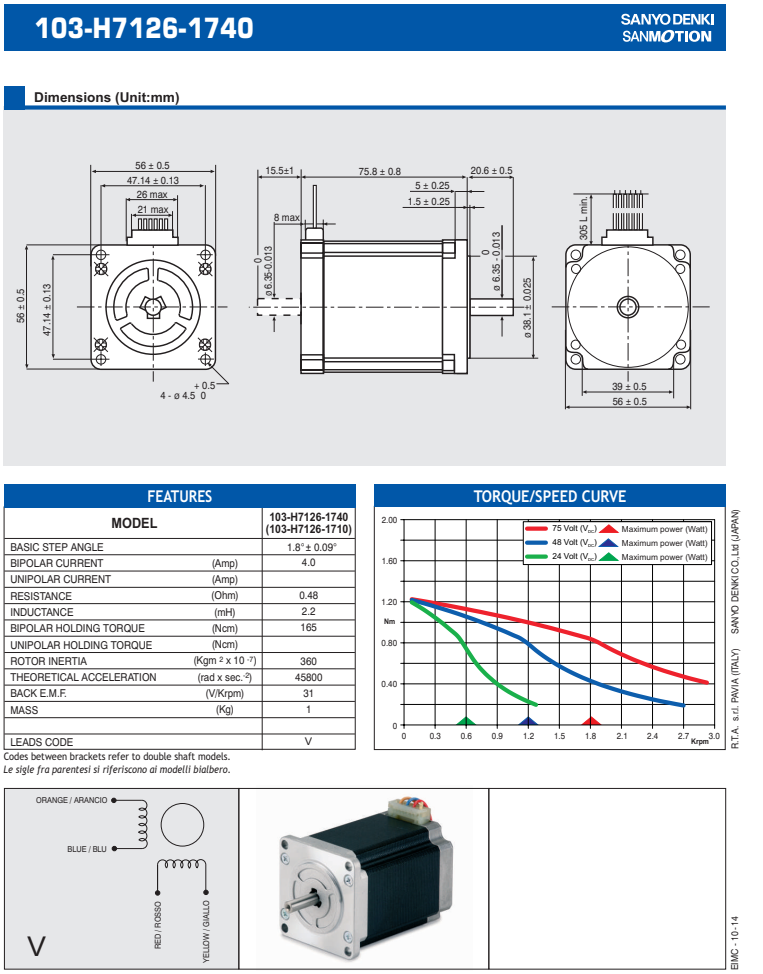
BLK
GRN
RED
BLU

STEPPERONLINE®

	APVD		10.30.2020	STEPPER MOTOR
	CHKD			
1:1	DRN			17HS19-2004S1
SCALE	SIGNATURE		DATE	

Figur G.3: Datablad for X- og Z-stegmotor. Hentet fra databladet Nema 17 [20].

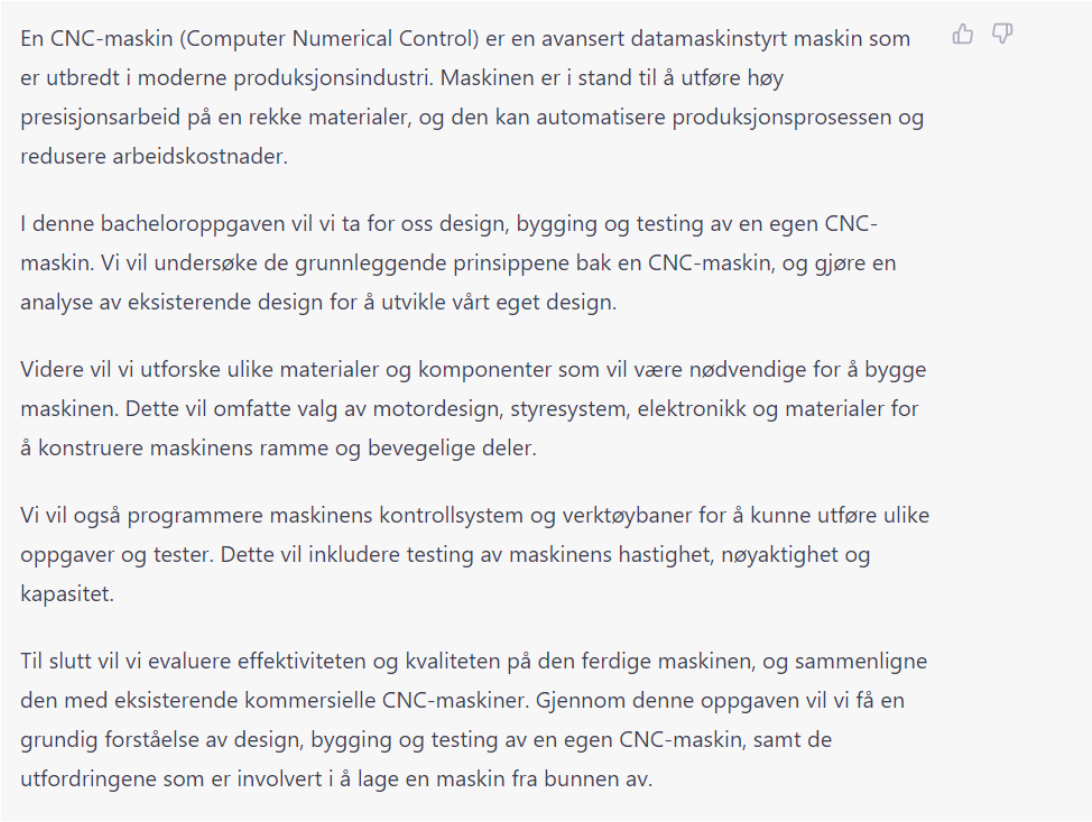
G.4 Sanyo Denki Step Motor



Figur G.4: Datablad for Y-stegmotor. Hentet fra databladet 103-H7126-1740 [19].

Vedlegg H

ChatGPT



En CNC-maskin (Computer Numerical Control) er en avansert datamaskinstyrt maskin som er utbredt i moderne produksjonsindustri. Maskinen er i stand til å utføre høy presisjonsarbeid på en rekke materialer, og den kan automatisere produksjonsprosessen og redusere arbeidskostnader.

I denne bacheloroppgaven vil vi ta for oss design, bygging og testing av en egen CNC-maskin. Vi vil undersøke de grunnleggende prinsippene bak en CNC-maskin, og gjøre en analyse av eksisterende design for å utvikle vårt eget design.

Videre vil vi utforske ulike materialer og komponenter som vil være nødvendige for å bygge maskinen. Dette vil omfatte valg av motordesign, styresystem, elektronikk og materialer for å konstruere maskinens ramme og bevegelige deler.

Vi vil også programmere maskinens kontrollsystem og verktøybaner for å kunne utføre ulike oppgaver og tester. Dette vil inkludere testing av maskinens hastighet, nøyaktighet og kapasitet.

Til slutt vil vi evaluere effektiviteten og kvaliteten på den ferdige maskinen, og sammenligne den med eksisterende kommersielle CNC-maskiner. Gjennom denne oppgaven vil vi få en grundig forståelse av design, bygging og testing av en egen CNC-maskin, samt de utfordringene som er involvert i å lage en maskin fra bunnen av.

Figur H.1: Inspirasjon til introduksjon ved hjelp av CHATGPT.