Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2022 Åpen
Forfatter(e): Kjetil Log Rogne, Kevin Tran	
Fagansvarlig: Karl Skretting Veileder(e): Karl Skretting, Ståle Freyer	
Tittel på bacheloroppgaven: Robotverktøy og system for å måle robothastighet gjennom en gitt bane Engelsk tittel: Robot tools and system for measuring robot speed through a given path	
Studiepoeng: 20	
Emneord: Arduino, ABB robot, punktsveising, simulering og automatisering	Sidetall: 37 + vedlegg/annet: 77 Stavanger 15. mai 2022

Innhold

Sammendrag

I denne rapporten tar vi for oss hvordan punktsveising kan gjennomføres via en robot. Først legger vi fram en beskrivelse av hvordan systemet vårt skal fungere. Deretter beskriver vi hvordan designet av verktøyet er, samt andre relevante komponenter. Vi legger også fram ulike koblingsskjemaer som viser hvordan lysdioder, fototransistorer og brytere skal kobles opp mot arduinoen. Oversikten over funksjonaliteten til programmet viser vi gjennom et tilstandsdiagram, før vi implementerer dette som kode i Arduino IDE.

Mot slutten av rapporten legger vi fram resultatet vårt og diskuterer dette. Vi kommer med forslag til eventuelle forandringer, samt vurderer vår endelige løsning. Avslutningsvis har vi en kort konklusjon som oppsummerer arbeidet vårt.

Forord

Denne oppgaven er skrevet ved Institutt for data- og elektroteknikk, på Universitetet i Stavanger. Forfatterne er Kjetil Log Rogne og Kevin Tran, hvor begge to studerer automatisering og elektronikkdesign på UiS.

I sammenheng med oppgaven har vi fått hjelp av ansatte ved UIS, og vi vil gjerne rette en takk til:

- **Karl Skretting** for veiledning og oppfølging
- **Ståle Freyer** for hjelp med utvikling av kode, samt utvikling av verktøy i Fusion360 og god veiledning
- **Romuald Karol Bernacki** for lån av laboratorieutstyr
- **Jon Fidjeland** for utstyr til oppkobling av verktøy

Kapittel 1

Introduksjon

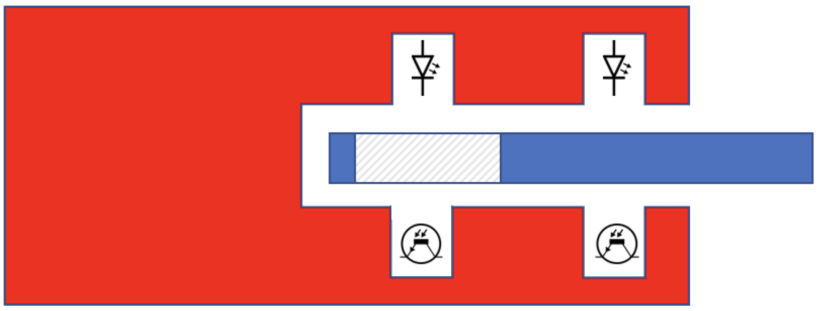
I dette kapittelet beskrives oppgaveteksten vår. Vi legger fram problemstillingen, hvor vi beskriver hva oppgaven går ut på og målet med oppgaven. Her beskrives det hvordan punktsveising fungerer, og hvordan vi løser det med tanke på utstyret som er tilgjengelig.

1.1 Problemstilling

Punktsveising er en sveiseprosess hvor man skjøter to plater sammen. Prinsippet bak punktsveising er at to kobberelektroder, som ligger på hver sin side av skjøten (i vårt tilfelle ørene), tilføres et lite strømstøt, slik at kontaktområde mellom platene får høy nok temperatur til at de kan plastifiseres og presses sammen. I vår oppgave skal vi ikke fysisk sveise, men vi skal bruke prinsippet med punktsveising til å utvikle et verktøy som kan simulere denne prosessen. Det ønsker vi å gjøre ved at roboten forflytter seg fra punkt til punkt med verktøyet festet til seg.

Målet vårt er å utvikle et verktøy som kan simulere denne punktsveisingen. Verktøyet skal være gaffelformet, bestående av to lysdioder og to fototransistorer. Vi må også lage punktene i banen den skal igjennom. Det er viktig at verktøyet er funksjonelt, i den forstand at det blir programmert riktig slik at verktøyet klarer å tolke informasjonen i punktene. Verktøyet skal festes

til roboten, for så å gå til ønsket punkt, stoppe i en viss tid, før den igjen går videre til neste punkt. Dette skal den gjøre gjennom hele banen vi lager. Banen skal bestå av et visst antall punkter som plasseres rundt omkring på et bord. For å få verktøyet til å lese av og tolke punktene programmerer vi verktøyet via en mikrokontroller. Når verktøyet er ferdig programmert fester vi det til en robot, slik at roboten kan bevege seg til punktene. Måten dette skal gjøres på er at roboten gir et kort signal rett før den er i ønsket punkt, som gjør at sveiseprosessen starter. Målet er at alt skal gå automatisk slik at man ikke trenger en datamaskin eller manuell styring. Man ønsker også at hele prosessen skal gå så raskt som mulig. Under ser du en figur som viser en oversikt over gaffelen når den er i et bestemt punkt.



Figur 1.1: Oversikt over gaffel med lysdioder og fototransistorer

Den rød delen av figuren er verktøyet vårt som roboten skal forflytte til de ulike punktene. Selve punktet er den blå delen av figuren. Som man kan se er det et hvitt område mellom de to blå delene og dette viser det lille hullet i punktet. Det er gjennom dette hullet lyset skal gå igjennom og belyse fototransistoren. Som man ser på figuren er det lyset fra den nederste lysdioden som skal gå gjennom hullet. Lyset fra den øverste lysdioden skal brytes. Det er kun når dette skjer og verkøyet har vært i punktet i en gitt tid at sveisingen er godkjent. Hvis ikke skal det rapporteres en feilmelding. En feilmelding sendes ikke ut før roboten har vært gjennom hele banen. På den måten får man ikke vite hvor i banen feilen skjer, men kun at det har skjedd en feil i ett av punktene på et tidspunkt i banen.

Kapittel 2

Bakgrunn

I dette kapittelet tar vi for oss verktøyet vi skal utvikle, samt andre nøkkelkomponenter og annet utstyr vi skal bruke for å gjennomføre punktsveisingen. Vi legger frem en detaljert beskrivelse av hvordan det gaffelformede verktøyet skal fungere og forklarer hvordan vi har tenkt til å lage det. Vi beskriver også kort de andre komponentene og bruksområdet de har i vår oppgave.

2.1 Utvikling av verktøy med fusion 360

For å løse denne oppgaven skal vi designe et gaffelformet verktøy. Verktøyet skal festes til en robot, slik at den kan simulere punktsveisingen. For å lage dette verktøyet har vi brukt 3D modellerings programmet Fusion 360. Fusion 360 er som sagt et design og modelleringsprogram, hvor vi kan konstruere og 3D printe figurer. Også punktene som skal brukes i punktsveisingen er 3D printet i fusion. (Selve 3D printingen av verktøyet og punktene har blitt utført av Ståle Freyer.) Vedlagt kan du se projeksjonstegningene av verktøyet og punktene som ble 3D printet. Tegningene viser verktøyet ovenfra. Dette kommer av at når verktøyet skal 3D printes gjøres dette lagvis, slik at den printer formene. 3D printeren sliter blant annet med å tolke tomrom i tegningene. [9]

2.2 Beskrivelse av verktøy

Verktøyet vi skal koble opp mot roboten er gaffelformet og består av to lysdioder og to fototransistorer. Disse står overfor hverandre slik at lysdioden lyser ett skarpt lys rett på fototransistoren. Målet med gaffelen er at vi skal bruke belysningen av fototransistoren til å vite om gaffelen er i et punkt eller ikke. Enkelt forklart er at hvis den ene fototransistor blir belyst og den andre ikke blir belyst er gaffelen i punktet, mens ved alle andre tilfeller er den ikke i punktet. Gaffelen må også være i punktet i en gitt tid. Selve gaffelen er 3D printet ved hjelp av Fusion 360. Under ser du bilde av den 3D printede gaffelen både skrått ovenfra og fra siden. Som man kan se på bilde 2.2 ser man at det er noen hull og det er her lysdiodene settes inn på den ene siden og fototransistorene settes inn på den andre siden.

Figur 2.1: Gaffel verktøy

Figur 2.2: Gaffel fra siden

2.2.1 Komponentbeskrivelse

Lysdiode:
Komponenten består av halvledermateriale, altså materiale som i utgangspunktet ikke leder strøm like godt som metalliske ledere. Fordelen med

halvledende materialer er at de består av to typer partikler, nemlig elektroner og elektronhull, som begge kan lede strøm. Når disse kombineres slik at elektronene fyller plassen til et elektronhull, frigjøres det energi og det sendes ut et lys fra halvlederen. På denne måten har man frigjort energien som elektromagnetisk stråling i form av lys og det er dette som skjer i en lysdiode. Lyset kan både være i det synlige spekteret eller være infrarødt. [10]

Vi bruker en lysdiode som sender ut et skarpt hvitt lys. Vi kobler dioden i en seriekobling med en spenningskilde fra arduino på 3.3V og en motstand på 200Ω. Vi sender derfor en strøm på 16.5 mA gjennom lysdioden og dette gir oss et godt lys. Disse lysdiodene er sentrale i den senere koden så det er viktig at vi får ett skarpt og godt lys.

Vi bruker også en rød lysdiode. Denne lysdioden skal brukes som et varsellys slik at den lyser når sveisingen ikke er ok. Denne kobles også i serie med en motstand på 200Ω. Dette medfører at det går en strøm på 16.5 mA som gir et godt nok lys. I forhold til den hvite lysdioden er det ikke så nøye med den røde lysdioden hvor skarpt og godt lyset er, ettersom det ikke er noen fototransistor som skal bli belyst. Poenget er at vi skal se når det lyser og når det ikke lyser.

En lysdiode av den typen vi bruker er ikke spesielt dyr. Ved å kjøpe inn en god mengde lysdioder ligger prisen på rundt 1kr per stykk[15]. Allikevel skal man ikke sløse med lysdiodene, så det er viktig å være nøye når man kobler opp, slik at man ikke sender for mye strøm gjennom dem. Lysdiodene vi bruker skal helst ikke overstige 20mA.

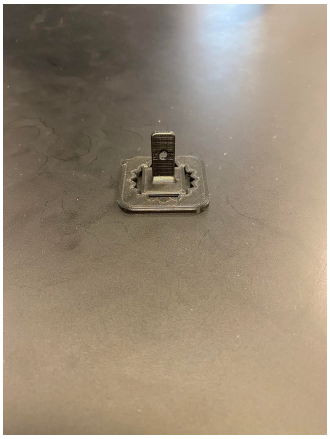
Figur 2.3: Bilde av lysdiode

Fototransistor:
En fototransistor ligner veldig designmessig på en lysdiode. Fototransistoren er også en strømledende komponent, men til forskjell fra dioden som sender ut et lys, er fototransistoren avhengig av å bli belyst for å fungere. Når den blir belyst absorberer transistoren lyset, slik at den kan registrere hvor mye lys som inntreffer. Disse lyspulsene som fototransistoren registrerer kan man igjen konvertere til digitale elektriske signaler. Det er disse elektriske signalene vi er ute etter. [2]

Fototransistoren kobler vi også i en seriekobling med en spenningsforsyning på 5V og med en resistanse på 47k Ω. Dermed går det ikke så mye strøm gjennom fototransistoren, men poenget er at den skal bli belyst av lysdioden. På den måten får vi ut digitale elektriske signaler som vi skal bruke i programmeringen av arduinoen for å identifisere om verktøyet er i ønsket punkt.

Figur 2.4: Bilde av fototransistor

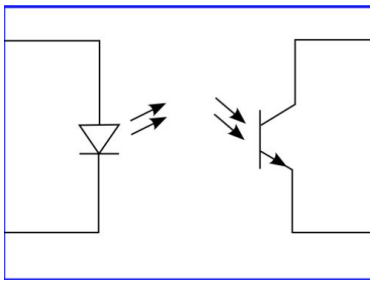
Ører:
Punktene vi skal bruke i banen er "ører" som er 3D printet i fusion. De er ca 2,5 cm høye og 1 cm brede. De er formet som et rektangel slik at den lengste siden står 90 grader opp fra riggen. Midt på rektangelet er det et lite hull som det skal sendes et lys gjennom. Målet er at gaffelen skal sende lyset gjennom dette hullet, slik at vi kan lage en kode som kan lese av og tolke verdien på fototransistoren. Det er dette verdiavviket mellom fototransistorene vi skal bruke for å definere om verktøyet er i punktet.



Figur 2.5: Bilde av øre som markerer hvor sveisepunkt er

Bryter:
For å simulere ett signal fra roboten har vi tatt i bruk en helt standard trykkbryter. Denne brukes til å starte sveiseprosessen ved et enkelt kort trykk og avslutter sveiseprosessen ved et langt trykk. Bryteren er også koblet i serie med en motstand på 4.7kΩ.

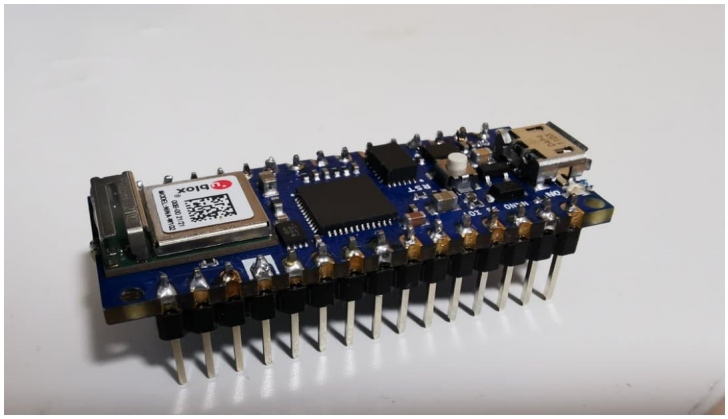
Optokobler:
En optokobler er en halvlederenhet som lar et elektrisk signal overføres mellom to isolerte kretser. En optokobler består av to komponenter, en LED som sender ut infrarødt lys og en fotosensitiv enhet som oppdager lys fra LED. [14]



Figur 2.6: Hvordan en optocoupler fungerer internt

Arduino:
For å gi verktøyet de ønskede egenskapene (se 2.2) bruker vi et kretskort av typen Arduino nano 33 iot. Dette er et enkelt kretskort bestående noen digitale og analoge innganger, samt en mikrokontroller av typen Arm Cortex-M0 32-bit SAMD21. Den har også en USB inngang, som vi kobler arduinoen til PC. For å koble arduinoen til breadboardet måtte vi først lodde alle pinnene på kretskortet.

