Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

Espen Tømmerstøl Roset
Håvard Gausemel Remøy

Design og produksjon av autonom drone

Bacheloroppgave i Automasjon og robotikk
Veileder: Erlend Magnus Lervik Coates
Medveileder: Kai Erik Hoff, Eirik Fagerhaug
Mai 2022

NTNU
Kunnskap for en bedre verden

Espen Tømmerstøl Roset
Håvard Gausemel Remøy

Design og produksjon av autonom drone

Bacheloroppgave i Automasjon og robotikk
Veileder: Erlend Magnus Lervik Coates
Medveileder: Kai Erik Hoff, Eirik Fagerhaug
Mai 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for IKT og realfag

NTNU
Kunnskap for en bedre verden

NTNU I ÅLESUND
PROSJEKTOPPGAVE

NTNU
Kunnskap for en bedre verden

Tittel:

Design og produksjon av autonom drone

Kandidatnummer (navn):

10003 (Espen Tømmerstøl Roset)

10023 (Håvard Gausemel Remøy)

Dato:	Emnekode:	Emne:	Dokument tilgang:
19/05/2022	IELEA2920	Bacheloroppgave automatisering	

Studium:	Ant. sider/vedlegg:	Bibl. nr:
Automasjon og robotikk	52 / 6	

Veileder:

Erlend Magnus Lervik Coates, Kai Erik Hoff, Eirik Fagerhaug

Sammendrag:

Denne rapporten omhandler vår utførelse av bacheloroppgave nr.25, *Design, bygging og regulering av autonom drone*. Gruppen tok utgangspunkt i et eksisterende design, tilpasset dette og produserte en drone med gode flygeegenskaper. Dronen opereres manuelt og har i tillegg autonome funksjoner som anti-kollisjon og ”return to home” som er implementert gjennom benyttelse av ROS (Robot operating system). Et stereo-kamera montert på dronen gjorde det mulig å regne ut et dybdekart av omgivelsene rundt dronen. Dette kartet kombinert med en fuzzy logic kontroller forhindrer dronen i å kollidere med omgivelsene. Gruppen er tilfredsstilt med dronens evne til å unngå kollisjoner. Det nåværende designet tillater videre utvikling av dronens ramme og funksjoner.

Innholdsfortegnelse

1 INNLEDNING **5**

2 TEORETISK GRUNNLAG **7**
2.1 Quadcopter . . . 7
2.2 Autonomi / Automatisk . . . 7
2.3 ROS - Robot operating system . . . 8
2.4 Flight controller . . . 9
2.5 Kalmanfilter . . . 9
2.6 Realsense T265 . . . 11
2.7 Disparity map . . . 11
2.8 Fuzzy logic regulering . . . 12

3 METODE OG MATERIALER **14**
3.1 Design og valg av komponenter . . . 14
3.1.1 Komponenter . . . 14
3.1.2 Ecalc . . . 15
3.1.3 Ramme . . . 16
3.1.4 Monteringsbraketter . . . 16
3.1.5 Koblingskjema . . . 17
3.2 Oppsett Flight-Controller . . . 18
3.2.1 Målinger og tilstander i Kalman filter . . . 19
3.3 ROS - Robot Operating System . . . 20
3.3.1 Systemoversikt . . . 20
3.3.2 Oppsett PC . . . 21
3.3.3 Oppsett Raspberry PI . . . 21
3.3.4 Programmering - Avstandsensorer . . . 23
3.3.5 Programmering - ControlLibrary . . . 23
3.3.6 Programmering - RTH Funksjoner . . . 25
3.3.7 Simulator . . . 26
3.3.8 Anti-kollisjonssystem . . . 28
3.4 Testing . . . 29
3.4.1 ESC og motorer . . . 30
3.4.2 Stabilitet under testflyving . . . 31
3.4.3 Flytid . . . 34
3.4.4 Posisjonpresisjon . . . 35
3.4.5 Landingspresisjon . . . 36
3.4.6 Strømforsyning Raspberry PI . . . 37

4 RESULTATER **39**
4.1 Konstruksjon . . . 39
4.2 Flyveevne . . . 40
4.2.1 Posisjonpresisjon . . . 40
4.2.2 Løftekapasitet . . . 41
4.2.3 Flyvetid . . . 41

4.3 Autonom modus 42
4.3.1 Anti-kollisjon - pålitelighet 42
4.3.2 Return to home - landingspresisjon 42
4.3.3 Return to home - pålitelighet 43

5 DRØFTING 44
5.1 Design og valg av komponenter 44
5.2 Bruk av ROS 45
5.3 Sensorer 45
5.3.1 Lidar 45
5.3.2 Qwiicc vl53l1x - Avstand 46
5.4 Video 46
5.5 Bruk av disparity map 46
5.6 Styringsystem 47
5.7 Læringsutbytte 47
5.7.1 Utviklingsprosess og prosjektstyring 47
5.7.2 CAD-Modellering og produksjon 47
5.7.3 ROS 48
5.7.4 C++ 48
5.7.5 Linux 48

6 KONKLUSJON 49

7 VEDLEGG 50

SAMMENDRAG

Denne rapporten omhandler vår utførelse av bacheloroppgave nr.25, *Design, bygging og regulering av autonom drone*. Gruppen tok utgangspunkt i et eksisterende design, tilpasset dette og produserte en drone med gode flygeegenskaper. Dronen opereres manuelt og har i tillegg autonome funksjoner som anti-kollisjon og ”return to home” som er implementert gjennom benyttelse av ROS (Robot operating system). Et stereo-kamera montert på dronen gjorde det mulig å regne ut et dybdekart av omgivelsene rundt dronen. Dette kartet kombinert med en fuzzy logic kontroller forhindrer dronen i å kollidere med omgivelsene. Gruppen er tilfredsstilt med dronens evne til å unngå kollisjoner. Det nåværende designet tillater videre utvikling av dronens ramme og funksjoner.

SUMMARY

This report presents our solution for bachelor task nr.25, *Design, building, and regulation of an autonomous drone*. An existing frame design was modified and produced with reliable flying capabilities. The drone is operated manually with assistance from an anti-collision system and a ”return to home” function. These functionalities have been implemented using ROS (Robot operating system). A dept-map of the environment is generated by utilizing the on-board stereo-camera. This depth-map is combined with a fuzzy logic controller to avoid collisions with objects in front, above or beneath the drone. The group is satisfied with the drones ability to avoid collisions on its course. The presented solutions allows for further development of the drones frame and functions

1 INNLEDNING

Droner er allerede en stor del av samfunnet vårt i dag, og vil bli større i fremtiden.[12][16] De har flere samfunnsnyttige bruksområder som for eksempel leveranser av medisin, søk -og redning, filmproduksjon og landbruk. Droner kan operere i fiendtlige eller utilgjengelige miljøer som kan utgjøre en fare for mennesker. Samtidig kan droner benyttes i vann, på land og i luften. Droner er som regel førerløse og blir styrt med fjernstyring eller med et autonomt system. Mange assosierer ordet ”drone” med et quadcopter. Et quadcopter benytter fire propeller til å fly i luften. Utvikling av autonome styringsystemer til disse dronene kan bidra til at de blir lettere å operere og at oppgaver som nevnt over i større grad kan utføres uten menneskelige inngrep.

I dette prosjektet var formålet å utforske muligheten for å benytte tilgjengelige materialer på campus til å bygge en stabil drone som kan fly innendørs. I tillegg ville gruppen jobbe mot utvikling av et semi-autonomt styringsystem for dronen. Styringsystemet skulle ha to moduser. Et modus der dronen styres manuelt ved hjelp av en joystick. I dette moduset vil operatøren få hjelp av styringsystemet for å unngå kollisjoner. Det andre moduset ble kalt ”return to home”. Når dette moduset aktiveres vil dronen følge banen som ble kjørt i manuell modus og returnere til punktet der flyvingen startet. Samtidig som dette pågår vil systemet være oppmerksom på nye hindringer og unngå å kollidere i disse. Det skulle også tilrettelegges slik at det ble mulig å bygge videre på styringsystemet for andre i fremtiden.

Med semi-autonomt styringsystem menes det at systemet vil ha forskjellig grad av autonomi avhengig av hvilket modus dronen opererer. For å sammenligne med SAE J3010[11] standarden for autonome kjøretøy vil dronen ha SAE Nivå 2 under manuell kontroll og SAE Nivå 4 under autonom flyving. En forklaring av disse begrepene kommer i kapittel 2.2

Relatert Arbeid

[3] Thomas Ian Grayston. “A Simplified Fuzzy-Logic Control System Approach to Obstacle Avoidance combining Stereoscopic Vision and Sonar”. PhD thesis. Honours thesis, University of Tasmania, 2006.

[8] Aggrey Shitsukane. “Fuzzy Logic Sensor Fusion for Obstacle Avoidance Mobile Robot”. In: Aug. 2018.

[10] Suat Karakaya and Hasan Ocak. “Fuzzy logic-based moving obstacle avoidance method”. In: *Global Journal of Computer Sciences: Theory and Research* (2019).

Gruppens bidrag kan oppsumeres med:

- Produsert CAD modell av drone som lar seg 3D-printe / Laserskjære.
- Produsert drone som flyr stabilt og som kan brukes videre i fremtiden.
- Programmert ROS node for konvertering av Xbox input til en datatype som kan sendes direkte til PX4 flight controller for manuell kontroll.
- Undersøkt og funnet nyttige ROS noder.
- Forbedret ROS node for utregning av disparity map.
- Forbedret qwiic bibliotek, lagt til funksjonalitet for valg av i2c bus.
- Programmert ROS node for anti-kollisjon som benytter disparity map og fuzzy-logic til å gi riktige roll/pitch/yaw pådrag slik at kollisjon blir forhindret i sanntid.
- Satt sammen komplett ROS miljø til operasjon av drone som kan videre utvikles.
- Produsert komplett prototype som beviser at en drone kan produseres på campus, med sanntids anti-kollisjon system.

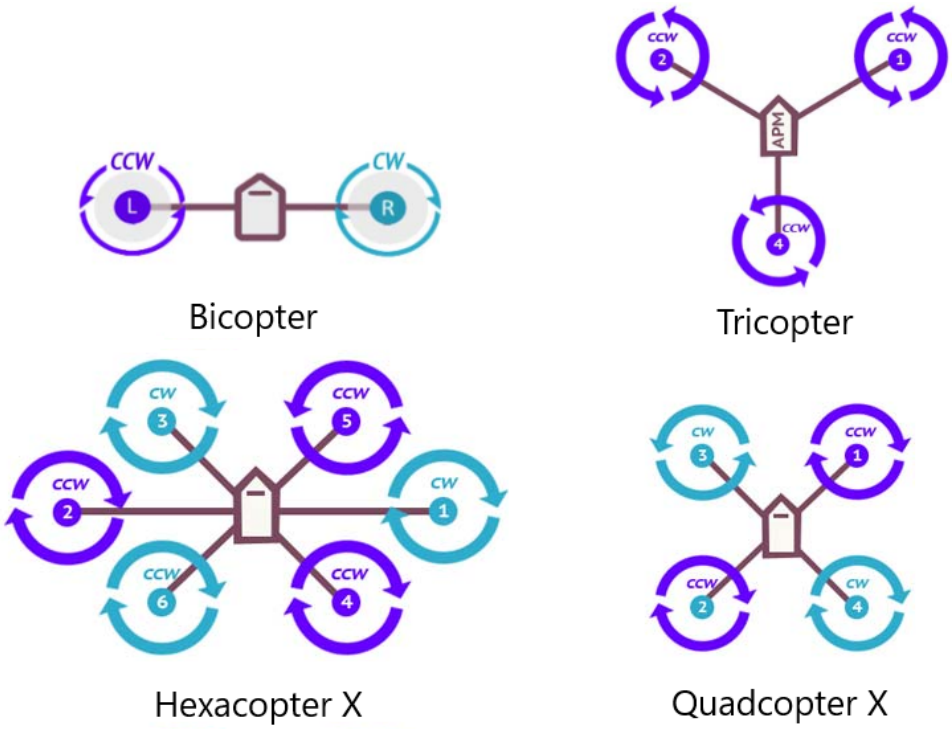
Resten av rapporten er bygd opp som følger:

- **Teoretisk grunnlag:** Dette kapittelet beskriver de teoretiske konseptene som leseren bør forstå før resten av rapporten leses.
- **Metode og materialer:** Dette kapittelet beskriver hvordan gruppen utførte prosjektet og hvilke metoder og materialer som ble brukt.
- **Resultater:** Dette kapittelet beskriver resultatene som ble oppnådd i prosjektet.
- **Drøfting:** Dette kapittelet beskriver gruppens meninger om resultatet, hva som kunne blitt gjort annerledes og hvilke området som kan bli forbedret.
- **Konklusjon:** Dette kapittelet konkluderer rapporten.

2 TEORETISK GRUNNLAG

2.1 Quadcopter

Det eksisterer i dag mange forskjellige typer droner. I figur 1 kan man se eksempel på fire ulike typer. Den tydeligste forskjellen mellom disse er antall propeller. Propellene kan også ha ulik plassering i forhold til rammen. Antall propeller og plasseringen av disse vil påvirke pris, prestasjon, stabilitet, og flytid. Quadcopteret er ansett for å best balansere disse egenskapene. I denne oppgaven bygges det en drone av typen quadcopter med propellene plassert i en X-formasjon som illustrert i figur 1.



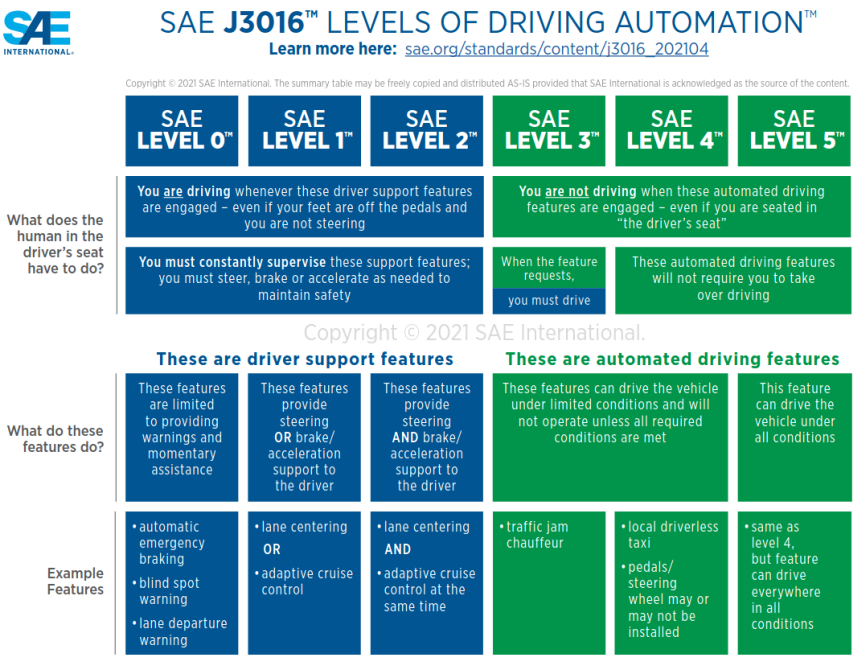
Figur 1: Forskjellige typer droner, hentet fra [14]

2.2 Autonomi / Automatisk

En autonom enhet eller innretning refererer til en enhet som kan utføre oppgaven sin uten menneskelige inngrep. En autonom enhet kan reagere på endringer i miljøet uten menneskelig intervensjon. Etter at autonome biler har komt på markedet kom det internasjonale forbundet SAE med en standard for forskjellige nivå av autonome kjøretøy. En grafisk fremstilling av standarden vises under i figur 2. Som nevnt i innledningen vil dronen i dette prosjektet ha SAE nivå 2 under manuell kontroll. Systemet får dette nivået siden dronen vil hjelpe operatøren og gir pådrag både på styringen og

hastigheten til dronen. I det autonome moduset vil systemet ha SAE nivå 4. Det vil ha dette nivået siden operatøren ikke trenger å styre dronen, men det vil kun fungere i gitte omgivelser (nærmere forklart i kapittel 4.3.1).

I motsetning til autonom trenger en automatisk enhet en grad av menneskelig intervensjon. I vårt tilfelle er en automatisk funksjon på dronen at den tar av til en gitt høyde etter at operatøren trykker på en knapp. Etter at operatøren trykker på knappen, gjør dronen resten av arbeidet uavhengig av operatøren.



