Automatisk utsetting og innhenting av sensorer i innsjø

Av Halvard Yri Adriaenssens

Veileder

Helge Balk

Universitetet i Oslo, våren 2020

Fysisk institutt

Det matematisk-naturvitenskapelige fakultet

Forord

Jeg valgte denne oppgaven fordi det var en naturlig fortsettelse på bacheloroppgaven jeg skrev, som handlet om utviklingen av et soldrevet autonomt fartøy. Fartøyet var ment til å kunne dekke store avstander på havet ved å være selvforsynt med energi via et solcellepanel. Jeg trivdes godt med å jobbe med den oppgaven, så da jeg fikk muligheten til å jobbe med en relatert oppgave som masteroppgave var dette noe jeg ønsket. Jeg likte godt at oppgaven var relatert til et virkelig behov i samfunnet.

Denne oppgaven har lært meg mye om meg selv, spesielt selvdisiplin og strukturering, da jeg har brukt store deler av 60 studiepoeng til å jobbe alene. Jeg har tilegnet meg helt ny kunnskap om maskinsyn og programmering, noe som har vært spennende og sette seg inn i. Jeg har fått muligheten til å sette opp innebygde datasystemer til å løse bestemte oppgaver, som er noe jeg syntes er interessant.

Jeg vil gjerne takke min veileder, førsteamanuensis Helge Balk, som har delt rikelig av sin ekspertise og erfaring. Jeg har satt stor pris på friheten jeg har fått til å gjøre oppgaven til min egen, samt hjelpsomheten du har vist når jeg har trengt noe.

Jeg vil også takke min far, Claude Arne Adriaenssens, som har hjulpet meg med å bygge en prototype som ble brukt til testing. I tillegg vil jeg takke for tips og forslag du har kommet med, samt diskusjonene vi har hatt når jeg har trengt å få et problem ut av mitt eget hode.

Mandal, 12.06.2020

Halvard Yri Adriaenssens

Abstrakt

Denne avhandlingen beskriver utviklingen og testingen til et heisesystem beregnet for automatisk overvåking av innsjøer. Motivasjonen bak oppgaven er å effektivisere overvåkingen av innsjøer, slik at dyrt og krevende nattarbeid kan byttes ut med autonome fartøyer som klarer seg selv. Målingene må gjøres på nattestid da fisken ikke svømmer i stim. Kranen består av to systemer: kransystemet systemet og kamerasystemet. Kransystemet drives av en steppermotor, som er koblet til en fiskesnelle. Snora fra snella går over en trinse. Rotasjonen til trinsen detekteres ved hjelp av en IR-sensor og brukes til å bestemme måleprobens dybde, om den har truffet bunn og om den har returnert til utgangsposisjonen. Dybden logges med tids- og datostempling ved hjelp av en Real Time Clock (RTC) modul og SD-kort modul. Kamerasystemet brukes til å detektere retningen på snora, som forteller hvor proben ligger i forhold til krana. Dette er nyttig for autopiloten i båten, slik at den kan justere for drift og holde seg rett over proben når målinger gjøres, som igjen fører til at dybdemålingene blir mer nøyaktige.

Innhold

Figurer

Tabeller

Forkortelser

CV	Computer Vision
ASV	Automated Survey Vessel
RasPi	Raspberry Pi
PWM	Pulsbreddemodulasjon
fps	frames per second
RTC	Real Time Clock
UTC	Coordinated Universal Time
MEMS	Mikro elektronisk-mekanisk system
I2C	Inter Integrated Circuit
SPI	Serial Peripheral Interface
CTD	Conductivity Temperature and Depth
ppm	parts per million
LVDT	Linear Variable Differential Transformer
CPU	Central Processing Unit

1 Introduksjon

EU har pålagt Norge å undersøke alle innsjøer over en gitt størrelse for å overvåke fiske- og plantelivet. Dette ble vedtatt i "EU Water Framework Directive" i år 2000 [8].

Nå gjennomføres denne prosessen ved at forskere kjører hydroakustiske målinger over en innsjø og tar målinger med ekkolodd. Disse målingene må tas på natten, når fisken ikke svømmer i stim, slik at størrelsen på enkeltindivider kan bestemmes. Med jevne mellomrom må det stoppes for å ta vannprofilmålinger. Ekkoloddet er avhengig temperaturprofilen for å beregne avstand og posisjon til fisken korrekt. Temperaturprofilen finnes ved å ta målinger med en egen måleprobe, som senkes ned gjennom vannmassene.

Dette er ikke en optimal løsning. Nattarbeid er kostbart og krevende, og for øyeblikket har ikke Norge god nok kapasitet til å overvåke alle innsjøene de er pålagt.

Derfor jobber UiO med å utvikle et autonomt overvåkings fartøy (Automated Survey Vessel (ASV)). Målet er å utvikle et allsidig fremdriftssystem med autopilot, et system til å ta seg av data fra ekkoloddet, og et system til utsetting/innhenting av måleproben. Systemene skal kunne monteres på stort utvalg av ulike små båter. Tanken er at det autonome fartøyet settes ut på kvelden med forhåndsprogrammerte veipunkter, samt interessepunkter der fartøyet skal stoppe og gjøre vannprofilmålinger. Fartøyene samles inn på morgenen og måledata lastes ned. Dermed kan forskerne fokusere på å analysere måledata på dagen i stedet for å ta målinger hele natten.

Tidligere på UiO har det blitt gjort arbeid på fremdriftssystemet og autopiloten til fartøyet. Fokuset med denne rapporten blir å utvikle systemet som skal utsette og innhente måleproben.

Dette er en kjent problemstilling og det er flere som har gjort et forsøk på å løse den. Blant annet G. Hitz *et. al.* [9], som i 2012 publiserte "Autonomous inland water monitoring design and application of a surface vessel,". De har spesialdesignet et skrog som kan manøvrere på en innsjø ut ifra forhåndsprogrammerte koordinater. De har også en egen vinsj til å heve og senke en måleprobe, men de har designet proben slik at den henger på slep i vannet hele tiden mens fartøyet kjører. Det vil si at den ikke kan måle vannprofilen nede i dypet.

OceanAlpha sin ESM30 er et annet eksempel på en ASV, men den er designet til å ta vannprøver

i overflaten, ned til en maks dybde på 0,5 meter [10].

2 Teori

Denne seksjonen tar for seg relevante konsepter og komponenter som blir brukt senere i oppgaven. Konseptene innenfor maskinsyn skal brukes til finne vinkelen til en linje ved hjelp av maskinsyn, og vil senere bli implementert via OpenCV på en Raspberry Pi. Komponentene RTC og steppermotor er sentrale komponenter som blir brukt i kransystemet som er utviklet.

2.1 Maskinsyn

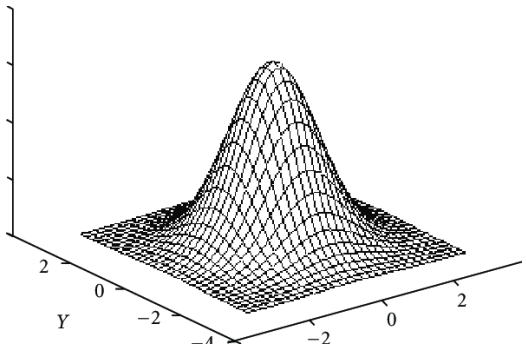
Maskinsyn er et vitenskaplig felt som tar for seg hvordan datamaskiner kan få forståelse fra digitale bilder eller videoer. Det kan brukes til å få en maskin til å utføre oppgaver og ta avgjørelser ut i fra hva den oppfatter.

2.1.1 Gauss glatting

Ett Gauss-filter eller Gauss-glatting brukes i bildebehandling til å redusere skarpheten i et bilde, samt fjerne støy og unødvendige detaljer. Formelen for et 2D gauss-filter er vist i formel 1 [11].

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

Der σ er standardavviket. En 2D Gauss fordeling med middelverdi (0,0) og $\sigma = 1$ er vist i figur 1.



Figur 1: 2D Gauss fordeling med middelverdi (0,0) og $\sigma = 1$ [1]

Det et Gauss filter gjør for å glatte ut et bilde er å ta et vektet gjennomsnitt av pikslene. De sentrale pikslene vil påvirke resultatet mer enn de som er lengre vekk. Eksempel på Gauss

filtrering er vist i figur 2. Som figuren viser ser det ut som bildet til høyre er ute av fokus. Dette gjør at støyen blir glattet over, og at vi mister unødvendig informasjon som bygningene i bakgrunnen.

(a) Original

(b) Gauss filter med $\sigma = 3$

Figur 2: Eksempel på bruk av Gauss filtrering

2.1.2 Sobelfilter

Et Sobel filter ser på endringene i et bilde i x- og y-retning for å bestemme om det er en kant eller ikke. Hver enkelt piksel vil ha en verdi som ligger mellom 0 og 255. For å sjekke om vi har en kant i bildet, brukes konvolusjonematrisene G_x og G_y vist i formel 2. A er bildet som det utføres kantdeteksjon på, og * operatoren er matriseoperasjonen kalt konvolusjon og ikke en vanlig matrisemultiplikasjon. Vi ser at G_x er symmetrisk med negative verdier på venstre siden, og positive på høyre siden. Dette er for å finne differansen i x-retning ved å plassere matrisen over hver enkelt av pikslene vi har i bildet. Det samme gjelder G_y, bare at den har negative verdier øverst, og positive nederst. Når vi bruker disse konvolusjonsmatrisene vil vi få to bilder som output, et for x-retningen og et for y-retningen. Vi kan si at Sobel filteret prøver å finne den deriverte av bildet.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A \tag{2}$$

Den første matrisen i ligning 3, representerer bildet vi ønsker å utføre kantdeteksjon på. Vi kan se ganske tydelig at det er en vertikal kant der, men en datamaskin vil ikke kunne se det samme helt uten videre. For å finne denne informasjonen kan G_x og G_y brukes på hver enkelt piksel. Et eksempel på utregning av denne verdien er vist i ligning 4, her har vi regnet ut verdien til kanten i x-retning og finner frem til at den er 200. Hadde denne verdien vært høyere hadde vi hatt en mer tydelig kant og motsatt. Hadde vi kjørt samme operasjonen for y-retning hadde vi endt opp med 0, fordi vi har ingen horisontal kant.

$$G_x = \begin{bmatrix} 50 & 50 & 100 & 100 \\ 50 & 50 & 100 & 100 \\ 50 & 50 & 100 & 100 \\ 50 & 50 & 100 & 100 \end{bmatrix} * \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} 50 & 50 & 100 \\ 50 & 50 & 100 \\ 50 & 50 & 100 \end{bmatrix} * \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} = -50 - 100 - 50 + 0 + 0 + 0 + 100 + 200 + 100 = \underline{\underline{200}} \tag{4}$$

Figur 3 viser Sobelfilter i bruk på et sudokuspill. For å finne G, som er et bilde som inneholder informasjonen fra G_x og G_y brukes $G = \sqrt{G_x^2 + G_y^2}$. Vi kan også finne gradienten til kanten ved å bruke formel 5, denne er nyttig å ha når vi skal prosessere bildet videre [12].

(a) Gråskala bilde (b) G_x, Sobel filter i x-retning (c) G_y, Sobel filter i y-retning

Figur 3: Eksempel på bruk av Sobel filter i x- og y-retning

$$\Theta = \arctan\left(\frac{G_x}{G_y}\right) \tag{5}$$

2.1.3 Canny kantdetetksjon

Canny kantdeteksjon tar outputen til et Sobelfilter og videreprosesserer det bildet til vi står igjen med kun de mest dominante kantene representert med linjer som er 1 piksel tykke.

Først bruker Canny et 5x5 Gauss-filter (2.1.1) vist i formel 6 for å fjerne unødvendige detaljer i bildet.

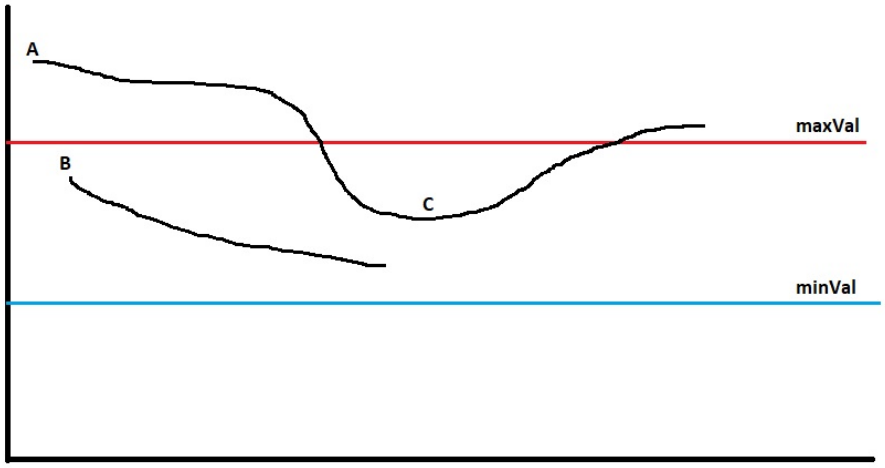
$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{6}$$

Så kjøres Sobelfilteret (2.1.2) i x- og y-retning slik at vi kan finne G og Θ.

Det Canny-filteret gjør er å fjerne unødvendig informasjon fra G, fordi vi egentlig ikke er interessert i tykke hvite linjer. Vi ønsker å ha de tykke hvite linjene representert med linjer som er 1 piksel tykke slik at vi vet nøyaktig hvor linjen går. Dette gjøres ved å finne den lokale maksimumsverdien. Hvis vi ser på gradienten til en kant fra Sobelfilteret så vil den ligne på en normalfordeling. Det Canny da gjør er å se bort ifra normalfordelingen og kun bruke middelverdien, fordi det er der kanten er sterkest. Da står vi igjen med kun de sterkeste kantene.

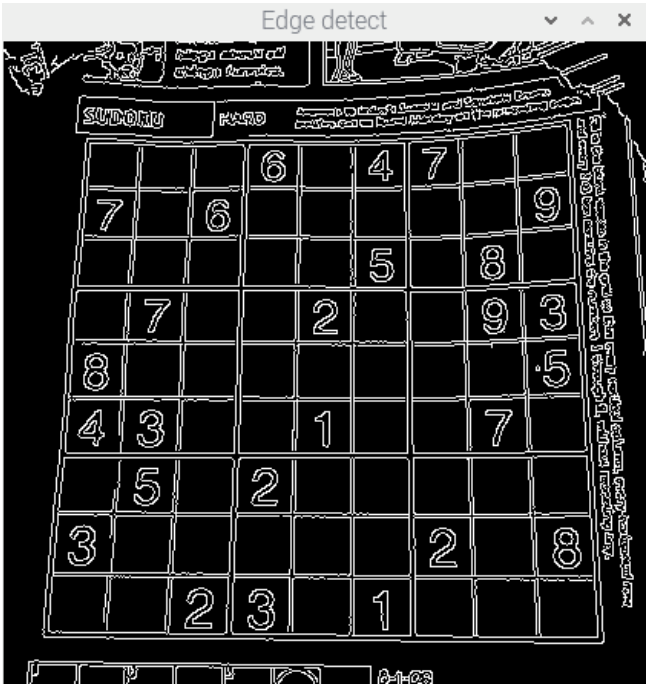
Neste steg går ut på å fjerne de kantene som er for svake, fordi om de er lokale maksimumsverdier så kan det være støy. For å gjøre dette brukes en metode som heter hysterese terskel. Alle kantene vil ha en verdi fra 0 til 255, der 0 er ingen kant og 255 er den sterkeste kanten vi kan ha. Vi må da sette en øvre og nedre terskel, minVal og maxVal. Hvis terskelen settes for lavt vil den ta med alle kantene, hvis den settes for høyt kan vi miste viktig informasjon i bildet. Alle kantene som har en verdi som er lavere enn minVal blir med en gang sett i bort fra. Vi er ute etter de kantene som har en verdi som er høyere enn maxVal. Hvis kanten har en verdi som er mellom minVal og maxVal, blir den kun tatt med hvis den er en del av en lengre kant som har verdier over maxVal. Et eksempel på dette er vist i figur 4. Strekene A, B og C er kanter representert endimensjonalt, og vi har satt bestemte verdier til minVal og maxVal. A vil bli tatt med fordi

den har en verdi over maxVal. B blir ignorert. C vil bli tatt fordi den er koblet til A og den ligger over minVal [13].



Figur 4: Canny hysterese terskel med minVal og maxVal

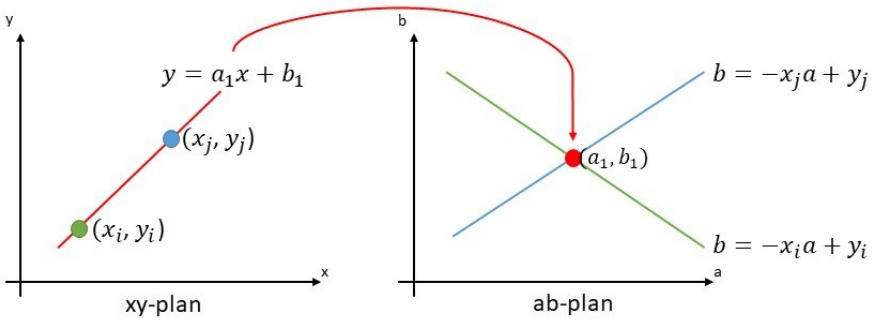
Figur 5 viser resultatet av Canny kantdeteksjon brukt på et sudoku spill. Hvis vi sammenligner dette resultatet med det vi fikk i figur 3, kan vi se at all støyen er borte, og vi sitter kun igjen med de sterkeste kantene.



Figur 5: Canny kantdeteksjon utført på sudoku spill

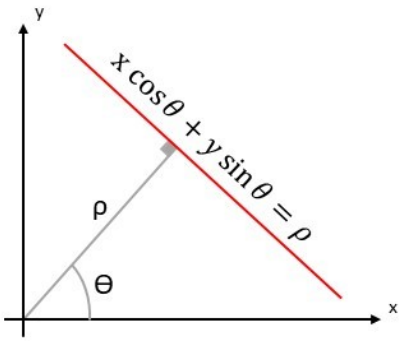
2.1.4 Hough linje transformasjon

Hough tranformasjon brukes til å finne enkle figurer eller mønstre i et bilde. I denne rapporten er vi kun interessert i linjer. Det Hough transformasjonen gjør er å bruke et stemmesystem som ser etter hvor mange punkter som tilfredsstiller kravene for en linje. Vi begynner med å skille mellom xy-planet som er bildeplanet, og ab-planet som er Hough parameter planet. Som vi ser i figur 6, kan en linje i xy-planet skrives som $y = a_1x + b_1$. Hvis vi flytter denne linjen over i ab-planet blir linjen gjort om til et punkt (a_1, b_1). Linjen i xy-planet vil bestå av mange punkter, som feks (x_i, y_i) og (x_j, y_j). Disse punktene kan og representeres i ab-planet som linjer. Det som er felles for disse linjene er at de vil krysse i samme punkt, nemlig (a_1, b_1).



