

Automatisk utskytningssystem for dronesverm

MARTIN MÆLAND
MARTIN D. HERMANSEN
JAN-HENRIK SKOGSTAD

VEILEDER
Kristian Muri Knausgård

Denne siden er bevisst gjort blank.

Forord

Etter tre år ved Universitetet i Agder markerer bacheloroppgaven slutten på et svært lærerikt kapittel. Vi er tre studenter som har samarbeidet om dette prosjektet. Oppgaven har vært det mest utfordrende og krevende arbeidet i løpet av studietiden, og alle fant stor interesse for temaet. Dette har ført til høy motivasjon, og et ønske om å lære og oppnå et resultat vi er fornøyde med.

Mekatronikkstudiet åpner for mange muligheter, både med tanke på arbeidslivet og en eventuell mastergrad. Vi tar med oss mye kunnskap fra bachelorprosjektet, og vi føler oss godt rustet til å utføre varierende arbeidsoppgaver innenfor regulering, elektronikk, data og mekanikk. Vi har lært oss hvordan ting henger sammen, hvordan vi tar for oss et prosjekt i praksis og hvor viktig det er å dokumentere prosesser detaljert. Vår oppfatning er at prosjekter som dette kan være svært viktige for fremtidens teknologi.

Gjennomføring av oppgaven har krevd en sammensetning av kompetanse fra samtlige disipliner tilhørende mekatronikk-utdanningen. For konstruksjon har det vært nødvendig med mekanisk kompetanse både for teknisk tegning, smart produksjon og brukervennlighet. Det har vært sentralt med elektronisk kompetanse for design og sammensetning av de ulike komponentene. Programmeringskunnskap har vært viktig for å få systemet til å fungere i sin helhet på en god måte.

Vi vil takke vår dyktige veileder Kristian Muri Knausgård for gode innspill og oppfølging. I løpet av prosjektperioden har han alltid vært tilgjengelig, og han har tatt både oss og våre spørsmål på alvor. Vi ønsker også å takke resten av mekatronikk-labben og fakultetet for teknologi og realfag. En stor takk for strålende samarbeid.

Grimstad, 28.mai 2021

Martin Mæland

Martin D. Hermansen

Jan-Henrik Skogstad

Denne siden er bevisst gjort blank.

Sammendrag

Droneteknologien utvikler seg i et enormt tempo. De siste årene har store teknologiselskaper forsøkt å fortsette denne trenden med å utvikle dronesvermer som kan anvendes til ulike arbeidsoppgaver i både private og industrielle sammenhenger. Dagens løsning består av å manuelt utplassere dronene før bruk. Dette er en svært ressurs- og tidkrevende prosess om dronesvermen er stor. I tillegg har Universitetet i Agder et ønske om å utvikle en dronesverm som kan brukes til forsknings- og utdanningsformål. Derfor blir det i denne oppgaven sett på muligheter til å utvikle en bedre løsning ved å lage et automatisk utskytingssystem for dronesverm.

Hensikten med prosjektet har vært å utvikle og produsere et automatisk utskytingssystem for dronesverm på opptil hundre droner. For å utvikle et enkelt og effektivt system, ble det utarbeidet ulike krav. Kravene går i korte trekk ut på at systemet må være modulært, skalarbart og dynamisk. I tillegg ble det satt som mål å produsere en prototype til testing av systemet.

Det er utredet ulike konsepter for hvordan det automatiske utskytingssystemet skal fungere. Disse ble vurdert ut i fra graden de oppfyller kravene. Konseptet som ble valgt består av å dele systemet inn i tre deler: én bakkestasjon, ti utskytningsenheter og opptil hundre droner. Dette gjør systemet skalarbart.

Fra bakkestasjonen kan brukeren kontrollere samtlige droner i systemet ved å kommunisere med utskytningsenhettene. Utskytningsenhettene rommer opptil ti droner, og står for oppbevaring og utskytning av disse. Programvaren til systemet er skrevet ved bruk av ROS 2, som gjør det enkelt å utvide funksjonalitetene ved senere anledninger.

Dronen er basert på arbeid fra tidligere studentprosjekter, og det er gjort forbedringer på elektronikk-pakken for å kunne bruke den til testing av systemet. For å fokusere på utviklingen av det automatiske utskytingssystemet, er det inkludert en kommersiell autopilot som står for stabilisering av dronen. Det er også implementert en ettkortsdatamaskin som skal kontrollere autopiloten og kommunisere med utskytningsenheten. Til ettkortsdatamaskinen er det designet et bærekort. Dette bærekortet kan i lengden videreføres til å erstatte autopiloten, og dermed redusere størrelsen og vekten på dronen.

For å teste løsningen av det utviklede systemet, har det blitt produsert én prototype. Prototypen består av én nedskalert versjon av utskytningsenheten og to droner. Resultatene fra testingen har vist at løsningen fungerer, og at den er enkel og effektiv å bruke. Det konkluderes med at det automatiske utskytingssystemet egner seg godt til sverm-eksperimenter.

Denne siden er bevisst gjort blank.

Innhold

Forord	i
Sammendrag	ii
Figurer	iv
Tabeller	v
Formelliste	vi
1 Introduksjon	1
1.1 Bakgrunn og motivasjon	1
1.2 Oppgavebeskrivelse	1
1.3 Rapportens inndeling	2
2 Teori	3
2.1 Elektronikk	3
2.1.1 Pixhawk 4 mini	3
2.1.2 Ettkortsdatamaskin	4
2.2 Programvare	5
2.2.1 Sanntidsoperativsystem	6
2.2.2 DDS og RTPS	6
2.2.3 Robot Operating System	7
2.2.4 PX4 Autopilot	13
2.2.5 PX4/ROS2-bro	14
2.3 Konstruksjon	16
2.3.1 3D-modellering	16
2.3.2 3D-printing	16
2.3.3 Vakuumforming	16
2.3.4 Sprøytestøping	16
2.4 Regelverk og anbefalinger	18
3 Konsept	19
3.1 Krav og kriterier	19
3.1.1 Effektivitet	20
3.1.2 Lisenser, sertifikater og lovverk	21
3.1.3 Kostnad	22
3.1.4 Modulært og skalerbart	23
3.1.5 Kommunikasjon	24
3.1.6 Verifikasjon av systemet	24
3.2 Automatisk utskytingssystem	25
3.2.1 Manuell utplassering av droner	25
3.2.2 Flere utskytingsenheter	26
3.2.3 Utskytingseenhet med magasin	27
3.2.4 En utskytingseenhet	28
3.3 Drone	29

3.3.1	Videreføre utviklingen av drone	29
3.3.2	Utvikle ny drone	29
3.3.3	Inkludere autopilot	29
3.3.4	Kjøpe drone	30
3.4	Konseptvalg	31
3.4.1	Rangering	31
3.4.2	Automatisk utskytningssystem	32
3.4.3	Drone	32
4	Systemarkitektur og -design	33
4.1	Systemdesign	33
4.2	Programvaredesign	35
4.2.1	ROS 2	35
4.3	Bakkestasjon	37
4.3.1	ROS 2 node	37
5	Integrt ettkortsdatamaskin	39
5.1	Valg av operativsystem	39
5.2	Valg av datamaskin	41
5.2.1	Drone	41
5.2.2	Utskytningsenhet	42
5.3	Bærekort til ettkortsdatamaskin	43
5.3.1	Funksjonaliteter	43
5.3.2	Kretsdesign og komponentvalg	44
5.3.3	Kretskortutlegg	49
5.3.4	Produksjon	54
5.4	Oppsett	56
5.4.1	Installasjon av operativsystem	56
5.4.2	Oppstart	58
5.4.3	Nødvendig programvare	65
6	Drone	66
6.1	Elektronikk og komponenter	66
6.1.1	Pixhawk 4 mini/Autopilot	66
6.1.2	Motorer	67
6.1.3	Motorregulator	67
6.1.4	LiPo Batteri	70
6.1.5	Strømmodul	71
6.2	Konstruksjon	72
6.2.1	Oppbygning	72
6.2.2	Produksjon	75
6.2.3	Montering	75
6.3	Programvare	76
6.3.1	Design	76
6.3.2	ROS 2 node	77
6.3.3	Implementasjon	80
7	Utskytningsenhet	85
7.1	Elektronikk og komponenter	85
7.1.1	Datamaskin i utskytningsenhet	85
7.1.2	Batteri	85
7.2	Konstruksjon	86
7.2.1	Oppbygning	87
7.2.2	Produksjon	89
7.3	Programvare	90

7.3.1	Design	90
7.3.2	ROS 2 node	90
7.3.3	Implementasjon	93
8 Testing		94
8.1	Simulering av systemet	95
8.1.1	Formål	95
8.1.2	Utførelse	95
8.1.3	Test-kriterier	95
8.2	Fly en drone med radio	95
8.2.1	Formål	95
8.2.2	Utførelse	95
8.2.3	Test-kriterier	95
8.3	Fly drone ut av utskytningsenhet med radio	96
8.3.1	Formål	96
8.3.2	Utførelse	96
8.3.3	Test-kriterier	96
8.4	Fly drone i utenbordskontroll modus	96
8.4.1	Formål	96
8.4.2	Utførelse	96
8.4.3	Test-kriterier	96
8.4.4	Forbedringer etter test	96
8.5	Fly to droner ut av utskytningsenhet	99
8.5.1	Formål	99
8.5.2	Utførelse	99
8.5.3	Test-kriterier	99
9 Resultater		100
9.1	Drone	100
9.2	Utskytningsenhet	103
9.3	Testing	105
9.3.1	Simulering av systemet	105
9.3.2	Fly en drone med radio	105
9.3.3	Fly drone ut av utskytningsenhet med radio	106
9.3.4	Fly drone i utenbordskontroll modus	106
9.3.5	Fly to droner ut av utskytningsenhet	108
9.4	Det automatiske utskytningssystemet	109
10 Diskusjon		111
10.1	Testing	111
10.1.1	Dronen slet med å fly	111
10.1.2	Fly ut av utskytningsenhet	112
10.1.3	Ustabilt system ved flyging i utenbordskontroll-modus	112
10.2	Drone	113
10.3	Utskytningsenhet	115
10.4	Det automatiske utskytningssystemet	117
11 Konklusjon		119
12 Videre arbeid		120
12.1	Drone	120
12.1.1	Tilpasser droneskrog	120
12.1.2	Kombinere elektroniske komponenter	120
12.2	Utskytningsenhet	121
12.2.1	Videreutvikle funksjonaliteter	121

12.2.2 Ettkortsdatamaskin og nettverk	121
12.3 Det automatiske utskytningssystemet	122
Ordliste	122
Akronymer og forkortelser	125
Bibliografi	126
A Fremgangsmåte for bruk av systemet	130
A.1 Montering	130
A.1.1 Drone	130
A.1.2 Utskytningsenhet	137
A.2 Hvordan bruke systemet	142
A.2.1 Forhåndskonfigurasjon	142
A.2.2 Flyging	145
B UML-skjema	148
C Programvare	150
C.1 Bakkestasjon	150
C.1.1 ROS 2 node i bakkestasjonen	150
C.2 Drone	153
C.2.1 ROS 2 node i drone	153
C.2.2 Oppstarts-skript for drone	156
C.3 Utskytningsenhet	157
C.3.1 ROS 2 node i utskytningsenhet	157
C.3.2 Oppstarts-skript for utskytningsenhet	158
C.4 Annet	159
C.4.1 Automatisk innlogging	159
C.4.2 Shell-skript for ROS 2 installasjon	159
C.4.3 Shell-skript for FastDDS installasjon	161
C.4.4 Shell-skript for nedlasting og bygging av arbeidsområde	162
C.4.5 Python script som logger temperatur av prosessoren til Raspberry Pi Compute Module 4	163
D Kretsskjema	164
D.1 Drone	164
D.2 Utskytningsenhet	168
E Kretsutlegg	172
E.1 Drone	172
E.2 Utskytningsenhet	174
F Komponentliste bærekort	176
F.1 BOM - Bærekort drone	176
F.2 BOM - Bærekort utskytningsenhet	177
G Kostnadslister	178
G.1 Drone	178
G.2 Utskytningsenhet	179
H Tekniske tegninger	180
H.1 Drone	180
H.2 Utskytningsenhet	180

Figurer

2.1	Pixhawk 4 Mini [5]	3
2.2	DDS og DDSI-RTPS modell [26]	7
2.3	ROS 2 arkitektur [29]	8
2.4	Mappestruktur etter 'colcon build'. Filer er fjernet fra treet.	9
2.5	Mappestruktur før colcon build og hvordan et Python arbeidsområde og pakke lages i ROS 2	10
2.6	Test node RQt	12
2.7	Grafisk fremstilling av noder, emner og navnerom i RQt	13
2.8	PX4 og ROS2 kommunikasjon [35]	13
2.9	PX4/ROS2-bro[38]	14
2.10	MicroRTPS agent startes med argument -t UDP	15
2.11	Eksempel på vakuumforming [46]	17
2.12	Eksempel på sprøytestøping [47]	17
3.1	Eksempel på oppsett til krav og kriterier	19
3.2	UML skjema over krav og kriterier for effektivitet	20
3.3	UML skjema over krav og kriterier for lisenser og sertifikater	21
3.4	UML skjema for krav og kriterier for kostnad	22
3.5	UML skjema over krav og kriterier for modulaert og skalerbarhet	23
3.6	UML skjema over krav og kriterier for kommunikasjon	24
3.7	UML skjema over krav og kriterier for mulig verifisering av systemet	24
3.8	Manuelt oppsett av droner	25
3.9	Flere utskytningsenheter	26
3.10	Utskytningsenhet med magasin	27
3.11	En utskytningsenhet	28
3.12	Illustrasjon av valgt konsept	32
4.1	Overordnet systemoversikt. Pilene representer hvilke enheter som kommuniserer og hvilken vei kommunikasjonen går	33
4.2	Oversikt over ønsket kommunikasjonsflyt i ROS 2	36
5.1	Ønsket visning i RQT	41
5.2	Mekanisk tegning av Raspberry Pi Compute Module 4 [58]	42
5.3	Kretsskjema av spenningsregulator-krets	44
5.4	Graf som sammenligner utspenning mot strømmen som går ut	45
5.5	Kretsskjema av strømbryter-krets	46
5.6	Molex konnektor, skyvebryter og GPIO skjematikk	48
5.7	Anbefalt banebredde fra KiCAD sin PCB Calculator med en strøm på 3 A	49
5.8	Teknisk tegning av yttermål på RPi CM4 bærekort til dronen	50
5.9	Komplett bærekort til drone	51
5.10	Teknisk tegning av yttermål RPi CM4 bærekort til utskytningsenheten	52
5.11	Plassering av de største komponentene bærekort utskytningsenhet	53
5.12	Bilde av herdeovnen som ble tatt i bruk	54
5.13	Graf av temperatursyklus	55
5.14	RPi CM4 montert på I/O-brett	56
5.15	Raspberry Pi Imager brukergrensesnitt	57

5.16 Konfigurering av nettverk. Passordet er sensurert for sikkerhetshensyn.	58
5.17 Søk i diagnostikk-meldingene.	59
5.18 Spørsmål i Ubuntu-forum.	60
5.19 Svar i Ubuntu-forum.	62
5.20 Arduino UART test-program.	63
5.21 "Serial0" settes av til konsoll.	64
6.1 GPS modul og Pixhawk 4 mini [71].	66
6.2 EMAX RS 1106 4500KV motor [72]	67
6.3 Electronic Speed Controller brukt i dronen.	68
6.4 Oppsett for ESC programmering.	69
6.5 ESC konfigurasjon.	69
6.6 LiPo-batteri brukt i dronen.	70
6.7 Strømmodul [73].	71
6.8 Sammenstilling i Solidworks av droneskrog.	72
6.9 Underside av skrog i Solidworks.	73
6.10 Første designløsning for innvendig oppsett i drone.	73
6.11 Andre designløsning for innvendig oppsett i drone.	74
6.12 Demontert drone uten bruk av verktøy.	75
6.13 Graf over emner og noder i drone n.	76
6.14 Oppsett for testing av kommunikasjon med sending av sensor data.	81
6.15 Graf over emner og noder ved testing av drone-noden.	82
7.1 Fullskalert utskytningsenhet med ti droner stablet i Solidworks.	87
7.2 Sammenstilling av utskytningsenhet uten lokk i Solidworks.	88
7.3 Sammenstilling av utskytningsenhet med lokk i Solidworks.	89
7.4 Graf over emner og node i utskytningsenhet n.	90
7.5 Mappestruktur oppstart-mappe.	93
8.1 Termisk måling av drone.	97
8.2 Orignal micrortps-agent.	98
9.1 To komplette droner.	100
9.2 Raspberry Pi Compute Module 4.	101
9.3 Komplett bærekort til drone.	101
9.4 Utskytningsenhet uten droner.	103
9.5 Komplett bærekort til utskytningsenheten.	104
9.6 Simulering av systemet.	105
9.7 Flyging med radio-kontroller.	106
9.8 Etter endringer gjort på micrortps-agenten.	107
9.9 Fly to droner ut av utskytningsenhet i utenbordskontroll modus.	108
9.10 Utskytningsenhet med to droner.	109
10.1 Gamle og nye propeller.	111
A.1 Begynner med tom skrogbunn	131
A.2 Monterer motorer i hver arm.	131
A.3 Strømmodul plasseres nederst i dronen.	132
A.4 Innfestning til batteriet settes over strømmodul ved hjelp av stolper i skroget.	132
A.5 Batteriet settes inn.	133
A.6 Innfestning for RPi CM4 tres over batteriet ved hjelp av stolpene.	133
A.7 RPi CM4 og bærekort blir skrudd på plass med fire skruer.	134
A.8 Innfestning for Pixhawk 4 mini settes oppå skruene til RPi CM4 og bærekort.	134
A.9 Pixhawk 4 mini plasseres mellom hjørnekantene.	135
A.10 Propeller blir skrudd på motorene.	135
A.11 Lokk settes på.	136

A.12 Ferdig montert drone.	136
A.13 Begynner med elektronikkboksen.	138
A.14 Føringsstolpene til langsiden av dronene settes inn og skrus fast med åtte skruer under elektronikkboksen.	138
A.15 Pleksivegger skyves på plass inn i ferdiglagde spor.	139
A.16 Bærekort med RPi CM4 skrus fast med fire skruer. Batteriet blir lagt på plass mellom egne sporvegger.	139
A.17 Tre pleksigulvplater blir lagt oppå og skrudd fast i hvert hjørne på elektronikkboksen.	140
A.18 Føringsstolper til kortsidan av dronene blir plassert i utkuttede spor og skrudd fast i nederste pleksigulv.	140
A.19 Utskytningsenhet ferdig montert med tre droner.	141
A.20 Brukergrensesnitt Angry IP Scanner.	142
A.21 Skru på utskytningsenheten.	143
A.22 Tekstfil for oppsett av identifikasjon for utskytningsenhet.	143
A.23 Tekstfil for oppsett av identifikasjon for drone.	144
A.24 Plasser dronen i den bestemte utskytningsenheten.	144
A.25 Plassering av utskytningsenhet på åpent område.	145
A.26 ROS 2 noden på bakkestasjonen.	145
A.27 Eksempel på kommando.	146
A.28 Liste over kommandoer.	147
 B.1 Fullt UML-skjema av krav og kriterier for drone og utskytningsenhet konsepter.	149
 E.1 Drone, kretskortutlegg	173
E.2 Utskytningsenhet, Kretskortutlegg	175
 H.1 Drone, Bill Of Materials	181
H.2 Drone, SkrogLokk	182
H.3 Drone, SkrogBunn	183
H.4 Drone, Innfestning-sammenstilling	184
H.5 USE, Motor-Sammenstilling	185
H.6 USE - Bill Of Materials	186
H.7 USE Nedre Konstruksjon	187
H.8 Pleksivegg Nede	188
H.9 Pleksivegg nede foran	189
H.10 Pleksigulv 1	190
H.11 Pleksigulv 2	191
H.12 Pleksigulv 3	192
H.13 Føringsstolpe Langside	193
H.14 Føringsstolpe Kortside	194
H.15 USE Nedre Sammenstilling	195
H.16 Lokk	196
H.17 Lokk Pleksivegg	197
H.18 Lokk Sammenstilling	198

Denne siden er bevisst gjort blank.

Tabeller

3.1	Rangering drone	31
3.2	Rangering utskytningsenhet	31
F.1	BOM - Bærekort drone.	176
F.2	BOM - Bærekort utskytningsenhet	177
G.1	Kostnadsliste, drone.	178
G.2	Kostnadsliste, utskytningsenhet.	179

Denne siden er bevisst gjort blank.

Formelliste

5.1	Forhold mellom R6 og R7 for ønsket utspenning.	44
5.2	Strøm gjennom R6.	45
5.3	Strøm gjennom R7.	45
5.4	Strøm gjennom R9	46
5.5	Utspenning av MOSFET driver.	46
5.6	Raspberry Pi Compute Module 4 avgitt effekt.	46

Denne siden er bevisst gjort blank.

Kapittel 1

Introduksjon

1.1 Bakgrunn og motivasjon

Droneteknologien har utviklet seg betydelig siden starten av 2000-tallet. Droner har i stor grad blitt kommersialisert, og brukes blant annet til filming, racing, overvåkning og levering av varer. I løpet av de siste årene har teknologiselskaper startet med forskning og utvikling av dronesvermer. Intel markedsfører sin teknologi ved å selge lysshow med dronesvermer bestående av flere hundre droner [1]. I fremtiden vil dronesvermer mest sannsynlig benyttes i både private og industrielle sammenhenger.

Dagens løsning består av å manuelt utplassere droner brukt i dronesvermer. Dette er en svært ressurs- og tidkrevende prosess om dronesvermen er stor. En video av Guinness World Records på YouTube fra 2020 viser dette tydelig. I videoen flyr 2066 droner samtidig, alle er manuelt utplassert [2].

I tillegg har Universitetet i Agder et ønske om å utvikle en dronesverm som kan brukes til forsknings- og utdanningsformål. Tidligere er det arbeidet med tre iterasjoner av drone-prosjekter på Universitet i Agder. Da har fokuset vært å designe og konstruere en modulær quad-rotor og drone-lab. Det er et ønske å bruke denne dronen videre til å utvikle en dronesverm.

1.2 Oppgavebeskrivelse

Oppgaven er å designe og konstruere et automatisk utskytningssystem til en dronesverm på opptil 100 droner. Systemet må designes til å bli brukt av en videreutviklet modulær quad-rotor laget av BSc.-oppgaver våren 2018 og 2019 [3] [4]. Designløsningen til det automatiske utskytningssystemet må testes. Derfor må det arbeides med å utvikle en flygbar drone. Det må også utarbeides en plan for hvordan dronesvermen skal lande. Det er essensielt at dronen og det automatiske utskytnings-systemet utvikles for å være modulært og skalerbart til videre arbeid med dronesverm-prosjektet.

1.3 Rapportens inndeling

Rapporten er delt inn i tolv kapitler. Første kapittel gir et innblikk i hva oppgaven som skal løses går ut på. Deretter blir den bakenforliggende teorien som er brukt under arbeidsprosessen forklart. Dette er for å gjøre leseren mer inneforstått med hva som kommer videre i metode- og resultatkapitlene.

Metodedelen av rapporten er delt inn i seks kapitler og innledes med konsept-kapittelet. Der blir de ulike konseptene til det automatiske utskytningssystemet beskrevet og rangert opp mot hverandre før det blir valgt. Deretter blir systemarkitekturen fremstilt. Den beskriver hvordan systemet er planlagt å fungere. Videre kommer kapitlene om ettkortsdatamaskin, drone og utskytningsenhet. Der blir prosessen bak selve designingen av det automatiske utskytningssystemet forklart. Det siste kapittelet i metodedelen presenterer formålet og strukturen til testene som er utført.

Resultatene i metodedelen er fremstilt i kapittel ni. Deretter blir resultatene diskutert i diskusjonskapittelet. Diskusjonen omhandler også hvordan designløsningen samsvarer med oppgavebeskrivelsen og kravene/kriteriene.

Hoveddelen av oppgaven avsluttes med konklusjonen som oppsummerer arbeidet, resultatene og diskusjonen. Rapporten avsluttes med et kapittel om videre arbeid. I kapittelet blir potensielt videre arbeid til det overordnede prosjektet undersøkt og diskutert. Gruppen kommer også med anbefalinger om videre arbeid i prosjektet.

Det er en fordel å lese teori og metode for å oppnå bedre forståelse av systemet. Det anbefales også å lese hver test samlet. Det vil si at metoden, resultatet og diskusjonen av én test bør leses ferdig før neste, slik at leseren kan få en bedre oppfatning av hvorfor de ulike testene ble utført. I tillegg er det en ord- og akronym/forkortelses liste etter videre arbeid kapittelet.

Kapittel 2

Teori

2.1 Elektronikk

2.1.1 Pixhawk 4 mini

Pixhawk 4 mini er en autopilot fra Pixhawk-programmet i samarbeid med Holybro og Auterion. Pixhawk 4 mini baserer seg på Pixhawk FMUv5 design-standarden og er optimalisert for å kjøre autopilot-programvaren PX4-Autopilot (PX4), se 2.2.4. Autopiloten gjør det det enkelt å stabilisere droner og få de flygbare. Internt har Pixhawk 4 mini en 32 Bit ARM CPU med klokkehastighet på 215 MHz. Samt ulike sensorer som to akselerometer/gyroskop med 6 frihetsgrader, et magnetometer og et barometer. Autopiloten kommer med en rekke spesifiserte innganger og utganger som benytter JST GH konnektorer. Disse portene er radiokontroll, CAN, GPS, telemetri, UART/I2C og strømforsyning. I tillegg er det åtte PWM-utganger som benyttes til å styre ulike motorer, fire PWM-innganger og en ADC-inngang [5].



Figur 2.1: Pixhawk 4 Mini [5]

Moduser

Pixhawk 4 mini med PX4-Autopilot programvare kan bli satt i flere ulike moduser. Blant annet full manuell kontroll (manual), stabiliseringsmodus (stabilized) og utenbordskontroll (offboard). Det varierer mellom modusene hva Pixhawk 4 mini forventer for å bli satt i en modus. Ved manuell kontroll kreves det minimalt av Pixhawk 4 mini, mens den krever noe mer ved stabiliseringsmodus. For å innta utenbordskontroll modus er det ytterligere strengere krav. Følgende krav må tilfredsstilles for at Pixhawk 4 mini skal gå inn i utenbordskontroll modus: må ha posisjonkontroll, for eksempel GPS, få en konstant strøm av settpunkter på minst 2 Hz, radio kontrolleren må være i dvale-modus, men må kunne ta kontroll over dronen til enhver tid, og dronen må være armert (armed) før det kan bli satt i utenbordskontroll modus [6].

2.1.2 Ettkortsdatamaskin

En ettkortsdatamaskin er en liten datamaskin som er laget på ett enkelt kretskort. Dette kretskortet deler felles struktur med en vanlig datamaskin, bare i miniatyr versjon. Det vil si at kretskortet hovedsakelig er bygget på mikroprosessor(er), minne og I/O. Ettkortsdatamaskiner blir ofte kalt for SBCer, det kommer fra det engelske begrepet "Single-board computer". Den mest typiske aplikasjonen for ettkortsdatamaskiner er som innebygde datakontrollere i ulike elektroniske apparat, men kan også brukes til de fleste andre formål som det normalt ville blitt brukt en datamaskin til. Dette er fordi de som oftest er svært modulære da de ofte har mange ubrukte GPIO pins som ligger lett tilgjengelige på brettet. Disse kan brukes til spesifikke ønskede formål [7].

Enkelte ettkortsdatamaskiner kommer uten generell Input/Output (I/O). Disse kalles "Computer-on-module" (COM). COMs er ettkortsdatamaskiner som er bygget for kravstore brukere som er avhengig av meget spesifikke funksjoner, uten unødvendig vekt eller plassbruk. For å få til dette benytter COMs seg av bærekort. Bærekortet kan designes av brukeren for å oppnå ønskede funksjoner. Det kan blant annet være ønskede innganger, utganger, sensorer, etc. Da COMs kun inneholder det mest nødvendige er de ofte svært kompakte, samtidig som de er høyst integrerte [8].

2.2 Programvare

Programvare er instruksjonene som forteller en datamaskin hva den skal gjøre. Det skiller mellom flere typer programvare, men de to vanligste er; programvare som driver systemer og programvare som driver applikasjoner. Programvaren som driver systemet kalles for operativsystem. Operativsystemet har som jobb å kontrollere alle maskinens ressurser, og se til at disse fungerer som forventet. Programvaren som driver applikasjoner har som jobb å utføre oppgavene som brukeren gir datamaskinen og som typisk gir informasjon til brukeren [9].

Linux

Linux er en av flere Unix-lignende operativsystem, og ble lansert i 1991 av Linus Torvalds for å bli brukt som kjerne for IBM datamaskiner. Linuxkjernen til Linus Torvalds er senere slått sammen med operativsystemet GNU, som humoristisk står for «GNU's Not Unix», og har da dannet operativsystemet Linux [10]. Operativsystemet er svært utbredt blant utviklere fordi det er et ikke-kommersielt operativsystem som går under "GNU General Public License (GPL)" [11]. Mens Linux er en etterligning av Unix, er det laget mange operativsystem som baserer seg på Linux.

Ubuntu

Ubuntu baserer seg på linuxdistribusjonen Debian, og er den mest bruke linuxdistribusjonen. Utgivelsene av Ubuntu kommer fra Canonical, et selskap som leverer programvare med åpen kildekode, som betyr at det er gratis å laste ned, bruke og dele. Utviklingen styres av et stort samfunn bestående av frivillige og andre selskaper som ønsker å lage en plattform som alle kan bruke [12]. Ubuntu tilbyr ulike versjoner, den vanligste er Ubuntu Desktop, som har et grafisk brukergrensesnitt likt Windows og Mac OS [13]. Ubuntu Server er en "hodeløs" versjon av Ubuntu Desktop, det vil si at det ikke er noe grafisk brukergrensesnitt, men et kommandolinjegrensesnitt [14]. Ubuntu Core er et minimalistisk operativsystem beregnet for innebygde systemer og spesialiserte applikasjoner. Med minimalistisk menes det at operativsystemet kun kommer med de nødvendigste pakkene som kreves for å kjøre, dette fører til at operativsystemet er lite og sikkert. En maskin som kjører Ubuntu Core brukes vanligvis over Secure Shell (SSH) [15]. SSH er et dataprogram og en nettverksprotokoll som gjør det mulig å få tilgang til en datamaskins kommandolinje.

ArchLinux

Arch Linux er et operativsystem som er en lettvekts og fleksibel Linux distribusjon som ifølge dem selv "Tries to Keep It Simple" [16]. Målet til utviklerne bak Arch Linux er å ha en Linuxdistribusjon som er bruker-sentrisk, det vil si at de ønsker at brukeren skal ha mulighet til å gjøre hva enn den ønsker, så langt brukeren er kapabel til det. For å bruke operativsystemet blir brukeren møtt med en minimalistisk base som kun inneholder det nødvendige. Det vil si at kun de mest nødvendige verktøyene er installert, samtidig som det er tilstrekkelig for at brukeren selv kan konfigurere sin installasjon [17].

NuttX

NuttX er et sanntidsoperativsystem (RTOS) som ble lansert i 2007 av Gregory Nutt. Målet med NuttX er å ha et sanntidsoperativsystem som kan brukes gjennom standardiserte operativsystemer. NuttX bruker standardene Portable Operating System Interface (POSIX) og American National Standards Institute (ANSI). Det er gjort gjennom å fordele ut kildekoden i mindre filer og flere statiske biblioteker. Selv om NuttX er et lite operativsystem, er det designet for å være skalerbart, og kan brukes i mikrokontrollere fra 8-bit til 64-bit [18]. Med et høyt fokus på å holde seg innenfor POSIX og ANSI sine standarder, er det enkelt å overføre og bruke filer fra Linux i NuttX [19].

Shell

I Linux og Unix operativsystem er skallet det første som brukeren møter. "Shell" (skallet) er et Unix uttrykk er brukerens kommandolinjegrensesnitt med systemet [20]. Nå for tiden er grafiske skrivebordsmiljø som brukerens grensesnitt dominerende. Bak det grafiske brukergrensesnittet ligger det en terminal som også kan bli brukt som grensesnitt for brukeren. Med brukergrensesnitt menes brukerens kommunikasjon med systemet gjennom skjerm, tastatur og eventuelt datamus. Ved bruk av grafisk brukergrensesnitt klikker brukeren seg rundt på skrivebordet, mens ved bruk av kommandolinjegrensesnitt må brukeren skrive kommandoer inn i en terminal [21].

I tillegg til at shell er et brukergrensesnitt kan det lages og kjøres shell-skript på Unix baserte operativsystem. Shell-skripts er dataprogram som inneholder shell-kommandoer og utfører disse. Shell-skripting er kjent for å være et avansert kodespråk, og blir derfor oftest brukt til enklere koder, selv om det også er laget program ved bruk av shell-skripting [22].

Bash (Bourne Again Shell) er skallet som kommer ved installasjonen av Ubuntu. Slik at når en terminal blir åpnet i skrivebordsversjonen av Ubuntu, eller en jobber i Ubuntu Server, er det bash som brukes.

2.2.1 Sanntidsoperativsystem

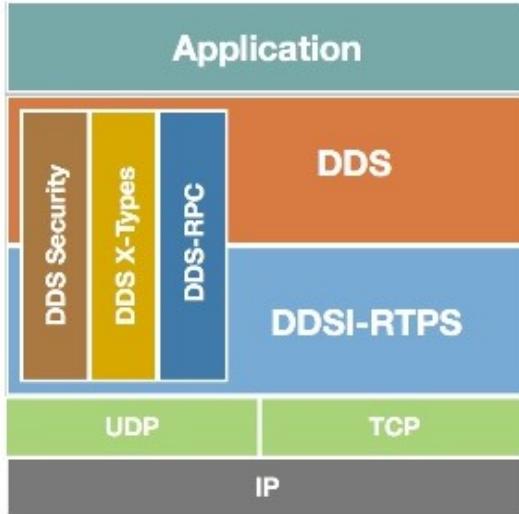
Et sanntidsoperativsystem (RTOS) må i tillegg til vanlige oppgaver som et operativsystem har, prioritere at kritiske oppgaver blir gjort innen tidsfristen. Det er vanlig å dele RTOS inn i to hovedtyper; myk-sanntid og hard-sanntid. I et mykt sanntidssystem er det ikke av stor betydning om tidsfristen alltid holdes, så lenge den som oftest gjør det. I et hardt sanntidssystem må systemet alltid holde tidsfristen [23].

Strømming av video er et myk-sanntid, da det ikke vil ha store konsekvenser for brukeren om tidsfristen holdes. I motsetning er airbaggen i en bil et eksempel på hard-sanntid. Dersom airbaggen ikke møter tidsfristen, enten om den løser ut for tidlig eller for sent, kan det ha katastrofale konsekvenser for brukeren. En applikasjon som kjører på et hardt sanntidssystem kan bli omtalt som deterministisk dersom oppgavene kan utføres innen en gitt margin av tidsfristen [23].

2.2.2 DDS og RTPS

Data Distribution Service (DDS) er en ende-til-ende middelvare-spesifikasjon som bruker et publish-subscribe-mønster for transport. Publish-subscribe transport vil si at data ikke blir sendt direkte til en mottaker, men at det blir sent til et tema. Slik at andre applikasjoner på samme maskin eller nettverk kan velge å lytte til dette temaet eller ikke [24].

For oppnå interoperabilitet benytter DDS seg av "Real Time Publish and Subscribe" (RTPS) tråd-protokollen, også kalt DDSI-RTPS. DDSI-RTPS er standardisert av Object Management Group og har som mål å opprette stabil kommunikasjon over upålidelige transport-protokoller, som for eksempel UDP [25]. DDS og DDSI-RTPS ligger mellom transport-laget og applikasjons-laget i Open Systems Interconnection (OSI)-modellen, som vist i Figur 2.2.

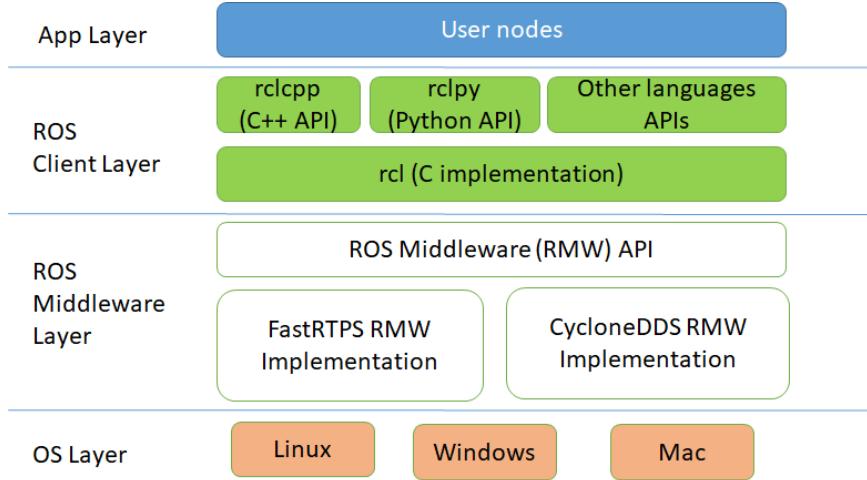


Figur 2.2: DDS og DDSI-RTPS modell [26].

2.2.3 Robot Operating System

Robot Operating System (ROS) er ikke et operativsystem, men en middelvare for robot programvareutvikling utviklet av Open Robotics. Middelvare er programvare som ligger mellom operativsystemet og applikasjonene som kjører. I praksis fungerer middelvaren som et skjult kommunikasjonslag som deler data mellom applikasjoner [27]. ROS gjør det enklere for brukerne å utvikle programvare ved å tilby en rekke biblioteker og verktøy. I tillegg er alt åpen kildekode, slik at alle i ROS-samfunnet kan bidra i utviklingen og publisere egendesignede pakker. Disse pakkene kan enkelt lastes ned og kjøres sømløst i ROS, gitt at det er benyttet samme versjon. Dersom enkelte pakker ikke er tilgjengelige kan disse lages selv ved å benytte verktøyene ROS tilbyr. Kode skrevet i ROS må bruke et av de støttede programmeringsspråkene Python eller C++.