Målet er å programmere mikrokontrolleren på arduinoen slik at vi kan tolke verdiene på fototransistoren og bruke disse til å skrive ett fungerende program i Arduino IDE. Arduino IDE er et gratis software program, som bruker språket C++, men med noen tilleggsfunksjoner.



Figur 2.7: Arduino

RobotStudio
For å kommunisere med ABB-roboten bruker vi RobotStudio. RobotStudio er et dataprogram, som er utviklet av selskapet ABB. Programmet ble utviklet for å gjøre det mulig å kommunisere med sine egne industrielle roboter og med RobotStudio kan man programmere og simulere disse robotene. En fordel med RobotStudio er at man kan legge til virtuelle kontrollere som er identiske med de virkelige robotene, for så å simulere disse. På den måten kan man jobbe offline med programmet uten å være tilkoblet den fysiske roboten. Man kan derfor også sjekke om programmet fungerer i praksis før man kobler opp mot den fysiske roboten. [13]

Her skal vi jobbe med roboten Norbert, som også ble brukt i et tidligere fag i ELE610. Vi har i dette faget i lab RS3 utviklet en kode for å plukke opp hockeypucker som ligger på forskjellige punkter på bordet. Denne koden blir i denne oppgaven videreutviklet til å tilpasse behovet vårt, slik at verktøyet kan lese av punktene som vi setter opp på bordet. Se vedlegg for kode av RS3 B

Kapittel 3

Konstruksjon

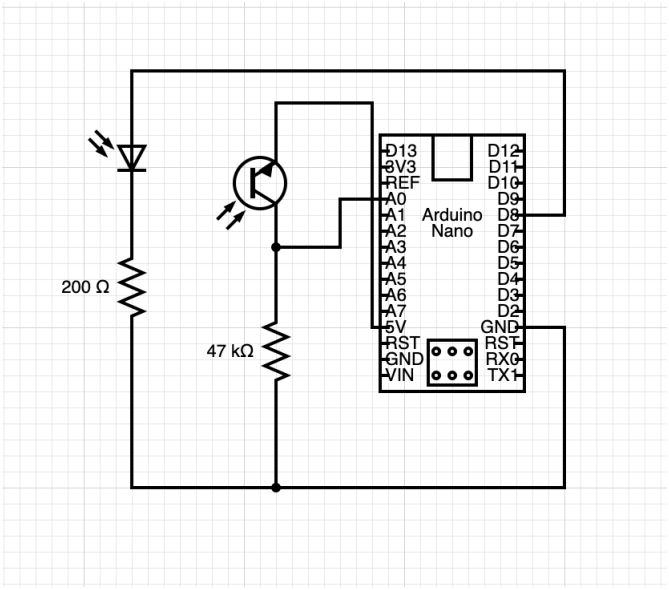
3.1 Løsningsmetode

I denne delen legger vi fram hvordan vi har gått fram for å løse denne oppgaven. Vi forklarer utviklingen av programmet stegvis, hvor vi legger fram de ulike koblingsskjemaene og tilstandsdiagrammene vi har brukt. Mot slutten av dette kapittelet legger vi fram den ferdige koden med forklaring.

3.2 Oppkobling med en diode og en fototransistor

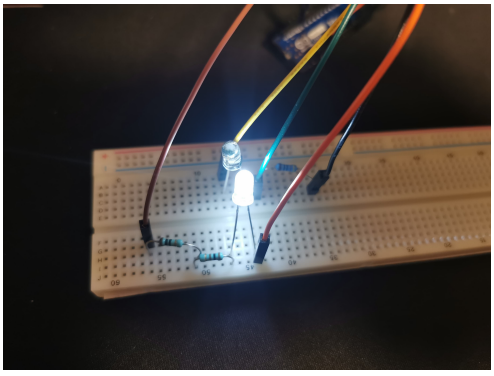
Det første vi gjorde var å lage ett enkelt koblingskjema for kretsen, hvor vi koblet arduinoen til en lysdiode og en fototransistor. Grunnen til at vi koblet med kun en lysdiode og en fototransistor først var for å lettere forstå hvordan komponentene fungerte før man lager en fullstendig krets. Koblingskjemaet ble da som følger:

3.2 Oppkobling med en diode og en fototransistor



Figur 3.1: Koblingskjema med en lysdiode og en fototransistor

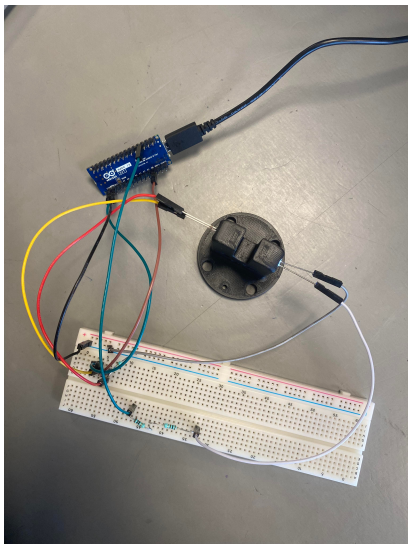
Deretter koblet vi opp kretsen på et breadboard etter koblingskjemaet ovenfor og koblet arduinoen til PCen for å kjøre koden. Målet med koden var at den skulle lese verdiene til fototransistoren slik at vi kan se forskjell på når fototransistoren blir belyst og når den ikke blir belyst. Først koblet vi lysdioden og fototransistoren direkte til breadboardet uten å sette de i gaffelen. Den oppkoblede kretsen ser man under.



Figur 3.2: Krets med en lysdiode og en fototransistor.

3.2 Oppkobling med en diode og en fototransistor

Som man ser ovenfor lyser dioden ett sterkt lys. Problemet her var bare at når vi kjørte koden fikk vi nesten ikke noe forskjell i verdier på avlesningen av fototransistoren. Dette gjorde at vi ikke kunne se forskjell på når fototransistoren ble belyst og når den ikke ble belyst. Vi prøvde også å legge ett lokk over kretsen slik at noe av omgivelseslyset forsvant, men verdiene vi fikk ut var fremdeles veldig like. Derfor prøvde vi å koble opp kretsen på nytt bare at vi denne gangen satte fototransistoren og lysdioden direkte i gaffelen. Kretsen så derfor slik ut.

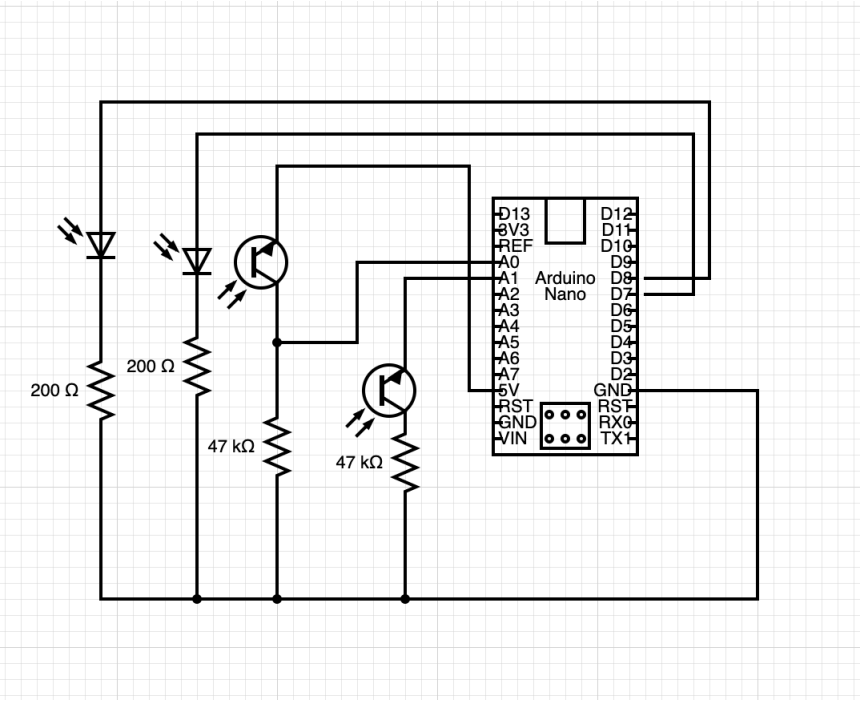


Figur 3.3: Oppkoblet krets med diode og transistor i gaffel.

Ved å sette lysdioden og fototransistoren i gaffelen fikk vi en mye mer konsentrert lystråle og vi fikk også tatt vekk noe av omgivelseslyset som kan virke forstyrrende. Dette ga oss et større utslag på fototransistoren. Vi fikk nå ett mye større avvik i verdien på fototransistoren når den blir belyst og når den ikke blir belyst. Dette avviket ble brukt som utgangspunktet når vi skulle programmere videre for å definere om gaffelen var i ett gitt punkt eller ikke.

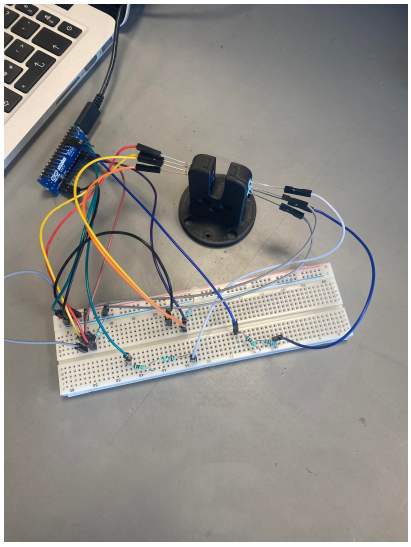
3.3 Oppkobling med to dioder og to fototransistorer

For at gaffelen skal forstå om den er i et punkt eller ikke, var vi nødt til å ta i bruk to lysdioder og to fototransistor. Dette er fordi punktene består av ett lite hull som lysstrålen skal kunne gå igjennom. Slik kan den ene fototransistoren bli belyst, mens den andre ikke blir belyst. Kretsen vil ikke være så ulik som den ovenfor, vi la kun til en ekstra lysdiode og fototransistor. Koblingsskjemaet for kretsen ser man under.



Figur 3.4: Koblingsskjema av krets med to lysdioder og to fototransistorer.

Deretter koblet vi opp kretsen etter koblingsskjemaet og satte lysdiodene og fototransistorene direkte inn i gaffelen. Som man ser på bildet nedenfor er begge lysdiodene og begge fototransistorene satt direkte inn i gaffelen slik at de belyser hver sin fototransistor.



Figur 3.5: Oppkobling av krets med to lysdioder og to fototransistorer direkte i gaffel.

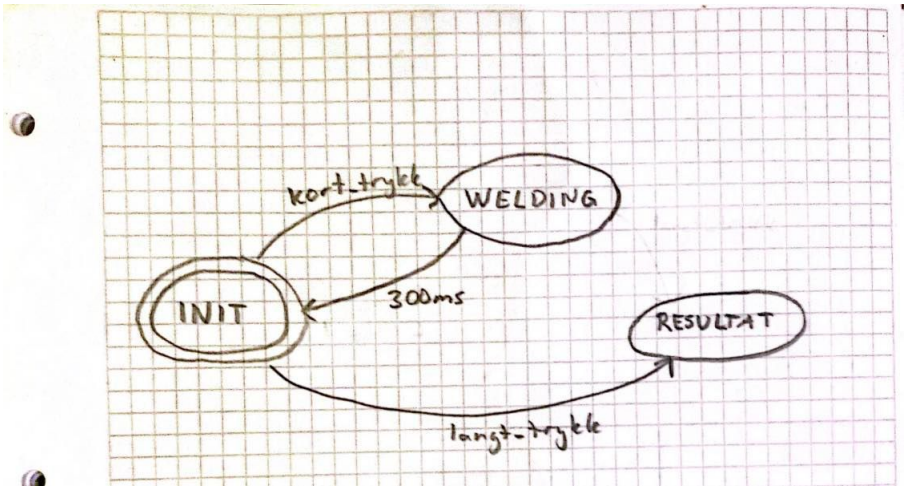
Nå som de to diodene lyste direkte på de to fototransistorene kunne vi undersøke verdiene som blir gitt ut på fototransistorene. Når vi nå kjørte koden fikk vi ut høye verdier når fototransistorene ble belyst og vesentlig lavere verdier når de ikke blir belyst. Avviket var fremdeles stort nok slik at det var tydelig når de ble belyst og og når de ikke ble belyst.

3.4 Løsning med fullstendig krets

3.4.1 Diagrammer

I denne delen legger vi fram de ulike diagrammene av den fullstendige kretsen vi har brukt for å løse oppgaven. Vi har laget kretsskjema for den elektriske oppkoblingen, samt et tilstandsdiagram som beskriver det fullstendige programmet på et overordnet nivå.

3.4.2 Tilstandsdiagram



Figur 3.6: Tilstandsdiagram som gir en overordnet oversikt over programmet.

Tilstandsdiagrammet vi har laget består av 3 tilstander. Under ser du forklaring på hva som skjer i de ulike tilstandene.

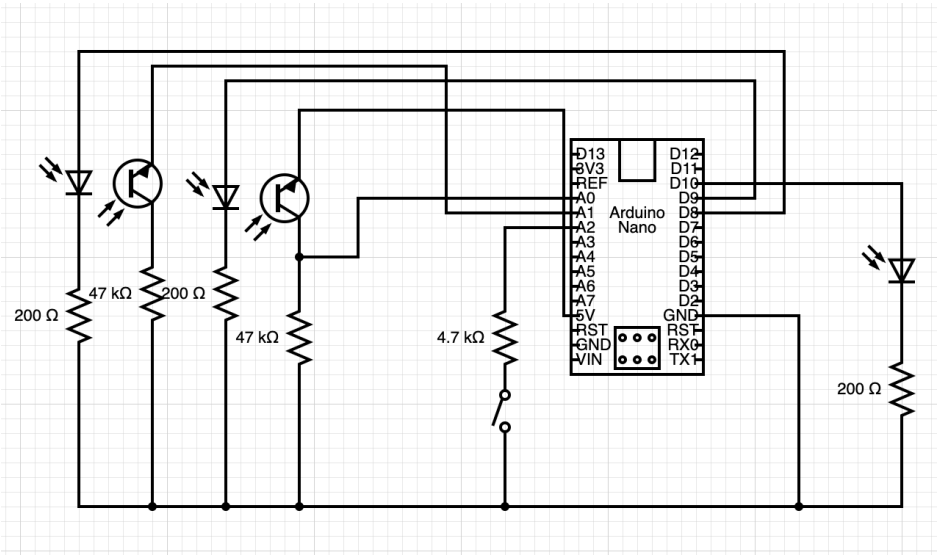
0 - INIT: Dette er klar tilstanden, hvor man venter på at det skal gis signal. Signalet for å starte sveisingen gis ved et kort trykk på knappen. For å få resultatet av sveisingen holder man inne knappen(altså et langt trykk).