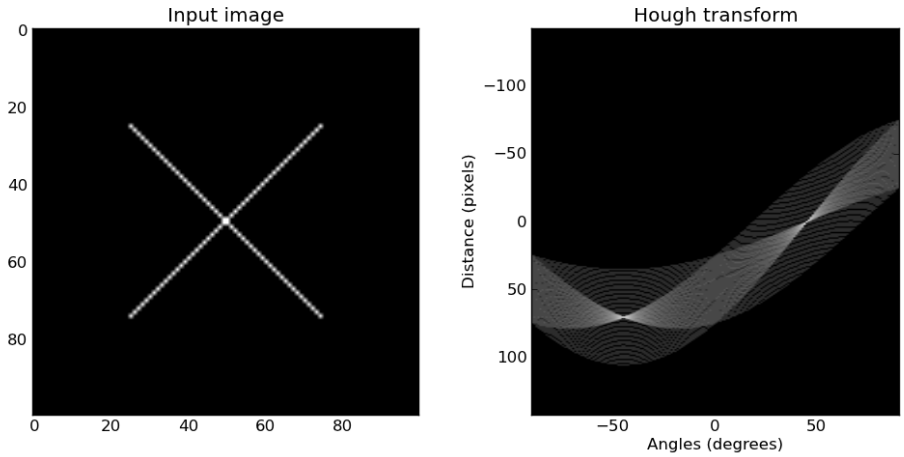
Figur 6: Sammenheng mellom bildeplanet og Hough parameter planet

Når vi bruker $y = ax + b$ til å representere linjer i bilder, kan vi få problemer hvis vi har vertikale linjer. En mer robust metode er å representere linjer med polarkoordinater. Som vist i figur 7, brukes ρ og θ i stedet for a og b. Der ρ er korteste avstanden fra origo til linjen, og θ er vinkelen til den korteste avstanden. Når vi vet x og y vil nå et punkt i bildet bli en sinuskurve i ab-planet.



Figur 7: Polar representasjon av linjer

Figur 8 viser et eksempel på en Hough transformasjon. Bildet til venstre består av to rette linjer. Hver enkelt av de hvite pikslene vil ha en tilsvarende sinuskurve. Som vi ser av den Hough transformerte, har vi to punkter der mange av sinuskurvene krysser, et for hver linje.



Figur 8: Eksempel på en Hough transformasjon

Åpen kilde biblioteket OpenCV kommer med to forskjellige implementasjoner av Hough linje transformasjon: HoughLines og HoughLinesP [14]. HoughLines fungerer som forklart over, den returnerer polarkoordinatene til en linje. HoughlinesP er en litt mer effektiv implementasjon av HoughLines, den returnerer endepunktene til de detekterte linjene $(x0, y0, x1, y1)$.

2.1.5 Parallellprogrammering

Parallellprogrammering tillater en Central Processing Unit (CPU) å kjøre to eller flere tråder samtidig. En tråd kan være et script som kjøres. Forskjellige tråder blir fordelt på ulike prosessorkjerner til en CPU. Når parallellprogrammering brukes kan flere deler av koden kjøres samtidig, slik at hele prosessen kan utføres mer effektivt.

2.2 RTC

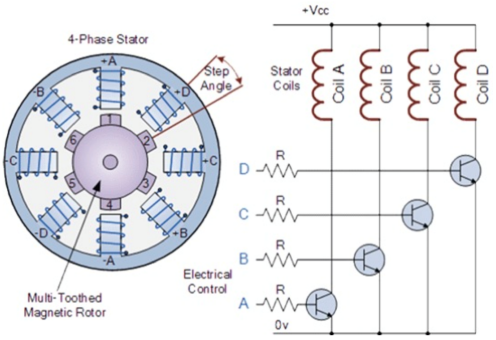
Egne Real Time Clock (RTC) kretser blir ofte brukt i datasystemer for å holde orden på tiden. Fordelen med RTC er at den frigjør prosessoren, og den er ofte mer presis enn andre metoder. RTCer har ofte en egen strømkilde (batteri) slik at de kan holde styr på tiden selv om resten av systemet er avslått. RTCer er som regel krystallstyrte, med en frekvens på 32,768 kHz. Den lave frekvensen gjør at strømforbruket også blir lavt. En billigere løsning enn å bruke

krystalloscillatorer, er å bruke en MEMS resonator. Ulempen med denne løsningen er at den er mer sensitiv for endringer i temperatur, og må temperaturkompenseres.

2.3 Steppermotor

En steppermotor virker ved å utføre individuelle små vinkeltrinn for å simulere kontinuerlig rotasjon. Antall trinn en steppermotor har, er bestemt av produsenten og påvirker nøyaktigheten til motoren. Hvis motoren kun har 4 trinn vil det si at den roterer 90° per trinn. En mer vanlig verdi er 200 trinn per rotasjon, da vil motoren rotere $\frac{360}{200} = 1,8°$ per trinn. Denne vinkelen kalles for trinns vinkelen [15].

Figur 9 viser en steppermotor med 6 tenner på rotoren, og fire fase stator. Trinns vinkelen bestemmes av hvilken spoler det settes strøm på. Tabell 1 viser at vi kan oppnå en nøyaktighet på 15° når vi bruker hele trinn med det oppsettet vi har i figur 9.



Figur 9: Forenklet steppermotor [2].

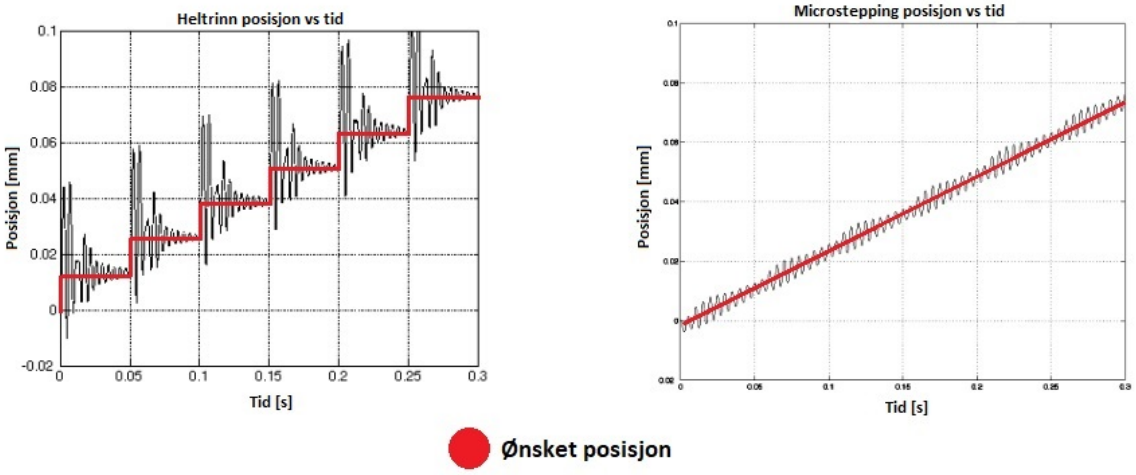
Trinnsvinkel	Spole med strøm
0°	A
15°	B
30°	C
45°	D
60°	A

Tabell 1: Vinkelen til rotoren ut ifra hvilken spole som er satt strøm på

2.3.1 Microstepping

Hvis vi nå introduserer halvtrinn kan vi doble nøyaktigheten med det samme oppsettet. Med halvtrinn så settes det strøm på to av spolene samtidig, slik at rotoren vil få en vinkel som er mellom to av spolene. Slik kan man fortsette å microsteppe ned til $\frac{1}{256}$ dels trinn, det vil si at man ender opp med 51 200 trinn per omdreining, og en nøyaktighet på $0,007°$. En av fordelene med microstepping er at motoren kan oppnå en jevnere rotasjon. Når heltrinn konfigurasjonen benyttes, spesielt i lave hastigheter, vil posisjonen oscillere, slik som vist i figur 10. Posisjonen vil også oscillere litt når microstepping brukes, men mye mindre enn når heltrinn brukes. I

praksis vil denne oscilleringen føre til vibrasjoner og lyd.



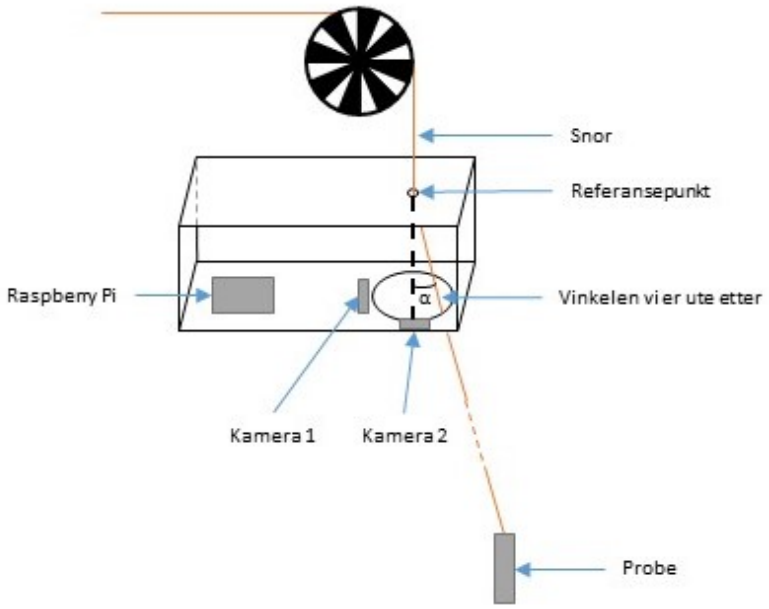
Figur 10: Heltrinn vs microstepping [3]

3 Matriell og metode

Denne seksjonene tar for seg utviklingen av systemet, hvilke konsepter som ble vurdert og hvilke løsninger som ble utforsket videre.

3.1 Maskinsyn konsept

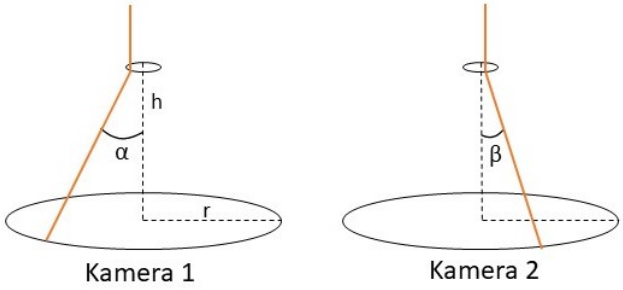
For å holde båten mest mulig i ro under målinger er det ønskelig å vite vinkelen på snora ned til sensoren. Denne informasjonen skal sendes til autopiloten slik at den kan kompansere for avdrift på båten. For å finne vinkelen til snora ble det utviklet et konsept basert på maskinsyn. Figur 11 viser et oppsett som bruker to kameraer for å finne vinkelen α.



Figur 11: Konsept for å måle snorvinkel ved hjelp av maskinsyn

Konseptet går ut på at hvert kamera klarer i finne snora og gjenkjenne den som en linje. Det er da mulig å bruke referansepunktet som utgangspunkt for å finne vinkelen mellom utgangsposisjonen (når snora henger rett ned) og nåværende posisjon. Se figur 12.

Når vi har vinkelen α og β er det mulig å bruke disse sammen med romposisjonene til referansepunktet, kamera 1 og kamera 2 for å finne snoren som en vektor i rommet. Med denne informa-



Figur 12: Finne snorvinklene med kamera 1 og kamera 2.

sjonen kan vi for eksempel fortelle autopiloten hvilken kvadrant snoren ligger, og autopiloten kan kompansere for dette. Det er naturlig å ha en radius fra senter som definerer akseptabelt avvik slik at ikke autopiloten gjør store endringer når snora bare så vidt er forskyvet fra senter.

3.1.1 Maskinvare og programvare

Det å skulle prosessere bilder fra to kameraer samtidig krever en del prosessorkraft, derfor er det tiltenkt en egen enhet for å ta seg av dette. Denne enheten vil være uavhengig av resten av systemet og vil nesten utelukkende kommunisere med autopiloten. Eneste kontakt med hovedkontrolleren vil være et eventuelt start og stopp signal. For å løse denne oppgaven er det tiltenkt å bruke en Raspberry Pi 4 Model B. Fordelen med Raspberry Pi (RasPi) er at det er mye brukt plattform med god support i samfunnet på nett. Samtidig er det en plattform som tilbyr mye kraft for pengene. RasPi er tilpasset OpenCV, som er det mest brukte åpen kilde maskinsyn biblioteket som er tilgjengelig. OpenCV har et grensesnitt for C++, Python, Java og MATLAB. I denne oppgaven er Python brukt. Det er tidligere blitt utført prosjekter som bruker to kameraer, en RasPi og OpenCV til å utføre bevegelsesdeteksjon på begge kameraene samtidig [16].

3.1.2 Alternative metoder for å måle snorvinkelen

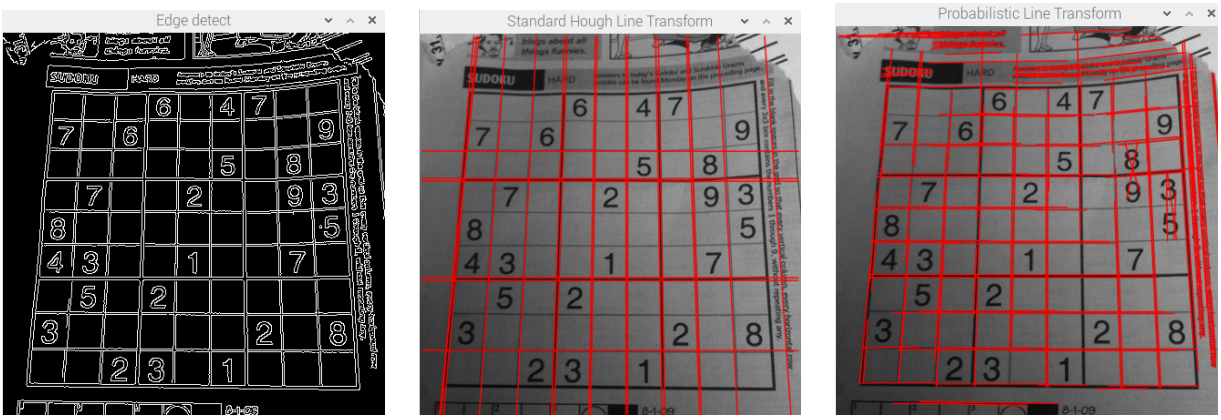
Alternative metoder kan bestå av laser avstandsmåler som glimrer i horisontalplaner og måler vinkel og av stand til snora. Hvis refleksjonen da måles går det an å si noe om vinkelen og avstanden til snora. Ellers kunne det vært mulig å bruke en Linear Variable Differential Transformer (LVDT) som er en mer mekanisk løsning. Snoren går igjennom et bevegelig punkt som er koblet til en LVDT og det da er mulig å bestemme posisjonen til snora. Denne løsningen består av bevegelige deler som kan komme i kontakt med vann. Jeg ønsket heller å jobbe med

kameraer og maskinsyn så derfor gikk jeg for den løsningen.

3.2 Utvikling av linjedeteksjon

3.2.1 Hvordan finne linjer i et bilde

For å finne en linje i bildet brukes teknikkene beskrevet i seksjon 2.1, der bildet først blir glattet med et Gauss filter. Etterpå brukes Sobel filter for å finne gradienten i bildet. Deretter brukes Canny kantdeteksjon, for å få alle kantene representert som 1 piksel tykke streker. Vi har nå bildet som vist i figur 13a. For å finne linjene brukes både HoughLines og HoughLinesP. HoughLines kjøres med en terskelverdi på 200, dvs at en linje må da ha 200 stemmer for å bli oppfattet som en linje. Utgangsverdiene fra HoughLines og HoughLinesP er de røde strekene vist i figur 13b og 13c. De røde strekene er lagt oppå gråskala bildet for å se at de er plottet riktig. HoughLines blir brukt videre i oppgaven, fordi den krever færre input variabler og var lettere å bruke enn HoughLinesP.



(a) Canny kantdeteksjon (b) HoughLines (c) HougLinesP

Figur 13: HoughLines og HoughLinesP brukt på et bilde

3.2.2 Hvordan finne vinkelen til en linje i en video

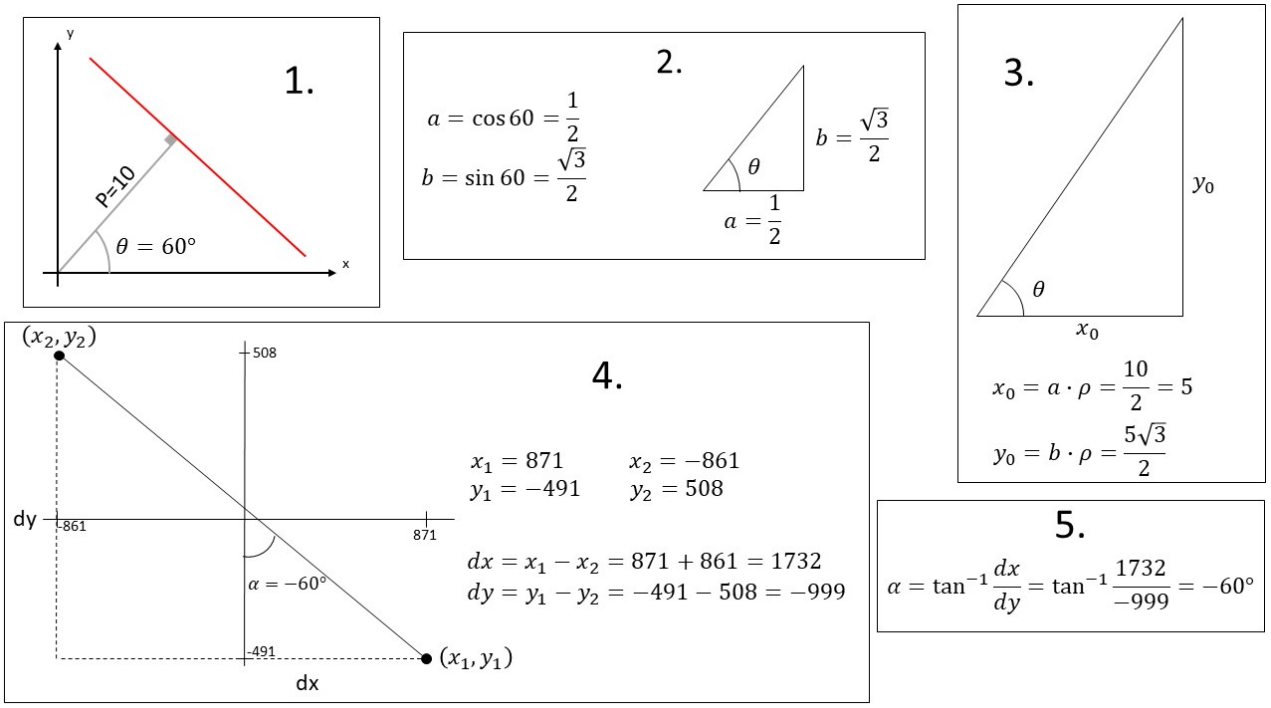
Neste steg går ut på å bytte stillbildet med en live video. Dette gjøres ved å bruke VideoCapture som leser et bilde ifra kameraet og lagrer det i en variabel "cap". Fremgangsmåtene for å finne en linje er helt lik som i seksjonen 3.2.1. Den største forskjellen er at hele prosessen plasseres inne i en while-loop som bruker cap.read() i starten av hver loop for å lese av kameraet på nytt. Nå er ikke alle linjene interessante lenger, kun den sterkeste linjen er viktig. Så i stedet

for å lete etter alle linjene som tilfredsstiller et visst minstekrav, så er kun den linjen med flest stemmer av interesse. Deretter kan trigonometri brukes for å finne vinkelen α. Vi får ut (ρ, θ) fra HoughLines. Problemet med å bruke θ er at den kun går fra 0 til 180 grader, og ikke negativt. Målet er å lese vinkelen α ut ifra y-aksen, med positiv rotasjon mot venstre. Fremgangsmåten for å finne vinkelen α er vist i formel 7.

$$
\begin{aligned}
a &= \cos\theta \\
b &= \sin\theta \\
x_0 &= a \cdot \rho \\
y_0 &= b \cdot \rho \\
x_1 &= x_0 + 1000 \cdot (-b) \\
y_1 &= y_0 + 1000 \cdot (a) \\
x_2 &= x_0 - 1000 \cdot (-b) \\
y_2 &= y_0 - 1000 \cdot (a) \\
dx &= x_1 - x_2 \\
dy &= y_1 - y_2 \\
\alpha &= \arctan\left(\frac{dx}{dy}\right) \cdot \frac{360}{2\pi}
\end{aligned}
\tag{7}
$$

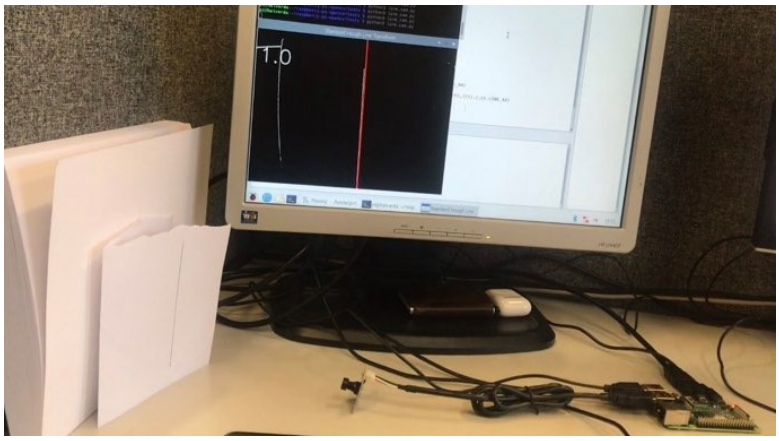
Et talleksempel med figurer er vist i figur 14. Utregningen består av 5 steg:

- Steg 1 er å lese av θ og ρ verdien til linjen.
- Steg 2 er å finne a og b, som skal brukes videre i steg 3 og 4.
- Steg 3 går ut på å finne (x_0, y_0), som er punktet der normalvektoren til linjen og linjen krysser. Hovedsakelig brukes x_0 og y_0 til å sørge for å unngå tilfeller der det blir delt på 0 i nevneren.
- Steg 4 går ut på å finne endepunktene til linjen vi er interessert i. Grunnen til at det ganges med 1000 når vi finner (x_1, y_1) og (x_2, y_2), er for å sørge for at punktene havner på utsiden av bildet med god margin. Når da linjen tegnes vil den krysse hele bildet. dx og dy regnes ut ifra disse punktene.
- Steg 5 bruker dx og dy til å regne ut α, som er vinkelen mellom linjen og y-aksen.



