Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2022 Åpen
Forfatter(e): Vilhelm Assersen og Elias Nodland	
Fagansvarlig: Karl Skretting Veileder(e): Karl Skretting	
Tittel på bacheloroppgaven: Forbedre kamerarigg for øving IA4 i ELE610 Engelsk tittel: Improving camera-assembly for assignment IA4 in ELE610	
Studiepoeng: 20	
Emneord: Bildebehandlig, bildefangst, mikrokontrolle, PWM 2	Sidetall: 41 + vedlegg/annet: 5 (19) Stavanger 15. mai 2022

Det Teknisk-Naturvitenskapelige Fakultet
Institutt for Data- og Elektroteknologi

Forbedre kamerarigg for øving IA4 i ELE610

Bacheloroppgave i Automatisering og Elektronikkdesign av

Vilhelm Assersen[1]
Elias Nodland[1]

Veileder
Karl Skretting[1]

Affiliasjoner
[1]Universitetet i Stavanger

15. mai 2022

SAMMENDRAG

I øving IA4 i emnet ELE610 ved Universitetet i Stavanger bruker studentene en kamerarigg for å eksperimentere med bildefangst og behandling.

Studentene utvinner nyttig informasjon (rotasjonshastighet) ved å finne visse indikatorer på en roterende skive og regne avvik eller forflytning over tid. Dette gjøres programmatisk ved hjelp av flere teknikker introdusert i tidligere øvinger.

Selv om den nåværende riggen er funksjonell og generelt problemfri, har den potensiale for forbedring. For eksempel er det ingen god måte å bekrefte at hastigheten som er målt er den faktiske hastigheten. På grunn av dette må studenter ty til øyemål eller andre metoder for å lese av reell hastighet.

Det er også vanskelig for studenter å komme i gang med øvingen på grunn av manglende dokumentasjon for hvordan kameraet brukes i koden.

Vi presenterer konstruksjon av en ny rigg, basert på den eksisterende riggen, men med noen endringer og forbedringspunkter, og vi dokumenterer viktige detaljer.

Vi presenterer også en mal som studenter kan bruke som forenkler programmeringen betraktelig slik at studenter kan fokusere på interessante utfordringer fremfor å sitte fast med manuell minnehåndtering.

ANERKJENNELSER

Vi vil uttrykke takknemlighet for de mennesker som har hjulpet oss i denne oppgaven.

Først og fremst vil vi takke *Karl Skretting* som har vært en fantastisk veileder gjennom hele prosjektet.

Vi vil også takke *Sjefingeniør Stål Freyer* for verdifulle råd, og bidrag i form av 3D-printede komponenter, og prosjektkode for den gamle riggen.

Innhold

Tabeller

Figurer

Kapittel 1

Introduksjon

I lab-øving IA4 i emnet ELE610 bruker studentene en kamerarigg og Python-programmering [1] for å eksperimentere med bildefangst og prosessering. Spesifikt handler øvingen om å finne rotasjonshastigheten til en skive ved å ta bilder og analysere forskjellige visuelle indikatorer som er på skiven. Bildene kan være tatt ved en kjent posisjon vhja. en trigger (en automatisk bryter), eller i sekvens hvor man bruker tidsintervallet mellom bildene for å finne vinkelhastighet. Bildene som blir tatt kan prosesseres ved bruk av Python-biblioteker som OpenCV [2] som er egnet for bildeprosessering.

Riggen er tilstrekkelig for arbeidet, men har noe potensiale for forbedring; for eksempel vet vi ikke den faktiske hastighet på skiven.

Da vi som studenter utførte denne øvingen, valgte vi å bruke et høyhastighetskamera for å telle rotasjoner og finne den ekte hastigheten; en mindre enn optimal løsning.

I denne rapporten presenterer vi vår løsning på oppgaven som gitt, samt hvilke elementer vi har valgt å fokusere på.

Når vi refererer til *oppgaven* mener vi oppgaven som gitt i vedlegg A. Når vi refererer til *øvingen* mener vi lab-øvingen IA4 som utføres i emnet.

1.1 Problemstilling

Den originale problemstillingen er flerdelt og er gitt i god detalj i vedlegg A. Målet med oppgaven er å bygge en ekstra rigg som kan brukes i lab-øvingen slik at flere grupper kan jobbe samtidig. Til dette formålet er det anskaffet en lik motor, mikrokontroller, strømforsyning, lysgaffel, og PWM-forsterker/motordriver for styring av motoren. Det er også satt av et tilsvarende, men litt forskjellig kamera. Motoren og lysgaffelen er allerede

montert på en plate med et 3D-printet hus. Vi har valgt å fokusere først og fremst på de delene som er nødvendige for en fungerende rigg.

De grunnleggende elementene som bør være på plass i en slik rigg er:

- motor,
- styring av motorhastighet, og
- kamera med trigger v.hj.a. lysgaffel.

Alle komponenter som er absolutt nødvendige er altså gitt. Allikevel er det praktisk å kunne stille på motorhastighet v.hj.a. et potensiometer, det kan være digitalt slik som på den gamle riggen, eller et enkelt analogt potensiometer.

Videre er det listet opp flere ting som kan være nyttige å ha; for eksempel en RPM-teller (Rotasjoner per minutt) og en liten skjerm for å vise faktisk hastighet på motoren slik at målt hastighet fra bildene kan sammenlignes med faktisk hastighet på en enkel måte.

Oppgaveteksten foreslår også å montere lysstripe for jevnere belysning, med formål om å forbedre bildefangst. Av vår erfaring med øvingen har vi sett at dette strengt tatt ikke er nødvendig.

Det er bedre løsning å stille blenderåpningen så stor som mulig for å slippe inn mest mulig lys istedet. Det ble også nevnt av sjefingeniør Ståle Freyer at det er god øvelse å arbeide under ulike lysforhold, siden det er en bedre representasjon av å arbeide ute i virkeligheten.

Det er heller ikke nødvendig å kunne montere motiver med bakgrunn for terninger da denne riggen ikke er i bruk i lab-øvingen hvor terninger er involvert. Vi antar at disse elementene kan innlemmes ved en senere anledning uten store endringer.

Foruten punktene nevnt i oppgaveteksten har vi også sett at det er nødvendig med en forenkling av programmeringsbiten: For å bruke kameraene benyttes det et Python-bibliotek, PyuEye [pipy-pyueye], som gir tilgang til avansert funksjonalitet som å sette bildehastighet, eksponeringstid, fokus, og mer. Dette biblioteket er beskrevet som:

> "... a lean wrapper implementation of Python function objects that represent uEye API functions. As there is no intelligence in the PyuEye interface, it is as near as possible to the uEye C API and always up-to-date."
>
> – beskrivelse av PyuEye-pakken [3].

Koden bærer et sterkt preg av C og benytter elementer som pekere, manuell håndtering av minne, allokering av bildebuffere, konfigurering via flagg heller enn funksjoner, osv.; konsepter som er lite forenelige med Python

som programmeringsspråk. Disse utfordringene, kombinert med lite eller ingen oppdatert dokumentasjon, gjorde at vi fikk et inntrykk om at den vanskeligste – og mest tidkrevende – delen av øvingen ble å få kameraet til å fungere med riktige innstillinger, heller enn å utforske bildebehandling for å utvinne nyttig informasjon fra en bildesekvens.

Vi har derfor satt som et ekstra mål å utvikle et lite abstraksjonslag som bygger oppå PyuEye igjen; med formål om å forenkle koden for oppsett av kamera, og derfor frigjøre mer tid og fokus til den "interessante" delen av øvingen. Sammen med dette ønsker vi også å gi flere brukbare eksempler for hvordan koden kan brukes for å konfigurere forskjellige parametre på enkelt vis.

1.2 Om emnet ELE610 - Praktisk robotteknikk

> Etter å ha tatt dette emnet skal studentene: 1) Kunne klargjøre (kalibrere) og bruke et enkelt kamera og et "industrikamera kamera for å ta bilder og overføre disse til ønsket plattform. 2) De skal ha grunnleggende forståelse for hvordan en ABB industrirobot brukes og kunne lage enkle programmer for den. 3) De skal ha grunnleggende forståelse for Python programmering for bildefangst, for bildebehandling og for kommunikasjon med robot."
>
> – Informasjonsside for ELE610 [4, læringsutbytte].

Dette emnet består av to hovedgrupper av lab-øvinger: (1) IA1-IA4: bildefangst med bildebehandling i Python, og (2) RS1-RS4: RobotStudio for styring av ABB-roboter. En større øving – RS5 – som forener de to hovedkategoriene kommer til slutt, hvor studentene skal bruke maskinsyn sammen med robotikk for å gjenkjenne, lokalisere, og manipulere objekter.

Utførelsen av øvingene blir gjort av grupper som består av to til tre studenter. Alt etter hvor mange grupper som blir dannet starter alle eller halvparten av gruppene samtidig på hver øving. Generelt sett er ikke dette er problem, men øving IA4 har kun én rigg, noe som kan føre til mangel på tid med riggen. Dette har generelt gått fint, men det hadde vært fint med én til operativ rigg.

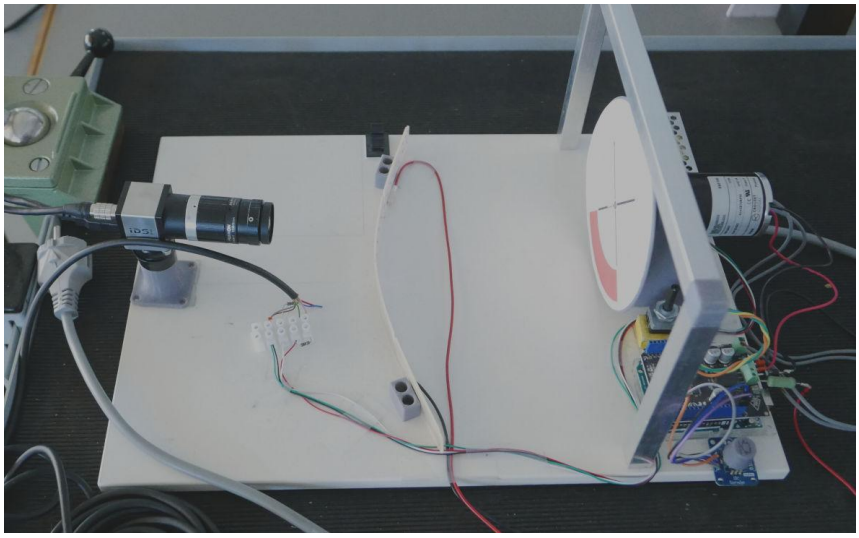
1.3 Tidligere arbeid

Dette underkapittelet handler om arbeidet som er gjort tidligere for denne lab-øvingen, og er hovedsaklig en beskrivelse av den originale riggen.

1.3.1 Nåværende rigg

Den eksisterende riggen er tilstrekkelig for utførelse av lab-øvingen. Riggen er forholdsvis enkel og består av

- strømforsyning,
- mikrokontroller,
- motor,
- led-stripe,
- motordriver for styring av motor og lys,
- kamera,
- skive med motiv,
- enkoder for justering av hastighet/lysstyrke, og
- lysgaffel for kameratrigger.



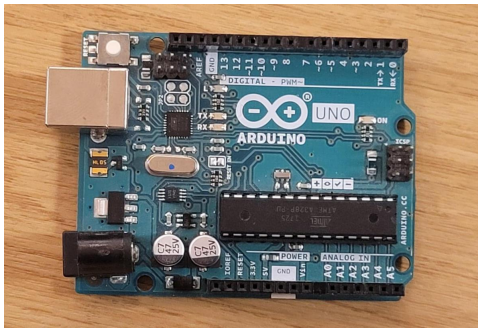
Figur 1.1: Den gamle riggen i sin helhet

Nedenfor beskrives hvordan de forskjellige komponentene fungerer og er satt sammen i den gamle riggen. Mer teknisk informasjon om komponentene finnes i kapittel 2.

Mikrokontroller: Arduino Uno

Arduino Uno er en mikrokontrollerplattform basert på ATmega328P, en 8-bits mikrokontroller med AVR-arkitektur[1] og er svært populær for små prosjekter hvor høy prosesseringskraft ikke er et krav [6]. AVR er en såkalt *Reduced Instruction Set Computer*-arkitektur (RISC).

Plattformen har flere inn- og utganger, hvorav noen kan benyttes som analoge innganger [7], andre for spesielle protokoller som I^2C(I2C). I riggen brukes Arduino sammen med en "hatt", strømforsyning, og enkoder for å styre motor og lys.



Figur 1.2: Arduino Uno

Motor: Crouzet 8980001

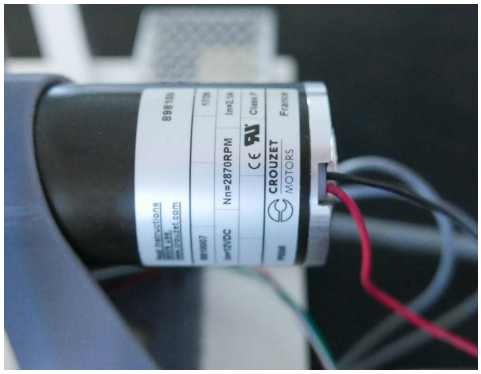
Motoren er en Crouzet 12V, børstet DC-motor [8, side 10]. Et bilde er inkludert i figur 1.3.

Motoren har flere spesifikasjoner for belastningsevne, dreiemoment, etc., men de er ikke relevante her ettersom riggen opererer totalt uten belastning.

Motordriver: 4A Motor Shield

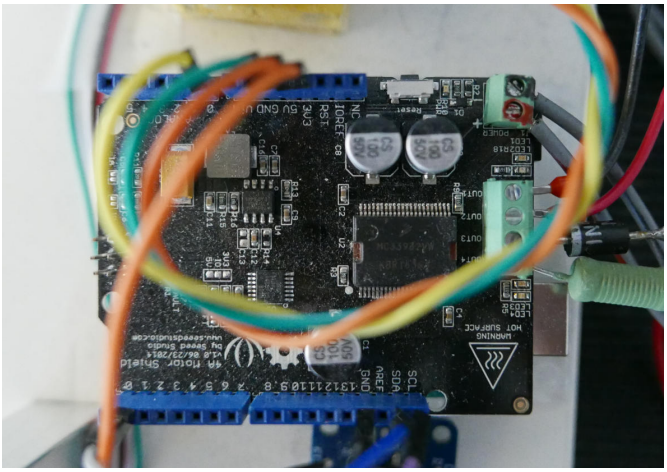
Digitale utganger på Arduino Uno er i stand til å gi 20 mA per utgang, og er begrenset til generelt lave verdier for total strømtrekk [9]. Dette er nok for å drive enkle sensorer og I/O, og for kommunikasjon med andre moduler, men ikke nok for å drive større komponenter som motorer. Ofte – gitt tilstrekkelig strømforsyning – er det mulig å jobbe rundt denne begrensningen ved å la utgangen styre en krafttransistor eller et relé som igjen styrer motoren.