Figur 2: SAE J3016, hentet fra [11]

2.3 ROS - Robot operating system

ROS (Robot operating system) er et open-source rammeverk for programmering av roboter. ROS inneholder mange pakker og biblioteker som forenkler programmering av roboter og gjør at man raskere kan utvikle et produkt. En prosess som kjører i ROS systemet blir kalt en node. Hver node kan kommunisere med hverandre gjennom meldinger kalt topics. Så typisk vil en node motta noe informasjon fra en annen node og bruke dette til å gjøre en beslutning eller oppgave. Når en node sender informasjon til et topic kalles det at noden publiserer til topicen, dersom en node skal motta informasjon fra et topic må noden abonnere på topicen. Så en topic kan bli sett på som en buss der meldinger blir publisert og abonnert på. En node kan også gjøre services for andre noder eller be om services fra andre noder. En service referer til en jobb noden kan

gjøre som for eksempel lese av en sensorverdi. Alle nodene i systemet er registrert hos en master, masteren legger til rette for peer to peer kommunikasjon mellom nodene.[5]

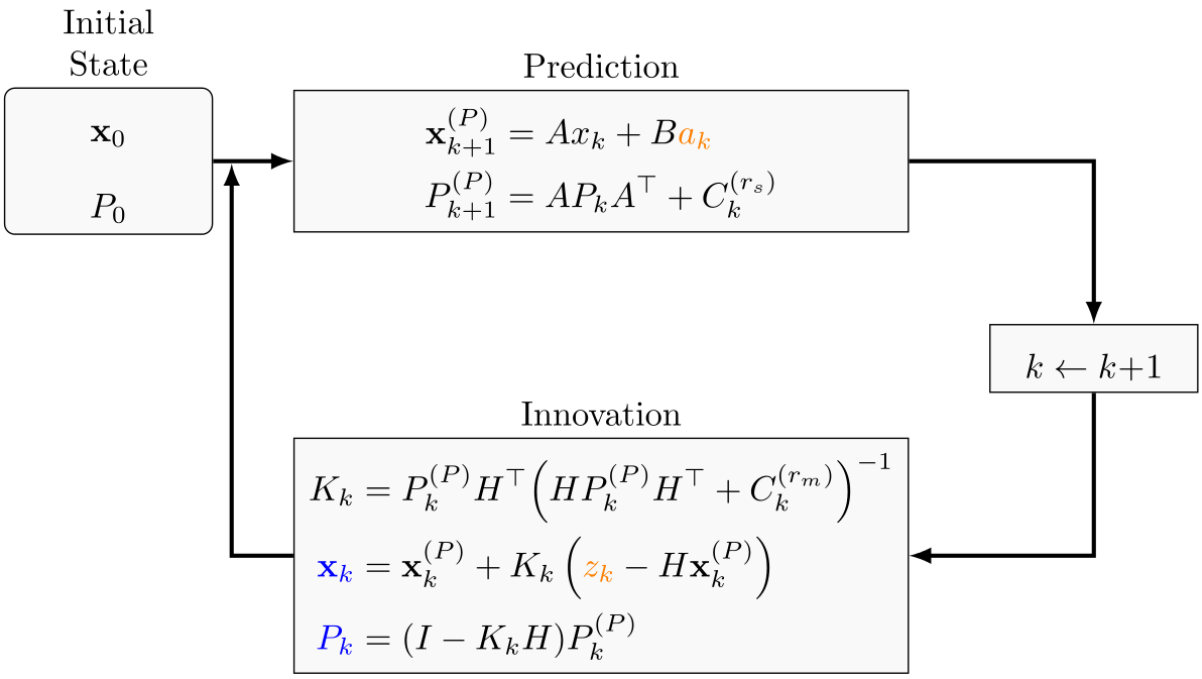
2.4 Flight controller

En flight controller (FC) er et kretskort med varierende kompleksitet som ofte blir referert til som ”hjernen” av en drone. På et quadcopter er funksjonen til en FC å bestemme rotasjonshastigheten til hver av motorene slik at den holder seg stabil i luften. I tillegg til dette skal den sørge for at dronen oppfører seg i henhold til de kommandoene som blir sendt til den. De fleste FC’er i dag kommer med forskjellige funksjoner som for eksempel å holde en låst orientering eller holde en spesifikk høyde. Mange FC blir ofte levert med integrerte sensorer som kompass og barometer. De mer kompliserte FC har i tillegg til integrerte sensorer egenskapen til å benytte eksterne sensorverdier. Denne egenskapen åpner opp muligheten til å implementere egne funksjoner utover det som er standard. Noen av de mest velkjente produsentene av flight controllerer og tilhørende firmware i dag er: Pixhawk, PX4 og Ardupilot. PX4 firmwaren er spesielt rettet mot autonom flyving. [9]

2.5 Kalmanfilter

I flight controlleren blir kalmanfilter brukt for å estimere de forskellige tilstandene til dronen. Kalmanfilter er en filteralgortime som gir et estimat for en eller flere tilstander i et system, og disse estimatene er optimale. Algoritmen er relativt enkel og det kreves ikke så mye datakraft for å gjennomføre filtreringen, dette gjør algoritmen til et godt alternativ i dette tilfellet.

Filteret gjør en iterativ prosess der det er to hovedoperasjoner som blir gjennomført i hver iterasjon. Den første operasjonen er prediksjon (a priori) i denne operasjonen blir tilstandene estimert basert på tidligere verdier. Den neste operasjonen er korrigering (a posteriori) i denne operasjonen blir sensorverdier brukt til å korrigere de predikerte estimatene. Når filteret skal estimere flere tilstandsvariabler blir utregningene gjort ved hjelp av matriseregning. [1]



Figur 3: Kalmanfilter algoritme, bilde hentet fra [6]

Forklaring av variabler:

- $x_{k+1}^{(P)}$ er den estimerte tilstanden som regnes ut før man foretar en måling. Det er gitt at man har forrige verdi eller en initiell verdi for den aktuelle tilstanden.
- A er matrisen som beskriver systemet og fungerer som en modell man bruker til å finne den neste verdien til en tilstand.
- $P_{k+1}^{(P)}$ matrisen beskriver usikkerheten i modellen og estimatene.
- $C_k^{(r_s)}$ matrisen er kovariansen av prosessstøy, denne matrisen bestemmer hvor mye modellen kan endre seg fra et tidssteg til neste.
- K_k er "kalman gain". Denne verdien bestemmer hvor stor innflytelse målingene z_k og estimatene fra modellen $x_{k+1}^{(P)}$ har på det endelige estimatet x_k. Dersom målingene er unøyaktige (har stor varians) vil det endelige estimatet basere seg mer på estimatene fra modellen, dersom målingene er mer nøyaktig vil de endelige estimatene basere seg mer på målingene.
- H matrisen brukes til å gjøre om målingene slik at de passer med modellen. Slik at rett måling blir sammenlignet med rett tilstands variabel i modellen.
- $C_k^{(r_m)}$ matrisen inneholder variansen til målingene, den beskriver altså hvor nøyaktige sensorene er.
- x_k er den estimerte tilstandsvariabelen korrigert ved hjelp av målingene.

- z_k er en måling av tilstandsvariabelen.
- P_k er den korrigerte usikkerheten i modellen og estimatene.

2.6 Realsense T265

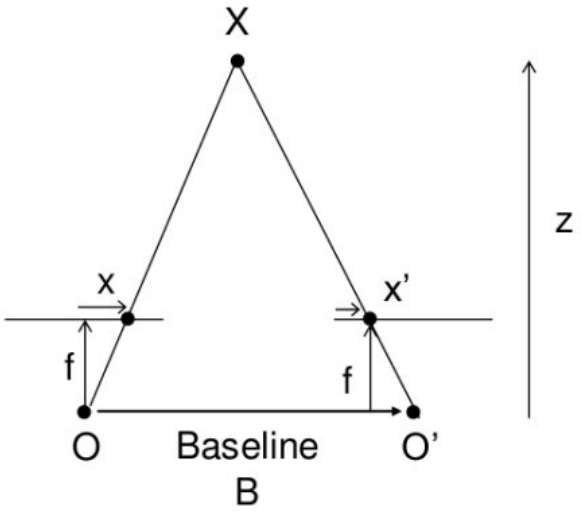
Intel Realsense T265 er kameraet som blir benyttet på dronen. Det brukes både til å gi brukeren et førstepersjonsperspektiv fra dronen og til å finne den relative posisjonen til dronen fra start. Enheten består av to vidvinkel(163± 5°) kamera, og en IMU som inneholder akselerometer og gyroskop. I tillegg inneholder den også Intels Movidius Myriad 2 VPU(Video Processing Unit) som er spesialbygd for bildebehandling og maskinsyn med høy ytelse per watt.

Det er hovedsaklig bildene fra de to kameraene som blir brukt for å finne posisjonen til kameraet. Måten bildene blir brukt på er at det først blir funnet et punkt i hvert bilde som korresponderer til hverandre. Man sammenligner koordinatene til punktet i hvert bilde og ser på forskjellen mellom koordinatene til punkta. Ved å se på denne forskjellen kan man finne ut hvor punktet ligger i rommet i forhold til kameraet. Videre finner kameraet slike punkt som ikke beveger seg over tid, for eksempel hjørnet i rommet. Kameraet kan da bruke desse punkta til å beskrive sin egen relative posisjon i forhold til dem.

Det er en SLAM algoritme som utfører de operasjonene nevt i det forrige avsnitt. SLAM står for ”simultaneous localization and mapping”[15]. Algoritmen kjøres lokalt på kameraet. IMUen blir brukt til å støtte og korrigere estimatene fra bildene. Dette gjør at kameraet klarer å finne sin relative posisjon med mindre enn 1% drift. [18]

2.7 Disparity map

I maskinsyn er misforhold (disparity) avstanden mellom korresponderende punkt i to forskjellige bildeplan. I et stereo kamera oppsett kan man finne denne avstanden for alle punkter i snittet av synsvinklene til kameraene. Satt sammen vil disse verdiene utgjøre et disparity map. Et disparity map kan videre benyttes til å estimere avstand. Figur 4 kan antas å være et stereo kamera oppsett hvor punktet X prosjekteres til to ulike origoer O og O' og produserer et punkt i hver av bildeplanene. Bildeplanene vil i praksis være sensorene til kameraet. Prosjekteringen gir punktene x og x' som er pixelkordinater i sensorene. Med figuren i tankene kan man tenke seg hvordan punktene x og x' beveger seg etterhvert som punkt X beveger seg vekk fra planet. Om X flytter seg uendelig langt vekk vil x og x' være plassert i senter av begge bildeplanene. I dette tilfellet sier man at graden av disparity er lav. Dersom X heller flyttes nærmere vil x og x' sin plassering i bildeplanene bli ulike i en større grad, og man har en høy grad av disparity.



Figur 4: Disparity for punkt X, bilde hentet fra [19]

$$disparity = x - x' = \frac{Bf}{Z} \tag{1}$$

Likning 1 viser at det er mulig å løse for avstanden Z gitt at grunnlinjen B og brennvidden f er kjent. Gjennom enkle matriseoperasjoner kan man derfor enkelt konvertere verdiene i et disparity map om til avstand:

$$Z = \frac{Bf}{disparity} \tag{2}$$

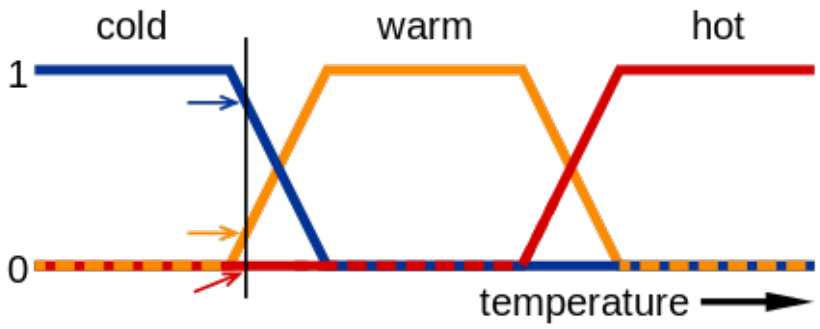
2.8 Fuzzy logic regulering

Fuzzy logic i motsetning til boolean logic (crisp logic) kan ha en varierende grad av sannhet. En fuzzy logic kontroller er godt egnet til å ta beslutninger basert på uklare inputverdier. For å ta beslutninger benytter en fuzzy controller definerte regler og medlemsfunksjoner. Et eksempel på to regler er :

'IF temperature IS hot THEN fan IS fast'
'IF temperature IS warm THEN fan IS normal'

I fuzzy logic er det vanlig å normalisere tallverdier mellom 0 og 1. Med figur 5 som eksempel kan man anta at 'temperature' er definert til å være et sted mellom 0 og 100. Ordene "cold", "warm", og "hot" beskriver inngangsvariabelen "temperature". For eksempel kan warm være 40-60 og hot 70-100. Dersom temperaturen da er 65, vil fuzzy kontrolleren se på temperaturen som "0.5 warm" og "0.5 hot". Med reglene gitt

over vil det da være naturlig for en fuzzy kontroller at fan skal gå med en hastighet mellom fast og normal. [2]



Figur 5: Eksempel av en medlemsfunksjon, bilde hentet fra [4]

3 METODE OG MATERIALER

I dette kapittelet beskriver vi hvordan vi gikk fram i utviklingsprosessen, både når det kommer til utførelse og hvilken programvare og materialer som ble brukt.

3.1 Design og valg av komponenter

3.1.1 Komponenter

Det første som ble gjort i design prosessen var å sette opp flere krav og spesifikasjoner som det var ønskelig at dronen skulle oppfylle. Spesifikasjonene var som følger:

- Flyvetid: 15-20min
- Størrelse: 330mm
- Thrust/Vekt ratio: >2
- Payload: Raspberry Pi4, Intel Realsense T265, lidar vurdert

Etter kravspesifikasjonen ble satt startet gruppen med å finne et overslag på hva totalvekten til dronen ville være. Da ble det undersøkt hva som er vanlig vekt på de forskjellige komponentene som trengs for å bygge en drone. Det ble satt opp et vektbudsjett og totalvekten ble funnet til å bli ca 830gram. Budsjettet vises i figur 6

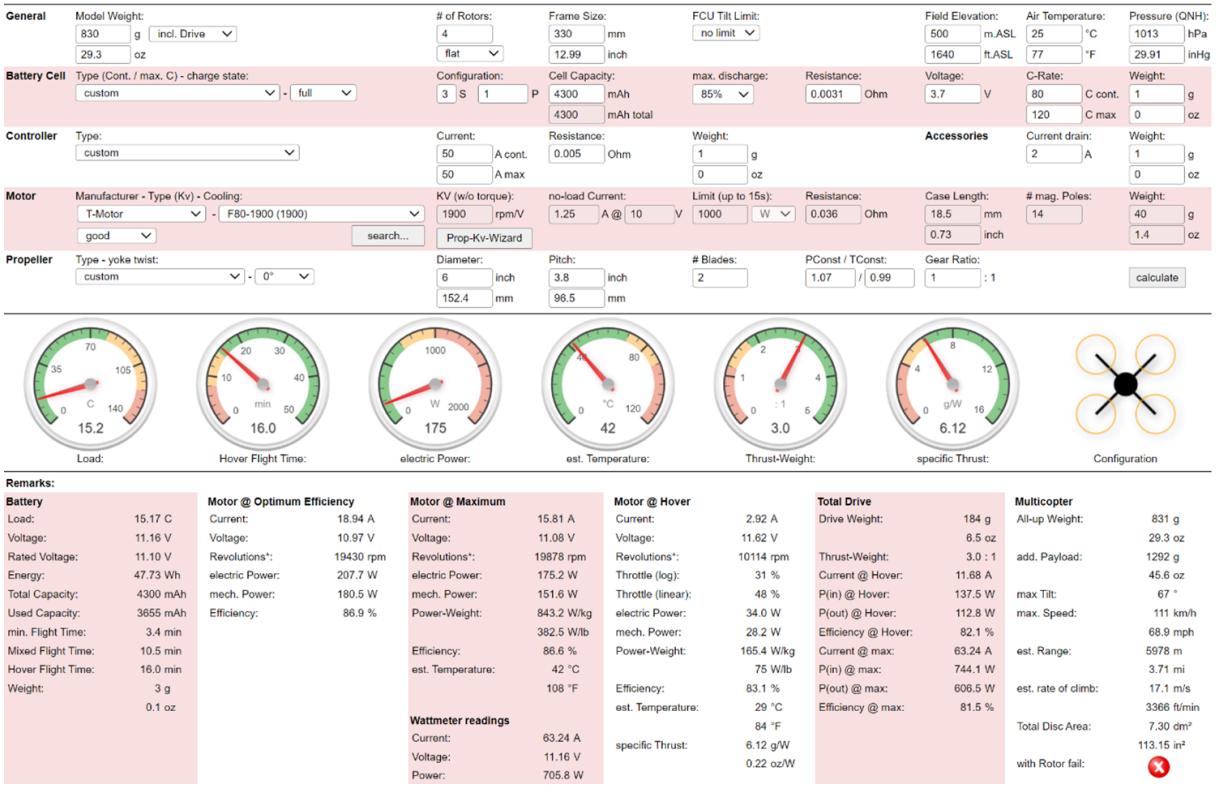
Komponent	Pris [kr]	Vekt [g]	Må bestilles	Leveringstid
Pixhawk 4	0,00	15,80	Nei	0dg
Raspberry Pi 4B 8GB	619	46	Ja	28.01.2022
Intel RealSense T265	0	55	Nei	0dg
3D-Print Ramme (Est.)	0	250	Nei	0dg
3s Lipo 4300mAh 50C x2	1198	244	Ja	1-5dg
4x Motor 1900KV	1156	158	Ja	1-5dg
ESC 4in1	649	20	Ja	1-5dg
Propeller 6x3.8 EMAX x5	295	4,6	Ja	1-5dg
BEC x5	120	1	Ja	1-5dg
Bronto Lipo-Safe Bag	99		Ja	1-5dg
VL53L1X avstandssensor x2	438	5	Ja	1-2dg
Frakt Elefun	115			
3D-Print plastikk*	249		Nei	
Akrylplate til laserskjærer**	70		Nei	
Div. festemateriell	129,9	30	Nei	
Loddetråd	99,9		Nei	
Sum	5 237,80	829,40		
*Pr. spole				
**Pr. plate				

Figur 6: Vekt og pris budsjett

Når den totale vekten er estimert må man finne en kombinasjon av motor, propell og batteri som gir nok thrust til formålet. For at dronen skal ha gode flyegenskaper er det ønskelig at thrust/vekt ratioen er slik at ved omtrent 50% pådrag vil dronen sveve i stillstand i luften.

3.1.2 Ecalc

Vurdering av komponenter ble gjort ved hjelp av Ecalc. Ecalc er en online simulator som estimerer flyegenskaper til en drone basert på egenskapene til benyttede komponenter. Simulatoren er i følge nettsiden deres benyttet av større selskaper som BOEING og BAE SYSTEMS. I figur 7 vises utregningene med de komponentene som ble valgt. Simulatoren estimerer at dronen får en Thrust-Weight ratio på 3 og flyvetid på 16min.