Robot Operating System 2 (ROS 2) er den nyeste versjonen av ROS. ROS 2 bygger, i motsetning til ROS 1, på Data Distribution Service (DDS). ROS 2 støtter flere leverandører av DDS, blant annet eProsima sin Fast RTPS og Eclipse sin Cyclone DDS. Både Fast RTPS og Cyclone DDS ligger i de binære pakkene når man laster ned ROS 2 [28]. Det er vist i figur 2.3 at ROS sin middelvare bygger på en allerede eksisterende DDS-implementasjon. Over middelvaren ligger ROS sine egne Application Programming Interface (API) for enklere utvikling av programvare.



Figur 2.3: ROS 2 arkitektur [29].

ROS 2 arbeidsflyt

For å bruke verktøyene ROS 2 har tilgjengelig må programmet kjøres. Det blir gjort ved å åpne terminalen på operativsystemet, for deretter å kjøre en bash-kode som ligger i installasjonsmappen av ROS 2. Kommandoen `source <vei_til_installasjonsmappen>/setup.bash` gjør dette. Når ”setup.bash” er kjørt, er verktøyene kun tilgjengelige i terminalvinduet som utførte kommandoen. Deretter kan blant annet arbeidsområder og pakker lages. Dokumentasjonen til ROS 2 forklarer hvordan arbeidsområder og pakker lages, samt hvordan verktøyene brukes [30].

Før det pakker lages bør det opprettes et dedikert arbeidsområde. Arbeidsområdet er en mappestruktur som inneholder ROS 2 pakker. For å ha et ryddig arbeidsområde bør mappestrukturen se slik ut; `/navn_på_arbeidsområde_ws/src/pakker`. Grunnen til at ”ws” blir lagt til i mappenavnet er fordi det står for workspace (arbeidsområde), for å vise at det er et dedikert arbeidsområde, og at ingen filer som ikke tilhører arbeidsområdet skal legges inn i mappen. Pakkene som legges inn i arbeidsområdet blir lagt i ”src” mappen. I arbeidsområdet kan pakkene bygges til kjørbare ROS 2 program. For å bygge pakkene som ligger i et arbeidsområde blir kommandoen `colcon build` kjørt i terminalen i den ytterste mappen av arbeidsområdet. Før `colcon build` blir kjørt er det viktig å ha sourcenet ROS 2. `Colcon build` oppretter tre mapper ytterst i arbeidsområde, disse mappene er ”build”, ”install” og ”log”. I ”build” ligger filer som tilhører selve byggeprosessen, i ”install” ligger selve installasjonen av pakkene i ROS 2 arbeidsområdet, og i ”log” ligger loggene fra byggeprosessen. Se Figur 2.4 for mappestrukturen i arbeidsområdet. Da disse er opprettet kan nodene som ligger i pakkene kjøres med kommandoen `ros2 run <pakkenavn> <nodenavn>`. Før nodene kan kjøres må arbeidsområdet bli startet på samme måte som ROS 2 blir startet. Det vil si at i ”install” mappen ligger det en fil kalt `local_setup.bash` som først må kjøres (sources).

```

rostest1@rostest1-VirtualBox:~/test_ws$ colcon build
Starting >>> test_pkg
Finished <<< test_pkg [1.10s]

Summary: 1 package finished [1.36s]
rostest1@rostest1-VirtualBox:~/test_ws$ tree -d
.
├── build
│   └── test_pkg
│       ├── build
│       │   └── lib
│       │       └── test_pkg
│       │           └── test_pkg.egg-info
│       └── install
│           └── test_pkg
│               ├── lib
│               │   └── python3.8
│               │       └── site-packages
│               │           └── test_pkg
│               │               └── __pycache__
│               │                   └── test_pkg-0.0.0-py3.8.egg-info
│               └── test_pkg
│                   ├── ament_index
│                   │   └── resource_index
│                   │       └── packages
│                   └── colcon-core
│                       └── packages
│                           └── test_pkg
│                               └── hook
└── log
    ├── build_2021-05-13_11-32-32
    │   └── test_pkg
    ├── build_2021-05-13_11-33-48
    │   └── test_pkg
    └── latest -> latest_build
        └── latest_build -> build_2021-05-13_11-33-48
src
└── test_pkg
    ├── resource
    └── test
        └── test_pkg

35 directories

```

Figur 2.4: Mappestruktur etter 'colcon build'. Filer er fjernet fra treeet.

En ROS 2 pakke blir laget med kommandoen i Kodelisting 2.1, om en vet navnet på noden som skal ligge i pakken kan argumentet `--node-name <nodenavn>` legges til. Ved å kjøre kommandoen blir det opprettet en mappe med pakkenavnet som ble skrevet i kommandoen, om argumentet med nodenavn ble lagt til, vil det lages en mappe og en fil i pakken med nodenavnet. Filtypen varier på om det ble laget en C++ eller Python pakke, som blir gjort med `--build-type` argumentet. I pakken blir det laget flere andre mapper og filer også. Filtypene i disse er på lik linje med noden ulike mellom C++ og Python pakker. Essensen i en ROS 2 pakke er den samme selv om det er benyttet forskjellige språk. Det ligger uansett en fil kalt `package.xml` som inneholder noen av avhengighetene til pakken, samt byggetype og pakkeinformasjon. En annen fil som er viktig er oppsettfilen, i Python heter den `setup.py`, mens i C++ heter den `CMakeLists.txt`. Disse inneholder alle avhengighetene som behøves i pakken.

Kodelisting 2.1: Kommando for å lage en pakke i ROS 2.

```
$ ros2 pkg create --build-type ament_cmake/ament_Python <pakkenavn>
```

En node i ROS 2 er egentlig kun en fil som inneholder kode av et programmeringsspråk. Det vanligste er at en node kun har en bestemt arbeidsoppgave. Som vil si at om det kreves flere større arbeidsoppgaver fra et system blir det heller opprettet nye noder eller pakker. I en node kan det opprettes emner som noden kan abonnere på eller publisere til. Disse emnene inneholder meldinger med variabler som blir transportert av DDS fra og til noder.

```
rostest1@rostest1-VirtualBox:~/test_ws
```

```
rostest1@rostest1-VirtualBox:~$ mkdir -p test_ws/src
rostest1@rostest1-VirtualBox:~$ cd test_ws/src/
rostest1@rostest1-VirtualBox:~/test_ws/src$ source /opt/ros/foxy/setup.bash
rostest1@rostest1-VirtualBox:~/test_ws/src$ ros2 pkg create --build-type ament_python --node-name test_node test_pkg
going to create a new package
package name: test_pkg
destination directory: /home/rostest1/test_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['rostest1 <martin.hermansen@hotmail.com>']
licenses: ['TODO: License declaration']
build type: ament_python
dependencies: []
node_name: test_node
creating folder ./test_pkg
creating ./test_pkg/package.xml
creating source folder
creating folder ./test_pkg/test_pkg
creating ./test_pkg/setup.py
creating ./test_pkg/setup.cfg
creating folder ./test_pkg/resource
creating ./test_pkg/resource/test_pkg
creating ./test_pkg/test_pkg/_init_.py
creating folder ./test_pkg/test
creating ./test_pkg/test/test_copyright.py
creating ./test_pkg/test/test_flake8.py
creating ./test_pkg/test/test_pep257.py
creating ./test_pkg/test_pkg/test_node.py
rostest1@rostest1-VirtualBox:~/test_ws/src$ cd ..
rostest1@rostest1-VirtualBox:~/test_ws$ tree
└── src
    └── test_pkg
        ├── package.xml
        ├── resource
        │   └── test_pkg
        ├── setup.cfg
        ├── setup.py
        └── test
            ├── test_copyright.py
            ├── test_flake8.py
            └── test_pep257.py
            └── test_pkg
                └── __init__.py
                    └── test_node.py
5 directories, 9 files
```

Figur 2.5: Mappestruktur før colcon build og hvordan et Python arbeidsområde og pakke lages i ROS 2.

For å ha oversikt over emnene og nodene i ROS 2 er det viktig med god struktur ved navngiving. Et verktøy som kan brukes til dette er navnerom (namespaces). Navnerom gjør det mulig å opprette flere instanser av emner og noder, men med ulikt navn. For eksempel kan emnet <emnenavn> bli gitt et navnerom. Da må noden som ønsker å abonnere på eller publisere til emnet referere til /<navnerom>/<emnenavn>. Se Figur 2.7.

En ROS 2-node skrevet i Python er bygd opp på følgende måte. Først importeres standardbiblioteket som hører til ROS 2 som inneholder nyttige klasser og funksjoner. I tillegg importeres meldingstypene som er planlagt å bruke i noden. Se Kodelisting 2.2.

Kodelisting 2.2: Importering av biblioteker og meldingstyper i ROS 2

```

1 # Importering av standard-biblioteket
2 import rclpy
3 from rclpy.node import Node
4
5 # Importering av meldingstyper
6 from std_msgs.msg import String

```

Videre må node-klassen defineres, her er den kalt `Test`. `Test`-klassen arver fra `Node`-klassen i `rclpy.node` og har med det en rekke funksjonaliteter. I `Test`-klassen defineres det to funksjoner; én instansierings-funksjon, og én funksjon som kalles hver gang abonnereren mottar en melding. I instansierings-funksjonen blir forelder-klassen instansiert med navnet `test_node`, som vist på linje 12 i Kodelisting 2.3.

Kodelisting 2.3: Definisjon av node-klasse.

```

8 class Test(Node):
9
10     def __init__(self):
11         # Instansiering av node
12         super().__init__('test_node')
13
14         # Instansiering av publiserer
15         self.test_publiser_ = self.create_publisher(String, "test_emne_02", 1)
16
17         # Instansiering av abonnerer
18         self.test_abonnerer_ = self.create_subscription(String, "test_emne_01",
19             self.mottatt_melding, 1)
20
21         # Print den mottatte meldingen
22         def mottatt_melding(self, melding):
23             self.test_publiser_.publish(melding)

```

På linje 15 i Kodelisting 2.3 lages det en publiser ved å kalle funksjonen `create_publisher` fra forelder-klassen. Samtidig settes tre parametere: Meldingstype, emne som meldingen skal publiseres på og antall meldinger som kan ligge i meldingskøen. Respektivt settes disse parameterene til `String`, `"test_emne_02"` og `1`.

På linje 18 i Kodelisting 2.3 lages det en abonner ved å kalle funksjonen `create_subscription` fra forelder-klassen. I denne instansieringen settes fire parametere: Meldingstype, emne, en funksjon som kalles hver gang abonnereren leser en melding fra emnet og antall meldinger som kan ligge i meldingskøen. Disse parameterene settes til `String`, `test_emne_01`, `self.mottatt_melding` og `1`. Funksjonen som kalles når abonnereren mottar meldingen defineres på linje 21 i Kodelisting 2.3, og tar inn to parametere: Seg selv, og meldingen som abonnereren har mottatt. Videre utfører denne funksjonen en handling, den publiserer meldingen videre til emnet `test_emne_02` ved å bruke publiseren definert tidligere.

Mot slutten av koden, linje 25 i Kodelisting 2.4, defineres main `main`-funksjonen, dette er funksjonen som kjører når noden kjøres fra ROS 2. I `main`-funksjonen lages det en instanse av `Test`-klassen, her navngitt `test_node`. Videre kalles `rclpy.spin(test_node)` fra hovedbiblioteket, denne funksjonen setter i gang noden og gjør at abonnenter starter å abonnere på `test_emne_01`.

Kodelisting 2.4: Definisjon av node-klasse.

```

24 # Hovedfunksjon som kjører når noden kalles fra ROS 2
25 def main(args=None):
26     rclpy.init(args=args)
27     test_node = Test()
28     rclpy.spin(test_node)
29     test_node.destroy_node()
30     rclpy.shutdown()
31
32 if __name__ == '__main__':
33     main()
34     len(string)

```

I RQt-verktøyet kan noden og emnene observeres og verifiseres. I Figur 2.6 er det tydelig at det er opprettet en node kalt `/test_node` som abонnerer på emnet `/test_emne_01` og publiserer til emnet `/test_emne_02`.

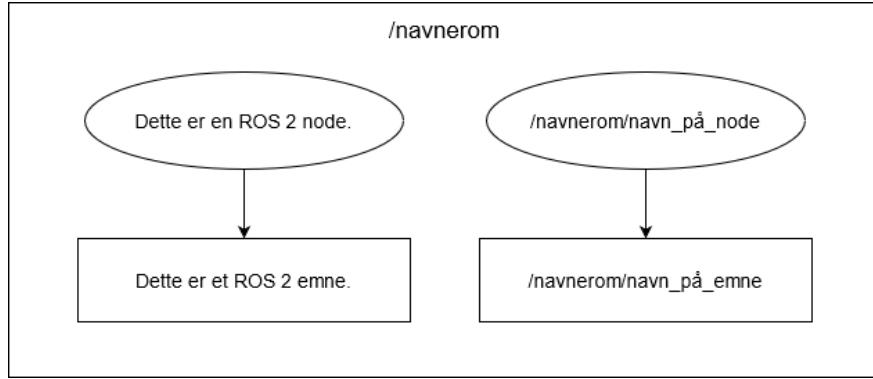


Figur 2.6: Test node RQt.

ROS 2 kommunikasjonsflyt

RQt er et verktøy med grafisk grensesnitt i ROS 2. Verktøyet blir installert samtidig som ROS 2 om skrivebordsversjonen installeres, men det kan også installeres separat ved en senere anledning. I RQt er det flere tilleggsmoduler som er nyttige å bruke til testing, observering og verifisering av ROS 2 systemer, blant annet kan emner både lyttes- og skrives til [31].

Når RQt viser en grafisk representasjon av kommunikasjonsflyten til ROS 2 blir noder, emner og navnene vist på samme måte som Figur 2.7. Ved beskriving av kommunikasjonsflyten i ROS 2 vil dette oppsettet bli brukt videre i rapporten.



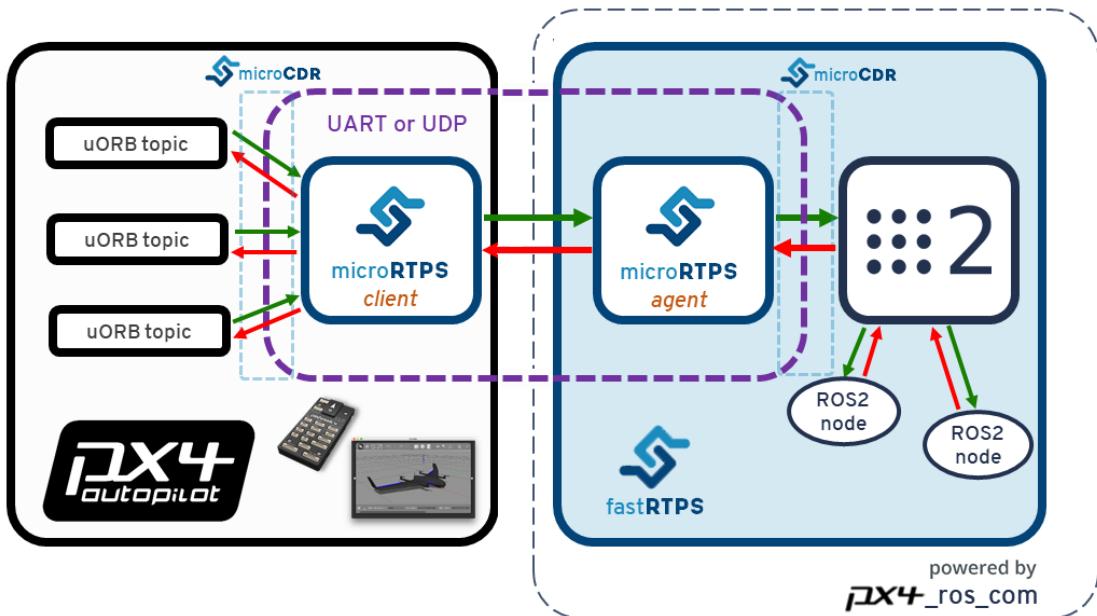
Figur 2.7: Grafisk fremstilling av noder, emner og navnerom i RQt.

2.2.4 PX4 Autopilot

PX4-Autopilot er en programvare med åpen kildekode til autopilot-systemer og er opprettholdt av Dronecode Foundation. Dronecode Foundation er en organisasjon som ønsker å lage en åpen kildekode-platform for droneindustrien. Organisasjonen er blant annet støttet av Auterion og Microsoft [32]. PX4 Autopilot-programvaren tilbyr navigasjon, kontroll-algoritmer for autonom flying, sikkerhetsfunksjoner og ulike estimatorer [33]. Dronecode Foundation tilbyr også en lettvekts kommunikasjonsprotokoll (MAVLINK), et bibliotek (MAVSDK) for enklere programvare-utvikling med MAVLINK og en kontrollstasjon for droner basert på MAVLINK (QGroundControl) [32].

Internt bruker PX4 meldingssystemet uORB, som er et meldings programmeringsgrensesnitt som benytter samme meldingsmønster som ROS, ”publish-subscribe”. Dette gjør at det kan opprettes kommunikasjon mellom en autopilot som kjører PX4-Autopilot, og en ekstern datamaskin som kjører ROS 2 ved å bruke en PX4/ROS2-bro [34]. I Figur 2.9 er det vist hvordan en slik bru fungerer.

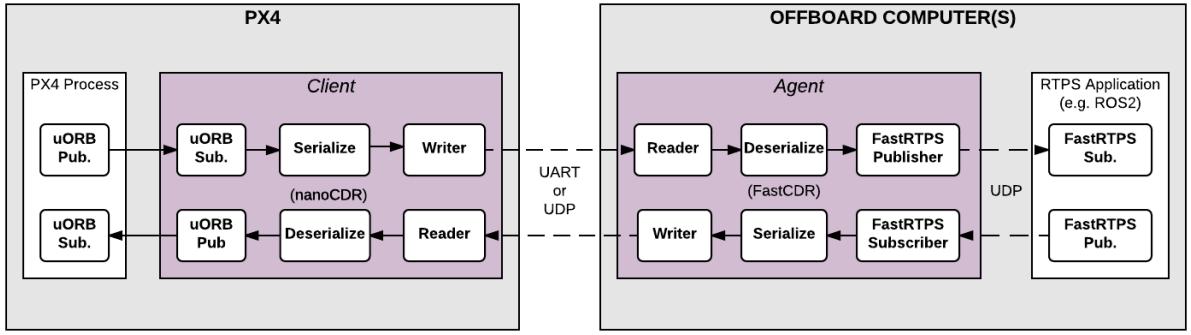
I PX4/ROS2-broen blir en micro versjon av datadistribusjonstjenesten FastRTPS (Fast DDS) brukt for å transportere og oversette meldingene. Denne micro versjonen heter microRTPS. Se Figur 2.8 for å se hvordan PX4/ROS 2-bro fungerer.



Figur 2.8: PX4 og ROS2 kommunikasjon [35].

2.2.5 PX4/ROS2-bro

MicroRTPS(DDS) er en lettvekts datadistribusjonstjeneste. Sammen med microCDR kan microRTPS stå for serialisering og transportering av meldinger, som fungerer som en sømløs overgang mellom ROS 2 og PX4-Autopilot. For at microRTPS skal fungere må den å ha én agent og én klient som kommuniserer seg i mellom. Da det er forskjellige meldingstyper som blir brukt av ROS 2 og PX4-Autopilot må meldingene oversettes. Selve oversettingen er det microCDR står for [36]. Som sett i Figur 2.9 står klienten for oversettingen av uORB emner, mens agenten står for kommunikasjonen med FastRTPS [37] [34]. Sammen står de for en PX4/ROS2-bro.



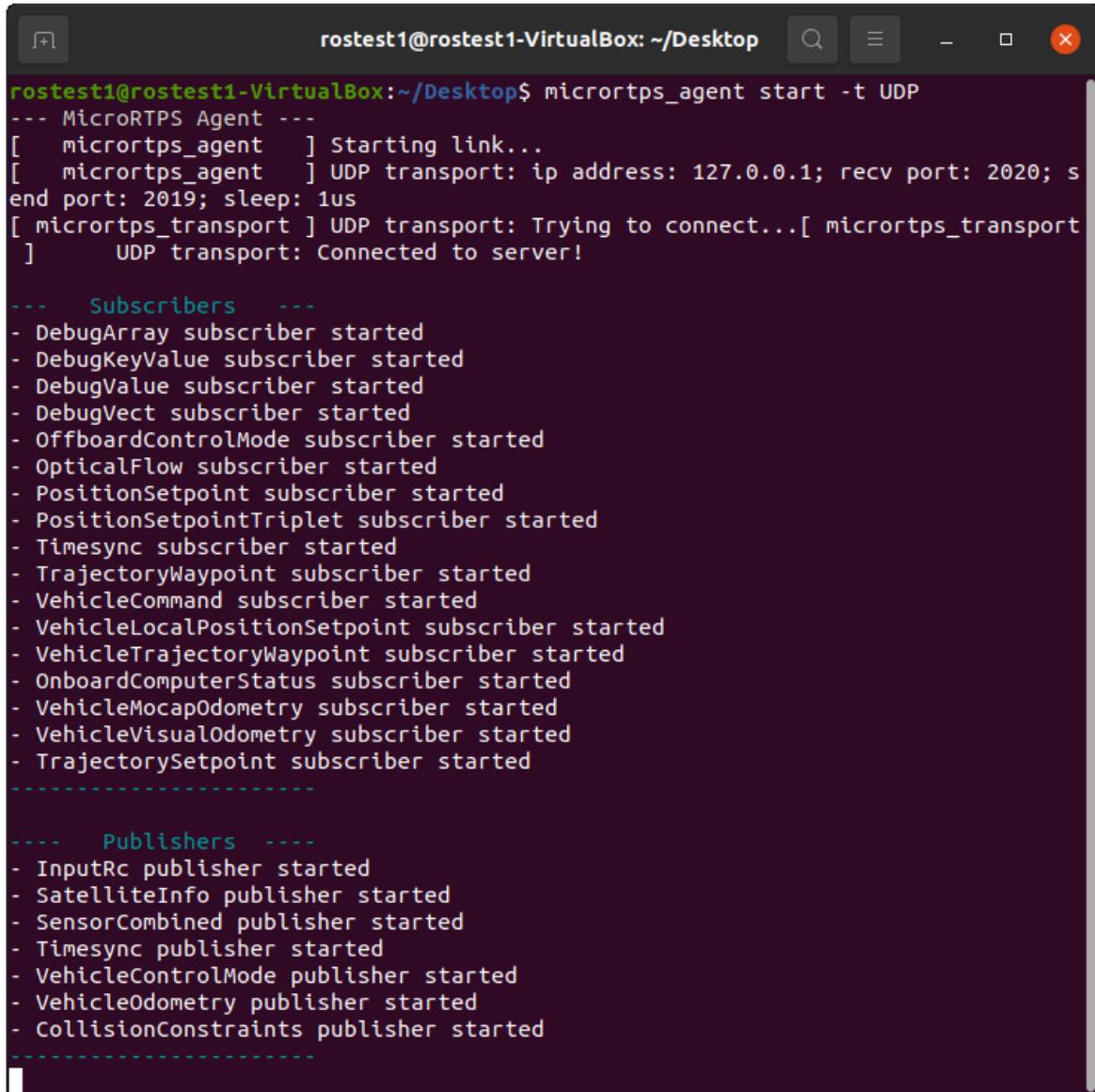
Figur 2.9: PX4/ROS2-bro[38].

Installasjonen av microRTPS klienten på PX4 blir gjort ved å benytte et byggeverktøy til å bygge en ny versjon av PX4-Autopilot [39]. Ved bruk av byggeverktøyet kan flere ulike versjoner av PX4-Autopilot bli bygget. En av versjonene som kan bli bygget er `px4_fmu-v5_rtps`, denne versjonen har microRTPS installert. Byggingen av versjonen kan bli gjort ved å følge instruksjonene på PX4 sine nettsider [40].

For å bruke microRTPS på en ekstern datamaskin brukes de ferdige ROS 2 pakkene `px4_ros_com` og `px4_msgs` [35] [41]. I `px4_msgs` ligger alle meldingstypene som er mulig å bruke i PX4/ROS2-broen. Og i `px4_ros_com` ligger microRTPS agenten. Pakken inneholder også noen eksempel-noder som kan brukes til å verifisere kommunikasjonen. MicroRTPS kan bygges og brukes gjennom ROS 2 på datamaskinen ved å legge pakkene inn i et ROS 2 arbeidsområde.

For å starte microRTPS klienten på PX4, må det enten kjøres en kommando ved oppstart av autopiloten, eller åpne terminalen til autopiloten etter oppstart og skrive inn kommandoen manuelt. Kommandoen som blir kjørt er `micrortps_client start`. Det kan legges flere argumenter til denne kommandoen, blant annet hvilken port klienten skal lese av og skrive til, hvilken kommunikasjonsprotokoll som skal brukes og hvilken baud-rate meldingene skal transporteres på. Før agenten kan startes på datamaskinen må ROS 2 og arbeidsområde som microRTPS pakkene ligger i sources. Deretter kan microRTPS agenten startes ved å skrive kommandoen `micrortps_agent start`. Argumentene som kan legges til er de samme som på klienten.

Om microRTPS er satt til standard vil det startes 24 emner ved oppstart. Noen av emnene inneholder data fra PX4-Autopilot, disse går under Publishers i Figur 2.10. Emnene under Subscribers er opprinnelig tomme, og blir brukt til å sende informasjon og kommandoer til PX4-Autopilot.



```
rostest1@rostest1-VirtualBox: ~/Desktop$ micrortps_agent start -t UDP
--- MicroRTPS Agent ---
[ micrortps_agent ] Starting link...
[ micrortps_agent ] UDP transport: ip address: 127.0.0.1; recv port: 2020; send port: 2019; sleep: 1us
[ micrortps_transport ] UDP transport: Trying to connect...[ micrortps_transport ]
[ micrortps_transport ] UDP transport: Connected to server!

--- Subscribers ---
- DebugArray subscriber started
- DebugKeyValue subscriber started
- DebugValue subscriber started
- DebugVect subscriber started
- OffboardControlMode subscriber started
- OpticalFlow subscriber started
- PositionSetpoint subscriber started
- PositionSetpointTriplet subscriber started
- Timesync subscriber started
- TrajectoryWaypoint subscriber started
- VehicleCommand subscriber started
- VehicleLocalPositionSetpoint subscriber started
- VehicleTrajectoryWaypoint subscriber started
- OnboardComputerStatus subscriber started
- VehicleMocapOdometry subscriber started
- VehicleVisualOdometry subscriber started
- TrajectorySetpoint subscriber started
-----
--- Publishers ---
- InputRc publisher started
- SatelliteInfo publisher started
- SensorCombined publisher started
- Timesync publisher started
- VehicleControlMode publisher started
- VehicleOdometry publisher started
- CollisionConstraints publisher started
```

Figur 2.10: MicroRTPS agent startes med argumentet -t UDP.

MicroRTPS kan bruke kommunikasjonsprotokollene UART og UDP. UART er en asynkron seriell kommunikasjonsprotokoll som ofte brukes til å kommunisere mellom to separate enheter. UDP (User Datagram Protocol) er en nettverks/kommunikasjonsprotokoll for dataoverføring. Kommunikasjonsprotokollen microRTPS bruker blir valgt ved å legge til argumentet `-t` `UART/UDP`. Ved å bruke UART vil microRTPS forsøke å kommunisere over en av portene på enheten programmet kjøres på. Dersom microRTPS blir satt til å bruke UDP vil den kommunisere med UDP-nettverket enheten tilhører.

2.3 Konstruksjon

2.3.1 3D-modellering

3D-modellering er den overordnede definisjonen for alle typer digitale visuelle programmer der det er mulig å kunne representer et digitalt objekt visuelt. Blant ingeniører er det bedre kjent som 3D CAD (Computer Aided-Design), et program opprettet for design og teknisk dokumentasjon for automatiserte prosesser. Det representerer og gir en nøyaktig matematisk oversikt i tre dimensjoner visuelt på en datamaskin[42].

Et godt brukt 3D-CAD program er Solidworks. Solidworks er et av verdens mest brukte CAD systemer spredd over 165 000 bedrifter. Programmet inneholder avanserte funksjoner for design, noe som gjør det mulig å designe komplekse objekter. Det inneholder også verktøy som simuleringer og styrkeberegninger, som gir en representasjon av hvordan et designet objekt vil oppføre seg i en reell situasjon [43].

2.3.2 3D-printing

3D-printing, også kjent som additiv produksjon, er en metode for å produsere tredimensjonale faste objekter designet i et 3D-modelleringsprogram. Det er en av de mest allsidige produksjonsmetodene tilgjengelig, og brukes av både bedrifter og private. 3D-printing ble opprinnelig kun brukt for testing av prototyper i de tidlige stadiene av prosjekter, men er i ferd med å bli en fullverdig produksjonsmetode. Med økende etterspørsel innenfor helse-, bil- og romfart sektorene, er det forventet at 3D-printing vil ha stor økonomisk vekst fremover [44].

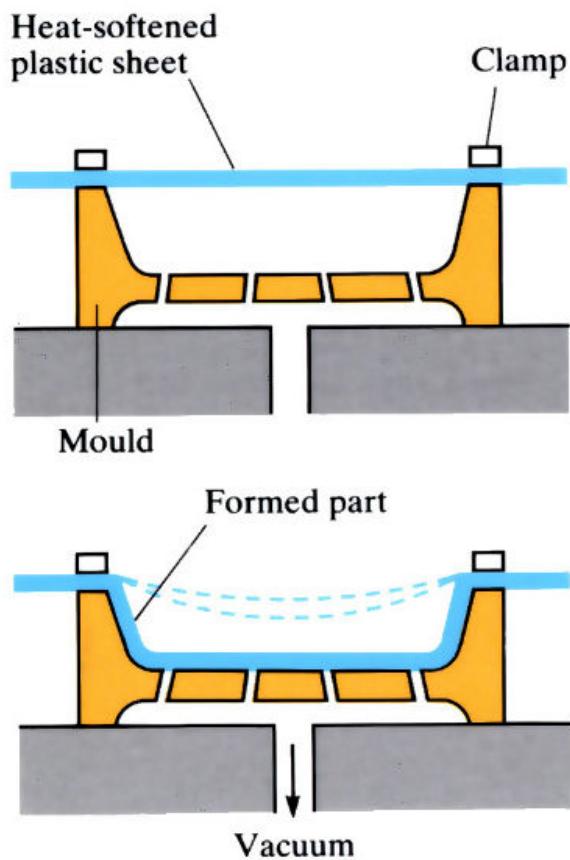
Prosessen for å 3D-printe begynner med at en CAD-fil blir omgjort til Stereolithography (STL). Deretter importeres den til et 3D-printers program. Her er det mulighet for å justere på mange ulike funksjoner for å oppnå en god print. Filen blir så analysert og kuttet opp i flere små horisontale lag. Ut fra dette opprettes en fremgangsmåte på hvordan filen vil bli printet ut lag for lag over i Z-retning. Filen omgjøres til en G-code-fil, og legges over i på Secure Digital (SD)-kort som settes i printeren[44].

2.3.3 Vakuumforming

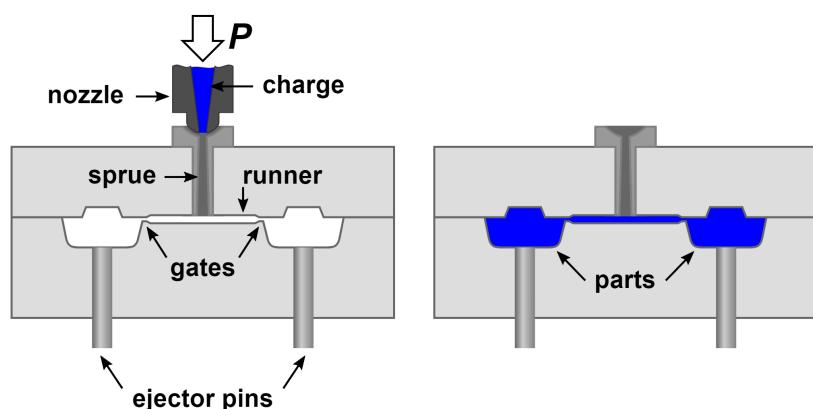
Vakuumforming er en metode for å produsere plastprodukter ved å forvarme plasten og bruke vakuum til å forme den. Metoden impliserer at det må designes en form som plasten formas etter. Det er viktig at denne er designet med slipp i riktig retning. Da det må være mulig å fjerne plastproduktet fra formen etter det har herdet. Om formen som blir brukt til vakuumformingen er av god kvalitet vil det kreves svært lite etterarbeid på et avkjølt plastprodukt. Dette gjør produksjonsmetoden god til masseprodusering [45].

2.3.4 Sprøyttestøping

Sprøyttestøping er en foretrekkende produksjonsmetode når det skal produseres mange identiske deler av plast. Produksjonsprosessen går ut på å injiserer et smeltet materiale i en forhåndslaget form. Formen er et skall av delen man ønsker produsert, og den er vanligvis laget av stål eller aluminium. Det flytende materialet blir deretter kjølt ned og stivner etter formens proporsjoner. Sprøyttestøping kan utføres med en rekke materialer som: metall, glass, termoplast, etc. Termoplast kan 3D-printes. Det har gjort prosessen med å produsere former for produksjon enklere, og mer effektiv. Det legger ikke fra seg betydelig restmateriale, blir prosessen mer ryddig. Prosessen er også gjentagende, ettersom formene kan brukes flere ganger over lengre perioder [45].



Figur 2.11: Eksempel på vakuumforming [46]



Figur 2.12: Eksempel på sprøytestøping [47].

2.4 Regelverk og anbefalinger

Luftfartstilsynet: Droneregler for åpen kategori

Gjennom bestemmelser fra samferdselsdepartementet og luftfartstilsynet, er det utarbeidet en forskrift som omhandler luftfart med droner i åpen kategori, spesifikk kategori og spesifisert kategori. Formålet med disse reglene er å få etablert et regelverk for bruk av droneflyging mellom Den europeiske union (EU) og Det europeiske økonomiske samarbeidsrådet (EØS) [48].

Åpen kategori baserer seg på regler satt opp for bruk av droner som er ansett som minst sannsynlig til å utgjøre risiko for ulykker. Det innebærer at all flyging skal skje på en åpen plass med nok synsrekkevidde opptil 120 meter, som er maksimal distanse mellom drone og pilot. Åpen kategori er delt inn i tre underkategorier; A1, A2 og A3. A3 - "Unna folk" er underkategorien som innebærer de største dronene, og gjelder for droner opptil 25 kg. Innen A3 må piloten ha gjennomført et nettkurs, og det må holdes 150 m avstand til bolig-, nærings-, industri- og rekreasjonsområder. Fra en vekt på 250 g og til 4 kg gjelder A2 - "Nærme folk", for å fly drone som går under A2 må piloten i tillegg til nettkurs ta eksamen på en trafikkstasjon. Om dronen er under 250 g går den under A1 - "Over folk". Dersom en drone under kategori A1 og ikke har kamera eller sensor, vil den sett på som et leketøy og piloten har derfor ikke behov for sertifikat eller nettkurs. Om den har kamera eller sensor må piloten inneha samme kompetanse som A3 [48].

Arbeidstilsynet: Tungt arbeid

Arbeidstilsynet har satt opp anbefalinger angående tungt arbeid i jobbsituasjoner for å forhindre arbeidsskader. Anbefalingene tar utgangspunkt i vanlige arbeidsoppgaver som bæring, løfting, heving og senking. Disse går ut på at tyngre enkeltløft ikke bør overskride en vekt på 25 kg i optimale forhold. Gjenstanden bør ikke heller flyttes mer enn 20 meter. Over en hel arbeidsdag bør det ikke overstige en total løftevekt på 6000 kg for stillestående og gående arbeid, og 3000 kg for sittende arbeid. Det må allikevel tas kontinuerlige vurderinger om belastninger og risiko rettet mot personen som skal utføre oppgaven på grunnlag av personens forutsetninger og helsesituasjon [49].

Kapittel 3

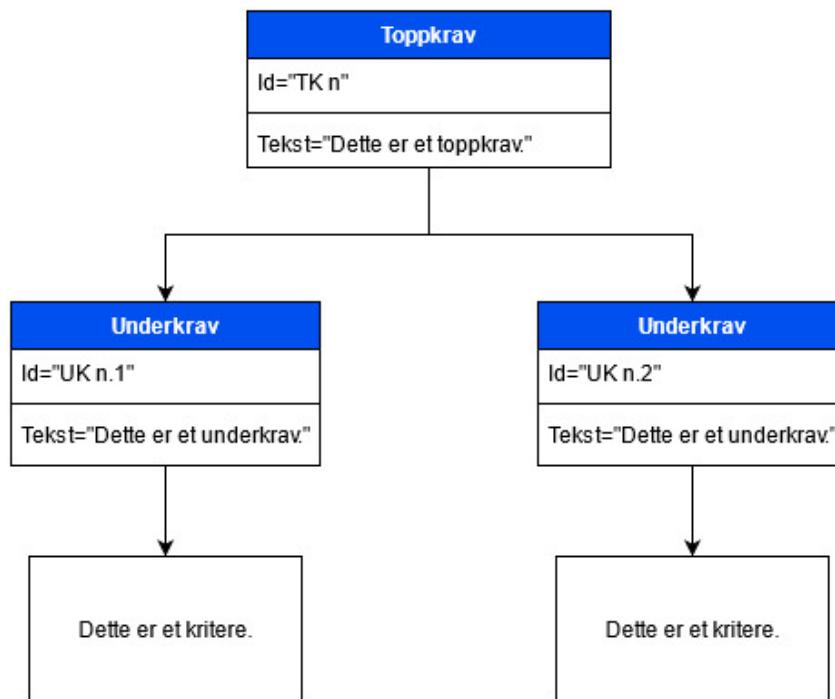
Konsept

Et konsept er en samling av ideer eller en plan som danner grunnlaget for utformingen av et produkt [50]. For prosjektet er det utarbeidet åtte prinsipielt ulike konsepter. Fire for utskytningsenheten og fire for dronen. Disse er vurdert ut fra krav og kriterier relatert til det overordnede prosjektet og dets langsigktig utvikling.

3.1 Krav og kriterier

Et krav er et bestemt ønske, behov eller etterspørsel for et produkt eller system som er spesifisert med målbare kriterier [50]. Et kriterium er et kjennetegn som kan brukes for å avgjøre om en nærmere bestemt betingelse er tilfredsstilt eller oppfylt [50]. De fleste kravene og kriteriene er generelle for det overordnede prosjektet samtidig som noen er spesifikke for det automatiske utskytningssystemet.