[1] Ifølge Atmel (produsenten av AVR-prosessorer) står ikke AVR for noe spesielt. Det er allikevel allmenn akseptert at AVR kan stå for *Alf og Vegards RISC-prosessor* [5].



Figur 1.3: Motoren som brukes i riggen

I dette tilfellet kan motoren trekke opp til 1.67 A ved 12 V, noe som langt overstiger hva mikrokontrolleren kan levere, som gjør at det er nødvendig med mer spesialisert maskinvare. Etter vår erfaring er det mulig å få motoren til å spinne med bare 5V fra en USB-port fra datamaskinen, men dette virker lite optimalt ettersom de fleste porter av denne typen er begrenset til 500mA ved 5V [10]. Derfor inkluderer riggen en "hatt" – eller et "skjold" – som sitter oppå Arduino og utvider mulighetene for styring av større motorer [11].



Figur 1.4: Motor-hatt, her montert oppå Arduino

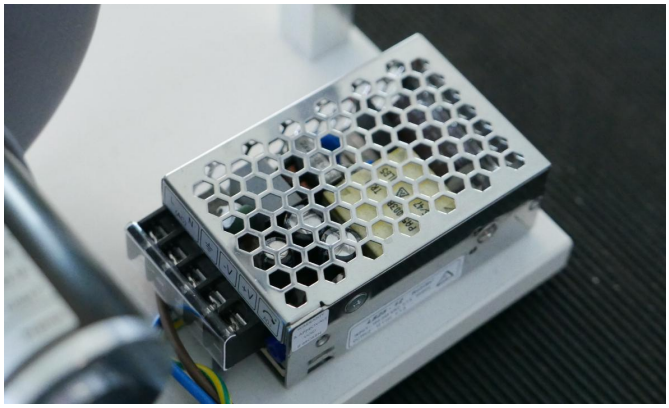
Skjoldet er en motordriver basert på MC33932, en integrert kraftkrets med to h-broer, og kan med det forsyne to motorer med opp til 5 ampere

hver [12].

I riggen brukes én av utgangene til å drive motoren, og den andre utgangen brukes for å forsyne en lysstripe.

Strømforsyning: LS25 12

Strømforsyningen er en generisk 12 V strømforsyning fra TDK-Lambda, vist i figur 1.5.



Figur 1.5: Strømforsyningen

Denne er koblet til hatten og gir strøm til hele riggen.

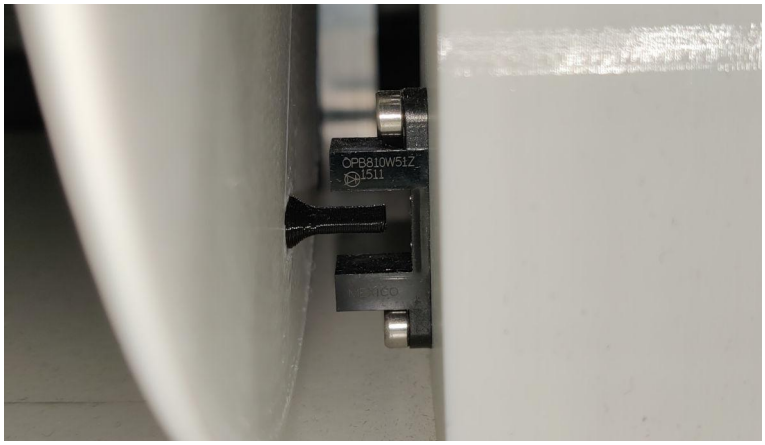
Lysgaffel: OPB810W51Z

Lysgaffelen er av typen OPB810W51Z fra OPTEK Technologies og er en såkalt *optisk bryter med spor* [13]. Den består hovedsaklig av tre deler:

- lampe,
- fototransistor, og
- innkapsling for å holde de to førstnevnte komponentene.

Lysgaffelen til riggen brukes som en bryter for kameraet. Skiven (til venstre i figur 1.6) roterer og tappen passerer gjennom åpningen på gaffelen én gang hver runde. Når dette skjer kan kameraet bruke signalet som kommando for å ta et bilde, kalt en *trigger*.

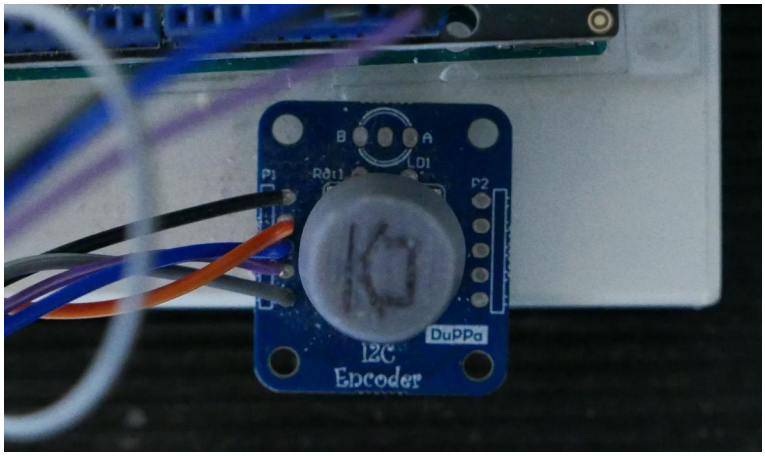
Det er mulig å legge inn en forsinkelse mellom når signalet registreres og når bildet blir tatt, og ettersom disken vil være i (omtrent) samme posisjon hver gang triggeren aktiveres, er det mulig å beregne vinkelhastighet på disken ved å beregne hvor mye den har rotert siden den var i den kjente



Figur 1.6: Lysgaffelen, her vist med "tappen" som bryter lyset når den passerer gjennom åpningen i gaffelen.

posisjonen. Dette er en av de anbefalte måtene å gjøre selve øvingen på.

Enkoder: DuPPa I2C Encoder

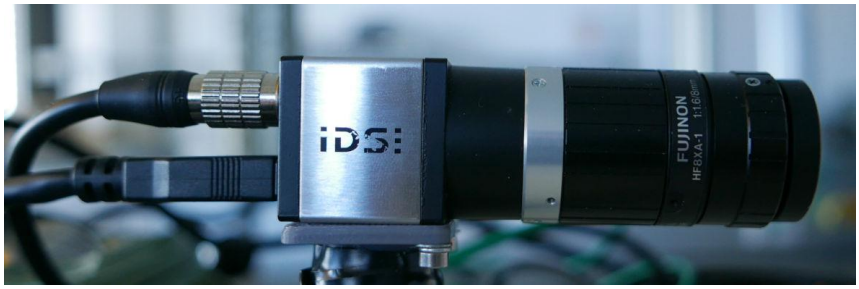


Figur 1.7: I2C Enkoder DuPPa

Dette er en enkoder plassert på et printet kretskort (PCB, eng.: Printed Circuit Board) sammen med en mikroprosessor som registrerer rotasjon og organiserer kommunikasjon over I2C.

Enkoderen brukes i riggen for å stille på PWM-verdien som styrer motoren og lys-stripen. Den har en integrert trykknapp som kan trykkes for å bytte mellom å styre lyset eller motoren.

Kamera: IDS UI-3360



Figur 1.8: Kameraet som brukes i den gamle riggen

Kamerahuset er et UI-3360 fra IDS og har en bildeoppløsning på 2048 ganger 1088 piksler og en bildehastighet på opptil 153 bilder per sekund [14].

Linsen på kameraet er en Fujinon HF8XA-1 fra Fujifilm. Linsen er låst til 8 mm brennvidde ("zoom"), har justerbart fokus, og blenderåpning som varierer mellom $f/1.6$ og $f/16$.

Av vår erfaring er det best å la blenderåpningen stå så stor som mulig ($f/1.6$). Dette gjør at linsen slipper inn mest mulig lys og eksponeringstiden kan stilles så lavt som mulig. Med dagslys fra vinduet har vi erfart at det ikke er nødvendig å skru på det kunstige lyset.

1.3.2 PyuEye

Kameraene fra IDS kan brukes som et generisk kamera v.hj.a. vanlige grensesnitt levert av operativsystemet. I dette tilfellet vil kameraet ha noen standard parametre som er satt og som gjelder.

Av vår erfaring med øvingen er ikke dette nok for å få tilstrekkelig kontroll over forskjellige parametre som eksponeringstid, trigger, og mer.

Derfor brukes PyuEye, et python-bibliotek som pakker et C-API (applikasjons-programmerings-grensesnitt, eng.: Application Programming Interface), og gir kontroll over disse parametrene [3]. Biblioteket minner mye om C, og krever forsiktig håndtering av pekere og manuelt allokert minne. Til gjengjeld gjør dette at dokumentasjonen for C-API'et gjelder i stor grad for PyuEye også.

1.3.3 Vår løsning av lab-øvingen fra høsten 2021

Vi valgte å bruke en videostrøm heller enn trigger-funksjonaliteten. Ved å ta flere bilder etter hverandre kunne vi bruke bildeprosessering for å finne kjente markører på skiven, og med dette finne ut hvor mye skiven hadde rotert siden forrige bilde ble tatt. Sammen med tidsintervallet mellom bildene kunne vi finne skivens rotasjonshastighet.

1.4 Oppbygging av rapport

I neste kapittel, kapittel 2, beskriver vi hvordan komponentene fungerer.

I kapittel 3 beskriver vi konstruksjon og programmering av selve riggen. Vi har valgt å strukturere denne delen av rapporten på en slik måte for å unngå overflødige kryssreferanser mellom et arkitektur-kapittel og et konstruksjons-kapittel.

I kapittel 4 dekker vi testing av den ferdige riggen og diskuterer noen av resultatene. Vi diskuterer også noen av de mindre polerte aspektene ved riggen, og vi foreslår eventuelle fremtidige forbedringer.

Kapittel 5 er dedikert for avsluttende ord, og vi konkluderer hva som er gjort og hva som kunne vært bedre.

Kapittel 2

Komponenter og teori

Her dekker vi nødvendig teori som vi bruker i konstruksjon av riggen.

2.1 Gjenbrukte komponenter

Vi går gjennom virkemåten til komponentene som allerede er på plass i den gamle riggen.

Arduino

Arduino kan programmeres på enkelt vis ved hjelp av Arduino's egen tekstbehandler, Arduino IDE [15], eller ved å bruke kommandolinjeverktøyet `arduino-cli` [16]. Språket som brukes er C++, men i en begrenset form sammenlignet med hva man vil finne på et brukersystem. Dette er forårsaket delvis av at Arduino opererer på et mye lavere nivå, samt at de fleste Arduino-enheter har forholdsvis svake prosessorer.

Et Arduino-program er hovedsaklig konstruert av to funksjoner: `setup()` og `loop()`. Koden i `setup()` kjøres én gang ved oppstart før `loop()` blir gjentatt uendelig (eller frem til Arduino møter en kritisk feil).

Arduino har noen titalls pinner/pins som kan styres v.hj.a. kode. For eksempel kan man sette opp en pin til å brukes som en digital utgang ved å skrive `pinMode(N, OUTPUT)` hvor `N` er navnet til pinnen som ønskes brukt. Når dette er gjort kan `digitalWrite(N, HIGH)` eller `analogWrite(N, 127)` brukes for å skrive digitale eller analoge verdier til utgangen. `analogWrite` er et noe misvisende navn ettersom det ikke er noe analogt ved det. Funksjonen setter en verdi i et register for å styre PWM på utgangen.

Et enkelt Arduino-program fra er vist i oppføring 2.1.

```
1  void setup() {
2    pinMode(LED_BUILTIN, OUTPUT);
3  }
4
5  void loop() {
6    digitalWrite(LED_BUILTIN, HIGH);
7    delay(1000);
8    digitalWrite(LED_BUILTIN, LOW);
9    delay(1000);
10 }
```

Oppføring 2.1: Enkelt Arduino-program for å få en diode til å blinke

Programmet her setter aller først opp den interne pinnen `LED_BUILTIN` som en utgang; deretter begynner den å blinke ett sekund på, ett sekund av, ett sekund på... Med denne enkle funksjonaliteten er det mulig å gjøre det aller meste med Arduino, men det finnes flere biblioteker som forenkler bruken svært mye i noen tilfeller.

Foruten enkel, lineær kode, støtter Arduino også avbrudd. Et avbrudd er et signal med spesiell prioritet som gjør at Arduino avbryter det den holder på med for å håndtere avbruddet. Arduino Uno støtter avbrudd på pinne 2 og 3. Et eksempel på oppsett av avbrudd er vist i oppføring 2.2.

```
1  void setup() {
2      pinMode(2, INPUT);
3      attachInterrupt(digitalPinToInterrupt(2),
4                      interrupt,
5                      RISING);
6  }
7
8  static int teller;
9  void interrupt() {
10     teller++;
11 }
```

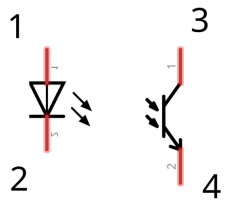
Oppføring 2.2: Enkelt avbrudd

Her setter vi opp et avbrudd på pin 2 som aktiveres når signalet går fra høyt til lavt (`RISING`) og øker en teller. Avbrudd er nyttige for tidskritiske hendelser. For eksempel når noe skjer samtidig som avbruddet og data må leses av med en gang fordi det ikke kan vente på at hovedløkken skal bli ferdig. Funksjoner som håndterer avbrudd bør generelt holdes så korte som

mulig.

Lysgaffel

Vi har inkludert et logisk skjema av lysgaffelen i figur 2.1.



Figur 2.1: Logisk sammensetning av lysgaffelen

Lysgaffelen fungerer slik at strøm kan passere gjennom fototransistoren når lys fra dioden skinner på den.

I den gamle riggen er lysgaffelen koblet opp slik at dioden får strøm fra kameraet inn på pins 1 og 2. Pin 1 er koblet til kameraets VCC OUT gjennom en 150 Ω motstand og pin 2 er koblet til kameraets GND. En pull-up motstand er koblet mellom positiv skinne og pin 3, og pin 3 er deretter koblet til kameraets Trigger IN. Pin 4 er koblet til GND.

Når noe blokkerer åpningen vil fototransistoren stenges og pull-up-motstanden vil trekke spenningen opp til skinnespenning. Dette gjør at kameraet mottar et høyt signal på Trigger IN. Ellers vil fototransistoren være åpen og spenningen trekkes ned til jord.