Figur 14: Eksempen på utregning av vinkel til linje

Oppsett med ett kamera er vist i figur 15. Det ble brukt et ark med en svart strek på til å simulere snora og en hvit perm ble brukt som bakgrunn for å forhindre støy. Ved å rotere på arket, vil bildet på skjermen med den røde streken og gradene i venstre hjørne, oppdatere seg i sanntid. Kameraet som blir brukt her har fiskeøyelinse, som vil gjøre at rette linjer vil bli oppfattet som mer buet når de beveger seg bort ifra senter til bildet. Dette må det tas hensyn til, fordi det er mer sannsynlig at rette linjer som er sentralt i bildet blir oppfattet som sterkest.



Figur 15: Linjedeteksjon med et kamera

3.2.3 Bestemmme bilder per sekund (fps)

Under utvikling og testing av linjedeteksjon med et kamera, ble RasPi prosessoren fort varm. Dette er med kun ett kamera, og målet er å bruke to. For å lette på arbeidsmengden til prosessoren ble det lagt inn en frames per second (fps) begrensning. Dette ble gjort ved å legge inn en tidsforsinkelse, slik at kameraet kun ble lest av når en oppdatering var ønsket. Nå kan programvaren bestemme hvor mange bilder per sekund som skal prosesseres. Antallet kan justeres i fra så mange fps prosessoren klarer, til så lavt man kan ønske seg. Målet for oppgaven krever ikke en høy fps. Hensikten med å detektere snorens vinkel er å hindre at båten driver, dermed burde 1 fps være tilstrekkelig.

3.2.4 Økt effektivitet i prossesorsystemet

For få til en raskere databehandling ble parallellprogrammering implementert. Dette ble gjort ved å følge guiden til Adrian Rosenbrock: "Increasing webcam FPS with Python and OpenCV" [17]. OpenCV sin funksjon cv2.VideoCapture() ble byttet ut og erstattet med funksjonen Videostream fra imultis biblioteket. Målet med dette er å dele prosessen opp i to. Problemet med cv2.VideoCapture er at den blokker prosessen, scriptet vil ikke gå videre før den har hentet et nytt bilde ifra kameraet. Dette kan løses ved å lage en egen prosess som kun skal fikse I/O oppgaver, altså henter den bilder fra kameraet og gjør det klart for hovedtråden. Videre vil hovedtråen stå for videoprosessering. Vi har da en prosessorkjerne som kun jobber med bildeprosessering som bare ber om et nytt bilde når den er ferdig med å prosessere det forrige. Den andre kjernen har da allerede hentet et bilde ifra kameraet og har det klar til levering. Hvis ikke parallellprogrammering hadde blitt brukt, ville den ene kjernen måtte hente et bilde fra kameraet, prosessere det bildet for å så hente ett nytt bilde. For å teste hvor stor forbedring dette faktisk førte til, ble programmet fps_demo.py kjørt. Dette er et eget program som kun tester antall bilder Raspberryen klarer å prosessere. Det ble kun brukt til testing, men selve parallellprogrammeringen ble også implementert i min programvare. Figur 16 viser at ved bruk av parallellprogrammering ble 165 bilder i sekundet prosessert. Uten dette, klarte systemet kun 20. Kameraet som blir brukt er begrenset til 30 fps, så med parallellprogrammering vil samme bildet blir prosessert flere ganger. Antall fps vil øke med 50%, men den store forbedringen kommer av at forsinkelser i systemet blir redusert kraftig.

I tillegg til at parallellprogrammering ble implementert i programvaren, ble bildene skalert ned

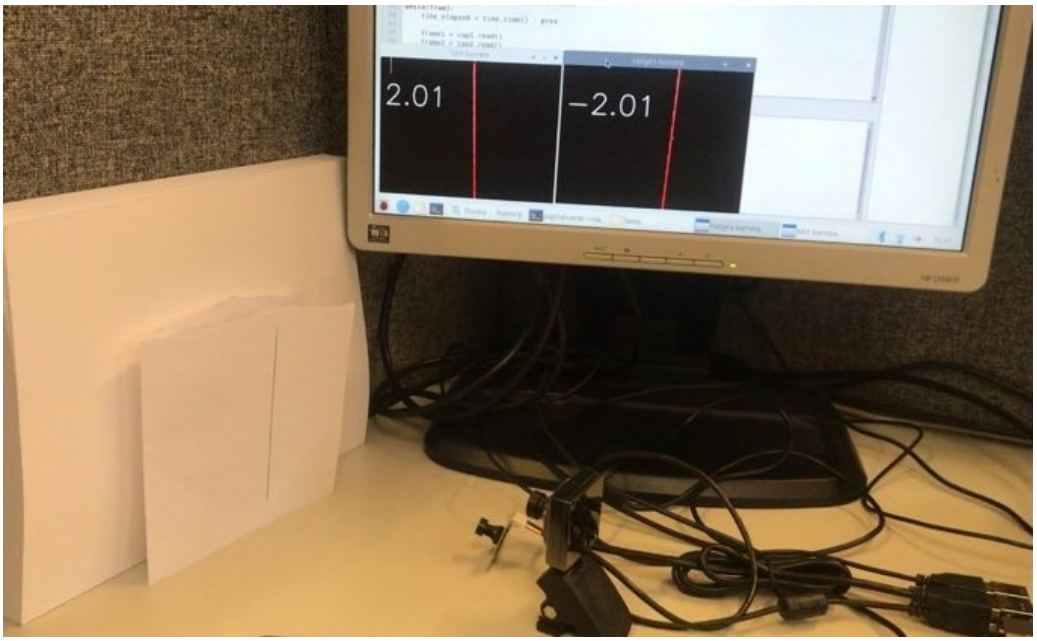
```
pi@halvarda:~/raspberry-pi-opencv/tests $ python3 fps_demo.py
[INFO] sampling frames from webcam...
[INFO] elasped time: 4.77
[INFO] approx. FPS: 20.96
[INFO] sampling THREADED frames from webcam...
[INFO] elasped time: 0.60
[INFO] approx. FPS: 165.95
```

Figur 16: Effekten av parallellprogrammering på kamerasystemet

til 400x400, for å minske antall piksler som ble behandlet.

3.2.5 Implementering av to kameraer

Et ekstra kamera ble koblet til og satt opp likt som det første kameraet. Oppsettet ser ut som vist i figur 17. Systemet er kapabel til å måle vinkelen, fra to sider, til den mest dominante linjen på bildet gitt en bestemt fps.



Figur 17: Linjedeteksjonssystemet med to kameraer implementert

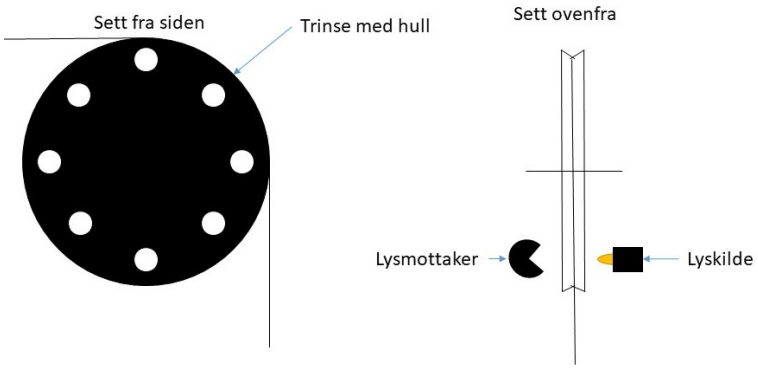
Det bakerste kameraet som er nærmest i figur 17 ble kun brukt under utvikling. Senere ble det byttet ut med et Raspberry Pi kamera. Raspberry Pi Model 4 kommer kun med en dedikert kameraport til kameraene de selv produserer. Disse kobles sammen ved hjelp av en båndkabel. For å klare to kameraer må derfor et USB-kamera også brukes.

3.3 Dybdemåling

Kravene som stiltes til snorsystemet, er at det skal ha konstant hastighet, som gjerne ligger på ca. 0,5 m/s. I tillegg er det ønskelig å vite nøyaktig hvor mye snor som er ute og hvor dypt proben befinner seg. Det er vanskelig å beregne en hastighet og dybde nøyaktig på grunn av endring i radiusen på trommelen. For et mer nøyaktig anslag av dybden brukes en trinse, som snoren går over. Tanken er at snoren har nok friksjon slik at den ikke vil gli. Da er det mulig å beregne dybden nøyaktig ut ifra rotasjonen til trinsen. De neste tre avsnittene beskriver løsningene som ble vurdert for å måle rotasjonen til trinsen.

3.3.1 Måling av rotasjon ved hjelp av lys

For å måle rotasjonen ble det vurdert flere mulige løsninger. Først ble en løsning med en lyskilde og lyssensor vurdert. En skisse av ideen er vist i figur 18. Trinsen vil ha x-antall hull som lyskilden kan lyse igjennom. Lysmottakeren vil da gi en puls for hver gang den mottar lys. Dette kan brukes til å regne ut hvor langt trinsen har rotert og da hvor mye snor som er ute. Fordelen med denne metoden er at det er ingen bevegelige deler, med unntak av trinsen.



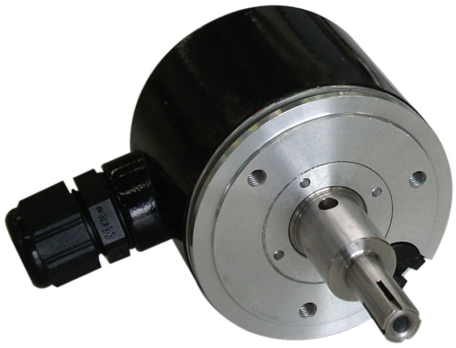
Figur 18: Mulig metode for å måle rotasjon ved hjelp av lys

Nøyaktigheten kan bestemmes ut ifra antall hull og trinsens diameter. For eksempel en trinse med 16 hull og en diameter på 10 cm vil gi en nøyaktighet på ca. 2 cm, se ligning 8.

$$\frac{\pi \cdot d}{Antall\ hull} = \frac{\pi \cdot 10cm}{16} \approx 1,96cm \tag{8}$$

3.3.2 Måling av rotasjon ved hjelp av rotasjonsmåler

En rotasjonsmåler (rotary enkoder) er en komponent som gir ut en puls når den roterer og kan se ut som figur 19. I mitt system ville det blitt brukt enten belte eller tannhjul til å få akselen til å rotere i takt med trinsen. Fordelen med denne løsningen er at måleren gir to signaler som er 90° faseforskjøvet, dette kan brukes til å bestemme hvilken vei trinsen roterer. En vanlig rotasjonsmåler gir ut 20 pulser per rotasjon noe som hadde vært godt nok. Dette fører til flere bevegelige deler som kan være en ulempe, spesielt når systemet skal brukes ute der vær og vind er en faktor.



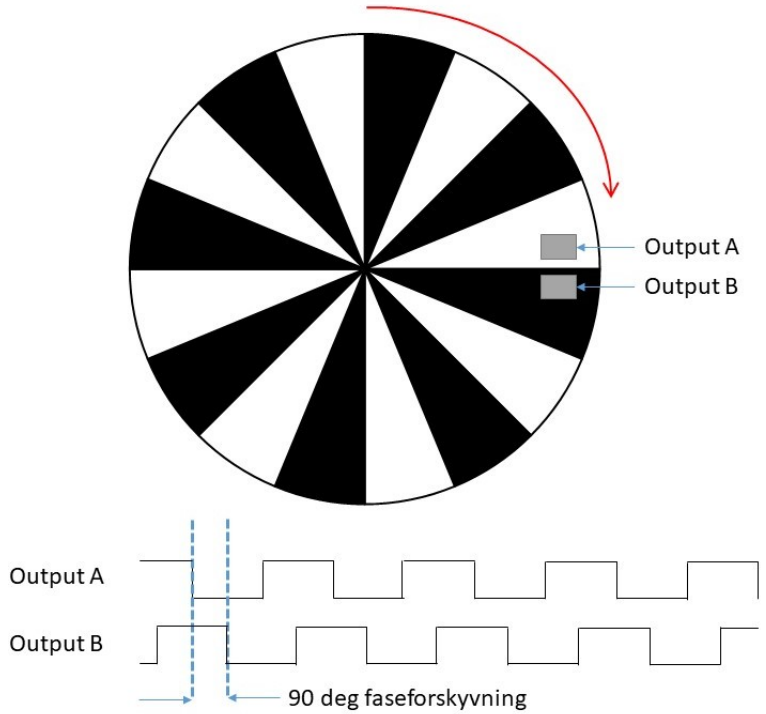
Figur 19: Eksempel på rotasjonsmåler [4]

3.3.3 Inkrementell enkoder ved hjelp av fotodetektor

En IR-sensor/fotodetektor er også et alternativ. Den er utstyrt med en IR avsender og mottaker. Ved hjelp av disse kan sensoren skille mellom svart og hvit, fordi en hvit bakgrunn vil reflektere IR signalet, mens den en svart bakgrunn vil absorbere signalet. Ved å bruke to IR-sensorer er det også mulig å bestemme rotasjonen til trinsen. Oppsettet kan da se ut som vist i figur 20. Dette systemet har minimalt med bevegelige deler, samtidig som det er mer robust når det kommer til påvirkning av eksternt lys. Men, siden det er planlagt å bruke en mikrokontroller til å lese rotasjonen og styre motoren, holder det med en IR-sensor. Fordi mikrokontrolleren vet hvilken retning motoren roterer og når den skifter retning. Denne løsningen ble til slutt foretrukket fordi den ikke har noen bevegelige deler, og det er kun en komponent som må monteres.

3.3.4 Snordrag

Informasjonen om snordraget skal brukes av systemet for å bestemme om proben treffer bunnen eller andre hindringer, og da kan avbryte målingen. Det må gjøres en vurdering om det holder



Figur 20: Inkrementell enkoder med to fotodioder

å kun vite når snora er stram og slakk, eller om det er interessant å vite snordraget presist. Hvis det holder å vite om proben har truffet bunn eller er heist opp, er det kun nødvendig å overvåke rotasjonen til trinsen. Fordi trinsen vil slutte å rotere når proben treffer bunn, da snora ikke har nok kraft på egenhånd til å rotere trinsen.

3.3.5 Trinse

Trinsen som ble valgt er vist i figur 21. Dimensjonene er presentert i tabell 2.

Figur 21: Trinsen som blir brukt i prosjektet

Diameter	90 mm
Bredde	11,5 mm
Dybden til sporet	5,8 mm
Bredden til sporet	6,2 mm
Diameteren til sporet	78,4 mm
Bolt	M8
Lengden til bolten	12 mm

Tabell 2: Dimensjonene til trinsen

Fordelen med denne trinsen er at den er utstyrt med en M8 bolt, slik at den enkelt kan monteres ved hjelp av et vinkelbeslag og en M8 låsemutter. Den er også utstyrt med 12 felter. Annethvert felt males hvit slik at de kan reflektere et IR signal. Dette gir en presisjon på ca. 20,525 mm (se ligning 9).

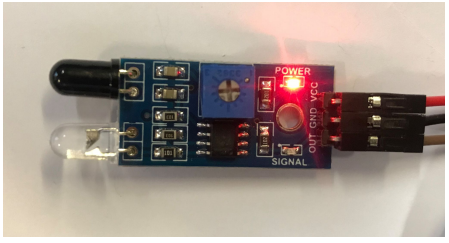
$$\frac{Omkrets}{Antall\ felt} = \frac{\pi \cdot D}{12} = \frac{\pi \cdot 78,4mm}{12} \approx 20,525mm \tag{9}$$

3.3.6 Valg av mikrokontroller til snorsystemet

Mikrokontrolleren som skal ta seg av snorsystemet har hovedsakelig to oppgaver. Den ene er å lese outputen fra IR-sensoren og gjøre det om til dybden som proben befinner seg på. Denne dybden må logges sammen med tidspunkt og enten lagres på en minnebrikke, eller overføres til PC-en ombord. Den andre oppgaven er å bestemme rotasjonen til motoren. Motoren skal skifte retning enten når proben treffer bunnen og snordraget blir mindre, eller når den har nådd den forhåndsbestemte dybden. For å løse dette ble det valgt en Arduino Mega. Jeg har erfaring med kontrolleren fra tidligere og det er en generelt mye brukt plattform med god support på nett.

3.3.7 IR-sensor

En IR-sensor ble brukt for å detektere når forskjellen mellom svarte og hvite felt på trinsen. Sensoren som ble valgt er laget av OSOYOO og brukes av smartbilder for å detektere hindringer slik at de kan unngås. Sensoren har et måleområde som er oppgitt til å være 2 cm - 30 cm, men en rask test viste at avstander på mindre enn 2 cm var ikke noe problem. Sensoren er utstyrt med to LED indikatorer og et potensiometer. LED indikatorene brukes til å indikere om strømen står på, og om IR-mottakeren leser av et signal. Potensiometeret brukes til å justere følsomheten. Når følsomheten økes kan sensoren detektere objekter over en lengre avstand. Blide av sensoren er vist i figur 22, oppkoblingen er vist i tabell 3.