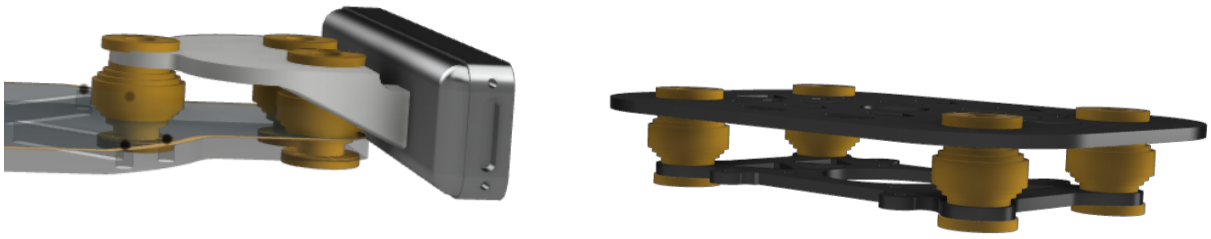
Figur 7: Beregninger for fastbestemte komponenter

3.1.3 Ramme

Til å designe rammen ble det tatt utgangspunkt i et eksisterende design fra DJI [7]. Endringer ble gjort i dette designet med hensyn på plassering av komponenter og redusering av vibrasjoner. Kroppen ble skjært ut i 4mm akrylplater med laserkutter, og armene er 3D-printet med PLA filament. En sammenligning av orginal og ny rammer er vist i figur 9.

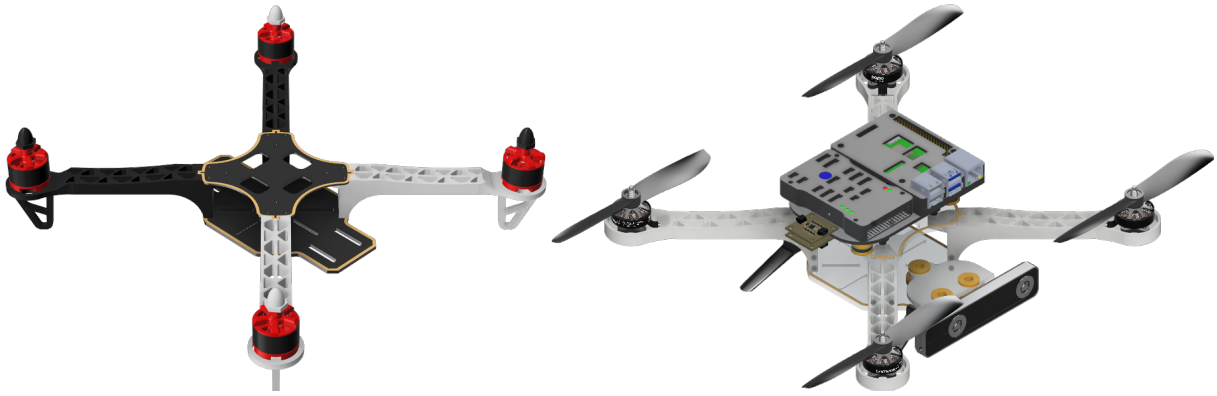
3.1.4 Monteringsbraketter

Monteringsbrakettene for T265 kameraet og Raspberry PI ble designet med hensyn på å dempe vibrasjoner. For å dempe høyfrekvente vibrasjoner ble det benyttet gummiballer som festemateriell. Figur 8 illustrerer de endelige løsningene.



(a) Brakett til kamera (b) Brakett til FC og PI

Figur 8: Løsning for vibrasjonsdemping

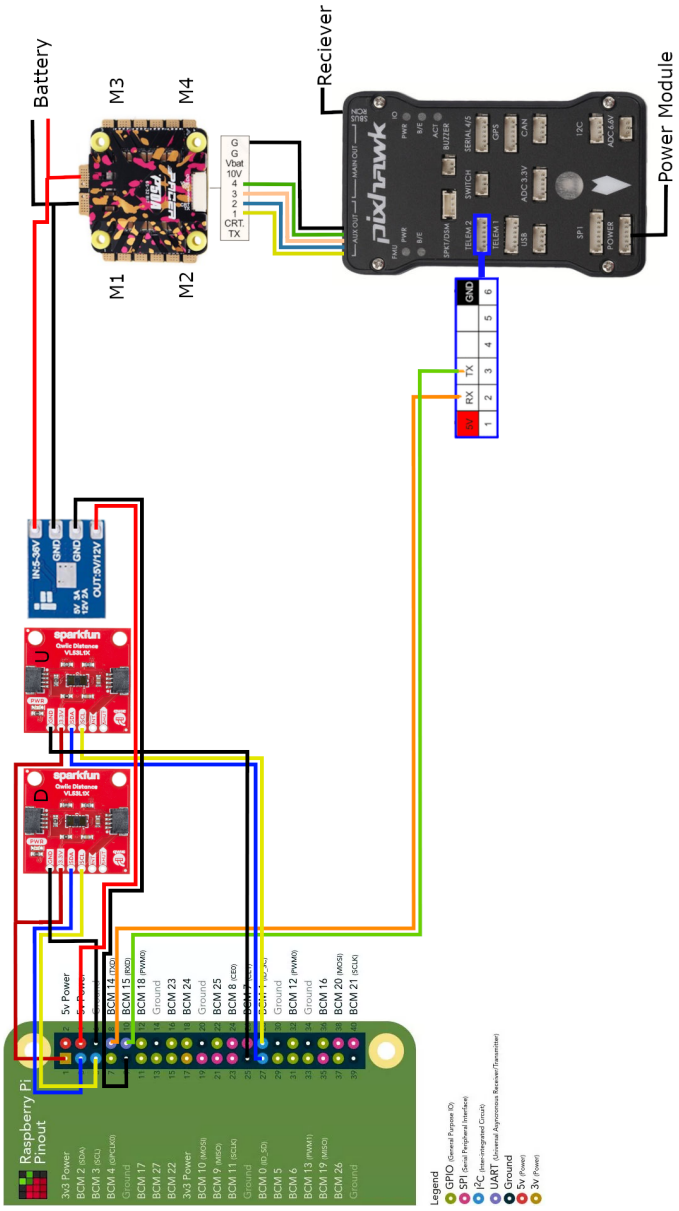


(a) Orginal (b) Ny

Figur 9: Sammenligning av orginalt og nytt design

3.1.5 Koblingskjema

Koblingskjema for dronens styringsystem vises i figur 10



Figur 10: Koblingskjema for dronen

3.2 Oppsett Flight-Controller

For å sette opp flight-controlleren benyttet vi programmet QGroundcontrol. Dette programmet gjør det mulig å konfigurere alle parametre på flight-controlleren. De fleste parametrene blir satt automatisk når man kjører de forskjellige kalibrerings-sekvensene. Andre parametrene som må justeres manuelt er:

- **MAV_1_CONFIG = TELEM 2**

  - Dette parameteret forteller pixhawken at det skal opprettes en kommunikasjonskanal med Raspberry PI på TELEM2 porten.

- **EKF2_AID_MASK = 280**
  - Dette parameteret bestemmer hvilken sensordata som skal fusjoneres i kalmannfilteret. Når denne er satt til 280 vil det si at posisjon, yaw vinkel og hastighet fra realsense kameraet skal brukes.

- **EKF2_EV_X = 0.095**
  - Dette parameteret setter den relative posisjonen til kameraet i forhold til flight-controlleren. Det finnes tilsvarende parameter for y og z-retning, i vårt tilfelle ble desse satt til henholdsvis 0.025 og -0.065.

- **EKF2_HGT_MODE = VISION**
  - Dette parameteret bestemmer hovedkilden til høgdeinformasjonen som skal brukes i kalmannfilteret. I vårt tilfelle ble denne satt til VISION, dette vil si at den bruker informasjon fra realsense-kameraet.

- **SYS_USE_IO = 0**
  - For å bruke Dshot protokollen som vår ESC bruker er det nødvendig å sette dette parameteret til null. Ved å sette denne til null blir pwm utgangene deaktivert og man bruker portene merket AUX istedet. AUX portene har støtte for å kommunisere med Dshot protokollen.

- **DSHOT_CONFIG = DShot600**
  - Denne parameteren aktiverer DShot og tallet bestemmer overføringshastighet, 600 = 600kb/s. Her er det viktig å merke seg at ikke alle ESCer støtter alle overføringshastigheter.

- **MAV_ODOM_LP = 1**
  - Denne parameteren aktiverer loopback av odometri dataen. Dette kan brukes til å sjekke at kalmanfilteret finner riktig posisjon og klarer å bruke dataen fra realsense-kameraet.

3.2.1 Målinger og tilstander i Kalman filter

Med de parameterene som ble satt i kapittelet over har vi valgt hvilke målinger som skal brukes i kalman filteret. Nedenfor blir målingene og tilstandene i kalmanfilteret listet opp. I tillegg til de interne sensorene i flight controlleren blir data fra T265 kameraet benyttet. Fra kameraet sendes posisjon, hastighet og orientering til flight controlleren, i tillegg sendes også usikkerheten til kameraets målinger. [17]

Målinger

- IMU - Vinkel, vinkelhastighet og akselerasjon i xyz-retning
- Magnetometer - xyz magnetometer data brukes som separate målinger
- Høyde - Høyde målinger fra T265 kamera.
- Eksternt maskinsyn - Posisjon, hastighet og vinkel målinger fra T265 kamera.

Tilstander

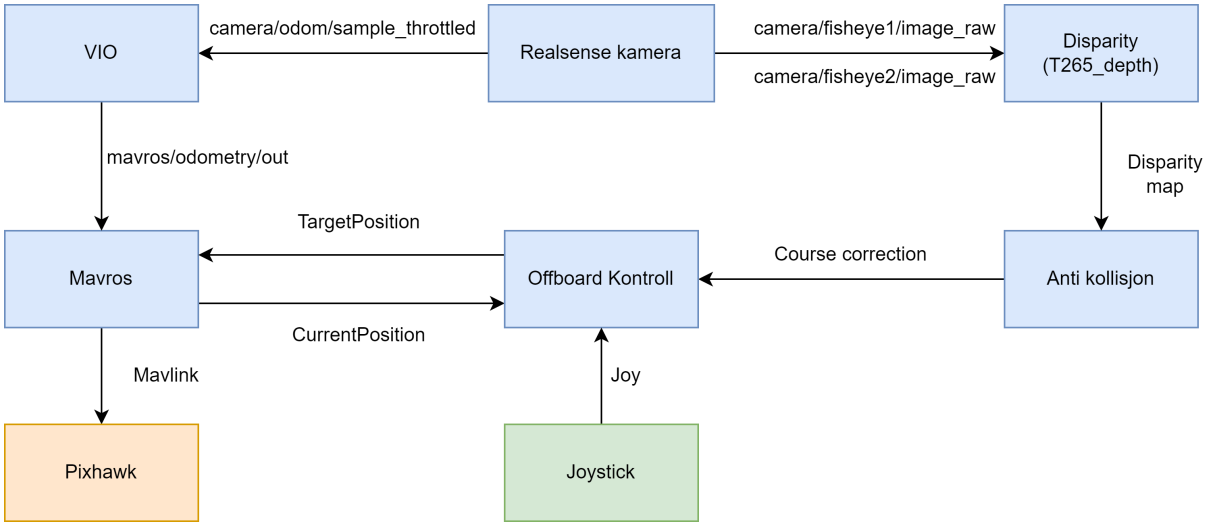
- Kvaternioner
- Hastighet, xyz [m/s]
- Posisjon, xyz [m]
- IMU delta vinkel bias, xyz [rad]
- IMU delta hastighet bias, xyz [m/s]
- Jordens magnetiske felt [gauss]
- Body magnetisk felt [gauss]
- Wind hastighet NE [m/s]

3.3 ROS - Robot Operating System

Utviklingen av styresystemet til dronen er basert rundt ROS. I dette kapittelet beskrives hvordan gruppen benyttet ROS, hvilke pakker som ble brukt, og hvilke som ble utviklet av gruppen. De sentrale prosessene som utgjør dronens styresystem vil også bli forklart i dette kapittelet.

3.3.1 Systemoversikt

I figur 11 er oversikten over de viktigste nodene i ROS miljøet og meldingene som blir sendt mellom den. Nodene er fargekodet, fargen representerer hvilken enhet de kjøres på. Blå kjøres på Raspberry Pi, grønn kjøres på PC og oransje kjøres på flight controlleren.



Figur 11: Systemoversikt ROS

3.3.2 Oppsett PC

For å sette opp ROS på pc-en er det nødvendig å først installere ubuntu 20.04 operativsystem. Det er flere måter dette kan gjøres på. De metodene vi har testet som fungerer bra er å enten lage en ny partisjon på harddisken på pcen og installere ubuntu her, eller installere ubuntu på en usb minnepenn. Begge metodene fungerer, men førstnevnt er anbefalt da det er raskere og mer stabilt. Etter å ha installert ubuntu 20.04 kan ROS settes opp ved å følge instruksjonene i dokumentasjonen deres, http://wiki.ros.org/noetic/Installation/Ubuntu.

Kommunikasjonen mellom ROS på PC og Rapsberry PI kan enkelt settes opp når det er på samme lokalnett. Dette gjøres ved å følge disse instruksjonene fra ROS dokumentasjonen, http://wiki.ros.org/ROS/Tutorials/MultipleMachines. I dette tilfellet kjører ROS Masteren på Raspberry Pien så da kan vi koble oss til den på PCen ved å kjøre kommandoen: export ROS_MASTER_URI=http://RPI-IP:11311, der man bytter ut RPI-IP med IP-adressen til raspberry pi. Man må også på begge maskinene legge til ip-adressen til den man vil kommunisere med i filen /etc/hosts. Når dette er gjort vil man på PCen ha tilgang til alle ROS noder, topics, services og actions som er tilgjengelig på Raspberry Pien.

3.3.3 Oppsett Raspberry PI

For å sette opp et ROS miljø på Raspberry PI ble det nødvendig å installere ubuntu 20.04 som operativsystem istedet for Raspian OS. Ubuntu 20.04 kan enkelt installeres ved å først laste det ned fra ubuntu sine sider, https://releases.ubuntu.com/20.04/, og deretter bruke raspberry imager for å lage en ”bootable” usb minnepenn. Når ubuntu

20.04 er installert kan man enkelt følge oppskriften som ligger i ROS sin dokumentasjon, http://wiki.ros.org/noetic/Installation/Ubuntu . Etter å ha satt opp ROS miljøet er dette de forskjellige pakkene man trenger for å kommunisere med realsense kameraet og pixhawk flight-controlleren:

- **mavros**
  - Denne pakken håndterer kommunikasjon mellom ROS på Raspberry PI og Flight-Controlleren. Pakken henter informasjon fra Flight Controlleren som for eksempel odometri data, og publiserer dette som topics i ROS. I tillegg blir det også mulig å publisere til topics som for eksempel mavros/setpoint_position/local, dette gjør det mulig å sende settpunkt til Flight Controlleren fra ROS. Pakken kan settes opp ved å følge instruksjonene på github siden, https://github.com/mavlink/mavros.
- **realsense2-camera**
  - Pakken fra realsense håndterer alt som har med realsense kameraet å gjøre. Den gjør at man i ROS kan hente ut bilde fra begge kameralinsene og den relative posisjonen til kameraet. Denne pakken settes også opp ved å følge instruksene gitt på github siden, https://github.com/IntelRealSense/realsense-ros#installation-instructions.
- **Auterion/VIO**
  - Denne pakken tar posisjonsdataen som realsense kameraet produserer, transformerer den til riktig koordinatsystem og sender det deretter til flight-controlleren. Pakken tar utgangspunkt i at kameraet er montert med linsen pekende nedover, og at kameraet er sentrert. I vårt tilfelle når kameraet er montert i front og translatert i forhold til flight-controlleren ble vi nødt til å modifisere ”launch” filen. Dette er den orginale filen:

```
<node pkg="tf" type="static_transform_publisher" name="tf_baseLink_cameraPose"
    args="0 0 0 0 1.5708 0 base_link camera_pose_frame 1000"/>
```

Figur 12: Orginal launch fil

Her er det de seks første argumentene som er interessante, de tre første er translasjon i xyz-retining og de tre siste er rotasjon i forhold til flight-controlleren. Så i vårt tilfelle vil vi ha argumentene args=”0.095 0.025 -0.065 0 0 0”. Pakken settes opp ved å følge instruksene på github siden, https://github.com/Auterion/VIO. Merk at ROS Noetic ikke er på listen over kompitable ROS versjoner, men etter at vi bygget pakken fra kildekoden hadde vi ingen problemer med å kjøre den.

3.3.4 Programmering - Avstandsensorer

En av pakkene som vi utviklet selv i ROS-miljøet var pakken som håndterte avlesning av sensordata fra to avstandsensorer som måler avstand til tak og gulv. ROS pakken fungerte slik at ved å starte to noder så ville måleverdien fra den nedre sensoren publiseres på topic ”distance_down”, og verdier fra sensoren som pekte opp ble publisert på ”distance_up” topic .

Sensorene var av typen Sparkfun-VL53L1X, på sparkfun sine sider kan man finne et bibliotek som dekker alle deres sensorer kalt ”sparkfun-qwiic”. Vi benyttet oss av dette biblioteket, men ble nødt til å skrive om deler av det for at det skulle fungerer slik som vi ville. Dette var på grunn av vi benyttet en egen i2c bus for hver av sensorene og når man benyttet biblioteket støttet det kun å bruke standard bussen på raspberry pi (/dev/i2c-1). Endringene vi utførte i hver fil er som følger:

- /.local/lib/python3.8/sites-packages/qwiic_i2c/linux_i2c.py :
  - I klassen LinuxI2C ble det lagt til et ekstra argument i konstruktøren kalt BusId. Verdien til argumentet blir lagret i en ny variabel kalt self.BusId. Videre på linje 135 må self.BusId legges inn som argument i funksjonskallet slik som dette _connectToI2CBus(self.BusId).
- /.local/lib/python3.8/sites-packages/qwiic_i2c/__init__.py :
  - I funksjonen getI2CDriver må det legges til et ekstra argument kalt BusId, dette argumentet må også legges inn på linje 107 i funksjonskallet slik som dette driverClass(BusId).
- /.local/lib/python3.8/sites-packages/qwiic_vl53l1x.py :
  - I konstruktøren til klassen QwiicVL53L1X må BusId legges inn som argument, det bør legges inn på denne måten: BusId = 1, slik at standardverdien vil være det samme som før. Dersom dette ikke gjøres kan dette skape problemer for andre klasser i qwiic pakken. På linje 519 må BusId legges inn i metodekallet slik som dette: qwiic_i2c.getI2CDriver(BusId).