For å grafisk beskrive de ulike kravene og kriteriene er det laget UML-skjema. Skjemene er delt opp i toppkrav, underkrav og kriterier. Toppkravene er kategorien som underkravene er en del av. Underkravene er en nærmere beskrivelse av delene som er ønsket å oppnå i kategorien. Kriteriene er testbare mål som godkjennes eller ikke godkjennes. Disse blir brukt til å vurdere om topp- og underkravene er oppfylt eller ikke.



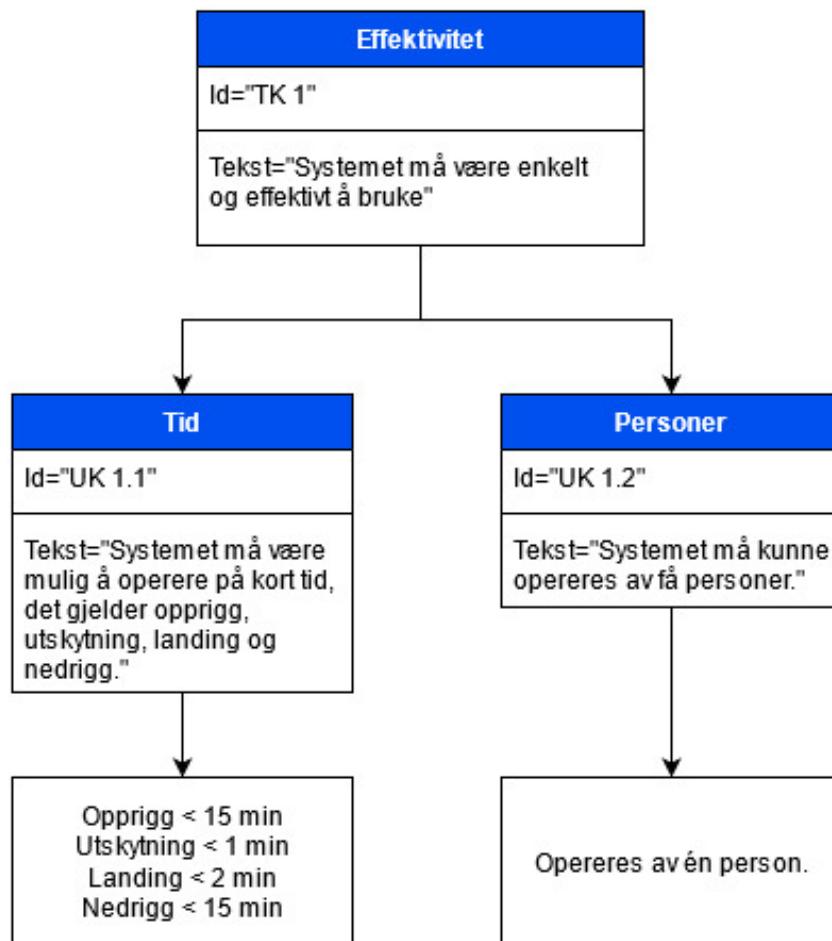
Figur 3.1: Eksempel på oppsett til krav og kriterier.

3.1.1 Effektivitet

Systemet må være enkelt og effektivt å bruke. Som vil si at systemet må være enkelt å operere på kortest mulig. Dette gjelder opprigg, utskytnign, landing og nedrigg.

Opprigg og nedrigg av systemet til utskytningsenhet og droner bør ikke ta mer enn 15 minutter. Opprigging innebærer å sette ut alt med nok distanse mellom hver utskytningsenhet og prosessen med å starte opp alle systemer via bakkestasjonen. For nedrigging vil det være å få hentet inn utskytningsenheter og droner fra testområdet, pakke alt sammen og skru av systemene. Selve utskytningen av dronene må holde seg under 1 minutt. Det betyr at dronene må kunne fly ut av utskytningsenheten hurtig, enten én og én eller samtidig. Landing av dronene må være utført på under 2 minutter. Det er gitt ekstra tid til dette med tanke på behovet for en sikker og ordentlig landing som er nødvendig for å lande trygt.

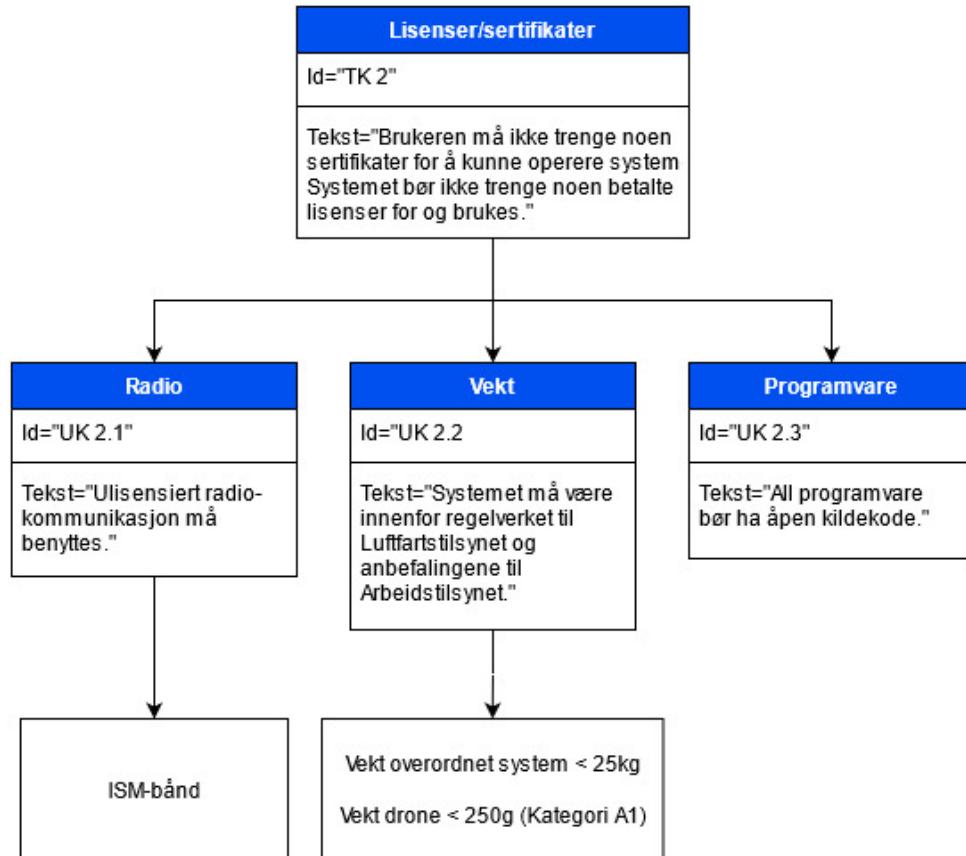
Hele systemet, altså droner, utskytningsenheter og bakkestasjon må være operativt for å kunne brukes alene. Alt nødvendig utstyr må være lett transporterbart, programvarebruksvennlig og enkelt forståelig.



Figur 3.2: UML skjema over krav og kriterier for effektivitet.

3.1.2 Lisenser, sertifikater og lovverk

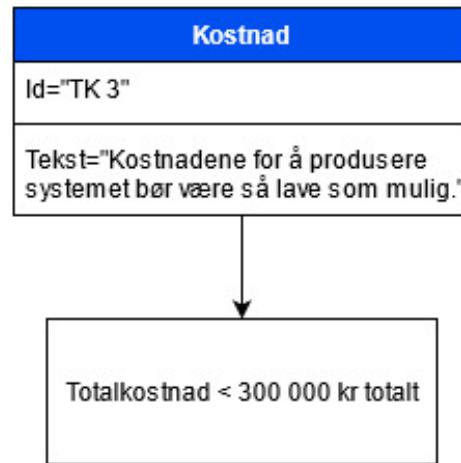
Når det kommer til bruken av hele systemet, må det ikke være nødvendig med noen form for sertifikater. Det innebærer å følge regler og anbefalinger satt opp av Luftfartstilsynet og Arbeidstilsynet nevnt i kapittel 2.4 om flyging av drone og tungt arbeid. Det bør heller ikke kreves betalte lisenser for og brukes. Det må dermed benyttes ISM-bånd-bånd for radio-kommunikasjon samt at all kildekode som er brukt bør være åpen.



Figur 3.3: UML skjema over krav og kriterier for lisenser og sertifikater.

3.1.3 Kostnad

For produksjon av droner og utskytningsenhet må det arbeides for å oppnå minst mulig kostnader. Utskytningsenheten bør ikke overskride makskostnaden på 5000kr, og en drone bør ikke koste mer enn 1500kr.

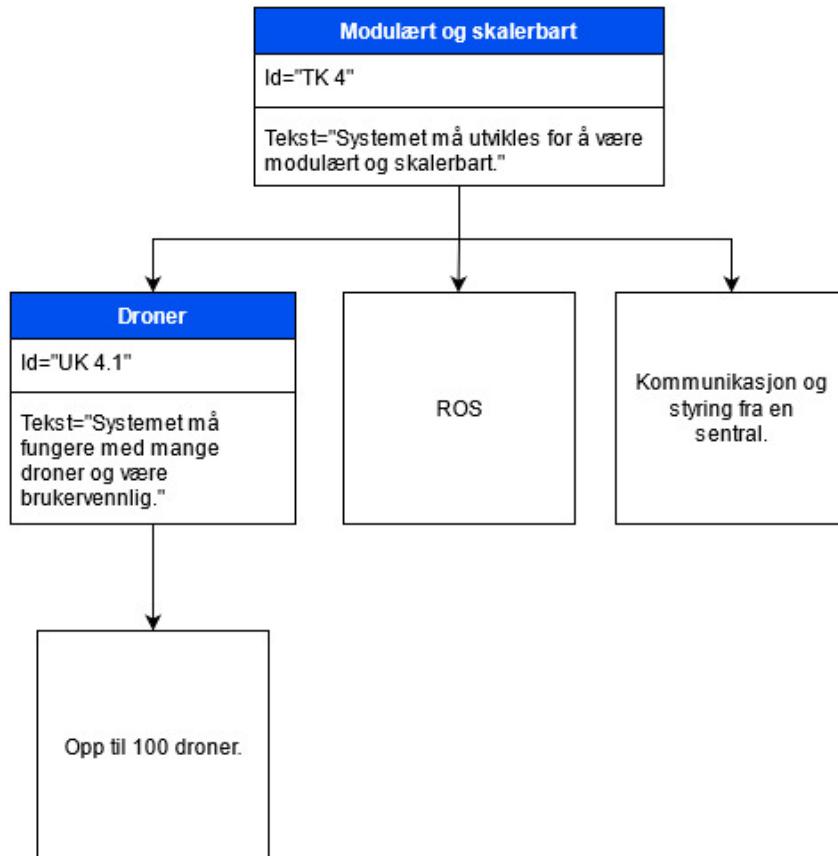


Figur 3.4: UML skjema for krav og kriterier for kostnad.

3.1.4 Modulært og skalerbart

For utviklingen og produksjonen av systemet, er modulær- og skalerbarhet viktig. Hele systemet må være designet slik at det kan enkelt skales opp uten å måtte gå gjennom store designendringer når det er snakk om opptil droner. Systemet må dermed ha muligheten til å fungere på opptil 100 droner, og det må holde seg brukervennlig.

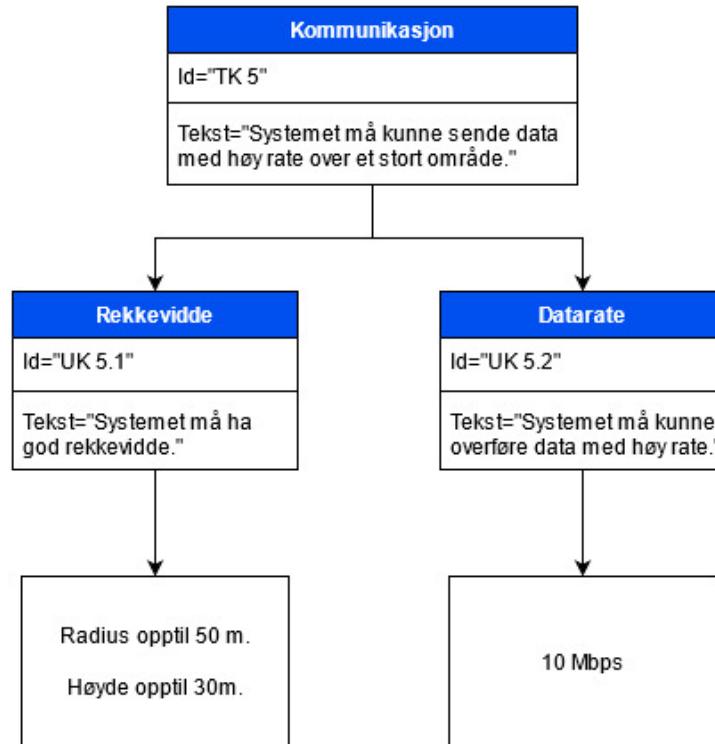
Med krav om modulær- og skalerbarhet er det nødvendig å ha en god programvare til å styre hele systemet. Det er viktig med et system som åpner muligheten for god kommunikasjon på enklest mulig måte. Det er derfor lagt opp til at systemet må bruke ROS som programvare, hvor det skal kunne ble styrt fra en separat bakkestasjon. Designet på systemet bør være gunstig i forhold til det overordnede dronesverm-prosjektet.



Figur 3.5: UML skjema over krav og kriterier for modulært og skalerbarhet.

3.1.5 Kommunikasjon

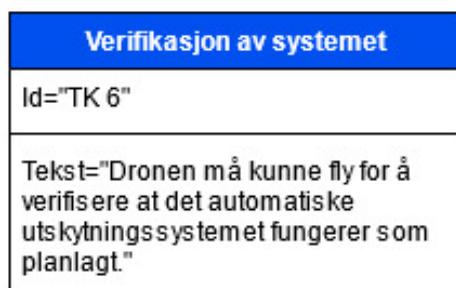
For at systemet skal fungere er det viktig å ha god kommunikasjon mellom alle ledd. Det er ønsket at system skal kunne sende data i sanntid med en datarate på minimum 10 Mbps. 10Mbps er satt da det bør være tilstrekkelig for å direktesende en video i HD og samtidig opprettholde en stabil kommunikasjon mellom enhetene. I tillegg bør kommunikasjonen dekke et større landområde med radius på opptil 50 m, og 30 m i høyden.



Figur 3.6: UML skjema over krav og kriterier for kommunikasjon.

3.1.6 Verifikasjon av systemet

Når et automatisk utskytningssystem for dronesverm skal utvikles, bør systemet verifiseres. Droneene som blir laget må kunne fly slik at samtlige tester kan gjennomføres.



Figur 3.7: UML skjema over krav og kriterier for mulig verifikasierte av systemet.

Komplett UML-skjema av krav og kriterier for konseptene til drone og utskytningsenhet ligger i appendiks B.1.

3.2 Automatisk utskytingssystem

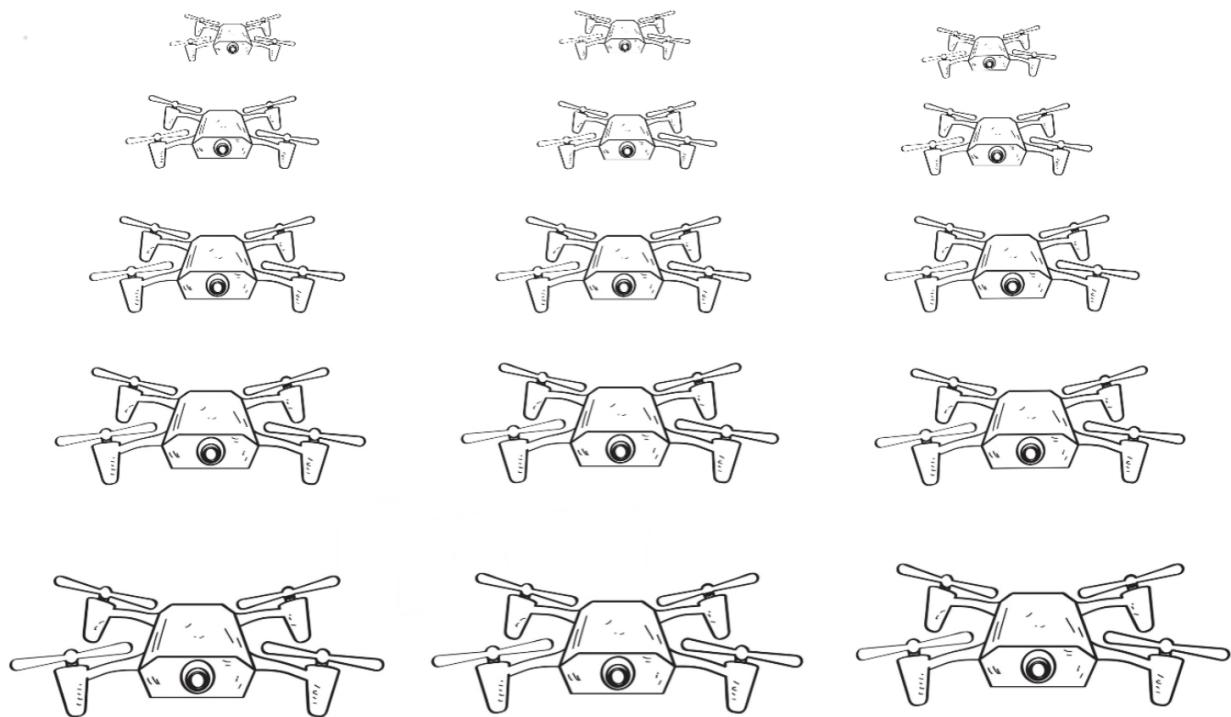
Ideen bak et automatisk utskytingssystem er å komme opp med en effektiv og smart løsning for å få dronene i luften. Det er satt opp fire konsepter som blir presentert under. Disse blir vurdert ut fra kravene og kriteriene i kapittel 3.1. I konseptene er det gått ut fra at det er 100 droner som deployeres.

3.2.1 Manuell utplassering av droner

Dette konseptet går ut på at dronene plasseres ut én og én utover et større område. Fra sin posisjon letter de automatisk. Når dronene skal lande, kuttes strømmen og dronene faller ned.

I dette konseptet må 100 droner plasseres ut én og én. Tidmessig vil det ta omtrent 50 minutter om en person bruker 30 sekunder i gjennomsnitt på å sette ut én drone alene. Utskytning vil gå raskt da dronene kan lette samtidig. Tiden det tar for dronene å lande vil også være minimal. Det vil derimot ta 1 time og 40 minutter for en person å hente alle dronene etter landing om det tar 1 minutt per drone. Grunnen til at det er satt 1 minutt på å hente dronene er fordi når strømmen kuttes vil dronene ha tilfeldige posisjoner, og da mest sannsynlig ligge spredt etter landing. I tillegg er det ikke gitt at alle dronene blir funnet.

Systemet vil være skalarbart, men om dronene plasseres ut med 1 meters avstand horisontalt og vertikalt, vil de totalt dekke 100 m^2 . Det vil si at om ytterlige droner skal plasseres ut vil det kreves et svært stort område. Det vil også være modulært om programvaren er skrevet riktig. Kommunikasjonen må foregå direkte mellom bakkestasjon og drone. Konseptet vil sannsynligvis være det billigste da det ikke trengs å bygge en utskytningsenhet. Til gjengjeld må det brukes mer penger på bakkestasjonen.



Figur 3.8: Manuelt oppsett av droner

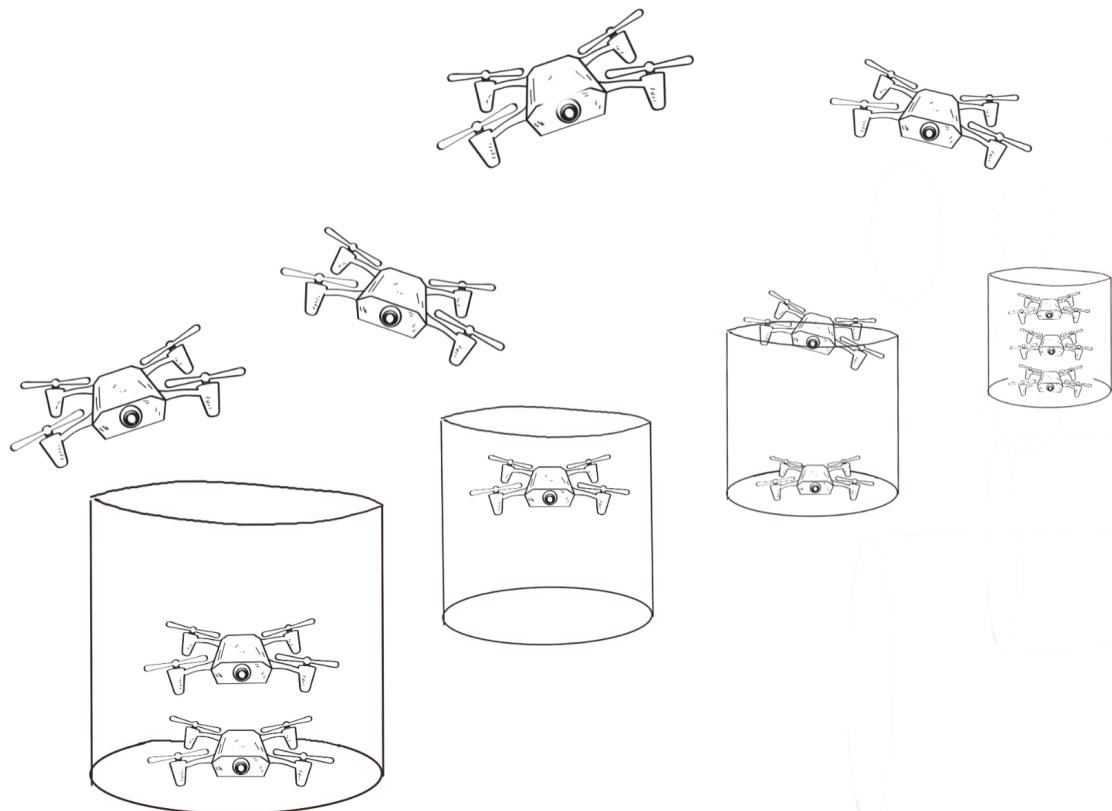
3.2.2 Flere utskytningsenheter

Dette konseptet går ut på å designe og bygge utskytningsenheter som kan oppbevare inntil 10 droner. Dronene skal fly ut automatisk og lande autonomt i sin dedikerte utskytningsenhet.

Løsningen med flere utskytningsenheter gjør at utstyret som skal plasseres ut reduseres betydelig. Det blir regnet med at det vil ta omtrent 1 til 1,5 minutt å sette ut hver utskytningsenhet. Oppriggingen vil ta 10-15 minutter for 100 droner fordi det er 10 utskytningsenheter som skal plasseres ut. For utskyting kan det regnes med et 5 sekunders mellomrom per drone. Det utgjør en periode på 50 sekunder. Landing vil foregå ved at dronen flyr tilbake til utskytningsenheten. Om første drone bruker 30 sekunder på å lande og resten bruker 12-15 sekunder, vil det ta 2,5 til 3 minutter før siste drone lander. Dette overskridet landingskriteriet. Nedrigging vil mest trolig ta 10 til 15 minutter.

Utskytningsenheten vil være skalerbar. Den kan designes til å romme så mange droner som ønskelig. Ved å koble hver utskytningsenhet til en bakkestasjon kan alt styres fra ett sted. Utskytningsenheten bør også kunne gjøre det lettere å oppnå ønsket hastighet og rekkevidde på kommunikasjonen.

Med flere enheter er det ønskelig at utskytningsenhetene kommuniserer med hverandre. Hver enhet må ha et ulisensiert bånd for radio-kommunikasjon og en ettkortsdatamaskin implementert. En slik tilrettelegging resulterer i mer plassbruk, som igjen vil gjøre utskytningsenhetene større. Kostnad og vekt vil også øke, men bør fortsatt være innenfor kriteriene.



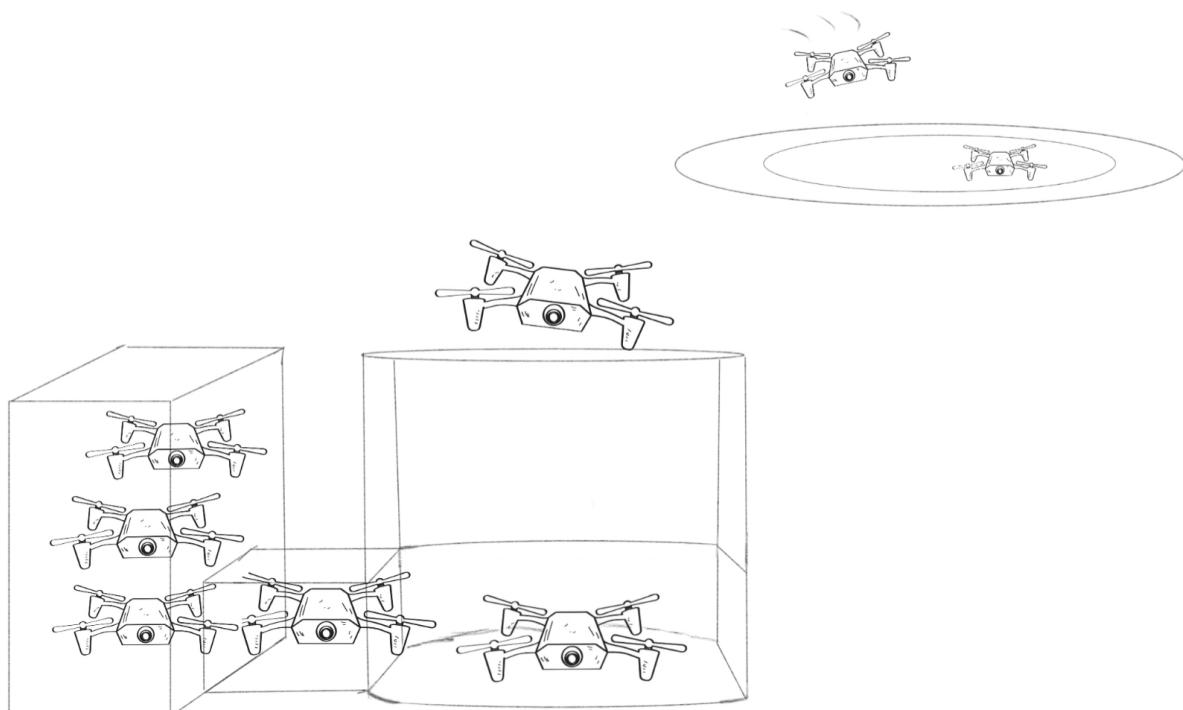
Figur 3.9: Flere utskytningsenheter

3.2.3 Utskytningsenhet med magasin

Dette konseptet går ut på å produsere en utskytningsenhet med magasinløsning og ha et annet dedikert landingsområde/system. Utskytningsenheten vil motta droner fra magasinet hvor de deretter blir pakket sammen.

Opprigging består av å sette ut en utskytningsenhet og laste den med magasiner. Anslått tid på opprigging er 5 minutter. Om 100 droner skal lette kan hvert magasin eksempelvis inneholde 10 droner. Utskytningen vil da foregå ved at brukeren bytter totalt 10 magasin. Dersom en drone bruker 15 sekunder fra magasinet til den er i luften og det tar 20 sekunder å bytte magasin, vil den siste dronen være i luften etter 11 minutter. Det kan benyttes flere utskytningsenheter, men for én person vil det bli en utfordring. Ved landing går konseptet ut på at dronene vil lande på et dedikert landingsområde. Anslått totaltid på landing er rundt 2 minutter. Nedrigging vil foregå i to omganger. Mens dronene er i luften vil utskytningsenhet bli pakket sammen. Etter dronene er landet må de ryddes. Dersom en person bruker 10 sekunder på å få en drone inn i magasinet, vil det ta 100 sekunder per magasin. Dette tilsier en tid på rundt 16 til 17 minutter.

Med tanke på at utskytningsenheterne blir ryddet vakk mens dronene er i luften vil det ikke være et poeng med kommunikasjons-maskinvare i enheten. Kommunikasjon må derfor bakkestasjon stå for. Det vil spare noe vekt og kostnad. Utskytningsenhet, magasinløsningen og landingssystem vil sannsynligvis være skalerbart og modulært.



Figur 3.10: Utskytningsenhet med magasin

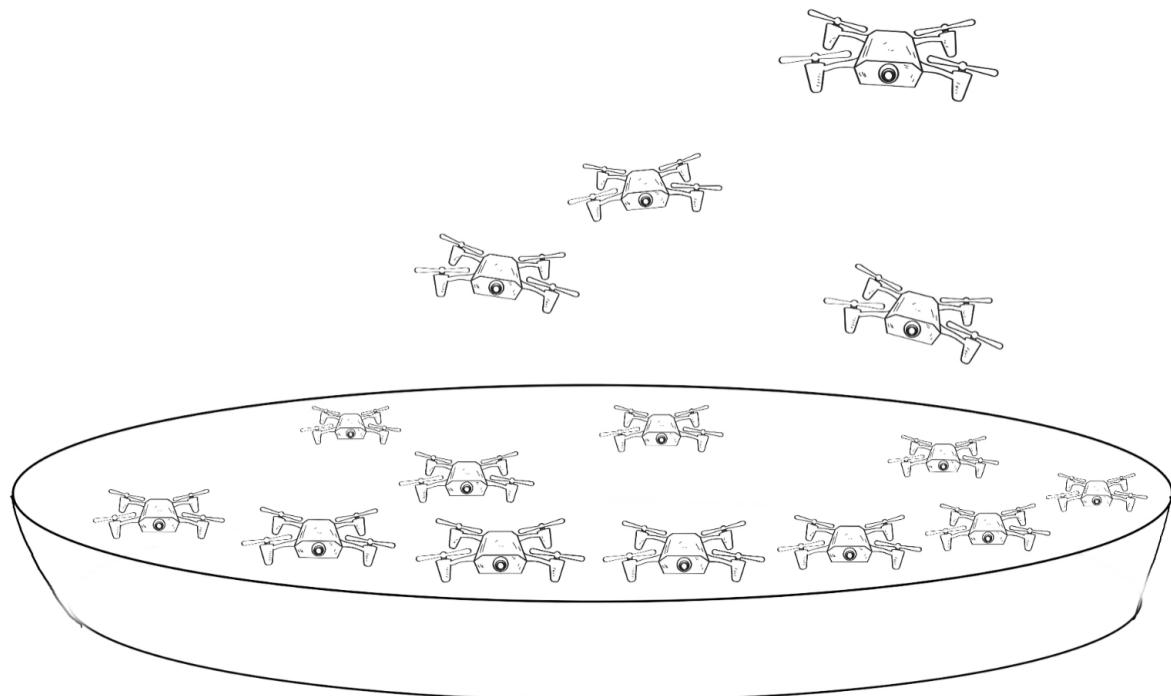
3.2.4 En utskytningsenhet

Dette konseptet går ut på å designe og bygge en utskytningsenhet som rommer 100 eller flere droner. Fra sine dedikerte plasser skal dronene lette og lande.

Det er vanskelig å si hvor lang tid opprigningen vil ta da det avhenger mye av utstyret som må brukes. Utskytningsenheten vil mest sannsynlig bli stor. Derfor virker det lite sannsynlig å komme innenfor kriteriet om opprigg på 15 minutter. Utskytningen vil derimot være svært effektiv, ettersom alle dronene kan lette samtidig fra sin bestemte plass. Å ha dronene i gang etter 5 sekunder og i luften innen 1 minutt bør være oppnåelig. Landingen skal ta omtrent like lang tid som utskytning om man løser faren for kollisjon. Nedrigging vil derimot bli lite effektivt, ettersom det blir omtrent lik prosedyre som ved opprigging.

Innenfor kravene med lisenser, sertifikater og lovverk, er det vekten som blir en usikkerhet. Størrelsen som er nødvendig vil føre til at vektkriteriet ikke er oppnåelig. Å benytte ulisensiert radio-kommunikasjon bør ikke være et problem.

Konseptet kan være modulært og skalerbart om utskytningsenheten blir designet deretter. Det bør være mulig å få implementert nødvendige komponenter slik at høy datarate og stor rekkevidde kan oppnås.



Figur 3.11: En utskytningsenhet

3.3 Drone

Det er utarbeidet konsepter for drone tilhørende utskytningsenheten. Hensikten med dette er å kunne teste systemet og bevise dets virkning. Det er satt opp fire konsepter for dronene som blir vurdert ut fra kravene og kriteriene i kapittel 3.1.

3.3.1 Videreføre utviklingen av drone

Dette konseptet går ut på å videreutvikle drone fra tidligere oppgaver. I oppgavene har det blitt utarbeidet løsninger på skrog, motorer, ESC, sensorer, autopilot og reguleringssystem, slik at hovedoppgaven blir å få systemet til å fungere i praksis.

I forhold til kravet om lisenser og sertifikater vil det være mulig å benytte seg av ulisensierte radiokommunikasjoner. Dronen er i tillegg designet etter kravene fra Luftfartstilsynet slik at den veier under 250 gram.

Det bør være mulig å legge til nødvendige komponenter for å kunne kommunisere med bakkestasjon. Det er derimot uvisst hvor enkelt det er å gjennomføre da systemet allerede nærmer seg komplett. For å få dronen til å fly, kreves det en del undersøkelse og testing av tidligere arbeid. Det kan være tidkrevende og utfordrende å sette seg inn i et allerede nesten komplett system. Det er anslått at dronen fra tidligere oppgaver har en kostnad på rundt 1500 kr.

3.3.2 Utvikle ny drone

Dette konseptet går ut på å utvikle og bygge en ny forbedret drone.

I forhold til kravet om lisenser og sertifikater vil det være mulig å benytte seg av ulisensierte radiokommunikasjoner. Ved å utvikle og bygge en drone fra start vil valg av samtlige komponenter kunne gjøres med tanke på vektkravet fra Luftfartstilsynet. Samtidig kan dronen ha åpen kildekode både når det kommer til komponenter som kjøpes inn og kode som utvikles internt.

Kommunikasjonsmessig er det mulig å bruke komponenter som møter kriteriene for kommunikasjon med én sentral og datarate. Kravet om at dronen må fly bør være oppnåelig da dronen utvikles fra start, men det kan være tidkrevende å utvikle alle systemene fra bunn. Ved å konstruere en ny drone kan dette konseptet betraktes å møte kostnadskriteriet, ettersom komponentene kan velges deretter.

3.3.3 Inkludere autopilot

Dette konseptet går ut på å benytte seg av enkelte løsninger fra tidligere oppgaver, samtidig som det inkluderes en autopilot for enklere kontroll av drone til testing. Denne autopiloten vil erstatte de egenutviklede kretskortene fra tidligere oppgaver da autopiloten har dette innebygget.

I forhold til kravet om lisenser og sertifikater vil det være mulig å benytte seg av ulisensierte radiokommunikasjoner. Ved å videreutvikle tidligere design kan komponentvalgene gjøres med tanke på vekt og åpen kildekode.

Ved å bytte ut og legge til komponenter i dronen fra tidligere iterasjoner, er det mulig å oppnå kommunikasjon med bakkestasjon for samtlige droner. I tillegg kan det legges til rette for høy datarate. Autopiloter er laget for å enklere kontrollere ubemannede systemer; det bør derfor være langt enklere å få dronen til å fly, men det blir en høyere kostnad. Det er anslått at en videreutviklet drone med autopilot vil koste rundt 2500 kr. For det overordnede prosjektet sin del er det senere mulig å bytte ut autopiloten med en ettkortsdatamaskin som kan kontrollere dronen.

3.3.4 Kjøpe drone

Dette konseptet går ut på å kjøpe inn nye droner.

Det er vanskelig å si om innkjøpt drone møter kravene til lisenser og sertifikater, fordi produsenter muligens benytter seg av ulike bånd for radio-kommunikasjon. Det finnes en rekke droner som kan brukes uten noen form for sertifikater. Derimot varierer det om produsenten har åpen kildekode eller ikke.

Det kan bli vanskelig å få en kommersiell drone til å kommunisere med bakkestasjon med høy datarate i en dronesverm. En innkjøpt drone skal kunne fly ut fra utskytningsenheten. Det blir dermed enkelt å teste utskytningssystemet. Å kjøpe inn kommersiell drone kan på en annen side være dyrt i forhold til kriteriet om kostnad. For det overordnede prosjektet er det få muligheter for modifikasjoner i en kjøpt drone.

3.4 Konseptvalg

3.4.1 Rangering

I tillegg til å ha vurdert konseptene individuelt er de sammenlignet i Tabell 3.1 og 3.2. Dette er gjort for å få en organisert oversikt. Konseptene er gitt poeng fra 1 til 10 ut ifra hvordan de oppfyller kravene og kriteriene. De irrelevante kravene og kriteriene for konseptene er markert i grått. Det er også lagt vekt på hvordan konseptene scorer i forhold til hverandre.