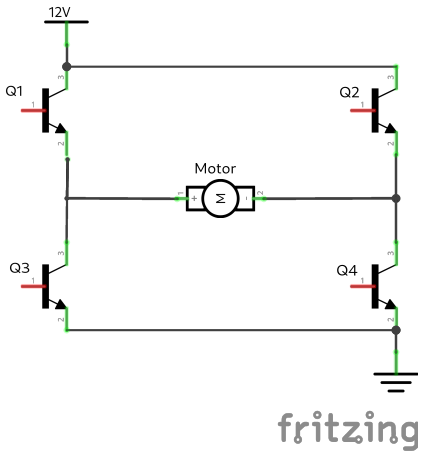
Motordriver

Motordriveren/hatten bruker som sagt to h-bruer, én for hver utgang (A og B) på kortet. En h-bro er en populær måte å kontrollere DC-motorer og består logisk sett av fire transistorer. En enkel modell er vist i figur 2.2

Ved å skru på transistorene Q1-Q4 i forskjellige konfigurasjoner, kan vi styre retningen som strømmen tar gjennom motoren. Vi kan med dette variere retning på selve motoren.

Vi kan konstruere en tabell av alle *lovlige* kombinasjoner av signalene Q1-Q4 som i tabell 2.1. Vi baserer tabellen på at flyt fra positiv til negativ pol på motoren fører til rotasjon med klokka.

Kombinasjoner hvor bare én av transistorene er aktivert vil ha samme



Figur 2.2: En enkel h-bru

Q1	Q2	Q3	Q4	Konsekvens
0	0	0	0	Fri rotasjon
1	0	0	1	Med klokka
0	1	1	0	Mot klokka
1	1	0	0	Brems
0	0	1	1	Brems

Tabell 2.1: Tabell over kombinasjoner som har effekt på motorens rotasjon.

effekt som at ingen er aktivert. Kombinasjoner hvor $Q_1 \times Q_3 + Q_2 \times Q_4 = 1$ (boolsk algebra) vil føre til kortslutning.

Hatten har to av disse bruene og hver bru styres ved hjelp av tre pins. Den første bruen styres av pins 5, 6, og 9 på Arduino og er konfigurert slik at pin 5 styrer hvilken skinne OUT 1 blir koblet til og pin 6 styrer hvilken skinne OUT 2 blir koblet til. Pin 9 blir kombinert med signalet fra pins 5 og 6 for å bestemme om transistorene er på eller ikke.

Formlene for om inngangene er på eller ikke, er gitt:

$$Q_1 = \overline{P_5} \times P_9$$
$$Q_2 = \overline{P_6} \times P_9$$
$$Q_3 = P_5 \times P_9$$

$$Q_4 = P_6 \times P_9$$

Som gir oss følgende tabell 2.2. Her representerer X hvilken som helst verdi. Hvis $P_9 = 0$ vil ingen transistorer være på.

P5	P6	P9	Konsekvens
X	X	0	Fri rotasjon
0	0	1	Brems
0	1	1	Med klokka
1	0	1	Mot klokka
1	1	1	Brems

Tabell 2.2: Kombinasjoner av verdier for pins 5, 6, og 9 og deres effekt på motorens rotasjon.

Pins 5 og 6 brukes for å styre motorens retning (eller om den bremser) mens pin 9 brukes for PWM-kontroll av motoren. Den samme oppførselen gjelder h-bru nummer to, men bruker henholdsvis pins 7, 8, og 10 fremfor pins 5, 6, og 9. Utgangene er kalt OUT 3 og OUT 4 fremfor OUT 1 og OUT 2.

Hatten har en del funksjonalitet som kan aktiveres v.hj.a. små "jumpers" (en komponent som kortslutter to kontaktpunkter). Vi benytter ingen slik funksjonalitet.

H-bru-chippen opererer mellom 8 og 28V og kan pulseres opp til 11kHz [12].

Signalkabel

Riggen benytter en kabel for å bruke signalet fra lysgaffelen som en "trigger" for kameraet [17]. Kabelen har åtte ledere og bruker en kontakt av typen HR25-7TP-8S fra Hirose.

Lederene er som følger:

- GND
- Flash OUT -
- GPIO 1
- Trigger GND
- Flash OUT +
- GPIO 2
- Trigger IN
- VCC

I den gamle riggen brukes kun fire av disse lederene: GND, VCC, Trigger GND, og Trigger IN.

2.2 Nye komponenter

I denne seksjonen beskriver vi de komponentene som avviker, eller er lagt til sammenlignet med det gamle designet, samt noen rettferdiggjørelser for hvorfor disse komponentene er valgt.

2.2.1 LCD: HiLetgo HD44780

Figur 2.3: Generisk 16x2 LCD som er kompatibelt med Arduino's LiquidCrystal-bibliotek, her montert sammen med enkoder.

En generisk *flytende-krystall-skjerm* (LCD, eng.: Liquid Crystal Display), basert på en klon av den nokså gamle LCD-kontrolleren HD44780 fra Hitachi [18]. LCD basert på denne kontrolleren kan brukes på enkelt vis ved hjelp av Arduinos `LiquidCrystal`-bibliotek [19].

Kontrolleren opererer enten i 8-bit, eller 4-bit modus. Skjermen har to rader med seksten ruter på hver rad.

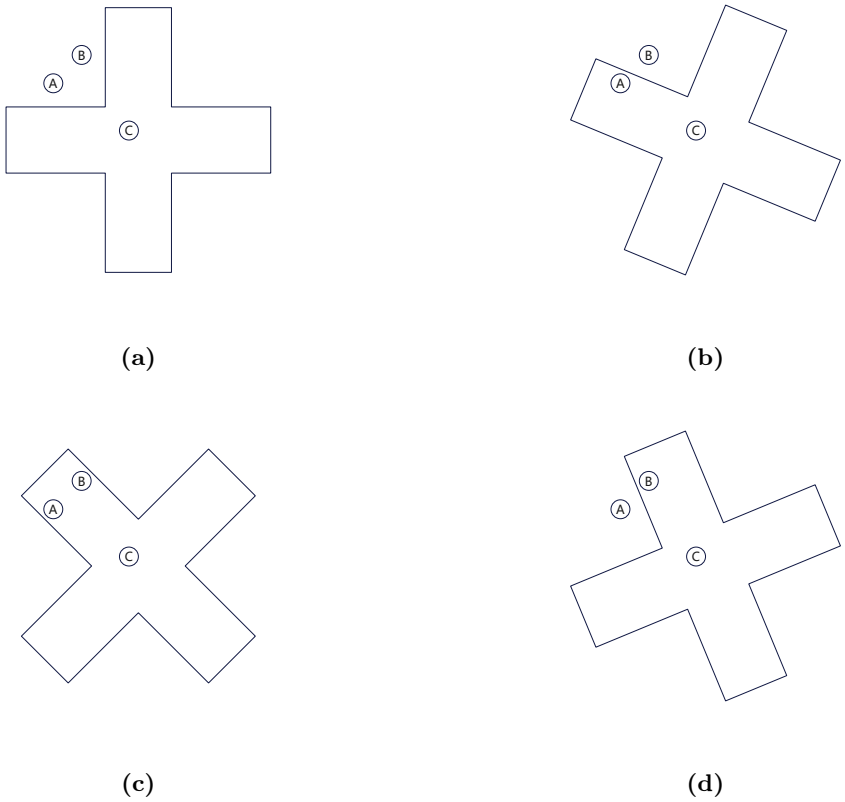
- `LiquidCrystal(rs, enable, d4, d5, d6, d7)` lager en instans av et `LiquidCrystal` objekt som representerer skjermens tilstand og brukes for å skrive tekst til skjermen.
- `begin(cols, rows)` instansierer skjermen med gitte dimensjoner.
- `print(data)` skriver tekst til skjermen.
- `setCursor(col, row)` setter posisjonen til "skrivehodet".
- `clear()` tømmer skjermen.
- `home()` returnerer skrivehodet til posisjon $(0, 0)$. Dette tilsvarer `setCursor(0, 0)`.

2.2.2 Enkoder med integrert trykknapp: WayinTop KY-040

Enkoderen vi brukte kan sees til venstre i figur 2.3. Dette er en generisk enkoder av typen KY-040 [20]. Den kan rotere uendelig i begge retninger og har integrert trykknapp. Når enkoderen skrus på, kan en føle små *klikk* for

hvert hakk enkoderen roterer. En full rotasjon er tilsvarende 20 endringer i posisjon, altså 20 *klikk*.

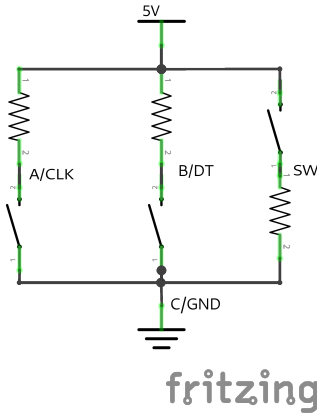
Selve enkoder-delen av innkapslingen kan illustreres som et "tannhjul" og tre kontaktpunkter – A, B, og C – som vist i figur 2.4; her illustrert med bare fire "tenner" istedetfor tjue som i den faktiske enkoderen.



Figur 2.4: En forenklet modell av enkoderen, her med bare fire "tenner". Tannhjulet roterer med klokken og lager periodisk kortslutning mellom C og de to andre kontaktpunktene.

Tannhjulet (krysset i figuren) roterer og kommer i kontakt med punktene og lager periodisk kortslutning mellom C og de to andre punktene. Enkoderen er montert på et kretskort sammen med pull-up-motstander som gjør at pins A og B vil gi høyt signal når de ikke er kortsluttet til C. En illustrasjon av enkoderens logiske oppbygning er vist i figur 2.5.

Fra disse illustrasjonene kan vi finne utgangsverdiene til A og B etter-



Figur 2.5: Enkoderen er konstruert slik at punktene A og B vil ha varierende kontinuitet med punkt C

hvert som hjulet roterer. Dette er vist i tabell 2.3, som viser utgangsverdiene for hver av subfigurene i figur 2.4.

Subfigur	A	B
a	1	1
b	0	1
c	0	0
d	1	0

Tabell 2.3: Utviklingen av verdiene til A og B etterhvert som enkoderen roterer.

Den gamle enkoderen har samme virkemåte, men har i tillegg en mikroprosessor som holder styr på tilstanden til enkoderen og sørger for kommunikasjon over I2C[1].

2.2.3 Kamera: UI-3140 og linse

Kameraet som er satt av er tilnærmet likt kameraet i den gamle riggen. Kameraet har en oppløsning 1280 ganger 1024 piksler og en maksimal bildehastighet på 169 bilder per sekund [22]. Eksponeringstiden kan stilles fra

[1]I2C, IIC eller I^{2}C er forkortelse for Inter-Integrated Circuit [21]

Figur 2.6: Kamera for ny rigg er av modellen IDS UI-3140

0.035ms til 434ms

2.3 Kort om PWM-styring av børstede motorer

Lineære regulatorer er egnet i tilfeller hvor spenningsnivået skal holdes konstant og nær inngangsverdien, og den totale effekten er generelt lav [23]. Desverre er denne typen regulator lite effektiv, og uegnet for å supplere større komponenter som motorer.

Pulsbreddemodulasjon (PWM, eng.: Pulse Width Modulation) brukes for å supplere motoren med strøm [24]. PWM fungerer ved å pulsere utgangen for å levere strøm i små øyeblikk av gangen, og er en svært populær metode for å styre komponenter som motorer eller lys. Ved å variere bredden på hver puls kan vi oppnå varierende gjennomsnittlig effekt.

Etterhvert som frekvensen øker, minner utgangen mer og mer om et analogt signal. Motoren fungerer også som spole, og har en treghet som gjør at den fungerer for å filtrere hastigheten.

Det er ønskelig å velge en frekvens for PWM-styring som er utenfor menneskets hørselsområde (over 20000Hz) for å unngå irriterende *spolehvin* (eng.: coil whine). Spolehvin er mer vitenskapelig kjent som *elektromagnetisk indusert akustisk støy* [25].

2.4 Tachometer

Et tachometer, også kjent som en turteller, er et instrument som måler rotasjonshastighet. For å konstruere et tachometer, kan vi ta signalet fra lysgaffelen og telle tiden mellom hver puls. Én puls tilsvarer én runde for skiven og vi kan dermed beregne rotasjonshastigheten som

$$\omega = \frac{360}{\Delta t}$$

hvor ω er gitt i grader per sekund og Δt er tid siden forrige puls. Vi kan konvertere denne verdien til rotasjoner per minutt (rpm) ved å dele på 6.

$$\text{rpm} = \frac{60}{\Delta t}$$

Vi bruker rotasjoner per minutt som mål for rotasjonshastighet.

2.5 PyuEye

Python-biblioteket PyuEye brukes for å få kontroll over funksjonalitet som trigger eller eksponeringstid. Biblioteket er preget av manuell minnehåndtering, primitive datatyper som ikke passer med Python's egne typer, og funksjoner som opererer på datastrukturer fremfor objekter med innebygd funksjonalitet.

Som eksempel er det i figur 2.3 inkludert kode for å sette trigger. En mye penere måte ville vært noe mer som `camera.setTrigger(TRIGGER_LO_HI, 25000)`.

```
1 ueye.is_SetExternalTrigger(self.hCam, ueye.IS_SET_TRIGGER_LO_HI)
2 d = ueye.int(delay)
3 ueye.is_SetTriggerDelay(self.hCam, d)
```

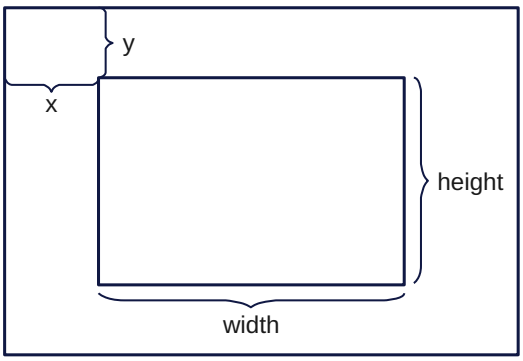
Oppføring 2.3: Kode for å sette trigger

Foruten trigger ønsker vi å kunne kontrollere noen forskjellige parametre via koden. Videre har vi forklart disse parametrene og hvordan vi kontrollerer dem.

2.5.1 Interesseområde/area of interest (AOI)

Interesseområdet til kameraet er den delen av kameraets sensor som er i bruk. Ved å redusere størrelsen på denne ender vi opp med færre piksler,

som fører til økt ytelse for både kameraet og for programmet som utfører prosessering.



Figur 2.7: Parametrene for interesseområde i PyuEye.

Vi setter interesseområdet med en gitt `x`, `y`, `width`, og `height` som spesifiserer et rektangel med en avstand fra øverste venstre hjørne.

```
1 rect_aoi = ueye.IS_RECT()
2 rect_aoi.s32X      = ueye.int(x)
3 rect_aoi.s32Y      = ueye.int(y)
4 rect_aoi.s32Width  = ueye.int(width)
5 rect_aoi.s32Height = ueye.int(height)
6 ueye.is_AOI(hCam, ueye.IS_AOI_IMAGE_SET_AOI,
7             rect_aoi, ueye.sizeof(rect_aoi))
```

Oppføring 2.4: Kode for å sette interesseområde

En fallgruve som man må være observant på er at bredden på interesseområdet må være delelig på 8. Om man setter `x` eller `width` til en verdi som ikke oppfyller dette vil man få et program som ikke starter. Det samme gjelder `y` og `height`, men disse kan bare være partall (delelig på 2).