Figur 22: IR-sensoren som blir brukt i prosjektet

IR-sensor	Arduino
OUT	8
VCC	3,3 V
GND	GND

Tabell 3: Oppkobling av IR-sensor

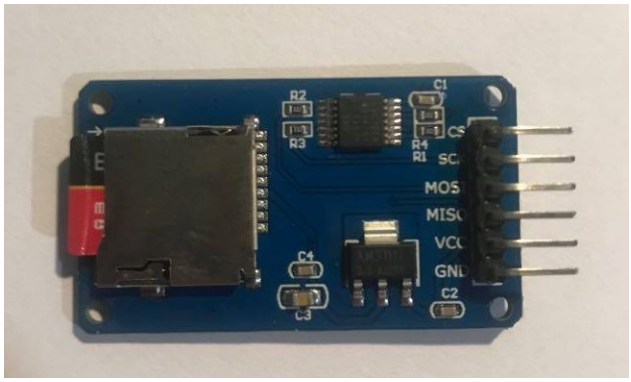
Utgangssignalet, merket OUT i tabellen over, fra sensoren leses av Arduinoen. Hver gang det leses av en stigende- eller synkende-flanke, inkrementeres eller dekrementeres telleren, som holder styr på den totale lengden, som trinsen har rotert. Retningen, som motoren roterer, bestemmer om lengde skal legges til eller trekkes fra.

3.3.8 Kompensere for båtens høyde

Siden dette systemet skal være mulig å bruke på et bredt utvalg av båter, vil avstanden fra oppheist posisjon til vannet variere med båttype. Dette bør det kompenseres for. Den enkleste løsningen, er å måle avstanden med et målebånd og legge den til som en konstant i programmet.

3.3.9 Datainnsamling

For å logge dybdedataen ble det innkjøpt en SD-kort modul, slik at Arduinoen kan lese og skrive til et SD-kort. Denne kommuniserer via Serial Peripheral Interface (SPI). Oppkoblingen til modulen er vist i tabell 4. Arduino IDE kommer med et bibliotek for å lese og skrive til SD-kort, dette ble benyttet for å logge data.



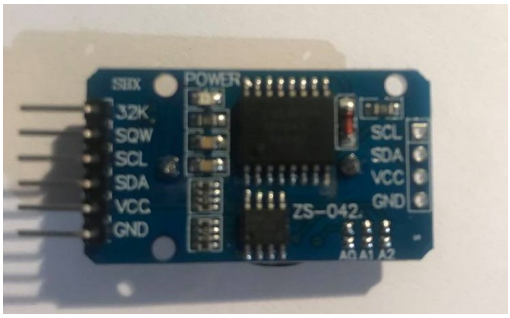
Figur 23: SD kort modulen som blir brukt i prosjektet

SD kort modul	Arduino
VCC	5V
GND	GND
MISO	50
MOSI	51
SCK	52
CS	52

Tabell 4: Oppkobling av SD kort modulen

3.3.10 RTC - Real time clock

For å kunne holde kontroll på tiden uten at Arduinoen er tilkoblet strøm, ble det brukt en RTC av typen DS3231. Denne er utstyrt med et eget batteri slik at den kan holde kontroll på tiden uten ekstern tilkobling til strøm. DS3231 kommuniserer via Inter Integrated Circuit (I2C). Oppkobling av RTC modulen er vist i tabell 5.



Figur 24: RTC modulen som ble brukt i prosjektet

DS3231	Arduino
VCC	5V
GND	GND
SDA	20
SCL	21

Tabell 5: RTC modulen som blir brukt i prosjektet

DS3231 biblioteket til Henning Karlsen ble brukt [18]. Første gang en RTC modul brukes må tiden settes manuelt, dette ble gjort via eksempelprogrammet DS3231_Serial_Easy. Da bestemmes ukedag, dato, og klokkeslett. Det som kan være litt vanskelig, er å få klokka til å bli mer nøyaktig enn ± 1 sekund, på grunn av kompileringstiden. Hvis klokkeslettet settes til 12:00:00, må kompileringen av programmet starte rundt 11:59:55, slik at den er ferdig kompilert til riktig tid. DS3231 brukes til å tidsstemple alle dybdemålingene, samt datostemple målesettet.

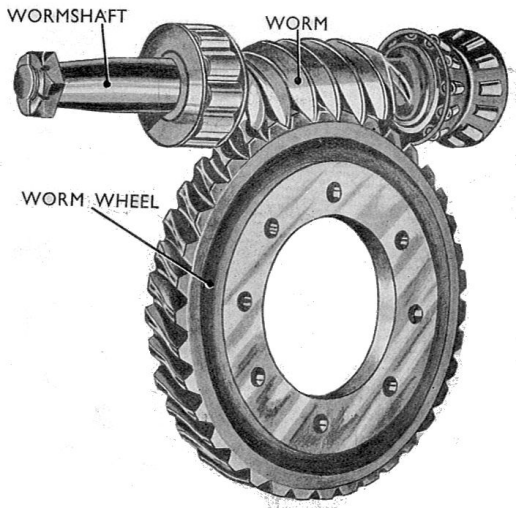
3.4 Steppermotor

3.4.1 Kriterier for valg av motor

For å kunne bruke kranen ute i feltet er det ønskelig å bruke en elektrisk motor, slik at strømforsyningen som forsyner elektronikken også kan brukes til å forsyne båten. Det er en fordel å ha en mekanisme som kan låse motoren når proben er heiset opp, slik at motoren ikke trenger å trekke strøm unødvendig.

3.4.2 Steppermotor eller ormgir

Det ble vurdert to forskjellige løsninger til dette problemet. Den første løsningen var å bruke et ormgir (worm gear). Det er satt sammen av en ormskrue og et ormhjul, se figur 25. Motoren kobles på ormskruen og ormhjulet festes til trommelen. Dette fører til at motoren kan påvirke trommelen, men ikke omvendt. Det finnes DC motorer som kommer med integrert ormgir. Som et vanlig gir vil også ormgiret ofre rotasjonshastighet for høyere dreiemoment.



Figur 25: Ormgir bestående av ormskrue og ormgir [5]

Ulempen med et ormgir er at det blir mye friksjon, som fører til redusert effektivitet, varme og slitasje. Det er vanlig å bruke en ormskrue i hardere materiale enn ormhjulet, slik at kun ormhjulet blir slitt, siden det er enklere å erstatte.

Den andre løsningen som ble vurdert var en steppermotor. Steppermotorer er mye brukt i mange applikasjoner, for eksempel 3D printere, fordi de er presise og enkle å styre. Men de har og to gunstige egenskaper som gjør dem interessante i mitt tilfelle. Det er tiltrekkingsmoment

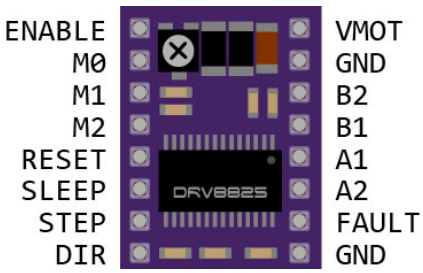
(detent torque) og holdemoment (holding torque). Begge disse er momenter som oppstår når motoren står i ro. Forskjellen er at holdemomentet oppstår når motoren er koblet til strøm, og tiltrekningsmomentet oppstår når strømmen ikke er koblet til. Holdemomentet er momentet som produsenten oppgir for å si noe om hvor sterk motoren er. Tiltrekningsmoment er på ca. 5-20% av holdemomentet, og eksisterer kun når vi har en steppermotor av typen permanent magnet eller hybrid, ikke når vi har en motor basert på variabel motstand. Fordi magneten inni motoren er den som skaper tiltrekningsmomentet. Tanken min er at tiltrekningsmomentet er stort nok til å holde proben i oppheist posisjon. Hvis ikke det er tilstrekkelig, finnes det et triks som gjør at det momentet blir høyere uten å trekke noe strøm. Når strømmen er koblet av, kan terminalene på motoren kortsluttes ved hjelp av et relé. Dette gjør at det kreves mer krefter for å rotere motoren. Forklaringen på dette er som følger: vanligvis når en steppermotor kjøres, tilføres det strøm for å få akselen til å rotere. Når motoren ikke er tilkoblet strøm, og akselen roteres ved hjelp av ytre krefter, vil hele motoren fungere som en generator. Når da terminalene på motoren er kortsluttet har ikke strømmen som genereres noen steder å gå. Da vil strømmen brennes bort som friksjon i viklingene. Dermed vil derfor kreves mer krefter for å bevege på rotoren.

Det ble valgt en NMEA 17 bipolar steppermotor med et holdemoment på 59 Ncm. Motoren kan tåler opp mot 2A og 12-24V. En bipolar motor er foretrukket over en unipolar, fordi de er mer effektive og produserer høyere moment.

3.4.3 Motorkontroller

For å kontrollere en steppermotor brukes en motorkontroller, den brukes som et mellomledd mellom mikrokontrolleren, strømforsyningen og motoren. Mikrokontrolleren klarer ikke alene å supplere motoren men nok strøm og spenning. Men den kan styre motorkontrolleren ved hjelp av et signal, slik at motoren kan flytte en mye større last.

Motordriveren som ble valgt er en DRV8825 som tåler opp mot 2A og 45V. Den har innebygde sikkerhetsfunksjoner mot for mye strøm, for høy temperatur og kortslutning. En oversikt over pinout, er vist i figur 26a. Driveren kan mikrosteppes ned til $\frac{1}{32}$dels trinn. Når motoren som er valgt har 200 trinn i utgangspunktet, vil en slik microstepping tilsvare 6400 trinn per omdreining. Hele microstepping tabellen er vist i figur 26b.



(a) Pinout til DRV8825

M0	M1	M2	Microstep resolution
Low	Low	Low	Full step
High	Low	Low	1/2 step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

(b) DRV8825 microstepping tabell

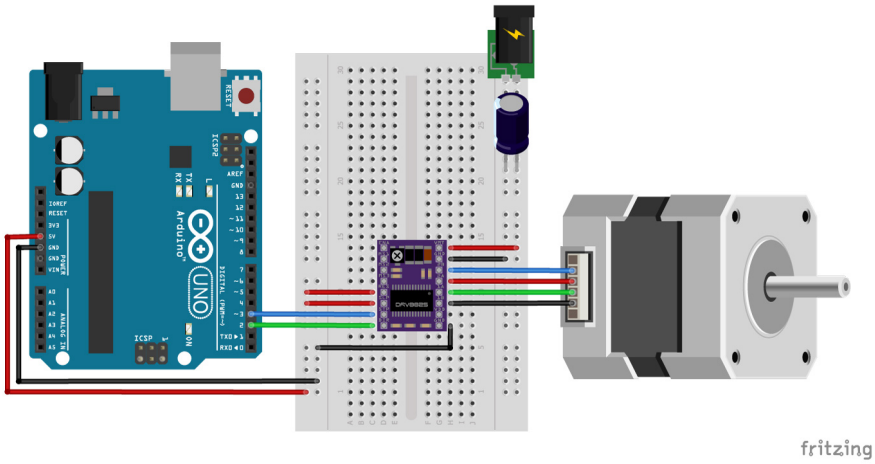
Figur 26: DRV8825 pinout og microstepping tabell [6]

3.4.4 Strømforsyning

Til testing er det blitt brukt en laptop lader som strømforsyning. Den kan tilføre 19V og 3,5A.

3.4.5 Oppkobling av steppermotor

Figur 27 viser oppkoblingen av steppermotor, motorkontroller og Arduino når motoren kjøres i fulltrinnmodus.



Figur 27: Oppkobling av steppermotor, motorkontroller og Arduino [6]

Det er spesielt to ting som er viktig å merke seg med denne figuren. Den første er at Arduinoen kan styre motoren ved hjelp av kun 2 koblinger, STEP og DIR. STEP styrer antall trinn i sekundet og DIR bestemmer retningen på rotasjonen. Den andre er at det er lagt inn en kondensator mellom VMOT og GND for å beskytte driveren mot ødeleggende LC spenningspikes. Når en elektronisk krets startes opp eller kobles til strøm, vil spenningen oscillere i et par mil-

lisekunder. Fordi om strømforsyningen sier at den er på 19 V, kan spenningstoppene komme opp i mye høyere spenning [19]. Disse toppene kan skade kretsen som kun tåler spenninger opp til 24V. Derfor brukes en kondensator på 100 uF.

3.4.6 Hvordan sette strømgrense for motorkontrolleren

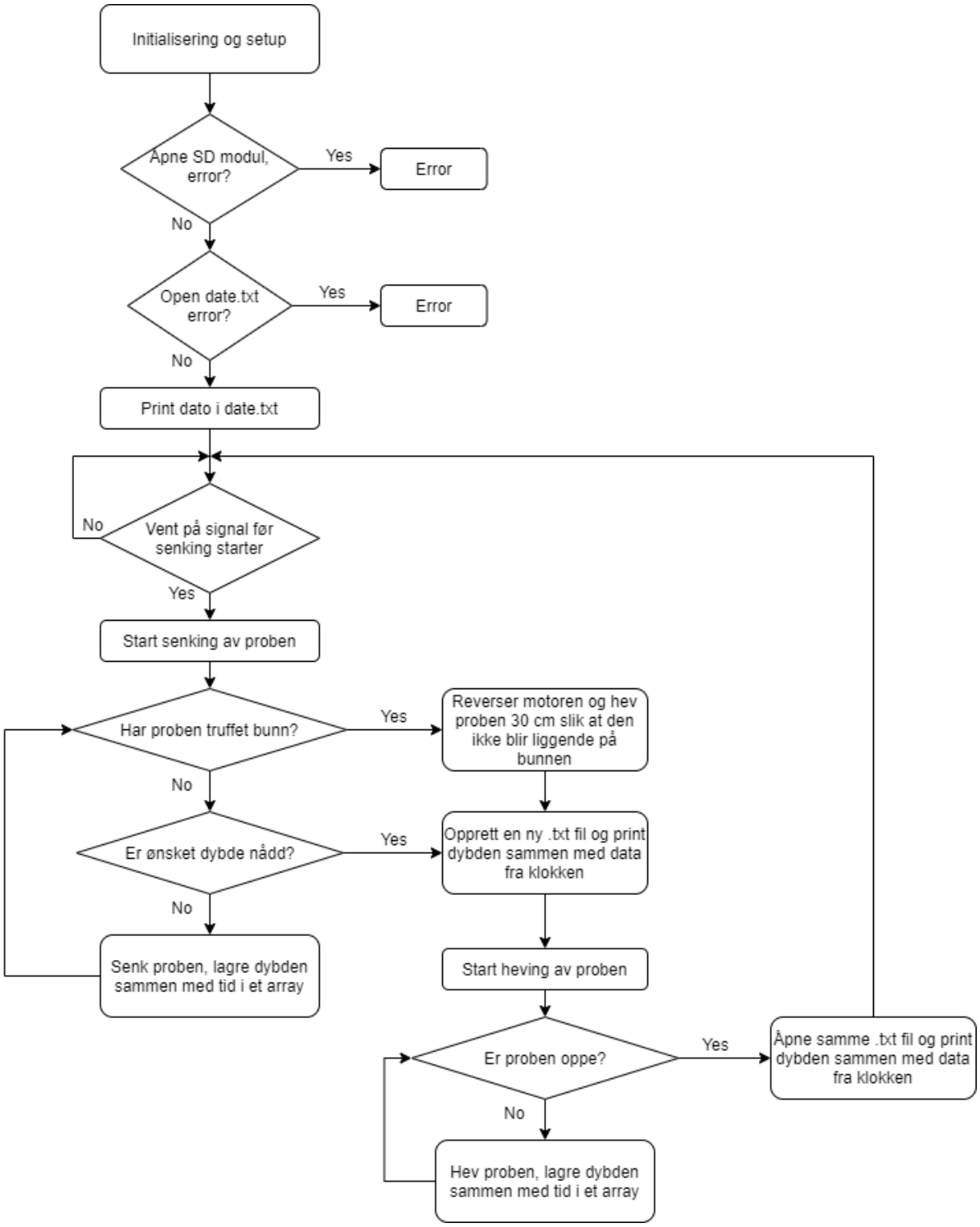
Før steppermotoren kan programmeres er det viktig å sette strømgrensen til driveren. For å gjøre dette brukes samme krets som vist i figur 26 minus motoren. Strømforsyningen og Arduinoen må være påskrudd for at motordriveren skal starte opp. Ved hjelp av et multimeter kan en referansespenning måles mellom potensiometeret, som er plassert øverst til venstre i figur 26a, og GND. Strømgrensen er lik 2 x VREF. Ved å justere potensiometeret til VREF = 1V, blir strømgrensen satt til 2A som er det maksimale driveren kan takle.

3.4.7 Programmering av steppermotoren

For å programmere steppermotoren har Arduino-biblioteket AccelStepper skrevet av Mike McCauley blitt benyttet [20]. Hovedsakelig har motoren blitt brukt til å holde en konstant hastighet, helt frem til dybdemålingen forteller motoren noe annet. Ved å kun endre på en variabel kan både hastighet og retning bestemmes.

3.5 Flytskjema til kransystemet

Flytskjema til kransystemet er vist i figur 28. Implementasjon til kode er gjort i Arduino IDE. Kildekoden er listet i vedlegg A.1.



Figur 28: Flytskjema til kransystemet

3.6 Konstruksjon

3.6.1 Kriterier

For å lettere kunne utvikle og teste kranen ble det bygget en prototype i tre. Målet med prototypen var å bygge et rammeverk som gjorde det lett å eksperimentere med plassering av sensorene og teste ut prinsippene. Ideen bak designet er at kranen skal kunne brukes over en liten kant. Den er ikke designet for en spesiell båt, men for å være allsidig.

Selve konstruksjonen består av en bunnplate, to sideplater, en endeplate og en ekstra etasje. Se figur 29. Formålet med å ha to etasjer er for å skille elektronikken ifra snelle og motor.

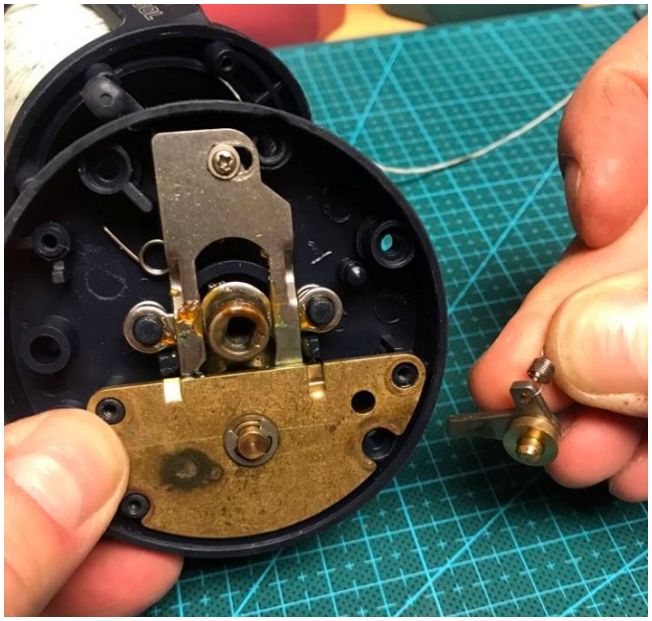
(a) Kranen sett ifra siden

(b) Kranen er utstyrt med to etasjer

Figur 29: Oversiktsbilder av kranen

3.6.2 Trommel

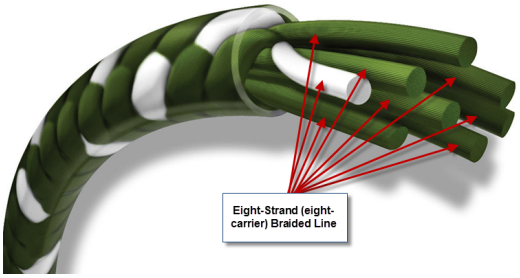
De viktigste kriteriene når det kommer til valg av trommel er at den må tåle diverse vær, vind og vann, fordi den skal brukes ute på innsjøer. En fiskesnelle er da et naturlig valg. En fiskesnelle har og den egenskapen at snoren fordeler seg jevnt utover trommelen. Dette gjør igjen at diameteren blir mer konstant, som fører til mindre variasjon hastighet. Alternativet hadde vært å sette sammen en egen trommel, men det innebærer mer arbeid og ingen signifikante fordeler annet en kanskje pris. Snellen som ble brukt hadde en bryter for å bestemme retningen på rotasjonen. For at snellen skulle kunne rotere begge veier ble den åpnet og en liten tapp ble fjernet, se figur 30



Figur 30: Metalltapp som ble fjernet fra innsiden av fiskesnellen, slik at den kan rotere begge veier uhindret

3.6.3 Snor

Det er ønskelig med en snor som ikke er elastisk, slik at dybdemålingene blir så nøyaktig som mulig. En vanlig fiskesene vil strekke seg for mye for dette tilfellet, så en flettet fiskesene er mye bedre egnet. Den er mye mindre elastisk, og har i tillegg mye høyere styrke i forhold til diameter. Styrken gjør at en mindre diameter kreves, som igjen gjør at friksjonen til vannet blir mindre. Mindre friksjon på senen vil føre til at strømninger i vannet ikke får like godt tak på snora. Flettet fiskesen er ofte mer synlig, noe som er ønskelig når vinkelen skal detekteres. Et eksempel på flettet fiskesene er vist i figur 31.