Tabell 3.1: Rangering drone

Drone		D1	D2	D3	D4
Effektivitet	Opprigging				
	Utskytning				
	Landing				
	Nedrigging				
	Få personer				
Lisenser/Sertifikater	Radiobånd	10	10	10	1
	Vekt	10	10	8	5
	Åpen kildekode	9	9	10	1
Kostnad	Kostnad	10	10	7	2
Modulært og skalerbart	ROS	10	10	10	1
	Styres fra bakkestasjon				
	100 eller flere droner				
Kommunikasjon	Kommunisere med bakkestasjon	10	10	10	5
	Datarate	10	10	10	5
Fly	Dronen må fly	6	7	9	10
	SUM:	75	76	74	30

Tabell 3.2: Rangering utskytningsenhet

Utskytningsenhet		USE1	USE2	USE3	USE4
Effektivitet	Opprigging	1	8	9	8
	Utskytning	10	8	4	10
	Landing	10	5	8	6
	Nedrigging	1	10	8	9
	Få personer	1	8	7	8
Lisenser/Sertifikater	Radiobånd	10	10	10	10
	Vekt	10	10	10	1
	Åpen kildekode	10	10	10	10
Kostnad	Kostnad	10	6	8	1
Modulært og skalerbart	ROS	10	10	10	10
	Styres fra bakkestasjon	7	10	10	7
	100 eller flere droner	1	8	7	7
Kommunikasjon	Kommunisere med bakkestasjon	10	10	10	10
	Datarate	10	10	10	10
Fly	Dronen må fly				
	SUM:	101	123	121	107

3.4.2 Automatisk utskytingssystem

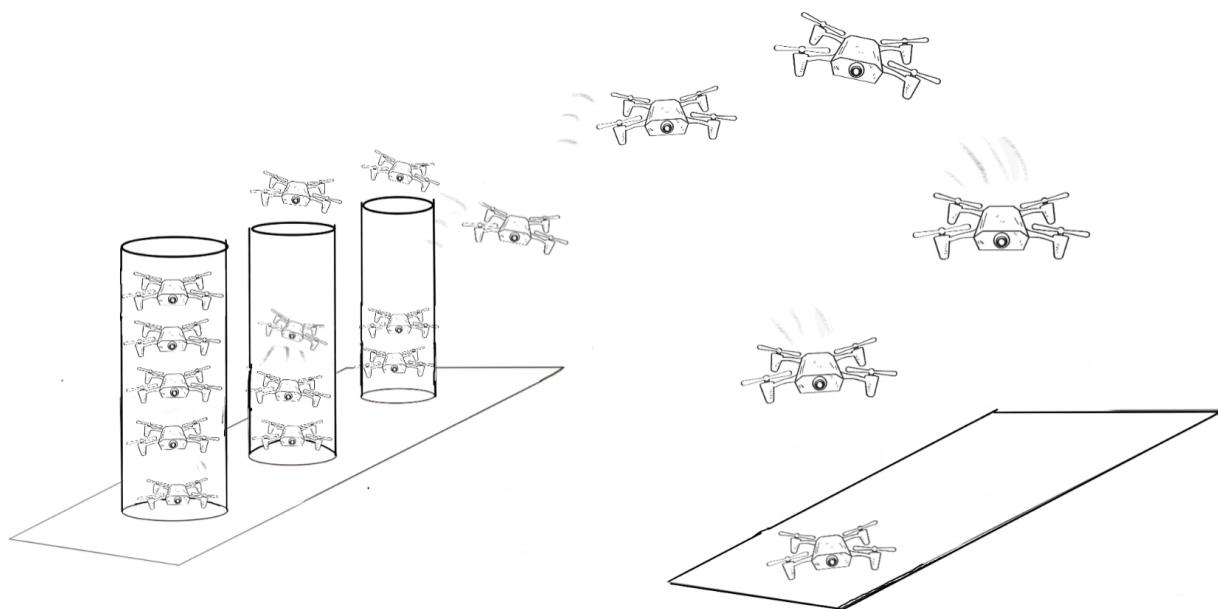
Valget av utskytningsenhet konseptet er tatt på grunnlag av de relevante kravene og kriteriene i kapittel 3.1, og poengsummen som ble gitt i kapittel 3.4.1. Av den grunn er det tydelig at konsept ”Flere utskytningsenheter” og ”Utskytningsenhet med magasin” kommer best ut. Konsept to scorer best på utskyting, mens konsept tre får høyest poengsum på kriteriene som omhandler hvordan dronene skal lande. Med tanke på det overordnede prosjektet er det derfor valgt å kombinere disse konseptene.

Konseptet går da ut på å lage et utskytingssystem maken til ”Flere utskytningsenheter”, og planlegge et landingssystem som i ”Utskytningsenhet med magasin”. Kommunikasjonen er ønsket å foregå som i konsept 2, altså gjennom utskytningsenhetene.

3.4.3 Drone

Ved valg av dronekonsept er det lagt vekt på de relevante kravene og kriteriene i kapittel 3.1, og poengsummen som ble gitt i kapittel 3.4.1. I valget er det også lagt vekt på det overordnede prosjektets fremtid, og arbeidstiden som må legges ned for å få en autonom flygbar drone. Derfor er konseptet ”Inkludere autopilot” valgt som dronekonsept.

Da det er mulig å bytte ut autopiloten med en ettkortsdatamaskin vil det trolig være mulig å få dronen under 1500 kr ved senere iterasjoner av prosjektet. Om autopiloten blir byttet ut ved en senere anledning, bør det være mulig å gjennomføre en liten patch som det automatiske utskytingssystemet kan benytte.



Figur 3.12: Illustrasjon av valgt konsept.

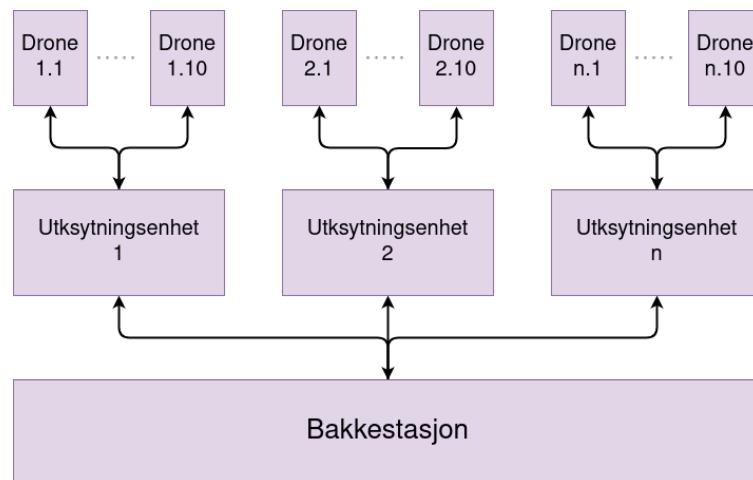
Kapittel 4

Systemarkitektur og -design

I kapittel 3.4 ble konseptene for systemet satt. For å lage et gjennomgående komplett system for automatisk utskyting av dronesverm er det viktig at det utarbeides en plan for hvordan dette kan gjøres på en smart og effektiv måte som både er modulær og skalerbar. I dette kapittelet blir det redegjort hvordan det automatiske utskytingssystemet for dronesverm skal fungere.

4.1 Systemdesign

Det er planlagt at systemet skal være skalerbart og modulært slik at det er mulig å bruke med flere droner. Ifølge kravet om modulerbarhet og skalerbarehet (3.1.4) må systemet fungere med opptil 100, samt være brukervennlig. For å oppnå dette, er det planlagt at brukerens grensesnitt til utskytingssystemet skal være en bakkestasjon. Fra denne bakkestasjonen kan brukeren kommunisere og kontrollere samtlige droner via utskytningsenhetsene, som vist i Figur 4.1.



Figur 4.1: Overordnet systemoversikt. Pilene representerer hvilke enheter som kommuniserer og hvilken vei kommunikasjonen går.

Bakkestasjon

Fra bakkestasjonen vil brukeren kunne kontrollere dronene ved å sende ønsket posisjon eller hastighet fra en terminal eller program. Brukeren vil også kunne få informasjon om dronenes status.

Utskytningsenhet

Utskytningsenheten vil være bindeleddet mellom bakkestasjonen og dronene. Hver utskytningsenhet rommer opp til og med 10 droner og skal ha kontinuerlig kommunikasjon med disse. Kommunikasjonen foregår via en datamaskin som er montert i enheten. Utskytningsenheten mottar informasjon som dronene skal følge fra bakkestasjonen og sender denne videre til de respektive dronene. Denne informasjonen kan inneholde ønsket posisjon og hastighet, eller en forespørsel om informasjon om en spesifikk drone. Fordelen med å ha en datamaskin i utskytningsenheten er å kunne fordele arbeidsmengden utover i systemet. På denne måten trenger ikke bakkestasjonen å kommunisere direkte med 100 droner, men den trenger kun å kommunisere med 10 utskytningsenheter. Dette gjør at bakkestasjonen kan være en enhet med lav prosessorkapasitet, som for eksempel en mobiltelefon, nettbrett, pc eller en annen type kontroller.

Drone

Dronen er det siste og handlende leddet i systemet. Dronen mottar ønsket posisjon eller hastighet fra sin utskytningsenhet. Samtidig som dronen mottar data, vil den kunne sende informasjon tilbake til utskytningsenheten dersom dette blir forespurt.

Kommunikasjon mellom enhetene

Kommunikasjonen mellom enhetene i systemet må være trådløs. For å møte kriteriet om å benytte ROS til programvareutvikling (3.1.4), er det bestemt å benytte ROS i alle enhetene. Dette samsvarer også med kriteriet om at all programvare bør ha åpen kildekode (3.1.2). Siden kommunikasjonen må foregå trådløst og ved bruk av ROS, er det besluttet å sette opp et lokalt Wi-Fi nettverk som kommunikasjonen kan foregå over. Fordelen med dette er at den nyeste versjonen av ROS, ROS 2, fungerer sømløst over Wi-Fi (2.2.3). I tillegg møter dette kriteriet om å benytte -bånd til kommunikasjon (3.1.2).

4.2 Programvaredesign

Som forklart i kapittel 4.1, skal dronene kontrolleres fra bakkestasjonen via utskytningsenheter ved å benytte ROS 2 over et lokalt Wi-Fi. Det må derfor lages en oversikt over hvordan programvaren skal settes opp på de ulike enhetene.

4.2.1 ROS 2

I Figur 4.2 er det vist hvordan programvaren settes opp i ROS 2. For å lage et dynamisk system blir koden til de ulike nodene skrevet identisk like. Dette gjør at nodene i utgangspunktet har samme navn ved kjøretid. Dette løses ved å benytte navnerom, som forklart i teori-kapittel 2.2.3.

Eksempelvis kan det ikke settes opp to utskytningsenheter med samme navn `/use_transfer`. Da vil det fort oppstå problemer. Disse nodene kan derimot startes i navnerom, som kategoriserer nodene og gjør at meldingene fra bakkestasjonen sendes til riktig utskytningsenhet. De to utskytningsenheterne blir for eksempel `/use_01/use_transfer` og `/use_02/use_transfer`.

Videre får de 10 dronene som tilhører `/use_01` navnerommet `/use_01/drone_01`, `/use_01/drone_02` og så videre. Det samme gjelder den andre utskytningsenheten. Dette gjør at når utskytningsenhet `/use_01` skal videresende en ønsket posisjon til sin `/drone_01`, så publiserer den til emnet `/use_01/drone_01/use_drone_setpoint`.

På denne måten kan det legges til svært mange utskytningsenheter ved å kun sette riktig navnerom ved oppstart.

Bakkestasjon

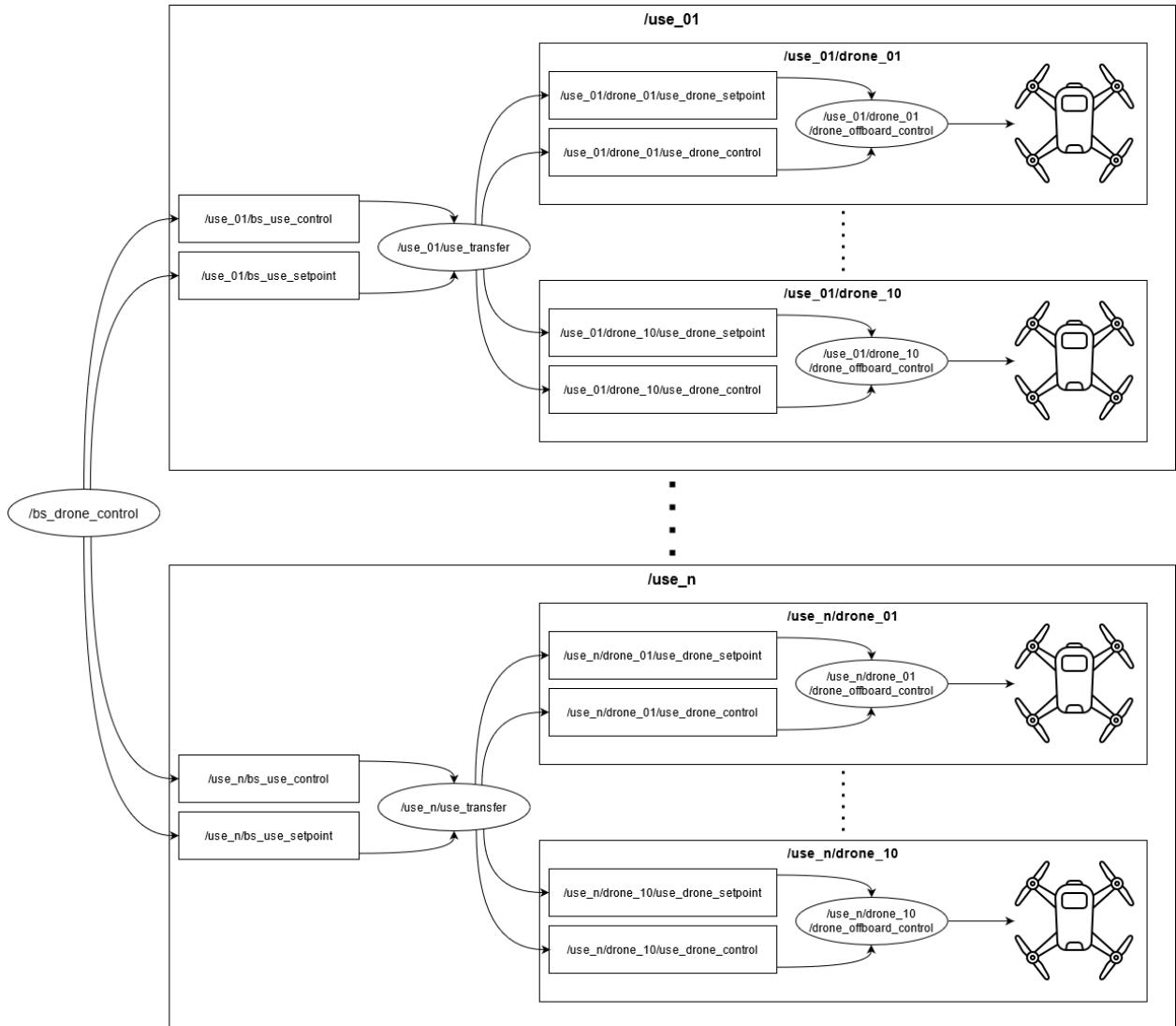
Den første noden `/bs_drone_control`, er noden som kjører i bakkestasjonen. Denne publiserer ønsket posisjon eller hastighet til emnet `/use_n/bs_use_setpoint` og kontroll-informasjon til emnet `/use_n/bs_use_control`.

Utskytningsenhet

I utskytningsenheten kjører noden `/use_n/use_transfer`. Denne noden abонnerer på de to emnene som bakkestasjonen publiserer til, analyserer meldingene og sender dem videre til den korrekte dronen ved å publisere til emnene `/use_n/drone_n/use_drone_setpoint` og `/use_n/drone_n/use_drone_control`.

Drone

I dronen kjører noden `/use_n/drone_n/drone_offboard_control`. Denne noden abонnerer på emnene fra utskytningsenheten dronen tilhører og analyserer meldingene. Dersom meldingen er en ønsket posisjon eller hastighet, blir dette videreført til autopiloten i dronen. Kontroll-meldingen blir også analysert og formidlet videre til dronen eventuelle kommandoer. Denne kontroll-meldingen er egendefinert og tilpasses hva som er nødvendig for å kontrollere dronen.



Figur 4.2: Oversikt over ønsket kommunikasjonsflyt i ROS 2.

4.3 Bakkestasjon

Som nevnt tidligere i kapittelet skal systemet styres fra bakkestasjonen. Dette kan være en hvilken som helst maskin så lenge den kan koble seg til kommunikasjonsnettverket og kjører ROS 2.

4.3.1 ROS 2 node

Noden som settes opp i bakkestasjonen er laget i henhold til slik det er beskrevet i kapittel 2.2.3 og baserer seg på ROS 2-versikten nevnt tidligere.

Importering av selvlagde meldinger

Først importeres de selvlagde meldingene. Disse har foreløpig ikke innhold, da det er uvisst akkurat hvilke variabler dronen trenger for å kunne kontrolleres. Ut i fra kommunikasjonsflyten vist i Figur 4.2, er det tydelig at bakkestasjonen skal kunne sende meldinger til to emner. Derfor forberedes det to meldinger som kan defineres senere når denne informasjonen er kjent.

Definisjon av klasse, "publishers" og "subscribers"

Deretter defineres klassen og forelder-klassen instansieres med parameteren `bs_drone_control` som setter navnet på noden. Videre lages det n "publishers": `bs_use_setpoint` og `bs_use_control`, med navnerom. Siden systemet skal være dynamisk, må det være mulig å legge til et gitt antall utskytningsenheter. Dette gjøres ved å lage en tekst-fil i `/home/bruker/`-mappen som må navngis `"system_setup.txt"` og legge til `"use: n"`, der n er antall utskytningsenheter som er i systemet. I instansierings-funksjonen kalles det en funksjon som analyserer tekstu-filen og henter ut antallet utskytningsenheter. En for-løkke i instansierings-funksjonen til klassen tar seg av å lage det korrekte antallet "publishers" og "subscribers" ut i fra antallet utskytningsenheter definert i `"system_setup.txt"`-filen.

While-løkke

Til slutt i instansierings-funksjonen til klassen kjøres det en funksjon som setter i gang en while-løkke. Denne while-løkken spør brukeren om en kommando og venter til den får svar. Etter brukeren har skrevet inn kommando-en, blir denne analysert og det blir publisert meldinger til emnene etter hva brukeren ønsker. Med dette blir terminalen et kommando-vindu for brukeren. Starten på while-løkken er vist i Kodelisting 4.1.

Kodelisting 4.1: While-løkke som tolker kommando fra brukeren.

```
59     while True:
60         send_control_input = "not enter"
61         send_control_input = input(">>")
62
63         # If input is blank, catch error
64         if (send_control_input == ""):
65             send_control_input = "blank"
66
67         # Check which use and drone to control
68         first_space = send_control_input.find(" ", 0, len(send_control_input))
69         if ( (send_control_input[:first_space].isdigit()) and (first_space != -1) ):
70             self.use = int(send_control_input[:first_space])
71             send_control_input = send_control_input[first_space+1:]
72
73             second_space = send_control_input.find(" ", 0, len(send_control_input))
74             if ( (send_control_input[:second_space].isdigit()) and (second_space != -1) ):
75                 self.drone_nr = int(send_control_input[:second_space])
76                 control_msg.drone = int(send_control_input[:second_space])
77                 setpoint_msg.drone = int(send_control_input[:second_space])
78                 send_control_input = send_control_input[second_space+1:]
```

Den komplette kildekoden ligger i appendiks C.1.1.

Kapittel 5

Integrert ettkortsdatamaskin

Det skal bygges et sammensatt system av droner og utskytningsenheter, samt en bakkestasjon. Dette systemet skal designes av samme personer. Derfor er det valgt å gjøre så mye som mulig felles i dronen, utskytningsenheten og bakkestasjonen. I dette kapitelet skal det sees nærmere på større designvalg som er tatt basert på dette.

Ifølge kapittel 3.1.4 i Krav og kriterier, må kommunikasjonen til hele systemet foregå ved bruk av ROS 2. For å kunne kjøre ROS 2 er det essensielt å bruke et operativsystem som er kompatibelt. Deretter må det sørget for at datamaskinene kan kjøre det valgte operativsystemet. Grunnet kravet om sanntidssystem må det også sørget for operativsystemet kan bygges/patches deretter. Datamaskinene bør også ha en prosessor med flere kjerner, da enkelte kjerner kan settes av til dedikerte arbeidsoppgaver. Dette vil sannsynligvis kreves senere i det overordnede prosjektet.

I tidligere iterasjoner av dronen har Raspberry Pi Zero W blitt brukt sammen med RPi sitt eget operativsystem [3] [4]. Da teknologien rundt "Internet of Things" utvikler seg i et enormt tempo, og det er tre år siden valget av operativsystem og ettkortsdatamaskin ble tatt, vil det bli sett nærmere på om bedre løsninger finnes i 2021. En annen grunn til at det ble valgt å se på et nytt operativsystem er fordi Raspberry Pi OS viste seg å være vanskelig å i de tidligere oppgavene [3] [4].

5.1 Valg av operativsystem

Det er ønsket å bruke samme operativsystem i dronene, utskytningsenhetene og bakkestasjonen. Flere ulike operativsystem ble først vurdert opp mot hverandre. NuttX, FreeRTOS, Raspberry Pi OS, Ubuntu og Arch Linux er noen av operativsystemene som ble vurdert. NuttX og FreeRTOS er sanntidsoperativsystemer, men mindre brukervennlige når det kommer til generell bruk. Raspberry Pi OS er et veldig brukervennlig operativsystem, men fra tidligere oppgaver i prosjektet har det vist seg å være vanskelig sanntids-patche. Ubuntu Server og Arch Linux er ikke sanntids-operativsystemer, men ifølge dokumentasjonen og forumene til operativsystemene bør de kunne sanntids-patches. Operativsystemene er godt støttet i miljøet, og er derfor nokså brukervennlige til tross for tekstbasert brukergrensesnitt [51] [52].

Av de nevnte operativsystemene ble det bestemt å se nærmere på Arch Linux og Ubuntu Server 20.04. Det er hovedsakelig fordi disse er nevnt i dokumentasjonen til ROS 2. Ubuntu og Arch Linux er rangert som henholdsvis "Tier 1" og "Tier 3" plattformer [30].

Da Arch Linux er en "Tier 3" plattform for ROS 2, ble det bestemt å forsøke å installere Arch Linux og ROS 2 på en virtuell datamaskin i Oracle VM VirtualBox først. Ved installasjon av Arch Linux ble installasjonsguiden på hjemmesiden deres fulgt [53]. Dette er en noe avansert prosess da brukeren blant annet må partisjonere og formatere selv via det tekstlige brukergrensesnittet. Det kreves også manuell installasjon av bootloader (gnome), pakkebehandlere (pacman) og superbruker (sudo). Da operativsystemet var satt opp, ble det forsøkt å laste ned ROS 2. Siden

Arch Linux er ”Tier 3” plattform har ikke ROS 2 laget binære pakker til operativsystemet [30]. Derfor måtte programmet bygges fra kildekoden ved hjelp av pacman. Dette ble gjort ved å følge guiden i Arch Linux dokumentasjonen [54].

For å teste om ROS 2 fungerte optimalt på Arch Linux, ble demo noden ”listener” kjørt på en annen virtuell datamaskin samtidig som Arch Linux kjørte demo noden ”talker”. De to virtuelle datamaskinene kommuniserte ved hjelp av NAT. Kommunikasjonen ble testet ved å kjøre kommandoen `ifconfig` for å lese IP-adressene, for deretter å pinge hverandre. Pingingen ble gjort ved å skrive kommandoen `ping <ip.adresse>`. Selv om de to virtuelle datamaskinene hadde kommunikasjon, fungerte ikke ROS 2 som det skulle. Arch Linux snakket korrekt, altså ”Publishing: [Hello World: 1, 2, 3 osv...]” ble skrevet, men den andre virtuelle datamaskinen mottok ikke meldingene. Det ble forsøkt å gjenta denne prosessen to ganger, samt feilsøke både operativsystemet, ROS 2, og benytte andre test-program, men det var til ingen nytte. De virtuelle datamaskinene fikk ikke kommunisert med DDS/FastRTPS/ROS 2.

Prosesssen over ble også testet med Ubuntu Server. Ubuntu Server er et mer brukervennlig program å installere enn Arch Linux. Nedlasting av programvare og installasjon på VM blir gjort ved å følge brukerveiledningen på Ubuntu sine hjemmesider [55]. Installasjonsprosessen foregår mer som en vanlig Windows/macOS/Ubuntu Desktop installasjon. ROS 2 blir lastet ned ved å benytte Debian pakker [56]. For å teste om ROS 2 installasjonen fungerte, ble demo nodene ”talker” og ”listener” brukt. Dette fungerte som det skulle, den ene virtuelle PCen skrev ”Publishing: [Hello World: 1, 2, 3, ...]”, mens den andre skrev ”I heard: [Hello World: 1, 2, 3, ...]”.

Grunnet problemene med ROS 2 på Arch Linux, ble det derfor bestemt å bruke Ubuntu 20.04. Det skal være mulig å laste ned ROS 2 på Arch Linux som fungerer feilfritt, men på grunn av manglende kompetanse, og at det er av mindre betydning for oppgaven, ble det valgt å gå videre uten å komme til bunns i problemet.

5.2 Valg av datamaskin

Grunnet ulike krav og kriterier for dronene, utskytningsenhetene og bakkestasjonen sees det nærmere på de ulike enhetene hver for seg.

5.2.1 Drone

Ved valg av ettkortsdatamaskin ble det lagt mest vekt på størrelse, vekt, operativsystem-støtte og prosessor. Det er på grunn av kravene om vekt, ROS 2 og sanntidssystem.

Banana Pi Zero M.2 (BPI) virket lenge som det åpenbare valget da den kun veier 16 gram, har en størrelse på 65x30mm, og har fire kjerner. Spesifikasjonene er tilnærmet identiske til Raspberry Pi Zero W som er brukt i tidligere oppgaver, men BPI har en raskere prosessor med flere kjerner. Det er viktig med tanke på å kunne kjøre i sanntid. I følge Banana Pi skal BPI-M2 Zero kunne kjøre operativsystem basert på Linux [57], som betyr at ROS 2 kan installeres.

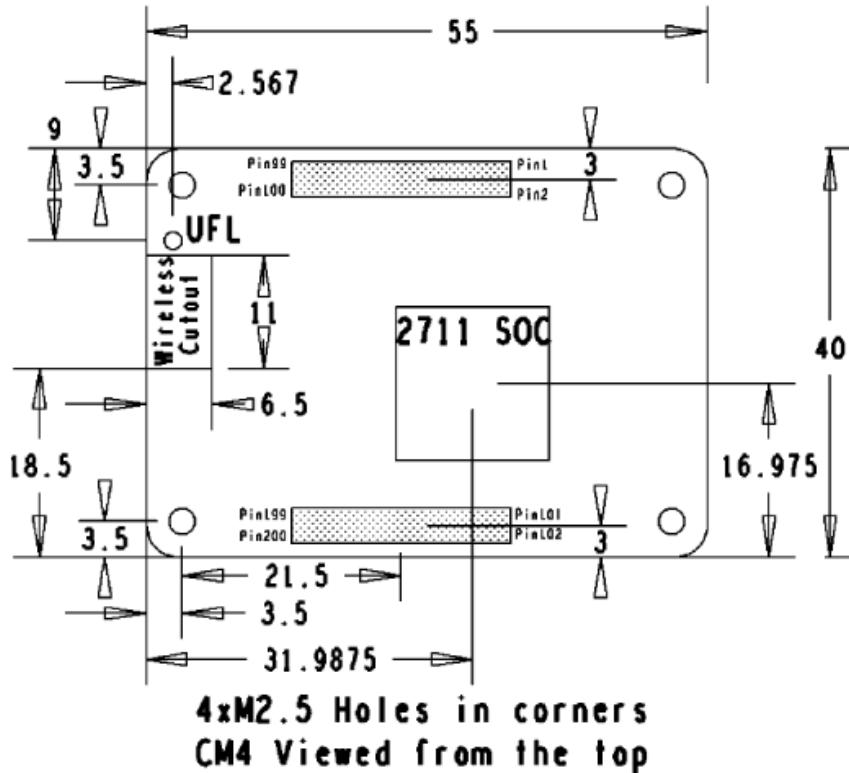
Da UiA hadde en Banana Pi Zero M.2 liggende, ble det forsøkt å installere Armbian på den ved hjelp av Armbian build. Deretter ble ROS 2 lastet ned. Kommunikasjonen til ROS 2 fungerte ved testing, demo nodene "talker" og "listener" ble kjørt, og printet ut tekst i terminalene. "Talker" ble kjørt på en bærbart datamaskin med Ubuntu Desktop, og BPI kjørte "listener". Da rqt ble sjekket, viste det seg at det lå et lite problem i programmet. "Talker" noden ble vist i grafen, men "listener" noden og emnet "chatter" var ikke synlig. Det ble derfor forsøkt å kjøre BPI som "talker" og den bærbare datamaskinen som "listener". BPIen snakket, men den bærbare datamaskinen fikk ingen melding. RQT viste heller ingenting. Etter feilsøking viste det seg at feilkilden mest sannsynlig var prosessoren, en 32-bit ARM Cortex-A7, noe ROS 2 ikke er kompatibelt med. Siden tidligere iterasjoner av dronen ble kjørt med ROS, Raspberry Pi OS og RPi Zero W (som er 32-bit) ble det regnet med at ROS 2 også ville være kompatibelt, men til dags dato er ROS 2 kun kompatibelt med 64-bit system [30].



Figur 5.1: Ønsket visning i RQT.

Ettersom ROS 2 kun er kompatibel med 64-bit prosessorer, kan hverken Banana Pi Zero M.2 eller Raspberry Pi Zero W brukes. Det finnes heller ingen ettkortsdatamaskiner som ligner disse med 64-bit prosessorene. For å løse dette, ble det valgt å finne et utvalg av potensielle ettkortsdatamaskiner og sammenligne disse.

Ut ifra sammenligningen kom det frem at kort som bærer 64-bit prosessorer generelt har større formfaktor enn kort med 32-bit prosessorer, derfor ble det også kikket på COM-typer. Grunnet datamaskinen skal brukes til testing og prototyping ble god dokumentasjon og stor kundebase vektlagt høyt, da det kan være en fordel for å unngå problemer. Ytelse ble også vektlagt, men ut i fra ulike leverandører og produsenter viste det seg at ytelse sannsynligvis ikke ville bli et problem. Det ble derfor bestemt å bruke Raspberry Pi Compute Module 4. RPi CM4 er en robust COM som er godt dokumentert og er hyppig nevnt i ulike forum. Den kommer i flere varianter, og varianten med best ytelse har blant annet innebygd Wi-Fi, 32 GB embedded Multi-Media Controller (eMMC) lagring og 8 GB random access memory (RAM). Og alle variantene benytter en Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1,5 GHz prosessor og har yttermål på 55 mm x 40 mm [58].



Figur 5.2: Mekanisk tegning av Raspberry Pi Compute Module 4 [58].

5.2.2 Utskytningsenhet

Valget av datamaskin for utskytningsenheten ble gjort med flere av de samme hensynene som for dronen. Ettersom Ubuntu 20.04 allerede var valgt som operativsystem, og kravene om ROS 2 og sanntidssystem er de samme. Ble det valgt å bruke samme datamaskin for utskytningsenheten som dronen, altså Raspberry Pi Compute Module 4.

Å bruke den samme datamaskinen i både utskytningsenheten og dronen har flere fordeler. Blant annet må det designes og produseres bærekort for å kunne bruke disse. Da kan det benyttes samme utgangspunkt i designprosessen. En annen fordel er at potensielle utfordringer som dukker opp kan bli de samme for både dronen og utskytningsenheten, slik at det er mulig å spare tid på problemløsing av samme system. RPi CM4 skal ha mer enn nok datakraft til å kontrollere sine 10 droner med fire kjerner på 1,5 GHz, 4 GB RAM og 32 GB eMMC.

5.3 Bærekort til ettkortsdatamaskin

Da RPi CM4 er en COM, må det designes og produseres et bærekort. Et bærekort til en COM-datamaskin består av det som vanligvis ville vært integrert i en vanlig ettkortsdatamaskin. Det vil si at alle bærekort må ha konnektorer som kobler bærekortet sammen med pinnene på datamaskinen, og forsyne datamaskinen med strøm. Om disse kravene er på plass kan bærekortet konfigureres for ønskede formål.

5.3.1 Funksjonaliteter

På grunn av at dronen og utskytningsenheten har ulike funksjonaliteter, blir disse redegjort hver for seg nedenfor.

Drone

Hovedfunksjonen til bærekortet i dronen er å kommunisere med utskytningsenheten og autopilot. For å kommunisere med utskytningsenheten er det bestemt å bruke Wi-Fi, noe Raspberry Pi Compute Module 4 har innebygd. For å kommunisere med autopiloten er det bestemt å bruke UART. Derfor ble Rx0 og Tx0 på pin 14 og 15 benyttet til dette. Ellers måtte kortet kun inneholde det grunnleggende for å bli satt opp, oppdatert, styrt og kjørt. Det eneste kravet er da strømforsyning. RPi CM4 kan programmeres ved bruk av et I/O-brett som kjøpes ved siden av før den plasseres i dronen. Senere kan den styres med SSH over Wi-Fi når den er plassert i dronen. Det er også valgt å legge inn en micro-USB i tilfelle I/O-brettet ikke er tilgjengelig.

I de tidlige iterasjonene av dronen har et 7,4 V 48 A LiPo-batteri forsynt dronen med strøm. Derfor vil et lignende batteri sannsynligvis bli tatt i bruk, og ble derfor brukt som utgangspunkt. Grunnet usikkerhet rundt flytid i de tidlige iterasjonene av dronen, er det ønsket å maksimere flytiden. Ved å bruke en fysisk bryter til å skru på RPi CM4, for så å bruke RPi CM4 til å styre strømmen til resten av dronen kan batteriet spares, samt at det kan benyttes som en kill-switch.

Utskytningsenhet

For at Raspberry Pi Compute Module 4 skal fungere som planlagt i utskytningsenheten må den ha enkelte funksjonaliteter. Det er bestemt at kommunikasjonen i systemet skal foregå over Wi-Fi. RPi CM4 har innebygd Wi-Fi-modul, så dette behøves ikke på bærekortet. Systemet må være mulig å flytte rundt uten å være koblet til strømnettet. Det må derfor være mulig å forsyne datamaskinen med strøm fra et LiPo-batteri tilsvarende det som er brukt i dronen. I tillegg må det være enkelt å skru datamaskinen av og på. Det er derfor valgt å benytte en fysisk bryter som er lett tilgjengelig. Til slutt bør datamaskinen kunne kobles til og kontrolleres fra en ekstern datamaskin.

Den siste funksjonaliteten er ikke nødvendig, men det er fint å kunne koble til en ekstern datamaskin. Det er mulig å fjernstyre RPi CM4 over Wi-Fi, men det er ikke alltid det er et trådløst nettverk som RPi CM4 er konfigurert til å kobles til. Fordelen med å kunne fjernstyre RPi CM4, er at det ikke er nødvendig å montere ettkortsdatamaskinen på et I/O-brett.

5.3.2 Kretsdesign og komponentvalg

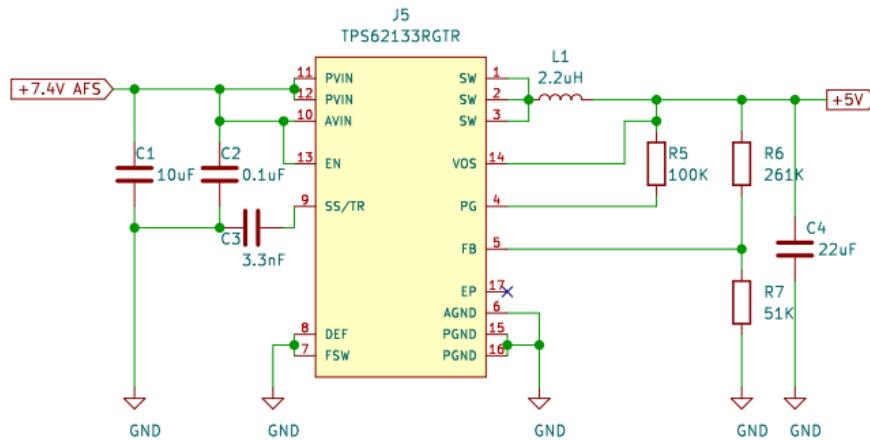
Drone

Ved kretsdesign av bærekort til Raspberry Pi Compute Module 4 som skal ligge i dronen er det jobbet ut ifra funksjonalitetene til dronen i kapittel 5.3.1.

Det første som ble gjort ved designning av kretsene til bærekortet, var å legge inn to stk Hirose DF40C-100DS-0.4 V konnektorer. Disse konnektorene kobler bærekortet til RPi CM4 sine utgangspinner. Det ble valgt å bruke en mal da disse ble lagt inn for å sørge for at pinnene var riktig nummerert. Det ville vært katastrofalt å legge en kabel på feil pin.

Da Hirose-konnektorene var på plass, ble det sørget for at Raspberry Pi Compute Module 4 fikk en tilstrekkelig og nøyaktig strømforsyningskilde. Batteriet som blir brukt i dronen vil være et 7,4 V 48 A LiPo-batteri. RPi CM4 skal ha en 5 V inngangspannning, som vil si at det må lages en spenningsregulator-krets. Det tas også i betraktnign at batteriet totalt kan gi ut 48 A. Om en slik strøm skulle kommet inn i Raspberry Pi Compute Module 4, ville den mest sannsynlig blitt ødelagt [58]. Derfor bør spenningsregulator-kretsen designes slik at den forhindrer at dette skjer.

I spenningsregulator-kretsen er det valgt å ta i bruk en ulineær step-down (buck) DC-DC spenningsregulator fra Texas Instruments som heter TPS62130 [59]. Den kan ha en variabel innspenning mellom 3 V og 17 V og samtidig gi ut en stabil spenning på 5 V som RPi CM4 krever. Dette er essensielt da LiPo-batteriets spenning vil variere mellom rundt 6 V og 8,4 V når den er utladet til fullladet. Den vil også falle i spenning under kraftig bruk, noe den vil oppleve når motorene i dronen har maksimalt pådrag. I tillegg kan spenningsregulatoren kun gi ut 3 A, som er viktig med tanke på å ikke ødelegge Raspberry Pi Compute Module 4.



Figur 5.3: Kretsskjema av spenningsregulator-krets.

Kretsen på Figur 5.3 er designet i henhold til databladet til TPS62130. Utspenningen blir konstant 5V om innspenningen er på 5-17 V. Dette er fordi forholdet mellom R6 og R7 er rundt 5,25. Se formel 5.1.

$$\frac{R6}{R7} = \frac{V_{out}}{0.8} - 1 = \frac{5V}{0.8} - 1 = \underline{\underline{5.11}} \quad (5.1)$$

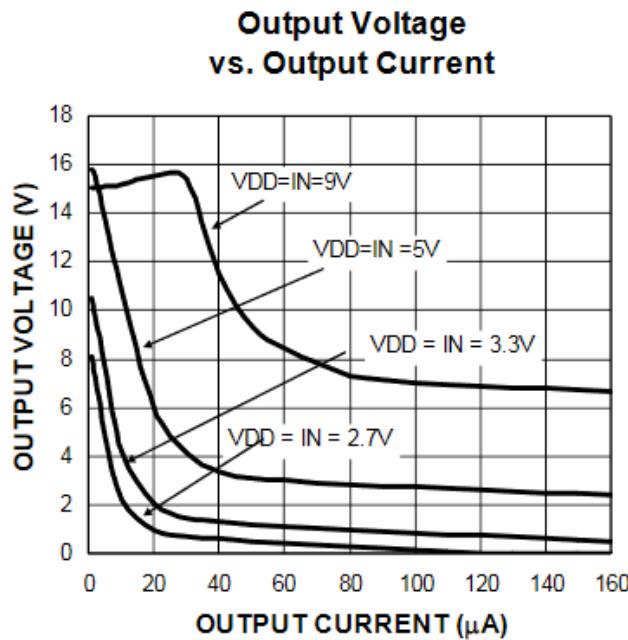
Grunnen til at det er valgt å bruke $51\text{k}\Omega$ og $261\text{k}\Omega$ er fordi dokumentasjonen til TPS62130 anbefaler at R7 ikke bør overstige $400\text{k}\Omega$, og at det bør gå minst $2\mu\text{A}$ strøm gjennom resistorene [59]. Da det er 5 V over motstandene, vil verdiene som er valgt møte kravene til databladet. For bevis se formel 5.2 og 5.3.

$$I = \frac{V_{out}}{R6} = \frac{5V}{261\text{k}\Omega} = \underline{\underline{20\mu\text{A}}} \quad (5.2)$$

$$I = \frac{V_{out}}{R7} = \frac{5V}{51k\Omega} = \underline{\underline{98\mu A}}$$
(5.3)

Videre er resten av komponentene i spenningsregulator-kretsen valgt ut fra anbefalinger i databladet [59].