1 - WELDING: Her har det blitt gitt et kort signal, altså et kort trykk på knappen. Da starter sveiseprosessen. Verktøyet beveger seg da inn til punktet. Der registrerer den om den oppfyller kravene til sveisingen, altså om den ene lysdioden blir belyst, mens den andre ikke blir det, samt at verktøyet er i punktet i den gitte tiden. Når sveisingen i punktet er ferdig og det har gått 300ms registreres resultatet før man sendes tilbake til INIT-tilstanden for å få ett nytt signal.

2 - RESULTAT: Hvis man gir et langt signal(langt trykk) avsluttes sveisingen. Da skrives resultatet ut, enten om sveisingen er godkjent eller ikke godkjent. En rød varsellampe tennes hvis sveisingen ikke er godkjent.

3.4.3 Koblingsskjema

Under ser du koblingsskjemaet for den fullstendige elektriske kretsen. Den består av tre lysdioder, to hvite og en rød, to fototransistorer og en bryter.

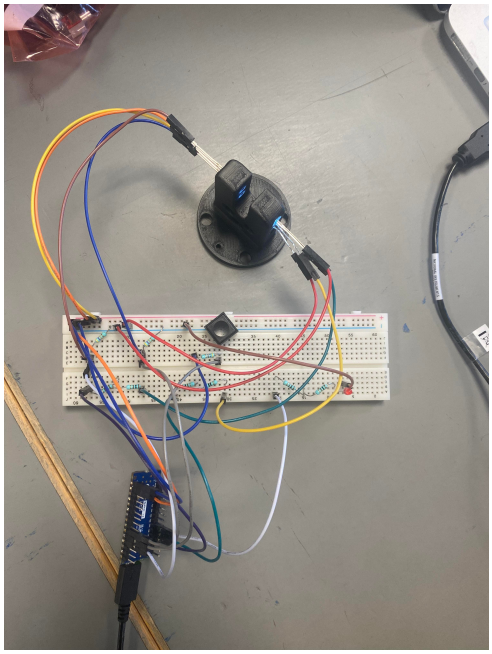


Figur 3.7: Koblingsskjema for fullstendig elektrisk krets

3.5 Oppkobling

For å koble opp den elektriske kretsen tok vi utgangspunkt i koblingskjemaet som er vist i seksjonen ovenfor 3.7. Som man kan se består kretsen av tre lysdioder, to fototransistor og en bryter. Selve arduinoen er koblet til en PC med en USB kabel. De to hvite lysdiode er koblet fra digitale innganger ettersom de settes enten til 1 eller 0, altså at de enten er på eller av. De kobles så i serie med en resistor på 200 Ω. Strømmen som går gjennom disse lysdiodene ligger på 16.5mA. Dette gir ett skarpt og godt lys, noe som er avgjørende ettersom disse plasseres i verktøyet og skal belyse fototransistorene. Den røde lysdioden er også koblet i serie med en motstand på 200 Ω. Denne lysdioden settes direkte i breadboardet og lyser hvis sveisingen ikke er ok. Fototransistorene er koblet fra analoge innganger ettersom vi skal bruke verdiene vi får ut fra de, altså ikke kun høy eller lav. Disse kobles

også i serie, men med en motstand på 47k. Dette fører til at det går en mye mindre strøm gjennom dem, men her er det belysningen fra diodene som er det viktige. Den siste komponenten er bryteren, som er koblet fra en analog inngang i serie med en 4.7k motstand og direkte til jord. Under kan du se hvordan den ferdigkoblede kretsen ser ut i praksis. Lysdiodene og fototransistorene er festet direkte i verktøyet, mens den røde lysdioden er koblet direkte i breadboard.



Figur 3.8: Oppkobling av elektrisk krets i praksis

3.6 Kode

I denne delen av oppgaven beskriver vi programmet vi har utviklet i Arduino IDE. Vi legger fram koden stegvis, alt fra definering av variabler og pinner, til case - setninger og løkker. Vi forklarer også hvordan den fungerer og hva som skjer i praksis når koden kjøres.

3.6.1 Setup

```
// Phototransistor setup
unsigned int Phototransistor = A0;
unsigned int Phototransistor1 = A1;
unsigned int lightVal;
unsigned int lightVal1;

// LED setup
unsigned int LED1 = 9;
unsigned int LED2 = 8;
```

Figur 3.9: Setup for fototransistor og LED

Her setter vi opp fototransistoren og LED- lysene som brukes. Vi har da satt fototransistorene til å lese fra de analoge pinnene A0 og A1, og LED-lysene fra de to digitale utgangene 9 og 8. I dette verktøyet er vi ikke interesserte i negative verdier, vi har derfor brukt "unsigned int" her for å fjerne muligheten til å lese et negativt tall.

```
// Case setup
const int INIT = 1;
const int WELDING = 2;
const int Results = 3;
unsigned state = INIT;

// ARRAY SETUP
#define maxtargets 10
bool WeldOK[maxtargets];
bool allWeldsOK;
uint8_t n = 0;
```

Figur 3.10: Setup for case

Som nevnt i tilstandsdiagrammet 3.6 , skal roboten gjennom ulike tilstander for å sjekke om sveisingen blir godkjent. Vi løser da dette ved bruk av "switch cases". Her setter vi opp de ulike tilstandene og et array som lagrer verdien til hver sveise test.

```
//Button Setup and RED LED
int RedLED= 7;
int buttonPin= A2;
int currentStateButton;

//TIME SETUP
const int SHORT_PRESS_TIME = 1000; // 1000 milliseconds
const int LONG_PRESS_TIME  = 1000; // 1000 milliseconds
int lastState = LOW;  // the previous state from the input pin
unsigned long pressedTime  = 0;
unsigned long releasedTime = 0;
```

Figur 3.11: Setup for knapp

Her har vi som nevnt tidligere brukt en trykknapp for å simulere et signal fra roboten, og et rødt lys for å simulere om det oppstod en feil under sveisingen. Vi har derfor definert trykknappen som analog inngang A2 og rødt LED-lys som digital inngang 7.

For å simulere et langt signal eller kort signal fra roboten. Har vi simulert dette ved å definere trykk tiden på knappen til at hvis den er under 1000 millisekunder så skal den fortsette sveisingen, og over 1000 millisekunder for å rapportere.

Her ser vi også at terskelverdien er satt til 400. Denne verdien ble valgt, siden vi gjorde et forsøk der vi koblet opp fototransitoren og led-lysene til verktøyet 3.3. Da fant vi ut hvilke verdier som returnerte da led-lysene stod på, og når led-lysene ble sperret. Fototransistorene returnerte 800-900 da lysene stod på, og 200-300 da lysene var av, vi valgte da naturligvis terskel verdien til å være 400.

Til slutt har vi også definert tiden sveisingen skal ta til å være 300 millisekunder, men siden vi ikke koblet opp mot roboten er ikke denne verdien bestemt. Den kan derfor endres til ønsket sveisetid etter oppkobling slik at den kan samarbeide bedre med roboten.

3.6.2 Initialiseringsrutine

```
void setup() {
  Serial.begin(9600);
  timeStart = millis();
//Inputs
pinMode(Phototransistor, INPUT);
pinMode(Phototransistor1,INPUT);
pinMode (buttonPin, INPUT);
//OUTPUTS
pinMode(LED1, OUTPUT);
pinMode(LED2,OUTPUT);
pinMode (RedLED, OUTPUT);
//LED ON
digitalWrite(LED1, HIGH); //skal muligens inn i loopen
digitalWrite(LED2,HIGH);
//START STATE
state = INIT;// start with INIT on
n = 0;
}
```

Figur 3.12: Void setup

Her definerer vi alle utgangene/inngangene, og slått på led-lysene på som standard. Og satt start tilstanden til "INIT".

3.6.3 Hovedløkke

```
void loop() {
  switch (state) {
    case INIT:    // Wait for signal from robot
      while (state == INIT) {
        //Serial.println("\nWaiting for signal from robot.");
        currentStateButton = digitalRead(buttonPin);

        if(lastState == HIGH && currentStateButton == LOW)  // button is pressed
          pressedTime = millis();
        else if(lastState == LOW && currentStateButton == HIGH){  // button is released
          releasedTime = millis();

        long pressDuration = releasedTime - pressedTime;

        if( pressDuration < SHORT_PRESS_TIME ){
          state = WELDING;
          Serial.println("\nA short press is detected");
        }
        if( pressDuration > LONG_PRESS_TIME ){
          Serial.println("\nA long press is detected");
          state = Results;
        }
      }
        // save the the last state
      lastState = currentStateButton;
      break;
      }
```

Figur 3.13: INIT

Her vil koden se annerledes ut når den kobles opp mot roboten. Vi har kun laget en kode for å simulere signalet fra roboten ved hjelp av en trykknapp. Her blir som sagt verdien fra knappen lest, og tiden noteres når den trykkes inn og slippes. Total trykk tid blir da regnet ut ved å ta tiden da knappen slippes minus da knappen trykkes inn. Og deretter gå i tilstanden "WELDING" hvis det er et kort trykk, og "Results" ved et langt trykk.

```
case WELDING:
  while (state == WELDING) {
    Serial.println("\nLeser verdier");
    lightVal = analogRead(Phototransistor);
    lightVal1= analogRead(Phototransistor1);
    if (lightVal > threshold  && lightVal1 < threshold ){ // RIKTIG
      timeElapsed = millis() - timeStart;
      if (timeElapsed > timeWeld) {
        Serial.println("\n ok test");
        //restart the timer and add weld ok to array
        timeStart = millis();
        WeldOK[n] = true;
        n++; //inkrementering av indeks i array'ene
        state = INIT;
        if (n >= maxtargets){
          Serial.println("\n Maximum number of targets exeeded! Starting over …");
          n = 0;
          allWeldsOK = true;
          state = INIT;
        }
      }
    }
```

Figur 3.14: WELDING

I "WELDING" tilstanden, begynner den å lese verdiene av fototransitorene og sjekker hvilken påstand som stemmer. Her ser vi at ønsket simulering for sveising, er når det kun kommer lys på den øverste fototransistoren, mens den andre er dekket til. Skjer dette, vil den deretter sjekke om den har sveiset etter ønsket sveisetid. Er begge disse kravene oppfylt, returnerer den da at sveisingen gikk bra og legger dette til i arrayet.

Vi har også lagt til en feilsøking, at hvis det oppstår for mange målinger så vil alt nullstilles og starte målingene på nytt. Dette er for å forhindre mulige feil der roboten sender flere signaler enn ønsket, som gjør at sveisingen kan starte opp på uønsket tid.

```
if (lightVal > threshold && lightVal1 > threshold) {  //Feil, begge lys er  på
  Serial.println("\n Ikke ok test");
  //is it time for the next state?
  timeElapsed = millis() - timeStart;
  if (timeElapsed > timeWeld) {
    //restart the timer and change the state
    timeStart = millis();
    allWeldsOK = false;
    WeldOK[n] = allWeldsOK;
    n++;
    state = INIT;
    if (n >= maxtargets){
        Serial.println("\n Maximum number of targets exeeded! Starting over …");
        n = 0;
        allWeldsOK = true;
        state = INIT;

    }

  }
}
```

Figur 3.15: Feil, begge lys er på

Her er et eksempel på et utfall som gjør at sveisingen ikke blir godkjent. Koden kjøres på samme måte som i godkjent sveising, men her sjekker den for mulige feil som kan oppstå. I dette eksempelet har vi satt at hvis begge lysverdiene overstiger terskelverdien, så returneres det en “allWeldOK = false;”. Vi har gjort det samme for de andre mulige utfallene også. Feilene som da kan oppstå er, begge lys er på eller at det går lys gjennom lysdiode 2 og ikke 1. Når “allWeldOK = false;” blir da hele arrayet satt til false", som vil si at verdien som returneres blir lik 0.

```
  case Results:
    while (state == Results) {
      for(int n = 0; n < maxtargets; n++){
        Serial.println(WeldOK[n]);
      }
      if (allWeldsOK == false){
       Serial.println("\n Ikke ok sveising");
       digitalWrite(RedLED,HIGH);
      }
      if (allWeldsOK == true){
       Serial.println("\n Ok sveising");
       digitalWrite(RedLED,LOW);
      }
        n = 0;
        allWeldsOK = true;
        state = INIT;
    break;
    }
 }
}
```

Figur 3.16: Results

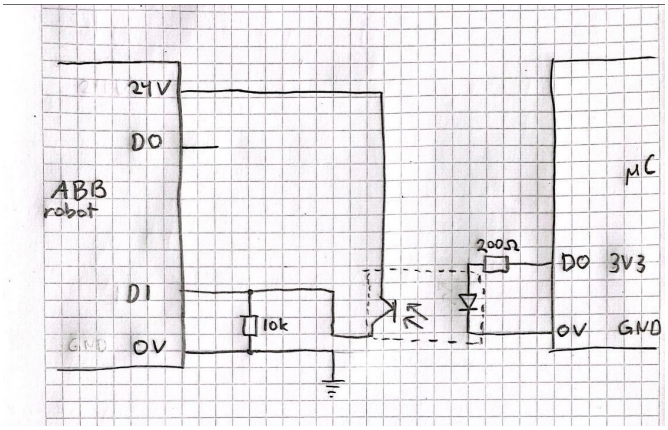
Når det kommer et langt signal, (simulert med et langt trykk på trykknappen), vil man gå i tilstanden "Results". Her bruker vi verdiene som samles opp fra "Welding". Her leses verdiene som returneres og skriver ut "ikke ok sveising" eller "ok sveising" ettersom verdiene som returneres er 0 eller 1. Etter resultatet skrives ut, resetter vi verdiene som samles opp og går tilbake til tilstanden "INIT".

3.7 Oppkobling mot robot

Vi har som nevnt tidligere ikke koblet fysisk opp mot roboten, så dette er hvordan man teoretisk kan koble opp (se 3.17). Vi har valgt å bruke en Optokobler og en "pull down" for å koble sammen ABB-roboten og mikrokontrolleren. Optokobleren er det som ligger inne i det stiplede område på koblingskjemaet ovenfor. Hensikten med dette er at strømmen som går gjennom roboten er langt over det mikrokontrolleren kan håndtere og motsatt. 24V som kommer ut fra roboten vil kortslutte mikrokontrolleren, mens 3,3V fra mikrokontrolleren kun vil være minimale endringer på roboten som vil bli tolket som støy.

En optokobler vil derfor være en løsningsmetode, siden den isolerer strømmene på hver side av kretsene for seg selv. "Pull downen" er da brukt for å få riktig logikk i kretsen. Vi vil at roboten skal holde seg lav når det ikke går en strøm gjennom kretsen. Logikken for kretsen vises i diagrammet lenger nede i seksjonen 3.18.