Når disse endringene er gjort er det klart for å velge hvilken bus man vil bruke. Når man skal deklarere et VL53L1X sensor objekt på bus 0 vil man gjøre det på denne måten: ToF = qwiic.QwiicVL53L1X(41, BusId=0).

3.3.5 Programmering - ControlLibrary

Dette biblioteket ble også utviklet av oss og består av følgende headerfiler og klasser:

- **ControllerConverter**, Denne klassen abbonerer på input dataen som kommer fra xbox-kontrolleren. Videre blir dataen konvertert til format som brukes av DroneController klassen. I tillegg er noen ekstra funksjoner lagt på slik at brukeren må trykke på forskjellige knappe kombinasjoner for å armere dronen, ta av og lande dronen. I klassen kjøres det også en egen tråd som har ansvar for å oppdatere input dataen uten å være avhengig av callbacks.

- **DroneController**, Denne klassen beskriver et objekt har den overordnede kontrollen over dronen og fungerer som en tilstandsmaskin. Objektet sender hastighet og posisjons settpunkt til Flight Controlleren med en frekvens på 20Hz. Tilstandene som maskinen kan ta er som følger:

  - **Startup**, her blir dronen initialisert, først venter programmet på at en tilkobling til flight controlleren skal bli opprettet. Deretter venter programmet på input fra brukeren om å skifte modus til OFFBOARD og armere dronen. Til slutt kan brukeren trykke på "take off" knappen og programmet vil da bytte tilstand til Takeoff.
  - **Takeoff**, i Takeoff tilstanden kjøres en funksjon som får dronen til å ta av og stabilisere seg på en predefinert høyde. Etter høyden er nådd skifter programmet tilstand til Flying.
  - **Flying**, i denne tilstanden er det brukeren som flyr dronen rundt i rommet ved hjelp av kontrolleren tilkoblet pcen. Dersom det blir oppdaget en hindring over eller under dronen skifter programmet tilstand til AvoidObstacle. Samtidig som brukeren flyr dronen vil dronen også bli påvirket av pådrag sendt fra anti-kollisjon programmet. Dette skal hjelpe brukeren å unngå å kollidere. Posisjonen til dronen blir også logget samtidig som dronen flyr. Hver gang dronen flyr lengre en 0.1m fra den forrige logga posisjonen vil den nye posisjonen legges til i loggen. Denne posisjonsloggen brukes i ReturnHome tilstanden. Dersom brukeren trykker på lande-knappen vil programmet skifte tilstand til Landing. Dersom brukeren trykker på return-home knappen vil programmet skifte tilstand til ReturnHome.
  - **ReturnHome**, I denne tilstanden vil dronen følge posisjonsloggen tilbake til posisjonen der dronen startet, når dronen kommer til den første logga posisjonen vil den lande der. Denne prosedyren kan avbrytes når som helst på ved å trykke på take-off knappen på kontrolleren, da vil programmet skifte tilstand til Flying. Da vil dronen igjen starte å logge nye posisjoner fra punktet der return home ble avbrutt. Når dronen returnerer er det også lagt inn funksjonalitet slik at kameraet alltid peker mot neste posisjon. Dette gjør at anti-kollisjons programmet fortsatt brukes når dronen er på vei "hjem". Dersom anti-kollisjons programmet gir beskjed om at dronen holder på å kollidere skifter maskinen tilstand til ReturnHomeAvoidance. I tillegg er det lagt inn dynamisk justering av hastigheten basert på om punkta dronen kjører mot er på linje eller om den er på vei mot en sving. Hvordan dronen holder riktig retning og dynamisk justerer hastigheten er beskrevet i kapitell 3.3.6

- **ReturnHomeAvoidance**, når dronen går i denne tilstanden vil den begynne å rygge bakover helt til anti-kollisjons programmet gir beskjed at det er trygt. Når det er trygt vil dronen fly oppover en halv meter og deretter prøve å kjøre forover igjen. Denne prosessen gjentas helt til dronen kommer over hindringen. Etter dronen har flydd over hindringen går den tilbake til ReturnHome tilstanden og flyr til de logga posisjonene igjen.
- **Landing**, i Landing tilstanden kjøres en funksjon som får dronen til å lande der den er.
- **Landed**, i Landed tilstanden venter programmet på input fra brukeren om å ta av igjen, dersom ”take off” knappen blir trykt skifter programmet tilstand til Takeoff.
- **AvoidObstacle**, i denne tilstanden vil hastigheten til dronen bli begrenset slik at den ikke skal kollidere med objekter over eller under dronen.

3.3.6 Programmering - RTH Funksjoner

For å peke dronen mot punktet den skal kjøre til blir først vektoren fra dronens posisjon til det neste punktet regnet ut, dette gjøres enkelt på følgende måte:

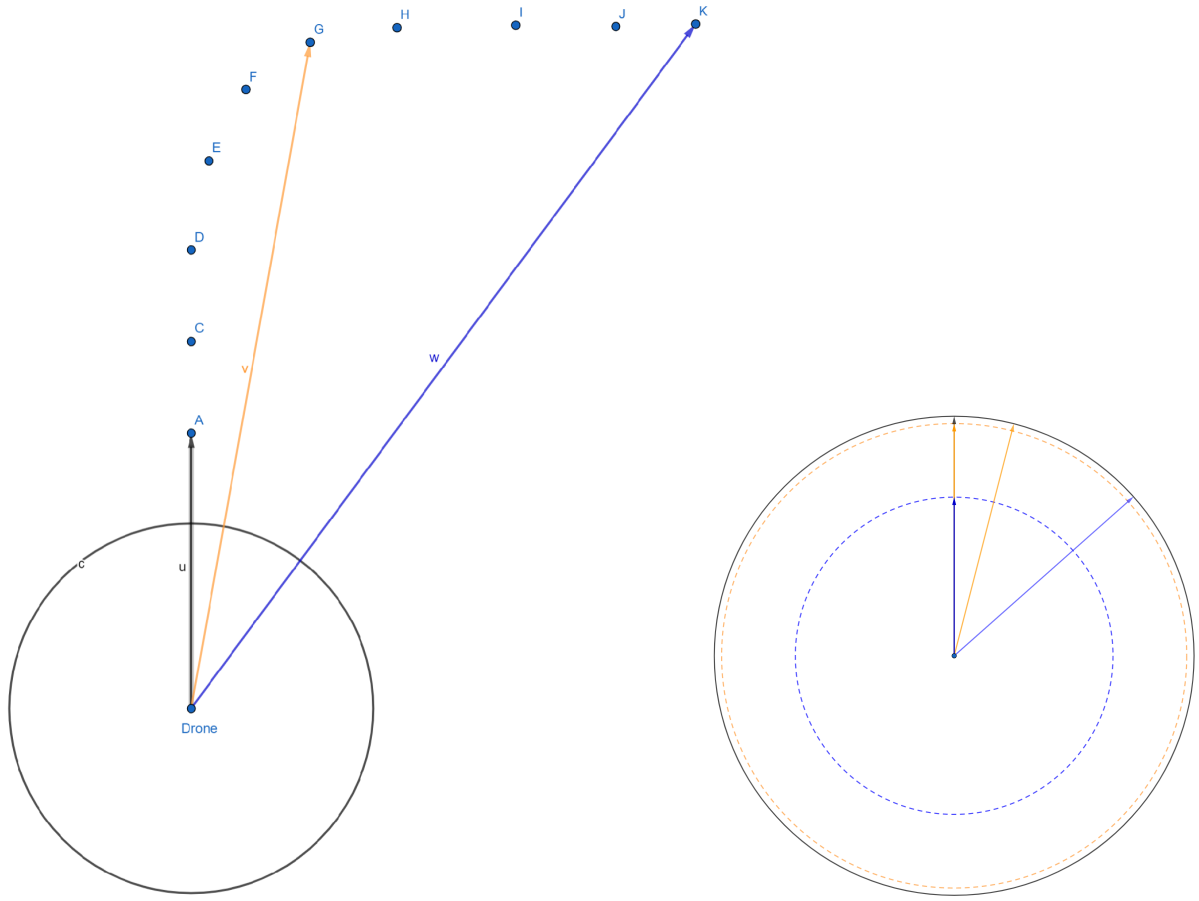
$$v = \begin{bmatrix} P_x - D_x \\ P_y - D_y \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{3}$$

Der P representerer det neste punktet dronen skal til og D representerer posisjonen til dronen. Videre blir denne vektoren sammenlignet med en vektor langs positiv x-akse som er lik $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Vinkelen mellom disse to vektorene representerer samme vinkel som dronen må ta for å peke mot punktet. Vinkelen θ regnes ut på følgende måte:

$$dot = 1 \cdot v_x + 0 \cdot v_y \tag{4}$$
$$det = 1 \cdot v_y - 0 \cdot v_x \tag{5}$$
$$\theta = atan2(det, dot) = atan2(v_y, v_x) \tag{6}$$

Hastigheten dronen har når den flyr fra punkt til punkt er avhengig av avstanden mellom punktene. I RTH funksjonen vår kan man se for seg en sfære rundt dronen når den returnerer. Alle logga punkt som er innenfor denne sfæren blir slettet slik at dronen ikke bruker tid på å kjøre mellom punkter som ble logget da dronen stod stille og ligger veldig nært hverandre. Problemet med denne tilnærmingen er at det kan føre til at dronen kutter svinger når den returnerer. For å motvirke dette ble det utviklet en funksjon som justerer størrelsen på sfæren, der størrelsen skal reduseres dersom dronen må endre retning og økes dersom dronen skal holde samme retning. Dette kan implementeres ved hjelp av vektorregning som illustrert i figur 13.

I figur 13a kan man observere 3 forskjellige vektorer der den gule(v) er for illustrasjon og den blå(w) og den sorte(u) er lik som implementasjonen vår. Den blå(w) vektoren baseres på et punkt som ligger 10 indekser lenger frem i listen i forhold til det nåværende punktet dronen er på vei mot. Den sorte vektoren(u) peker mot det nåværende punktet som dronen er på vei mot. Størrelsen til sfæren blir da proporsjonal med dot-produktet mellom w og u vektorene. På denne måten vil sfærestørrelsen være størst når w og u ligger på linje og reduseres etterhvert som vinkelen mellom w og u øker. For at sfærestørrelsen kun skal være avhengig av denne vinkelen normaliseres vektorene før utregning av dotproduktet. Sfærestørrelsen som resultat av vinkelen mellom de to vektorene er illustrert i figur 13b. For å unngå at størrelsen blir 0 eller negativ blir funksjonen forskøvet slik at den alltid er større enn 0.



(a) Illustrasjon av sfære rundt drone

(b) Etter normalisering

Figur 13: Illustrasjon av sfærestørrelse

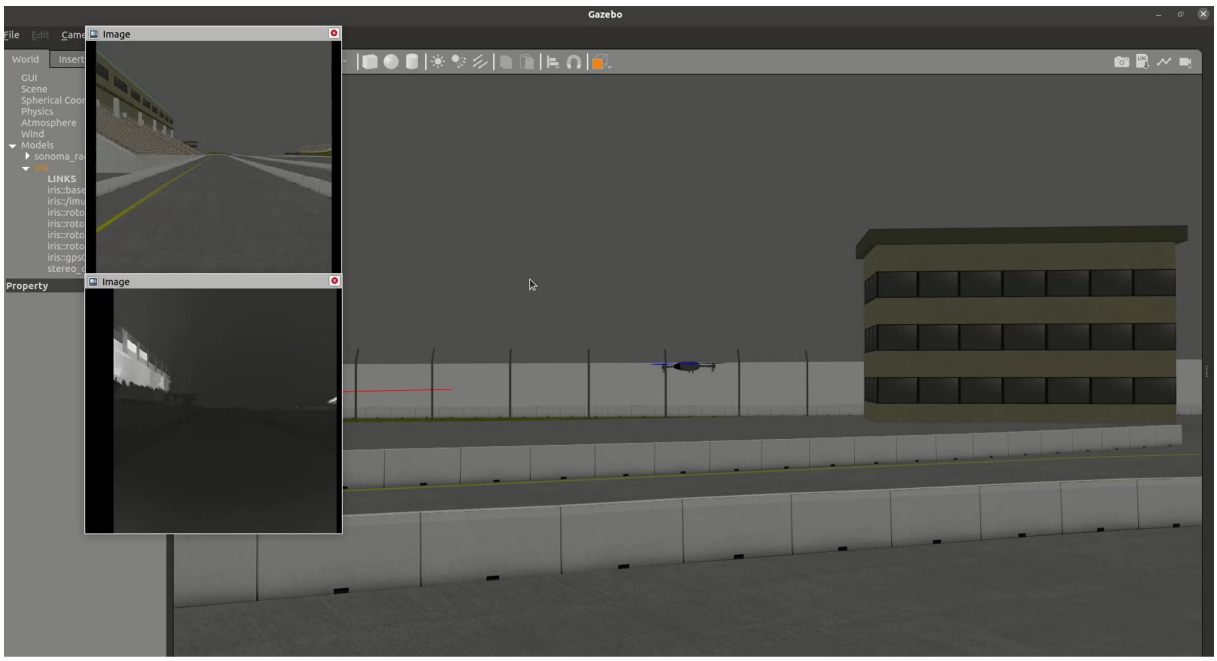
3.3.7 Simulator

Under utviklingsperioden ble en simulator satt opp slik at det ble mulig å teste nye funksjoner der før man testet på den faktiske dronen. På PX4 sine sider ligger koden

som trengst for å bygge en simulator med gazebo og dette er tilgjengelig gratis. På siden ligger det også instrukser til hvordan man skal starte byggingen og lage til et skript slik at simulatoren kommuniserer med ROS miljøet. Vi gjorde tre endringer for at simulatoren skulle fungere slik som vi ville.

1. La til *#!/bin/bash* øverst i skript filen slik at skriptet blir kjørt som bash og finner ”source” kommandoen.
2. Endret linje 16 i filen *PX4-Autopilot/launch/mavros_posix_sitl.launch* til <arg name=”sdf” default=”$(find mavlink_sitl_gazebo)/models/iris_stereo_camera /iris_stereo_camera.sdf”/>
3. Endret linje 43 og 61 i filen *PX4-Autopilot/Tools/sitl_gazebo/models/stereo_camera /stereo_camera.sdf* til <stddev>0.0</stddev>

Disse endringene gjorde at modellen i simuleringen fikk et stereo-kamera som vi kunne benytte for å teste bilde-behandlingskode og anti-kollisjonsystemet. Med dette fikk gruppen en simulator som oppførte seg likt som den virkelige dronen. Den største forskjellen er at disparity verdiene fra kameraet i simulatoren og virkeligheten ikke samsvarer i forhold til avstanden til et objekt. Nedenfor i figur 14 kan man observere et skjermbilde fra simulatoren som også viser bilde fra dronen i simuleringen og tilhørende disparity map.

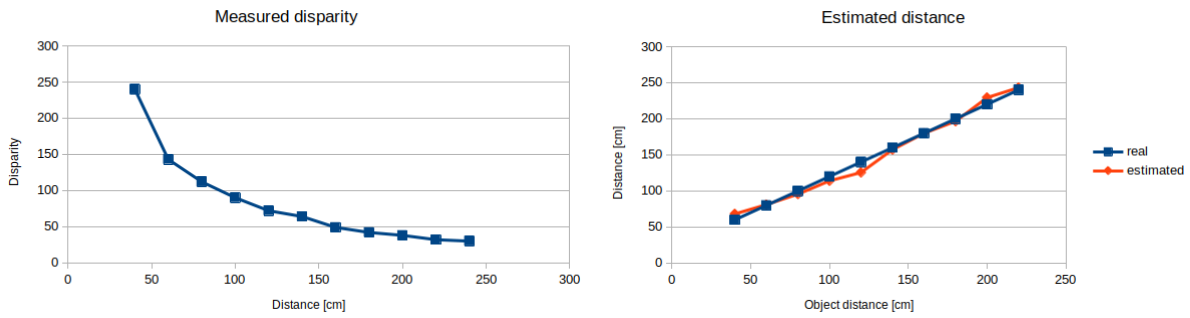


Figur 14: Skjermbilde av simulator

3.3.8 Anti-kollisjonssystem

For at dronen skulle bli enklere og tryggere å fly ble det implementert et anti-kollisjonssystem. Systemets formål var å vurdere behovet for kursendringer. Systemet benytter video fra t265 kameraet, samt målinger fra avstandssensorene som peker rett opp og ned fra dronen. Ettersom T265 er et stereo-kamera er det mulig å utnytte disparity for estimering av avstander. Til beregning av disparity ble det benyttet en ROS node ”t265_depth” som genererer og publiserer bilder hvor pixelverdiene representerer grad av disparity.

For å konvertere disparity om til avstand måtte vi estimere en funksjon for dette. Verdier av disparity ved forskjellige avstander ble målt, og ved hjelp av minste kvadraters metode ble parameterene til funksjonen funnet. Målingene er vist i figur 15a. En egenskap til disparity er at det inverse er tilnærmet lineær med avstanden. Dette gjør at en lineær funksjon gir et godt estimat av avstanden. Figur 15 sammenligner faktisk avstand med estimert avstand.