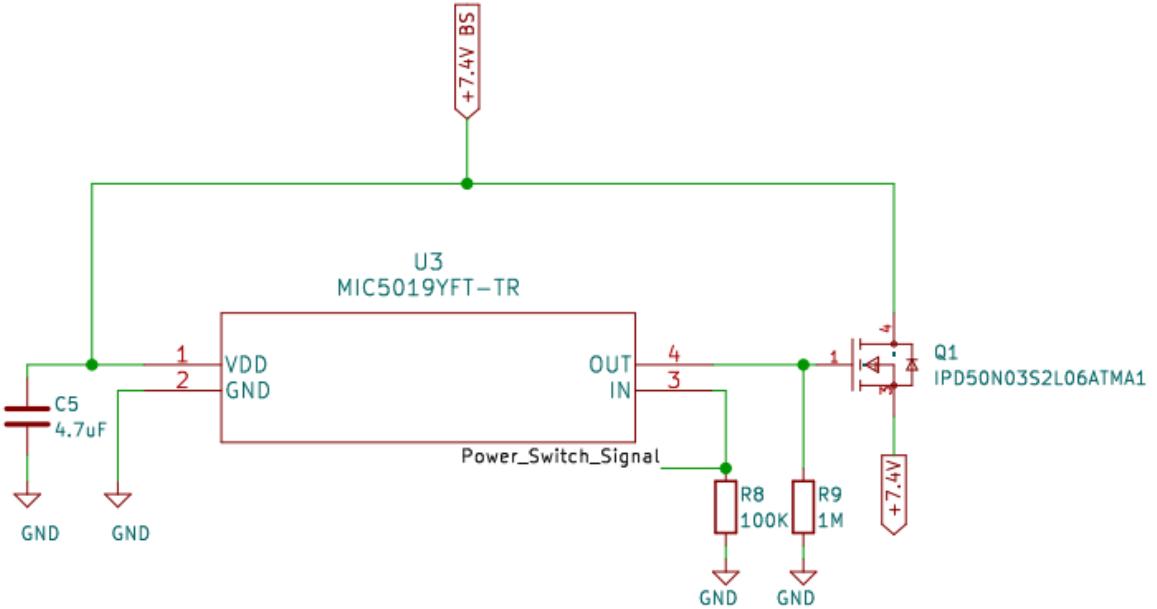
Som nevnt tidligere er det valgt å bruke Raspberry Pi Compute Module 4 til å skru av og på resten av systemet i dronen. For å gjøre dette ble det designet en N-channel høy-side Metal Oxide Semiconductor Field Effect Transistor (MOSFET) bryterkrets. Ved å velge en riktig MOSFET driver kan et signal fra ettkortsdatamaskinen skru den av og på, og da ha samme brukermåte som en fysisk bryter. Siden markedet er fullt av N-channel MOSFETer, ble det bestemt å finne en MOSFET driver som passet formålet før først. Driveren som ble valgt er en MIC5019 fra Microship. Denne driveren kan både kontrollere signal i både høy- og lav-side applikasjoner, og har de nødvendige funksjonene som trengs. Den kan blant annet ha en innspenning på +2,7 V til +9 V og styre kretser som går helt fra 0V til nærmere +19 V. Grunnen til at MOSFET driveren kan styre kretser opp til 19 V, er fordi den kan øke spenningen slik at gate driveren på MOSFETen kan være opp til 19V. Det vil si at driveren kan ha lik V_{in} som selve MOSFETen, og likevel kunne styre den. Videre blir MOSFET driveren styrt av et høyt eller lavt logisk signal. De trekker mindre en $1\mu A$ når den er skrudd av, og trekker rundt $150\mu A$ når den er påskrudd. Det er lite med tanke på batteritid [60].



Figur 5.4: Graf som sammenligner utspenning mot strømmen som går ut.

Grunnen til at det tidlig ble valgt å designe en høy-side bryterkrets er fordi det er en nokså stor krets hvor en del av komponentene må være koblet til jord. Spenningsfallet som kan oppstå ved en høy-side bryterkrets er ikke farlig, da motorene som har blitt brukt i tidligere iterasjoner av dronen skal kunne kjøre med noe variabel spenning. I tillegg vil det mest sannsynlig bli brukt en strømmodul som er plassert mellom bryterkretsen og motorene/autopiloten.

Det er valgt å bruke en N-channel MOSFET da disse generelt er bedre rustet til å takle høyere strømmer enn P-channel MOSFETer [61]. Dette er viktig siden motorene som blir brukt i dronen vil mest sannsynlig trekke opp mot 30 A.



Figur 5.5: Kretsskjema av strømbryter-krets.

Ut ifra databladet ble da kretsen i Figur 5.5 designet. Ifølge databladet trengtes kun kondensator C5 å legges inn. Det ble likevel valgt å bruke to motstander (R8 og R9) for å sørge for at kretsene de er koblet til går til jord når komponenten er avskrudd, eller noe uforutsett skulle skje. I databladet er det en graf som sammenligner utspenning i forhold til strømmen ut, se Figur 5.4. Grafen viser i korte trekk at om det går mer enn rundt $40\mu A$ i kretsen, vil spenningen reduseres drastisk og muligens havne under 7,4 V. Dette vil si at det ikke blir høy nok gate spenning til at MOSFET driveren kan styre MOSFETen og unngå større spenningsfall over MOSFETen. Derfor er det valgt å bruke en $1M\Omega$ resistor på R9 da strømmen som går i kretsen vil være rundt $19\mu A$. Se formel 5.4 for bevis. Ifølge grafen i Figur 5.4 vil da spenningen ut av MOSFET driveren være på rundt 14 V til 16 V. $V_{DD}=IN=7,4$ V er desverre ikke beskrevet i grafen. Det var en antagelse som ble gjort under design. Om kretsen ikke fungerer planlagt kan motstanden byttes ut med en høyere.

$$I = \frac{V_{out}}{R9} = \frac{19V}{1M\Omega} = \underline{\underline{19\mu A}} \quad (5.4)$$

Grunnen til at V_{out} er satt til 19 V i formel 5.4 er fordi MOSFET driveren når maksgrensen på 19 V. Se formel 5.5 for bevis.

$$V_{out} = 4 * Vin - 2.8V \quad V_{out} = 4 * 7.4 - 2.8V = \underline{\underline{26.4V}} \quad (5.5)$$

Da både spenningsregulator- og strømbryter-kretsene var designet ble en fysisk bryter lagt inn i kretsskjema mellom batteri og Raspberry Pi Compute Module 4. Den fysiske bryteren er en liten av-på skyve-bryter som tåler en maksspenning på 24 V, og den har bryter-rating på 6 W. Maksspenning vil bli værende innenfor da batteriet har omtrent 8 V fult ladet. Derimot vil bryterratingen på 6 W bli overskredet ifølge databladet til Raspberry Pi Compute Module 4. Der står det at ettkortsdatamaskinen har typisk operasjon-strømtrekk på rundt 1,4 A, som ville gitt ut en effekt på 11,2 W. Se formel 5.6.

$$P = 8V * 1,4A = \underline{\underline{11,2W}} \quad (5.6)$$

I databladet til RPi CM4 står det også at det typiske operasjons-strømtrekket varier stort ut ifra bruk og operativsystem [58]. Siden det blir brukt et operativsystem med kommandolinjegrensesnitt

som har få programmer som kjører i bakgrunnen, er det høyeste strømtrekket målt til 0,86 A. Det vil gi en effekt på $P = 8V * 0,8A = 6,4W$. Det er fortsatt høyere effekt enn kravene databladet til bryteren tillater. Det regnes derimot med at dette vil gå fint. Om det skulle bli et problem er det mulig å bytte bryteren med en kabel, og bruke batteriets konnektor som hovedbryter istedet.

Etter at den fysiske bryteren ble bestemt, ble det lagt til: aktivitets-LED-kretser, noen GPIO jumper-hull og en åpen lodde-pad til å styre EEPROM skrivebeskytter. LED-kretsene ble lagt til for å fungere som status-indikasjon på Raspberry Pi Compute Module 4. GPIO hullene ble lagt til da det er lurt å ha ekstra GPIO pins å koble til. Den åpne loddepadden ble lagt til fordi den kan gjøre det umulig å skrive til EEPROM lagringen som er innebygget i RPi CM4. Dette kan forhindre at uønskede får tilgang til datamaskinen.

Videre ble det også lagt til en 6-pin JST-konnektor. Den er av samme type som går ut av autopiloten. UART skal brukes som kommunikasjon, derfor er det kun RX0, TX0 og jord som er koblet til konnektoren. En micro-USB konnektor er også lagt til. Kretsen til micro-USBen er designet til å fungere som en Device-USB. Det vil at en annen PC som kan skrive til eller lese EEPROM lagringen i RPi CM4.

Utskytningsenhet

Designet av bærekortet til Raspberry Pi Compute Module 4 som skal sitte i utskytningsenheten er gjort med utgangspunkt i funksjonalitetene som er beskrevet i kapittel 5.3.1. Ut i fra funksjonalitetene legges det en plan for hvordan bærekortet må designes slik at resultatet blir som planlagt. Det må designes en spenningsregulator-krets som håndterer spenningen fra batteriet og tilfører den nødvendige spenningen til datamaskinen. Det må også legges inn en fysisk bryter slik at datamaskin kan skrus av og på. For å kunne koble til og styre datamaskinen eksternt, må det legges opp til dette på bærekortet ettersom det ikke er noen innganger eller utganger på RPi CM4.

Det blir ikke fokusert så mye på å optimalisere yttermålene til bærekortet. Det er fordi utskytningsenheten er stor, og det vil uansett være god plass til bærekortet. Ved å sette yttermålene tidlig, kan utskytningsenheten og bærekortet designes samtidig.

Etter designet er planlagt kan det velges komponenter som skal være på bærekortet. Under bruk må det først og fremst kobles til et LiPo-batteri. For at dette skal være mulig, må det være en konnektor på bærekortet av samme type som utladningspluggen på batteriet. Etter noen undersøkelser viser det seg at det er Molex Mini-Latch som brukes, det må dermed benyttes en tilsvarende konnektor på bærekortet.

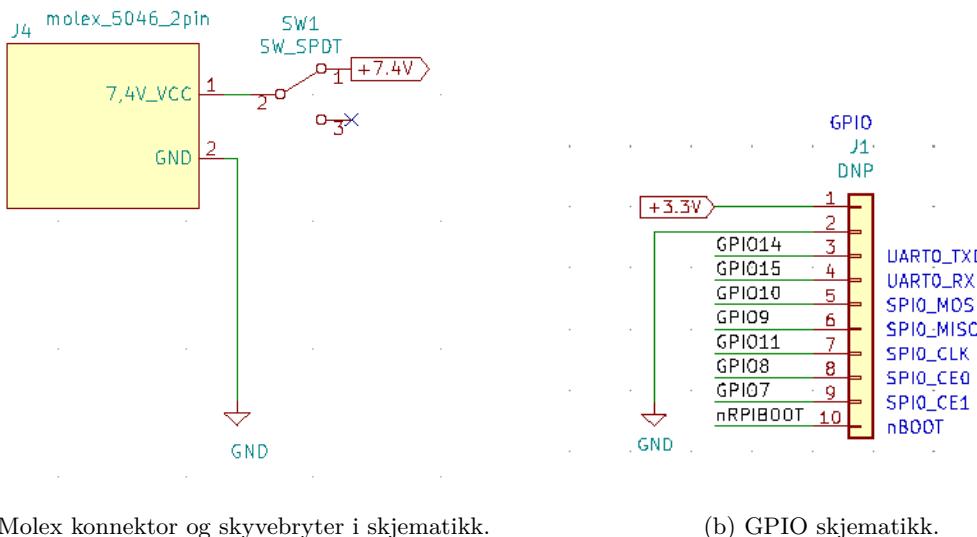
Videre må strømmen gå via en bryter slik at det er mulig å skru av og på strømtilførselen. For å gjøre dette ble det valgt en skyvebryter som tåler den oppgitte maksimale strømmen RPi CM4 kan trekke, 3 A [58]. Etter bryteren må det være en spenningsregulator som reduserer spenningen som kommer fra batteriet fra 7,4 V til 5 V. Bærekortet i dronen har en spenningsregulator-krets som gjør akkurat dette. For å unngå unødvendig dobbeltarbeid ble denne kretsen også brukt på bærekortet til utskytningsenheten.

For tilkobling ble det bestemt at det var nyttig med en USB-C og en HDMI konnektor. Med disse konnektorene kan det enkelt kobles til en skjerm og tastatur. Ettersom denne datamaskinen skal kjøre Ubuntu Server 20.04, er grensesnittet kun et terminalvindu. Derfor kreves det kun skjerm og tastatur for å bruke datamaskinen. Både USB-C og HDMI ble bestilt fra Molex, siden denne leverandøren har gode datablader og 3D-modeller som kan brukes under designprosessen.

Etter at designet var planlagt og de største og viktigste komponentene var valgt, ble bærekortet designet. Det ble i dette prosjektet benyttet KiCad som programvare for både tegning av skjematikk, design av kretskort og for eksport av produksjonsfiler. For å montere RPi CM4 på bærekortet kreves det at monteringshullene og konnektorene er korrekt plassert. Konnektorene

som benyttes til montering av RPi CM4 er to Hirose-konnektorer. For å sørge for at disse stemmer overens ved montering, ble det valgt å benytte en mal som utgangspunkt til design i KiCad. Denne malen er satt opp med riktige dimensjoner på både monteringshull og plassering av konnektorer. Det er i tillegg lagt inn i malen beskrivelser på pinnene til Hirose-konnektoren. Disse beskrivelsene forklarer hva de ulike pinnene representerer i henhold til RPi CM4. Dette er essensielt å vite for at koblingene på kortet skal bli riktig.

Etter at malen var åpnet ble strømkretsen satt inn og de største komponentene plassert og koblet opp til Hirose-konnektorene. RPi CM4 har flere 5 V innganger. 5 V fra strømkretsen ble koblet til alle disse pinnene. I tillegg ble alle jord-pinner koblet til felles jord. Molex-konnektoren til LiPo-batteriet og skyvebryteren ble koblet opp som vist i Figur 5.6 (a). Det ble også satt inn en grønn og en rød LED som henholdsvis ble koblet til ”nLED_Activity” og ”nPWR_LED” på Hirose-konnektoren. Disse LED-ene vil gjøre det mulig å observere status på datamaskinen ved bruk.



Figur 5.6: Molex konnektor, skyvebryter og GPIO skjematikk.

Videre ble HDMI-konnektoren satt inn og koblet til de korrekte pinnene på Hirose-konnektoren. Så ble USB-C-en satt inn, denne ble koblet opp på en litt annen måte. Pinne D+ og D- på USB-C-en ble koblet til pinne USB2_P og USB2_N, respektivt, via ESD beskyttelseskomponenten TPD2EUSB30 fra Texas Instruments. Pinne VBUS på USB-C-en ble koblet til USB_OTG_ID i parallel med to motstander på $2,2\text{ k}\Omega$. Jord-pinnen til USB-C-en ble koblet til felles jord.

Til slutt ble det bestemt at bærekortet også skulle ha noen generelle innganger og utganger (GPIO), disse i form av platede hull i kretskortet som det kan loddes ledninger til (se Figur 5.6 (b)). Fordelen med dette er at det er mulig å utvide funksjonaliteten til datamaskinen uten å måtte designe og produsere et helt nytt bærekort. RPi CM4 kan supplere 3,3 V og 1,8 V spennin ut til perifere enheter. Denne 3,3 V pinnen på Hirose-konnektoren ble koblet til den første pinnen i GPIO-rekken. Dermed vil mindre enheter som maksimalt krever 3,3 V forsyne seg fra denne pinnen. Det ble også lagt til et platet hull som går til GND.

Videre ble følgende GPIO-pinner fra Hirose-konnektoren koblet til rekken: 14, 15, 10, 9, 11, 8 og 7. Disse pinnene har, i tillegg til å være generelle innganger og utganger, enkelte spesielle funksjoner. Respektivt brukes GPIO-pinne 14 og 15 som standard til UART0 TX og UART0 RX. GPIO-pinne 10, 9, 11, 8 og 7 brukes som standard til 0_MOSI, 0_MISO, 0_CLK, 0_CE0_N og 0_CE1_N. Dette betyr at det er mulig å koble til perifert utstyr som benytter UART og kommunikasjonsprotokollene. Den siste pinnen i GPIO-rekken er koblet til Hirose-konnektor pinne 93, som kalles ”nRPIBOOT”. Denne pinnen er internt i RPi CM4 trukket høy. Om ”nRPIBOOT”

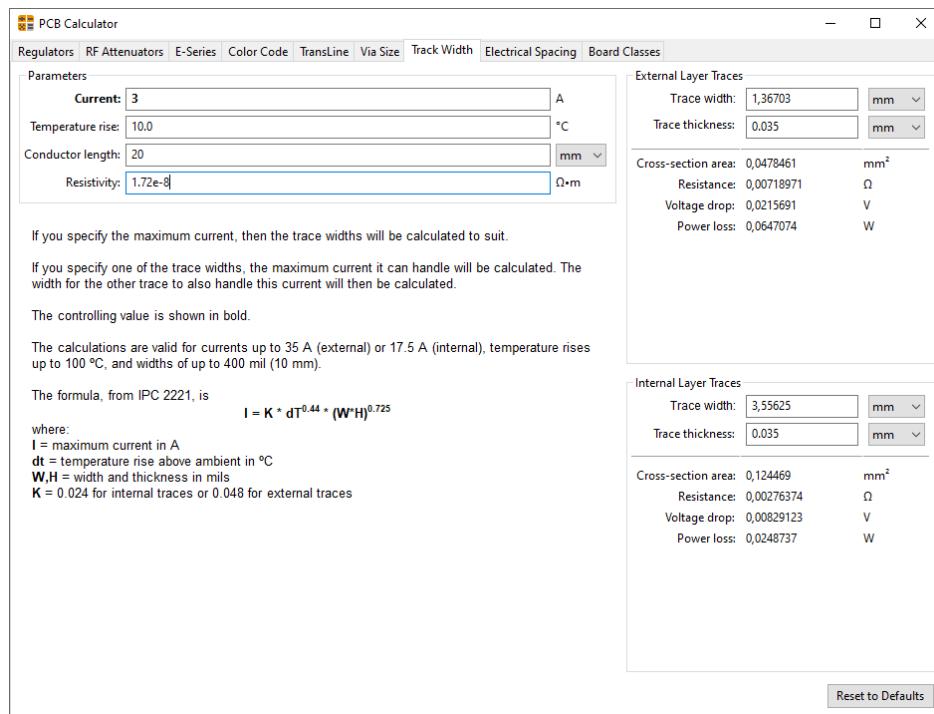
kobles til jord tvinger den en omstart av datamaskinen. Dette er kjekt dersom operativsystemet skulle fryse.

For å se komplett skjematikk på bærekort til dronen og utskytningsenheten, se appendiks D.1 og D.2.

5.3.3 Kretskortutlegg

Ved design av kretskortutlegg til bærekortene, er de samme designreglene fulgt for både dronen og utskytningsenheten. Banebredden er valgt på bakgrunn av den maksimale ut-strømmen fra RPi CM4 og kapasiteten til den ønskelige produsenten. Ifølge banebredde-kalkulatoren som følger med KiCad trenger ikke banene å være tykkere enn $0.5 \mu\text{m}$. Ifølge JLCPCB, en kinesisk kretskortprodusent, er minimum banebredde 0.127 mm for baner med kobbervekt på rundt 28.35 g og 0.2 mm for baner med kobbervekt på rundt 56.70 g [62]. Det er valgt å følge den største minimumsbredden til produsenten, 0.2 mm [63].

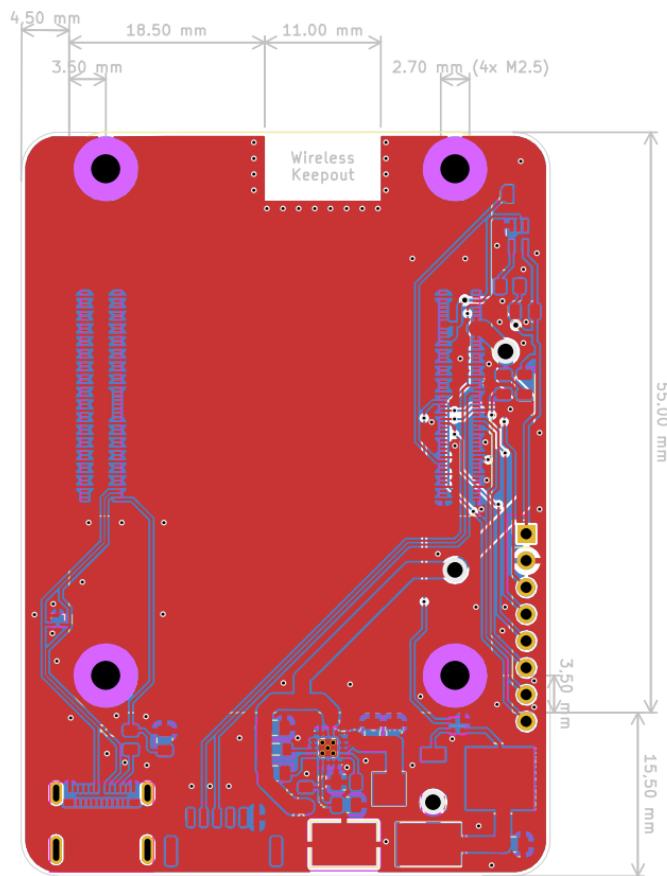
En bredde på 0,2 mm er fint for de fleste banene. Baner med høyere strømføring krever derimot bredere baner. Et eksempel på en slik bane er den som starter ved batteriet, videre gjennom spenningsregulatoren og ender ved 5 V-pinnene på Hirose-konnektorene. Ved å bruke banebredde-kalkulatoren til KiCad og sette inn det strømmen RPi CM4 maksimalt kan trekke, blir det foreslått en banebredde på minimum 1.37 mm (se Figur 5.7), dette er en del bredere enn banen brukt ellers på kortet. Derfor er det valgt å bruke en banebredde på 1.4 mm på denne høy-strømførende banen fra batteriet og inn til Hirose-konnektoren.



Figur 5.7: Anbefalt banebredde fra KiCAD sin PCB Calculator med en strøm på 3 A.

Drone

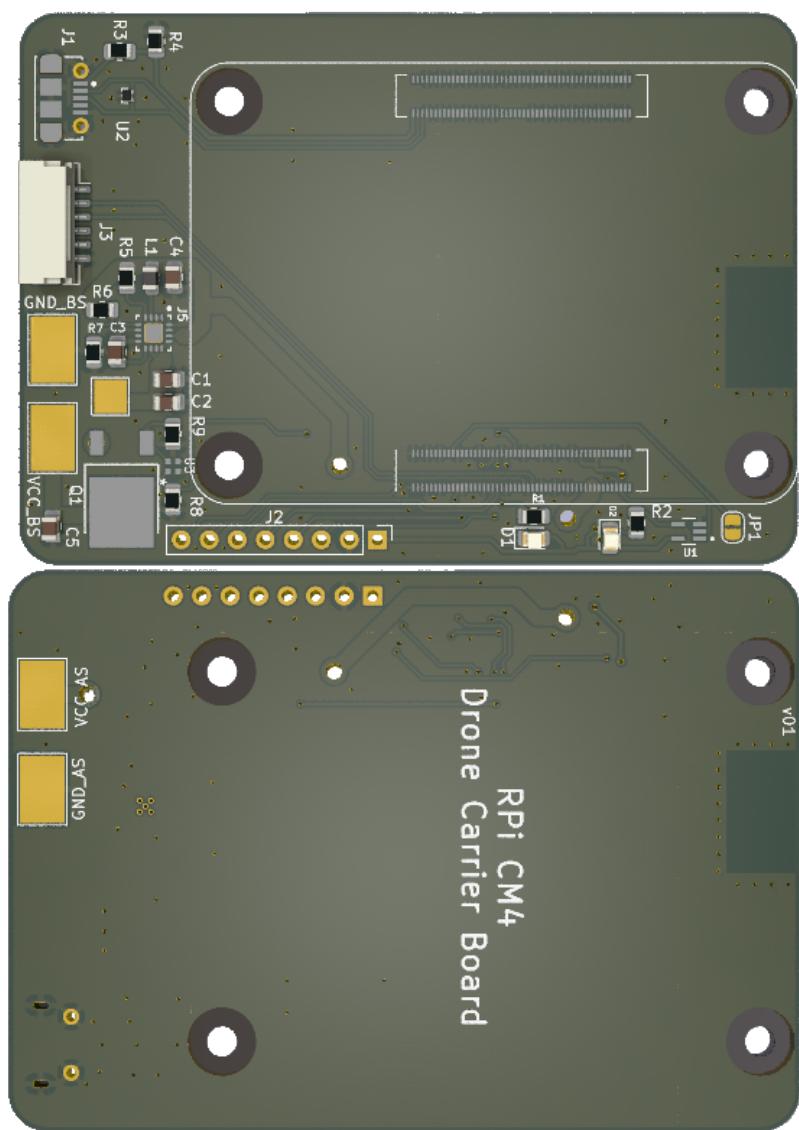
For at et bærekort skal være kompatibelt med RPi CM4 er det viktig at konnektorene mellom RPi CM4 og bærekortet, samt skruehullene, er korrekt plassert. En annen viktig designregel for dette kortet er at komponenter ikke bør plasseres rett under ettkortsdatamaskinen, da den har komponenter som kan komme i kontakt med komponenter på bærekortet. Dette bunner ut i at RPi CM4 som nevnt har to stykk Hirose DF40C-100DS-0.4V konnektorer som står for alle pinouts på RPi CM4. Konnektorene er rundt 1,54 mm høye, som vil si at avstanden mellom bærekortet og RPi CM4 vil bli på rundt 1,54 mm. Derfor ble bærekortet gjort 15,5 mm lengre og 9 mm bredere enn RPi CM4 for å få plassert komponentene på sidene istedenfor, se Figur 5.8. Komponentene kunne vært festet på begge sider av bærekortet, men da det bør være mulig produsere kretskortet på universitetet er det isteden valgt å ekspandere kortet.



Figur 5.8: Teknisk tegning av yttermål på RPi CM4 bærekort til dronen.

Da skruehullene og størrelsen på bærekortet ble satt, var det første som ble gjort å plassere Hirose-konnektorene. Deretter ble resten av de større komponentene og konnektorene plassert ut. Konnektorene ble satt ved kantene, mens resten av komponentene ble flyttet lengre inn på kortet. Heretter ble resten av de mindre komponentene plassert ut i henhold til plasseringen av de større komponentene.

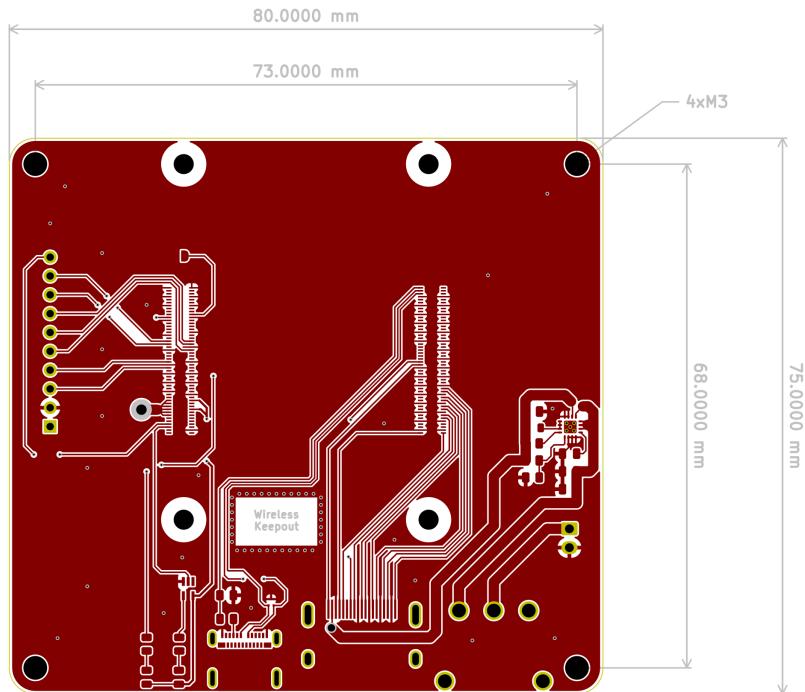
Da alle komponentene var plassert ut ble det begynt å tegne opp ledningsnettet. På grunn av plasseringen på enkelte komponenter ble det underveis oppdaget at det ble umulig å fullføre ledningsnettet. Mindre justeringer på plasseringene til komponentene ble derfor justert underveis.



Figur 5.9: Komplett bærekort til drone.

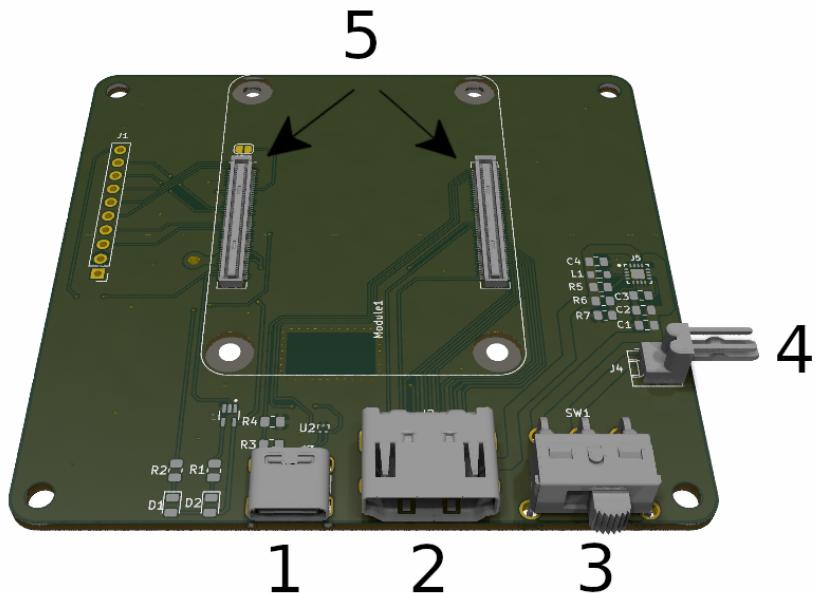
Utskytningsenhet

Ved design av kretskortutlegg til utskytningsenheten ble yttermålene på kretskortet og monteringshullene tegnet opp først. Som vist i Figur 5.10 er kretskortet 80 mm x 75 mm. Som nevnt tidligere ble det tatt utgangspunkt i en mal for å få nøyaktig plassering av Hirose-konnektorene og monteringshullene til RPi CM4. Hirose-konnektorene bygger rundt 1,54 mm og på RPi CM4 er det montert komponenter både på under- og oversiden. Det bør derfor unngås å plassere komponenter på oversiden av kretskortet i dette området. Som vist i Figur 5.11 er malen til RPi CM4 plassert omtrent midt på kretskortet i bredderetning og langs bakre kant i lengderetning. Hirose-konnektorene er markert med tallet 5, mens de ytre kantene av malen er markert med hvit linje. Dette gjør det enklere å se hvor de andre komponentene ikke kan plasseres.



Figur 5.10: Teknisk tegning av yttermål RPi CM4 bærekort til utskytningsenheten.

Videre ble de største komponentene, som USB-C, HDMI, skyverbryteren og Molex-konnektoren, plassert. Det er ønskelig med enkel tilgang til disse komponentene. Derfor ble disse plassert langs kanten på kretskortet. 3D-modellen som vises i Figur 5.11 er generert i KiCad og viser plasseringene av USB-C, HDMI og skyverbryteren, henholdsvis markert som 1, 2 og 3. I tillegg viser modellen plasseringen av Molex-konnektoren til batteriet (4). Denne er også plassert langs kanten, men på høyre side siden planen er at batteriet skal plasseres ved siden av kortet.



Figur 5.11: Plassering av de største komponentene bærekort utskytningsenhet.

Komponentene som tilhører spenningsregulator-kretsen ble plassert over Molex-konnektoren. Det vil si at disse komponentene ligger i nærheten av både Molex-konnektoren og skyvebryteren. Dette gjør det enklere når banene skal tegnes. Komponentene som tilhører LED-ene ble plassert ved siden av USB-C konnekturen for at det skal være enklere å se status på datamaskinen når den er i bruk. De generelle inngangene og utgangene ble plassert på venstre side, der de ikke i veien for noen av de andre komponentene. Det er også forventet at det er god plass på denne siden av kortet dersom det er ønskelig med utvidelser av funksjonalitetene.

Etter alle komponentene var plassert ut ble banene tegnet. KiCad har en hendig løsning som kalles ratsnest. Ratsnest er et verktøy som viser hvilke pinner på de ulike komponentene som skal kobles til hverandre ved hjelp av hvite linjer. Denne funksjonen ble også brukt under plassering av komponentene som en veiledning på hvor komponentene bør plasseres for å unngå mest mulig krøll når banene skal tegnes. Ved å bruke disse hvite linjene tegnes banene mellom pinner som må ha kontakt. Der baner krysser hverandre er det brukt via-er. En via er et hull som går gjennom kretskortet og kobler en bane på oversiden med en bane på undersiden.

5.3.4 Produksjon

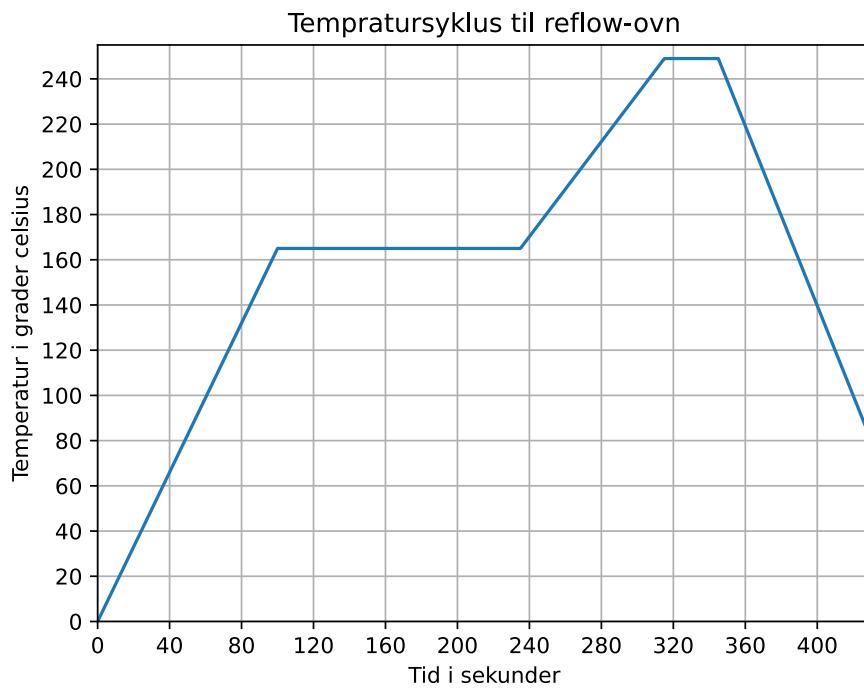
Kretskortene er designet til å kunne produseres for hånd ved hjelp av utstyret som er tilgjengelig på universitetet. Det er valgt å benytte metoden reflow-lodding. Denne metoden gjør det lettere å produsere kretskortene siden loddepasta først påføres på de nødvendige padene. Deretter plasserer komponentene ovenpå loddepastaen. Pastaen fungerer som et lim slik at kretskortet kan flyttes på uten at komponentene beveger seg. Når alle komponentene er festet på skal kretskortet inn i ovnen for å reflowes, sammen med loddepastaen og komponentene. Det vil si at ovnen settes på en syklys som gjør at loddepastaen herder, og dermed holder komponentene.

Under reflowing når ovnen høye temperaturer. Derfor er det viktig å ha satt en syklus som både tar hensyn til at loddepastaen herder og makstemperaturen til de ulike komponentene. Komponenten som har den laveste varmebegrensingen for bærerkortene er Hirose-konnektorene. Ifølge databladet kan de maksimalt nå en temperatur på 250°C, og kan maks være på 220°C i 60 sekunder. Under forvarmingsprosessen kan konnektorene maksimalt ligge på mellom 150°C og 180°C i 90 til 120 sekunder [64]. Da det var kjøpt inn ekstra Hirose-konnektorer og bærekart, ble det besluttet å gjennomføre en test. Syklusen ble først satt til å forvarme kretskortet på 165°C i 110 sekunder, og deretter reflowe på 220°C i 60 sekunder. Resultatet av dette var at loddepastaen ikke fikk herdet. Komponentene var dermed dårlig festet, som førte til dårlig kontinuitet i flere av banene på kretskortet. Dette skjer fordi loddepastaen som ble brukt, CHIPQUIK TS391SNL, har en smeltemperatur på 220°C. Og er ifølge databladet anbefalt å ha en reflow syklus som varer i 110 sekunder som når en topptemperatur på 249°C. Forvarmingsprosessen er anbefalt å vare fra 90 til 180 sekunder, på 150°C til 175°C [65]. Derfor ble det istedenfor forsøkt å endre reflow syklusen til anbefalingene fra loddepastaen sitt datablad. Hirose-konnektorene tålte den nye reflow syklusen selv om databladet fraråder temperaturene som ble brukt i syklusen [64].



Figur 5.12: Bilde av herdeovnen som ble tatt i bruk.