2.5.2 Eksponeringstid

Eksponeringstiden på kameraet bestemmer hvor lenge kameraet skal slippe inn lys for et bilde. Dette justerer altså hvor mørkt eller lyst det ferdige bildet blir. For problemet som skal løses er det ønskelig med en lav eksponeringstid slik at disken rekker å bevege seg minst mulig mens bildet blir

tatt. Dette fører til et klarere bilde som er lettere å analysere.

Med dagslys er det realistisk å kunne bruke en eksponeringstid på 2ms eller kortere.

```
1 ms = ueye.DOUBLE(exposure)
2 ueye.is_Exposure(hCam, ueye.IS_EXPOSURE_CMD_SET_EXPOSURE,
3                  ms, ueye.sizeof(ms))
```

Oppføring 2.5: Kode for å sette eksponeringstid

2.5.3 Ta bilde

Til slutt trenger vi funksjonalitet for å ta bildene. Etter tilstrekkelig oppsett er det mulig å gjøre dette ved å først kalle på `ueye.is_FreezeVideo`, som kopierer bildedata inn i minne, for deretter å kalle `ueye.get_data` for å hente dataene fra kameraet. Om trigger er satt opp vil kameraet vente til trigger aktiveres før et bilde blir tatt.

2.6 OpenCV

OpenCV (for *Open Computer Vision*, eng.: åpent maskinsyn) er en samling biblioteker og funksjonalitet som fokuserer på maskinsyn i sanntid [2].

Studentene bruker denne funksjonaliteten for å behandle og utvinne informasjon fra bilder.

Det viktigste vi trenger å vite er at OpenCV forventer at bildene er i et spesielt format. Bildene fra kameraet, tatt ved hjelp av PyuEye er i et annet format enn forventet.

Fra kameraet får vi en éndimensjonal liste av verdier hvor OpenCV forventer en flerdimensjonal liste/matrise. Dimensjonene på denne listen vil variere etter formatet på bildet. For eksempel vil et bilde i monokrom være representert av en liste med dimensjoner $w \times h \times 1$ hvor w og h er bredde og høyde henholdsvis, og 1 representerer at det bare er én fargekanal.

Bildene vi får fra PyuEye er en liste pikselverdier med tre kanaler per piksel (blå, grønn, rød). Vi kan omtolke denne listen ved å benytte oss av et annet bibliotek: NumPy [26]. NumPy integrerer tett med svært mange andre biblioteker og inneholder funksjonalitet for å operere på større mengder data med høyere hastighet enn Python tradisjonelt er i stand til. I dette tilfellet trenger vi egentlig bare én funksjon: `reshape`.

`numpy.reshape` mottar en mengde data og en ønsket "form" og prøver

deretter å omtolke gitt data slik at det passer inn i formen. Vi ønsker å tolke bildet fra PyuEye som en tredimensjonal matrise med bredde og høyde lik `width` og `height`, samt dybde lik tre (én per fargekanal). Vi gjør dette v.hj.a. `numpy.reshape(img, (height, width, 3))`.

Fra kameraet har vi fått et bilde i RGB-format (rød, blå, grønn) i motsetning til BGR-format som er forventet av noen andre funksjoner, e.g. for å vise frem bilder som blir tatt. Vi kan konvertere fra førstnevnte til sistnevnte ved å bruke `cv2.cvtColor(img, cv2.RGB2BGR)`.

Etter alt dette er gjort er det mulig å vise bilde med normale farger på pc-skjermen med `cv2.imshow("image", img)`.

Kapittel 3

Vårt arbeid med konstruksjon av ny rigg og mal

Her forklarer vi hvordan den nye riggen er konstruert og hvordan malen er skrevet for å forenkle studentenes utførelse av øvingen. Denne delen av rapporten fungerer som dokumentasjon for riggen.

Vi har konstruert en rigg som oppfyller de viktigste punktene satt i oppgaven. Riggen er tilstrekkelig for øvingen som skal utføres og det er mulig å konfigurere alt som bør kunne konfigureres. Koden for riggen ligger vedlagt i vedlegg B. Malen ligger vedlagt i vedlegg D.

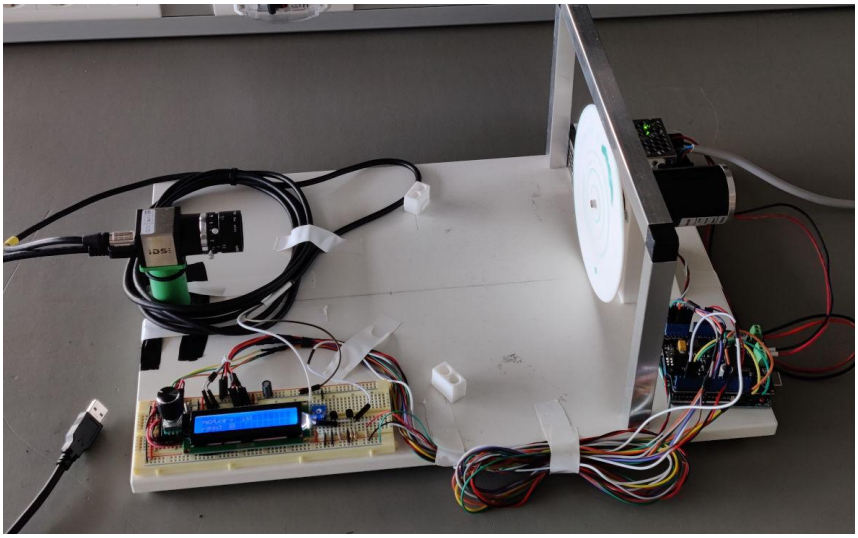
Et kretskjema med fullt overblikk og et "rottereir" (grafisk overblikk) er inkludert i vedlegg E.

3.1 Oppkobling og programmering av nye komponenter

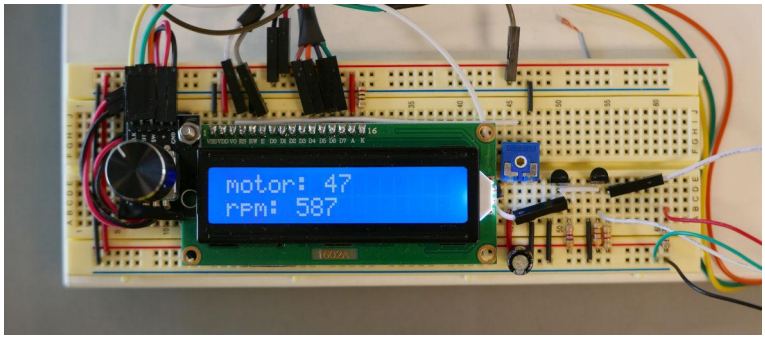
I dette delkapittelet forklarer vi oppkobling og programmering av komponentene. Vi har valgt denne strukturen fremfor å presentere hele oppkoblingen før hele programmeringen fordi vi mener at dette gir et bedre overblikk for hvordan prosjektet er modularisert.

3.1.1 Motordriver og motor (og lys)

Vi bruker samme oppkobling som den gamle riggen. Det vil si at vi benytter pins 5-10 på Arduino for å kontrollere h-bruene på motordriver-hatten og at motoren er koblet slik at svart ledning går til A- (OUT1) på motordriveren, og rød ledning går til A+ (OUT2). Gitt at A+ gir positiv spenning, og



Figur 3.1: Den ferdige riggen uten motiv på disken.



Figur 3.2: Alle komponentene montert på koblingsbrettet.

at A- tilsvarer jord, vil motoren da rotere med klokka, sett fra kameraets perspektiv.

Som nevnt i forrige kapittel, er det ønskelig med høyest mulig frekvens på PWM for å tilnærme et analogt signal. Motordriveren benytter pins 9 og 10 for PWM-signaler. Disse pinnene opererer med en frekvens på 490Hz [27].

Det er mulig å justere hastigheten på PWM ved å skrive spesielle verdier til spesielle registere [28]. Pins 9 og 10 som brukes for PWM til hatten kan konfigureres til 31000Hz, 4000Hz, 490Hz, 123Hz, eller 31Hz. Vi ønsker høyest

mulig verdi ettersom vi vil ha raskest mulig PWM. Derfor skulle vi nok ønske at vi kunne velge 31000Hz. Dessverre har motor-hatten en begrensning på 11kHz så vi må begrense oss til 4000Hz. For å sette frekvensen lik 4000Hz bruker vi `TCCR1B = TCCR1B & B11111000 | B00000010;`.

Vi bygger koden deretter. Koden er vist i oppføring 3.1.

```
1  #define PIN_MOTOR_A 5
2  #define PIN_MOTOR_B 6
3  #define PIN_MOTOR_PWM 9
4  #define MOTOR_PWM_MIN 10
5  #define MOTOR_PWM_MAX 120
6
7  static uint8_t motor_pwm;  // defaults to 0
8  void setup() {
9    pinMode(PIN_MOTOR_A, OUTPUT);
10   pinMode(PIN_MOTOR_B, OUTPUT);
11   pinMode(PIN_MOTOR_PWM, OUTPUT);
12
13   TCCR1B = TCCR1B & B11111000 | B00000010;
14
15   digitalWrite(PIN_MOTOR_A, LOW);
16   digitalWrite(PIN_MOTOR_B, HIGH);
17   analogWrite(PIN_MOTOR_PWM, 0);
18
19   motor_pwm = MOTOR_PWM_MIN;
20 }
```

Oppføring 3.1: Kode for å sette opp motorhatten og gjøre klar for PWM-kontroll av motor og lys."

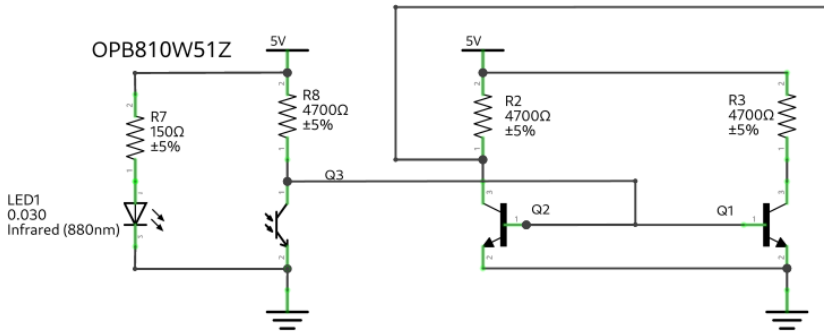
Det skjer et par ting i koden her. Først og fremst har vi definert noen pinner for å lettere huske hva som er hva. Dette gjør også at dersom vi bytter hvilken pinne som er koblet til hvor, er det bare å endre definisjonen og ikke alle lokasjoner i koden. Det neste vi har gjort er å definere noen grenser for hvor lavt eller høyt PWM-verdien kan gå. Formålet med dette er å forhindre at motoren går unødvendig fort eller at lyset blir unødvendig sterkt. Disse verdiene er hentet fra koden til den gamle riggen.

Resten av koden passer på at alle pinner som styrer motoren er satt som utganger og at PWM-frekvensen stilles riktig. De siste par linjene passer på at motoren får riktig polaritet og at startverdien til PWMen er korrekt. Koden her er repetert omtrent dobbelt opp ettersom utgangen for lys også skal styres, men for å unngå at koden tar opp unødvendig mye plass har vi valgt å utelate det.

3.1.2 Lysgaffel og kamera-trigger

For at lysgaffelen skal kunne benyttes av både Arduino og kamera i samarbeid, er vi nødt til å endre oppkoblingen litt, sammenliknet med den gamle oppkoblingen. I vår konfigurasjon får lysgaffelen strøm fra Arduino istedetfor fra kameraet. Dette gjør at tachometeret fungerer uten at kameraet er koblet til en PC.

Vi har inkludert et lite utdrag fra kretsskjemaet i figur 3.3.



Figur 3.3: Vi deler signalet fra lysgaffelen slik at det kan brukes til både Arduino og til kamera.

Ledningen som går opp, over, og ut til høyre går til pin 2 på Arduino og fungerer som avbrudd. Den tilsynelatende ubrukte transistoren er der for kameraet til å bruke. Dette er illustrert i rottereiret, men dukker ikke opp i kretsskjemaet.

Transistoren fungerer som en logisk inverter, som gjør at signalet vi får inn på Arduino, og på kameraet, er høyt mesteparten av tiden og så faller ned til jord når lyset brytes.

Dette er ikke et problem ettersom trigger og avbrudd går på transisjoner og ikke på tilstander. E.g. et signal som bare er høyt vil ikke ha noen effekt; det som har en effekt er et signal som går fra høyt til lavt eller omvendt.

Programmeringen er enkel og benytter et avbrudd, samt tidspunktet for forrige avbrudd for å bestemme hastighet.

```
1 #define PIN_OPTICAL_TRIGGER 2
2
3 void setup() {
4   pinMode(PIN_OPTICAL_TRIGGER, INPUT);
```

```
5   attachInterrupt(digitalPinToInterrupt(PIN_OPTICAL_TRIGGER),
6                   handleOpticalTrigger,
7                   RISING);
8 }
9
10 static bool checkTrigger;
11
12 void loop() {
13   static unsigned long rpm;
14   auto unow = micros()
15   static unsigned long prev_trig;
16
17   // nullstill rpm hvis det gaar mer enn ett sekund siden
18   // forrige oppdatering
19   if (unow - prev_trig > 1000000) {
20     rpm = 0;
21     prev_trig = unow;
22     forceUpdate = true;
23   }
24
25   if (checkTrigger) {
26     checkTrigger = false;
27     auto dt = unow - prev_trig;
28     if (unow < prev_trig) {
29       dt = 60000000;
30     }
31     prev_trig = unow;
32     rpm = 60000000 / (dt + 1);
33   }
34 }
35
36 static void handleOpticalTrigger() {
37   checkTrigger = true;
38 }
```

I håndteringsfunksjonen for triggeren settes bare ett flagg som sier at vi burde sjekke triggeren. Inni hovedløkka sjekker vi først om det er gått for lang tid mellom hver puls før vi sjekker om vi skal gjøre noe med triggeren. Hvis vi har mottatt en puls på triggeren kan vi som sagt beregne `rpm` som $60/(\Delta t)$ hvor Δt er tidsintervallet i sekunder. Ettersom vi benytter `micros()` er vi nødt til å multiplisere resultatet med 1000000. Derfor beregnes `rpm = 60000000 / (dt + 1)`. Vi setter `+ 1` og vi antar at `dt` aldri kan bli lavere enn 0. Ved å legge til 1 forsikrer vi oss om at vi aldri vil få en null-i-nevner-feil.