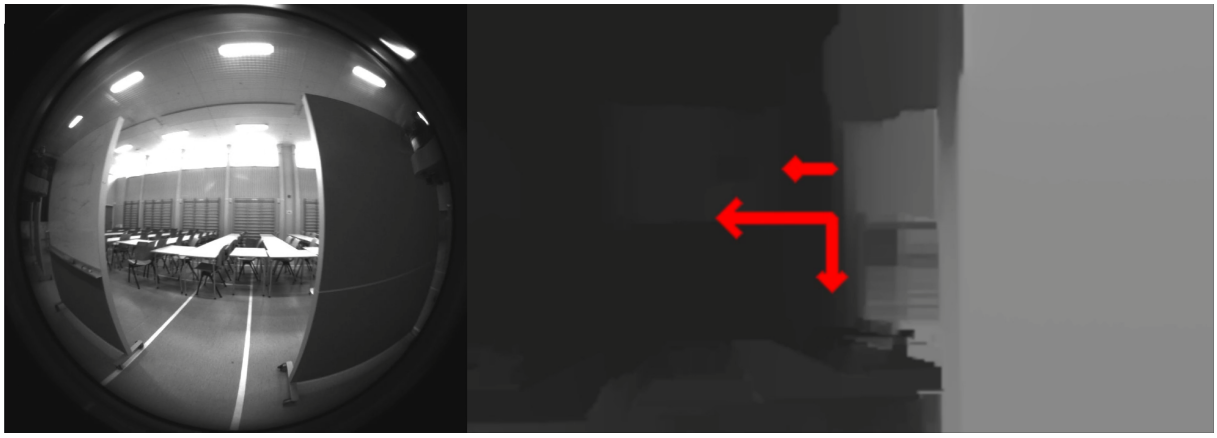


(a) Grad av disparity ved ulike avstander

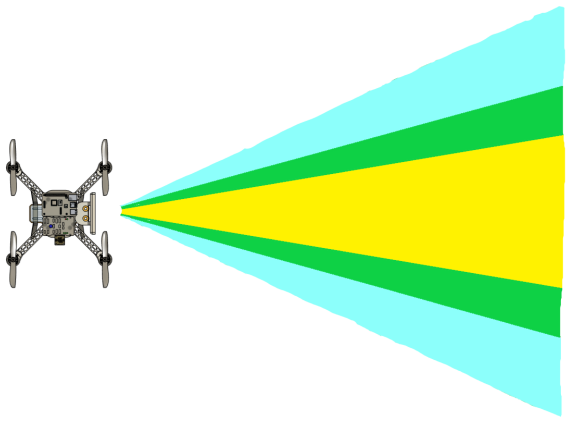
(b) Estimert avstand etter kalibrering

Figur 15: Kalibrering av avstand som funksjon av disparity

I første utgave av dette systemet var det satt en grense for avstand hvor dronen umiddelbart stoppet og forhindret dronen i å kjøre fremover. Dette demonstrerte at det var mulig å benytte et disparity map for å detektere objekt forran dronen. Systemet ble derfor videre utviklet og det ble utviklet en fuzzy logic kontroller for å ta beslutninger basert på objektets størrelse og plassering. Systemet tar også hensyn til målt avstand over og under dronen som også påvirker beslutningene til systemet. Disse blir videre publisert som et topic med navn ”course_correction”, og inneholder verdiene for roll og pitch verdi, samt binære verdier som varsler om dronen flyr for lavt eller høyt. I figur 16 kan man se skjermbilde fra videofeeden samt tilhørende dybdekart. De røde pilene i figuren illustrerer beslutningene til fuzzy kontrolleren hvor den øverste er yaw, og under er roll/pitch. For å spare prossessorkraft ble dybdekartet kun generert for et mindre utsnitt av det som var tilgjengelig fra stereokameraet.



Figur 16: Direkte video fra dronen og dybdekart med illustrering av respons fra fuzzy kontroller.



Figur 17: Indeling av soner for dybdekart. Grønt er overlappende soner.

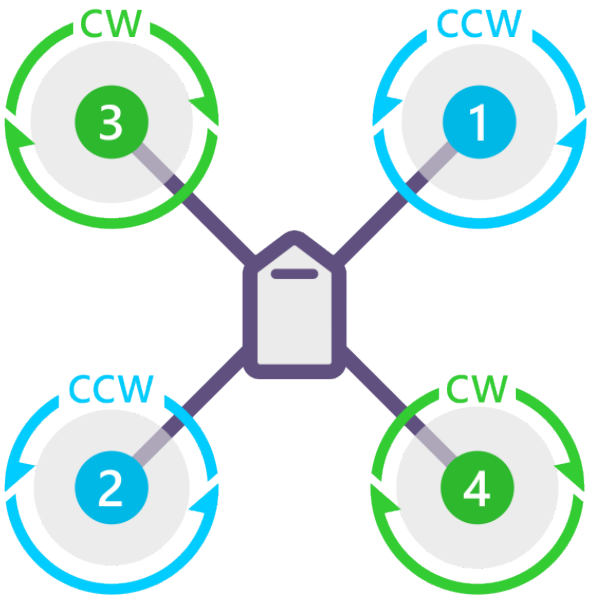
I figur 17 illustreres hvordan dybdekartet er delt opp i tre overlappende soner. Objekter som er omtrent rett forran dronen vil kun registreres i den gule sonen, og objekter som er helt i ytterkant av dybdekartet registreres i venstre eller høyre blå sone. Dersom objekter befinner seg i overgangen mellom disse sonene vil de bli registrert begge. Fuzzy kontrolleren benytter disse sonene sammen med størrelse og avstand til å avgjøre om det er nødvendig å korrigere kursen til dronen.

3.4 Testing

På grunn av de potensielle konsekvensene av en ulykke under flyvning, ble det gjort grundige tester i forkant av testflyvningene for å redusere sannsynligheten for en ulykke. Fra testene ble det avdekket flere potensielle kilder for ulykker. All testing som involverte fysisk kontakt med dronen ble gjort med propeller avskrudd.

3.4.1 ESC og motorer

Første test av motorene ble utført ved å kjøre én motor om gangen for å sikre riktig dreieretning og funksjonalitet. Her ble det observert at en av motorene reagerte forskjellig fra de andre gitt samme inputsignal. En inspeksjon av koblingspunktene ble gjort og det ble konkludert at problemet måtte være internt i motoren. Et nytt forøk ble gjort hvor det utviklet seg røyk fra motoren. Motoren ble byttet ut med en ny som fungerte. Dreieretningen til motorene ble satt i henhold til figur 18.



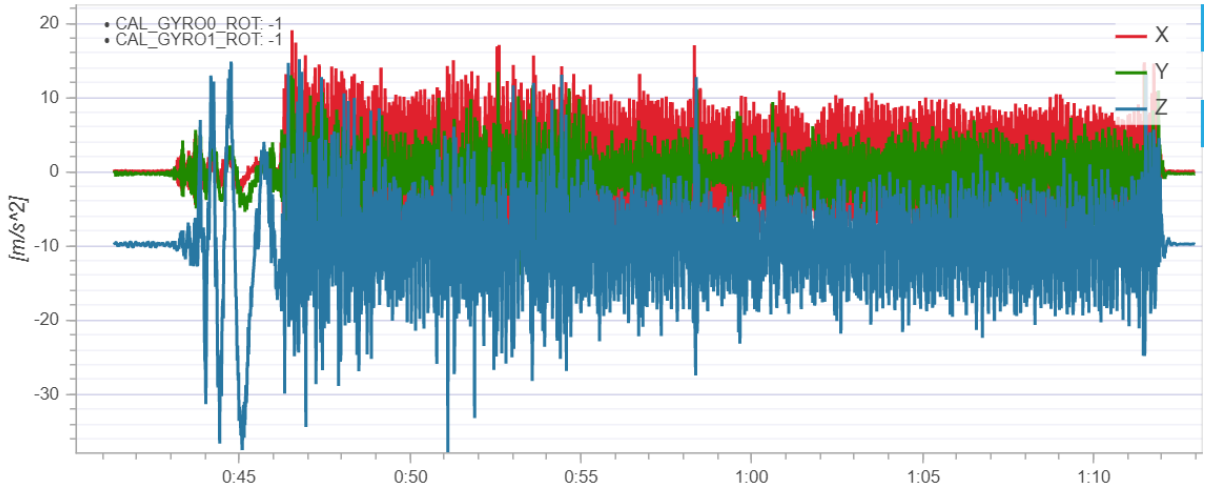
Figur 18: Dreieretning og nummer for motorer i en X-ramme, hentet fra [13]

Neste test gikk ut på å bekrefte at motorene var riktig orientert. Selv om dette ble gjort i første test, var det ønskelig å sikre dette ytterligere. Et misforhold i orienteringen til motorene ble ansett som en feil med større konsekvenser. For å bekrefte riktig orientering ble dronen aktivert i en modus med aktiv stabilisering og deretter fysisk tiltet i roll- og pitch-retning. Forventet respons var at hastigheten på motorene skulle øke og minke i forsøk om å kompensere for den tilførte ubalansen. Denne testen bekreftet riktig respons fra motorene.

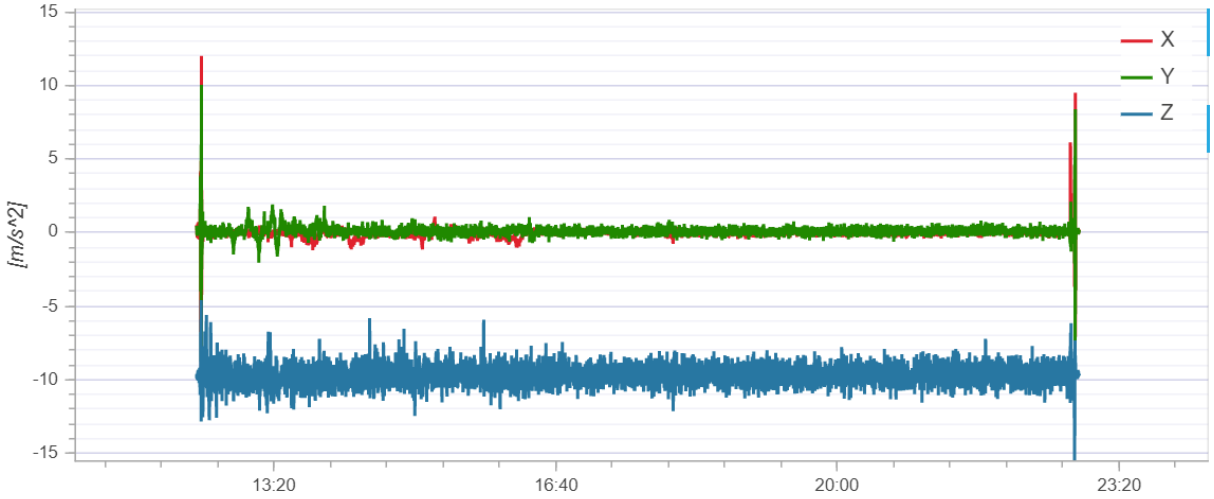
3.4.2 Stabilitet under testflyving

Ved første testflyvning hadde vi problemer med at dronen startet å vibrere med en konstant frekvens så fort den forlot bakken. Vibrasjonene var så store at det var vanskelig å holde kontrollen på den. Vibrasjonene førte også til at realsense-kameraet ikke klarte å finne posisjonen sin da dette kameraet ikke tåler noe særlige vibrasjoner. Det ble utforsket flere hypoteser på hvorfor disse vibrasjonene oppstod. Den første hypotesen var at rammen vi 3d-printet var for fleksibel og dette førte til at motorene vridde seg og skapte ustabilitet på denne måten. Dette ble avkreftet ved at vi tok en video av dronen i sakte film(240fps) og observerte i denne filmen at rammen holdt seg stiv under oscillasjonene.

Etter å ha avkreftet at det var rammens fleksibiletet som foråsaket vibrasjonene ledet dette gruppen til å tru at det kunne være feil parametre for PID-kontrolleren som skapte vibrasjoner i stedet. Denne hypotesen viste seg å være rett. Ved å først endre airframe parameteret på flight-controlleren til en rammetype som hadde samme størrelse som vår ramme fikk vi et bedre utgangspunkt for PID-justering. Når vi startet justering var tanken at P-leddet var for høyt og dermed gjorde reguleringen ustabil og skapte de stående oscillasjonene. Ved å halvere P-leddet på både roll og pitch reguleringen fikk vi en mer tilfredstillende oppførsel som var stabil uten oscilleringer. Nedenfor i figur 19, 20 og 21 viser forskjellige avlesninger av sensordata fra testflyvinger, både før justering ble utført og etter.



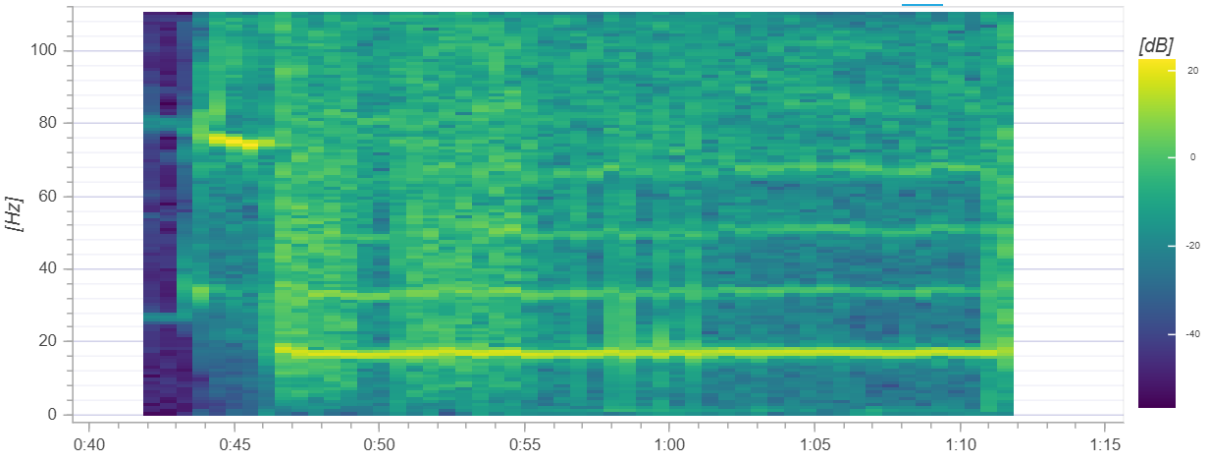
(a) Rå akselerasjonsdata før justering



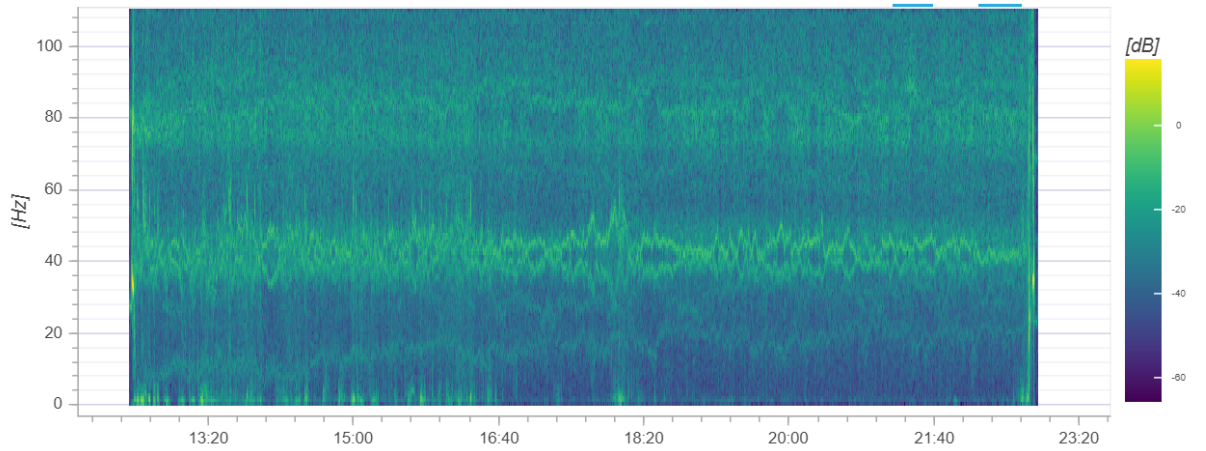
(b) Rå akselerasjonsdata etter justering

Figur 19: Rå akselerasjonsdata fra testflyvning

I figur 19a kan det se ut som vi har en veldig støyete sensor men i realiteten er det ustabiliteten til reguleringen som skaper oscillasjoner og dette blir plukket opp av sensoren. I figur 19b kan man se en stor forbedring etter tuning av PID-kontrolleren.



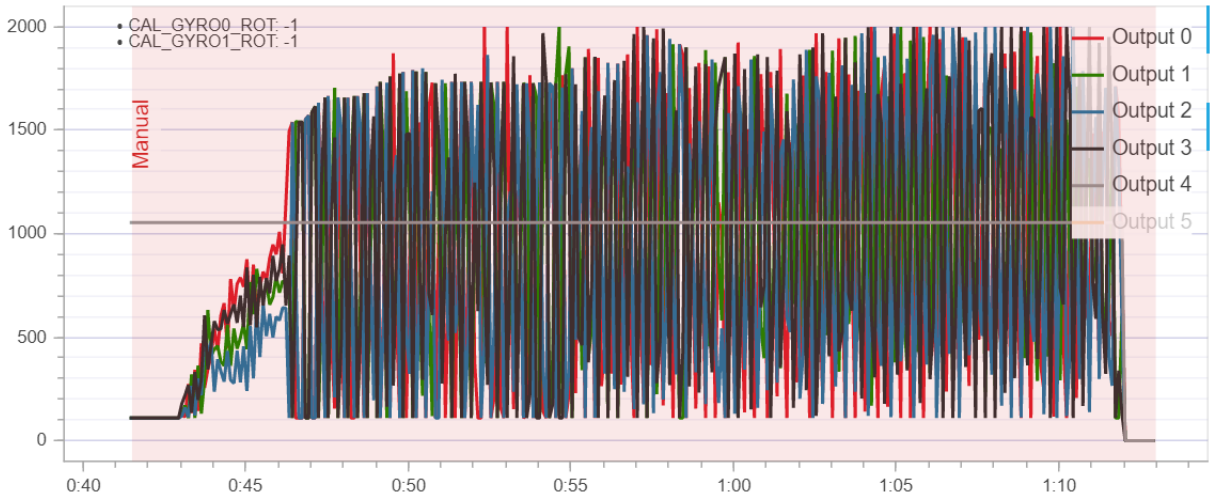
(a) Spektrogram akselleresjon før justering



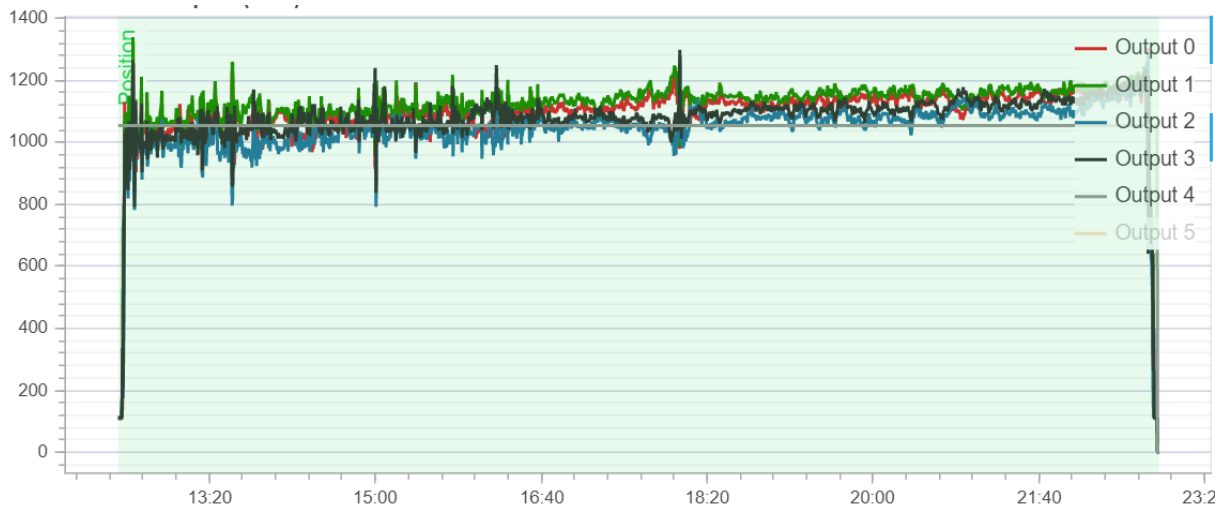
(b) Spektrogram akselleresjon etter justering

Figur 20: Spektrogram akselleresjon test flyvning

I 20a kan man i spektrogrammet se at dronen hadde en sterk frekvenskomponent ved ca 20Hz og harmoniske komponenter lengre oppe i frekvensspekteret. I figur 20b kan man se at de sterke frekvenskomponentene er borte etter at PID-kontrolleren ble justert.