Syklusen som da ble brukt til å lodde bærekartene startet da med en formvarmingsprosess på 165°C som varte i 135 sekunder. Deretter steg temperaturen gradvis i 80 sekunder til den nådde 249°C. Temperaturen holdt seg så på 249°C i 30 sekunder før ovnen skrudde seg av, og gradvis sank tilbake til romtemperatur. Se Figur 5.13.



Figur 5.13: Graf av temperatursyklus.

Når ovnen har vært på 249°C i 30 sekunder kan døren åpnes og bærekortene kan tas ut. Da kortene var avkjølt ble gjenværende konnektorer loddet på manuelt. De gjenværende konnektorene som ble loddet på var skyvebryteren og MOLEX Mini-Latch, som ble loddet på bærekortet til utskytningsenheten. Disse ble ikke satt på under reflow-prosessen da de ikke ville tålt temperaturen. Samtidig som disse ble loddet på ble også de andre konnektorene på kretskortene tilført litt mer tinn for å sørge for at de var godt festet.

Da kretskortene var ferdig produsert ble de testet for å verifisere at de fungerte som planlagt. Se resultatene av testene i kapittel 9.1 og 9.2.

5.4 Oppsett

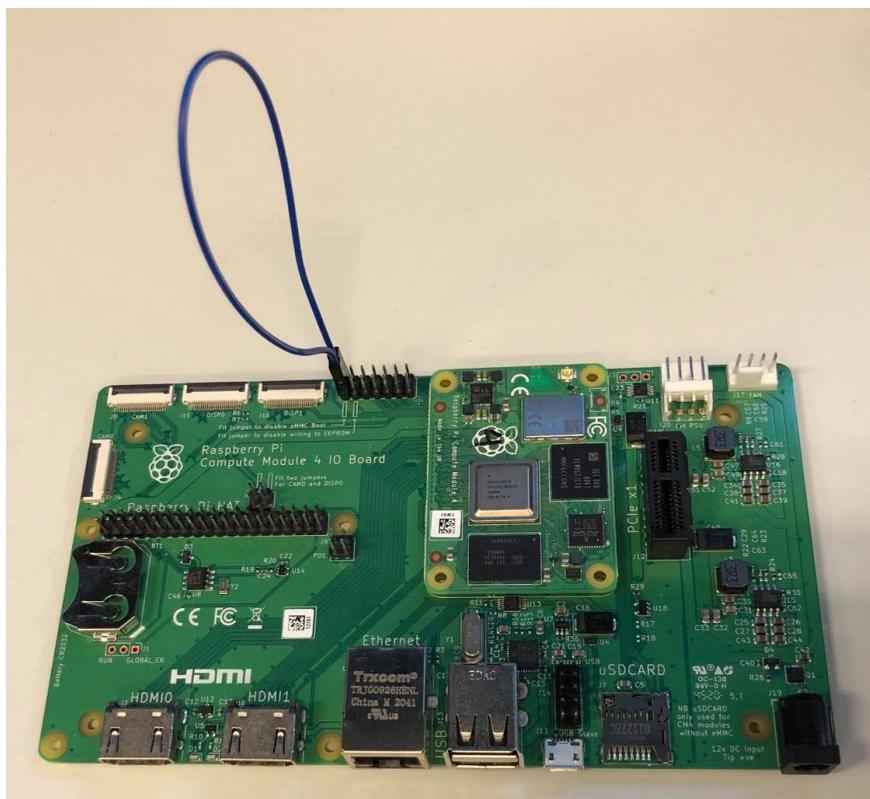
Den valgte datamaskinen må konfigureres slik at den fungerer som planlagt. Med dette menes at RPi CM4 må utføre en rekke operasjoner ved oppstart som for eksempel å koble seg til et bestemt Wi-Fi med en statisk ip-adresse. I prosjektet skal det lages flere droner og én utskytningsenhet, som tilsier at det kreves flere Raspberry Pi Compute Module 4. Følgende oppsett er utført på samtlige enheter.

5.4.1 Installasjon av operativsystem

For å få det ønskede operativsystemet på datamaskinen må det gjennomføres enkelte steg. Ettersom datamaskinen som er valgt har integrert 32 GB eMMC lagring, installeres operativsystemet direkte på enheten. Dette avviker fra hva som er normalt med andre RPi-enheter, der operativsystemet ofte installeres på et SD-kort som må være i ved bruk.

Montering på I/O-brett

Først monteres Raspberry Pi Compute Module 4 på IO-brettet utviklet av Raspberry Pi Foundation. Dette brettet er nyttig å ha under denne prosessen siden den har en rekke innganger og utganger. Dette gjør det enklere å konfigurere enheten etter at operativsystemet er installert. Etter at enheten er montert må ”nRPI_BOOT”-pinnen kobles til GND, som vist i Figur 5.14. Dette gjør at når enheten skrus på vil den ikke starte opp som normalt fra eMMC, men den vil starte opp fra USB. Videre må det også brukes en ekstern datamaskin for å installere det valgte operativsystemet, i dette prosjektet ble det brukt en bærbart pc med Ubuntu 20.04.



Figur 5.14: RPi CM4 montert på I/O-brett.

Programvare for installasjon av operativsystem

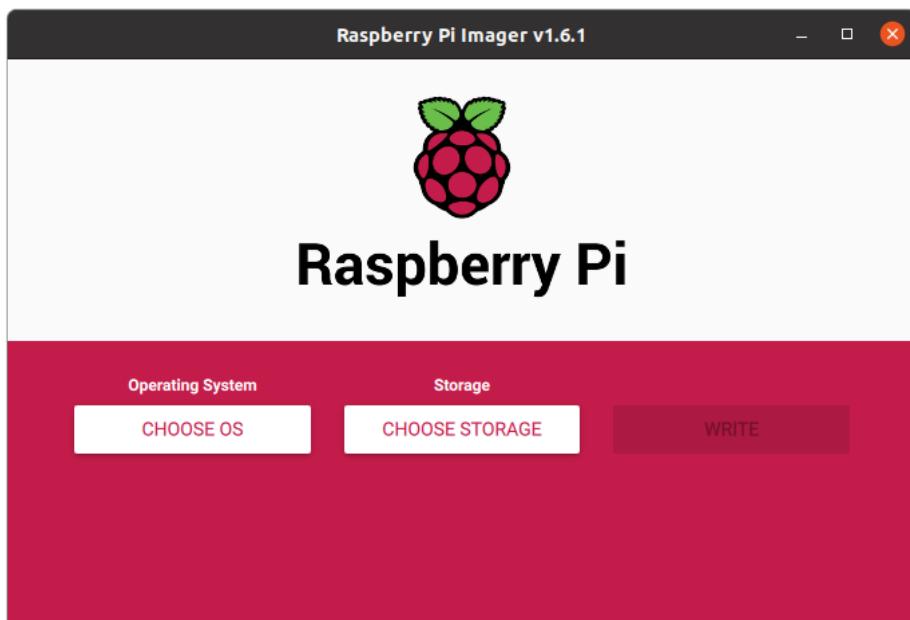
På den eksterne datamaskinen klones usbboot-pakkebrønnen fra Raspberry Pi sin Github ved å kjøre kommandoen vist i Kodelisting 5.1. Parameteren `--depth=1` gjør at det kun er siste revisjon av pakkebrønnen som klones ned.

Kodelisting 5.1: Kloning av usbboot-pakkebrønn.

```
$ git clone --depth=1 https://github.com/raspberrypi/usbboot
```

Usbboot er et verktøy utviklet av Raspberry Pi Foundation og gjør at den tilkoblede datamaskinen tolker Raspberry Pi Compute Module 4 som en ekstern harddisk. Siden dette kun er kildekoden til verktøyet, må den bygges for å kunne brukes. Før dette anbefaler Raspberry Pi Foundation å sjekke om det mangler noen avhengigheter i systemet. Ifølge utvikleren trengs pakken "libusb" for å kunne bygge verktøyet [66]. Denne pakken installeres ved å kjøre komandoen `sudo apt install libusb-1.0-0-dev` i terminalen. For å bygge verktøyet må komandoen `make` kjøres i terminalen fra rotmappen til kildekoden. Etter byggeprosessen, brukes verktøyet ved å skrive komandoen `sudo ./rpiboot` i terminalen.

Den eksterne datamaskinen er da klar for å oppdage RPi CM4 på I/O-brettet, derfor kobles det til en micro-USB kabel fra I/O-brettet til den eksterne datamaskinen. Deretter kobles strømforsyningen til I/O-brettet og systemet skrur seg på. Etter at rpiboot er ferdig, dukker enheten opp på den eksterne datamaskinen i `"/dev"`-mappen, som leses ved å kjøre `lsblk` komandoen i terminalen. Herfra finnes det flere fremgangsmåter for å installere det valgte operativsystemet. I dette prosjektet ble det valgt å benytte Raspberry Pi Foundation sin egen installasjonsprogramvare, Raspberry Pi Imager. Programvaren har et vennlig brukergrensesnitt og fungerer utmerket til å installere det valgte operativsystemet på enheten (5.15).



Figur 5.15: Raspberry Pi Imager brukergrensesnitt.

Først må image-filen av operativsystemet som skal installeres på enheten lastes ned. Dette gjøres direkte fra Ubuntu sine nettsider ettersom det er Ubuntu Server 20.04 som skal brukes. Ubuntu har en egen versjon som er laget spesifikt for Raspberry Pi enheter. Disse finnes i "Downloads"-menyen og under "Ubuntu for IoT", der kan det velges "Raspberry Pi 2,3 or 4". Ubuntu Server kan lastes ned både som 32-bit og 64-bit. I dette prosjektet lastes 64-bit versjonen ned, siden det er nødvendig for at ROS 2 skal fungere. Deretter kan installasjonsprosessen starte ved å skrive den nedlastede image-filen til RPi CM4.

5.4.2 Oppstart

Etter at installasjonen er ferdig trekkes strømkabelen ut, og kabelen som kobler ”nRPI_BOOT” til jord og micro-USB kabelen fjernes. Skjerm og tastatur kobles til ved bruk av HDMI- og USB-portene på IO-brettet, deretter kobles I/O-brettet til strøm igjen. Enheten starter opp og innloggingen til Ubuntu dukker opp på skjermen.

USB aktivering

Da oppstartsprosessen var ferdig, fungerte ikke tastaturet. Hverken ved å bruke en av de standard USB-portene eller micro-USB porten. Feilen ble funnet i databladet, der det stod at USB-portene på RPi CM4 i utgangspunktet er deaktivert for strømsparing [58]. Dette ble fikset ved å koble enheten opp til en ekstern datamaskin. Fra den eksterne datamaskinen kan konfigurasjonsfilen endres, dette er en fil som kan endre konfigurasjonen til enheten ved oppstart. I denne filen legges det til `dtoverlay=dwc2,dr_mode=host`. Ved omstart er USB-portene aktivert og konfigurasjonen fortsetter som planlagt.

Wi-Fi konfigurasjon

Når datamaskinen skrur seg på, er det ønskelig at den kobler seg til et forhåndsbestemt Wi-Fi. Siden Ubuntu Server 20.04 har kommandolinje-brukergrensesnitt, kreves det en annen fremgangsmetode enn ved grafisk brukergrensesnitt. I systemfilene må ”50-cloud-init.yaml” endres slik at datamaskinen vet hvilket nettverk den skal koble seg til (5.16). Denne filen ligger i systemmappen ”/etc/netplan”. Filen inneholder i utgangspunktet kun linjene over ”wifis”, men for at datamaskinen skal koble til Wi-Fi må linjene fra og med ”wifis:” legges til. Figur 5.16 viser en konfigurasjon der datamaskinen kobler seg til Wi-Fi-nettverket ”student-FoU”, som er et Wi-Fi tilgjengelig i maskinhallen på UiA. I tillegg til å koble seg til dette nettverket settes det også en statisk IP-adresse. Denne adressen er ulik på de forskjellige enhetene for å unngå konflikter.

```
# This file is generated from information provided by the datasource. Changes
# to it will not persist across an instance reboot. To disable cloud-init's
# network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
    ethernets:
        eth0:
            dhcp4: true
            optional: true
        version: 2
    wifis:
        wlan0:
            dhcp4: false
            addresses: [10.245.30.151/24]
            gateway4: 10.245.30.1
            nameservers:
                addresses: [8.8.8.8, 4.4.4.4]
            optional: true
            access-points:
                "student-FoU":
                    password: "_____"
```

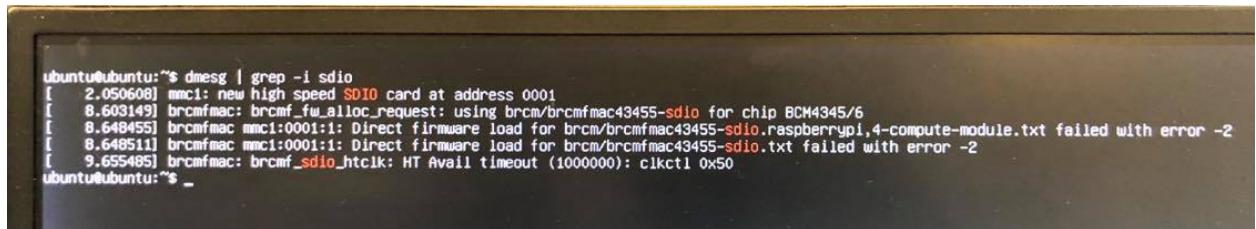
Figur 5.16: Konfigurerering av nettverk. Passordet er sensurert for sikkerhetshensyn.

Etter ”50-cloud-init.yaml”-filen er lagret oppdateres nettverks-innstillingene på RPi CM4 ved å kjøre kommandoen `sudo netplan apply`. Da skal datamaskinen koble seg til nettverket som er konfigurert. I dette tilfellet skjedde det ikke. Selv ved omstart ville ikke datamaskinen koble seg til det riktige nettverket. Kommandoen `ip a` ble kjørt for å se om det konfigurerte Wi-Fi nettverket dukket opp. Det ble oppdaget at det er ikke eksisterte noen ”wlan0”-grensesnitt på datamaskinen. Dette var overraskende da RPi CM4 enhetene ble bestilt med innebygget Wi-Fi. For å feilsøke

problemet videre ble følgende kommando kjørt i terminalen `sudo hwinfo --network --short`. Kommandoen lister opp tilgjengelige nettverksgrensesnitt. Ved å skrive kommandoen dukket kun ”eth0” og ”lo” opp, her skulle også ”wlan0” (Wi-Fi) vært oppført.

Driverproblem

Ettersom ”wlan0”-grensesnittet ikke dukket opp, var det mistanke om at det var et driver-problem. Det ble gjort søk i diagnostikk-meldingene som logges ved oppstart, følgende kommando ble kjørt `dmesg | grep -i sdio`. Dette er en kommando som søker gjennom diagnostikk-meldingene og sorterer ut meldinger som inneholder ”sdio”, og ignorerer forskjellen på store og små bokstaver. Fra søket ble det som forårsaket feilen trolig funnet. På linje 3 (se Figur 5.17) så det ut som at fastvaren prøver å laste RPi CM4 driver-konfigurasjonen til Broadcom BCM43455-brikken som sitter på datamaskinen, men feiler. Det så også ut som det er to driver-konfigurasjoner som manglet, én spesifikt til RPi CM4 og én til brikken generelt.



```
ubuntu@ubuntu:~$ dmesg | grep -i sdio
[ 2.050608] mmc1: new high speed SDIO card at address 0001
[ 8.603149] brcmfmac: brcmf_fw_alloc_request: using brcm/brcmfmac43455-sdio for chip BCM4345/6
[ 8.648455] brcmfmac mmc1:0001:1: Direct firmware load for brcm/brcmfmac43455-sdio.raspberry,4-compute-module.txt failed with error -2
[ 8.648511] brcmfmac mmc1:0001:1: Direct firmware load for brcm/brcmfmac43455-sdio.txt failed with error -2
[ 9.655489] brcmfmac: brcmf_sdio_htclk: HT Avail timeout (1000000): clkctl 0x50
ubuntu@ubuntu:~$
```

Figur 5.17: Søk i diagnostikk-meldingene.

Videre ble det gjort søk i mappen der driverene ligger, i Ubuntu Server 20.04 ligger disse i mappen ”/lib/firmware/brcm/”. Ved å liste innholdet i mappen dukker det opp en mengde drivere og konfigurasjoner, ved å studere listen blir det lagt merke til at driver-konfigurasjonene ”brcm43455-sdio.txt” og ”brcmfmac43455-sdio.raspberry,4-compute-module.txt” ikke eksisterer. Det er merkelig da dette operativsystemet, i følge nettsiden til utvikleren, har støtte for Raspberry Pi 4 som har akkurat den samme brikkesettet som Raspberry Pi Compute Module 4 [67]. Etter flere søk på internett etter andre som muligens har opplevd samme problem, ble det funnet et par forum-innlegg, men ingen løsninger på problemet. Figur 5.18 viser til en bruker med samme problem.

Raspberry Pi 4 Compute module 4 CM4 no Wifi with Ubuntu 20.04 server

Asked 1 month ago Active 26 days ago Viewed 216 times

I have installed Ubuntu 20.04 Server on a Raspberry pi CM4 (not the standard pi, the compute module), but cannot figure out how to enable the wifi. the CM4 has it physically installed and the board's wifi works under raspios (after running through raspi-config and setting it up), but with the ubuntu image provided by ubuntu, i cannot even see wlan0 (or any wifi adapter) when running

`ifconfig`

or

`ls /sys/class/net`

i am only shown that eth0 and lo exist. i have a feeling it must have to do with enabling the correct spi port or wifi parameters in the dto, but cannot find any documentation on it. Anyone else seeing this with the new compute module?

`wireless`

Share Edit Follow

asked Mar 25 at 12:58

D David Wilmot
1

The Overflow Blog

- ↗ Level Up: Creative part 8
- ↗ Don't push that bu software that flies

Featured on Meta

- ⌚ Testing three-vote 13 network sites
- ⌚ We are switching t 10, 2021
- .ask Policy Clarification Standard Support*
- .ask End of Support No (Xenial Xerus) reac

Looking for a job

- next → Senior DevOps Risk Focus 🌐 N
REMOTE

Figur 5.18: Spørsmål i Ubuntu-forum.

Driverløsning

For å forsøke å løse problemet ble det søkt på internett etter de manglende driver-konfigurasjonene. På Github finnes det en pakkebrønn som inneholder diverse gammel fastvare til drivere og driver-konfigurasjoner, men som også oppdateres kontinuerlig med nyere og ulanserte versjoner [68]. Pakkebrønnen er laget av RPi-Distro og heter "firmware-nonfree". I mappen "brcm" ligger det en rekke drivere og driver-konfigurasjoner til Broadcom BCM-brikker, blant annet én av de manglende driver-konfigurasjonen, "brcmfmac43455-sdio.txt". Inne i brcm-mappen på Raspberry Pi Compute Module 4 lastes driver-konfigurasjonen ned ved å kjøre kommandoen:

Kodelisting 5.2: Nedlasting av driver fra Github.

```
$ sudo wget https://raw.githubusercontent.com/RPi-Distro/firmware-nonfree/master/brcm/brcmfmac43455-sdio.txt
```

Den siste driver-konfigurasjonen ble ikke funnet på internett. Det ble derimot lagt merke til at driver-konfigurasjonene "brcmfmac43456-sdio.raspberrypi,4-compute-module.txt" og "brcmfmac43456-sdio.raspberrypi,4-model-b.txt" eksisterer. Disse er driver-konfigurasjonene til Broadcom-brikken BCM43456, som er en annen enn den som er montert på enheten brukt i dette prosjektet. Ettersom det både finnes både en Raspberry Pi 4 og en RPi CM4 versjon og disse skal ha samme brikkesett, er det interessant å sammenligne innholdet i disse driver-konfigurasjonene.

Sammenligningen gjøres ved å kjøre kommandoen `diff brcmfmac43456-sdio.raspberrypi,4-compute-module.txt brcmfmac43456-sdio.raspberrypi,4-model-b.txt` i terminalen. Ut ifra sammenligningen kom det frem at filene var identiske. Siden "brcmfmac43455-sdio.raspberrypi,4-model-b.txt"-konfigurasjonen var tilgjengelig, ble det forsøkt å duplisere denne, for deretter å endre navn til "brcmfmac43456-sdio.raspberrypi,4-compute-module.txt". Ved omstart ble kommandoen `sudo hwinfo --network --short` kjørt, og da dukket "wlan0" opp og enheten koblet seg til det Wi-Fi-nettverket som ble satt opp tidligere.

Siden dette er et problem flere har og som ikke er besvart, ble det formulert et svar til innlegget vist over (Figur 5.18). I svaret ble det forsøkt å forklare at problemet er identisk og hvordan problemet ble løst. Svaret vises i Figur 5.19.

Problem

- 1 My student research group had this issue after flashing RPi/CM4 with Ubuntu Server 20.04 LTS; when running `ls /sys/class/net` and `sudo hwinfo --network --short` only eth0 and lo exists.

Problem solving

We ran `dmesg | grep -i sdio` and discovered an error regarding failed to load driver for Wi-Fi chipset ([image](#)).

When checking the drivers installed, by running `ls /lib/firmware/brcm/`, there were two drivers missing ([image](#)):

```
brcmfmac43455-sdio.txt  
brcmfmac43455-sdio.raspberrypi,4-compute-module.txt
```

Solution

note: be aware of the difference between 43455 and 43456.

1. We found the first driver on github: [https://github.com/RPi-Distro/firmware-nonfree/
blob/master/brcm/brcmfmac43455-sdio.txt](https://github.com/RPi-Distro/firmware-nonfree/blob/master/brcm/brcmfmac43455-sdio.txt)

Just download it in the folder using

```
wget https://raw.githubusercontent.com/RPi-Distro/firmware-nonfree/master/brcm/brcmfmac43455-sdio.txt
```

2. For the second driver we did not find anything, however, we compared the the following drivers:

```
brcmfmac43456-sdio.raspberrypi,4-compute-module.txt  
brcmfmac43456-sdio.raspberrypi,4-model-b.txt
```

and discovered they where identical. Just duplicate the `brcmfmac43455-sdio.raspberrypi,4-model-b.txt` and rename it to `brcmfmac43456-sdio.raspberrypi,4-compute-module.txt`.

After reboot Wi-Fi works!

Distro issue? We believe this may be a distro-issue..

Share Edit Delete Flag

edited Apr 14 at 11:04

answered Apr 14 at 8:43

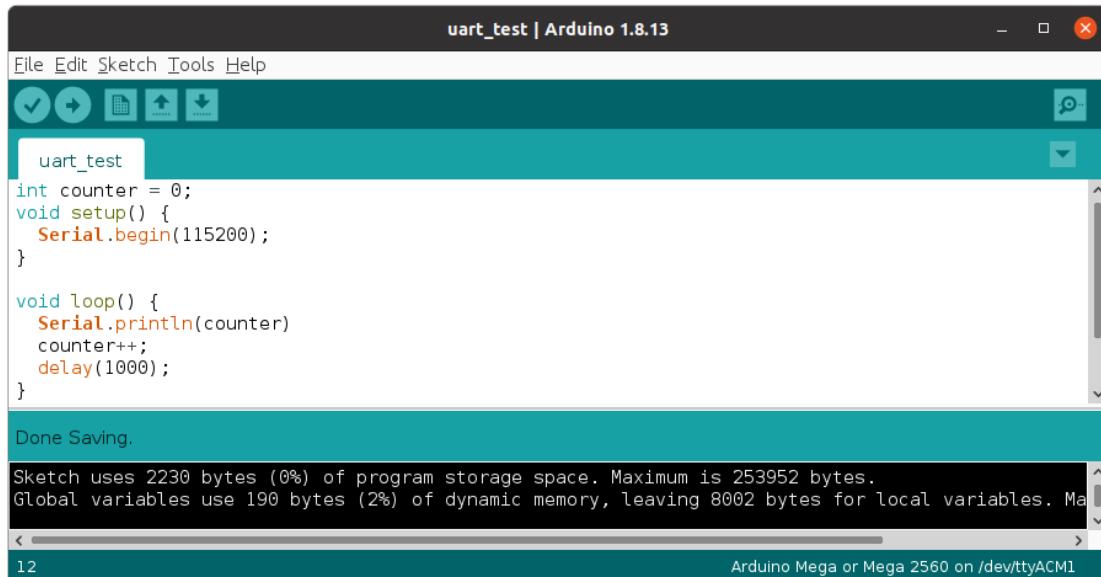
 Martin Maeland
11 • 2

Figur 5.19: Svar i Ubuntu-forum.

UART kommunikasjon

Videre er det planlagt at Raspberry Pi Compute Module 4 skal kommunisere med autopiloten ved å benytte UART-kommunikasjonsprotokoll. Det er ønskelig å teste at UART-kommunikasjonen fungerer feilfritt før konfigurasjonen fortsetter. Det ble derfor satt opp en test med UART-kobling mellom RPi CM4 montert på I/O-brettet og en Arduino Mega2560.

Først ble det koblet en kabel fra TX0 (GPIO14) på RPi CM4 til RX0 (GPIO0) på Arduino. Deretter ble det koblet en kabel fra RX0 (GPIO15) på RPi CM4 til TX0 (GPIO1) på Arduino. Videre ble det skrevet et enkelt program til Arduinoen som sendte en int over UART én gang i sekundet. Hver gang int-en ble sendt, økte den med én, som vist i Figur 5.20. BAUD-raten ble satt likt for både RPi CM4 og Arduinoen til 115200.



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** uart_test | Arduino 1.8.13
- Menu Bar:** File Edit Sketch Tools Help
- Tool Buttons:** Checkmark, Refresh, Open, Save, Upload, Download, Print, Help
- Sketch Area:** The code for 'uart_test' is displayed:

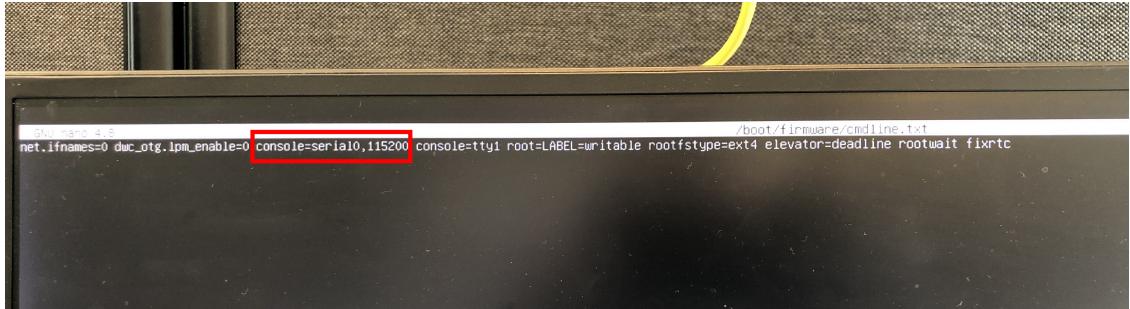
```
uart_test
int counter = 0;
void setup() {
    Serial.begin(115200);
}

void loop() {
    Serial.println(counter)
    counter++;
    delay(1000);
}
```
- Status Bar:** Done Saving.
- Serial Monitor:** Shows the message: "Sketch uses 2230 bytes (0%) of program storage space. Maximum is 253952 bytes. Global variables use 190 bytes (2%) of dynamic memory, leaving 8002 bytes for local variables. Ma".
- Bottom Status:** 12 Arduino Mega or Mega 2560 on /dev/ttyACM1

Figur 5.20: Arduino UART test-program.

Etter at programmet ble lastet ned på Arduinoen startes automatisk løkken som sender data over UART. På RPi CM4 er det mulig å høre på innkommende UART-meldinger ved å kjøre kommandoen `echo /dev/ttys0`, der ttys0 er standard UART-port på RPi CM4 [58]. Etter at denne kommandoen ble kjørt, dukket det opp en feilmelding i stedet for de forventede meldingene fra Arduinoen. Det var en ”permission denied”-feilmelding, som typisk omhandler rettigheter. Det ble forsøkt å kjøre samme kommando som tidligere, men nå med `sudo` foran. Dette resulterte i samme feilmelding som tidligere.

Ved å kjøre kommandoen `ls -l /dev/ttys0` er det mulig å se rettighetene til ttys0-enheten. Ifølge rettighetene skal det være mulig for superbrukeren å både lese og skrive til denne enheten. Det er tidligere gjort endringer i konfigurasjons-filen, der er det ikke noe som overkjører lesing og skriving til enheten. Filen ”/boot/firmware/cmdline.txt” sjekkes også, siden denne sender parametere til kjernen ved oppstart. Her finnes det noe tekst som tilsier at ”serial0” settes av til konsollen, se det i Figur 5.21 som er markert i rødt. Ved å fjerne denne teksten, og gjennomføre en omstart av systemet fungerte kommandoen `cat /dev/ttys0` som forsøkt tidligere. UART-kommunikasjonen betraktes etter dette som fungerende.



Figur 5.21: ”Serial0” settes av til konsoll.

Styre MOSFET-strømbryter

I 5.3.2 ble det designet en krets som gjør det mulig for Raspberry Pi Compute Module 4 å skru resten av systemene i dronen av og på. For å sende signal til MOSFET driveren er det planlagt å bruke pinne 27 på RPi CM4. Om pinnen settes lav vil MOSFET-en være av, og dermed vil det ikke gå strøm til resten av systemene i dronen. Om pinnen blir satt høy vil MOSFET-en bli påskrudd, og strømmen til resten av dronen vil da skrus på.

For å gjøre dette må pinne 27 bli satt aktiv og gjort klar til å brukes. Dette gjøres ved å skrive 27 til filen ”/sys/class/gpio/export”. Når det blir gjort vil mappen ”/sys/class/gpio/gpio27” opprettes. Inne i mappen vil det også opprettes flere filer. Filene som gjør at pinnen kan bli styrt til det ønskede formålet er ”direction” og ”value”. Den første filen, ”direction”, blir laget for at systemet skal vite om pinnen skal være en utgang eller inngang, da pinnen skal styre et signal vil den i dette tilfellet settes som en utgang. ”Value”-filen sjekker en boolsk variabel. Om variabelen er 0 vil utgangen være lav, og om variabelen er 1 vil utgangen være høy. Da pinne 27 er gjort klar og satt som en utgang, kan kretsen den er koblet til bli satt høy og lav i henhold til verdien i ”value” filen. Dette kan bli gjort ved å kjøre kommandoene vist i Kodelisting 5.4.

Kodelisting 5.3: Hvordan sette pinne 27 høy og lav på en Raspberry Pi Compute Module 4.

```
# Gjor pin 27 klar til bruk
sudo sh -c "echo 27 > /sys/class/gpio/export"

# Sett pin 27 som en utgangspinne
sudo sh -c "echo 'out' > /sys/class/gpio/gpio27/direction"

# Sett Pin 27 hoy
sudo sh -c "echo '1' > /sys/class/gpio/gpio27/value"

# Sett pin 27 lav
sudo sh -c "echo '0' > /sys/class/gpio/gpio27/value"
```

Grunnen til at `sudo sh -c` blir lagt til før `echo '<tekst>' > <ønsket fil>`, er fordi filene ligger i mapper som kun super-brukeren i Ubuntu har tilgang til. Ved skriving til filer ved bruk av `echo` kommandoen er det en bug i Ubuntu som ikke lar brukeren skrive `sudo echo`. Derfor blir dette arbeidet gjort ved å kjøre kommandoen `sh -c "<tekst>"` med sudo-rettigheter. Denne kommandoen oppretter og kjører et midlertidig shell-skript med innhold `<tekst>`.

Automatisk oppstart

For å ha et automatisk system er det essensielt at Raspberry Pi Compute Module 4 kan starte og kjøre ønskede program ved oppstart. I Ubuntu kan dette gjøres ved å skrive kommandoer i ”.bashrc” filen. Hver gang en ny terminal blir åpnet i Ubuntu blir kommandoene i den nevnte filen kjørt. Det kan for eksempel legges inn en kommando i denne filen som kjører et Python-skript. Skriptet vil da bli kjørt ved oppstart av ny terminal. Da det ikke er ønsket at skriptet skal kjøres hver gang en ny terminal åpnes, kan det legges til en if-setning som gjør at skriptet kun kjøres ved oppstart. En slik if-setning i bash-skript ser slik ut:

Kodelisting 5.4: If-setning i .bashrc som kun blir kjørt om terminalen er tty0.

```
if [[ "$(tty)" == "/dev/tty0" ]]; then
    python3 /<vei_til_python_skript>/<python_skript>.py &
fi
```

For at python-skriptet skal bli kjørt automatisk ved oppstart, må det da først logges inn på tty0. Dette kan bli gjort ved å følge instruksjonene vist i appendiks C.4.1.

5.4.3 Nødvendig programvare

Det må installeres nødvendig programvare på enhetene for at systemet skal fungere som planlagt. Det er ønskelig å benytte den nyeste versjonen av ROS, som ifølge dokumentasjonen er ROS 2 Foxy Fitzroy [69]. For å laste ned ROS 2 ble installasjons-veilederen for Debian-pakker som ligger i dokumentasjonen fulgt [70]. Etter installasjon bekreftes det at den var vellykket ved å kjøre kommandoen `source /opt/ros/foxy/setup.bash`, og deretter kommandoen `ros2 topic list`. Fraværet av feilmelinger tyder på en vellykket installasjon.

Siden denne installasjonsprosessen skal gjentas flere ganger, er det laget et shell-script som gjennomfører installasjonen automatisk. Installasjonsprosessen ligger i appendiks C.4.2.

Kapittel 6

Drone

6.1 Elektronikk og komponenter

Ettersom det er tatt utgangspunkt i dronen som er utviklet i tidligere oppgaver, er mange av komponentene de samme for denne versjonen av dronen. De største endringene er implementasjon av autopiloten og ettkortsdatamaskinen med følgene dette har.

6.1.1 Pixhawk 4 mini/Autopilot

Det er valgt å benytte autopilot for enklere flyging av drone til testingen av det automatiske utskytningssystemet for dronesverm. Det er valgt å bruke Pixhawk 4 mini, som er en autopilot fra Pixhawk programmet som er laget for å kjøre PX4-Autopilot-programvaren. (Teori om PX4-Autopilot ligger i 2.2.4). Denne autopiloten ble valgt da det internt på universitetet allerede var kjennskap til denne type autopilot. I tillegg møter kontrolleren en rekke av kravene som er stilt i kapittel 3.1. Pixhawk 4 mini har åpen kildekode på all sin programvare. Meldingssystemet ligner på ROS sitt system som gjør det enkelt for disse å kommunisere. Kontrolleren kjører på et sanntidsoperativsystem, som er gunstig for det overordnede prosjektet.

Autopiloten har portene som er nødvendige for å bli styrt av og kommunisere med en ekstern datamaskin gjennom en UART-kobling. I tillegg har den en rekke andre funksjoner som er behjelpeelige ved testing av det automatiske utskytningssystemet for dronesverm. Se kapittel 2.1.1.



Figur 6.1: GPS modul og Pixhawk 4 mini [71].

6.1.2 Motorer

I tidligere oppgaver er det gjort beregninger på strøm, pådrag og motortype for den type dronekropp som benyttes i dronesverm-prosjektet. Det er ikke sett videre på dette, da arbeidet som er gjort tidligere er tilsynelatende hensiktsmessig og grundig. Designet av dronekroppen for dette prosjektet er gjort med utgangspunkt i det forrige designet, slik at samme motorer trygt kan benyttes [3].

Motoren er av typen Emax RS1106II, med 4500 KV spesifikasjon. Denne spesifikasjonen forteller at motoren kan rotere 4500 ganger i minuttet ved påsatt 1 V spenning uten last. Regnestykket er lineært slik at med 7.4 V spenning som motorregulatorene i dette prosjektet, se kapittel (6.1.3), blir regnetstykket følgende: $4500KV * 7.4V = 33300rpm$. Det vil si at motorene kan rotere 33 300 ganger i minuttet uten last.



Figur 6.2: EMAX RS 1106 4500KV motor [72]

6.1.3 Motorregulator

I den første oppgaven om modulær drone er det gjort beregninger på, og valgt motorregulator. Det er derfor ikke sett på alternative regulatorer i denne rapporten, da den fra tidligere ser ut til å fungere som forventet [3].

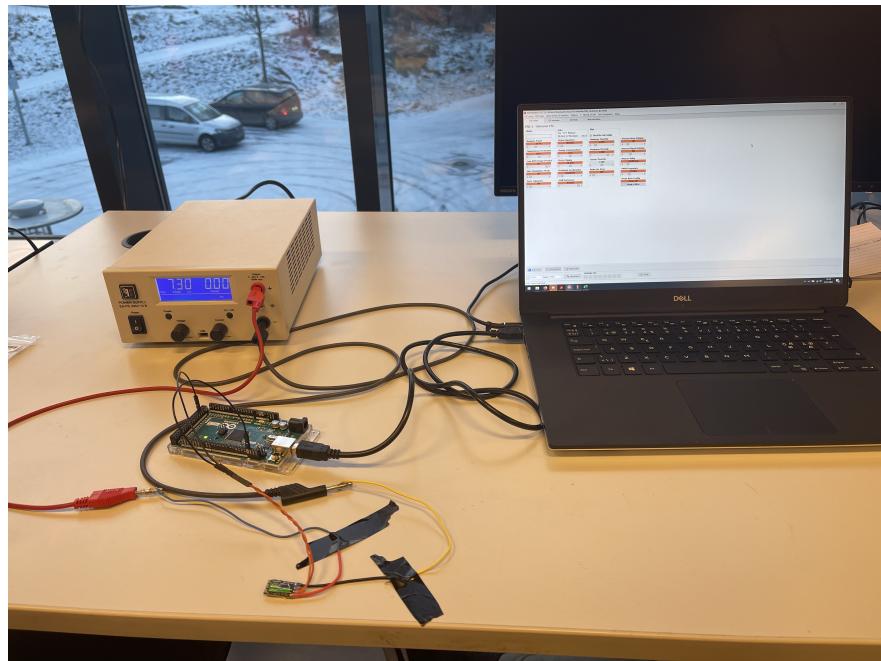


Figur 6.3: Electronic Speed Controller brukt i dronen.