3.1.3 Enkoder

Enkoderen er koblet opp slik at CLK fra enkoderens kretskort er koblet til pin 3 på Arduino og er konfigurert som et avbrudd. DT er koblet til pin 4 og avleses som nødvendig.

Enkoder	Arduino
GND	GND
+	5V
SW	11
CLK	3
DT	4

Tabell 3.1: Tabell over koblinger: Enkoder til Arduino.

For å koble opp trykknappen valgte vi å bruke en vanlig digital pin ettersom vi har brukt opp alle pins som støtter avbrudd. Dette er et forholdsvis minimalt problem ettersom trykknappen benyttes relativt sjelden, sammenliknet med enkoderens `clk` eller lysgaffelens utgang. På den gamle riggen ble trykknappen brukt for å bytte mellom styring av motorens pådrag og styring av lysstyrke. Vi har beholdt denne funksjonaliteten.

Koden for å håndtere enkoderen er noe lang så vi velger å forklare den i to deler. Den første delen håndterer selve enkoderen og er vist i oppføring 3.2.

Vi setter opp et avbrudd som aktiveres hver gang CLK fra enkoderen endrer verdi. Inni funksjonen som håndterer avbruddet leser vi av enkoderens DT og CLK og lagrer resultatet i noen globale variabler. Til slutt setter vi et flagg som sier at vi bør sjekke enkoderens tilstand så snart vi har sjansen.

Inni `loop()` sjekker vi om enkoderen har oppdateringer som vi må håndtere. Hvis det er tilfellet setter vi flagget til `false` igjen. Vi setter også `forceUpdate` lik `true`; dette forklares senere i kapittelet. Deretter sjekker vi om vi bør stille på lysets pwm-verdi eller motorens. Her har vi utelatt koden for lyset for å spare plass, men den er omtrent lik koden for motoren.

Metoden for å bestemme retning på rotasjonen avhenger av observasjonen at når CLK endrer seg *før* DT betyr det at enkoderen roterer med klokken (vanligvis ansett som positiv retning for skruknotter som dette). I dette tilfellet vil verdiene være ulike. I motsatt retning vil DT endre seg før CLK og vi vil lese av like verdier.

For å programmere trykknappen måtte vi være kreative. Helst skulle vi

```
1  #define PIN_ENCODER_CLK 3
2  #define PIN_ENCODER_DT 4
3
4  void setup() {
5    pinMode(PIN_ENCODER_CLK, INPUT);
6    digitalWrite(PIN_ENCODER_CLK, HIGH);
7    attachInterrupt(digitalPinToInterrupt(PIN_ENCODER_CLK),
8                    handleClk,
9                    CHANGE);
10   pinMode(PIN_ENCODER_DT, INPUT);
11 }
12
13 static bool checkEncoder;
14 static uint8_t clk;
15 static uint8_t dat;
16
17 void loop() {
18   static bool light_adjust;
19
20   bool forceUpdate = false;
21   if (checkEncoder) {
22     checkEncoder = false;
23     forceUpdate = true;
24     if (light_adjust) {
25       ...
26     } else {
27       if (clk != dat) {
28         if (motor_pwm < MOTOR_PWM_MAX) motor_pwm++;
29       } else {
30         if (motor_pwm > MOTOR_PWM_MIN) motor_pwm--;
31       }
32       analogWrite(PIN_MOTOR_PWM, motor_pwm);
33     }
34   }
35 }
36
37 static void handleClk() {
38   dat = digitalRead(PIN_ENCODER_DT);
39   clk = digitalRead(PIN_ENCODER_CLK);
40   checkEncoder = true;
41 }
```

Oppføring 3.2: "Vi håndterer enkoderen ved hjelp av et avbrudd."

hatt én til avbruddspinne, men ettersom vi har brukt to stykker og Arduino ikke har flere enn to er vi nødt til å bruke noe annet.

```
1  #define PIN_ENCODER_SW 11
2
3  void setup() {
4    pinMode(PIN_ENCODER_SW, INPUT);
5  }
6
7  void loop() {
8    auto unow = micros();
9    static bool sw_low;
10   static unsigned long prev_sw;
11   if (digitalRead(PIN_ENCODER_SW)) {
12     if (sw_low && unow - prev_sw > 100000) {
13       sw_low = false;
14       prev_sw = unow;
15       light_adjust = !light_adjust;
16       forceUpdate = true;
17     }
18   } else {
19     sw_low = true;
20   }
21 }
```

Oppføring 3.3: "Vi er tom for pins som støtter avbrudd så vi er nødt til å ty til litt spesiell kode for å håndtere trykknappen"

Først sjekkes det om knappen er trykket inn. Deretter sjekkes det om knappen var sluppet ut i forrige iterasjon (knappen ble altså trykket nylig) og om dette er tilfellet så sjekker vi at det er gått lang nok tid siden forrige trykk. Hvis alt stemmer blir verdien av `light_adjust` snudd slik at om vi justerer motorens hastighet vil vi nå justere lysets styrke, og omvendt.

3.1.4 Skjerm

LCD-modulen krever minimum seks pins, hvorav fire data-pins (D4-D7) brukes for å sende data, én pin (EN, ”enable”) bestemmer om skjermen er mottakelig for kommandoer, og én pin (RS, ”register select”) velger om data tolkes som instruksjoner eller som symboler. Alle andre pins er ikke koblet til noe.

Vi bruker pins A0-A5 på Arduino som digitale pins.

Etter å ha dekket de to forhenværende komponentene virker programmeringen av skjermen relativt enkel. Koden er inkludert i oppføring 3.4.

Aller først etablerer vi en skjerm, en instans av `LiquidCrystal`. I `setup()`-funksjonen initialiserer vi størrelsen på skjermen v.hj.a. `lcd.begin(16,`

LCD	Arduino
VSS	GND
VDD	5V
RW	GND
A	5V
K	220R→5V
RS	A1
EN	A2
D4	A3
D5	A4
D6	A5
D7	A6

Tabell 3.2: Tabell over koblinger: LCD til Arduino.

2 og skriver en melding med `lcd.print("...")`.

Inni `loop()` sjekker vi om skjermen burde oppdateres, og om den skal det så tømmer vi skjermen og skriver ut den riktige pwm-verdien og rpm-verdien.

3.2 Kameramal

Kameraet er montert på riggen som i den gamle riggen; med en 3D-printet fot.

Vi har valgt å legge ved en mal som kan brukes som utgangspunkt for å gjennomføre øvingen. I denne seksjonen beskriver vi hvordan metodene i malen fungerer og hvorfor de er inkludert. En full mal finnes i vedlegg D.

3.2.1 Struktur

Vi har valgt å gjøre det veldig enkelt. Vi definerer en klasse kalt `Camera` som inneholder informasjon som trengs for å bruke kameraet.

Videre i dette delkapittelet definerer vi funksjonene `init()`, `capture()`, og `stop()`.

3.2.2 init()

`init()` er en funksjon definert på klassen `Camera` som konfigurerer og starter et kamera. Funksjonen tar syv parametre: `x`, `y`, `width`, `height` som definert i delkapittel 2.5.1, eksponeringstid i millisekunder - `exposure`,

```
#include <LiquidCrystal.h>
auto lcd = LiquidCrystal(14, 15, 16, 17, 18, 19);
void setup() {
  lcd.begin(16, 2);
  lcd.print("Hello, World!");
}

void loop() {
  if (forceUpdate) {
    prevUpdate = unow;
    // update display
    lcd.clear();
    lcd.home();

    if (light_adjust) {
      lcd.print("light: ");
      lcd.print(light_pwm);
    } else {
      lcd.print("motor: ");
      lcd.print(motor_pwm);
    }

    lcd.setCursor(0, 1);
    lcd.print("rpm: ");
    lcd.print(rpm);
  }
}
```

Oppføring 3.4: Programmering av skjermen."

```
class Camera:
    def __init__(self, number: int):
        self.pcMem = ueye.c_mem_p()
        self.memId = ueye.int()
        self.pitch = ueye.INT()
        self.hCam  = ueye.HIDS(number)
```

Oppføring 3.5: "Camera er en klasse som pakker funksjonalitet fra PyuEye inn i et enklere grensesnitt."

om trigger skal være aktiv - `trigger`, og i hvilket tilfelle, hvor lenge skal triggeren vente - `delay` (oppgitt i mikrosekunder).

3.2.3 Interesseområde/Area Of Interest (AOI)

Det første vi setter er interesseområdet. Dette er gjort i oppføring 3.6.

```
1  # inni init()
2  x      = x - x%8
3  width  = width - width%8
4  y      = y - y%2
5  height = height - height%2
6
7  self.width  = ueye.int(width)
8  self.height = ueye.int(height)
9  self.bpp    = ueye.int(24)
10
11 rect_aoi = ueye.IS_RECT()
12 rect_aoi.s32X      = ueye.int(x)
13 rect_aoi.s32Y      = ueye.int(y)
14 rect_aoi.s32Width  = ueye.int(width)
15 rect_aoi.s32Height = ueye.int(height)
16 ueye.is_AOI(self.hCam, ueye.IS_AOI_IMAGE_SET_AOI,
17             rect_aoi, ueye.sizeof(rect_aoi))
```

Oppføring 3.6: Sette interesseområde for kamera"

Det første som skjer er at vi runder ned `x` og `width` til nærmeste tall som er delelig på 8. Vi gjør det samme for `y` og `height`, men ned til nærmeste partall. Deretter lagrer vi verdiene for bredde og høyde og vi setter hvor mange bits vi ønsker per piksel.

Til slutt lager vi en struktur basert på et `IS_RECT()` som holder informasjon som brukes når interesseområdet skal settes. Vi setter verdiene i strukturen og setter interesseområdet med `ueye.is_AOI(...)`.

3.2.4 ColorMode og minne

De neste få linjene kode setter fargemodus for kameraet og allokerer nok minne til å holde et bilde som er bredde ganger høyde ganger bits per piksel. Koden er vist i oppføring 3.7.

Den første funksjonen setter fargemodus til `..._RGB8_...` som betyr tre kanaler og åtte bits per kanal per piksel. Dette tilsvarer de 24 bits per piksel vi satte av tidligere.

De neste tre funksjonene allokerer en bildebuffer som kan holde et bilde av interesseområdet.

3.2.5 Eksponeringstid

For å stille eksponeringstiden bruker vi linjene vist i oppføring 3.8.

```
# inni init()
ueye.is_SetColorMode(self.hCam, ueye.IS_CM_RGB8_PACKED)

ueye.is_AllocImageMem(self.hCam, self.width, self.height,
                      self.bpp, self.pcMem, self.memId)

ueye.is_AddToSequence(self.hCam, self.pcMem, self.memId)

ueye.is_InquireImageMem(self.hCam, self.pcMem, self.memId,
                        self.width, self.height, self.bpp,
                        self.pitch)
```

Oppføring 3.7: Forskjellige småfunksjoner i oppstart"

```
# inni init()
ms = ueye.DOUBLE(exposure)
ueye.is_Exposure(self.hCam,
                 ueye.IS_EXPOSURE_CMD_SET_EXPOSURE,
                 ms, ueye.sizeof(ms))
```

Oppføring 3.8: Stille eksponeringstiden"

3.2.6 Trigger

For å sette trigger bruker vi linjene vist i oppføring 3.9.

```
# inni init()
if trigger:
    ueye.is_SetExternalTrigger(self.hCam,
                               ueye.IS_SET_TRIGGER_LO_HI)
    d = ueye.int(delay)
    ueye.is_SetTriggerDelay(self.hCam, d)
```

Oppføring 3.9: "Aktivere trigger"

Denne koden kjører altså bare dersom trigger er satt til `true`.

3.2.7 Start kamera

Endelig, etter at alt av oppsett er gjort, kan vi starte kameraet med:
`ueye.is_CaptureVideo(self.hCam, ueye.IS_DONT_WAIT)`

3.2.8 Ta bilde

Funksjonen `capture()` brukes for å ta bilder og er inkludert i oppføring 3.10.

```
1 def capture(self):
2     ueye.is_FreezeVideo(self.hCam, ueye.IS_WAIT)
3     img = ueye.get_data(self.pcMem, self.width, self.height,
4                         self.bpp, self.pitch, False)
5     img = np.reshape(img, (self.height.value, self.width.value, 3))
6     return img
```

Oppføring 3.10: Ta et bilde"

Her gjør vi bare som beskrevet i teorien.

3.2.9 Stopp

Vi inkluderer selvfølgelig også en funksjon for å stoppe kameraet.

```
1 def stop(self):
2     ueye.is_StopLiveVideo(self.hCam, ueye.IS_FORCE_VIDEO_STOP)
3     ueye.is_FreeImageMem(self.hCam, self.pcMem, self.memId)
4     ueye.is_ExitCamera(self.hCam)
```

Oppføring 3.11: Stoppe kameraet."

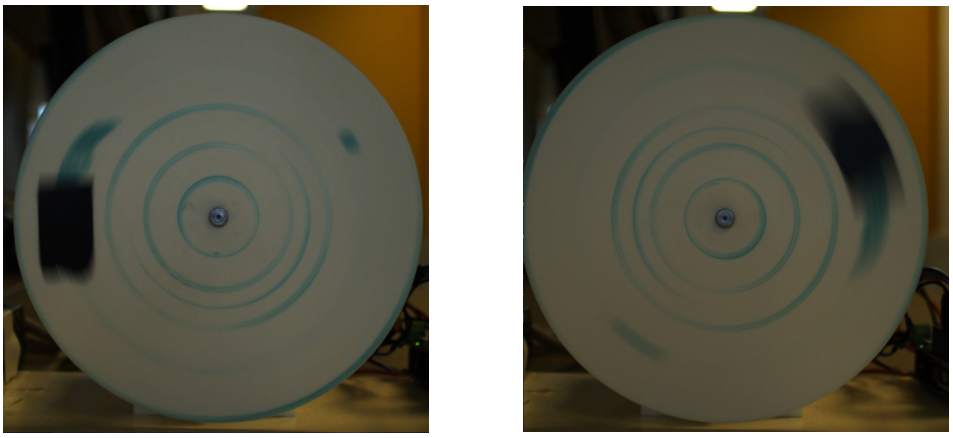
3.2.10 Utgangspunkt

Studentene bør ha et utgangspunkt å begynne med. Vi har laget en enkel `if __name__ == "__main__":` for dette formålet, inkludert i oppføring 3.12.

I dette eksempelet starter vi et kamera med interesseområde, eksponeringstid, og trigger med 25ms forsinkelse. Figur 3.4 viser (a) bilde tatt ved 500 runder per minutt, og (b) bilde tatt ved 1500 runder per minutt. Det er en klar og tydelig forskjell i posisjon mellom bildene.

```
1  if __name__ == "__main__":
2      cam = Camera(0) # create new camera
3      # Start camera with certain settings
4      cam.init(x = 176, y = 10,
5               width = 928, height = 928,
6              exposure = 2.5,
7              trigger = True, delay = 25000)
8      while True:
9          img = cam.capture()
10         img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
11
12         # CODE HERE
13
14         cv2.imshow("video", img)
15         if cv2.waitKey(25) & 0xFF == ord('q'):
16             break
17     cam.stop()
```

Oppføring 3.12: "Minimalt program som bruker trigger for å ta bilde hver gang skiven passerer."