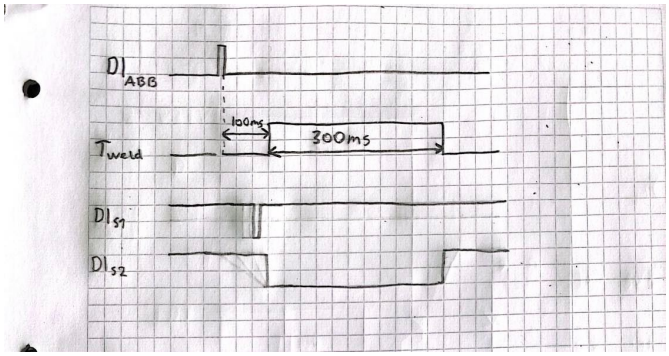
Under ser du koblingsskjemaet for hvordan vi hadde tenkt å koble mikrokontrolleren opp mot roboten.



Figur 3.17: Koblingskjema for oppkobling mot robot

Som man ser på skjemaet under 3.18 starter ikke sveiseprosessen før det gis ett kort signal fra ABB robot. Når det korte signalet gis begynner roboten å

bevege seg mot sveisepunkt. Vi har satt tiden til å være 100 ms før roboten er i sveisepunktet. Den ytterste lysdioden belyser fototransistoren gjennom hele prosessen, med unntak av når verktøyet føres inn i punkt. Da vil det være et kort tidsrom hvor lyset brytes som du ser i skjemaet under for S1. S2 vil forbli høyt fram til sveiseprosessen starter. Da vil den gå lavt og forbli lavt gjennom hele sveiseprosessen. Selve sveiseprosessen varer i 300ms. Ettersom vi ikke fysisk har koblet opp mot roboten er dette slik vi hadde tenkt at det vil fungere og tidene er derfor ikke reelle, men kun definert av oss.



Figur 3.18: Skjema som viser logikk

3.7.1 Kode

Koden her i rapid er ikke ferdigstilt, men kun et løsningsforslag som må videre utvikles slik at arduino og roboten snakker sammen. Tidene og avstanden til de forskjellige punktene må også være mer presise slik at verktøyet treffer ørene som ønsket.

3.7.2 Oppsett

```
MODULE PunktSveising


    TASK PERS wobjdata wobjTableN:=[FALSE,TRUE,"",[[150,-500,8],[0.707106781,0,0,-0.707106781]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget target_K0:=[[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

    CONST num SveisHeight := 30;
    CONST num safeHeight := 200;
    CONST num nPos := 5;

    CONST num ShortPulse := 0.1;
    CONST num LongPulse := 1.5;

    VAR num LastPostNo := 0;
    VAR num nOnPos{nPos} := [1, 1, 1, 1, 1];
    VAR robtarget targets{nPos} := [
        [[0,-180,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[180,-180,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[180,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[-180,180,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
];
    VAR num currentPos := 1;
    VAR BOOL run:=TRUE;
```

Figur 3.19: Oppsett

Her har vi som tidligere nevnt, tatt utgangspunktet i RS3. I koden ovenfor har vi definert de ulike posisjonene til punktene. Her er det valgt 5 punkter på bordet som roboten skal innom og sjekke om sveisingen blir godkjent.

3.7.3 Proc_main

```
PROC main()
    MoveL Offs(target_K0, 0, 0, safeHeight),v500,z10,tGripper\WObj:=wobjTableN;

    WHILE run DO
        TPWrite "Test";
        TPReadFK svar, "Start";
        IF svar= 1 THEN
            TPReadnum til " Hva er start posisjonen";
            moveBane til;
        ENDIF
    endwhile


ENDPROC
```

Figur 3.20: Main

Her er koden for det som skal vises på flexpendanten. Her har man alternativet til å trykke start som da vil kjøre roboten gjennom punktene på bordet. Man må også oppgi startposisjonen for der sveisingen skal starte.

3.7.4 Proc moveSveis

```
PROC moveSveis(num fra, num til)
    VAR robtarget p1;
    VAR robtarget p2;
    PulseDO(FALSE);
    p1 := Offs(targets{fra}, 0, 0, SveisHeight*nOnPos{fra}-30);
    p2 := Offs(targets{til}, 0, 0, SveisHeight*nOnPos{til});
    MoveJ Offs(p1, 30, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
    MoveJ Offs(p1, 30, 0, 5),v50,z10,tGripper\WObj:=wobjTableN;
    PulseDO(ShortPulse);
    MoveL Offs(p1, 10, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
    MoveL Offs(p1, 0, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
    WaitTime 0.2;
    MoveL Offs(p1, 0, 0, 30),v500,z10,tGripper\WObj:=wobjTableN;
    MoveJ Offs(p1, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;

    MoveJ Offs(p2, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
    MoveJ Offs(p2, 0, 0, 30),v500,z10,tGripper\WObj:=wobjTableN;
    PulseDO(ShortPulse);
    MoveL Offs(p2, 10, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
    MoveL Offs(p2, 0, 0, 5),v10,fine,tGripper\WObj:=wobjTableN;
    WaitTime 0.2;
    MoveL Offs(p2, 0, 0, 60),v500,z10,tGripper\WObj:=wobjTableN;
    MoveL Offs(p2, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
ENDPROC
```

Figur 3.21: Sveising

For at sveisingen skal være mest mulig effektiv i praksis, må man starte opp sveisingen rett før den kommer til punktet. Dette løser vi ved å sende en "Shortpulse" fra roboten rett før den har kommet til punktet, som da sender et kort signal til arduino som starter "welding". Vi regner med at det går noen millisekunder før den er punktet. Vi setter derfor "waitime" til 0.2 og ikke 0.3 som er like lenge som sveise tiden. Roboten står da i punktet i 0.2 sekunder før den beveger seg mot det andre punktet. Her må koden som sagt finjusteres slik at tidene fungerer optimalt.

3.7.5 Proc PulseDo

```
    PROC PulseDO(num duration)
        SetDo ABB_Scaleable_IO_DO1, 1;
        WaitTime duration;
        SetDo ABB_Scaleable_IO_DO1, 0;
    ENDPROC
ENDMODULE
```

Figur 3.22: Signal fra roboten

"PulseDo" ble brukt i sveise prosessen for å sende et signal fra roboten. Den fungerer slik at den sender et signal som tilsvarer "duration", som ble definert tidligere til å være enten "ShortPulse" eller "LongPulse".

3.7.6 Proc moveBane

```
PROC moveBane(num fraPostNo)
    FOR i from 0 to nPos DO
        IF i <> fraPostNo THEN
            LastPostNo := fraPostNo + 1;
        ENDIF
    ENDFOR
    FOR i FROM 0 TO nOnPos{LastPostNo} DO
        moveSveis LastPostNo, i;
    ENDFOR
    PulseDO(LongPulse)
    result := DInput(ABB_Scaleable_IO_DO1)
ENDPROC
```

Figur 3.23: Loop gjennom punktene

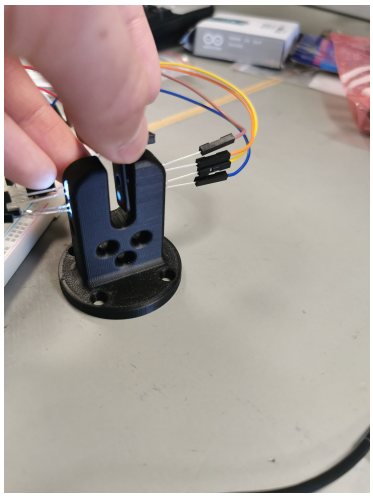
3.7 Oppkobling mot robot

Her er koden for å kjøre gjennom banen. Her definerer vi siste posisjon som start posisjonen + 1. Denne loopen vil kjøre helt til den har vært innom alle punktene, etter det vil roboten sende et langt signal som skal spør om en rapport fra arduino. Arduino skal gi tilbakemelding om sveise prosessen ble godkjent eller ikke, og returnerer enten 0 eller 1 tilbake til den digitale inngangen på roboten.

Kapittel 4

Resultat

Ettersom vi ikke koblet opp verktøyet fysisk med roboten, testet vi verktøyet manuelt. Vi holdt punktene i gaffelen 4.1, og simulerte korte og lange signaler ved hjelp av trykknappen.



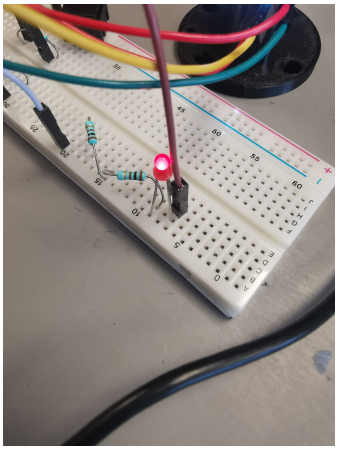
Figur 4.1: Manuell test av punktsveising

Resultatet var som forventet. Testen vi kjørte med en ikke godkjent sveis, vil fortsatt resultere i et lavt signal selv om de andre er godkjent og dermed

skriver ut “ikke ok sveising” og den rød varsellampen lyser.

```
A long press is detected
0
1
1
1
1
0
1
0
0
0

Ikke ok sveising
```

Figur 4.2: Array_Feil

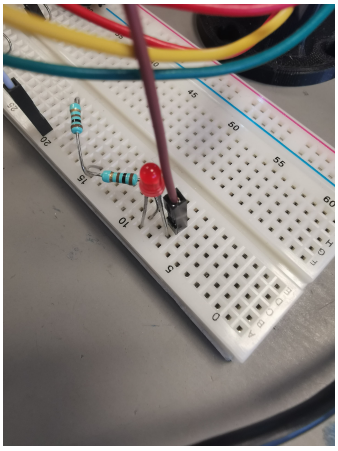


Figur 4.3: Lys_på

Testen for godkjent sveising foregikk på samme måte som tidligere, eneste forskjellen er at vi holdt punktet på samme plass slik at alle testene ble godkjent. Vi får da ut et høyt signal, og led-lyset forblir da av og vi får ut “Ok sveising”

```
A long press is detected
1
1
1
1
1
0
1
0
0
0

Ok sveising
```

Figur 4.4: Array_riktig



Figur 4.5: Lys_av

Kapittel 5

Diskusjon og konklusjon

Oppgaven vår gikk ut på å utvikle ett program som skal kunne simulere en automatisk punktsveising. Ettersom kodingen av dataprogrammet tok mye lenger tid enn først antatt ble vi nødt til å løse oppgaven på en litt alternativ måte. Vi valgte å gå vekk fra automatisk styring fra roboten og prioriterte heller å få til en fungerende manuell løsning av punktsveisingen.

5.1 Valg av løsning

For å sette igang punktsveisingen, valgte vi å bruke en trykknapp. Denne fungerer i praksis som et signal fra roboten. Ved ett kort trykk settes prosessen i gang og ved ett langt trykk avsluttes prosessen. Dette fungerte egentlig meget bra etter sin hensikt. Når knappen trykkes, føres gaffelen bort til punktet og blir i punktet en viss tid, deretter leser dataprogrammet om betingelsene oppfylles, før den returnerer resultatene. Når punktsveisingen er ferdig sendes ett nytt signal og den samme prosedyren gjentas. I utgangspunktet skulle roboten sendt disse signalene for så å gå automatisk bort til punktet, lese av og returnere til startposisjonen.

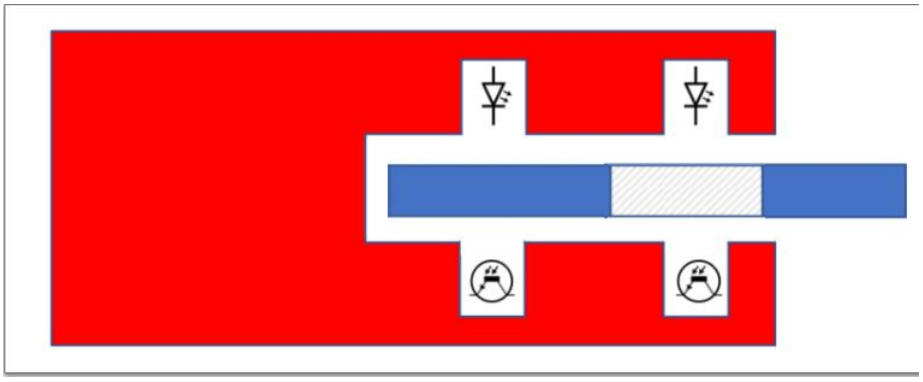
Det som er bra med den løsningen vi valgte er at punktsveisingen blir gjennomført etter sin hensikt. Programmet utfører det den skal og forstår betingelsene som er satt, før den returnerer resultatene. Den eneste faktoren

er at vi er nødt til å manuelt bevege gaffelen til punktet, for så å holde den der i en gitt tid og ta den tilbake til startposisjonen. Optimalt skulle hele denne prosessen vært automatisert, men vi ble som sagt nødt til å prioritere å få det til manuelt siden ting tok lenger tid enn forventet.

5.2 Valg av design

Vi valgte å gå for et gaffelformet verktøy, hvor man kunne sette inn to lysdioder og to fototransistorer overfor hverandre. På denne måten kunne vi få ett konsentrert hvitt lys direkte på fototransistorene. I starten hadde vi noe problemer med belysningen, på den måten at vi ikke fikk noen særlig gode verdier ut av fototransistoren. Dette løste seg imidlertid ganske greit når lysdiodene ble satt direkte inn i verktøyet slik at det ble minimalt med forstyrrende omgivelseslys. Da fikk vi ut gode verdier, hvor avviket var stort nok mellom belyst og ikke belyst. Dette mener vi også viser at designet på verktøyet passer utmerket for å løse oppgaven.

En endringen vi gjorde i forhold til det som blir beskrevet i problemstillingen er at vi valgte den yttereste lysdioden og ikke den innerste lysdioden(som vises på figur 1.1 i problemstillingen) som dioden som skal lyse gjennom punktet. Grunnen til at vi gjorde denne endringen var at når vi designet punktene, ble hullet plassert noe lavere enn ønsket. Dette gjorde det umulig for verktøyet å gå til punktet ovenfra og få lysstrålen gjennom punktet fra den innerste lysdioden. Derfor valgte vi heller å benytte den ytterste, slik at vi kunne gå i punktet ovenfra. Eventuelt kunne dette bli løst ved å gå inn i punktet fra siden, men ettersom dette ikke har noen hensikt for funksjonaliteten til verktøyet droppet vi det. Under ser du en oversikt over verktøyet i et punkt, hvor den ytterste lysdioden belyser fototransistoren.



Figur 5.1: Oversikt over gaffel, etter endring

5.3 Videreutvikling

Kodene er som sagt ikke helt fullstendige fordi vi har ikke opprettet en kobling mellom ABB-roboten og mikrokontrolleren. Her har vi kun brukt teoretiske porter i Rapid, og en trykknapp i Arduino. For å få en fullstendig kode må man derfor finne definere inngangene/utgangene og få dem til å "kommunisere" sammen.