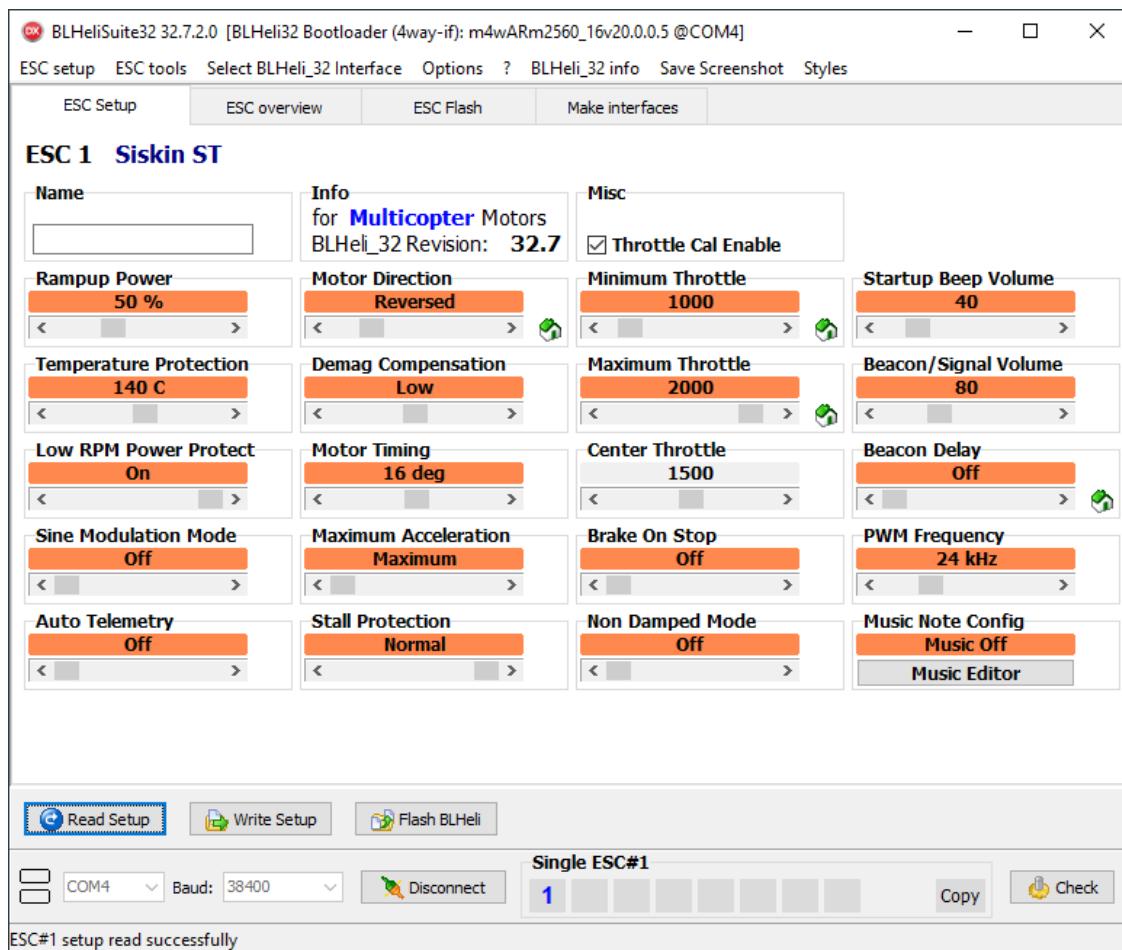
For programmeringen av motorregulatorene ble det benyttet en Arduino Mega2560 flashet med programvare fra BLHeliSuite32. Programmeringen ble gjennomført på følgende måte:

- Regulatorene ble tilført spenning fra en strømforsyning (se Figur 6.4) ut ifra hva som ble anbefalt i databladet.
- Signal-ledningen ble koblet til pinne 51 på arduinoen og jord ble koblet til jord-pinnen.
- Arduinoen ble koblet til en datamaskin med BLHeliSuite32-programvaren.

Fra programvaren kan konfigurasjonen til motorregulatorene leses av og endres. Det ble tatt utgangspunkt i konfigurasjonen fra oppgaven skrevet i 2019 om modulær drone med enkelte modifikasjoner [4]. Da motorregulator CAN-kontrolleren ikke er benyttet, er det ikke nødvendig å justere maksimal og minimalt pådrag. To av regulatorene ble konfigurerert med motsatt motor-retning i henhold til autopilot-oppsettet. I tillegg ble 'Beacon Delay' skrudd av for alle regulatorene (Figur 6.5).



Figur 6.4: Oppsett for ESC programmering.



Figur 6.5: ESC konfigurasjon.

6.1.4 LiPo Batteri

Batteriet som skal tilføre strøm til alle komponentene er samme type batteri som brukt i tidligere oppgaver. De tidligere oppgavene har allerede gjort beregninger på denne typen batteri sammen med motorerene og motorregulatorene. I tillegg har universitetet mange på lager, som er en fordel når det kommer til testing. Batteriet som benyttes er et Turnigy LiPo-batteri og har spesifikasjonene 2 s, 1600 mAh og 20-30 C. 2 s står for at batteriet består av to celler. MAh forteller hvor stor kapasitet batteriet har og kan levere per time. Dette batteriet kan tilføre 1600 mAh eller 1.6 A i én time når det er fulladet. Beregningen er som følger: C-tallet multiplisert med kapasiteten i ampere. I dette tilfellet blir det $30 \times 1,6A = 48A$.

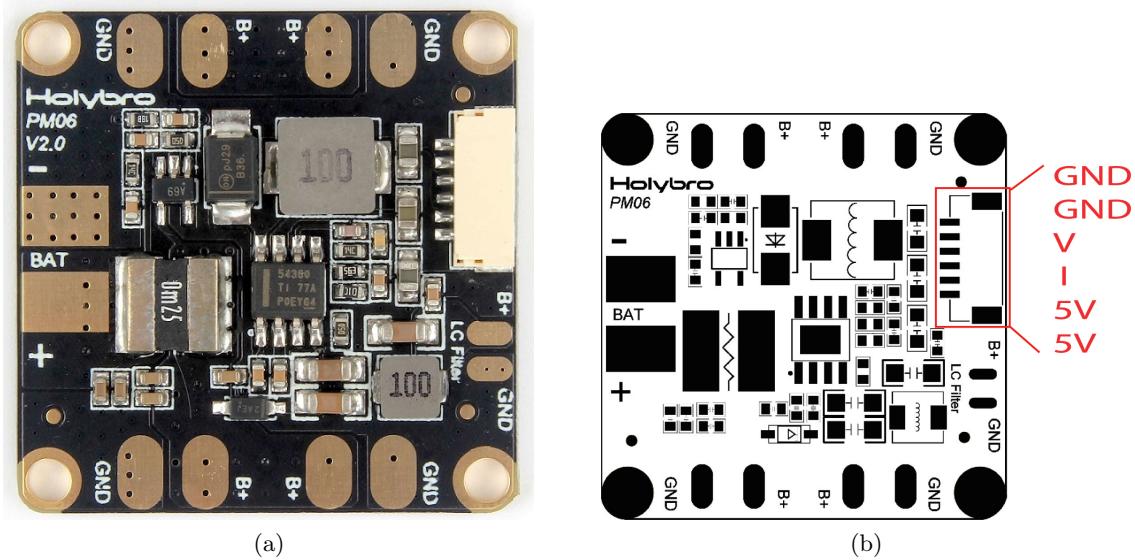
Batteriet har to konnektorer, én for utladning og én for lading. Konnektoren for utladning er en Molex Mini-Latch (noen steder refert til som Losi mini), og har to pinner: Pluss og minus. Konnektoren for lading er en JST-XH, og har tre pinner. Denne konnektoren kalles balanseringsplugg da den er laget slik at cellene i LiPo-batteriet skal lades opp på riktig måte.



Figur 6.6: LiPo-batteri brukt i dronen.

6.1.5 Strømmodul

For å fordele strømmen fra batteriet ut til de ulike komponentene kreves det en strømmodul. Strømmodulen fra tidligere oppgaver er ikke videreført, ettersom det med Pixhawk 4 mini følger med en passende modul. Denne strømmodulen er koblet rett til batteriet og har respektive utganger til motorregulatorene og Pixhawk [73]. Siden strømmodulen er tilpasset maskinvaren til Pixhawk 4 mini ble det vurdert at det var det beste alternativet.



Figur 6.7: Strømmodul [73].

6.2 Konstruksjon

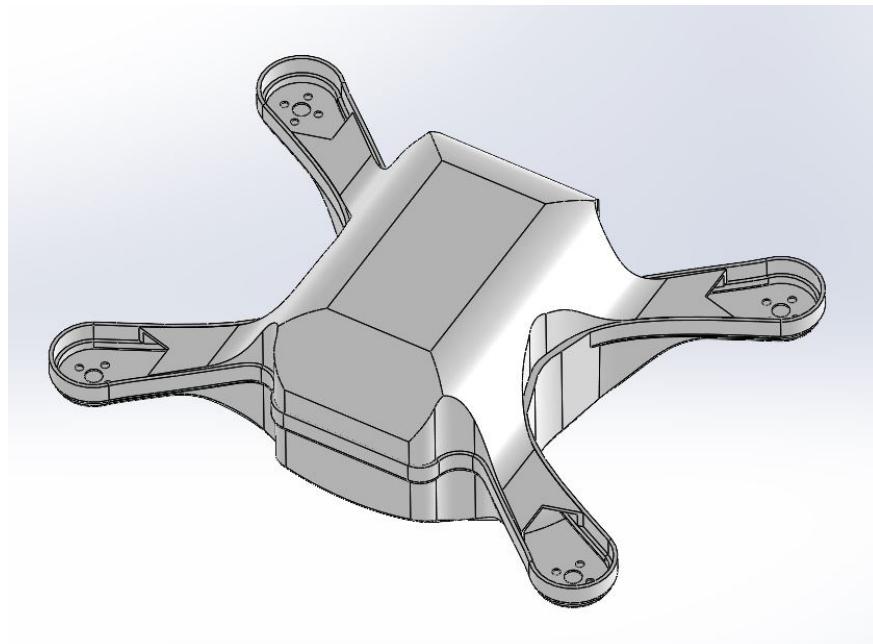
Ut ifra konseptvalget for dronen gjort i kapittel 3.4.3, ble det bestemt å inkludere autopilot i elektronikkpakken. Det resulterte i at det tidligere dronedesignet ikke fungere på grunn av store endringer. Derfor ble det nødvendig med et nytt innvendig oppsett med nok plass til test-komponentene. Det ble derfor bestemt å redesigne dronens skrog og lokk, samt et nytt oppsett og plassering for komponentene. Et ønske var å kunne beholde omtrent samme bortimot likt ytterdesign for å ikke forandre allerede eksisterende demonteringsløsning mellom skrog og lokk. Det nye dronedesignet er dermed en videreføring og baserer seg på forrige gruppens tidligere filer. Dette er gjort for å unngå å måtte starte på nytt, noe som ville vært tidkrevende.

6.2.1 Oppbygning

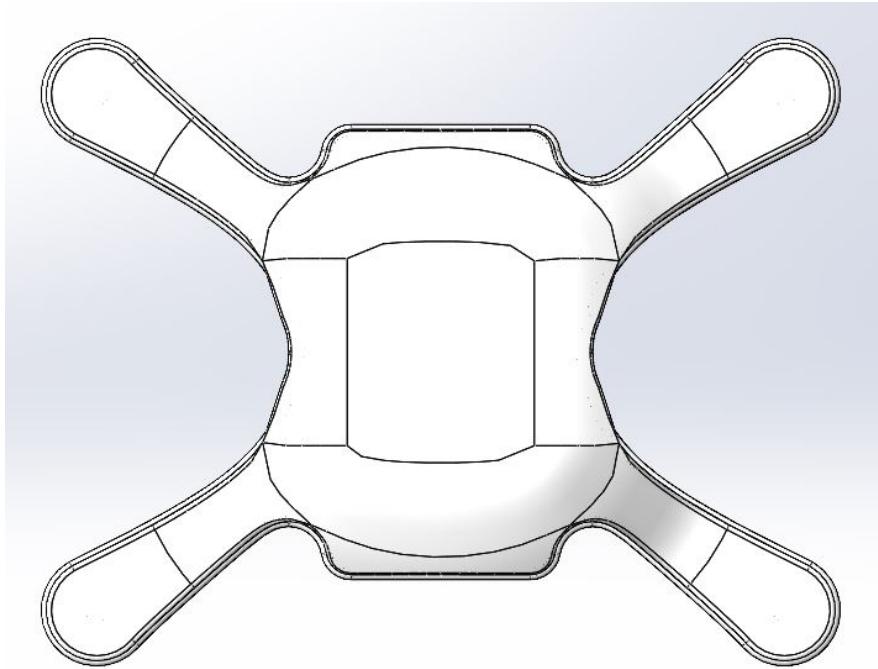
Skrog

Det oppgraderte skrogdesignet har flere likhetstrekk sammenlignet med forrige gruppe. Designet på mål, vinkler, armer og hovedutforming er tilsvarende likt fra tidligere, men det er gjort enkelte endringer, se Figur 6.8. Designet på dronen har fått et enklere og mer glatt design. Undersiden av skroget har blitt mer ovalt, se Figur 6.9, som skaper større plass til elektronikken i dronen. Armene, der motorene sitter, har blitt gjort bredere på hver side for å gi bedre plass til motorregulatorene. Det har også blitt designet fire stolper som er basen til innfestning til komponentene. Veggene på langsiden av skroget er også blitt rundet ut for å passe med føringstolpene i Utskytningsenheten, se Figur . Lokket til dronen er designet for å passe med skroget og gi nok plass til elektronikken.

Siden designet til dronen er forholdsvis lik forrige gruppens design, er det derfor valgt å ikke gjøre en FEM-analyse.



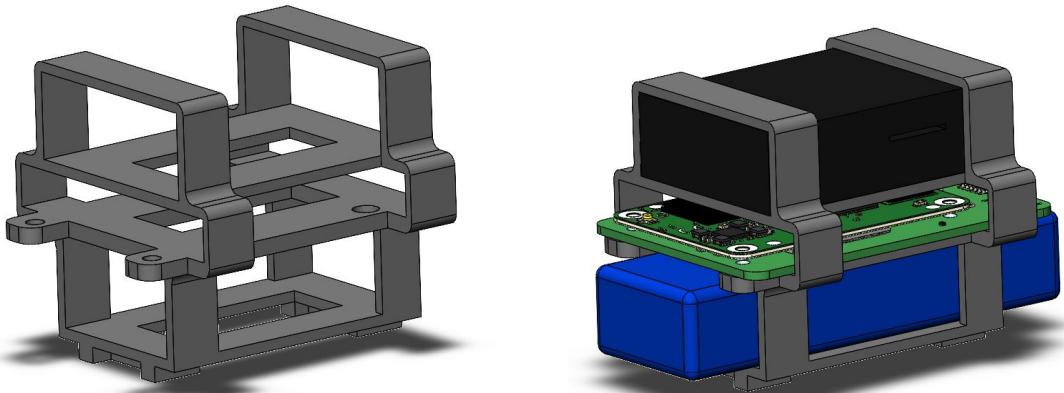
Figur 6.8: Sammenstilling i Solidworks av droneskrog.



Figur 6.9: Underside av skrog i Solidworks.

Innvendig oppsett

Med flere nye komponenter lagt til i elektronikkpakken, måtte det kommes opp med en festeløsning til komponentene. Før design kunne starte, ble det satt opp en plan til plassering av komponentene. Det ble valgt å fokusere på Pixhawk 4 mini og LiPo-batteri, begge er store komponenter med hver sine krav til posisjon. Batteriet er den største komponenten, både størrelsesmessig og vektmessig, noe som har mye å si for dronens tyngepunkt. Pixhawk 4 mini bør ligge horisontalt med motorene i dronen for best mulig posisjonsstyring. Med dette ble det bestemt å gå for en vertikal oppsettsløsning med batteriet sentrert i midten av dronen, og Pixhawk på toppen.



(a) Innvendig oppsett u/komponenter.

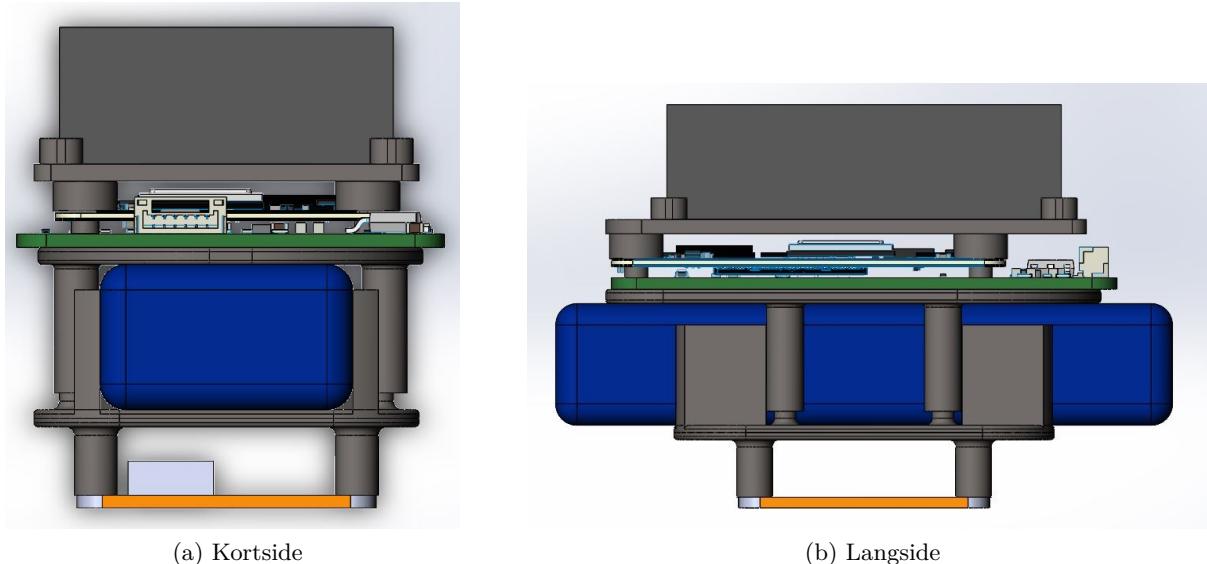
(b) Innvendig oppsett m/komponenter.

Figur 6.10: Første designløsning for innvendig oppsett i drone.

I løpet av prosjektet ble det satt opp flere designforslag. Første design vist i Figur 6.10 var en hylleløsning. Den bestod av én del hvor alle komponenter ble plassert inn i hver sin hylle. Tanken rundt dette var at det skulle være mulig å løfte ut alle komponenter i og jobbes med utenfor droneskroget. Alle komponentene ble satt inn ved og presses inn, utenom RPi CM4 som ble skrudd på plass. Etter mye inn og ut med komponentene ble det oppdaget at hver hylle ble slitt, noe som gjorde at komponentene ikke ble låst fast ordentlig lenger. Det oppstod også et problem med å

feste innfestningen i dronen. Uten komponenter og kabler i dronen/innfestningen var det enkelt å plassere hylleløsningen i dronen. Derimot var det vanskeligere når fullt av kabler og konnektorer var i veien. Dette gjorde det utfordrende å sette alt inn i dronen.

I Figur 6.11 (a) og (b) vises siste resultat av oppsettet til komponentene. Oppsetts rekkefølgen og plasseringen av komponentene i forhold til hverandre er fremdeles lik sammenlignet med Figur 6.10, men splittet opp i tre deler for bedre montering. Den nevnte løsningen er satt opp rundt at hver komponent har sin egen plattform. Plattformene har en sporløsning med sylinder på undersiden som tres på stolpene. Dette er gjort for alle plattformene til komponentene.



Figur 6.11: Andre designløsning for innvendig oppsett i drone.

Tekniske tegninger av droneskrog og innvendig oppsett ligger i appendiks H.1.

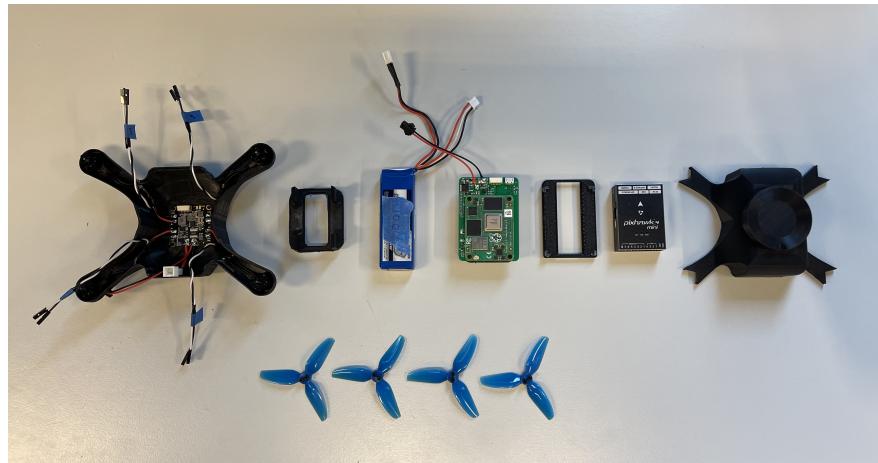
6.2.2 Produksjon

Dronen består av fem deler: skrogbunn, skrogtopp, innfestning til batteriet, innfestning til RPi CM4 og innfestning til Pixhawk 4 mini. Med at den nye dronen er en prototype, har alle deler blitt 3D-printet på universitetet. Materialet brukt for delene er Polyactic acid (PLA), en type bioplast som er mye brukt til 3D-printing. Med flere 3D-printere lett tilgjengelig, var det enkelt å printe ut flere iterasjoner av skrog, og innfestninger for å sjekke at alt passet fysisk.

6.2.3 Montering

Med komponenter valgt, innfestingmåte bestemt og konstruksjonen av dronen ferdig designet ble en drone montert.

Monteringsprosessen av en drone ligger i appendiks A.1.1.



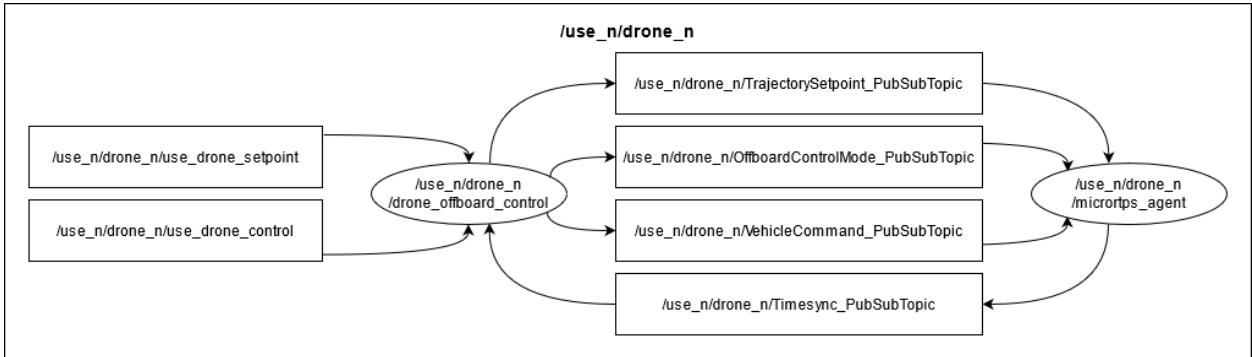
Figur 6.12: Demonert drone uten bruk av verktøy.

6.3 Programvare

Programvaren i dronen kan hovedsakelig deles til å inngå i to systemer; Pixhawk 4 mini og Raspberry Pi Compute Module 4. Hvor Pixhawk 4 mini sin maskinvare (PX4-Autopilot) står for selve flygingen av dronen, og RPi CM4 kontrollerer dronen. Disse skal kommunisere gjennom en PX4/ROS 2-bro.

6.3.1 Design

Ved design av programvaren til dronen ble det tatt utgangspunkt i den overordnede systemarkitekturen. Det var essensielt at koden som ble skrevet til dronen var dynamisk slik at lik programvare kunne bli brukt på dronene. For å gjøre dette i ROS 2 ble navnerom brukt. En node er laget, den har som formål å stå for kontroll av dronen. Denne noden sender og mottar all nødvendig informasjon til/fra Pixhawk 4 mini gjennom PX4/ROS 2-broen.



Figur 6.13: Graf over emner og noder i drone n.

For å ha kontroll over dronen fra en ekstern datamaskin må den settes i utenbordskontroll, det måtte derfor sørges for å kunne endre modusene til dronen. Ved oppstart starter microRTPS agenten flere emner, blant annet `VehicleCommand_PubSubTopic`. Dette emnet ble brukt til å endre modusene på dronen, samt styre andre essensielle funksjoner som armer og disarmer. Dronen krever en konstant strøm av settpunkt med en rate på mer enn 2 Hz, for å kunne bli satt i utenbordskontroll. For å sende settpunkt-meldinger ble emnet `TrajectorySetpoint_PubSubTopic` brukt. Dette emnet inneholder alle nødvendige variabler for å styre dronene med enten hastighets- eller posisjonskontroll, som var nok til å bevise at det automatiske utskytningsystemet for dronesverm fungerer. Siden PX4-Autopilot kjører på sanntidsoperativsystemet NuttX, er det svært strenge krav til tidssynkronisering på ulike prosesser og meldinger. Tidsvariablene i PX4 programvaren måtte derfor hentes ut, og legges til i meldingene som ble publisert. For å hente ut tidsvariablene må noden i dronen abonnere på `Timesync_PubSubTopic`. Valget om å styre dronen med posisjons- eller hastighetskontroll foregår i emnet `OffboardControlMode_PubSubTopic`. I noden ble de boolske variablene som tilhører emnet og som styrer posisjons- eller hastighetskontroll satt sanne, da det er ønskelig at begge kan brukes.

Dronen styres ved å sende kommandoer til dronen gjennom en utskytningsenhet som abонerer på emner bakkestasjonen publiserer. Meldingene som blir publisert fra bakkestasjonen inneholder derfor tilstrekkelig informasjon til å publisere alle de nødvendige meldingene fra RPi CM4 til PX4. Derfor ble emnene `use_drone_control` og `use_drone_setpoint` brukt som planlagt.

6.3.2 ROS 2 node

Det er en rekke krav til informasjonen PX4 må motta fra en ekstern datamaskin for å bli satt i, og kontrollert utenbordskontroll. Noden i dronen som sender informasjonen til PX4 blir skrevet spesifikt for å oppnå kravene, og styre dronen.

De første linjene i koden som ble skrevet importerer de nødvendige meldingstypene som blir brukt av emnene. En egendefinert meldings-pakke (`ds_msgs`) er laget for å skille meldingene fra standard-meldingene som ligger i `px4_msgs` pakken. Standardbibliotekene som ROS 2 bruker, og en Python pakke som gjør det mulig å skrive til systemterminalen blir også importert.

Deretter defineres klassen, og noden instansieres. Publiseringene lages så med emne- og meldings parameterene satt i henhold til hverandre. Maks antall meldinger i meldingskøen blir satt til 10. Etterfølgende av publiseringene instansieres abonnentene med funksjonene som skal kalles ved mottatt melding, se Kodelisting 6.1.

Kodelisting 6.1: Publiseringene og abonnentene lages.

```
20     # Creating publishers
21     self.offboard_mode_publisher_ = self.create_publisher(
22         OffboardControlMode, "OffboardControlMode_PubSubTopic", 10)
23     self.trajectory_setpoint_publisher_ = self.create_publisher(
24         TrajectorySetpoint, "TrajectorySetpoint_PubSubTopic", 10)
25     self.vehicle_command_publisher_ = self.create_publisher(VehicleCommand, "
26         VehicleCommand_PubSubTopic", 10)

27     # Creating subscribers
28     self.timesync_subscriber_ = self.create_subscription(Timesync, "
29         Timesync_PubSubTopic", self.timesync, 10)
30     self.use_drone_setpoint_subscriber_ = self.create_subscription(
31         TrajectorySetpointDS, "use_drone_setpoint", self.fetch_trajectory_setpoint, 10)
32     self.control_subscriber_ = self.create_subscription(DroneControl, "
33         use_drone_control", self.drone_control, 10)
```

Før noden begynner å spinne med en rate på 20 Hz startes og nullstilles settpunkt-meldingen som publiseres til microRTPS agenten, opprettes det en lokal tidsvariabel, og andre lokale flagg som brukes til å kontrollere dronen. Alle flaggene bortsett fra `land_` blir satt lave ved oppstart av noden.

Kodelisting 6.2: Medlemsvariablene i dronen.

```
30     # Member trajectory setpoint message
31     self.trajectory_msg_ = TrajectorySetpoint()

32     # Member variables
33     self.timestamp_ = 0

34     # Flags
35     self.arm_ = False
36     self.land_ = True
37     self.launch_ = False
38     self.armed_ = False
39     self.switch_px_ = False
40     self.px_status_ = False

41     # Timer running at 20 Hz
42     timer_period = 0.05
43     self.timer = self.create_timer(timer_period, self.timer_callback)
```

Noen av de lokale flaggene blir endret av abonnenten `control_subscriber_`, variablen `arm_` er en av disse. I nodens spinnende funksjon vil dronen bli armert om variabelen blir satt høy samtidig som `armed_` er satt lav (se Kodelisting 6.3).

Kodelisting 6.3: If setning som armerer dronen.

```

50     # This arms the drone
51     if self.arm_ == True and self.armed_ == False:
52         self.arm_vehicle()
53         self.armed_ = True

```

Om variablene i if-setningen er korrekt i henhold til det som er satt, vil funksjonen `arm_vehicle()` kjøres. Denne funksjonen setter parameterene i variablen `VEHICLE_CMD_COMPONENT_ARM_DISARM` til `(1.0, 0.0)`, og publiserer samtidig meldingen til `VehicleCommand_PubSubTopic`. I if-setningen blir også den lokale variablen `armed_` satt høy. Det gjør at noden vet at dronen har blitt armert, og if-setningen vil derfor kun kjøres én gang.

For å disarmere dronen må meldingsvariablen `arm_` bli satt lav, og den lokale variablen `armed_` må være høy (se Kodelisting 6.4). I if-setningen blir funksjonen `disarm_vehicle()` kjørt. Denne funksjonen setter, og publiserer parameterene `(0.0, 0.0)` til meldingsvariablen `VEHICLE_CMD_COMPONENT_ARM_DISARM`. Neste funksjon som blir kjørt, `set_stabilized_mode()`, setter dronen i stabiliserings modus. Det blir gjort ved at parameterene `(1, 2)` blir satt i meldingsvariablen `VEHICLE_CMD_DO_SET_MODE` i emnet `VehicleCommand_PubSubTopic`. Deretter blir flaggene `armed_` og `launch_` satt lave. På lik måte som i if-setningen for å armere, blir `armed_` satt lav for at setningen kun skal kjøres en gang. Variablen `launch_` blir satt lav da den brukes til å lette dronen, og det åpenbart ikke er mulig for dronen å ta av når den er disarmert.

Kodelisting 6.4: If setning som disarmerer dronen.

```

56     # This disarms the drone
57     if self.arm_ == False and self.armed_ == True:
58         self.disarm_vehicle()
59         self.set_stabilized_mode()
60         self.armed_ = False
61         self.launch_ = False

```

Selv utskytningen av dronen skjer i funksjonen `set_offboard_mode()`. Den settes først parameterene `(1, 6)` på meldingsvariablen `VEHICLE_CMD_DO_SET_MODE`, og deretter publiserer dette til emnet `VehicleCommand_PubSubTopic`. For at dette kan skje må variablene `launch_` og `armed_` være samme.

Kodelisting 6.5: If setning som starter flyging.

```

61     # This launches the drone
62     if self.launch_ == True and self.armed_ == True:
63         self.set_offboard_mode()
64
65         self.publish_offboard_control_mode()
66         self.trajectory_setpoint_publisher_.publish(self.trajectory_msg_)

```

PX4-Autopilot krever en konstant strøm av settpunkt, og hvilken type utenbordskontroll den skal bli styrt av. Disse blir publisert i `publish_offboard_control_mode()` og `trajectory_setpoint_publisher_.publish(self.trajectory_msg_)`. Førstnevnte publiserer til `OffboardControlMode_PubSubTopic` emnet. I funksjonen er alle meldings-variablene bortsett fra tidsvariabelen satt konstant da posisjons- eller hastighetskontroll uansett skal brukes. (Se Kodelisting 6.5 for if-setningen og appendiks C.2.1 for variablene som blir satt).

`Trajectory_setpoint_` publisereren har som formål å kun videresende meldingene fra `use_drone_setpoint_` abonnementen. Derfor er meldingstypen den bruker satt til at når `use_drone_setpoint_` mottar en melding, skal medlemsvariablene i `trajectory_msg_` umiddelbart settes lik de mottatte meldingene fra utskytningsenheten. Dette blir gjort i funksjonen `fetch_trajectory_setpoint`, som kjører hver gang noden mottar en melding fra emnet `use_drone_setpoint` (se Kodelisting 6.6). Hver variabel må settes lik den andre like variabelen da emnet `TrajectorySetpointDS` fra `ds_ros2_msgs` har én variabel mer en `px4_msgs`. I Kodelisting 6.5 er publisereren av settpunkter og utenbordskontroll modus skrevet direkte inn i `timer_callback` funksjonen, og blir dermed sent konstant med en rate på 20 Hz.

Kodelisting 6.6: Hent og sett settpunkter fra utskytningsenhet til drone.

```

87 # Fetch setpoints
88 def fetch_trajectory_setpoint(self, use_msg):
89     self.trajectory_msg_.x = use_msg.x
90     self.trajectory_msg_.y = use_msg.y
91     self.trajectory_msg_.z = use_msg.z
92     self.trajectory_msg_.yaw = use_msg.yaw
93     self.trajectory_msg_.vx = use_msg.vx
94     self.trajectory_msg_.vy = use_msg.vy
95     self.trajectory_msg_.vz = use_msg.vz
96     self.trajectory_msg_.acceleration = use_msg.acceleration
97     self.trajectory_msg_.jerk = use_msg.jerk
98     self.trajectory_msg_.thrust = use_msg.thrust

```

For å lande dronen blir flagget `land_` i meldingen `drone_control` satt høy. Når den blir høy vil dronens z-punkt i koordinatsystemet bli satt til `NaN`. Siden z-punktet blir satt til `NaN` vil hastighetsstyringen ta over kontrollen av dronen. Variabelen `vz` blir derfor satt til 0.2 som gjør at dronen flyr nedover med en hastighet på 0,2 m/s.

Kodelisting 6.7: If setning som starter landing av dronen.

```

67 # This lands the drone
68 if self.land_ == True:
69     self.trajectory_msg_.z = float("NaN")
70     self.trajectory_msg_.vz = 0.2

```

De to siste if-setningene i `timer_callback` funksjonen gjør det mulig å skru Pixhawk 4 mini av eller på. Det blir gjort ved å skrive kommandoene som er nevnt i kapittel 5.2 til systemterminalen. Samme feilsjekk er innlagt i if-setningene i Kodelisting 6.8 som i armer og disarmer funksjonene. Altså sørger feilsjekken for at if-setningene kun blir kjørt én gang slik at den ikke prøver å sette pinne 27 høy når den allerede er høy.

Kodelisting 6.8: Skru av og på Pixhawk 4 mini.

```

72 # This switches the pixhawk
73 if self.switch_px_ == True and self.px_status_ == False:
74     os.system("sudo sh -c 'echo '1' > /sys/class/gpio/gpio27/value'")
75     self.px_status_ = True
76 elif self.switch_px_ == False and self.px_status_ == True:
77     os.system("sudo sh -c 'echo '0' > /sys/class/gpio/gpio27/value'")
78     self.px_status_ = False

```

Timesync abonnementen har nokså lik funksjon som setpoint abonnementen. Forskjellen er at den tar imot tidsvariablen til PX4 og lagrer den som en lokal tidsvariabel. Den lokale tidsvariabelen blir så lagt ved i alle publiseringene som blir sendt tilbake.

6.3.3 Implementasjon

Det ble bestemt å bruke UART tilkobling mellom PX4-Autopilot og Raspberry Pi Compute Module 4. MicroRTPS blir da brukt. For at microRTPS skal fungere må én klient og én agent bli startet på hver sin side av tilkoblingen.

Kommunikasjon mellom enhetene

I den opprinnelige programvaren som Pixhawk 4 mini kommer med er ikke microRTPS installert, og en ny fastvare måtte derfor installeres. Den nye fastvaren ble installert ved å laste ned kildekoden til PX4-Autopilot (PX4) fra GitHub. I kildekoden er det flere valg til fastvarer som kan bygges. Fastvaren som ble bygd for å ha microRTPS tilgjengelig på PX4-Autopilot var `px4_fmu-v5_rtps`. For å bygge og installere fastvaren ble instruksjoner fra dokumentasjonen fulgt [40].

For å bruke microRTPS på ettkortsdatamaskinen ble ROS 2 pakkene `px4_ros_com` og `px4_msgs` lastet ned i et ROS 2 arbeidsområde. Pakkene har flere avhengigheter som trengs før de kan bygges, disse ble derfor lastet ned før byggingen ble satt i gang. Se kapittel 2.2.3 for hvordan arbeidsområder lages og pakker bygges. Selv om avhengighetene var lastet ned, feilet byggingen av pakkene på RPi CM4. For å finne feilen ble argumentet `--event-handlers console_direct+` lagt til i kommandoene som starter byggingen. Event handlers argumentet gjør at hvert steg av byggeprosessen blir skrevet ut i terminalen. Ut ifra dette viste det seg at pakkene var så tunge for prosessoren å bygge at den begynte å henge seg opp. Dette ble fikset ved å legge til argumentet `--executor sequential`, som gjør at byggingen kun skal foregå med én prosess av gangen. I tillegg til det siste argumentet ble det satt av en ledig kjerne som datamaskinen vil ha tilgjengelig til å gjennomføre andre essensielle prosesser ved siden av byggingen. Dette ble gjort med å kjøre kommandoene vist i Kodelisting 6.9.

Kodelisting 6.9: Kommandoer brukt for å sette av en kjerne under bygging av ROS 2 pakker.

```
$ cpu = $(nproc)
$ cpu_in_use = '$(($cpu-1))'
$ export MAKEFLAGS = '-j $cpu_in_use'
```

Pakkene ble da bygd, og microRTPS kunne kjøres fra Raspberry Pi Compute Module 4.