Figur 31: Nærbilde av flettet fiskesene [7].

3.6.4 Mulighet for ekstra kontaktflate mellom snor og trinse

Hvis det ikke er tilstrekkelig med friksjon mellom snoren og trinsen kan snoren legges en ekstra runde rundt trinsen. Dette vil gi en kontaktflate som er fem ganger større.

3.6.5 Plassering av nøkkelkomponenter

Trinsen er den viktigste komponenten. Plasseringen til den påvirker plasseringen til alle de andre komponentene, så den plasseres først. Det som er viktig å vurdere da er at det må være plass til begge kameraene ca. 10-15 cm unna. IR-sensoren må kunne festes i nærheten. Det må også være nok avstand ifra veggen til at snoren ikke blir hindret og kan bevege seg fritt. Selve trinsen er festet i en vinkel som ble laget på mekanisk verksted, og vinkelen er igjen festet i en planke som går på tvers i enden av krana.

Når posisjonen til trinsen er bestemt kan snella plasseres. Den festes i andre etasje, dvs på samme høyde som trinsen. IR-sensoren settes ca. 2 cm unna trinsen, den har en oppgitt rekkevidde på 2-30 cm. USB-kameraet festes i den ene sideveggen, 10 cm unna snora, og RasPi-kameraet festes i 2. etasje 15 cm unna. Det at lengden er ulik er ikke et problem i mitt tilfelle. Både kameraene og IR-sensoren er festet med dobbeltsidig teip til en vinkel som er skrudd fast i konstruksjonen.

Posisjonen til steppermotoren kan nå bestemmes ut ifra snella. Akselen til snellen og akselen til motoren må være på linje slik at en akselkopling kan brukes til å overføre kreftene fra motoren til snella. Begge deler er plassert på hver sin kloss slik at akslene er på lik høyde. Så er motorfestet justert slik at det passer. Oversiktsbilde av 2 etasje er vist i figur 32. I figuren kan vi også se at innsiden av kranen er foliert svart, dette er for å få størst mulig kontrast mellom snora og bakgrunnen.

Figur 32: Oversiktsbilde av 2. etasje

3.7 Erfaringer og problemer

3.7.1 Implementering av to kameraer

For å lese av en videostrøm i OpenCV må kilden spesifiseres. Med ett kamera tilkoblet, var det ingen problemer. Med da kamera nummer 2 ble koblet til, kom det opp feilmeldinger markert med V4L (Video 4 Linux) og V4L2, og at feil videokilde var brukt. Etter feilsøking ble det funnet ut at kameraene skulle ha vid0 og vid1 som kilde, men programmet ville fortsatt ikke kompilere. For å løse problemet var det nødvendig å først kompilere scriptet uten kameraene tilkoblet, deretter kjøre det med kameraene tilkoblet. Det gjorde at systemet fungerte som tiltenkt.

3.7.2 Skrive til SD-kort mens motoren kjører

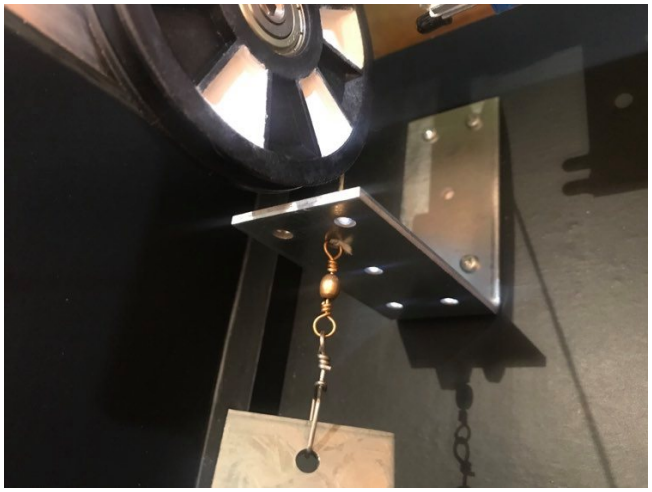
For å skrive til et SD-kort må det først initialiseres, før det kan åpnes og skrives til. Når Arduinoen er ferdig med å skrive og lese, lukkes SD-kortet og scriptet går videre. Dette er en prosess som blokkerer prosessoren. Da det ble forsøkt å skrive til SD-kortet mens steppermotoren kjørte, førte dette til at motoren kjørte veldig hakkete. Problemet var at motoren bevegde seg rundt 500 trinn i sekundet, basert på inputs ifra Arduinoen. Dermed tok det for lang tid å skrive til SD-kortet mellom trinnene. Det som ble løsningen på dette var å lagre dybden med klokkeslettet i ett array, og heller skrive alt til SD-kortet på en gang, mens motoren stod stille.

3.7.3 Kranen returnerte ikke til utgangspunktet etter kjøring

Da kransystemet først kjørte i sin helhet, var det programmert å slutte å heise opp loddet da den inkrementelle dybdetelleren returnerte til 0. Det førte til at loddet ville stoppe ca. 4-8 cm lavere enn det var da systemet startet. For å løse dette ble det lagt inn en fysisk stopper, i form av et metallbeslag med det lite hull til snoren, se figur 33. Dette hullet blir senere brukt som referansepunkt. Programmet ble modifisert slik at kranen skal heise opp loddet helt til trinsen slutter å rotere. For hver gang dette skjer resettes dybdetelleren til 0.

3.7.4 Problem med IR-sensor og direkte sollys

Da kranen først prøvde å kjøre ute skjedde det ingenting. Den stod ute i vårsola med direkte sollys inn i kranen bakfra. Jeg antok at det var IR-sensoren som ikke leste det den skulle, fordi når den ikke oppdateres i løpet av 2 tidels sekund tror kranen at den har truffet bunn. Kranen vil begynne å heise opp igjen, men den er allerede heist opp. Kranen ble forsøkt dekket



Figur 33: Fysisk blokk som sørger for at kranen heies opp til samme punkt hver gang

til ved å legge en finerplate på toppen og en genser til å dekke bakdelen av kranen, uten at det hjalp. Deretter ble det lastet opp et program som kun leste av verdien fra IR-sensoren, og den ga ut en konstant verdi. Kranen ble tatt med inn og samme program ble lastet opp. Denne gangen leste IR-sensoren det den skulle. Problemet kan være at sola lyser direkte på mottageren, fordi sollys inneholder bølgelengder som mottageren er på utkikk etter. Det er og en mulighet at temperaturen er årsaken til at sensoren gir en konstant verdi ut. Dette ble ikke undersøkt videre, fordi kranen hovedsakelig er ment til å brukes på nattestid.

4 Testing og resultater

4.1 Dybdemål

Det var ønskelig å teste nøyaktigheten til dybdemålingen. Tidligere hadde dybdemålingen kun blitt testet fra 1 meters høyde. Kranen ble nå plassert på en høyde, som ved hjelp av en laser avstandsmåler, målte 3,85 meter fra bakken til bunnen av loddet. Koden brukt til å kjøre kransystemet er listet i vedlegg A.1.

Figur 34: Kran satt opp på terrassen for å teste dybdemålingen

4.1.1 Første gjennomkjøringer fra terrasse

Kranen ble stilt inn slik at den skulle senke et lodd 5 meter, slik at loddet konsekvent kom til å treffe bakken. Kranen oppførte seg som forventet, loddet ble senket til det traff bakken, motoren skiftet retning og loddet ble heist til topps igjen. Kranen kjørte fem runder før målingene ble sjekket. Målingene viste at kranen ble senket: 3.84m, 3.80m, 3.82m, 3.82m og 3.80m. Altså var målingene innenfor 5 cm av den målingen som ble gjort med laser avstandsmåleren.

4.1.2 Test 1, justere heve og senke hastigheten

Avstanden som skulle måles hadde blitt økt. Dermed var det ønskelig å øke hastigheten, slik at målingene kunne gjennomføres raskere. Problemet med å øke hastigheten oppstår når loddet treffer bunnen, fordi snellen rekker å gi ut mer snor før systemet merker at trinsen ikke roterer lengre. Dette fører til at mer snor må snurres inn før loddet begynner å bevege på seg. Snoren alene klarer ikke å få trinsen til å rotere. Det systemet da ser er at motoren heiser opp snor, men trinsen beveger seg ikke. Dermed antar systemet at loddet er heiset opp og motoren stopper. Et triks som hjalp på dette problemet var å snurre snora en ekstra gang rundt trinsen. Da vil den ekstra snora som snurres ut havne mellom trinsen og snellen, og ikke mellom loddet og trinsen. Ved å implementere dette kunne steppermotoren øke hastigheten fra 500 trinn per sekund til 700.

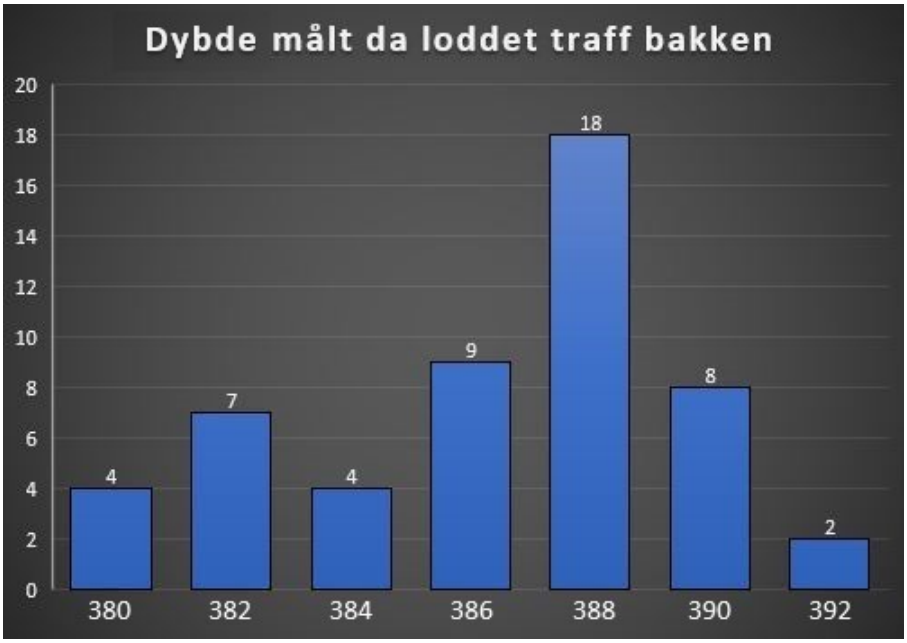
Samtidig som hastigheten ble økt, ble dybdemålingene også logget. Dataene viste en sammenheng mellom hastigheten og lengden som ble målt, ved at lengden ble lengre når hastigheten økte. Dette gir mening, med tanke på at mer snor blir snurret ut etter at loddet har truffet bakken. Figur 35 viser resultatene av dybdemålende som ble gjort i test 1. Grunnen til at målingene hopper med 2 cm for hver gang er at et felt på trinsen tilsvarer 2,0525 cm. Målingene blir rundet av til nærmeste hele cm. Hvis målinger innenfor [384, 386] regnes som korrekt, er 13 målinger korrekt, 11 er for korte og 28 er for lange. Hovedandelen av målingene måler litt for mye, men alle målingene havner innenfor et intervall på 12 cm.

4.1.3 Test 2, konstant hastighet i vind

Test 2 ble gjennomført med samme oppsett som test 1, det som er ulikt er at hastigheten er nå satt konstant og målingene ble gjort en annen dag hvor det var betydelig mer vind.

4.1.4 Observasjoner under testing

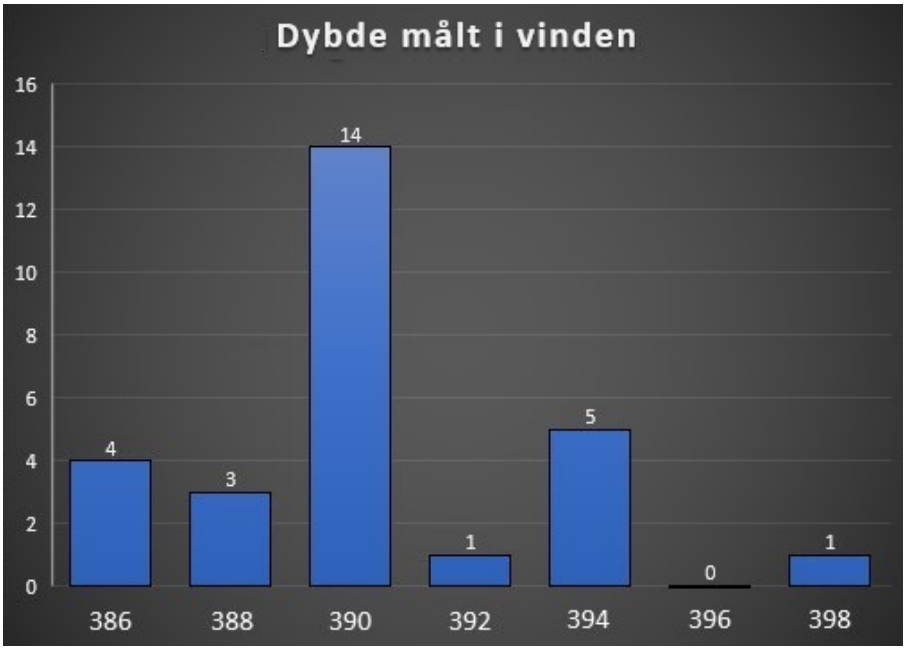
Under testingen var det ganske tydelig at vinden påvirket både loddet og snoren. Loddet veivet frem og tilbake og snoren hang ikke lengre loddrett, men mer i en bue. Det førte til at da loddet traff bakken var det overflødig med snor mellom trinsen og loddet. Konsekvensen av dette var det samme som i test 1; loddet ble stående på bakken fordi motoren ikke rakk å sveive inn nok snor til at trinsen begynte å rotere. Dette skjedde 20 ganger av i løpet av de 48 gjennomkjøringene som ble gjort.



Figur 35: Første dybdemålingene fra terrassen

4.1.5 Resultat av test i vind

Resultatet av test 2 er presentert i figur 36, og som forventet er målingene nå forskjøvet enda lengre mot høyre. Den lengste målingen viser 398 cm, noe som er 13 cm mer enn referansemålet på 385 cm. Denne målingen ble mest sannsynlig gjort under et vindkast og skiller seg derfor ut.



Figur 36: Resultat av testing av dybdemål i vinden

4.1.6 Sammenheng mellom vind og strømninger i vannet

I utgangspunktet var det ikke meningen å teste kranen i vind, men da kranen begynte å kjøre var det tydelig at det gikk an å trekke likheter mellom vind og strøm i vannet. Vinden forskjøv både snoren og loddet slik som strømninger i vannet også vil gjøre. Som resultatet av denne testen viste, vil vind/strøm påvirke dybdemålingene, som igjen vil føre til større usikkerheter.

4.1.7 Test 3, optimalisert oppsett

Som følge av at både test 1 og test 2 ble gjennomført med enten varierende hastighet eller under varierende forhold, ble en 3. test satt opp. Denne gangen for å se hvor presis kranen er på egenhånd. Testen er satt opp likt som test 2, med unntak av at vinden har løyet. Totalt ble det gjort 75 målinger, der loddet ble stående på bakken to ganger.

4.1.8 Resultat av test av nøyaktighet og presisjon til dybdemålingene

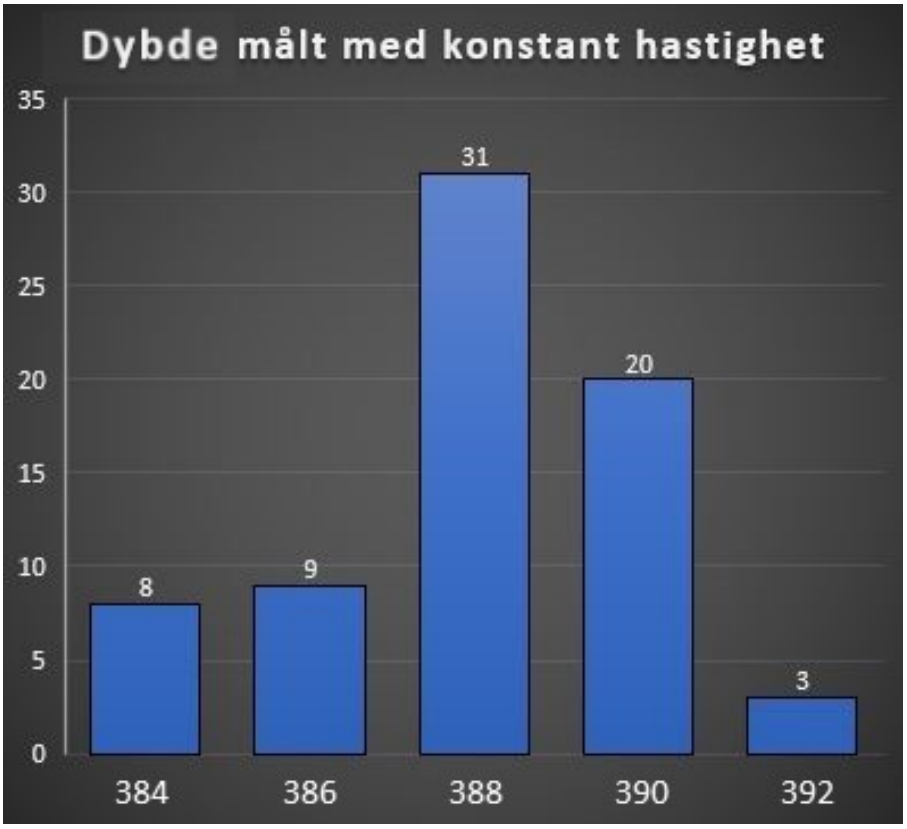
Figur 37 viser resultatet av målingene. Figuren viser en høyere presisjon enn i test 1 og test 2, men flesteparten av målingene er fortsatt for lange. Det er mer naturlig at målingene blir for lange enn for korte, fordi kranen er stilt inn slik at loddet uansett skal treffe bakken. Når loddet treffer bakken kan trinsen fortsatt ha momentum og rotere litt ekstra, eller så kan det være at snoren ikke er 100% rett.