Det skal også være mulig å sveise selv om punktene på bordet ligger på forskjellige vinkler, dette er da ikke løst i rapid koden. Her er det kun lagt til faste definerte punkter på bordet, roboten vil derfor kun treffe ørene hvis de ligger i samme posisjon og vinkel.

Det ble også tidligere nevnt at tidene ikke er optimalisert. For at roboten skal jobbe mest mulig effektivt, må det da bli gjort flere målinger og finjusteringer som gjør at roboten kan bevege seg raskest mulig, men fortsatt få tilstrekkelig med tid til å lese av verdiene.

5.4 Konklusjon

For å konkludere vil vi si at dette har vært en krevende, men svært lærerik oppgave. Underveis i oppgaven fant vi ut at ting var mer tidkrevende enn vi først antok. Dette gjorde at vi tok ett valg mot slutten, nemlig å prioritere å få til en fungerende manuell løsning av oppgaven. En trykknapp erstattet signalet fra roboten og vi måtte fysisk flytte på verktøyet til punktene. Dette ble ikke en optimal løsning, men den fungerer i den forstand at den leser og tolker informasjonen i sveisepunktene og rapporterer tilbake.

En av grunnene til at denne bacheloroppgaven har vært tidkrevende for oss, er på grunn av en noe redusert bakgrunnkunnskap. Vi hadde lite erfaringer med Arduino og mikrokontroller generelt, noe som medførte at vi brukte mye tid på forstå og sette opp de elektriske kretsene og selve kodingen av mikrokontrolleren. Men når det er sagt, har dette prosjektet gitt oss masse kunnskap om hvordan en mikrokontroller fungerer og elektriske kretser. Vi har nå fått et bedre startpunkt som vi kan bruke videre i fremtiden, og et bedre innblikk i hvordan det er jobbe i elektronikken.

Bibliografi

[1] *analogReference()*. URL: https://www.arduino.cc/reference/en/language/functions/analog-io/analogreference/. (accessed: 15.03.2022).

[2] Paul Bjørn Andersen. *Fototransistor*. URL: https://snl.no/fototransistor. (accessed: 03.02.2022).

[3] Paul Bjørn Andersen. *Halvledere*. URL: https://snl.no/halvledere. (accessed: 03.02.2022).

[4] *Arduino Nano 33 Iot*. URL: https://store-usa.arduino.cc/products/arduino-nano-33-iot. (accessed: 12.04.2022).

[5] *Arduino Tutorial 3: Understanding How Breadboards Work*. URL: https://www.youtube.com/watch?v=CfdaJ4z4u4w&list=LL&index=2. (accessed: 20.02.2022).

[6] *Beginners: using the switch - case statement*. URL: https://forum.arduino.cc/t/beginners-using-the-switch-case-statement/680177. (accessed: 08.03.2022).

[7] Tor John Rødsås Einar Halmøy Almar Almar Næss. *Sveising*. URL: https://snl.no/sveising. (accessed: 20.01.2022).

[8] *Functions*. URL: https://www.arduino.cc/reference/tr/. (accessed: 05.03.2022).

[9] *Hva er Fusion360*. URL: https://www.autodesk.no/products/fusion-360/overview?term=1-YEAR&tab=subscription. (accessed: 11.04.2022).

[10] Jacob Linder. *Lysdiode*. URL: https://snl.no/lysdiode. (accessed: 03.02.2022).

[11] *switch...case*. URL: https://www.arduino.cc/reference/tr/language/structure/control-structure/switchcase/. (accessed: 05.03.2022).

[12] *Technical reference manual RAPID Instructions, Functions and Data types*. URL: https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf?fbclid=IwAR2rOWtGLweYWRZv65t87w2dEnj_VgoBgPPUTqjFumIYB3azmUmpMJopsvI. (accessed: 13.04.2021).

[13] *The world's most used offline programming*. URL: https://new.abb.com/products/robotics/robotstudio. (accessed: 27.04.2022).

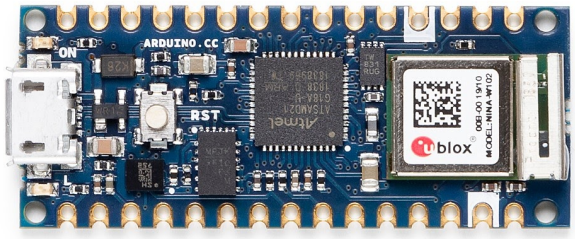
[14] *What is an Optocoupler and How it Works*. URL: https://www.jameco.com/Jameco/workshop/Howitworks/what-is-an-optocoupler-and-how-it-works.html. (accessed: 08.04.2022).

[15] *WP1503SGC*. URL: https://www.digikey.no/no/products/detail/kingbright/WP1503SGC/3084100?utm_adgroup=LED%20Indication%20-%20Discrete&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Optoelectronics&utm_term=&productid=3084100&gclid=CjwKCAjw9qiTBhBbEiwAp-GE0eY6Hqu0LMc554zu_bN2W4ijc50Aa7TwmTL0cA4Ue5UHrw0Ecq8X2RoCB44QAvD_BwE. (accessed: 02.03.2022).

Vedlegg A

Datablad

Product Reference Manual
SKU: ABX00027



Description

Nano 33 IoT is a miniature sized module containing a Cortex M0+ SAMD21 processor, a WiFi+BT module based on ESP32, a crypto chip which can securely store certificates and pre-shared keys and a 6 axis IMU. The module can either be mounted as a DIP component (when mounting pin headers), or as a SMT component, directly soldering it via the castellated pads.

Target areas:

Maker, enhancements, basic IoT application scenarios

Features

- **SAMD21G18A**
  - **Processor**
    - 256KB Flash
    - 32KB Flash
    - Power On Reset (POR) and Brown Out Detection (BOD)
  - **Peripherals**
    - 12 channel DMA
    - 12 channel event system
    - 5x 16 bit Timer/Counter
    - 3x 24 bit timer/counter with extended functions
    - 32 bit RTC
    - Watchdog Time
    - CRC-32 generator
    - Full speed Host/Device USB with 8 end points
    - 6x SERCOM (USART, I^2C, SPI, LIN)
    - Two channel I^2S
    - 12 bit 350ksps ADC (up to 16 bit with oversampling)
    - 10 bit 350ksps DAC
    - External Interrupt Controller (up to 16 lines)

- **Nina W102**
  - **Module**
    - Dual Core Tensilica LX6 CPU at up to 240MHz
    - 448 KB ROM, 520KB SRAM, 2MB Flash
  - **WiFi**
    - IEEE 802.11b up to 11Mbit
    - IEEE 802.11g up to 54MBit
    - IEEE 802.11n up to 72MBit
    - 2.4 GHz, 13 channels
    - 16dBm output power
    - 19 dBm EIRP
    - -96 dBm sensitivity
  - **Bluetooth® BR/EDR**
    - Max 7 peripherals
    - 2.4 GHz, 79 channels
    - Up to 3 Mbit/s
    - 8 dBm output power at 2/3 Mbit/s
    - 11 dBm EIRP at 2/3 Mbit/s
    - -88 dBm sensitivity
  - **Bluetooth® Low Energy**
    - Bluetooth® 4.2 dual mode
    - 2.4GHz 40 channels
    - 6 dBm output power
    - 9 dBm EIRP
    - -88 dBm sensitivity
    - Up to 1 Mbit/
  - **MPM3610** (DC-DC)
    - Regulates input voltage from up to 21V with a minimum of 65% efficiency @minimum load
    - More than 85% efficiency @12V
  - **ATECC608A** (Crypto Chip)
    - Cryptographic co-processor with secure hardware based key storage
    - Protected storage for up to 16 keys, certificates or data
    - ECDH: FIPS SP800-56A Elliptic Curve Diffie-Hellman
    - NIST standard P256 elliptic curve support
    - SHA-256 & HMAC hash including off-chip context save/restore
    - AES-128 encrypt/decrypt, galois field multiply for GCM

- **LSM6DSL** (6 axis IMU)
  - Always-on 3D accelerometer and 3D gyroscope
  - Smart FIFO up to 4 KByte based
  - ±2/±4/±8/±16 g full scale
  - ±125/±250/±500/±1000/±2000 dps full scale

Contents

1 The Board

As all Nano form factor boards, Nano 33 IoT does not have a battery charger but can be powered through USB or headers.

NOTE: Arduino Nano 33 IoT only supports 3.3V I/Os and is **NOT** 5V tolerant so please make sure you are not directly connecting 5V signals to this board or it will be damaged. Also, as opposed to Arduino Nano boards that support 5V operation, the 5V pin does NOT supply voltage but is rather connected, through a jumper, to the USB power input.

1.1 Application Examples

Weather station: Using the Arduino Nano 33 IoT together with a sensor and a OLED display, we can create a small weather station communicating temperature, humidity etc. directly to your phone.

Air quality monitor: Bad air quality may have serious effects on your health. By assembling the Nano 33 IoT, with a sensor and monitor you can make sure that the air quality is kept in indoor-environments. By connecting the hardware assembly to an IoT application/API, you will receive real time values.

Air drum: A quick and fun project is to create a small air drum. Connect your Nano 33 IoT and upload your sketch from the Create Web Editor and start creating beats with your audio workstation of your choice.

2 Ratings

2.1 Recommended Operating Conditions

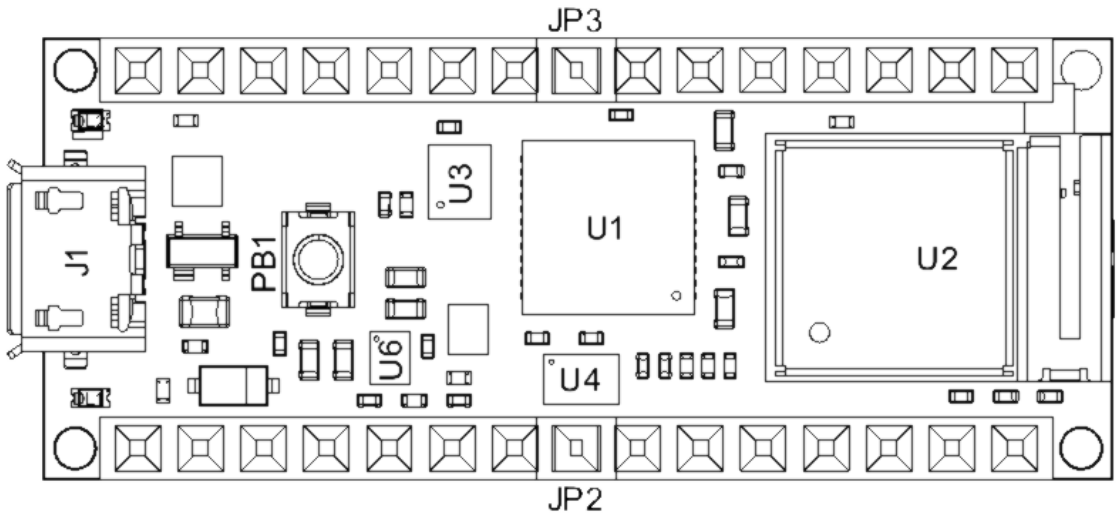
Symbol	Description	Min	Max
	Conservative thermal limits for the whole board:	-40 °C (40 °F)	85°C (185 °F)

2.2 Power Consumption

Symbol	Description	Min	Typ	Max	Unit
VINMax	Maximum input voltage from VIN pad	-0.3	-	21	V
VUSBMax	Maximum input voltage from USB connector	-0.3	-	21	V
PMax	Maximum Power Consumption	-	-	TBC	mW

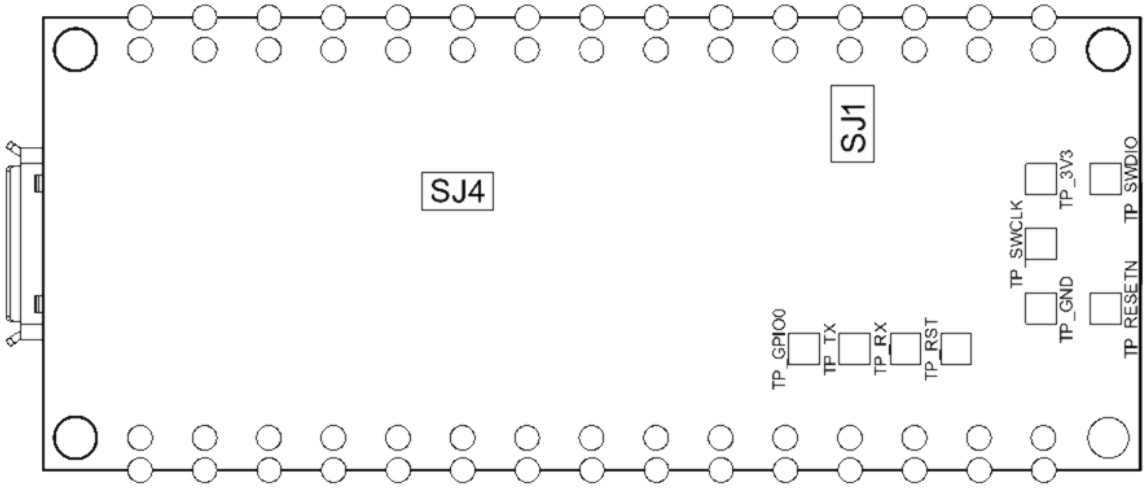
3 Functional Overview

3.1 Board Topology



Board topology top

Ref.	Description	Ref.	Description
U1	ATSAMD21G18A Controller	U3	LSM6DSOXTR IMU Sensor
U2	NINA-W102-00B WiFi/BLE Module	U4	ATECC608A-MAHDA-T Crypto Chip
J1	Micro USB Connector	PB1	IT-1185-160G-GTR Push button



Board topology bottom

Ref.	Description	Ref.	Description
SJ1	Open solder bridge (VUSB)	SJ4	Closed solder bridge (+3V3)
TP	Test points	xx	Lorem Ipsum

3.2 Processor

The Main Processor is a Cortex M0+ running at up to 48MHz. Most of its pins are connected to the external headers, however some are reserved for internal communication with the wireless module and the on-board internal I^2C peripherals (IMU and Crypto).

NOTE: As opposed to other Arduino Nano boards, pins A4 and A5 have an internal pull up and default to be used as an I^2C Bus so usage as analog inputs is not recommended.

Communication with NINA W102 happens through a serial port and a SPI bus through the following pins.

SAMD21 Pin	SAMD21 Acronym	NINA Pin	NINA Acronym	Description
13	PA08	19	RESET_N	Reset
39	PA27	27	GPIO0	Attention Request
41	PA28	7	GPIO33	Acknowledge
23	PA14	28	GPIO5	SPI CS
21	GPIO19	UART RTS		
24	PA15	29	GPIO18	SPI CLK
20	GPIO22	UART CTS		
22	PA13	1	GPIO21	SPI MISO
21	PA12	36	GPIO12	SPI MOSI
31	PA22	23	GPIO3	Processor TX Nina RX
32	PA23	22	GPIO1	Processor RX Nina TX

3.3 WiFi/BT Communication Module

Nina W102 is based on ESP32 and is delivered with a pre-certified software stack from Arduino. Source code for the firmware is available **[9]**.