(a) **(b)**

Figur 3.4: Bilde tatt med trigger og 25ms forsinkelse ved to forskjellige hastigheter.

Kapittel 4

Resultater og diskusjon

4.1 Testing

Det er ikke mye å teste med en sammensetning som dette ettersom vi ikke har noen tekniske krav eller spesifikasjoner som skal følges. Likevel er det noen ting vi kan inspisere.

4.1.1 Enkoder

Enkoderen opererer noenlunde trinnvis og er noe ustabil. Det er vanskelig å stille inn til en eksakt verdi på grunn av dette, men det er heller ikke nødvendig med eksakte verdier for typen kontroll som er gjort her. Vanligvis vil Arduino registrere to trinn for hvert "hakk" som en føler når en skrur på enkoderen, men i noen tilfeller vil kun ett trinn registreres.

Trykknappen fungerer uten problemer og bytter mellom å stille pwm-verdien for motoren og pwm-verdien for LED-stripen. Verdien som aktivt stilles, er vist på skjermen.

4.1.2 Tachometer

Vi har bekreftet at verdien som vises på skjermen stemmer overens med den faktiske hastigheten på motoren. Dette ble gjort ved å gjøre opptak av motoren i sakte film og deretter telle antall rotasjoner innen et tidsintervall.

Det er ingen aktiv kontroll av motorens hastighet; dvs. det er ingen kode som monitorerer motorens hastighet og stiller inngangsverdier for å oppnå en ønsket verdi. På grunn av dette har motoren en tendens til å variere i hastighet over tid.

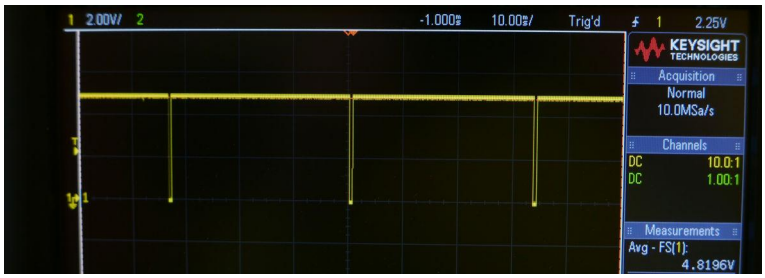
4.1.3 Lab-mal

Vi har testet lab-malen både med og uten trigger og vi har variert triggerens ventetid og har sett at resultatet er brukbart. Funksjonaliteten er tilstrekkelig for utførelse av lab-øvingen og vi finner ingen åpenbare problemer med koden.

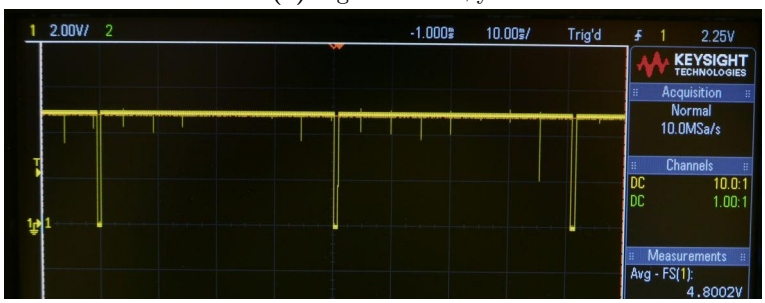
4.1.4 Spøkelsesfeil

I vår siste testing av riggen dukket opp et nytt problem som ikke hadde vist seg selv før. Av og til begynner klokkesignalet fra enkoderen og signalet fra lysgaffelen å støye; tilsynelatende tilfeldig. Denne støyen er så stor at Arduino til tider registrerer falske målinger på pins 2 og 3, som gjør at enkoderen og tachometeret oppfører seg ufint og gir feil verdier.

På grunn av den korte tiden som gjenstår har vi ikke fått diagnostisert årsak for feilen, men figur 4.1 viser signalet uten og med støy.



(a) Signal uten støy.



(b) Signal med støy.

Figur 4.1: Signalet som gis til Arduino fra lysgaffelen.

Støyen ser ut til å ha en frekvens på rundt 200Hz, gitt at hver rute er 10ms og det kommer 2 "tapper" i hver rute. Av det vi vet er det ingen kilde i nærheten som genererer digitale signaler ved 200Hz som kan skape den

støyen vi har observert. Dagen etter var støyen for det meste forsvunnet, men periodisk hopper rpm-verdien vist på skjermen til rundt 12000 rpm, som samsvarer med støy på 200Hz.

4.2 Fremtidig arbeid og forbedringer

Selv om riggen er funksjonell og ikke viser noen store problemer er det noen punkter en bør være observant på; spesielt om man ønsker å forbedre riggen videre i fremtiden.

4.2.1 Kretskort

Som nevnt er ikke koblingsbrett ment for permanente installasjoner. Vibrasjoner forekommer, fjærene i brettet kan bli løse over tid, og uhell kan skje hvor ledninger faller ut fra hvor de hører til.

Vi håper selvsagt at dokumentasjonen som er lagt frem i denne rapporten er tilstrekkelig for å kunne re-konstruere riggen i et "dommedagsscenario" om alt skulle gå galt, men det kan være fint med en mer permanent installasjon, loddet på et printet kretskort, eller et perforert hullkort [29].

4.2.2 Kjøre riggen over lengre tid

Vi har ikke utført testing som overstiger mer enn noen få minutter. Koden kan potensielt vise problemer rundt 70 minutter når `micros()`-funksjonen vil overflomme og vende rundt til 0.

4.2.3 Finne feilen

Vi vet fortsatt ikke hva som lager forstyrrelser på 200Hz. For at riggen skal være mest mulig stabil og brukbar burde denne feilen lukes ut da den er noe problematisk. Ting som kan prøves er å korte ned signalbaner, isolere ledninger bedre, og gjøre omfattende målinger.

Kapittel 5

Konklusjon

Vi har konstruert en kamerarigg som er i stand til å gjøre samme jobb som den gamle og noen nye ting. Den nye kamerariggen er også i stand til vise rotasjonshastigheten til skiven på en lcd-skjerm fortløpende. Dette gjør det mulig å se ut fra skjermen om studentene har funnet korrekt turtall eller ei.

Vi har også skrevet en mal som gjør det enklere for studentene å komme i gang med øvingen slik at de får brukt mer tid på interessante utfordringer fremfor å grave i gammel dokumentasjon og gjøre minnehåndtering i C.

Det er noe småplukk igjen som dukket opp helt på slutten som vi ikke fikk luket ut, men for det aller meste av tiden fungerer riggen uten problemer.

Vedlegg A

Original oppgavetekst: Forbedre kamerarigg for IA4 oppgave i ELE610

Oppgave passer kanskje som Bachelor-oppgave eller kanskje bedre som prosjektoppgave i ELE630. I begge tilfeller fortrinnsvis to studenter i lag.

A.1 Bakgrunn og problemstilling

Denne oppgaven IA4 går ut på å ta bilder av ei roterende skive montert på en kamerarigg. Bilde tas med trigger, og ved å bruke forsinkelse og eksponeringstid så bør det være muligå finne et rimelig godt estimat av rotasjonshastigheten for skiva. Oppgaven fungerer rimelig bra, men noen svakheter burde blitt utbedret. Det er bare en kamerarigg som er operativ. Denne beholdes foreløpig som den er. Men det har de siste årene vært mange studenter, og vi forventer fortsatt mange studenter. Derfor er det ønskelig med en kamerarigg til. Samtidig er det også ønske om noen forbedringer.

- Vi bør få en til bildefangstrigg operativ. Ramme rundt skiva er for å feste et stivt ark med nytt motiv eller bakgrunn for terninger på, men løsningen her er gjerne ikke helt optimal.
- Vi vet ikke faktisk rotasjonshastighet.
- For å finne vinkel for skiva ut fra et bilde trenger en jevn lyssetting og god oppløsning i det dynamiske fargeområdet, altså god utnytting av dynamisk område der pikselverdiene kan være. Kompensering av skjev

belysning med skalering er ikke ei god løsning i praksis. Dette skyldes delvis over eller under-eksponering, altså at piksel i de sterkt opplyste delene av bildet alle har verdi 255 (med lang eksponeringstid) eller at piksel i de mørkeste delene har verdi 0 (med kortere eksponeringstid) og delvis fordi det dynamiske området nødvendigvis må bli mindre når en kompenserer for lysstyrke.

- Oppgavetekst bør gjerne forbedres.

A.2 Oppgavetekst

Foreslå ei mer robust oppgave RS5. Arbeidet bør inneholde.

1. Det er deler til en ny rigg, første steg er da å få satt denne sammen slik at den virker og kan brukes. LED belysning er ikke nødvendig i første omgang. Kan løsning for feste av ark med motiv eller bakgrunn forbedres?

2. Det er en Arduino mikrokontroller som brukes for å styre hastigheten på skiva (med PWM?). Helt uavhengig av denne er det en lysgaffel som er tilpasset en tagg på skiva, denne lysgaffelen er nå koblet direkte til kamera uten noen forbindelse med Arduino. Det er ønskelig å koble dette sammen slik at Arduiono kan telle omdreininger og finne rotasjonshastighet og vise den på et lite enkelt LED display.

3. Montere og styre lysstyrke med LED lys plassert symmetrisk om skiva for å oppnå jevn lysstyrke. Disse kan monteres på ei ramme, plassert et stykke framfor skiva i retning mot kamera. Dette kan være tilsvarende ramme som står rundt skiva, men her med LED enten på begge sidene eller oppe og nede.

4. Gi forslag til hvordan jeg kan forbedre oppgaveteksten med bedre forklaring av prinsippet med å ta bilde med trigger, forklare og gi forståelige hint for hvordan vinkel for rød sektor på skiva kan finnes, forklare og gi forståelige hint for hvordan rotasjonshastighet for skiva kan finnes.

A.3 Krav til forkunnskaper

Dette er primært en oppgave for de som har tatt faget ELE610 i år eller i fjor. Det er bra om en kan programmere Arduino, eller ønsker å lære

seg dette. Det kan passe som Bachelor-oppgave eller som prosjektoppgave i ELE630. I begge tilfeller fortrinnsvis to studenter i lag.

A.4 Veiledere

Ståle Freyer og/eller Karl Skretting er aktuelle veiledere her. Ståle Freyer er personen som bygget den gamle riggen. Karl Skretting er personen som er satt opp som offisielle veileder og har ansvar for ELE610 og ELE630 som bruker riggen.

Vedlegg B

Ny kildekode for Arduino og programmeringsprosess

Dette er en komplett oppsummering av både koden og prosessen som brukes for å programmere Arduino. Noen endringer er gjort for å få kildekoden til å passe i dokumentet, men selve koden er ikke endret.

B.1 Kildekode

Dette er kildekoden for Arduino slik den er brukt i prosjektet.

```
1  #include <LiquidCrystal.h>
2
3  // pin for trigger
4  #define PIN_OPTICAL_TRIGGER 2 // interrupt
5
6  // pins for encoder
7  #define PIN_ENCODER_CLK 3 // interrupt
8  #define PIN_ENCODER_DT 4
9  #define PIN_ENCODER_SW 11
10
11 // pins and limits for motor
12 #define PIN_MOTOR_A 5
13 #define PIN_MOTOR_B 6
14 #define PIN_MOTOR_PWM 9
15 #define MOTOR_PWM_MIN 10
16 #define MOTOR_PWM_MAX 120
17
18 // pins and limits for light
19 #define PIN_LIGHT_A 7
20 #define PIN_LIGHT_B 8
```

```
21 #define PIN_LIGHT_PWM 10
22 #define LIGHT_PWM_MIN 0
23 #define LIGHT_PWM_MAX 25
24
25 // init display (uses analog pins as digital output)
26 auto lcd = LiquidCrystal(14, 15, 16, 17, 18, 19);
27 static uint8_t motor_pwm;  // defaults to 0
28 static uint8_t light_pwm;  // defaults to 0
29 void setup() {
30   // --- ENCODER SETUP ---
31   attachInterrupt(digitalPinToInterrupt(PIN_ENCODER_CLK),
32                   handleClk,
33                   CHANGE);
34   pinMode(PIN_ENCODER_DT, INPUT);
35   pinMode(PIN_ENCODER_SW, INPUT);
36
37   // --- TRIGGER SETUP ---
38   attachInterrupt(digitalPinToInterrupt(PIN_OPTICAL_TRIGGER),
39                   handleOpticalTrigger,
40                   RISING);
41
42   // --- LCD SETUP ---
43   lcd.begin(16, 2);
44   lcd.print("Hello, World!");
45
46   // --- MOTOR SETUP ---
47   pinMode(PIN_MOTOR_A, OUTPUT);
48   pinMode(PIN_MOTOR_B, OUTPUT);
49   pinMode(PIN_MOTOR_PWM, OUTPUT);
50
51   // for PWM frequency of 3921.16 Hz
52   // pins 9 & 10
53   // motor likes this better :)
54   TCCR1B = TCCR1B & B11111000 | B00000010;
55
56   digitalWrite(PIN_MOTOR_A, LOW);
57   digitalWrite(PIN_MOTOR_B, HIGH);
58   analogWrite(PIN_MOTOR_PWM, 0);
59
60   // --- LIGHT SETUP ---
61   digitalWrite(PIN_LIGHT_A, LOW);
62   digitalWrite(PIN_LIGHT_B, HIGH);
63   // will use same PWM frequency as motor
64   analogWrite(PIN_LIGHT_PWM, 0);
65
66   // --- DEFAULT VALUES ---
67   motor_pwm = MOTOR_PWM_MIN;
68   light_pwm = LIGHT_PWM_MIN;
69 }
```

```
70
71 // global variables, may be updated in interrupt handlers
72 static bool checkEncoder;
73 static uint8_t clk;
74 static uint8_t dat;
75 static bool checkTrigger;
76
77 void loop() {
78   static unsigned long rpm;
79   static bool light_adjust;  // adjust light- or motor-pwm
80   auto unow = micros();      // time now
81
82   // if encoder has been updated, force the display to update
83   bool forceUpdate = false;
84   // check encoder then increment or decrement the
85   // light or motor pwm
86   if (checkEncoder) {
87     checkEncoder = false;
88     forceUpdate = true;
89     if (light_adjust) {
90       if (clk != dat) {
91         if (light_pwm < LIGHT_PWM_MAX) light_pwm++;
92       } else {
93         if (light_pwm > LIGHT_PWM_MIN) light_pwm--;
94       }
95       analogWrite(PIN_LIGHT_PWM, light_pwm);
96     } else {
97       if (clk != dat) {
98         if (motor_pwm < MOTOR_PWM_MAX) motor_pwm++;
99       } else {
100        if (motor_pwm > MOTOR_PWM_MIN) motor_pwm--;
101      }
102      analogWrite(PIN_MOTOR_PWM, motor_pwm);
103    }
104  }
105
106  static unsigned long prev_trig;
107  if (unow - prev_trig > 1000000) {
108    rpm = 0;
109    prev_trig = unow;
110    forceUpdate = true;
111  }
112
113  if (checkTrigger) {
114    checkTrigger = false;
115    auto dt = unow - prev_trig;
116    if (unow < prev_trig) {
117      // looped around, set dt such that rpm becomes 0
118      dt = 60000000;
```

```
119       }
120       prev_trig = unow;
121       rpm = 60000000 / (dt + 1);
122     }
123
124     // if the switch was low in the previous iteration
125     static bool sw_low;
126     // last time the switch was pressed
127     static unsigned long prev_sw;
128     if (digitalRead(PIN_ENCODER_SW)) {
129       // if the state changed from low to high in this iteration
130       // and the appropriate time has passed for debouncing
131       if (sw_low && unow - prev_sw > 100000) {
132         sw_low = false;
133         prev_sw = unow;
134         light_adjust = !light_adjust;
135         forceUpdate = true;
136       }
137     } else {
138       sw_low = true;
139     }
140
141     static unsigned long prevUpdate;
142     if (unow - prevUpdate > 1000000) {
143       // if more than a second passes since we last wrote something
144       // to the display, force it to update
145       forceUpdate = true;
146     }
147
148     if (forceUpdate) {
149       prevUpdate = unow;
150       // update display
151       lcd.clear();
152       lcd.home();
153
154       // display the pwm-value we're currently adjusting
155       if (light_adjust) {
156         lcd.print("light: ");
157         lcd.print(light_pwm);
158       } else {
159         lcd.print("motor: ");
160         lcd.print(motor_pwm);
161       }
162
163       lcd.setCursor(0, 1);
164       lcd.print("rpm: ");
165       lcd.print(rpm);
166     }
167 }
```

```
168
169 // --- INTERRUPT HANDLERS ---
170
171 // interrupt handlers should be kept as short as possible
172 // all real logic is handled in main loop, these just
173 // ensure appropriate capture of values.
174
175 // interrupt handler for the encoder clk
176 static void handleClk() {
177   // read pins, then set flag
178   clk = digitalRead(PIN_ENCODER_CLK);
179   dat = digitalRead(PIN_ENCODER_DT);
180   checkEncoder = true;
181 }
182
183 // interrupt handler for the optical trigger
184 static void handleOpticalTrigger() {
185   // just set flag
186   checkTrigger = true;
187 }
```