4.1.9 Andre merknader gjort under test av dybdemål

Hittil har det fokusert på hvor langt loddet ble senket, men det er også interessant å se hvor langt loddet ble hevet. I realiteten er det installert en stopper, slik at kranen heiser loddet opp til den treffer stopperen og det er samme punktet hver gang. I koden er dybdemåleren satt til å nullstilles hver gang loddet treffer stopperen. Men verdien til dybdemåleren stoppet ikke alltid på null. Etter å ha studert datasettene ser det ut som dybdemåleren som oftest stopper en plass mellom 0 cm og -12 cm. Merk at systemet måler dybde som positivt, -12 cm betyr da at systemet måler 12 cm for mye. Med dette tatt i betraktning begynner det å danne seg et mønster hvor kranen måler en avstand som er for lang, både på vei opp og ned.

Under test 3 ble det brukt en stoppeklokke til å måle tiden som kranen brukte på å heve loddet 3,85 meter, når den var stilt inn til 700 trinn per sekund. Tiden ble målt til å være 8,3 sekunder, noe som tilsvarer en hastighet på cirka 0,46 m/s.

Samtidig som kranen kjørte og tok dybdemål, ble temperaturen til motoren og motorkontrol-



Figur 37: Testing av dybdemål under optimale forhold med konstant hastighet

leren overvåket. Etter å ha kjørt motoren i over 20 min, gikk det an å kjenne at den var litt varmere enn omgivelsene. Motorkontrolleren derimot, steg gradvis i temperatur fra starten av testen. Etter cirka 15 min ble den så varm at det ikke lenger gikk an å ta på kjøleribben som er festet på.

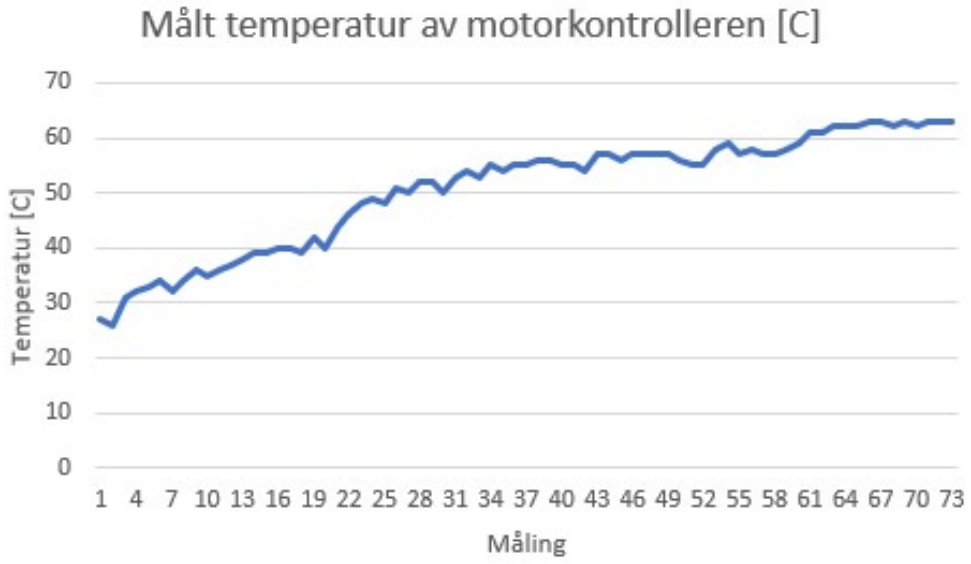
4.2 Temperaturutviklingen til motorkontrolleren

Ettersom temperaturen til motorkontrolleren ble ganske høy i løpet av testingen til dybdemålingen, var det ønskelig å teste hvor høyt temperaturen steg. Dette er for å unngå at komponenten blir ødelagt eller får varige skader under bruk. Ifølge databladet har motorkontrolleren en termisk beskyttelse som slår inn idet den når 150°C. For å måle temperaturen ble det brukt et infrarødt termometer som vist i figur 38.

Figur 38: Infrarødt termometer brukt til å måle temperaturen på motorkontrolleren

Kranen ble satt opp inne på en pult og ble programmert til å kjøre konstant opp og ned med 2 sekunders pause på toppen, slik at temperaturen kunne måles og noteres mellom hver heving/senking. Før testingen startet ble temperaturen til motorkontrolleren målt til å være 20,5°C.

Figur 39 viser resultatet av temperaturmålingene. Grafen viser at temperaturen starter på 27°C, dette er fordi kranen ble kjørt noen runder først for å se at alt var satt opp riktig. Stigningen til grafen avtar etterhvert og grafen flater ut på rundt 63°C.



Figur 39: Temperaturutviklingen til motorkontrolleren når kranen kjøres konstant

4.3 Drakraft

Så langt har det blitt brukt et vinkeljern som lodd under testing. Vinkelen har en vekt på cirka 95 gram, noe som er mindre enn det systemet burde kunne takle. Målet er å kunne løfte en måleprobe som kan veie opp mot 1 kg når den er over vann.

4.3.1 Hva skjer når vekten på loddet økes?

Det ble lagt på 50 gram på loddet, slik at totalvekten økte til 145 gram, det resulterte i at tiltrekkingsmomentet ikke lenger var tilstrekkelig for å holde loddet på egenhånd. Tiltrekkingsmomentet, som oppstår når motoren ikke er tilkoblet strøm, er kun avhengig av magnetene som er inne i motoren, og kan derfor ikke økes uten å oppgradere motoren. Det ble koblet på strøm, og holdemomentet hadde ikke noe problem med å holde loddet oppe. Kranen ble startet og den hadde ikke problemer med å senke loddet, men da loddet skulle heves hoppet motoren bare over trinn og loddet stod stille.

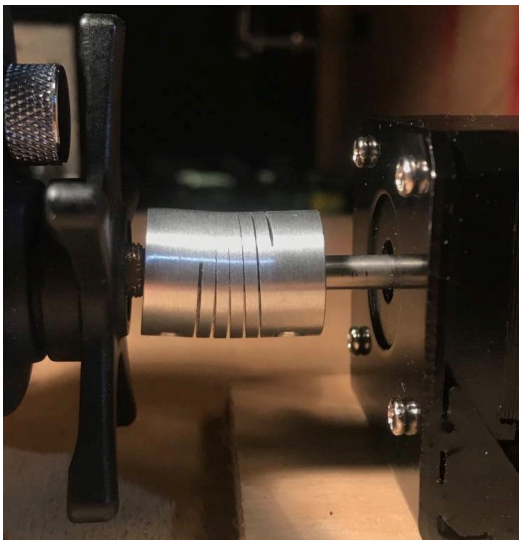
4.3.2 Maks drakraft til systemet

For å teste den maksimale drakraften til systemet ble det brukt en bagasjevekt i enden av snoren i stedet for et lodd. Deretter ble det lastet opp ett program som gjorde at motoren sveivet inn snoren med konstant hastighet. Bagasjevekten ble holdt i ro samtidig som motoren prøvde å sveive inn. Som vist i figur 40 er systemet i stand til å løfte 120 g slik som det er satt opp nå.

Figur 40: Drakraft målt ved hjelp av bagasjevekt

4.3.3 Svakheter som fører til tapte krefter i løftesystemet

Akselkoblingen mellom motoren og snellen er vanskelig å få montert 100% rett, fordi festepunktet på snellen er laget av messing som deformerer seg litt når akselkoblingen strammes til. Dette er demonstrert i figur 41, hvor akselkoblingen på oversiden er komprimert og undersiden er utvidet. Dette kommer av at akselkoblingen er utstyrt med en fjær slik at den tillater slike små unøyaktigheter, men det kreves et større moment for å rotere akslingen.



Figur 41: Bøy i akselkoblingen som fører til tapte krefter

En annen svakhet er snellen som er giret 1:3, 1 omdreining på motoren fører til 3 omdreininger på snellen. Dette er ikke en ideell kombinasjon med den motoren som brukes nå, fordi motoren allerede sliter med lite kraft. Mengden snor på snellen vil påvirke radiusen på snellen og drakraften. Som formel 10 viser, vil en økning i radius føre til reduksjon av kraft. Under testing var diameteren på snellen målt til 44 mm. I det endelige produktet vil det være ønskelig å ha begrenset med snor på snella.

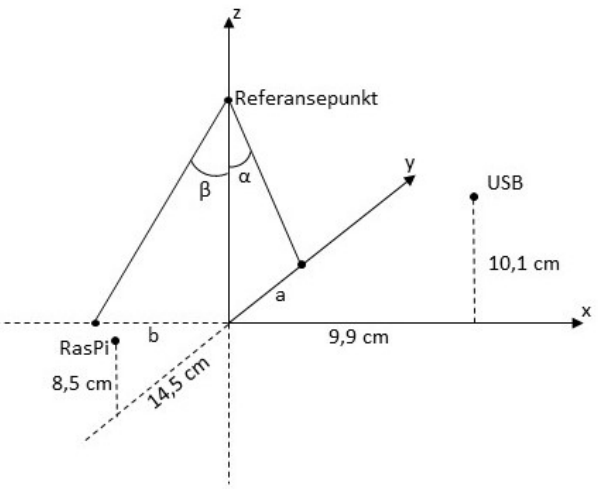
$$\begin{aligned} Moment &= Kraft \cdot Arm \\ Kraft &= \frac{Moment}{Arm} \end{aligned} \tag{10}$$

4.4 Vinkelmålesystem

Formålet med denne testen er å sette vinkelmålesystemet på prøve. Det er ønskelig å finne ut hvor nøyaktig systemet er, og hvor stor rekkevidde det har. Både i forhold til mitt formål og i forhold til hva systemet faktisk er kapabel til. Koden brukt til å kjøre kamerasystemet er listet i vedlegg B.1.

4.4.1 Testoppsett

Prototypen min blir brukt som testoppsett for å teste hvor nøyaktig systemet klarer å måle vinkler. Det blir definert et koordinatsystem der x-aksen går fra snora mot USB kameraet, y-aksen går fra RasPi kameraet mot snora og z-aksen er vertikal med positiv retning oppover. snora utgjør origo, og xy-planet dekker under undersiden av fronten til kranen. Det er et referansepunkt der snora går igjennom et hull som ligger 16 cm over origo på z-aksen. Det er viktig å merke seg at diameteren på hullet som snora går igjennom er 5 mm. Linsen til USB kameraet har koordinatene $(9.9, 0, 10.1)$[cm] og RasPi kameraet ligger på $(0, -14.5, 8.5)$[cm]. Koordinatsystemet er vist i figur 42. Kameraene er festet slik at referansepunktet alltid vil være i senter rett over bildet.



Figur 42: Definisjon på koordinatsystemet brukt til å finne snoras orientering

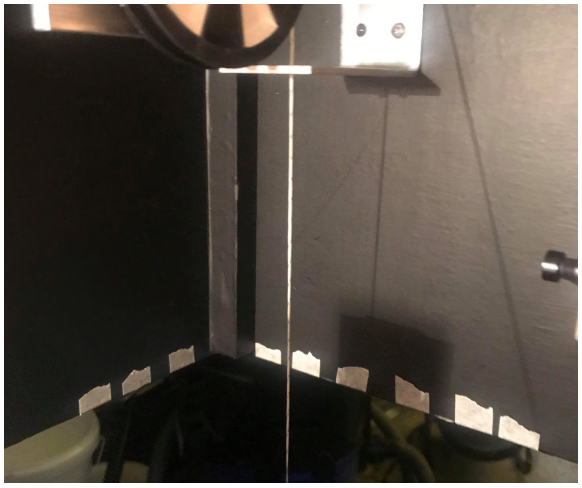
4.4.2 Konsept for testing av vinkelmålesystemet

For å kunne si noe om hvor presis vinkelmålingen til kameraene er må de sammenlignes med noe. Det er derfor kommet opp med en alternativ metode for å finne samme vinkelen som

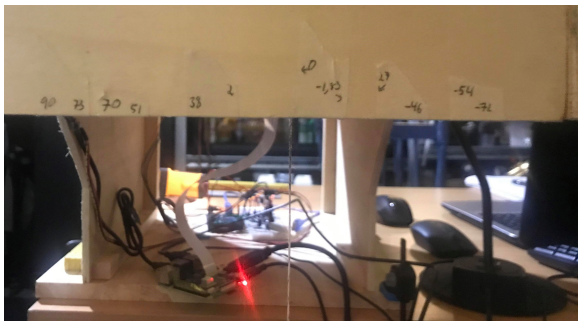
kameraene ser. Denne metoden baserer seg på lengdemål gjort ved hjelp av et elektronisk skyvelær. For å få dette til brukes punkter på kranen, der det er mulig å måle avstanden fra origo og plassere snoren i samme punktet slik at kameraene kan måle en vinkel. Ved å bruke posisjonen til punktet som er målt sammen med de andre faste punktene som er definert, er det mulig å finne frem til samme vinkelen som kameraene måler.

4.4.3 Målesett

For å finne punkter som kunne brukes flere ganger ble det valgt å ta målinger langs de to veggene som er dekket i svart. Dette førte til to målesett, et der x-verdien var konstant, og et hvor y-verdien var konstant. Det ble gjort ved å markere forskjellige punkter med teip, for å så måle xy-posisjonen manuelt for hånd, som vist i figur 43. Alle punktene som ble målt ligger i xy-planet, så z verdien er ikke relevant. Millimeter målene ble notert på andre siden av veggen, på selve teipbitene. Deretter ble snora ført bort til de målte punktene for å notere vinklene som kameraene leste. Vinkelmålingene ble avrundet til nærmeste hele grad, fordi da vinkelen ble lest på skjermen hendte det at vinkelen ikke landet på en verdi, men hoppet mellom forskjellige verdier. Det ble da valgt enten medianen eller den verdien som dukket opp oftest.



(a) Innsiden av kranen



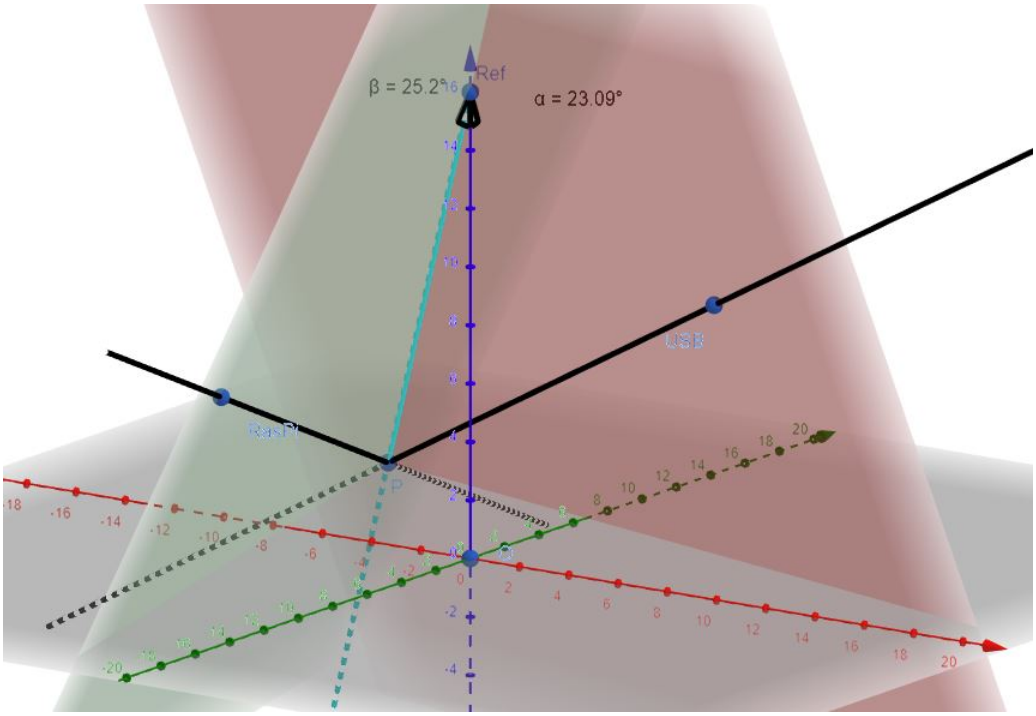
(b) Utsiden av kranen, med millimetermål notert på teipen

Figur 43: Oppsett for å markere xy-posisjoner med hjelp av teip for å teste nøyaktigheten til vinkelmålesystemet

4.4.4 Behandling av data

Kameraene finner to vinkler ut ifra z-aksen, den ene går langs x-aksen (β) og den andre går langs y-aksen (α). Begge vinklene har spissen i referansepunktet. Se figur 42.

Siden kameraene ikke har dybdesyn, vil det si at linjen som kan sees på skjermen egentlig er et 3 dimensjonalt plan i rommet og snora ligger i det planet. Planene bestemmes av koordinatene til Ref, koordinatene til kameraene og vinkelen som kameraene måler. Snora vil være lik linjen som oppstår når disse to planene krysses. Dette er vist i figur 44 hvor snora er representert med en turkis linje. Figuren ble laget i 3D-kalkulatoren til Geogebra [21].



Figur 44: Sammenheng mellom vinkelen som måles ved hjelp av kamera og posisjonen som måles i koordinatsystemet (P)

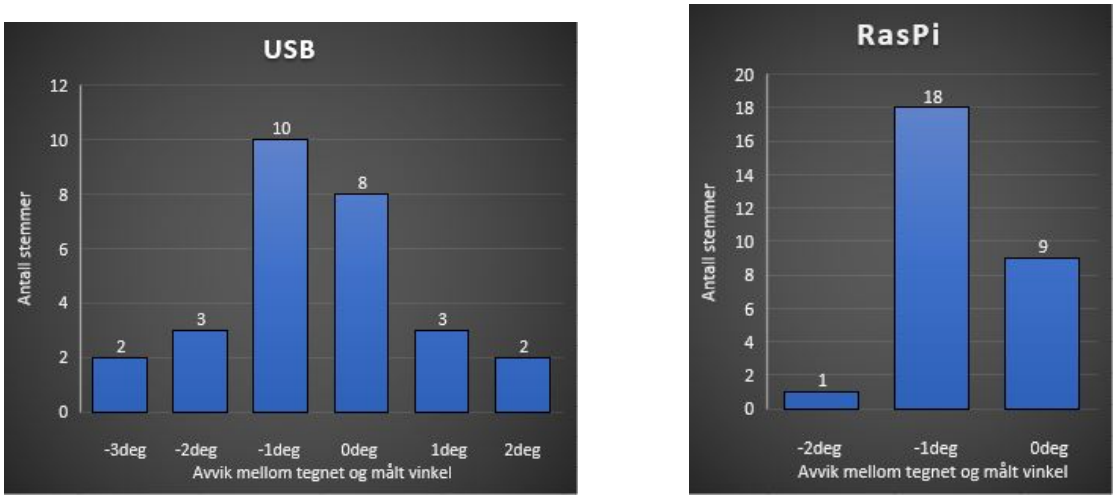
Figur 44 kan også brukes til å forklare hvordan vi finner α og β ut ifra xy-posisjonen (P) som blir målt for hånd. Først blir posisjonen til USB, RasPi, Ref, P og origo plottet. Disse punktene kan brukes til å plotte det røde og grønne planet, samt snora. For å finne vinklene α og β som jeg måler med kameraene, må snora prosjekteres på y-aksen for α og x-aksen for β.

4.4.5 Resultat av vinkeltestingen

Før resultatet av testen blir presentert, er det viktig å påpeke at denne testen kun sammenligner en vinkel som er funnet ved hjelp av to forskjellige metoder, der ingen av metodene er nøyaktige.

Men testen gir en god indikasjon på hvorvidt kamerasystemer måler realistiske verdier.