NOTE: Reprogramming the wireless module's firmware with a custom one will invalidate compliance with radio standards as certified by Arduino, hence this is not recommended unless the application is used in private laboratories far from other electronic equipment and people. Usage of custom firmware on radio modules is the sole responsibility of the user.

Some of the module's pins are connected to the external headers and can be directly driven by ESP32 provided SAMD21's corresponding pins are aptly tri-stated. Below is a list of such signals:

SAMD21 Pin	SAMD21 Acronym	NINA Pin	NINA Acronym	Description
48	PB03	8	GPIO21	A7
14	PA09	5	GPIO32	A6
8	PB09	31	GPIO14	A5/SCL
7	PB08	35	GPIO13	A4/SDA

3.4 Crypto

The crypto chip in Arduino IoT boards is what makes the difference with other less secure boards as it provides a secure way to store secrets (such as certificates) and accelerates secure protocols while never exposing secrets in plain text.

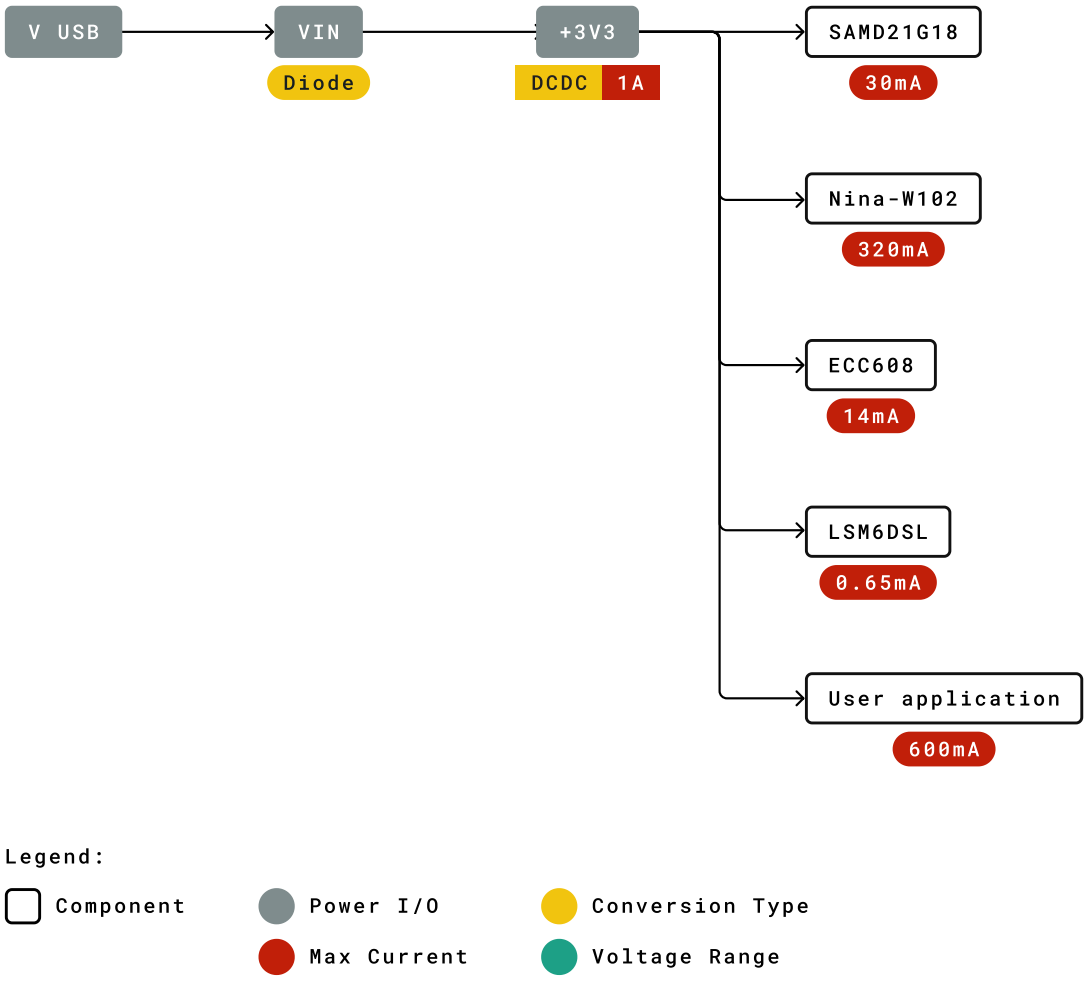
Source code for the Arduino Library that supports the Crypto is available **[10]**

3.5 IMU

Arduino Nano 33 IoT has an embedded 6 axis IMU which can be used to measure board orientation (by checking the gravity acceleration vector orientation) or to measure shocks, vibration, acceleration and rotation speed.

Source code for the Arduino Library that supports the IMU is available **[11]**

3.6 Power Tree



Power tree

4 Board Operation

4.1 Getting Started - IDE

If you want to program your Arduino 33 IoT while offline you need to install the Arduino Desktop IDE [1] To connect the Arduino 33 IoT to your computer, you'll need a Micro-B USB cable. This also provides power to the board, as indicated by the LED.

4.2 Getting Started - Arduino Web Editor

All Arduino boards, including this one, work out-of-the-box on the Arduino Web Editor [2], by just installing a simple plugin.

The Arduino Web Editor is hosted online, therefore it will always be up-to-date with the latest features and support for all boards. Follow **[3]** to start coding on the browser and upload your sketches onto your board.

4.3 Getting Started - Arduino IoT Cloud

All Arduino IoT enabled products are supported on Arduino IoT Cloud which allows you to Log, graph and analyze sensor data, trigger events, and automate your home or business.

4.4 Sample Sketches

Sample sketches for the Arduino 33 IoT can be found either in the "Examples" menu in the Arduino IDE or in the "Documentation" section of the Arduino Pro website [4]

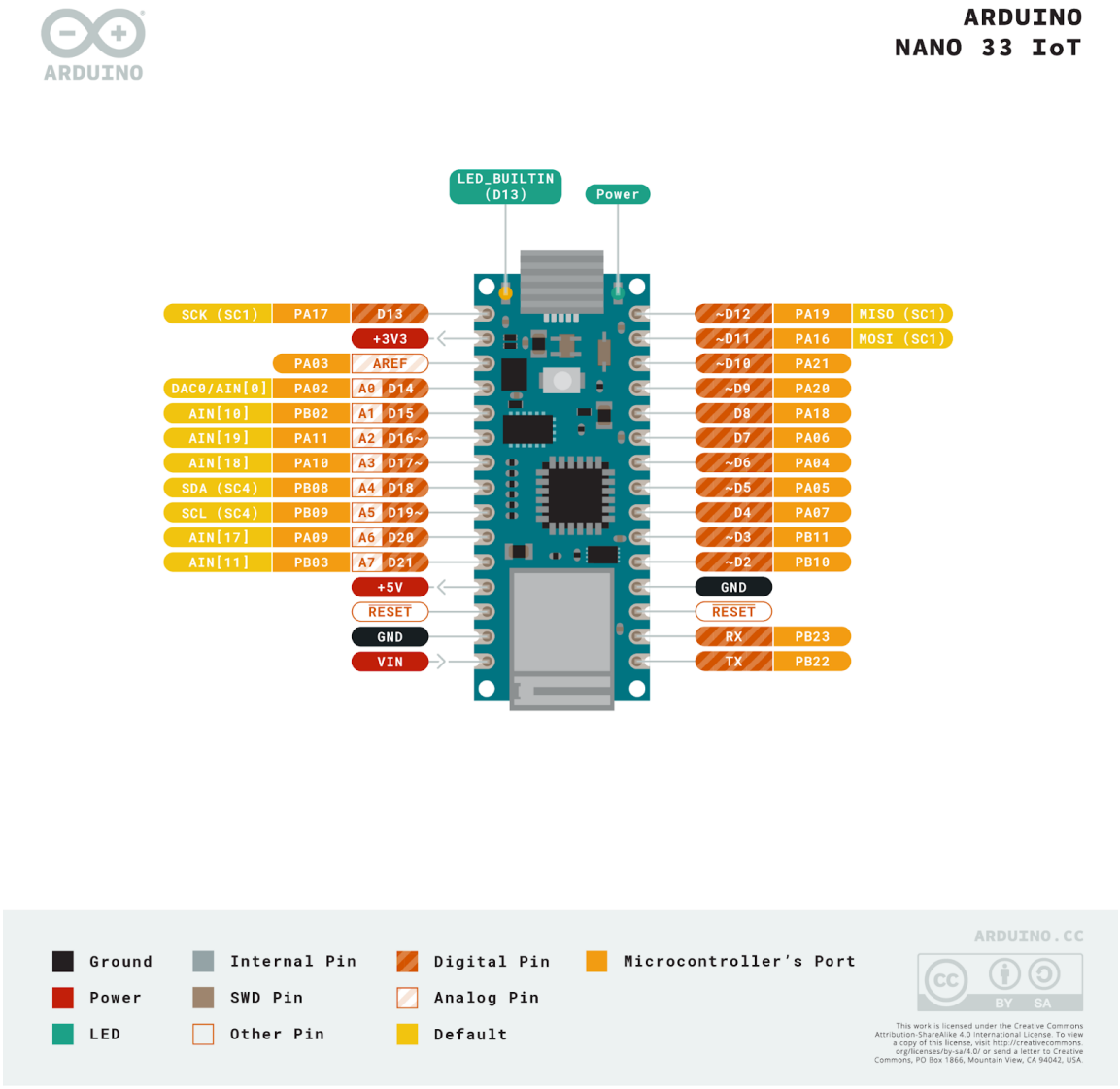
4.5 Online Resources

Now that you have gone through the basics of what you can do with the board you can explore the endless possibilities it provides by checking exciting projects on ProjectHub **[5]**, the Arduino Library Reference **[6]** and the online store **[7]** where you will be able to complement your board with sensors, actuators and more

4.6 Board Recovery

All Arduino boards have a built-in bootloader which allows flashing the board via USB. In case a sketch locks up the processor and the board is not reachable anymore via USB it is possible to enter bootloader mode by double-tapping the reset button right after power up.

5 Connector Pinouts



Pinout

5.1 USB

Pin	Function	Type	Description
1	VUSB	Power	Power Supply Input. If board is powered via VUSB from header this is an Output **(1)**
2	D-	Differential	USB differential data -
3	D+	Differential	USB differential data +
4	ID	Analog	Selects Host/Device functionality
5	GND	Power	Power Ground

1. The board can support USB host mode only if powered via the V_{USB} pin and if the jumper close to the VUSB pin is shorted.

5.2 Headers

The board exposes two 15 pin connectors which can either be assembled with pin headers or soldered through castellated vias.

Pin	Function	Type	Description
1	D13	Digital	GPIO
2	+3V3	Power Out	Internally generated power output to external devices
3	AREF	Analog	Analog Reference; can be used as GPIO
4	A0/DAC0	Analog	ADC in/DAC out; can be used as GPIO
5	A1	Analog	ADC in; can be used as GPIO
6	A2	Analog	ADC in; can be used as GPIO
7	A3	Analog	ADC in; can be used as GPIO
8	A4/SDA	Analog	ADC in; I2C SDA; Can be used as GPIO **(1)**
9	A5/SCL	Analog	ADC in; I2C SCL; Can be used as GPIO **(1)**
10	A6	Analog	ADC in; can be used as GPIO
11	A7	Analog	ADC in; can be used as GPIO
12	VUSB	Power In/Out	Normally NC; can be connected to VUSB pin of the USB connector by shorting a jumper
13	RST	Digital In	Active low reset input (duplicate of pin 18)
14	GND	Power	Power Ground
15	VIN	Power In	Vin Power input
16	TX	Digital	USART TX; can be used as GPIO
17	RX	Digital	USART RX; can be used as GPIO
18	RST	Digital	Active low reset input (duplicate of pin 13)
19	GND	Power	Power Ground
20	D2	Digital	GPIO
21	D3/PWM	Digital	GPIO; can be used as PWM
22	D4	Digital	GPIO
23	D5/PWM	Digital	GPIO; can be used as PWM
24	D6/PWM	Digital	GPIO, can be used as PWM
25	D7	Digital	GPIO
26	D8	Digital	GPIO

Pin	Function	Type	Description
27	D9/PWM	Digital	GPIO; can be used as PWM
28	D10/PWM	Digital	GPIO; can be used as PWM
29	D11/MOSI	Digital	SPI MOSI; can be used as GPIO
30	D12/MISO	Digital	SPI MISO; can be used as GPIO

5.3 Debug

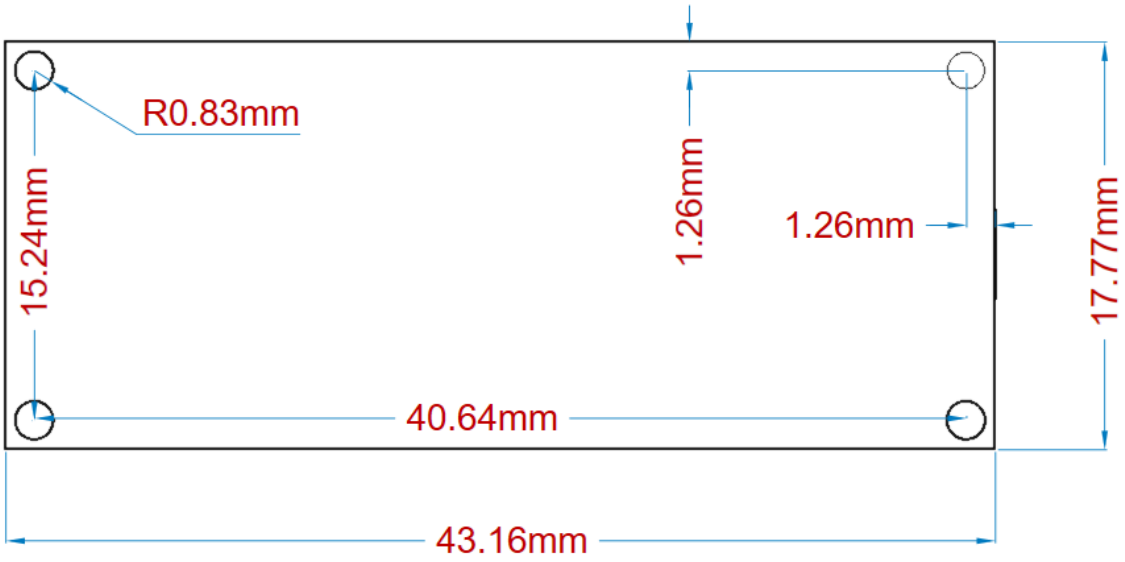
On the bottom side of the board, under the communication module, debug signals are arranged as 3x2 test pads with 100 mil pitch. Pin 1 is depicted in Figure 3 – Connector Positions

Pin	Function	Type	Description
1	+3V3	Power Out	Internally generated power output to be used as voltage reference
2	SWD	Digital	SAMD11 Single Wire Debug Data
3	SWCLK	Digital In	SAMD11 Single Wire Debug Clock
4	UPDI	Digital	ATMega4809 update interface
5	GND	Power	Power Ground
6	RST	Digital In	Active low reset input

6 Mechanical Information

6.1 Board Outline and Mounting Holes

The board measures are mixed between metric and imperial. Imperial measures are used to maintain a 100 mil pitch grid between pin rows to allow them to fit a breadboard whereas board length is Metric.