Oppføring B.1: Kode for Arduino i ny rigg

Vedlegg C

Gammel kildekode

Dette er kildekoden slik den var i den gamle riggen. Merk at koden refererer til en ekstern "header" (`i2cEncoderLib.h`) som må inkluderes manuelt [30]. Noen endringer er gjort for å få kildekoden til å passe i dokumentet, men selve koden er ikke endret.

C.1 Kildekode

```
1  /*H*************************************************************
2  * FILENAME :        bildelab.ino
3  *
4  * DESCRIPTION :
5  *               Control of rotating image target.
6  *
7  * NOTES :
8  *
9  * AUTHOR :      Staale Freyer
10 * START DATE :  20 Aug 2018
11 *
12 * CHANGES :
13 *
14 * REF NO  VERSION DATE    WHO     DETAIL
15 *
16 *
17 *H*/
18
19
20 //https://playground.arduino.cc/Main/RotaryEncoders
21 //https://www.cui.com/product/resource/c14.pdf
22
23 //PIN's definition
```

```
24
25 #include <stdio.h> // for function sprintf
26 #include "math.h"
27 #include "i2cEncoderLib.h"
28
29 #define motorPinA 5
30 #define motorPinB 6
31 #define motorPWM  9
32
33 #define LEDpinA   7
34 #define LEDpinB   8
35 #define LEDPWM   10
36
37 #define minMotorPWM    18
38 #define maxMotorPWM   120
39 #define minLedPWM       0
40 #define maxLedPWM      25
41
42 #define maxChange  500
43 #define tUpdate  50
44
45 #define ecfg (INTE_ENABLE
46               | LEDE_ENABLE
47               | WRAP_DISABLE
48               | DIRE_LEFT
49               | IPUP_ENABLE
50               | RMOD_X1 )
51
52 const int IntPin = 12;
53 unsigned int encoder_status;
54 long maxvalue = maxMotorPWM;
55 long minvalue =  minMotorPWM;
56 long counter  =   minvalue;
57 double econfig  = ecfg;
58
59 //Class initialization with the I2C addresses
60 i2cEncoderLib encoder = i2cEncoderLib(0x30);
61
62
63 void setup() {
64   Serial.begin(115200);
65   Serial.print("\e[3J");
66   Serial.println("\nBildelab:");
67
68   analogWrite(motorPWM, 0);
69   analogWrite(LEDPWM, 0);
70
71   pinMode(LEDpinA, OUTPUT);
72   digitalWrite(LEDpinA, LOW ); // Set polarity neg OUT3
```

```
73    pinMode(LEDpinB, OUTPUT);
74    digitalWrite(LEDpinB, HIGH); // Set polarity pos OUT4
75
76    pinMode( motorPinA, OUTPUT);
77    digitalWrite(motorPinA, LOW);
78    pinMode( motorPinB, OUTPUT);
79    digitalWrite(motorPinB, HIGH);
80
81    pinMode( motorPWM,  OUTPUT);
82
83    setPwmFrequency(motorPWM, 2); //4kHz
84
85    pinMode(IntPin, INPUT);
86    Serial.println(econfig, HEX);
87    encoder.begin(econfig);
88    encoder.writeCounter(counter);
89    encoder.writeMax(maxvalue);
90    encoder.writeMin(minvalue);
91    encoder.updateStatus();
92
93  }
94
95  void loop() {
96    int nd;
97    unsigned long pct;
98    unsigned long setPWM  = 1;
99    unsigned long PWM     = 0;
100   unsigned long LEDint = 0;
101   long diffPWM = 0;
102   boolean LEDon = false;
103   boolean Mode = true;
104   unsigned long n = 0;
105
106   char op[60];
107   while (true) {
108     if (encoder.readStatus(E_PUSH)) {
109       //encoder.writeCounter(0); // Stop if button pushed
110       Mode = ! Mode;
111       if (Mode) {
112         encoder.writeMin(minMotorPWM);
113         encoder.writeMax(maxMotorPWM);
114         encoder.writeCounter(PWM);
115       }
116       else {
117         encoder.writeMin(minLedPWM);
118         encoder.writeMax(maxLedPWM);
119         encoder.writeCounter(LEDint);
120       }
121     }
```

```
122     if (encoder.updateStatus())
123       if (Mode)
124         PWM = encoder.readCounterByte();
125       else
126         LEDint = encoder.readCounterByte();
127     diffPWM = PWM - setPWM;
128     setPWM += limit(&diffPWM, -maxChange, maxChange);
129     if (PWM == minvalue) {
130       setPWM = 0;
131     }
132
133     analogWrite(motorPWM, setPWM);
134     analogWrite(LEDPWM, LEDint*10);
135     pct = setPWM * 100;
136     pct /= maxMotorPWM;
137     limit(pct, 0, 100);
138     sprintf(op, "%6d %6d %6d %6d%% %6d \r", n++,
139                 (long)PWM,
140                 (long)setPWM,
141                 (long)pct,
142                 (long)LEDint);
143     Serial.print(op);
144     delay(tUpdate);
145   }
146 }
147
148 //https://playground.arduino.cc/Code/PwmFrequency
149 bool setPwmFrequency(int pin, int divisor) {
150   byte mode;
151   switch (divisor) {
152     case 1: divisor = 1; break;     // 31250 Hz /    1 = 31250 Hz
153     case 2: divisor = 8; break;     // 31250 Hz /    8 =  3906 Hz
154     case 3: divisor = 64; break;    // 31250 Hz /   64 =   488 Hz
155     case 4: divisor = 256; break;   // 31250 Hz /  256 =   122 Hz
156     case 5: divisor = 1024; break; // 31250 Hz / 1024 =  30,5 Hz
157     default: break;
158   }
159   if (pin == 5 || pin == 6 || pin == 9 || pin == 10) {
160     switch (divisor) {
161       case 1: mode = 0x01; break;
162       case 8: mode = 0x02; break;
163       case 64: mode = 0x03; break;
164       case 256: mode = 0x04; break;
165       case 1024: mode = 0x05; break;
166       default: return false;
167     }
168     if (pin == 5 || pin == 6) {
169       TCCR0B = TCCR0B & 0b11111000 | mode;
170     } else {
```

```
171       TCCR1B = TCCR1B & 0b11111000 | mode;
172     }
173   } else if (pin == 3 || pin == 11) {
174     switch (divisor) {
175       case 1: mode = 0x01; break;
176       case 8: mode = 0x02; break;
177       case 32: mode = 0x03; break;
178       case 64: mode = 0x04; break;
179       case 128: mode = 0x05; break;
180       case 256: mode = 0x06; break;
181       case 1024: mode = 0x07; break;
182       default: return false;
183     }
184     TCCR2B = TCCR2B & 0b11111000 | mode;
185   }
186   return true;
187 }
188
189 //https://forum.arduino.cc/index.php?topic=37804.0
190 static inline int8_t sgn(int val) {
191   if (val < 0) return -1;
192   if (val == 0) return 0;
193   return 1;
194 }
195
196 int limit(long *x, long low, long high) {
197   if (*x > high)
198     *x = high;
199   else if (*x < low)
200     *x = low;
201   return *x;
202 }
```

Oppføring C.1: Kode for Arduino i gammel rigg

Vedlegg D

Mal for lab-øving

Her legger vi frem en mal for lab-øving IA4. Malen inneholder de viktige elementene som bildefangst og kamerakonfigurasjon med trigger, interesseområde, og eksponeringstid.

Det bør være mulig å bruke metodene definert her i et mer komplett program med GUI og interaktivitet, men vi mener at dette er et godt grunnlag for å løse kjernen av oppgaven.

Koden er konstruert for å være så enkel som mulig å bruke. Nederst er det inkludert et utgangspunkt som gjelder for kameraet på den nye riggen.

```
1  import cv2
2  import matplotlib.pyplot as plt
3  import numpy as np
4  from pyueye import ueye
5  import datetime
6
7  class Camera:
8      def __init__(self, number: int):
9          self.pcMem = ueye.c_mem_p()
10         self.memId = ueye.int()
11         self.pitch = ueye.INT()
12         self.hCam  = ueye.HIDS(number)
13
14
15     # init starts the camera with given settings
16     # x and y are the horisontal and vertical offsets from the top
17     #   left corner
18     # x and width are rounded down to the nearest multiple of 8
19     # y and height are rounded down to the nearest multiple of 2
20     # exposure is a float number
21     # delay is given in microseconds as an int
22     def init(self, x: int = 0, y: int = 0, width: int = 1000,
```

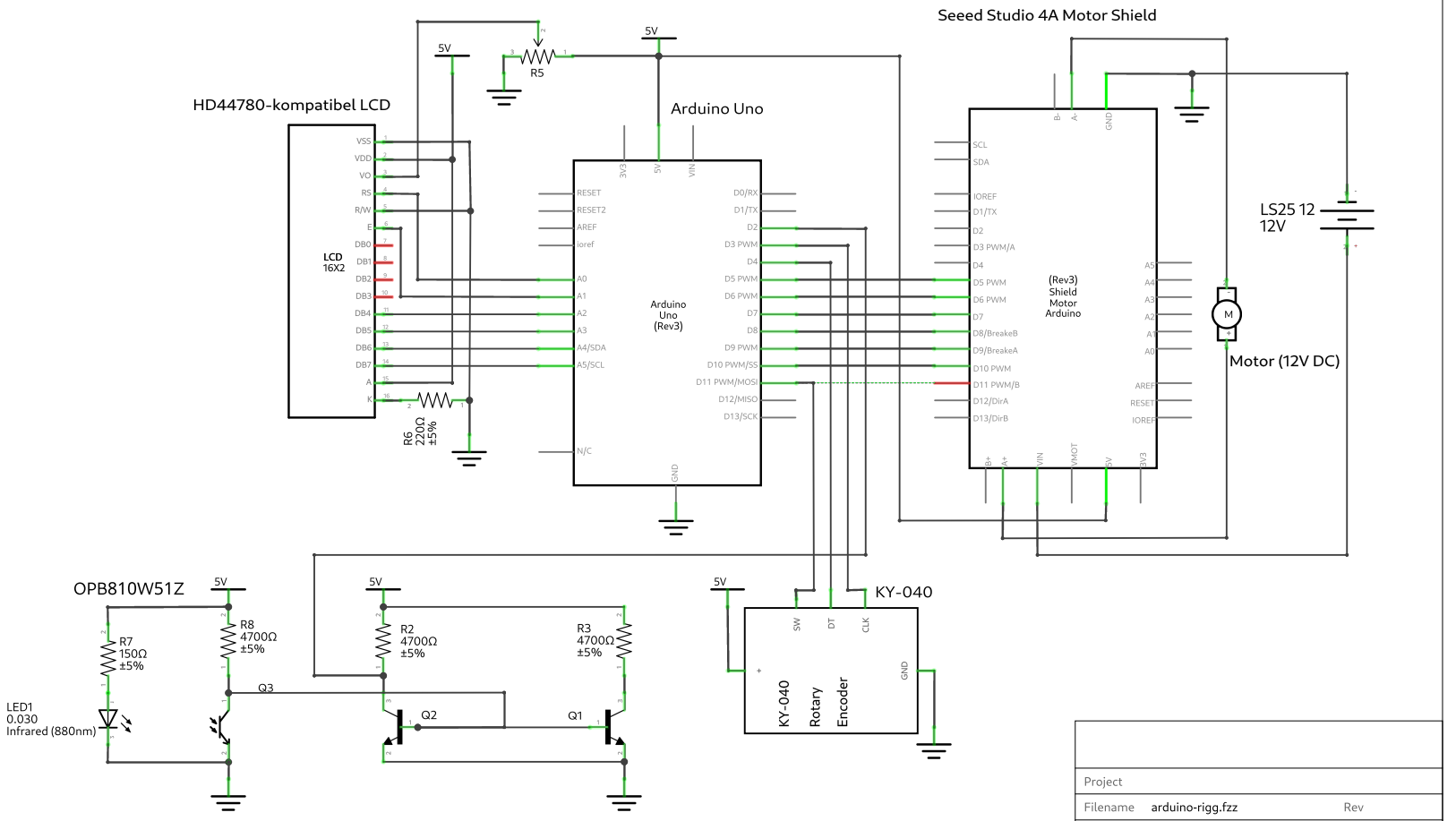
```
23              height: int = 1000, exposure: float = 2.0,
24              trigger: bool = False, delay: int = 0):
25          ueye.is_InitCamera(self.hCam, None)
26
27          # We do this automatically instead of letting
28          # students locate this hard to find bug
29          x = x - x%8
30          width = width - width%8
31          y = y - y%2
32          height = height - height%2
33
34          self.width  = ueye.int(width)
35          self.height = ueye.int(height)
36          self.bpp    = ueye.int(24)
37
38          rect_aoi = ueye.IS_RECT()
39          rect_aoi.s32X      = ueye.int(x)
40          rect_aoi.s32Y      = ueye.int(y)
41          rect_aoi.s32Width  = ueye.int(width)
42          rect_aoi.s32Height = ueye.int(height)
43          ueye.is_AOI(self.hCam, ueye.IS_AOI_IMAGE_SET_AOI,
44                      rect_aoi, ueye.sizeof(rect_aoi))
45
46          ueye.is_SetColorMode(self.hCam, ueye.IS_CM_RGB8_PACKED)
47          ueye.is_AllocImageMem(self.hCam, self.width, self.height,
48                                self.bpp, self.pcMem, self.memId)
49          ueye.is_AddToSequence(self.hCam, self.pcMem, self.memId)
50          ueye.is_InquireImageMem(self.hCam, self.pcMem, self.memId,
51                                  self.width, self.height, self.bpp,
52                                  self.pitch)
53
54          ms = ueye.DOUBLE(exposure)
55          ueye.is_Exposure(self.hCam,
56                           ueye.IS_EXPOSURE_CMD_SET_EXPOSURE,
57                           ms, ueye.sizeof(ms))
58
59          if trigger:
60              ueye.is_SetExternalTrigger(self.hCam,
61                                         ueye.IS_SET_TRIGGER_LO_HI)
62              d = ueye.int(delay)
63              ueye.is_SetTriggerDelay(self.hCam, d)
64
65          ueye.is_CaptureVideo(self.hCam, ueye.IS_DONT_WAIT)
66
67
68      # capture captures an image in BGR format
69      # if trigger is set, waits for trigger
70      # returns the image as a numpy multidimensional array
71      def capture(self):
```

```
72          ueye.is_FreezeVideo(self.hCam, ueye.IS_WAIT)
73          img = ueye.get_data(self.pcMem, self.width, self.height,
74                              self.bpp, self.pitch, False)
75          img = np.reshape(img,
76                           (self.height.value, self.width.value, 3))
77          return img
78
79
80      # stop stops the camera and exits cleanly
81      def stop(self):
82          ueye.is_StopLiveVideo(self.hCam, ueye.IS_FORCE_VIDEO_STOP)
83          ueye.is_FreeImageMem(self.hCam, self.pcMem, self.memId)
84          ueye.is_ExitCamera(self.hCam)
85
86
87  if __name__ == "__main__":
88      cam = Camera(0) # create new camera
89      # Start camera with certain settings
90      cam.init(x = 176, y = 10,
91               width = 928, height = 928,
92              exposure = 2.5,
93              trigger = True, delay = 25000)
94      while True:
95          img = cam.capture()
96          img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
97
98          # CODE HERE
99
100         cv2.imshow("video", img)
101         if cv2.waitKey(25) & 0xFF == ord('q'):
102             break
103     cam.stop()
```