Avviket mellom den målte vinkelen, og den som ble funnet ved hjelp av plotting er presentert i figur 45. Avviket til USB varierer mellom fra $2°$ til $-3°$ og avviket til RasPi varierer fra $0°$ til $-2°$. Årsaken til at RasPi er mer presist kan være at det er et dyrere kamera (379 kr), som er tilpasset til å brukes sammen med en Raspberry Pi, mens USB kameraet er et billig kamera (ca. 150 kr) med fiskeøyelinse. Ut ifra histogrammene kan vi se at målingene ikke er normalfordelt rundt 0, men heller trekker mot -1. Det kan være flere grunner til dette, for eksempel kan det være at konstruksjonen og festet til kameraet ikke er i vater. Kameraene er kun festet med dobbeltsidig teip, så det skal ikke mye til for at kameraene roterer. Det kan skje hvis kameraene blir dultet borti, eller ledningene blir dratt i. Det er mulig å justere dette før målingene gjøres, når snora henger loddrett kan kameraene roteres med å vri på dem til vinkelen på skjermen viser 0 grader. Histogrammene er plottet ut ifra tabell 6.



(a) Avviket til USB.

(b) Avviket til RasPi.

Figur 45: Avviket mellom vinkelen som ble målt av kamerasystemet og vinkelen som ble funnet ut ifra en xy-posisjon.

4.4.6 Test av rekkevidde

Ut ifra tabell 6 så var aldri avviket større en 3 grader. Dermed er systemet testet og godkjent når vinkelen til USB kameraet er mellom $[-38°, 45°]$ og RasPi kameraet er innenfor $[-53°, 40°]$

Tabell 6: Sammenligning av vinkler målt av systemet og vinkler tegnet ut ifra xy-koordinater

	A	B	C	D	E	F	G	H	I	J
1	x	y	USB målt	USB tegnet	USB avvik		RasPi målt	RasPi tegnet	RasPi avvik	
2	-7,5	-16,7	-38	-38	0	deg	-53	-52	-1	deg
3	-7,5	-14,9	-33	-35	2	deg	-49	-48	-1	deg
4	-7,5	-12,2	-28	-30	2	deg	-45	-44	-1	deg
5	-7,5	-10,4	-25	-26	1	deg	-42	-41	-1	deg
6	-7,5	-8,5	-21	-22	1	deg	-40	-39	-1	deg
7	-7,5	-6,7	-17	-17	0	deg	-37	-37	0	deg
8	-7,5	-4,2	-11	-11	0	deg	-35	-34	-1	deg
9	-7,5	-2,6	-7	-7	0	deg	-33	-32	-1	deg
10	-7,5	-1,8	-5	-5	0	deg	-32	-32	0	deg
11	-7,5	0	0	0	0	deg	-31	-30	-1	deg
12	-7,5	1,6	4	4	0	deg	-30	-29	-1	deg
13	-7,5	3,4	10	9	1	deg	-28	-28	0	deg
14	-7,2	6,5	15	18	-3	deg	-21	-20	-1	deg
15	-5,4	6,5	17	19	-2	deg	-16	-16	0	deg
16	-4,6	6,5	18	19	-1	deg	-14	-13	-1	deg
17	-2,7	6,5	19	20	-1	deg	-9	-8	-1	deg
18	-1,8	6,5	20	21	-1	deg	-6	-5	-1	deg
19	0	6,5	21	22	-1	deg	0	0	0	deg
20	2	6,5	22	24	-2	deg	5	6	-1	deg
21	3,8	6,5	24	25	-1	deg	10	11	-1	deg
22	5,1	6,5	26	27	-1	deg	14	15	-1	deg
23	7	6,5	28	29	-1	deg	19	20	-1	deg
24	7,3	6,5	29	29	0	deg	20	21	-1	deg
25	9	6,5	30	31	-1	deg	25	25	0	deg
26	11	6,5	33	35	-2	deg	28	30	-2	deg
27	12,8	6,5	35	38	-3	deg	33	33	0	deg
28	14,7	6,5	41	42	-1	deg	37	37	0	deg
29	16,5	6,5	45	46	-1	deg	40	40	0	deg

4.5 Test i mørket

Siden kranen primært skal brukes på nattestid, er det relevant å teste hvorvidt systemet fungerer i mørket. Spesielt for å se hvordan kameraene og IR-sensoren oppfører seg når lysforholdene blir dårligere. For å finne ut av dette ble kranen satt opp i et rom hvor lysforholdene kunne reguleres.

4.5.1 IR-sensor i mørket

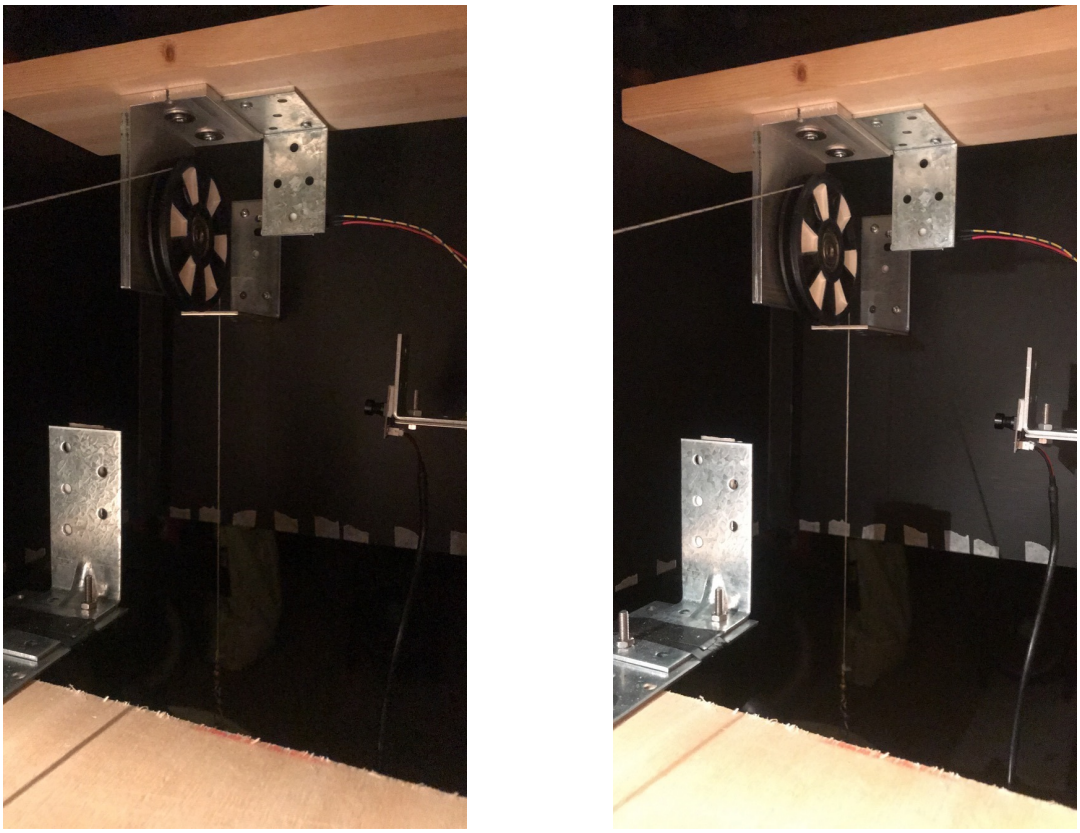
IR-sensoren ble ikke påvirket av mørket, dette er som forventet fordi IR-sensoren har sin egen sender og mottaker. Dette gjør at den kan reflektere signaler av den hvite overflaten selv i dårlige lysforhold.

4.5.2 Kamera i mørket

Kameraene hadde større problemer når lysforholdene ble dårligere. Det var kun en lyskilde ca. meter unna kranen, som kun lyste på utsiden av kranen. Det var ingen direkte lys på snoren i det området som var synlig for kameraene, utfallet av dette var at kameraene hadde problemer med å se snoren. Dette var fordi det ikke lenger var nødvendig med kontrast mellom snora og bakgrunnen. Det ble eksperimentert med en ekstra lyskilde i form av en kontorlampe som lyste inn i krana. Med lys direkte på snora førte det til en skygge på veggen som kameraene plukket opp. Stort sett valgte kameraene snora, fordi den har større kontrast til bakveggen, men ved et par tilfeller ble skyggen vurdert som en sterkere linje. Dette er noe som må vurderes ved implementasjon av en eventuell lyskilde.

For å teste mengden lys som er nødvendig var det interessant å måle lux verdien når det akkurat er tilstrekkelig med lys, og når det er for lite lys til at kameraene klarer å oppfatte snora. Lysenheten lux er lysmengden som faller på en overflate, eller lumen per kvadratmeter. For å måle lux ble det brukt en app som heter Lux Light Meter Pro til iPhone [22]. Ettersom jeg har lite erfaring med måling av lys, brukte jeg appen først til å måle på forskjellige mørke og lyse områder. Jeg fikk verdier på under 50 da jeg målte steder som ikke var direkte lyst opp, jeg målte 0 på mørke områder, jeg målte verdier på 200-300 da jeg målte rett under kontorlampa, og jeg fikk verdier på flere tusen da jeg målte direkte inn i lampen. Så målingene virket greie og virket som om de var skalert riktig. Appen bruker kameraet til å måle lux verdien, og jeg skjønte fort at det ikke gikk an å måle direkte på snora, men at jeg heller kunne måle lyset

som faller på flatene bak. Problemet med dem er at de er foliert svarte for å få kontrast mellom snora og bakgrunnen, og appen hadde store problemer med å måle på svarte overflater. Dette førte til at jeg brukte en hvit bakgrunn da jeg prøvde å ta målinger. Men selv med en hvit bakgrunn klarte ikke lux måleren å skille mellom tilstrekkelig belysning og for lite belysning, den viste 0 uansett. En visuell representasjon på tilstrekkelig lys er vist i figur 46. Ut ifra denne testen kom det også frem at USB kameraet klarte å finne snoren i mindre lys enn det RasPi kameraet klarte. Dette er antageligvis fordi USB kameraet har en større linse.



(a) For lite lys

(b) Tilstrekkelig med lys

Figur 46: Sammenligning av for dårlige lysforhold med tilstrekkelige lysforhold til at kamera-systemet skal fungere

4.6 Drift i RTC-modulen

I databladet til DS3231 står det oppført at den har en stabilitet på ±2 parts per million (ppm) under optimale forhold. ±2 ppm vil i praksis si at etter 1 million sekunder, eller 11 dager, 13 timer, 46 minutter og 40 sekunder vil klokken ha driftet med ±2 sekunder. For å teste dette ble RTCen min stilt etter klokka på laptopen den 25. Mars, slik at de var innenfor et sekund av hverandre. Over to måneder senere den 28. Mai ble klokken igjen sammenlignet for å se hvor mye RTCen hadde driftet. Resultatet er vist i figur 47.

```
15:02:30.518 -> Thursday 28.05.2020 -- 15:03:25
```

Figur 47: Resultatet av drift til en DS3231 over to måneder

Tidspunktet til venstre er tiden fra laptopen, og tiden til høyre er fra RTCen. Det vil si at laptop klokken lå 55 sekunder bak RTCen. På 64 dager har altså RTCen driftet 55 sekunder, noe som tilsvarer en nøyaktighet på ca ±10 ppm. Se formel 11.

$$\begin{aligned} 64d &= 5529600s \\ \frac{5529600s}{5,5296} &= 1000000s \\ \frac{55s}{5,5296} &\approx 9,95s \end{aligned} \tag{11}$$

5 Diskusjon

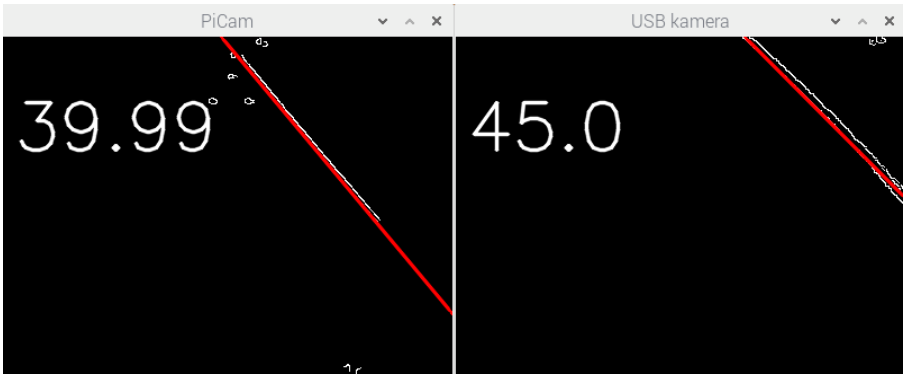
Målet med denne oppgaven var å utvikle et system som kan senke og ta opp sensorer, ha nøyaktig kontroll på hvor dypt måleproben er senket til enhver tid og registrere dette sammen med tid og måle snordrag og snorvinkel.

5.1 Kamerasystemet

Resultatet av testingen i seksjon 4.4 var bedre enn forventet. Gjennom utviklingen har alltid systemet virket raskt og presist, men jeg hadde ikke sett for meg at nøyaktigheten skulle være så høy at det er maskinvaren som setter grensene. RasPi-kameraet var i 28 av 29 tilfeller maks en grad unna.

5.1.1 Usikkerheter i testingen

Selve avlesningen av vinkelen fører til usikkerheter i målingene. Det hendte at programvaren ikke fant en bestemt vinkel, men den varierte innenfor et område. USB-kameraet kunne på det meste variere med 5° selv når snoren var helt rett og holdt i ro. Dette var og tilfellet med RasPi-kameraet men ikke i like stor grad. Jeg noterte meg at desto større vinkelen var, desto mer varierte de detekterte målingene. Grunnen til at dette er tilfellet er fordi programvaren veksler mellom å finne over- eller undersiden av snoren. Figur 48 er et eksempel på dette, linjen er bøyd så vinkelen kan variere ut ifra hvilken del av linja programvaren velger. Det som ble gjort var å notere enten medianen eller den vinkelen som var lengst på skjermen.



Figur 48: Når vinkelen blir større, vil det bli større forskjell om programvaren finner over- eller undersiden av snoren. Dette gjelder spesielt for USB-kameraet.

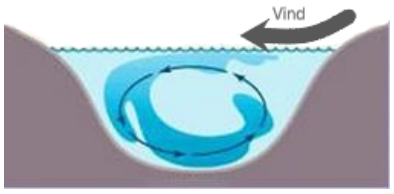
Oppmåling med målebånd og geometriske beregninger ble benyttet for å teste det optiske vinkelmålesystemet. Ingen av metodene er nøyaktige, men metodene er uavhengige og kan dermed sammenlignes for å gi et godt bilde på hvorvidt kamerasystemet fungerer. Det faktum at RasPi-kameraet har betydelig bedre presisjon enn det USB-kameraet har, er med på å underbygge testen. Fordi testen klarer å skille mellom et kamera som er egnet, og et som ikke er egnet til formålet. Det er verdt å merke seg at denne testen er begrenset til målinger langs veggen til konstruksjonen, fordi testen er avhengig av faste punkter som det er mulig å måle xy-posisjonen til og få snoren til å gå igjennom samme punktet.

Usikkerheten til vinkelen, som kamerasystemet måles mot, stammer i fra lengdemålene. Det å måle millimetermål for hånd er ikke optimalt, men for å minimalisere feilen ble det brukt et elektronisk skyvelær. Skyvelæret egent seg veldig bra i mitt tilfelle, fordi spissene på baksiden kunne settes fast i finerplaten for å sørge for at skyvelæret ikke kunne gli ut av posisjon. I tillegg ble det brukt teipbiter for å sørge for at målingene hadde samme utgangspunkt.

Til slutt er det verdt å nevne at referansepunktet ikke er 100% nøyaktig, fordi det er et hull med diameter på 5 mm som snora går igjennom. Det vil si at referansepunktet kan variere på ca. $\pm 2,5$ mm i x- og y-retning. Dette fører da til en ulikhet mellom den målte og tegnede verdien.

5.1.2 Mangler i systemet

I innsjøer vil vind ofte føre til at vannlaget på toppen vil bevege seg med vinden, men da må vann lenger nede bevege seg motsatt vei. Dette fører til sirkulasjon i innsjøen, se figur 49. Systemet mitt er begrenset til å kun måle retningen på snora i det den går ned i vannet. Når den er under vann klarer ikke systemet å se hvilken retning snora har. Med sirkulasjon kan da proben drive motsatt vei av det kamerasystemet måler på toppen.



Figur 49: Sirkulasjon i en innsjø

5.1.3 Oppgradering av kameraene

For å forbedre kamerasystemet ville jeg byttet ut begge kameraene. RasPi-kameraet ville jeg byttet ut med Pi NoIR Camera V2, som også er et originalt Raspberry Pi kamera. Forskjellen er at NoIR kameraet ikke har et infrarødt filter og kan dermed se i mørket med IR belysning. Det er praktisk å bruke et Raspberry Pi kamera sammen med Raspberry Pi, men Raspberryen kommer dessverre kun med en kamera port. Dermed måtte det andre kameraet byttes ut med et som ligner på NoIR, men med USB tilkobling. Ulempen med å legge til en lyskilde er at fisker generelt tiltrekkes av lys. Det ville gått imot ideen om å gjøre målingene på nattestid, når fisken ikke svømmer i stim, hvis fisken samlet seg rundt båten uansett. Hvorvidt fisken tiltrekkes av IR lys er ikke sikkert, noen ferskvannsfisker har utviklet synet slik at de kan se infrarødt lys [23]. Dette ville krevet videre testing.

5.1.4 Videre arbeid med kamerasystemet

Kamerasystemet er satt opp som et separat system, hvor den målte vinkelen er vist på en skjerm. Det neste steget burde være å implementere den informasjonen inn i autopiloten. Systemet i seg selv er mer enn kapabel til å si noe om hvor måleproben er med en presisjon som er høyere enn nødvendig. For autopiloten sin del er det viktigste å vite hvilken kvadrant snoren går til, slik at den kan kjøre frem eller tilbake, høyre eller venstre. Men muligens en dødsone i midten, slik at autopiloten slipper å konstant justere posisjonen.

5.2 Kransystemet

Resultatene fra seksjon 4.1 viser at dybdemålingen holder seg innenfor en feilmargin på 2% når det ikke er påvirket av ytre faktorer som feks vind. Resultatene antyder at det er tilstrekkelig med friksjon mellom trinsen og snora til at den ikke vil gli, og IR-sensoren overser felter på trinsen.

5.2.1 Mangel på motorkraft

Systemet klarte kun å løfte en vekt på 120g, noe som er for lite i forhold til de oppgavene kranen er ment for. Det forsvinner litt krefter i akseloverføringen, snella og trinsen, men den enkleste måten å oppgradere systemet på er å bytte ut motoren. Jeg tror fortsatt at en steppermotor er et godt alternativ, fordi den kan kontrolleres veldig nøye og på grunn av holdemomentet. Men

en el-motor med integrert ormgir kan også vurderes. Som følge av større motor kan systemet løfte mer vekt, som igjen vil føre til at snora ikke vil bøye seg så mye som den gjorde i vinden. Dette kan minke avviket når systemet blir påvirket av ytre krefter.

5.2.2 Hvor kommer den ekstra dybden i målingene fra?

Resultatene fra seksjon 4.1 viser at dybdemåleren som oftest måler for mye dybde. Jeg tror dette kommer av at trisen roterer for mye. I det loddet treffer bakken har trinsen fortsatt et rotasjonsmoment. Påvirkningen fra snoren vil minke fordi det kun er vekten fra snoren og friksjon ifra kulelagrene som påvirker trinsen. Dette kan føre til at trisen fortsetter å rotere etter at loddet har truffet bakken. Samtidig når loddet treffer bakken vil motoren fortsette å kjøre i noen få tidels sekund etter at den registrerer at trinsen har stoppet, dette fører til at ekstra snor blir frigjort fra snellen. Når motoren skifter retning igjen må den først begynne med å sveive inn det ekstra snorstykket. Dette kan føre til rotasjon i trinsen, spesielt når snoren er viklet en ekstra runde rundt trinsen. Dermed har dybdemåleren målt heving i systemet før loddet i det hele tatt har flyttet på seg.