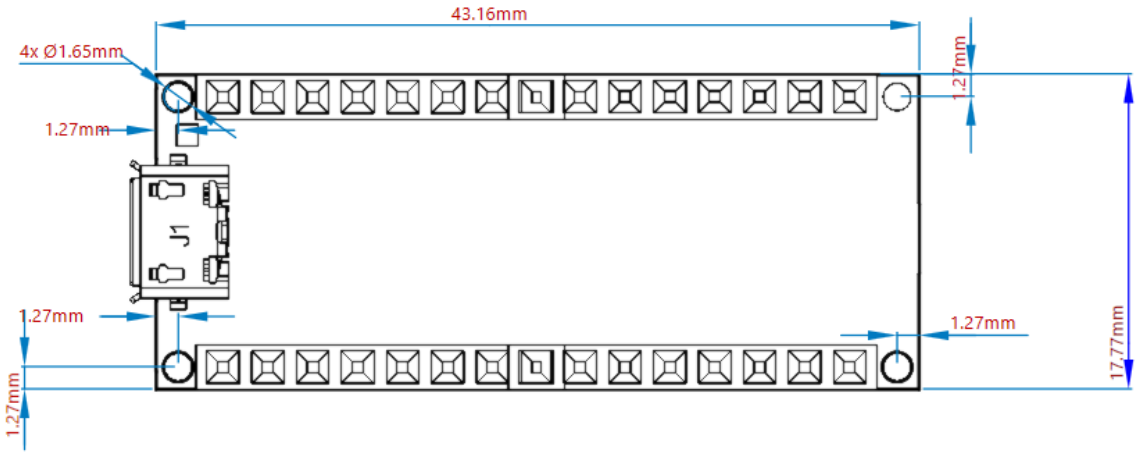


Layout

6.2 Connector Positions

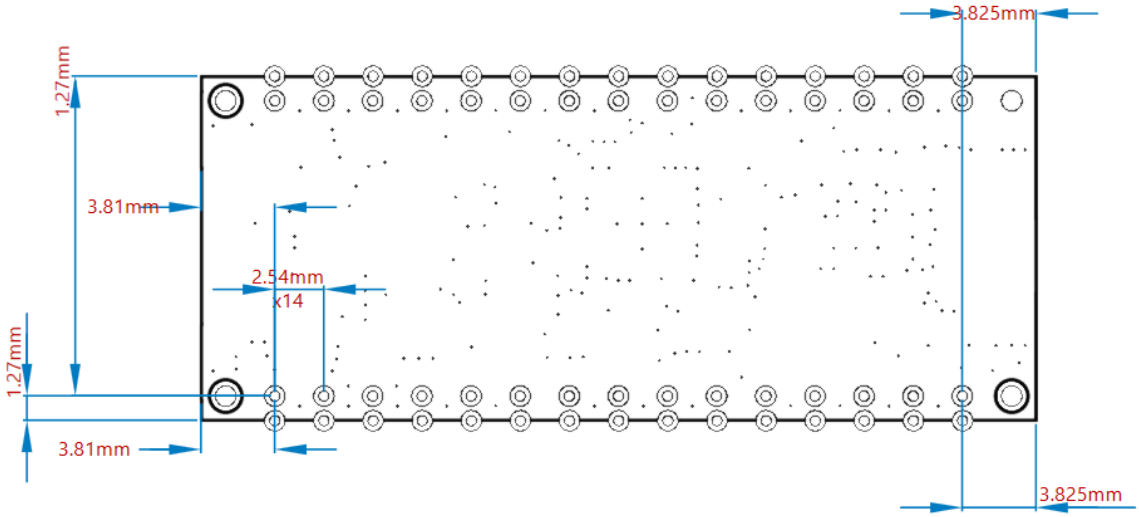
The view below is from top however it shows Debug connector pads which are on the bottom side. Highlighted pins are pin 1 for each connector'

Top view:



Top side connectors

Bottom view:



Bottom side connectors

7 Certifications

7.1 Declaration of Conformity CE DoC (EU)

We declare under our sole responsibility that the products above are in conformity with the essential requirements of the following EU Directives and therefore qualify for free movement within markets comprising the European Union (EU) and European Economic Area (EEA).

7.2 Declaration of Conformity to EU RoHS & REACH 211 01/19/2021

Arduino boards are in compliance with RoHS 2 Directive 2011/65/EU of the European Parliament and RoHS 3 Directive 2015/863/EU of the Council of 4 June 2015 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

Substance	Maximum limit (ppm)
Lead (Pb)	1000
Cadmium (Cd)	100
Mercury (Hg)	1000
Hexavalent Chromium (Cr6+)	1000
Poly Brominated Biphenyls (PBB)	1000
Poly Brominated Diphenyl ethers (PBDE)	1000
Bis(2-Ethylhexyl} phthalate (DEHP)	1000
Benzyl butyl phthalate (BBP)	1000
Dibutyl phthalate (DBP)	1000
Diisobutyl phthalate (DIBP)	1000

Exemptions : No exemptions are claimed.

Arduino Boards are fully compliant with the related requirements of European Union Regulation (EC) 1907 /2006 concerning the Registration, Evaluation, Authorization and Restriction of Chemicals (REACH). We declare none of the SVHCs (https://echa.europa.eu/web/guest/candidate-list-table), the Candidate List of Substances of Very High Concern for authorization currently released by ECHA, is present in all products (and also package) in quantities totaling in a concentration equal or above 0.1%. To the best of our knowledge, we also declare that our products do not contain any of the substances listed on the "Authorization List" (Annex XIV of the REACH regulations) and Substances of Very High Concern (SVHC) in any significant amounts as specified by the Annex XVII of Candidate list published by ECHA (European Chemical Agency) 1907 /2006/EC.

7.3 Conflict Minerals Declaration

As a global supplier of electronic and electrical components, Arduino is aware of our obligations with regards to laws and regulations regarding Conflict Minerals, specifically the Dodd-Frank Wall Street Reform and Consumer Protection Act, Section 1502. Arduino does not directly source or process conflict minerals such as Tin, Tantalum, Tungsten, or Gold. Conflict minerals are contained in our products in the form of solder, or as a component in metal alloys. As part of our reasonable due diligence Arduino has contacted component suppliers within our supply chain to verify their continued compliance with the regulations. Based on the information received thus far we declare that our products contain Conflict Minerals sourced from conflict-free areas.

8 FCC Caution

Any Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference

(2) this device must accept any interference received, including interference that may cause undesired operation.

FCC RF Radiation Exposure Statement:

1. This Transmitter must not be co-located or operating in conjunction with any other antenna or transmitter.
2. This equipment complies with RF radiation exposure limits set forth for an uncontrolled environment.
3. This equipment should be installed and operated with minimum distance 20cm between the radiator & your body.

English: User manuals for license-exempt radio apparatus shall contain the following or equivalent notice in a conspicuous location in the user manual or alternatively on the device or both. This device complies with Industry Canada license-exempt RSS standard(s). Operation is subject to the following two conditions:

(1) this device may not cause interference

(2) this device must accept any interference, including interference that may cause undesired operation of the device.

French: Le présent appareil est conforme aux CNR d'Industrie Canada applicables aux appareils radio exempts de licence. L'exploitation est autorisée aux deux conditions suivantes :

(1) l' appareil nedoit pas produire de brouillage

(2) l'utilisateur de l'appareil doit accepter tout brouillage radioélectrique subi, même si le brouillage est susceptible d'en compromettre le fonctionnement.

IC SAR Warning:

English This equipment should be installed and operated with minimum distance 20 cm between the radiator and your body.

French: Lors de l' installation et de l' exploitation de ce dispositif, la distance entre le radiateur et le corps est d 'au moins 20 cm.

Important: The operating temperature of the EUT can't exceed 85°C and shouldn't be lower than -40°C.

Hereby, Arduino S.r.l. declares that this product is in compliance with essential requirements and other relevant provisions of Directive 2014/53/EU. This product is allowed to be used in all EU member states.

Frequency bands	Maximum output power (ERP)
863-870Mhz	-3.22dBm

9 Company Information

Company name	Arduino SA.
Company Address	Via Ferruccio Pelli 14 6900 Lugano Switzerland

10 Reference Documentation

Reference	Link
Arduino IDE (Desktop)	https://www.arduino.cc/en/Main/Software
Arduino IDE (Cloud)	https://create.arduino.cc/editor
Cloud IDE Getting Started	https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-4b3e4a
Forum	http://forum.arduino.cc/
SAMD21G18	http://ww1.microchip.com/downloads/en/devicedoc/40001884a.pdf
NINA W102	https://www.u-blox.com/sites/default/files/NINA-W10_DataSheet_%28UBX-17065507%29.pdf
ECC608	http://ww1.microchip.com/downloads/en/DeviceDoc/40001977A.pdf
MPM3610	https://www.monolithicpower.com/pub/media/document/MPM3610_r1.01.pdf
NINA Firmware	https://github.com/arduino/nina-fw
ECC608 Library	https://github.com/arduino-libraries/ArduinoECCX08
LSM6DSL Library	https://github.com/stm32duino/LSM6DSL
ProjectHub	https://create.arduino.cc/projecthub?by=part&part_id=11332&sort=trending
Library Reference	https://www.arduino.cc/reference/en/
Arduino Store	https://store.arduino.cc/

11 Revision History

Date	Revision	Changes
04/15/2021	1	General datasheet updates

Vedlegg B

Kode

```
MODULE E458GripperTest
    ! Module for testing gripper, UiS KS Nov. 29, 2018

    ! wobj in the middle of the table, x-axis along short side of table, y-axis along long axis
  TASK PERS wobjdata wobjTableN:=[FALSE,TRUE,"",[[150,-500,8],[0.707106781,0,0,-0.707106781]],[[0,0,0],[1,0,0,0]]];

    ! robtargets
  CONST robtarget target_K0:=[[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
  !CONST robtarget target_K1:=[[0,-200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
    !CONST robtarget target_K2:=[[-200,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

    CONST num puckHeight := 30;
    CONST num safeHeight :=240;
    CONST num nPos := 7;

    VAR num LastPostNo := 0;
    VAR num nOnPos{nPos} := [5, 0, 0, 0, 0, 0, 0];
    VAR robtarget targets{nPos} := [
        [[0,-200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[200,-200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[200,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[-200,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[0,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[200,200,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]]];
    VAR num currentPos := 1;
    VAR BOOL run:=TRUE;
    VAR num svar;
    VAR num fra;
    VAR num til;


  PROC main()
      MoveL Offs(target_K0, 0, 0, safeHeight),v500,z10,tGripper\WObj:=wobjTableN;
      !movePuck 1, 4;
        !movePucK 1, 4;
        !movePucK 1, 4;
        !movePucK 1, 4;
        !movePucK 1, 4;
        !movePuck_back 3,1;
        !movePuck_back 3,1;
        !movePuck_back 3,1;
        !movePuck_back 3,1;
        !movePuck_back 3,1;
        !moveStack 3,1;
        !flipStack 1;

        WHILE run DO
            TPWrite "Test";
            TPReadFK svar, "VALG", "Move puck", "Move puck back", "Move stack","Flip stack","test";
            IF svar= 1 THEN
                TPReadnum fra, "Hvor vil du flytte fra?";
                TPReadnum til, "Hvor vil du flytte til?";