Oppføring D.1: Mal for bildelab

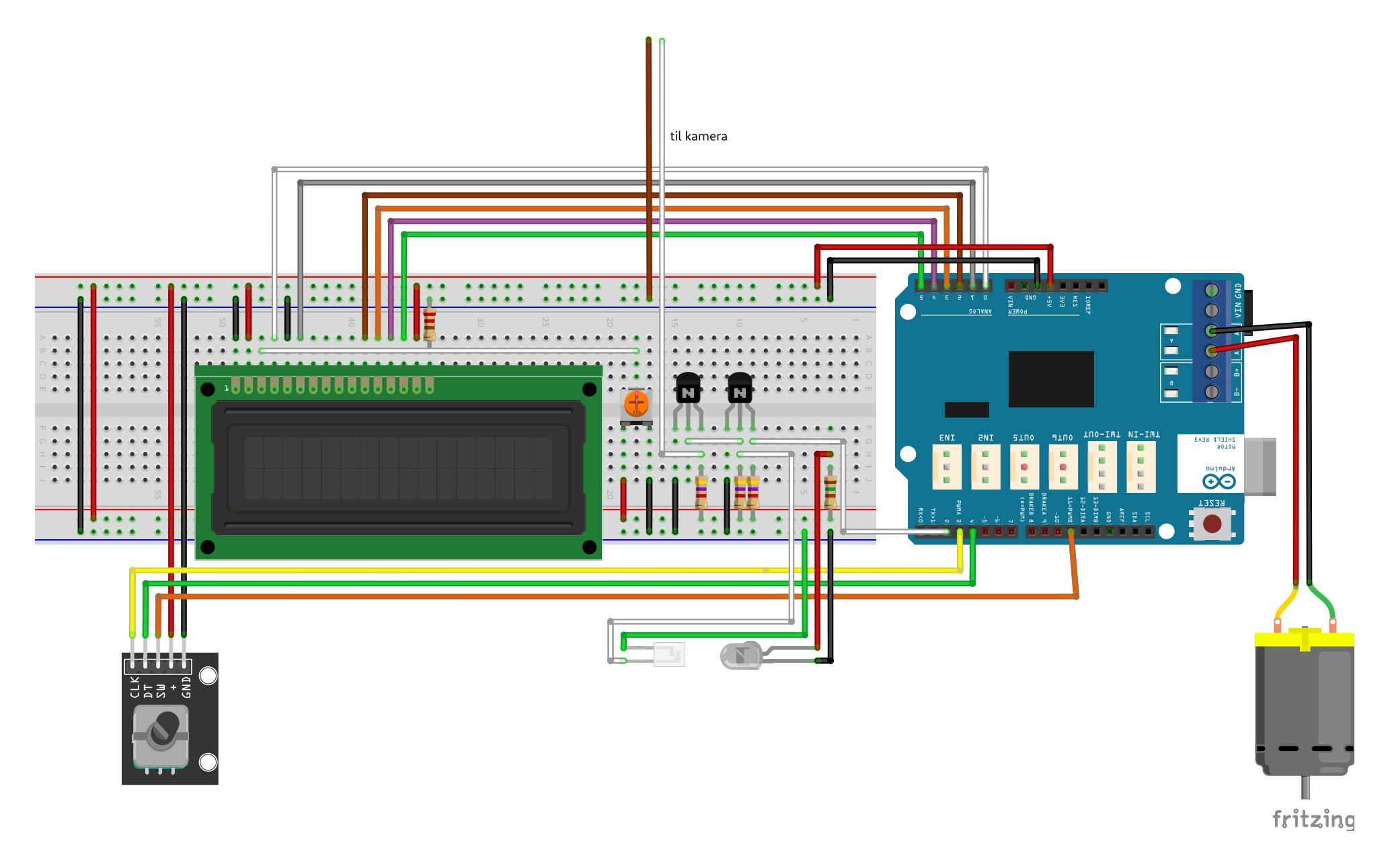
Vedlegg E

Kretsskjema og rottereir


Seeed Studio 4A Motor Shield
HD44780-kompatibel LCD
Arduino Uno
5V
R5
LCD
16X2
VSS
VDD
VO
RS
R/W
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A
K
R6
220Ω
±5%
RESET
RESET2
AREF
Ioref
3V3
5V
VIN
A0
A1
A2
A3
A4/SDA
A5/SCL
N/C
GND
D0/RX
D1/TX
D2
D3 PWM
D4
D5 PWM
D6 PWM
D7
D8
D9 PWM
D10 PWM/SS
D11 PWM/MOSI
D12/MISO
D13/SCK
Arduino
Uno
(Rev3)
SCL
SDA
IOREF
D1/TX
D2
D3 PWM/A
D4
D5 PWM
D6 PWM
D7
D8/BreakeB
D9/BreakeA
D10 PWM
D11 PWM/B
D12/DirA
D13/DirB
B-
A-
GND
A5
A4
A3
A2
A1
A0
AREF
RESET
IOREF
B+
A+
VIN
VMOT
5V
3V3
(Rev3)
Shield
Motor
Arduino
LS25 12
12V
M
Motor (12V DC)
OPB810W51Z
R7
150Ω
±5%
R8
4700Ω
±5%
LED1
0.030
Infrared (880nm)
Q3
R2
4700Ω
±5%
Q2
R3
4700Ω
±5%
Q1
KY-040
SW
DT
CLK
GND
KY-040
Rotary
Encoder
Project
Filename
arduino-rigg.fzz
Rev
Date
14 May 2022 17:36:58
Sheet
1/1

a

fritzing


til kamera
CLK
DT
SW
+
GND
VIN
GND
5V
RES
3V3
IOREF
A0
A1
A2
A3
A4
A5
ANALOG
POWER
M1
M2
B+
B-
TWI-IN
TWI-OUT
OUTB
OUTS
INS
IN2
IN3
MOTOR SHIELD REV3
Arduino
RESET
SCL
SDA
AREF
GND
13-SCK
12-MISO
11-MOSI
~10
BRAKEB 9
BRAKEA 8
7
~6
~5
4
PWMA 3
2
TX→1
RX←0
fritzing

Bibliografi

[1] Python Software Foundation. «Python». (2022), adresse: https://www.python.org/ (sjekket 11.05.2022).

[2] OpenCV team. «OpenCV». (2022), adresse: https://opencv.org/ (sjekket 11.05.2022).

[3] IDS. «pipy: pyueye - Python bindings for uEye API.» (2021), adresse: https://pypi.org/project/pyueye (sjekket 30.03.2022).

[4] UiS. «Praktisk robotteknikk ELE610». (2022), adresse: https://www.uis.no/nb/course/ELE610_1 (sjekket 05.04.2022).

[5] Wikipedia bidragsytere. «AVR microcontrollers». (2022), adresse: https://en.wikipedia.org/wiki/AVR_microcontrollers (sjekket 08.05.2022).

[6] Arduino. «Arduino Uno & Genuino Uno». (2022), adresse: https://www.arduino.cc/en/main/arduinoBoardUno (sjekket 01.05.2022).

[7] ——, «Analog Input Pins». (2022), adresse: https://docs.arduino.cc/learn/microcontrollers/analog-input (sjekket 01.05.2022).

[8] *BRUSH MOTORS - The power of silence*, Crouzet, 2017. adresse: https://no.mouser.com/datasheet/2/92/Crouzet_brochure_Dcmind_brush_english-1108910.pdf (sjekket 01.05.2022).

[9] ——, «Arduino Uno & Genuino Uno - Technical Specifications». (2022), adresse: https://www.arduino.cc/en/main/arduinoBoardUno#techspecs (sjekket 01.05.2022).

[10] C. P. Solutions. «What are the Maximum Power Output and Data Transfer Rates for the USB Standards?» (2020), adresse: https://resources.pcb.cadence.com/blog/2020-what-are-

the-maximum-power-output-and-data-transfer-rates-for-the-usb-standards (sjekket 01.05.2022).

[11] S. Studio. «4A Motor Shield». (2021), adresse: https://wiki.seeedstudio.com/4A_Motor_Shield/ (sjekket 01.05.2022).

[12] *5.0 A Throttle Control H-Bridge*, Freescale Semiconductor, 2012. adresse: https://files.seeedstudio.com/wiki/4A_Motor_Shield/res/MC33932.pdf (sjekket 01.05.2022).

[13] *Wide Gap Slotted Optical Switch*, OPTEK Technology, 2013. adresse: https://no.mouser.com/datasheet/2/414/OPTIS01664_1-2564908.pdf (sjekket 14.05.2022).

[14] *UI-3360CP-NIR-GL Rev.2 (AB00624)*, IDS Imaging Development Systems GmbH, 2022. adresse: https://en.ids-imaging.com/IDS/datasheet_pdf.php?sku=AB00624 (sjekket 02.05.2022).

[15] Arduino. «Arduino IDE». (2022), adresse: https://www.arduino.cc/en/software (sjekket 15.05.2022).

[16] ——, «arduino-cli». (2022), adresse: https://github.com/arduino/arduino-cli (sjekket 15.05.2022).

[17] *AD00044.03*, IDS Imaging Development Systems GmbH, Ukjent år. adresse: https://en.ids-imaging.com/tl_files/downloads/STORE/acc_spec/233/datasheet/AD00044.03_EN.pdf (sjekket 02.05.2022).

[18] *Datasheet LCD*, Hitachi Ltd., 1998. adresse: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf (sjekket 11.05.2022).

[19] ——, «LiquidCrystal Library». (2019), adresse: https://www.arduino.cc/en/Reference/LiquidCrystal (sjekket 30.03.2022).

[20] *Datasheet Enkoder*, Handson Technology, 2018. adresse: https://www.handsontec.com/dataspecs/module/Rotary%20Encoder.pdf?fbclid=IwAR00jxxk2KswjfxrdtXHRNp5-H-SU3-XJzRz8KjqZF1nRfI_0kxGcPbx5kE (sjekket 11.05.2022).

[21] Wikipedia bidragsytere. «I2C». (2022), adresse: https://en.wikipedia.org/wiki/I%5C%C2%5C%B2C (sjekket 08.05.2022).

[22] *UI-3140CP-C-HQ Rev.2 (AB00614)*, IDS Imaging Development Systems GmbH, 2022. adresse: https://en.ids-imaging.com/IDS/datasheet_pdf.php?sku=AB00614 (sjekket 08.05.2022).

[23] ——, «Linear regulator». (2022), adresse: https://en.wikipedia.org/wiki/Linear_regulator (sjekket 08.05.2022).

[24] ——, «Pulsbreddemodulasjon». (2022), adresse: https://no.wikipedia.org/wiki/Pulsbreddemodulasjon (sjekket 08.05.2022).

[25] ——, «Electromagnetically induced acoustic noise». (2022), adresse: https://en.wikipedia.org/wiki/Electromagnetically_induced_acoustic_noise (sjekket 13.05.2022).

[26] NumPy. «NumPy». (2022), adresse: https://numpy.org/ (sjekket 15.05.2022).

[27] Arduino. «analogWrite()». (2020), adresse: https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/ (sjekket 11.05.2022).

[28] eTechnophiles. «How To Change Frequency On PWM Pins Of Arduino UNO». (2020), adresse: https://www.etechnophiles.com/change-frequency-pwm-pins-arduino-uno/ (sjekket 15.05.2022).

[29] Wikipedia bidragsytere. «Perfboard». (2022), adresse: https://en.wikipedia.org/wiki/Perfboard (sjekket 11.05.2022).

[30] F. Saimon. «I2C Encoder Arduino Library». (2017), adresse: https://github.com/Fattoresaimon/i2cencoder/tree/master/Arduino%20Library (sjekket 01.05.2022).