(a) Pådrag før justering



(b) Pådrag etter justering

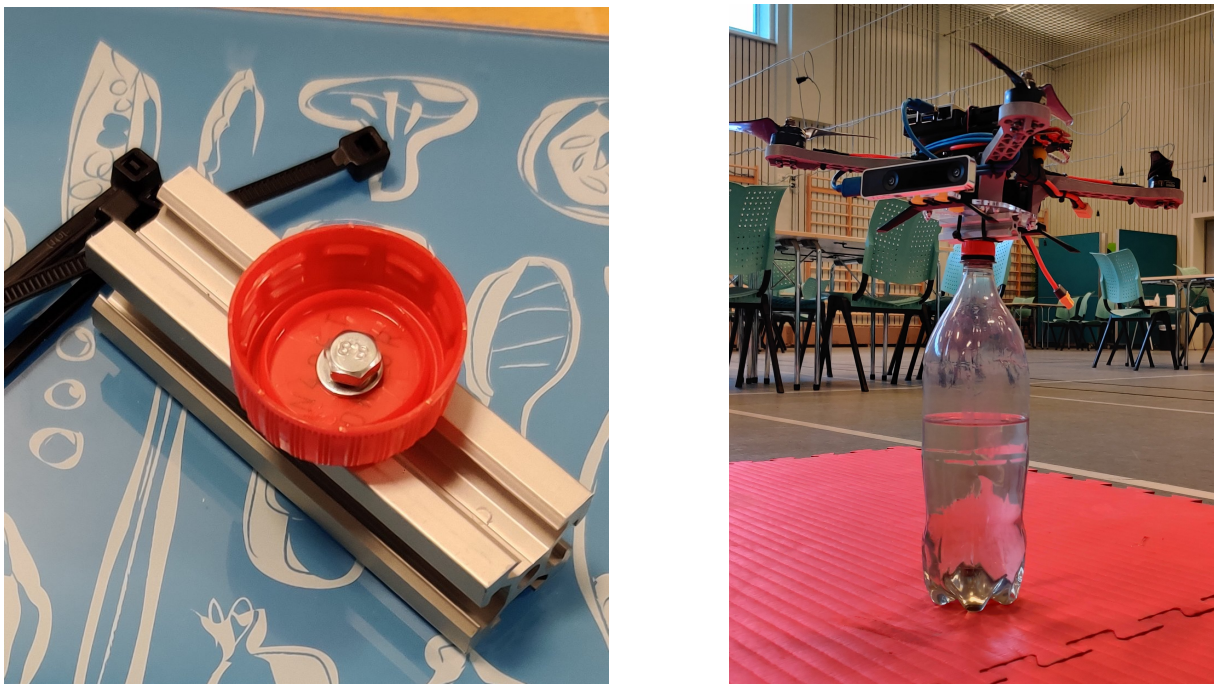
Figur 21: Pådrag fra testflyvning

I figur 21a og 21b illustreres pådraget som blir sendt til hver motor (Output 0-3). Før justering ble gjennomført kan man se at pådraget oscillerer mellom minimal og maksimal verdi. Dette stammer fra dårlige parameter i PID-kontrolleren. Etter testflyvingen i figur 21a ble motorene svært varme på grunn av den store variasjonen i pådraget. Etter justering ser pådraget mer stabilt ut og motorene holdt seg også kjølige gjennom flyturen.

3.4.3 Flytid

For å gi et bilde av hvor lenge dronen kunne fly ved forskjellige laster ble dette først estimert i simuleringsverktøyet Ecalc, og deretter testet i virkeligheten. Testen gikk ut på å

løfte en flaske vann og holde denne i luften til batteriet ble 15%. Denne testen ble gjentatt flere ganger med ulik mengde vann i flasken. Batteriet ble ladet helt opp i forkant av alle testene, og batteriets gjenværende prosentverdi ble lest fra Qgroundcontroll.



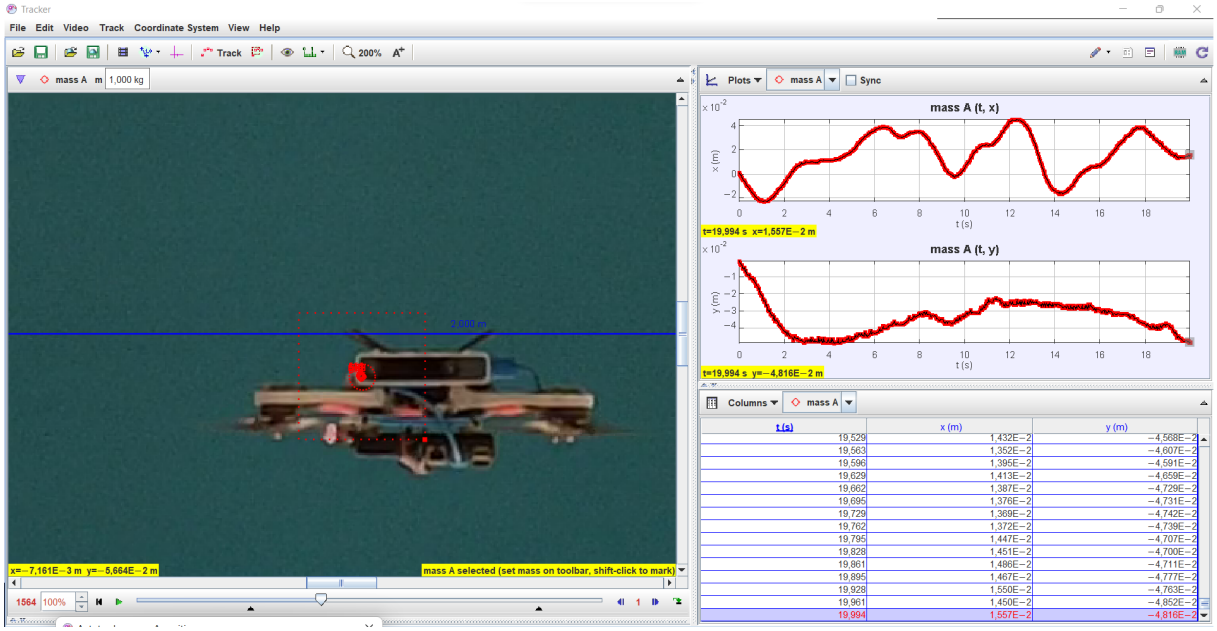
(a) Feste (b) Drone med last

Figur 22: Testoppsett for variabel last

3.4.4 Posisjonpresisjon

For å få finne presisjonen til posisjoneringsystemet på dronen ble det laget et enkelt men effektivt testoppsett som kan observeres i figur 23a. Først ble det satt opp to vegger med kjent avstand mellom de (2 meter), samt en vegg mellom disse lenger bak for å minske forstyrrelser i bakgrunnen. Et kamera ble plassert normalt og sentrert på veggene med samme høyde som dronen skulle holde. Dronen ble flydd manuelt til den var midt mellom de to veggene og på samme høyde som kameraet og deretter ble den satt i posisjonsmodus slik at den ville prøve å holde seg på dennne posisjonen i rommet. På dette tidspunktet ble de gitt et signal til kameraet slik at gruppen kunne vite når de skulle starte sporingen av dronen. For å gjøre selve sporingen av posisjonen til dronen ble programmet Tracker benyttet. Dette kan observeres i figur 23b.

(a) Oppsett testing av posisjonspresisjon



(b) Oppsett test

Figur 23: Skjermbilde fra Tracker programmet

3.4.5 Landingspresisjon

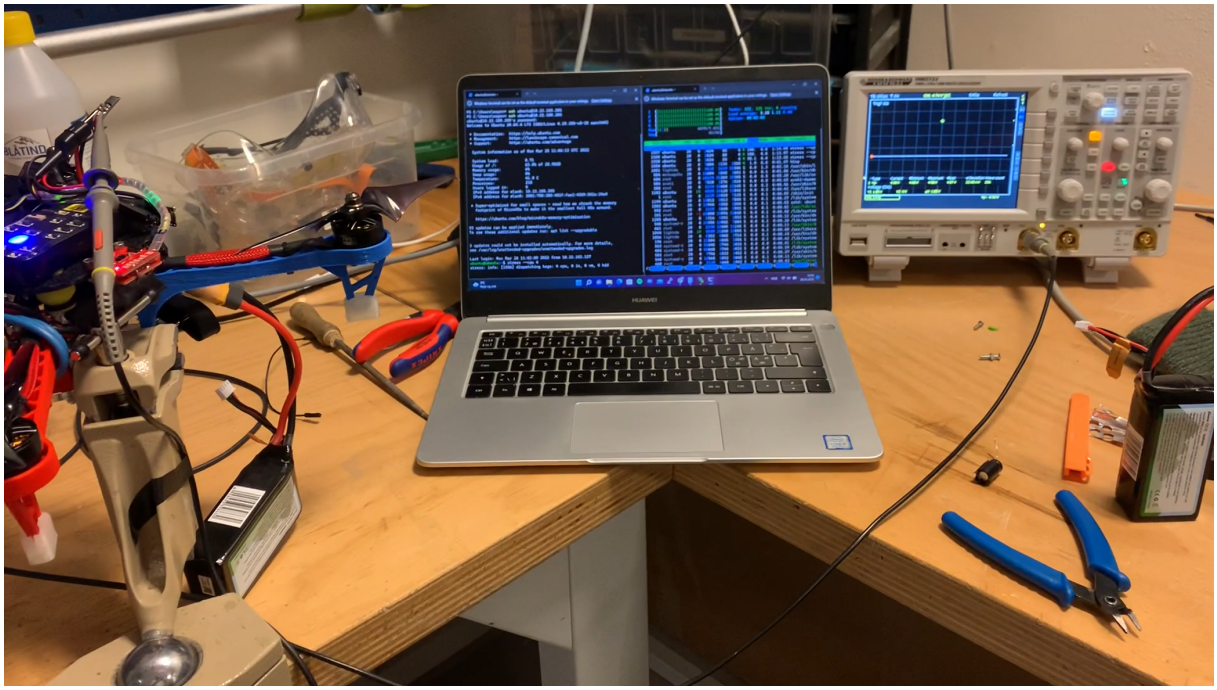
For å finne presisjonen til landingsfunksjonen i return to home systemet ble det gjort 15 uavhengige tester. Posisjonen som dronen startet fra ble markert på gulvet. Dronen ble flydd i varierende lengder fra startposisjonen, deretter ble return to home aktivert. Etter at dronen hadde landet ble avstanden fra landingspunktet til startpunktet målt og notert.

3.4.6 Strømforsyning Raspberry PI

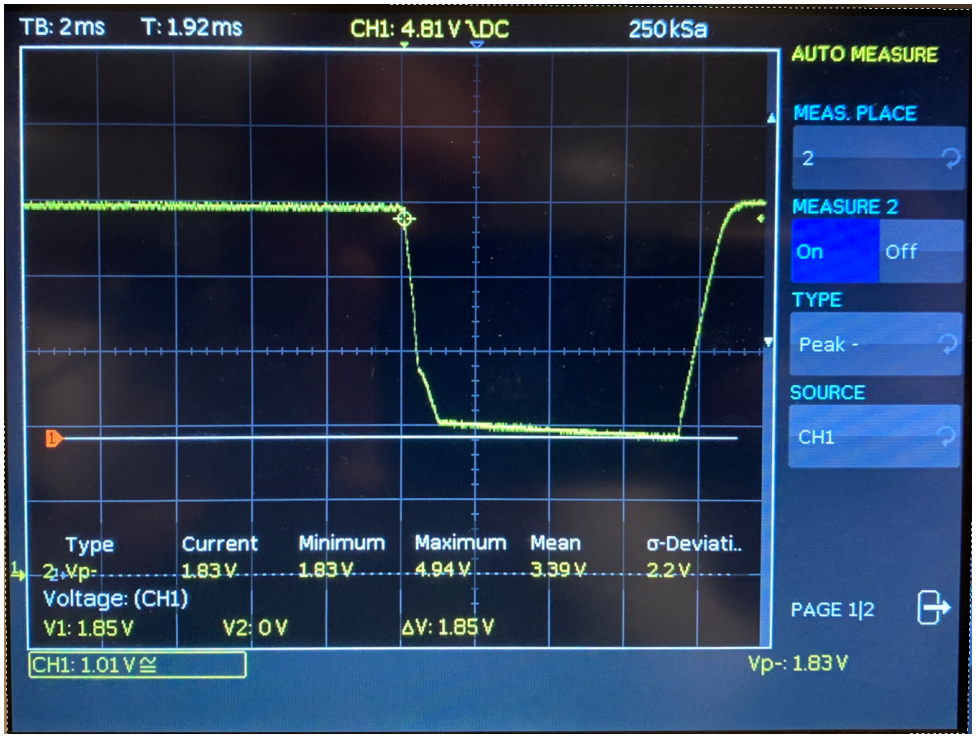
Under testflyvinger hadde vi problemer med at Raspberry Pien skrudde seg av og på og det var vanskelig å se noen klar sammenheng mellom tilfellene hvor dette skjedde. Den eneste sammenhengene vi kunne se var da batterispenningen begynte å komme ned på rundt nominell spenning (11,3V) så skjedde det oftere. Dette ledet oss til å tro at problemet muligens skyldes dårlig strømforsyning til Pien.

For å undersøke dette nærmere satt vi opp et eksperiment der dronen var fast montert i en skrustikke på labben, dette gjorde det mulig kjøre motorene på fullt og belaste systemet i kontrollerte former. Videre installerte vi et "stress test" program på Pien som belastet alle prosessorkjernene 100%, med motorene på fullt og dette programmet kjørende på Pien ble hele systemet fullt belastet. Neste steg i eksperimentet var å koble til et oscilloskop på lederen som forsynte Pien med strøm og observere spenningsnivået under full belastning.

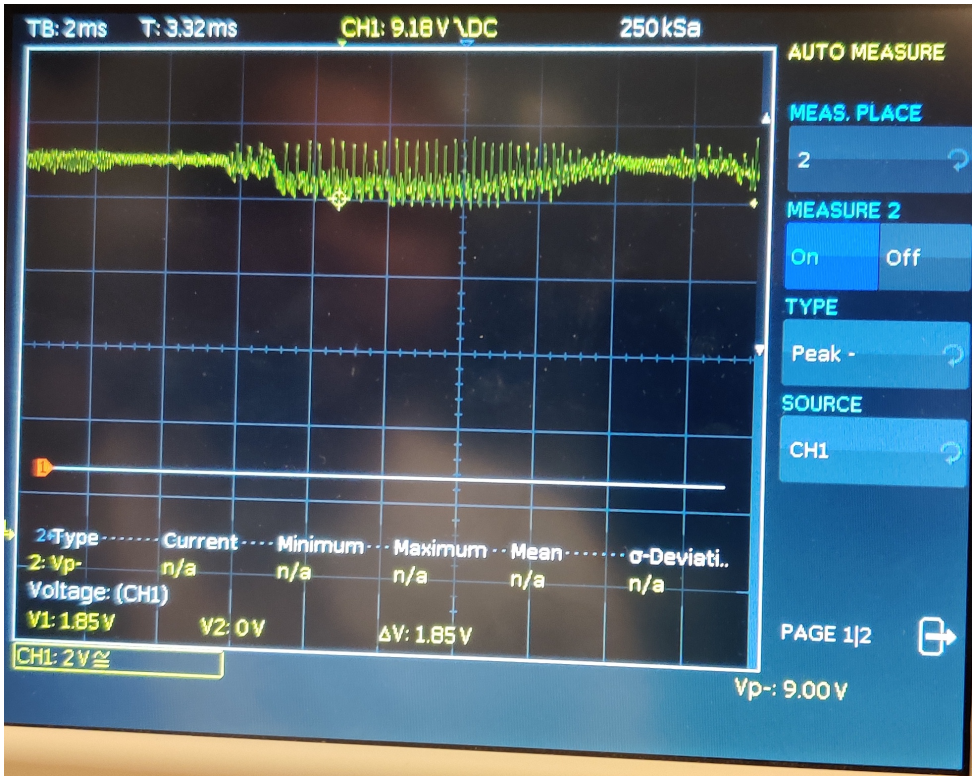
Resultatet fra eksperimentet kan sees i figur 25, der ser man tydelig at spenningen til Pien har et fall ned til 1.83V før spenningen går tilbake til 5V. Man ser også at spenningen ligger på det lave nivået i ca 8ms. Dette var helt klart problemet som gjorde at Pien gikk av og på. Under eksperimentet ble batteri spenningen først regulert til 10V av ESCen og deretter regulert til 5V av en BEC-konverter, denne 5V spenningen forsynte Pien. Spenningen som ESCen leverte kan observeres i figur 26, gruppen har konkludert med at det var det ustabile spenningsnivået ut fra ESCen som foråsaket problemene. BEC-konverteren ble derfor forsynt direkte fra batteriet i stedet og dette fikset problemet, vi hadde ingen flere spenningsfall.