5.2.3 Hypotetisk kransystem uten bruk av trinse

Kransystemet er utviklet med utgangspunkt i at en trinse skulle brukes, men hvordan ville systemet sett ut uten trinse? Først og fremst måtte kritiske deler av oppgaven løses annerledes. Rotasjonen kunne blitt målt direkte på snella, men diameteren her ville endret seg etter hvor mye snor som er sluppet ut. Hvis rotasjonen ble målt på motoren måtte giringen til snella tas hensyn til, og den er bare ca. 1:3. Dermed vil rotasjonsmålingen også her ha unøyaktigheter. I tillegg måtte et helt nytt system utvikles for å detektere om proben treffer bunn, fordi snellen vil ikke stoppe å rotere slik trinsen vill. Dette ville krevd et mer avansert system med mulighet til å måle snordraget.

5.2.4 Fremtidig testing

Systemet burde blitt testet sammen med en Conductivity Temperature and Depth (CTD) i vann, for å kunne sammenligne dybdemålingene som leses ut ifra trykket. Dette kunne gitt en god indikasjon på om målingene på vei opp og ned er realistiske. Mitt system har kun blitt testet i luft, og hvor stort avvik det blir i målingene når samme avstand dekkes flere ganger. En test med CTD var planlagt, men CTDen var ikke klar i tide. Den utvikles av en medstudent, som har blitt forsinket pga. korona situasjonen.

5.3 Konstruksjon

Det konstruerte kransystemet består av et hus i tre, som holder og beskytter motor og heisverk, trinse og vinkeldetektorsystem.

5.3.1 Klargjøring for felt

Før systemet er klart til å brukes i feltet må det være værsikkert. Elektronikken burde plasseres i en vanntett boks. Nipler kan monteres i boksen slik at ledningene fra IR-sensor, kamera og strømforsyning kan nå inn til elektronikken. Det burde også tilrettelegges for å bruke 12V-batterier som strømforsyning. Ved å seriekoble to 12V-batterier oppnås en spenning som er gunstig for steppermotoren, og med en step-down konverter kan Raspberry Pi og Arduino også forsynes med strøm.

6 Konklusjon

Denne oppgaven har presentert utviklingen av en kran beregnet som utstyr for en automatisk båt som skal overvåke innsjøer. Motivasjonen bak oppgaven er å automatisere overvåkingen av innsjøer. Dette er viktig fordi vi i dag ikke har ressurser til å gjøre det manuelt.

Det konstruerte kranssystemet med de valgte prinsippene fungerte generelt bra. Visse modifikasjoner bør gjøres før systemet kan tas i bruk.

Konseptet med å bruke maskinsyn til å bestemme probens posisjon i forhold til kranen var vellykket. Testing viser at systemet klarer å måle vinkelen til en snor med et maksimalt avvik på 2° når et egnet kamera brukes. Løsningen er altså nøyaktig og den består kun av ikke-bevegelige deler, noe som er gunstig når systemet skal brukes utendørs i ulike værforhold.

Den implementerte parallellprosesseringen klarte å effektivisere bruken av prosessoren. Antall bilder som behandles per sekund kunne økes med nesten 800%.

Implementeringen av en begrensning av fps var en suksess. Brukeren kunne sette en bestemt fps etter behov så lenge kameraene kunne støtte denne bilderaten.

Systemet har en stor rekkevidde når det gjelder å håndtere mulige vinkler som snoren kan oppnå. Kameraene er avhengig av en viss minstebelysning for å klare å detektere snora.

Dybdemålingen ved hjelp av en trinse fungerte bra. Det viste seg å være en enkel metode for holde kontroll på hvor mye snor som hadde passert trinsen. Testingen viste at denne løsningen gav en nøyaktighet på 2% ved en hastighet på 0,46 m/s når systemet ble brukt på land.

Bruken av en steppermotor til å drive systemet førte til en heving og senkning som var kontrollert og jevn, men motoren som ble brukt til testing var underdimensjonert. Den må derfor byttes ut mot en kraftigere steppermotor for at kranen skal kunne løfte ønsket vekt. Motorkontrolleren fungerte bra sammen med motoren som ble brukt, men den må også erstattes med en kraftigere enhet hvis kraftigere motor blir implementert.

Det å bruke en IR-sensor til å kontrollere rotasjonen til trinsen var vellykket. IR-sensoren kunne håndterte mange nye oppdateringer i sekundet, men den var følsom for direkte sollys, noe det må tas hensyn til hvis systemet skal brukes på dagtid.

Systemet logger dybden sammen med en tidsstempling på et SD-kort på en god måte, slik at det er enkelt å vite hvor dypt måleproben har vært til enhver tid.

RTC modulen som ble brukt, er nødvendig for at systemet skal kunne holde kontroll på tiden, men det viste seg at den hadde en avdrift som var fem ganger større enn oppgitt i databladet. Dette er imidlertid ikke noe problem i denne applikasjonen.

Referanser

[1] D. Sathyamoorthy and R. Palanikumar, "Linear and nonlinear approach for dem smoothening," *Discrete Dynamics in Nature and Society*, vol. 2006, 01 2006.

[2] "Controlling a stepper motor with an easydriver and lm555 timer |," http://www.ella-dagan.com/controlling-a-stepper-motor-with-an-easydriver-and-lm555-timer/, (Funnet 01/24/2020).

[3] D. Collins, "Microstepping for stepper motors - linear motion tips," https://www.linearmotiontips.com/microstepping-basics/, November 2017, (Funnet 21/02/2020).

[4] "Rotary encoder - wikipedia," https://en.wikipedia.org/wiki/Rotary_encoder, (Funnet 06/25/2019).

[5] "File:worm final drive (manual of driving and maintenance).jpg - wikimedia commons," https://commons.wikimedia.org/wiki/File:Worm_final_drive_(Manual_of_Driving_and_Maintenance).jpg, (Funnet 06/26/2019).

[6] B. de Bakker, "Stepper motor with drv8825 and arduino tutorial (4 examples)," https://www.makerguides.com/drv8825-stepper-motor-driver-arduino-tutorial/, 2019, (Funnet 03/06/2020).

[7] "Best braided fishing line - bc fishing journal," https://www.bcfishingjournal.com/fishing-lines/best-braided-fishing-line/, (Funnet 06/26/2019).

[8] "Eur-lex - 32000l0060 - en - eur-lex," https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0060, October 2000, (Funnet 03/28/2019).

[9] G. Hitz, F. Pomerleau, M.-E. Garneau, C. Pradalier, T. Posch, J. Pernthaler, and R. Siegwart, "Autonomous inland water monitoring design and application of a surface vessel," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 62–72, Mars 2012.

[10] OceanAlpha, "Esm30 unmanned surface vehicle, oceanalpha," https://www.oceanalpha.com/product-item/esm30/, Mai 2015, (Funnet 08/06/2020).

[11] OpenCV, "Smoothing images — opencv 2.4.13.7 documentation," https://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html, (Funnet 07/06/2019).

[12] M. Pound, "Finding the edges (sobel operator) - computerphile - youtube," https://www.youtube.com/watch?v=uihBwtPIBxM, November 2015, (Funnet 10/22/2019).

[13] ——, "Canny edge detector - computerphile - youtube," https://www.youtube.com/watch?v=sRFM5IEqR2w, November 2015, (Funnet 10/22/2019).

[14] "Opencv: Hough line transform," https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html, (Funnet 11/04/2019).

[15] Learn Enginering, "How does a stepper motor work?" https://www.youtube.com/watch?v=eyqwLiowZiU, Oktober 2016, (Funnet 01/24/2020).

[16] A. Rosebrock, "Multiple cameras with the raspberry pi and opencv - pyimagesearch," https://www.pyimagesearch.com/2016/01/18/multiple-cameras-with-the-raspberry-pi-and-opencv/, Januar 2016, (Funnet 06/27/2019).

[17] A. Rosenbrock, "Increasing webcam fps with python and opencv - pyimagesearch," https://www.pyimagesearch.com/2015/12/21/increasing-webcam-fps-with-python-and-opencv/, Desember 2015, (Funnet 05/06/2020).

[18] H. Karlsen, "Ds3231 - rinky-dink electronics," http://www.rinkydinkelectronics.com/library.php?id=73, August 2014, (Funnet 06/03/2020).

[19] Pololu, "Understanding destructive lc voltage spikes," https://www.pololu.com/docs/0J16/all, (Funnet 12/05/2019).

[20] M. McCauley, "Accelstepper: Accelstepper library for arduino," https://www.airspayce.com/mikem/arduino/AccelStepper/, April 2020, (Funnet 03/06/2020).

[21] "3d-kalkulator - geogebra," https://www.geogebra.org/3d, (Funnet 07/04/2020).

[22] "Lux light meter pro," https://apps.apple.com/us/app/lux-light-meter-pro/id1292598866, Desember 2017, (Funnet 13/05/2020).

[23] T. Bhandari, "Freshwater fish, amphibians supercharge their ability to see infrared light, washington university in st. louis," https://source.wustl.edu/2015/11/freshwater-fish-amphibians-supercharge-their-ability-to-see-infrared-light/, November 2015, (Funnet 10/06/2020).

Vedlegg

A Kildekode i Arduino

A.1 Hovedprogrammvaren til Arduino

```
1 //Including libaries
2 #include <AccelStepper.h>
3 #include <SD.h>
4 #include <SPI.h>
5 #include <DS3231.h>
6
7 const int IRsensor = 8; //IR sensor INPUT
8
9 //Nessesary definitions for steppermotor
10 #define stepPin 3
11 #define dirPin 2
12 #define motorInterfaceType 1
13 AccelStepper stepper = AccelStepper(motorInterfaceType, stepPin,
     dirPin);
14
15 unsigned long prevmillis; // To store time
16 int timesincelastupdate;
17 int lastupdate;
18
19 double depth; // Depth measurement
20 const double wanteddepth = 500; // [cm]
21
22 boolean currentstate; // Current state of IR input scan
23 boolean prevstate; // State of IR sensor in previous scan
```

```
24 boolean bottom;
25
26 //Initiating RTC and SD
27 File myFile;
28 DS3231  rtc(SDA, SCL);
29 int pinCS = 53; // Pin 10 on Arduino Uno
30 int textFileNumber;
31
32 const unsigned int numReadings = 200;
33 float depthArr[numReadings];
34 String timeArr[numReadings];
35 unsigned int j;
36
37 void setup()
38 {
39   stepper.setMaxSpeed(1000);
40
41   pinMode(IRsensor,INPUT);
42   lastupdate = 0;
43   prevstate = LOW;
44   bottom = 0;
45   Serial.begin(9600);
46
47   pinMode(pinCS, OUTPUT);
48   textFileNumber = 1;
49
50   // SD Card Initialization
51   if (SD.begin())
52   {
53     Serial.println("SD␣card␣is␣ready␣to␣use.");
54   } else
```

```
55   {
56     Serial.println("SD card initialization failed");
57     return;
58   }
59   rtc.begin();
60
61   myFile = SD.open("date.txt", FILE_WRITE);
62
63   // if the file opened okay, write to it:
64   if (myFile) {
65     myFile.print("This was recorded on: ");
66     myFile.println(rtc.getDateStr());
67     myFile.close();
68   }
69   else
70   {
71     Serial.println("error opening date.txt");
72   }
73 }
74
75 void loop()
76 {
77   currentstate = digitalRead(IRsensor); // Read IR sensor state
78   prevstate = currentstate; // store this scan (prev scan) data for
         next scan
79   lastupdate = millis();
80   timesincelastupdate = (millis() - lastupdate);
81   j = 0;
82   Serial.println(textFileNumber);
83   String x;
84   String y;
```

```
85  y = String(textFileNumber);
86  x = String(y + ".txt");
87
88  for(int i = 0; i < 100000 && timesincelastupdate < 180 && depth <
        wanteddepth; i++)
89  {
90    stepper.setSpeed(-650);
91    stepper.runSpeed();
92    timesincelastupdate = (millis() - lastupdate);
93
94    currentstate = digitalRead(IRsensor); // Read IR sensor state
95    if( prevstate != currentstate) // If there is change in input
96    {
97      depth = depth + 2.0525; // Update the depth
98      lastupdate = millis();
99      prevstate = currentstate;
100     depthArr[j] = depth;
101     timeArr[j] = rtc.getTimeStr();
102     j++;
103   }
104 }
105 while(depth < wanteddepth && prevstate == currentstate) //Means
       that the probe has hit bottom
106 {
107   stepper.setSpeed(400);
108   stepper.runSpeed();
109   currentstate = digitalRead(IRsensor); // Read IR sensor state
110   bottom = 1;
111 }
112 if(bottom == 1);
113 {
```

```
114     depth = depth - 2.0525; // Update the depth
115     lastupdate = millis();
116     prevstate = currentstate;
117     depthArr[j] = depth;
118     timeArr[j] = rtc.getTimeStr();
119     j++;
120     bottom = 0;
121   }
122
123     delay(500);
124     myFile = SD.open(x.c_str(), FILE_WRITE);
125     if (myFile)
126     {
127       for(int k = 0; k < j; k++)
128       {
129         myFile.print(timeArr[k]);
130         myFile.print("␣");
131         myFile.println(depthArr[k]);
132       }
133       myFile.close(); // close the file
134       j = 0;
135     }
136       // if the file didn't open, print an error:
137     else
138     {
139       Serial.println("error␣opening␣test.txt");
140     }
141
142     lastupdate = millis();
143     timesincelastupdate = 0;
144     for(int i = 0; i < 100000 && timesincelastupdate < 180; i++)
```

```
145     {
146     stepper.setSpeed(750);
147     stepper.runSpeed();
148     timesincelastupdate = (millis() - lastupdate);
149
150     currentstate = digitalRead(IRsensor); // Read IR sensor state
151     if( prevstate != currentstate) // If there is change in input
152     {
153       depth = depth - 2.0525; // Update the depth
154       //Serial.println(depth);
155       prevstate = currentstate;
156       lastupdate = millis();
157       timeArr[j] = rtc.getTimeStr();
158       depthArr[j] = depth;
159       j++;
160     }
161   }
162
163   myFile = SD.open(x.c_str(), FILE_WRITE);
164     if (myFile)
165     {
166       for(int k = 0; k < j; k++)
167       {
168         myFile.print(timeArr[k]);
169         myFile.print("␣");
170         myFile.println(depthArr[k]);
171       }
172
173       myFile.close(); // close the file
174
175       j = 0;
```

```
176     }
177       // if the file didn't open, print an error:
178       else {
179         Serial.println("error opening test.txt");
180       }
181   textFileNumber++;
182   delay(500);
183   depth = 0;
184   delay(1000);
185   Serial.println("Ferdig med maling");
186   //while(currentstate == prevstate) //Make a while statements,
        that waits for an update to the IR sesor, before running the
        whole main loop again.
187   //{
188   //  currentstate = digitalRead(IRsensor);
189   //}
190 }
```

B Kildekode i python

B.1 Linjedeteksjon med to kameraer

```
1 # import the necessary packages
2 from __future__ import print_function
3 from imutils.video import VideoStream
4 import numpy as np
5 import datetime
6 import imutils
7 import time
8 import cv2 as cv
9
```

```
10 # initialize the video streams and allow them to warmup
11 print("[INFO] starting cameras...")
12 cap1 = VideoStream(src=0).start()
13 cap2 = VideoStream(usePiCamera=True).start()
14
15 time.sleep(2.0)
16
17 frame_rate = 20  # define wanted framerate
18 prev = 0
19
20 # loop over frames from the video streams
21 while(True):
22     time_elapsed = time.time() - prev
23
24     frame1 = cap1.read()  # pulls one frame from cap1
25     frame2 = cap2.read()
26
27     frame1 = imutils.resize(frame1, width=400)  # resize the image
       ↪ to lower the number of pixels to work with
28     frame2 = imutils.resize(frame2, width=400)
29
30     if time_elapsed > 1./frame_rate:
31         prev = time.time()
32
33         gray1 = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)  # convert
           ↪ the image to greyscale
34         dst1 = cv.Canny(gray1,100,150, None, 3)          # apply
           ↪ Canny edge detector
35         cdst1 = cv.cvtColor(dst1, cv.COLOR_GRAY2BGR)     # make a
           ↪ copy of dst1
```

```
36          lines1 = cv.HoughLines(dst1, 1, np.pi / 180, 75, None, 0,
            ↪ 0)  # apply HoughLine transform
37
38          if lines1 is not None:
39              # use info from the HoughLine to find the endpoints of
                ↪ the line
40              rho = lines1[0][0][0]
41              theta = lines1[0][0][1]
42              a = np.cos(theta)
43              b = np.sin(theta)
44              x0 = a * rho
45              y0 = b * rho
46              x1 = int(x0 + 1000*(-b))
47              y1 = int(y0 + 1000*(a))
48              x2 = int(x0 - 1000*(-b))
49              y2 = int(y0 - 1000*(a))
50
51              # find the angle of the line
52              dx = x1 - x2
53              dy = y1 - y2
54              alpha1 = np.arctan(dx/dy) #* (360/(2*np.pi))
55              ralpha1 = round(alpha1 * (360/(2*np.pi)), 2)  # round
                ↪ it of to two decimals
56
57              # draw the line on cdst1 as well as writing the rounded
                ↪ angle
58              cv.line(cdst1, (x1,y1), (x2,y2), (0,0,255), 2,
                ↪ cv.LINE_AA)
59              font = cv.FONT_HERSHEY_SIMPLEX
60              cv.putText(cdst1, str(ralpha1),(10,100), font,
                ↪ 2,(255,255,255),2,cv.LINE_AA)
```

```
61
62          # repeat the same steps, but for frame 2
63          gray2 = cv.cvtColor(frame2, cv.COLOR_BGR2GRAY)
64          dst2 = cv.Canny(gray2,100,150, None, 3)
65          cdst2 = cv.cvtColor(dst2, cv.COLOR_GRAY2BGR)
66          lines2 = cv.HoughLines(dst2, 1, np.pi / 180, 75, None, 0,
            ↪ 0)
67
68          if lines2 is not None:
69              rho = lines2[0][0][0]
70              theta = lines2[0][0][1]
71              a = np.cos(theta)
72              b = np.sin(theta)
73              x0 = a * rho
74              y0 = b * rho
75              x1 = int(x0 + 1000*(-b))
76              y1 = int(y0 + 1000*(a))
77              x2 = int(x0 - 1000*(-b))
78              y2 = int(y0 - 1000*(a))
79
80              dx = x1 - x2
81              dy = y1 - y2
82
83              alpha2 = np.arctan(dx/dy) #* (360/(2*np.pi))
84              ralpha2 = round(alpha2* (360/(2*np.pi)), 2)
85
86              cv.line(cdst2, (x1,y1), (x2,y2), (0,0,255), 2,
                ↪ cv.LINE_AA)
87              font = cv.FONT_HERSHEY_SIMPLEX
88              cv.putText(cdst2, str(ralpha2),(10,100), font,
                ↪ 2,(255,255,255),2,cv.LINE_AA)
```

```
89
90      # show two vindows with the two camera feeds
91      cv.imshow("USB kamera",cdst1)
92      cv.imshow("PiCam",cdst2)
93
94      if cv.waitKey(1) & 0xFF == ord('q'):
95          break
96
97
98  # do a bit of cleanup
99  print("[INFO] cleaning up...")
100 cv.destroyAllWindows()
```