```

```
55                movePuck fra,til;
56            ELSEIF svar=2 THEN
57                TPReadnum fra, "Hvor vil du flytte fra?";
58                TPReadnum til, "Hvor vil du flytte til?";
59
60                movePuck_back fra,til;
61            ELSEIF svar = 3 THEN
62                TPReadnum fra, "Hvor vil du flytte fra?";
63                TPReadnum til, "Hvor vil du flytte til?";
64
65                moveStack fra,til;
66            ELSEIF svar=4 THEN
67                TPReadnum til, "Hvilken stack skal du flippe?";
68                flipStack til;
69            ELSE
70                TPWrite "test";
71            ENDIF
72        endwhile
73
74
75  ENDPROC
76
77
78    PROC moveStack(num fraPostNo, num tilPostNo)
79        VAR num newPos;
80        FOR i from 0 to nPos DO
81            IF i <> fraPostNo AND i <> tilPostNo THEN
82                newPos := i;
83            ENDIF
84        ENDFOR
85        FOR i FROM 0 TO nOnPos{fraPostNo}-1 DO
86            movePuck fraPostNo, newPos;
87        ENDFOR
88        FOR i FROM 0 TO  nOnPos{newPos}-1 DO
89            movePuck newPos, tilPostNo;
90        ENDFOR
91
92    ENDPROC
93
94    PROC flipStack(num fromPostNO)
95        VAR num k:=0;
96        FOR i FROM 1 to nOnPos{fromPostNO}+1 DO
97            k := k+ 1;
98            IF i <> fromPostNO THEN
99                movePuck fromPostNO, i;
100           ENDIF
101       ENDFOR
102       FOR i FROM 1 TO k DO
103           IF i <> fromPostNO THEN
104               movePuck i, fromPostNO;
105           ENDIF
106       ENDFOR
107
108
109   ENDPROC
110
111
112   PROC movePuck(num fra, num til)
113       VAR robtarget p1;
114       VAR robtarget p2;
```

```
        closeGripper(FALSE);
        p1 := Offs(targets{fra}, 0, 0, puckHeight*nOnPos{fra}-30);
        p2 := Offs(targets{til}, 0, 0, puckHeight*nOnPos{til});
        MoveJ Offs(p1, 30, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p1, 30, 0, 5),v50,z10,tGripper\WObj:=wobjTableN;
        MoveL Offs(p1, 10, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(p1, 0, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
        WaitTime 1.0;
        closeGripper(TRUE);
      MoveL Offs(p1, 0, 0, 30),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p1, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p2, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p2, 0, 0, 30),v500,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(p2, 0, 0, 5),v10,fine,tGripper\WObj:=wobjTableN;
        WaitTime 1.0;
        closeGripper(FALSE);
      MoveL Offs(p2, 0, 0, 60),v500,z10,tGripper\WObj:=wobjTableN;
        MoveL Offs(p2, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
        MoveL Offs(p2, 0, 0, safeHeight),v500,z10,tGripper\WObj:=wobjTableN;
        !MoveJ Offs(p2, 0, 0, safeHeight),v500,z50,tGripper\WObj:=wobjTableN;
        nOnPos{fra} := nOnPos{fra} - 1;
        nOnPos{til} := nOnPos{til} + 1;
    ENDPROC

    PROC movePuck_back(num fra, num til)
        VAR robtarget p1;
        VAR robtarget p2;
        closeGripper(FALSE);
        p1 := Offs(targets{fra}, 0, 0, puckHeight*nOnPos{fra}-30);
        p2 := Offs(targets{til}, 0, 0, puckHeight*nOnPos{til});
        MoveJ Offs(p1, 0, 0, 100),v500,z50,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p1, 0, 0, 5),v200,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(p1, 0, 0, 5),v10,fine,tGripper\WObj:=wobjTableN;
        WaitTime 1.0;
        closeGripper(TRUE);
      MoveL Offs(p1, 0, 0, 30),v100,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p1, 0, 0, 100),v200,z50,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p2, 0, 0, 100),v500,z50,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p2, 0, 0, 30),v200,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(p2, 0, 0, 5),v10,fine,tGripper\WObj:=wobjTableN;
        closeGripper(FALSE);
        WaitTime 1.0;
      MoveL Offs(p2, 0, 0, 30),v50,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p2, 0, 0, 100),v10,z50,tGripper\WObj:=wobjTableN;
        nOnPos{fra} := nOnPos{fra} - 1;
        nOnPos{til} := nOnPos{til} + 1;
    ENDPROC

    PROC getPuck(robtarget pos)
        MoveJ Offs(pos, 0, 0, 200),v500,z50,tGripper\WObj:=wobjTableN;
        MoveJ Offs(pos, 0, 0, 50),v200,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(pos, 0, 0, 10),v50,fine,tGripper\WObj:=wobjTableN;
        closeGripper(TRUE);
      MoveL Offs(pos, 0, 0, 50),v50,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(pos, 0, 0, 200),v200,z50,tGripper\WObj:=wobjTableN;
    ENDPROC

    PROC putPuck(robtarget pos)
        MoveJ Offs(pos, 0, 0, 200),v500,z50,tGripper\WObj:=wobjTableN;
        MoveJ Offs(pos, 0, 0, 50),v200,z10,tGripper\WObj:=wobjTableN;
```

```
175       MoveL Offs(pos, 0, 0, 10),v50,fine,tGripper\WObj:=wobjTableN;
176         closeGripper(FALSE);
177       MoveL Offs(pos, 0, 0, 50),v50,z10,tGripper\WObj:=wobjTableN;
178         MoveJ Offs(pos, 0, 0, 200),v200,z50,tGripper\WObj:=wobjTableN;
179     ENDPROC
180
181 ENDMODULE
```

```cpp
// Phototransistor setup
unsigned int Phototransistor = A0;
unsigned int Phototransistor1 = A1;
unsigned int lightVal;
unsigned int lightVal1;

// LED setup
unsigned int LED1 = 9;
unsigned int LED2 = 8;

// Case setup
const int INIT = 1;
const int MEASURE = 2;
const int Results = 3;
unsigned state = INIT;

//Values
unsigned int threshold = 400; //Treshold value for phototransistor, if <400 lights are off, if >400 lights are on.
const int timeWeld = 300;
unsigned long timeStart, timeElapsed;

// ARRAY SETUP
#define maxtargets 10
bool WeldOK[maxtargets];
bool allWeldsOK;
uint8_t n = 0;

//Button Setup and RED LED
int RedLED= 7;
int buttonPin= A2;
int currentStateButton;

//TIME SETUP
const int SHORT_PRESS_TIME = 1000; // 1000 milliseconds
const int LONG_PRESS_TIME  = 1000; // 1000 milliseconds
int lastState = LOW;  // the previous state from the input pin
unsigned long pressedTime  = 0;
unsigned long releasedTime = 0;

void setup() {
  Serial.begin(9600);
  timeStart = millis();
//Inputs
pinMode(Phototransistor, INPUT);
pinMode(Phototransistor1,INPUT);
pinMode (buttonPin, INPUT);
//OUTPUTS
pinMode(LED1, OUTPUT);
pinMode(LED2,OUTPUT);
pinMode (RedLED, OUTPUT);
//LED ON
digitalWrite(LED1, HIGH); //skal muligens inn i loopen
digitalWrite(LED2,HIGH);
//START STATE
state = INIT;// start with INIT on
n = 0;
}

```

```
59 void loop() {
60   switch (state) {
61     case INIT:    // Wait for signal from robot
62       while (state == INIT) {
63         //Serial.println("\nWaiting for signal from robot.");
64         currentStateButton = digitalRead(buttonPin);
65 
66         if(lastState == HIGH && currentStateButton == LOW)  // button is pressed
67           pressedTime = millis();
68         else if(lastState == LOW && currentStateButton == HIGH){  // button is
   released
69           releasedTime = millis();
70 
71         long pressDuration = releasedTime - pressedTime;
72 
73         if( pressDuration < SHORT_PRESS_TIME ){
74           state = MEASURE;
75           Serial.println("\nA short press is detected");
76         }
77         if( pressDuration > LONG_PRESS_TIME ){
78           Serial.println("\nA long press is detected");
79           state = Results;
80         }
81       }
82          // save the the last state
83       lastState = currentStateButton;
84       break;
85       }
86 
87 
88 
89 
90     case MEASURE:
91       while (state == MEASURE) {
92         Serial.println("\nLeser verdier");
93         lightVal = analogRead(Phototransistor);
94         lightVal1= analogRead(Phototransistor1);
95         digitalWrite(RedLED,LOW);
96         if (lightVal > threshold  && lightVal1 < threshold ){ // RIKTIG
97           timeElapsed = millis() - timeStart;
98           if (timeElapsed > timeWeld) {
99             Serial.println("\n ok test");
100             //restart the timer and add weld ok to array
101             timeStart = millis();
102             WeldOK[n] = true;
103             n++; //inkrementering av indeks i array'ene
104             state = INIT;
105             if (n >= maxtargets){
106               Serial.println("\n Maximum number of targets exeeded! Starting over
   …");
107               n = 0;
108               allWeldsOK = true;
109               state = INIT;
110             }
111           }
112         }
113         if (lightVal > threshold && lightVal1 > threshold) {  //Feil, begge lys er
   på
114           Serial.println("\n Ikke ok test");
115            //is it time for the next state?
```

```
116           timeElapsed = millis() - timeStart;
117           if (timeElapsed > timeWeld) {
118              //restart the timer and change the state
119             timeStart = millis();
120             allWeldsOK = false;
121             WeldOK[n] = allWeldsOK;
122             n++;
123             state = INIT;
124             if (n >= maxtargets){
125                 Serial.println("\n Maximum number of targets exeeded! Starting over
…");
126                 n = 0;
127                 allWeldsOK = true;
128                 state = INIT;
129
130             }
131
132           }
133         }
134         if (lightVal < threshold && lightVal1 < threshold) { //Feil, begge lys er av
135           Serial.println("\n Ikke ok test2");
136           //is it time for the next state?
137           timeElapsed = millis() - timeStart;
138           if (timeElapsed > timeWeld) {
139              //restart the timer and change the state
140                //restart the timer and change the state
141             timeStart = millis();
142             allWeldsOK = false;
143             WeldOK[n] = allWeldsOK;
144             n++;
145             state = INIT;
146             if (n >= maxtargets){
147               Serial.println("\n Maximum number of targets exeeded! Starting over
…");
148               n = 0;
149               allWeldsOK = true;
150               state = INIT;
151             }
152           }
153         }
154         if (lightVal < threshold && lightVal1 > threshold) { //Feil, lys oppe er av
155           Serial.println("\n Ikke ok test3");
156           //is it time for the next state?
157           timeElapsed = millis() - timeStart;
158           if (timeElapsed > timeWeld) {
159              //restart the timer and change the state
160                //restart the timer and change the state
161             timeStart = millis();
162             allWeldsOK = false;
163             WeldOK[n] = allWeldsOK;
164             n++;
165             state = INIT;
166             if (n >= maxtargets){
167               Serial.println("\n Maximum number of targets exeeded! Starting over
…");
168               n = 0;
169               allWeldsOK = true;
170               state = INIT;
171             }
172           }
```

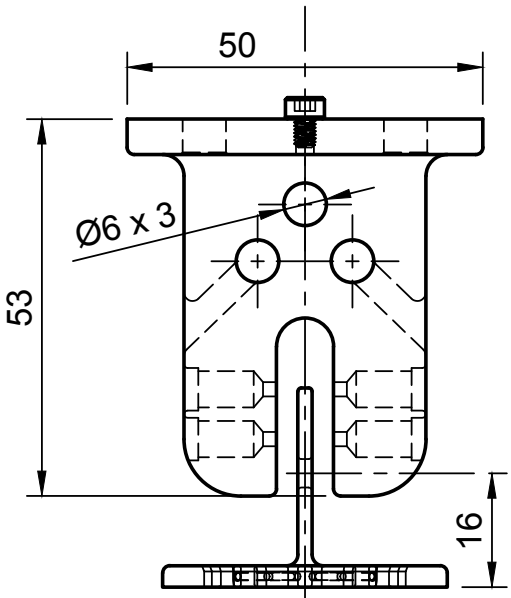
```
        }

      break;
      }


    case Results:
      while (state == Results) {
        for(int n = 0; n < maxtargets; n++){
          Serial.println(WeldOK[n]);
        }
        if (allWeldsOK == false){
         Serial.println("\n Ikke ok sveising");
         digitalWrite(RedLED,HIGH);
        }
        if (allWeldsOK == true){
         Serial.println("\n Ok sveising");
         digitalWrite(RedLED,LOW);
        }
          n = 0;
          allWeldsOK = true;
          state = INIT;
      break;
      }
  }
}


```

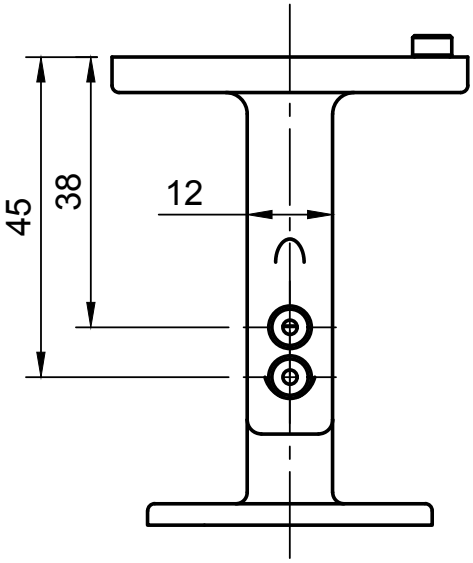
```
MODULE PunktSveising


  TASK PERS wobjdata wobjTableN:=[FALSE,TRUE,"",[[150,-500,8],
[0.707106781,0,0,-0.707106781]],[[0,0,0],[1,0,0,0]]];

  CONST robtarget target_K0:=[[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

    CONST num SveisHeight := 30;
    CONST num safeHeight := 200;
    CONST num nPos := 5;

    CONST num ShortPulse := 0.1;
    CONST num LongPulse := 1.5;

    VAR num LastPostNo := 0;
    VAR num nOnPos{nPos} := [1, 1, 1, 1, 1];
    VAR robtarget targets{nPos} := [
        [[0,-180,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[180,-180,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[0,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[180,0,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
        [[-180,180,0],[0,1,0,0],[0,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] ,
];
    VAR num currentPos := 1;
    VAR BOOL run:=TRUE;



  PROC main()
      MoveL Offs(target_K0, 0, 0, safeHeight),v500,z10,tGripper\WObj:=wobjTableN;

        WHILE run DO
            TPWrite "Test";
            TPReadFK svar, "Start";
            IF svar= 1 THEN
                TPReadnum til " Hva er start posisjonen";
                moveBane til;
            ENDIF
        endwhile


  ENDPROC


    PROC moveBane(num fraPostNo)
        FOR i from 0 to nPos DO
            IF i <> fraPostNo THEN
                LastPostNo := fraPostNo + 1;
            ENDIF
        ENDFOR
        FOR i FROM 0 TO nOnPos{LastPostNo} DO
            moveSveis LastPostNo, i;
        ENDFOR
        PulseDO(LongPulse)
        result := Dinput(ABB_Scaleable_IO_DO1)
    ENDPROC


```

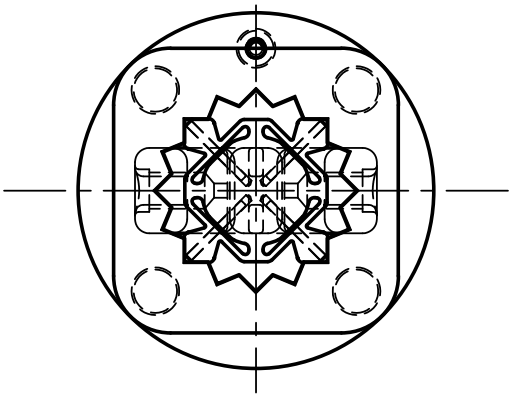
```
    PROC moveSveis(num fra, num til)
        VAR robtarget p1;
        VAR robtarget p2;
        PulseDO(FALSE);
        p1 := Offs(targets{fra}, 0, 0, SveisHeight*nOnPos{fra}-30);
        p2 := Offs(targets{til}, 0, 0, SveisHeight*nOnPos{til});
        MoveJ Offs(p1, 30, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p1, 30, 0, 5),v50,z10,tGripper\WObj:=wobjTableN;
        PulseDO(ShortPulse);
        MoveL Offs(p1, 10, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(p1, 0, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
        WaitTime 0.2;
      MoveL Offs(p1, 0, 0, 30),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p1, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;

        MoveJ Offs(p2, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
        MoveJ Offs(p2, 0, 0, 30),v500,z10,tGripper\WObj:=wobjTableN;
        PulseDO(ShortPulse);
        MoveL Offs(p2, 10, 0, 5),v10,z10,tGripper\WObj:=wobjTableN;
      MoveL Offs(p2, 0, 0, 5),v10,fine,tGripper\WObj:=wobjTableN;
        WaitTime 0.2;
      MoveL Offs(p2, 0, 0, 60),v500,z10,tGripper\WObj:=wobjTableN;
        MoveL Offs(p2, 0, 0, 100),v500,z10,tGripper\WObj:=wobjTableN;
    ENDPROC

     PROC PulseDO(num duration)
        SetDo ABB_Scaleable_IO_DO1, 1;
        WaitTime duration;
        SetDo ABB_Scaleable_IO_DO1, 0;
    ENDPROC
ENDMODULE
```

Vedlegg C

Projeksjonstegninger







Dept.	Technical reference	Created by	Approved by
		Ståle Freyer 11/05/2022	
		Document type	Document status
		Title: Welding simulator	DWG No.
			Rev. / Date of issue / Sheet: 1/1