Figur 24: Eksperiment oppsett



Figur 25: Måling av spenningsforsyning til Pi fra BEC



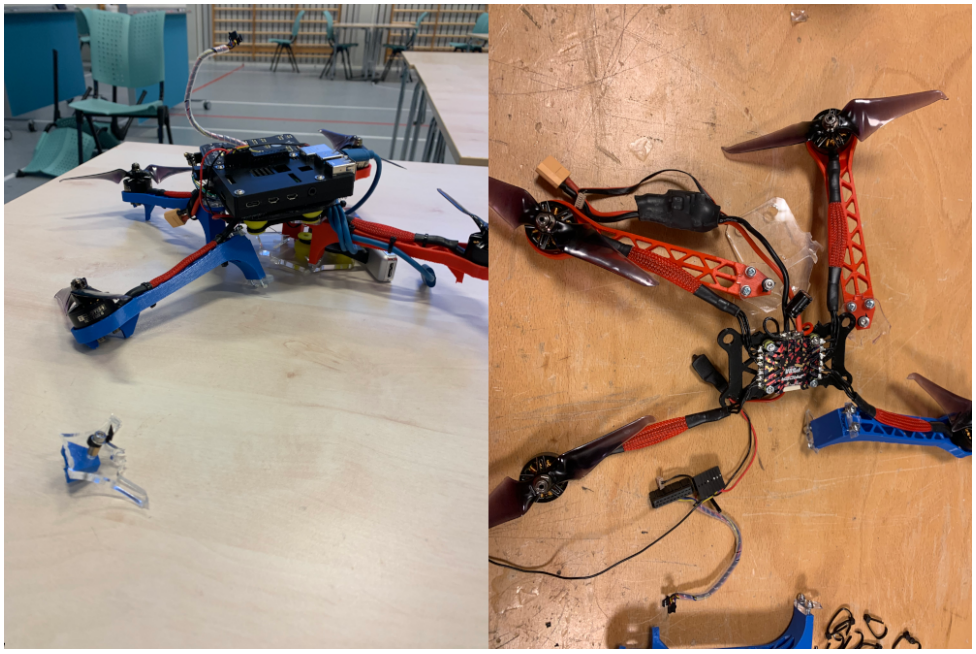
Figur 26: Måling av spenningsforsyning til BEC fra ESC-BEC

4 RESULTATER

4.1 Konstruksjon

Dronen gruppen har bygd flyr svært stabilt. Dronen har kollidert og krasjlandet noen ganger gjennom prosjektperioden og i de fleste tilfellene er det kun propeller som måtte byttes. Designet som gruppen hentet inspirasjon fra ble forsterket og tilpasset slik at det lot seg 3d-printe på en god måte og samtidig takle vibrasjoner og krasjlandinger. Det ble også designet vibrasjonsdempende braketter for kamera og flightcontroller. Man kan observere fra kamera livestreamen at vibrasjonsdempingen fungerer siden bilde er klart selv om dronen er i luften. Det at dronen flyr stabilt er også et bevis på at vibrasjonsdempingen til flightcontrolleren fungerer bra.

Når det kommer til å svare på oppgaven om å bruke mest mulig materialer fra campus har gruppen også gjort dette. Alle armer og braketter er 3d-printet med printere (og PLA-filament) som befinner seg på campus Ålesund. De to platene som holder dronen sammen ble skjært ut ved hjelp av laserkutter som også befinner seg på campus. Realsense-kameraet og flightcontrolleren befant seg også på campus og kom fra tidligere prosjekt utført der. Ved å ha designet dronen på en slik måte betydde det at dronen relativt raskt kunne repareres ved en større kollisjon. Akrylplatene kan skjæres ut på minutter og armene kan printes i løpet av en dag. Dette kom godt med under utviklingen av dronen, nedenfor i figur 27 er et eksempel på hvordan dronen så ut etter et større krasj.



Figur 27: Resultat av krasj

Som man kan se fra figur 27 er alle armene og akryplatene på dronen ødelagt etter krasjen. I dette tilfellet hadde dronen et fritt fall på ca.5m før den traff gulvet. På grunn av at vi kunne produsere delene selv var dronen reparert i løpet av 1.5 arbeidsdager. Dersom vi hadde reservedeler på lager ville reparasjonstiden blitt betydelig kortere.

Når det kommer til hvor bra dronen oppfylte kravspesifikasjonene som ble satt i starten av prosjektet er gruppen delvis fornøyd med resultatet. Nedenfor går vi gjennom spesifikasjonene:

- **Vekt:** Den prosjekterte totalvekten var 830g, når dronen var ferdig bygd veide den 970g.
- **Størrelse:** Den prosjekterte størrelsen på dronen var 330mm på diagonalen, størrelse på ferdig drone stemte med dette.

4.2 Flyveevne

4.2.1 Posisjonpresisjon

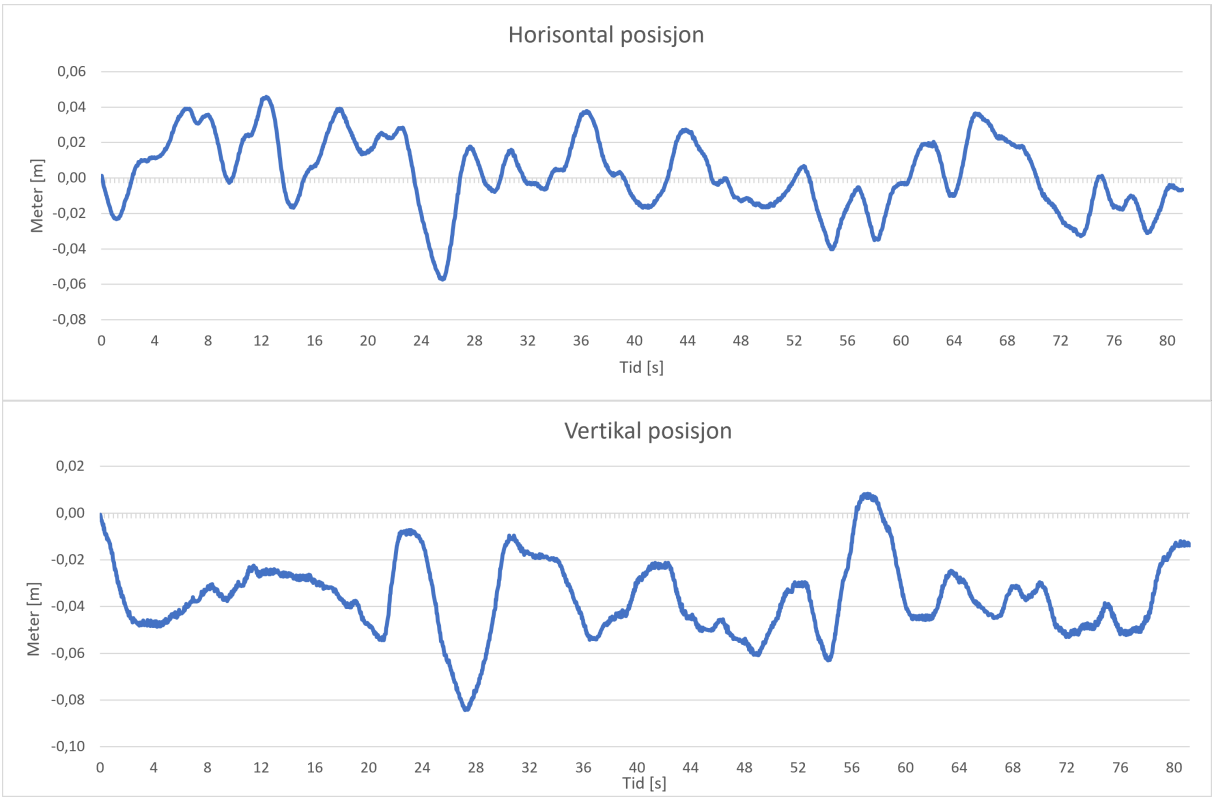
Nedenfor i figur 28 kan resultatet fra testing av posisjonspresisjon observeres.

Horisontal posisjon

- Standardavvik: $0.021m$
- Maksimal verdi: $0.046m$
- Minimal verdi: $-0.057m$

Vertikal posisjon

- Standardavvik: $0.016m$
- Maksimal verdi: $0.008m$
- Minimal verdi: $-0.084m$



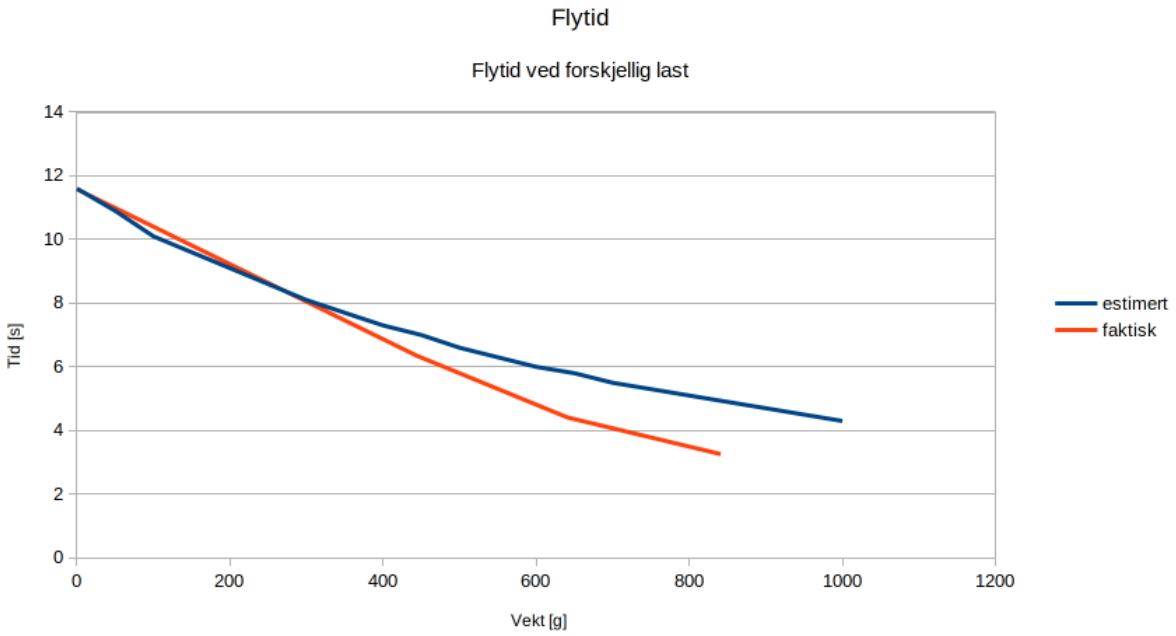
Figur 28: Måleresultat posisjonspresisjon

4.2.2 Løftekapasitet

Den teoretiske løftekapasiteten til dronen var 1kg dette ble regnet ut ved hjelp av eCalc simulatoren. I gruppens test ble det svært vanskelig å fly dronen etter å ha festet 1kg med last. Dronen klarte fortsatt å lette men den hadde ikke nok motorkapasitet til å stabilisere seg i luften. Så den teoretiske verdien stemmer i dette tilfellet bra med virkeligheten. Den maksimale løftekapasiteten til dronen er 1kg.

4.2.3 Flyvetid

Resultatet fra flyvetid testen kan observeres i figur 29. Man kan se at uten last ligger flyvetiden på 11,6 min. 11,6min er litt under kravspesifikasjonen vår og kommer av at dronen ble tyngre en det som var planlagt. I denne testen sammenligner også vi teoretisk (estimert med eCalc) flyvetid med faktisk.



Figur 29: Flyvetid med forskjellig last

4.3 Autonom modus

4.3.1 Anti-kollisjon - pålitelighet

Anti-kollisjons systemet fungerer som tenkt. Systemet gjør stort sett inngrep når det er nødvendig, men dersom operatøren har lyst å kollidere med dronen klarer han det. Målet til gruppen var ikke å lage et system som gjør det umulig å kollidere, men heller lage et system som hjelper operatøren å holde seg på rett kurs og som gjør det enklere å styre dronen. Svakheten med systemet kommer fra ustabiliteten i disparity bildet. Dersom hindringene dronen er på vei mot ikke har tilstrekkelige trekk som kan matches, kan ikke disparity regnes ut. Et eksempel på slike hindringer er helt hvite vegger. Så lenge hindringene har gode trekk gjør systemet det lettere for operatøren å unngå kollisjoner. Systemet hjelper operatøren med å holde riktig kurs (yaw vinkel), rulle bort fra hindringer dersom dronen kommer for nært et hinder langs en av sidene, og rygge dersom et hinder kommer for nært forran dronen.

4.3.2 Return to home - landingspresisjon

Resultatet fra landingspresisjons testen kan observeres under i tabell 1 og 2

Test Nr	Avstand [m]
1	0.08
2	0.30
3	0.27
4	0.26
5	0.25
6	0.30
7	0.16
8	0.26

Tabell 1: Resultat fra test nummer 1-8

Test Nr	Avstand [m]
9	0.23
10	0.15
11	0.19
12	0.20
13	0.22
14	0.10
15	0.22
-	-

Tabell 2: Resultat fra test nummer 9-15

Testene gir dette resultatet:

- **Gjennomsnitt avstand fra startposisjon: 0.21m**
- **Standardavvik: 0.06m**

4.3.3 Return to home - pålitelighet

Return to home systemet fungerer som tenkt. Systemet er robust og det er sjeldent at dronen ikke klarer å komme seg tilbake til der den startet. Spesielt etter at anti-kollisjon systemet ble knyttet sammen med return to home ble det et robust system. Dersom nye hinder oppstår når dronen returnerer hjem er systemet avhengig av at de nye hindringene har tilstrekkelige trekk slik at anti-kollisjons systemet blir aktivert. Dersom det ikke oppstår nye hinder i tiden mellom dronen flys manuelt og return to home aktiveres er systemet ikke avhengig av anti-kollisjons systemet.

5 DRØFTING

5.1 Design og valg av komponenter

Når det kommer til designet og valg av komponenter i prosjektet er gruppen for det meste fornøyd. Dronen fløy stabilt og kunne håndtere mindre kollisjoner og fall uten å bli ødelagt. Likevel er det noen deler som gruppen ville endret dersom de startet prosjektet på nytt.

Dersom gruppen skulle designet en ny drone ville de hatt større fokus på enklere utskifting av komponenter. Slik designet er nå er det tidkrevende å skifte en ødelagt arm. Dette kunne med fordel blitt redesignet slik at man kunne skifte armene uten å demontere andre deler av rammen. Gruppen ville også ha utforsket muligheten til enklere montering av batteri.

Det er også andre mindre deler av dronen som gruppen ville ha endret. Kabelen til T265 kameraet er unødvendig lang og tung. Det ble gjort forsøk på å lage en kortere kabel, men dette førte til ustabil bildeoverføring. Det burde i neste iterasjon av designet bestilles en ny og kortere kabel til kameraet. I tillegg ville også gruppen endret designet til den vibrasjondempede platen som holder Raspberry Pien og Flight Controlleren. Gummifestene kunne med fordel blitt montert med en vinkel slik at de pekte mer mot senter av platen. Med gummifestene montert slik ville de hjulpet mer på vibrasjoner i horisontal retning.

Gruppens valg om å benytte Intels Realsense T265 er de fornøyd med. Kameraet fungerte utmerket så lenge det var montert med god vibrasjonsdemping. Til tross for dette bør neste iterasjon av dronen bruke et annet kamera, ettersom Intel har besluttet å ikke produsere T265 kameraet lengre. Noen alternativer til T265 kameraet kan være:

- ZED 2 - Dette stereokameraet har mange av de samme funksjonene som T265 kameraet, men mangler intels VPU chip og er derfor avhengig av mer datakraft hos verts-maskinen.
- OAK-D - Dette er et rimligere alternativ men er på samme måte som ZED 2 avhengig av mer datakraft hos verts-maskinen.
- Realsense D435i - Dette punktskykameraet fra Intel skal fortsatt produseres og kan være et godt alternativ. Med dette kameraet blir man nødt til å ta en annen tilnærming en disparity map og heller bruke punktskyen til anti-kollisjon.

Når det kommer til kombinasjonen av Raspberry Pi 4B og Pixhawk Flightcontroller er gruppen fornøyd med valget. Raspberry Pien ligger litt under 100% belastning når alle programmer kjører. Det finnes alternativer til Raspberry Pi med mer datakraft, men de veier mer og bruker mer strøm så da må man veie opp flyvetid mot datakraft. Det

finnes også alternativer til Pixhawken, men fra de undersøkelsene gruppen har gjort virker det som Pixhawk og PX4 firmware som et av de beste alternativene. Dersom man skal bytte til en ny Flightcontrollerer bør man vurdere Pixhawk 4 Mini siden denne er mindre og ikke har så mange overflødige funksjoner.

5.2 Bruk av ROS

Å benytte ROS er en av avgjørelsene gruppen er mest fornøyd med. Mange eksisterende ROS noder har vært nyttig i utviklingen av funksjonene til dronen. Dette har gjort veien fra idé til implementasjon kortere og enklere i motsetning til å skulle utvikle tilsvarende løsninger selv. ROS har i tillegg forenklet veksling av data mellom forskjellige prosesser gjennom publisher/subscriber-konseptet som er forklart i kapitel 2.3. Dersom gruppen skulle implementere et slikt konsept selv ville det tatt store mengder tid, og begrenset utviklingen av funksjonalitetene til dronen.

For å kunne utnytte funksjonalitetene i ROS må man først gjøre seg kjent med hvordan det anvendes. I begynnelsen var det utfordrende å sette opp ROS miljøet og mye feilsøking måtte til før det tilslutt virket. Derimot er tiden innhentet da det har spart gruppen mye tid gjennom oppgaven, spesielt på grunn av at gruppen slapp å implementere nettverk og inter-prosess kommunikasjon.

Det kan være anbefalt for de som eventuelt skal bygge videre på dronen, eller utføre en liknende oppgave å benytte ROS 2. Selv om ROS 2 var tilgjengelig under tidsperioden for denne oppgaven var støtten for relevante komponenter begrenset og det ble besluttet at ROS var det beste valget. Det vil derimot være en mulighet for at bredere støtte eksisterer på et senere tidspunkt.

5.3 Sensorer

5.3.1 Lidar

I begynnelsen hadde gruppen tenkt å benytte en lidar til å registrere omgivelsene rundt dronen. Dette ble lagt på is av ulike grunner. En av grunnene var at de tilgjengelige lidarene på campus ble vurdert til å være for store og tunge til formålet. Det ble funnet en lidar (LiDAR_LD06) som scorer bedre med tanke på vekt og størrelse som kan være mulig å utforske i en lignende oppgave. Denne lidaren var ikke tilgjengelig på campus og måtte eventuelt bestilles.

5.3.2 Qwiicc vl53l1x - Avstand

Selv om dronen kjenner høyden sin relativt til start er det ikke en triviell oppgave å ha kontroll på endring i tak- og gulvhøyde. I tillegg er det mulig at objekter kan befinne seg over og under dronen. Avstandssensorer ble benyttet for å gjøre det mulig å implementere en ekstra barriere mot kollisjon i tak og gulv. En svakhet med de valgte sensorene er at de har et begrenset synsfelt på 15-27 grader. Den lave vinkelen på synsfeltet gjør at sensorene har en blindsone hvor objekter ikke vil bli detektert. Sensorene er også plassert mot dronens venste side som gjør blindsonen større på høyresiden. Det har derimot ikke blitt utforsket muligheter til å minimere denne blindsonen da sensorenes primære oppgave er å forebygge at dronen treffer tak eller gulv.

5.4 Video

For å operere dronen utenfor siktelinjen er det nødvendig å ha en direkte videostrøm fra dronen. Denne videostrømmen burde ha så lav forsinkelse som mulig samtidig som kvaliteten er godt nok til at piloten kan orientere seg. I denne oppgaven ble det benyttet videostrøm fra realsensekameraet med lav forsinkelse. Videostrømmen fra realsensekameraet har et vidt synsfelt som for et utrent øye kan gjøre at objekter oppfattes lenger unna enn de faktisk er. Dette kan føre til at dronen blir ekstra utfordrene å fly i miljøer hvor det er mange objekter å ta hensyn til. Et alternativ som vil gjøre dronen enklere å fly utenfor siktelinjen kan være å montere et sekundært kamera med et mindre synsfelt.

5.5 Bruk av disparity map

Valget om å benytte disparity map til å unngå kollisjoner er gruppen tilfredsstilt med. Med tanke på omfanget av oppgaven var det ikke nok tid til å implementere mer avanserte kartleggings algoritmer. Bruken av disparity map produserte et akseptabelt estimat av avstander til objekter. Genereringen av disparity map krevde heller ikke mer datakraft enn det som er tilgjengelig på en Raspberry Pi, men det var heller ikke mye datakraft til overs for andre programmer. Dersom mer tid var tilgjengelig ville gruppen utforsket bruken av nevrale nettverk til å lage disparity map. Bruk av nevrale nettverk vil sannsynligvis spare datakraft og samtidig gi like gode resultater.

Gruppen har også vurdert andre alternativer til disparity map som kan være aktuelle for fremtidig arbeid med dronen. Det som ble vurdert var å implementere en SLAM algoritme og bruke dette til å navigere i rommet. Da med enten en lidar eller et punktsky kamera montert på dronen. En slik algoritme ville gitt mer nøyaktig kartlegging av rommet dronen befinner seg i. En slik algoritme ville også tilrettelagt for mer bruk av historisk data til navigasjon. Likevell ville man nok fortsatt være avhengig av å ha et

kamera på dronen slik at operatøren kan se fra dronens perspektiv.

5.6 Styringsystem

Gruppen er tilfredsstilt med utformingen av dronens styresystem. Systemet inneholder de funksjonene gruppen så for seg i begynnelsen av prosjektet. Kontrollene anses som enkle og kontrolleren som blir brukt er en type som mange er kjent med fra før. Dette gjør at det ikke kreves så mye opplæring eller tilvenning før man kan fly dronen.

5.7 Læringsutbytte

5.7.1 Utviklingsprosess og prosjektstyring

Under prosjektperioden har gruppen lært mye om prosessen fra en idé til en prototype. Gruppen startet med en ide om hvordan de så får seg dronen i slutten av prosjektet og begynte å gjøre undersøkelser. Etter å ha gjort undersøkelser og funnet ut hva som er mulig, startet gruppen med planlegging og det ble laget krav til spesifikasjoner. Det ble gjort undersøkelser om hvilke komponenter som kunne passe til kravet. Å velge riktige komponenter var en lærerik prosess som tvang gruppen til å lære seg hva de forskjellige spesifikasjonene til komponentene betydde og hvordan de fungerte og kommuniserte i lag med hverandre. Videre jobbet gruppen med montering og fikk der lære hvordan valgene som ble tatt i modelleringen påvirket den faktiske monteringen av dronen. Gruppen fikk erfare hvor viktig der er med en god planlegging.

Under hele prosjektet har gruppen også lært om prosjektstyring. Gruppen har erfart at der er viktig med gode planer og avtaler mellom medlemmene. Ved å ha en god plan som blir oppdatert jevnlig er det enklere å opprettholde framgangen i prosjektet. Det samme gjelder også for gode rutiner, gjennom hele prosjektsperioden har gruppen følgt normale arbeidstider så langt det var mulig. Det kom også godt med å ha faste møter med veiledere der gruppen på forhand utarbeidet en fremdriftsrapport, og kunne diskutere fremdrift og videre plan med veilederene.

5.7.2 CAD-Modellering og produksjon

I starten av prosjektet var gruppen delvis kjent med CAD-verktøy og 3D-printing. Laserkutting var en ny erfaring for begge medlemmer, men vi ble raskt kjent med den tilgjengelige laserkutteren på campus. Prosjektet har bidratt til økt erfaring innenfor disse feltene gjennom en iterativ design og produksjonsprosess hvor vi har forsøkt å balansere vektredusering, vibrasjonsdemping, og robusthet.

5.7.3 ROS

Gruppen har lært svært mye om ROS og hvordan bruke det. I starten av prosjektet hadde gruppen ingen erfaring med ROS. Det ble en bratt læringskurve for gruppen. Gruppen benyttet seg både av gratis ressurser på internett og bøker for å lære seg å bruke ROS. I løpet av prosjektet har gruppen tilegnet seg kunnskap slik at de både kan lage sine egne programmer og benytte de programmene som allerede er tilgjengelig. I tillegg har gruppen lært seg hvordan de skal sette opp hele systemet fra start både på Raspberry Pi og på PC.

Dersom dette prosjektet videreføres i fremtiden anbefaler gruppen at de som tar over også lærer seg ROS og benytter dette. Tiden som blir spart ved å slippe å lage nettverks og inter-prosesss kommunikasjon gjør opp for tiden det tar å lære seg ROS. I tillegg inneholder også ROS andre nyttige funksjoner som for eksempel logging av all kommunikasjon mellom noder. Det finnes også en funksjon for å spille av de loggede verdiene slik at man kan teste hvordan systemet reagere på disse uten å fysisk kjøre roboten. Denne funksjonen kom spesielt godt med når gruppen skulle justere fuzzy logic kontrolleren.

5.7.4 C++

Før dette prosjektet hadde gruppen noe erfaring med C++, men har gjennom prosjektet lært mer. Spesielt har gruppen lært mye om bruken av C++ kombinert med ROS. Gruppen har lært seg hvordan man i C++ abonnerer og publiserer til forskjellige topics med forskjellige meldingstyper. Vi har også lært hvordan man lager bibliotek med klasser som blir gjort tilgjengelige for andre C++ programmer i ROS miljøet. Vi har tilegnet seg nok kunnskap til å utvikle egne programmer, og hvordan tilpasse tilgjengelige programmer til andre formål.

5.7.5 Linux

Siden ROS kun er tilgjengelig på Linux, ble gruppen også tvunget til å lære seg dette operativsystemet. Gjennom prosjektet har gruppen lært seg å bruke mange av funksjonene av funksjonene som er tilgjengelige i linux og har i stor grad jobbet jobbet gjennom terminalen i operativsystemet. I tillegg har gruppen fått erfaring med å sette opp forskjellige typer kommunikasjon mellom enheter som for eksempel SSH mellom Linux enheter og Windows enheter.

6 KONKLUSJON

Gruppen er tilfreds med resultatet oppnådd i denne oppgaven. I starten av oppgaven ble det gitt en større grad av frihet hvor målene ble avgjort. Gruppen tenker disse målene er fullført, og har samtidig hatt verdifulle læreutbytter gjennom utførelsene. Dronen flyr stabilt og er lett å kontrollere. Dronen består hovedsakelig av materialer som var tilgjengelig hos campus Ålesund. Dronens ramme kan repareres uten behov for bestilling av deler da disse kan bygges på campus. I planlegging og produksjonsdelen av prosjektet ble det forsket på de forskjellige komponentene som utgjør et quadcopter og hvordan disse påvirker flygeevnen. Gruppen valgte en kombinasjon av komponenter som resulterte i presise flygeegenskaper og fornuftig flytid. Gruppen har arbeidet og tilegnet seg verdiful erfaring innen bruk av CAD-software, 3d-printer og laserskjærer.

Det autonome systemet som ble utviklet er gruppen også tilfreds med. Systemet oppfører seg i henhold til det gruppen så for seg i begynnelsen av oppgaven. Anti-kollisjon systemet hjelper operatøren av dronen med å unngå kollisjoner, men kan overstyres. Return to home systemet fungerer også meget bra, spesielt i kombinasjon med anti-kollisjon systemet. Totalt sett er det et robust system som sjeldent feiler. Systemet legger til rette for videre utvikling i fremtiden.

Dersom systemet skal viderutvikles har gruppen noen forslag til forbedringer. Designet på rammen kan forbedres med hensyn på at deler enklere kan skiftes. Et kratigere alternativ til Raspberry PI kan vurderes om ytterligere prosesskrevende funksjoner skal implementeres. Selv om anti-kollisjon systemet har prestert bra i testene gjennomført i denne oppgaven, er det ikke testet i mer varierte og utfordrene miljøer. Forbedringspotensialet i anti-kollisjon systemet er derfor stort, og kunne vært en oppgave i seg selv dersom man ønsker at systemet skal fungere i mer utfordrende situasjoner som for eksempel trappeoppganger eller gjennom trange åpninger.

7 VEDLEGG

1. Kildekode
2. 3D-filer
3. Demovideo
4. Risikovurdering
5. Prosjektplaner
6. Beskrivelse av vedlegg

Referanser

[1] Greg Welch, Gary Bishop, et al. “An introduction to the Kalman filter”. In: (1995).

[2] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson education, 2005.

[3] Thomas Ian Grayston. “A Simplified Fuzzy-Logic Control System Approach to Obstacle Avoidance combining Stereoscopic Vision and Sonar”. PhD thesis. Honours thesis, University of Tasmania, 2006.

[4] fullofstars. *Warm fuzzy logic member function.gif*. [Online; accessed Apr, 10, 2022] https://commons.wikimedia.org/w/index.php?curid=2870420. 2007.

[5] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.

[6] Martin Thoma. *Kalman-Filter*. [Online; accessed Jan, 25, 2022] https://commons.wikimedia.org/wiki/File:Kalman-filter_en.svg. 2016.

[7] pk051782510751075. *DJI_F330_QUADROTOR FRAME*. [Online; accessed Jan, 20, 2022] https://grabcad.com/library/dji_f330_quadrotor-frame-1. 2018.

[8] Aggrey Shitsukane. “Fuzzy Logic Sensor Fusion for Obstacle Avoidance Mobile Robot”. In: Aug. 2018.

[9] Michal Šustek and Zdeněk Úředníček. “The Basics of Quadcopter Anatomy”. In: *MATEC Web of Conferences* 210 (Jan. 2018), p. 01001. DOI: 10.1051/matecconf/201821001001.

[10] Suat Karakaya and Hasan Ocak. “Fuzzy logic-based moving obstacle avoidance method”. In: *Global Journal of Computer Sciences: Theory and Research* (2019).

[11] SAE International. *SAE Standards News: J3016 automated-driving graphic update*. [Online; accessed May, 12, 2022] https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic. 2019.

[12] Bård Amundsen. *Droner kartlegger nå hele kysten av Norge*. [Online; accessed May, 12, 2022] https://forskning.no/drone-technology/droner-kartlegger-na-hele-kysten-av-norge/1704107. 2020.

[13] Ardupilot. *Connect ESCs and Motors*. [Online; accessed May, 16, 2022] https://ardupilot.org/copter/docs/connect-escs-and-motors.html. 2021.

[14] Ashish Mishra. *Different Types of Drone Frames: Monocopter to Octocopter*. [Online; accessed May, 12, 2022] https://circuitdigest.com/article/different-types-of-drone-frames-monocopter-to-octocopter. 2021.

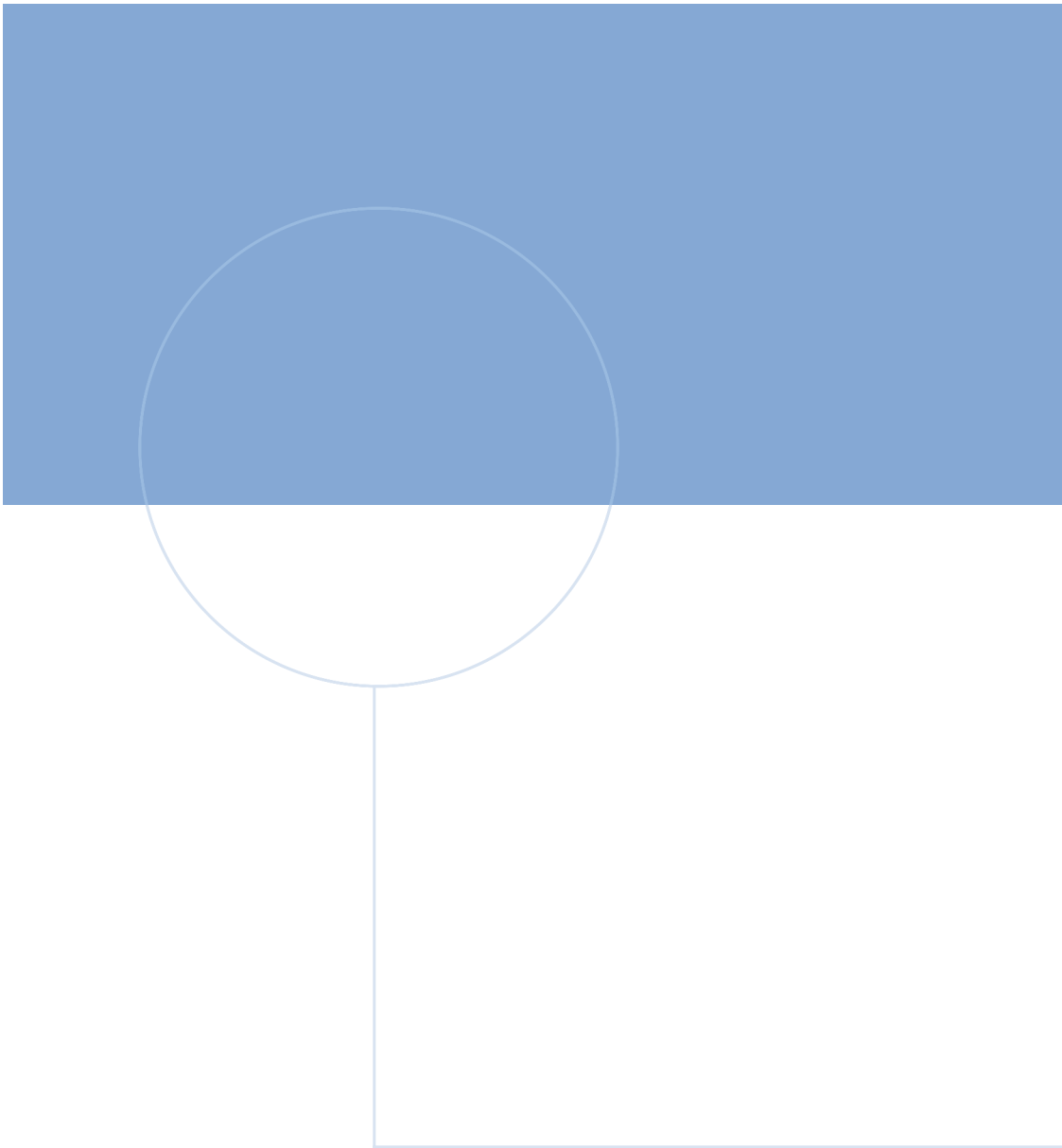
[15] Timothy D Barfoot. *State estimation for robotics*. 2021.

[16] Janne Karin Brodin. *Droner lager digital revolusjon i landbruket*. [Online; accessed May, 12, 2022] https://forskning.no/informasjonsteknologi-landbruk-nmbu-norges-miljo-og-biovitenskapelige-universitet/droner-lager-digital-revolusjon-i-landbruket/1939979. 2021.

[17] PX4 Autopilot. *EKF Overview*. [Online; accessed May, 12, 2022] https://docs.px4.io/v1.12/en/advanced_config/tuning_the_ecl_ekf.html. 2021.

[18] Intel. *Intel Realsense Tracking Camera T265*. [Online; accessed Jan, 26, 2022] https://www.intelrealsense.com/visual-inertial-tracking-case-study/. Unknown.

[19] OpenCV. *Stereo depth*. [Online; accessed Apr, 10, 2022] https://docs.opencv.org/3.4/stereo_depth.jpg. Unknown.



NTNU

Kunnskap for en bedre verden