Når både Pixhawk 4 mini og Raspberry Pi Compute Module 4 har microRTPS installert, kan klienten og agenten henholdsvis bli startet. På klienten er porten som microRTPS leser og skriver til standard satt til `/ttyAMA0`. Derfor ble argumentet `-d /dev/ttyS3` lagt til. Et annet argument som ble lagt til er `-b 921600`. Det argumentet ble lagt til for å sette en høy baud-rate slik at meldingene kan sendes og mottas raskt. Settpunkt-meldingen som blir sendt vil ha maksimalt 20 variabler, på enten 32 eller 64 bit. Om alle er 64 bit vil det si at $64\text{bit} * 20 = \underline{1280\text{bit}}$ blir sendt ved hver melding. Da det vil være flere meldinger som skal bli publisert ble det lagt til en sikkerhetsmargin på at 10 slike meldinger sendes. Det tilsvarer da at $1280\text{bits} * 10 = \underline{12800\text{bits}}$ sendes konstant. Med en baud-rate på 921 600 vil det si at meldingene kan sendes med en rate på $\frac{921600\text{bit/s}}{12800\text{bit}} = \underline{72\text{Hz}}$. Som er godt over den faktiske raten i noden på 20 Hz.

De samme argumentene ble lagt til da agenten ble startet på datamaskinen, men porten som skrives til og leses av ble satt til `/dev/ttyS0`. For å unngå en unødvendig feil som kan oppstå ble også argumentet `-t` UART lagt til på begge enhetene. Standard skal microRTPS klienten og agenten kommunisere over UART, men kan ved uheldige tilfeller bli satt til UDP.

For å verifisere at kommunikasjonen funket som forventet, ble den testet ved å kjøre en eksemplar ROS 2 node kalt `SensorCombinedListener` som ligger i `px4_ros_com` pakken. Den noden sender et utvalg av sensordata fra Pixhawk 4 mini til Raspberry Pi Compute Module 4.

Som sett i Figur 6.14 virket kommunikasjonen som planlagt. I oppsettet ble det for enkelhetsskyld brukt en Raspberry Pi 3 da Raspberry Pi Compute Module 4 ikke var tilgjengelig. På grunn av store likheter mellom de forskjellige generasjonene av ettkortsdatamaskinene fra Raspberry Pi gikk dette fint.



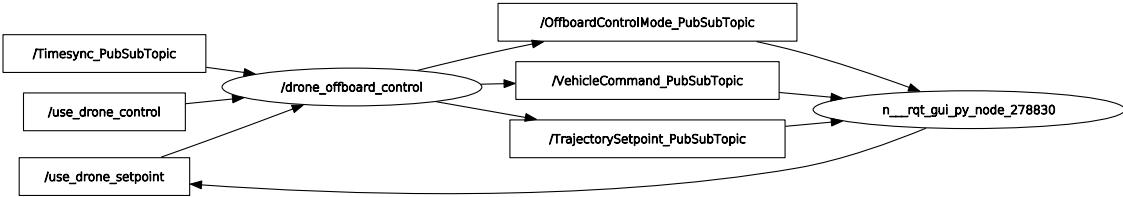
Figur 6.14: Oppsett for testing av kommunikasjon med sending av sensor data.

Verifisering av node

Når noden til dronen var ferdig skrevet, og kommunikasjonen mellom Raspberry Pi Compute Module 4 og PX4-Autopilot var kontrollert. Ble programvaren som er skrevet til noden testet for å verifisere at den fungerte som planlagt. Testingen av noden foregår ved bruk av verktøyene i RQt. RQt ble brukt til å monitorere emnene som blir publisert av noden, og dermed sjekke at korrekt

informasjon blir sendt til riktig emne. Verktøyet kan også brukes til å simulere bakkestasjonen og utskytningsenheten. Det vil si at RQt kan sette verdier til meldingsvariablene i emnene `use_drone_control` og `use_drone_setpoint`, og dermed ha kontroll over drone-noden.

Noden ble testet som eneste aktive node på ROS 2 nettverket. Det vil si at den abonnerte på emner uten informasjon fra andre noder, og publiserte heller ikke til andre noder. Som sett i Figur 6.15 ble RQt istedenfor satt til å abonnere og publisere til et utvalg av emnene.



Figur 6.15: Graf over emner og noder ved testing av drone-noden.

Da RQt var satt opp til å publisere og abonnere på emnene laget av `drone_offboard_control` noden. Ble verdier satt på meldingsvariablene i `use_drone_control` og `use_drone_setpoint` emnene for å sjekke at noden oppførte seg riktig. Samtlige kombinasjoner ble testet.

Navnerom

Navnerommet dronen vil ta i bruk er avhengig av hvilken utskytnignsenhet den tilhører, og hvilken drone i den gitte utskytningsenheten den faktisk er. For å starte en ROS 2 node med navnerom må argumentet `--ros-args -r __ns:=/<navnerom>` legges til ved oppstart av noden.

Dette blir et problem da microRTPS agenten startes uten navnerom, og derfor kun prøver å høre på emner uten navnerom. Det ble undersøkt hvordan microRTPS agenten kunne bli gitt et navnerom som gjorde at den kunne abonnere på, og publisere til de korrekte emnene. På tiden dette ble undersøkt virket det som om det ikke var mulig da dokumentasjonen ikke nevnte noe om dette. Derfor ble det først undersøkt å gå inn i koden til microRTPS agenten, og manuelt endre emnenavnene i kildekoden. Da kildekoden til microRTPS agenten ble analysert, ble det oppfattet at et argument kunne sette navnerom på agenten og emnene som ble opprettet. Tilsynelatende kunne dette gjøres ved å legge til argumentet `-n <navnerom>` ved oppstart av microRTPS agenten [74]. Se linje 131 i Kodelisting 6.10.

Kodelisting 6.10: Utdrag fra kildekoden til microRTPS agent [74].

```

110 static int parse_options(int argc, char **argv)
111 {
112     int ch;
113
114     while ((ch = getopt(argc, argv, "t:d:w:b:p:r:s:i:fhv:n:")) != EOF)
115     {
116         switch (ch)
117         {
118             case 't': _options.transport      = strcmp(optarg, "UDP"
119 ) == 0?
120                         options::eTransports::UDP
121                         : options::
122                         eTransports::UART;      break;
123             case 'd': if (nullptr != optarg) strcpy(_options.device,
124 optarg);      break;
125             case 'w': _options.sleep_us      = strtoul(optarg,
126 nullptr, 10);      break;
127             case 'b': _options.baudrate      = strtoul(optarg,
128 nullptr, 10);      break;
129             case 'p': _options.poll_ms       = strtoul(optarg,
130 nullptr, 10);      break;
131             case 'r': _options.recv_port      = strtoul(optarg,
132 nullptr, 10);      break;
133             case 's': _options.send_port      = strtoul(optarg,
134 nullptr, 10);      break;
135             case 'i': if (nullptr != optarg) strcpy(_options.ip,
136 optarg);      break;
137             case 'f': _options.sw_flow_control = true;
138                 break;
139             case 'h': _options.hw_flow_control = true;
140                 break;
141             case 'v': _options.verbose_debug = true;
142                 break;
143             case 'n': if (nullptr != optarg) _options.ns = std::
144 string(optarg) + "/"; break;
145             default:
146                 usage(argv[0]);
147                 return -1;
148         }
149     }
150 }
```

Da agenten ble startet med argumentet lagt til opprettet den emnene med det gitte navnerommet.

På grunn av at det nevnte argumentet ikke var nevnt i dokumentasjonen ble det sendt en forespørsel til utviklerne bak kildekoden om å endre dokumentasjonen. Forespørselen gikk gjennom, og gruppen er dermed bidragsytere til dokumentasjonen i PX4ⁱ.

ⁱhttps://github.com/PX4/PX4-user_guide/pull/1251

Automatisering

Siden det vil være mange droner i dronesvermen som bruker programvaren er det laget flere skript gjør installasjonsprosessen av de ulike programmene mer automatisert. Et annet skript som gjør at de nødvendige programmene vil bli kjørt ved oppstart av RPi CM4 er også laget.

For å automatisere installasjonsprosessen, laste ned avhengigheter og bygge arbeidsområdet, er det laget to shell-skript: "Fastdds_installation.sh" laster ned, og installerer FastDDS som microRTPS agenten og pakkene fra PX4 er avhengige av. "Ros2_px4_ws_builder.sh" laster ned, og bygger PX4- og drone-pakkene.

Kildekoden til shell-skriptene ligger i appendiks C.4.3 og C.4.4.

I tillegg til shell-skriptene er det laget et python-skript som starter alle nødvendige program ved oppstart av dronen. Skriptet starter med å åpne en tekstfil hvor det er skrevet hvilken utskytningsenhet dronen tilhører, og hvilken drone den er. Det er for å finne ut av hvilket navnerom emnene i dronen bruker. Altså om det står "use: 01" og drone: 01" i tekstfilen, vil navnrommet bli satt til /use_01/drone_01. Skriptet bruker også det gitte navnerommet ved oppstart av microRTPS agenten og noden. Python-skriptet er lagt til i ".bashrc"-filen som kjøres automatisk ved oppstart av datamaskinen.

Se appendiks C.2.2 for kildekoden til Python-skriptet.

Kapittel 7

Utskytningsenhet

Utskytningsenheten skal fungere som bindeleddet mellom bakkestasjonen og dronene. Fra bakkestasjonen vil den motta data til sine 10 droner, og videresende informasjonen til korrekt drone. Utskytningsenheten skal også kunne motta informasjon om dronenes status, og sende denne informasjonen tilbake til bakkestasjonen om ønskelig. I tillegg vil utskytningsenheten fungere som oppbevaring, og som utskytnings-plattform for dronene.

For å kunne lage en enhet med de nevnte funksjonalitetene må den inneholde ulik elektronikk og komponenter. I tillegg må den ha et design som er hensiktsmessig siden dronene skal oppbevares, og fly ut fra enheten. En programvare i henhold til systemarkitekturen i kapittel 4.1 må også lages.

7.1 Elektronikk og komponenter

Utskytningsenheten må utstyres med følgende komponenter, en datamaskin som kan kobles til et lokal Wi-Fi, og et batteri som gjør enheten slipper å være avhengig av nett-strøm.

7.1.1 Datamaskin i utskytningsenhet

Datamaskinen som skal være plassert i utskytningsenheten er, som skrevet i kapittel 5.2, en Raspberry Pi Compute Module 4. Til denne er det designet og produsert et bærekort.

7.1.2 Batteri

I utskytningsenheten er det også et LiPo-batteri identisk til det som sitter i dronen. RPi CM4 trekker maksimalt 3 A, det vil si at batteriet kan forsyne ettkortsdatamaskinen med strøm i rundt 30 minutter ved full kapasitet [58]. Beregningen er vist i utregning 7.1.

$$varighet = \frac{1.6Ah}{3A} = 0.5333h = 0.5333 * 60min = 32min \quad (7.1)$$

7.2 Konstruksjon

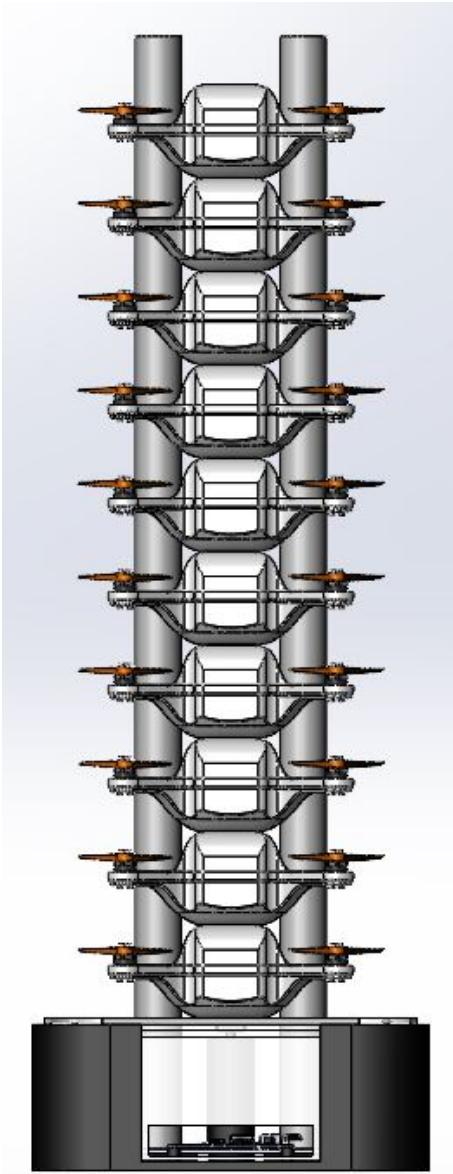
Utskytningsenheten er et nytt tilskudd i det overordnede dronesverm-prosjektet. Det er planlagt å lage en utskytingenhet som dronene skal kunne oppbevares i, og kunne fly ut av. Siden en slik konstruksjon ikke er laget tidligere, er det viktig å sette opp gitte funksjonaliteter, og funksjonaliteter utskytningsenheten skal kunne oppfylle.

Med tidsrammen gitt for prosjektet og avgjørelsen om at kun to droner skulle konstrueres, virket det ikke nødvendig å lage en fullskalert versjon av utskytningsenheten. Opprinnelig skal utskytningsenheten ha muligheten til å romme ti droner. Av den grunn ble det bestemt å lage en mindre versjon for to testdroner. Det viktige rundt utskytningsenheten er at designet skal kunne fungere like fint for ti droner, som det skal gjøre med to droner.

Det ble sett på flere ulike konsepter for hvordan dronene skulle oppbevares sammen innad i en enhet og ikke være i veien for hverandre under flyging. Om dronene skal fly ut én og én eller alle samtidig var også noe som ble tatt opp. Til slutt ble det blesluttet å stable dronene oppå hverandre vertikalt. Med dette, måtte det settes opp et rammeverkløsning til å holde dronene i posisjon inni utskytningsenheten til enhver tid og ha muligheten til å lette uten å være i veien for propellene.

Ettersom langsiden på skroget til dronen er buet ble det naturlig å sette opp sirkulære føringsstolper på hver side, se Figur 7.1. Dronens skrog ble tilpasset etter nevnt i kapittel 6.2.1. Det ble gjennomført manuelle tester for å se hvordan dronen mulig kunne eventuelt oppføre seg når på vei opp. Der ble det oppdaget at dronen kunne fremdeles bøye seg fram og tilbake, noe som resulterte i at propellene traff føringsstolpene. For å unngå dette, ble det satt opp føringsstolper på kortsidene for dronene. Denne gangen i rektangulær form for å passe skroget sine flate kortsider.

Med potensialet til å lage et kompakt design, måtte det også tas i betraktning at utskytningsenheten bør ha en enkel byggeprosess. Det vil si at utskytningsenheten bør være lett å sette sammen med få deler. Det ble bestemt at designet skulle bestå av en hovedkonstruksjon, hvor det er lagt inn festepunkter til resterende deler og elektronikk gjennom skruehull, gjengt messing-innsats eller ferdiglagde spor. Valgt løsning gjorde at oppsettet av utskytningsenheten ble et byggesett, noe som gjorde det viktig å ikke ha dimensjonsfeil i hovedkonstruksjonen før det ble sendt til produksjon.



Figur 7.1: Fullskalert utskytningsenhet med ti droner stablet i Solidworks.

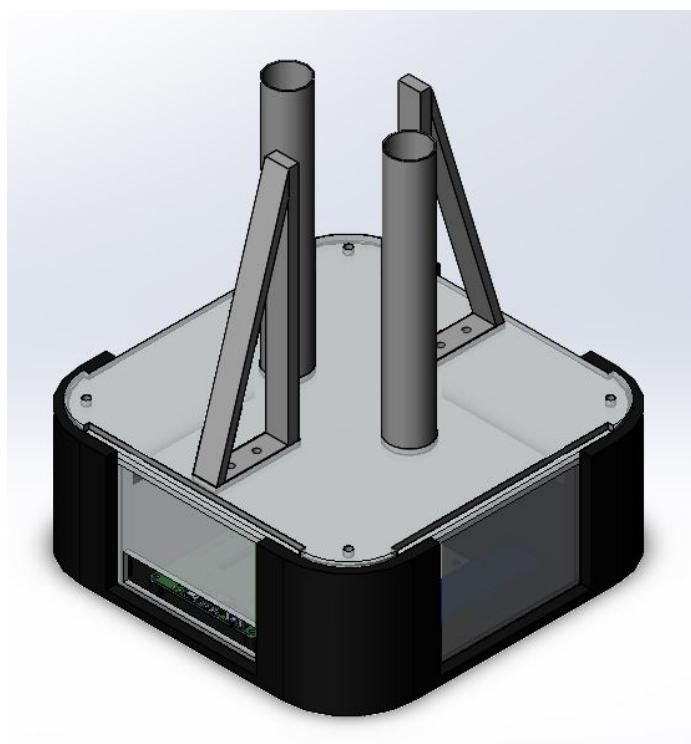
7.2.1 Oppbygning

Elektronikkboksen er hovedkonstruksjonen for hele utskytningsenheten. Bakgrunnen til designet av elektronikkboksen kommer av ønsket om ha et eget rom til alt elektronikk og festepunkter på et sted, som ikke er i veien for dronene under utskyting. Den er designet som en firkantet plattform med fire avrundede hjørnevegger. Hjørneveggene har hver sin innrammede flate med festeanordning for gulvet dronene stables oppå. Mellom hjørneveggene er det lagt opp spor på hver side til vegger. Det er plassert én sylinder midt i utskytningsenheten. Denne fungerer som ekstra støtte til gulvet dronene er plassert på. Det er også lagt opp innfestningspunkt for batteriet, bærerkortet og to av føringstolpene.

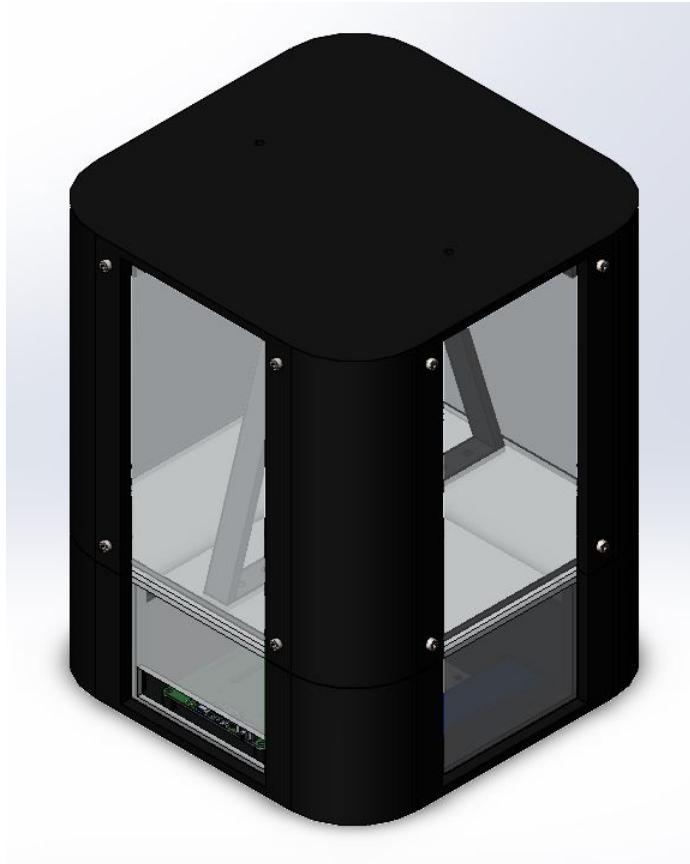
Føringstolpene er rammeverket som holder dronene på plass i utskytningsenheten. Det består av to sylinder, og to rectangle trekantede stolper. De er satt opp etter å ligge inntil dronen på hver side, sylinderne har et mellomrom på 63 mm, og trekantstolpene 93 mm. Disse er plassert med bestemte dimensjoner for å holde dronene på plass, og hjelpe dronene med å være stabile ved flyging ut av utskytningsenheten. Føringstolpene på langsiden av dronene er festet nede i elektronikkboksen som nevnt tidligere, mens rektangulære stolper plassert på kort siden av droene er festet i gulvet.

For veggger og gulv i elektronikkboksen ble det brukt pleksiglass. Det er kuttet ut fire veggplater og tre gulvplater, alle med en tykkelse på 4 mm. Veggene er enkle rektangulære plater, som passer i sporene mellom hjørneveggene i elektronikkboksen. Disse blir sklidd på plass i ferdiglagde spor i elektronikkboksen. Gulvet til dronene er delt i tre plater for å oppnå ønsket tykkelse, og er firkantede med mer kompliserte utskjæringer for skruehull til hvert hjørne og føringsstolpene på langsiden som er festet inni elektronikkboksen. I platene er det rektangulære utskjæringer på de to øvre platene og skruehull på nedeste plate, for plass til føringsstolpene til kortsiden av drone- ne nevnt over. I Figur 7.2 vises det full sammenstilling av utskytningsenheten uten lokk i Solidworks.

Lokket satt på utskytningsenheten i Figur 7.3 har samme design som elektronikkboksen. Den har også samme løsningen som elektronikkboksen med spor til pleksiglassveggene, og fire skruehull innfestninger på hver side. Det er også lagt opp skruehull på toppen av lokk konstruksjonen til et eventuelt håndtak. En festeløsning mellom elektronikkboksen og lokket har derimot ikke blitt sett på ettersom med tanke på viktigheten av lokket for første prototype og tidsrammen for prosjektet.



Figur 7.2: Sammenstilling av utskytningsenhet uten lokk i Solidworks.



Figur 7.3: Sammenstilling av utskytningsenhet med lokk i Solidworks.

Monteringsprosessen av utskytningsenheten ligger i appendiks A.1.2, og tekniske tegninger av utskytningsenheten ligger i appendiks H.2.

7.2.2 Produksjon

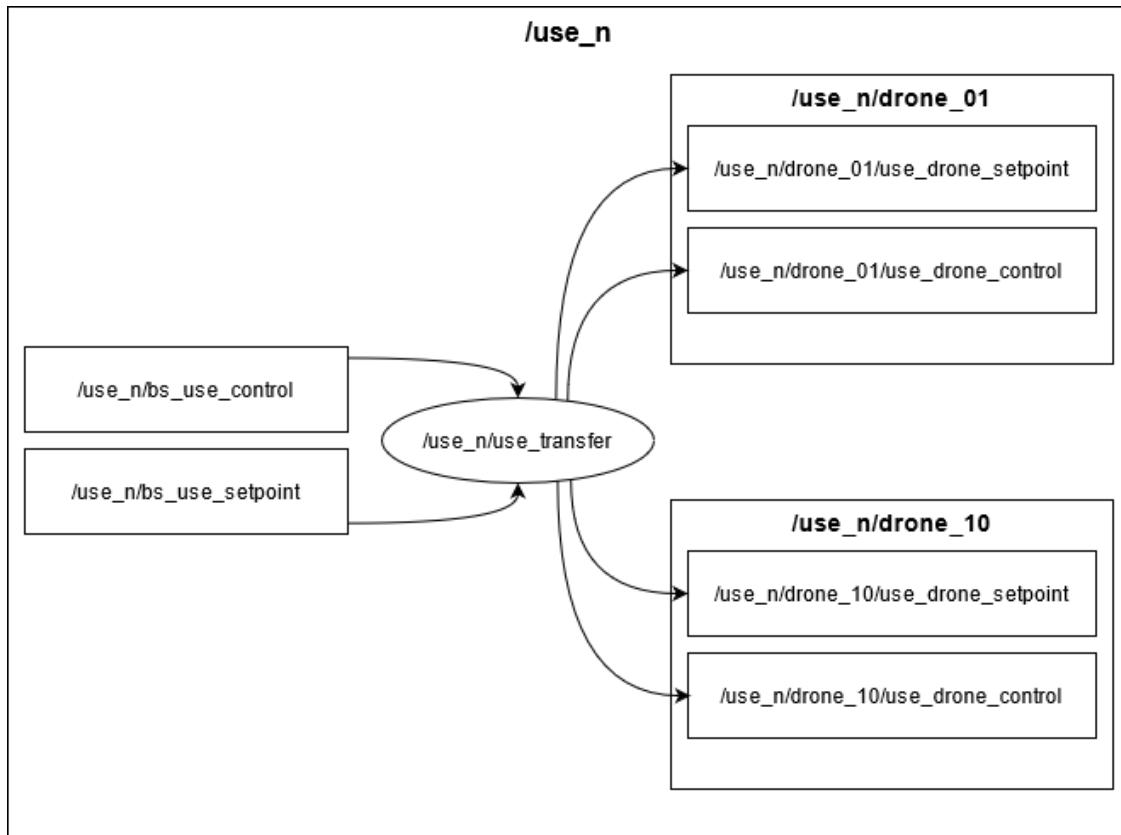
Utskytningsenheten konstruert for prosjektet er en prototype. For å produsere utskytningsenheten har det blitt brukt ulike produksjonsmetoder. Elektronikkboksen er 3D-printet av en mer avansert printer på UiA. Fordelen med dette er at printeren er mer nøyaktig. Materialet brukt er Onyx. Føringsstolpene til langsiden av dronene ble kuttet opp etter ønsket lengde og sveiset sammen på en felles plate for bedre stivhet. Føringsstolpene til kortssiden av dronene er 3D-printet, og festet i gulvet dronene står oppå. Pleksiglasset brukt til vegger i elektronikkboksen og plattform for dronene ble laserkuttet på universitetet.

7.3 Programvare

Programvaren i utskytningsenheten består ROS 2-noden som kjøres ved oppstart.

7.3.1 Design

Utskytningsenheten skal motta informasjon fra bakkestasjonen, analysere denne informasjonen og sende informasjonen videre til den korrekte dronen. Som vist i Figur 7.4 må noden abonnere på meldinger som publiseres på emnene `/use_n/bs_use_setpoint` og `/use_n/bs_use_control`. I tillegg må noden publisere til korrekt drone på emnene `/use_n/drone_n/use_drone_setpoint` og `/use_n/drone_n/use_drone_control`.



Figur 7.4: Graf over emner og node i utskytningsenhet n.

7.3.2 ROS 2 node

Importering av selvlagde meldinger

Først importeres de selvlagde meldingene. Disse meldingene må stemme med meldingene som publiseres til emnene fra bakkestasjonen og meldingene som sendes til dronen. Ved å bruke disse selvlagde meldingene er det enkelt å forsikre seg om at de inneholder det de trenger og ikke noe annet. Meldingen som sendes til `/use_n/bs_use_setpoint` inneholder informasjon om ønsket posisjon eller hastighet, og hvilken drone utskytningsenheten skal sende meldingen videre til. Meldingen som sendes til `/use_n/bs_use_control` inneholder egendefinerte kommandoer, rettere sagt fire variabler med boolske verdier. Disse stemmer med kommandoene brukt i dronen.

Definisjon av klasse og lese navnerom

Videre defineres klassen, og forelder-klassen instansieres med navnet `use_transfer`. Så hentes navnerommet noden startes i. Navnerommet hentes ved å kjøre funksjonen `get_namespace()` fra forelder-klassen. En node startes i et navnerom ved å legge på følgende argument i terminalen etter nodenavnet `--ros-args -r __ns:="/navnerom"`. Det er dette navnerommet som blir lagret i variabelen `self.my_namespace_` på linje 15 i Kodelisting 7.1.

Kodelisting 7.1: Definisjon av node-klasse i utskytningsenhet.

```
8 class Transfer(Node):
9
10    def __init__(self):
11        # Init parent-class
12        super().__init__('use_transfer')
13
14        # Get namespace
15        self.my_namespace_ = super().get_namespace()
```

Definisjon av publisher og subscribers

Deretter må det lages publishers og subscribers. Som nevnt tidligere i kapittelet skal noden i utskytningsenheten abonnere på to emner. Disse emnene defineres på linje 18 og 25 i Kodelisting 7.2. Meldingen "TrajectorySetpoint" inneholder informasjon om ønsket posisjon eller hastighet. Meldingen "DroneControl" inneholder egendefinerte kommandoer. Videre kalles funksjonene `self.recv_setpoints` og `self.recv_control` som kjøres når subscriberene mottar posisjonsmeldinger og kommando-meldinger, respektivt. Disse funksjonene er definert senere i koden.

I ROS 2 er det slik at emner som noden abonnerer på bruker navnerommet som defineres ved oppstart, det skjer derimot ikke med emner noden publiserer til. Derfor må navnerommet legges til når publisherene lages. Noden i utskytningsenheten skal publisere til to emner per drone. Det er planlagt at hver utskytningsenhet skal romme 10 droner, så det lages derfor 20 publishers. For å slippe å lage disse én og én, er det brukt for-løkker, som vist på linje 20 og 27 i Kodelisting 7.2.

For posisjons-meldingene defineres det en liste med kun et element, 0, som skal romme alle publishers. For-løkken kjører ni ganger der den lager publisher som publiserer til emnetnet `/use_drone_setpoint` med navnerommet som noden startet med og navnerommet til dronen som øker med 1 hver iterasjon. Det vil si at listen `self.trajectory_publishers`, etter for-løkken, har 10 elementer der det første elementet er 0 og de på følgende er `/self.my_namespace_/drone_01/use_drone_setpoint`, `/self.my_namespace_/drone_02/use_drone_setpoint`, og så videre opp til `/self.my_namespace_/drone_09/use_drone_setpoint`. Etter for-løkken legges det til en publisher med drone-navnerom 10 slik at det eksisterer en posisjons-publisher for alle dronene i utskytningsenheten.

For kontroll-meldingene er det gjort samme som for posisjons-meldingene, foruten at emnet som publiseres til er `/use_drone_control`.

Kodelisting 7.2: Definisjon publisher og subscribers i utskytningsenhet-noden.

```

17     # Trajectory setpoint transfer subscriber and publishers
18     self.trajectory_subscriber_ = self.create_subscription(TrajectorySetpoint,
19         'bs_use_setpoint', self.recv_setpoints, 10)
20     self.trajectory_publishers = [0]
21     for drone in range (1,10):
22         self.trajectory_publishers.append(self.create_publisher(
23             TrajectorySetpoint, self.my_namespace_ + '/drone_0' + str(drone) + '/use_drone_setpoint', 10))
24         self.trajectory_publishers.append(self.create_publisher(TrajectorySetpoint
25             , self.my_namespace_ + '/drone_10' + '/use_drone_setpoint', 10))
26
27     # Drone control transfer subscriber and publishers
28     self.droneControl_subscriber = self.create_subscription(DroneControl, ''
29         'bs_use_control', self.recv_control, 10)
30     self.droneControl_publishers = [0]
31     for drone in range (1,10):
32         self.droneControl_publishers.append(self.create_publisher(DroneControl
33             , self.my_namespace_ + '/drone_0' + str(drone) + '/use_drone_control', 10))
34         self.droneControl_publishers.append(self.create_publisher(DroneControl,
35             self.my_namespace_ + '/drone_10' + '/use_drone_control', 10))

```

Subscriber-funksjoner

Den første subscriber-funksjonen, `recv_setpoints(self, msg)`, kalles når `self.trajectory_subscriber_` mottar meldinger fra emnet `/bs_use_setpoint`, som vist i Kodelisting 7.3. Denne funksjonen leser av hvilken drone meldingen er ment for og publiserer meldingen videre til det den korrekte dronen. Dersom dronen oppgitt har identifikasjon "999" vil det startes en for-løkke som sender meldingen til alle de 10 dronene som hører til utskytningsenheten. Fordelen med dette er å kunne sende ønsket posisjon eller hastighet til alle dronene samtidig. Ønsket posisjon er relativt til dronens start-posisjon. Dersom dronen ikke har identifikasjon "999" vil det foregå en sjekk om dronen har identifikasjon innenfor 1 til og med 10. Dersom dette er godkjent vil meldingen sendes til dronen med identifikasjonen oppgitt i meldingen.

Kodelisting 7.3: Definisjon av subscriber-funksjon til posisjons-meldinger.

```

36 def recv_setpoints(self, msg):
37     # Transfer setpoint to correct drone
38     if (msg.drone == 999):
39         for publisher in self.trajectory_publishers:
40             if (publisher != 0):
41                 publisher.publish(msg)
42
43     elif (msg.drone > 0 and msg.drone < 11):
44         self.trajectory_publishers[msg.drone].publish(msg)
45
46     # Send log
47     self.i_trajectory += 1
48     self.get_logger().info('Transferred TrajectorySetpoint: %d' % self.
        i_trajectory)

```

Den andre subscriber-funksjonen, `recv_control(self, msg)`, kalles når `self.droneControl_subscriber` mottar meldinger fra emnet `bs_use_control`, som vist i Kodelisting 7.4. Denne funksjonen gjør akkurat det samme som i den første subscriber-funksjonen bortsett fra at den kjører når `self.droneControl_Subscriber` mottar meldinger.

Kodelisting 7.4: Definisjon subscriber-funksjon til kommando-meldinger.

```
51 def recv_control(self, msg):
52     # Transfer control to correct drone
53     if (msg.drone == 999):
54         for publisher in self.droneControl_publishers:
55             if (publisher != 0):
56                 publisher.publish(msg)
57
58     elif (msg.drone > 0 and msg.drone < 11):
59         self.droneControl_publishers[msg.drone].publish(msg)
60
61     # Send log
62     self.i_droneControl += 1
63     self.get_logger().info('Transferred DroneControl: %d' % self.
i_droneControl)
```

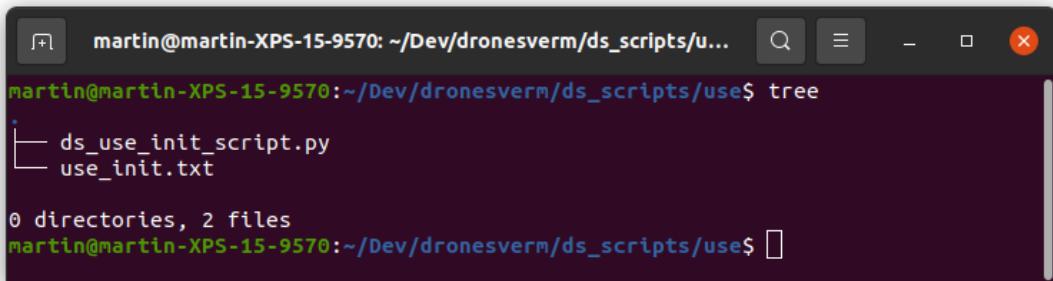
Kildekoden til noden ligger i appendiks C.3.1

7.3.3 Implementasjon

Når noden er satt opp og klar til bruk må det legges inn at den kjører automatisk ved oppstart. Fra bash kjøres det et python-skript ved oppstart som gjennomfører de nødvendige stegene for å starte ROS 2 og noden i riktig navnerom.

Skript for oppstart med navnerom

Det er laget en mappe på RPi CM4 som skal være i utskytningsenheten, kalt ”use”. I denne mappen er det laget to filer: ”ds_use_init_script.py” og ”use_init.txt”. Strukturen er vist i Figur 7.5.



The terminal window shows the following directory structure:

```
martin@martin-XPS-15-9570:~/Dev/dronesverm/ds_scripts/use$ tree
.
└── ds_use_init_script.py
    └── use_init.txt

0 directories, 2 files
```

Figur 7.5: Mappestruktur oppstart-mappe.

Banen til python-skriptet legges inn i ”.bashrc”-filen som forklart i kapittel 5.3, slik at den kjører ved oppstart. I python-skriptet blir først banen til der skriptet ligger lagret. Dette er fordi skriptet videre leter i tekst-filen som ligger i samme mappe. På denne måten kan disse to filene, plasseres vilkårlig i systemet så lenge de ligger i samme mappe. I tekst-filen blir det søkt etter ordet ”use:”, dersom dette blir funnet lagrer skriptet tallet som står oppført bak til en variabel. Videre blir ROS 2 kjørt og noden til utskytningsenheten starter med navnerom i forhold til tallet som er lest fra tekst-filen.

Oppstarts-skriptet ligger i appendiks C.3.2.

Kapittel 8

Testing

På bakgrunn av å skulle lage et velfungerende system som i fremtiden, forhåpentligvis, kan anvendes til ulike arbeidsoppgaver både i private og industrielle sammenhenger, er det blitt satt opp ulike tester som omhandler systemets funksjoner, og redegjør for hva om eventuelt kan forbedres.

Under testing av utskytningsenheten ble det brukt like prinsipp som under testing av de ulike systemene i dronene. Ved design og produksjon av utskytningsenheten var ønsket om å få et ferdig produkt høyt vektlagt. Utskytningsenheten ble derfor regelmessig testet med mindre iterasjoner for hvert steg for å indikere om det vil fungere som planlagt, og blitt iterert deretter ved hvert 'steg' av prosessen til ferdig prototype.

I løpet av design- og produksjonsprosessen har det blitt utført mindre tester med jevne mellomrom. Dronene er satt sammen av komponenter og programvare fra flere leverandører. I og med at det har blitt designet egne bærekort, har det vært essensielt å holde seg unna kompatibilitetsproblemer. Derfor ble alle delsystemer testet individuelt, og feil ble rettet opp før utviklingen fortsatte. Dette ble gjort for å minimere potensielle feil under testingen av hele systemet.

Alle fysiske tester gjennomføres med visse sikkerhetstiltak for å sikre at det ikke forekommer skade på eiendom eller personer. Testingen foregår på åpne områder med god klaring til bygninger. Det vil alltid være én person ansvarlig for sikkerheten til andre personer i området. Testingen foregår alltid under trygge værforhold og etter retningslinjene til Luftfartstilsynet som er nevnt i teori-kapittel 2.4. Personer som deltar i testingen må bruke vernebriller, hanske og refleksvest.

Ved testing av dronen og det automatiske utskytingssystemet til dronesverm er dronen montert på samme måte som nevnt i kapittel 6.2.1. Sensorer og andre parametere Pixhawk 4 mini er avhengig av, er kalibrerte og satt via programmet QGroundControl. QGroundControl er et program som er utviklet av Dronicode som blant annet brukes til å sette opp droner med PX4-Autopilot maskinvare, det vil si at programmet bistår brukeren med å kalibrere sensorer og sette ulike parametere som kreves for å sette opp et brukende fartøy [75]. Videre er det montert en radio-mottaker og en telemetri-modul. Radio-mottakeren må være montert siden dronen i løpet av testingen skal fly med radio-kontroller. I tillegg vil radio-kontrolleren bli en sikkerhetsfunksjon ved flyging i utenbordskontroll modus. Telemetri-modulen som er montert på gjør det mulig å motta informasjon til en datamaskin som kjører QGroundControl fra autopilot-maskinvaren mens den er i luften. I tillegg kan telemetri-modulen brukes til å endre autopilot-maskinvarens parametere og kalibrere sensorer uten å måtte koble til en PC med USB kabel.

På grunn av at PX4-Autopilot krever ekstern posisjons-informasjon for å fly i utenbordskontroll modus, se teorikapittel 2.1.1, er det også påmontert en GPS på dronen under testingen.

Kommunikasjonen mellom bakkestasjonen, utskytningsenheten og dronen foregår med ROS

2 over WiFi, under testingen blir det satt opp et hotspot for at enhetene skal kunne kommunisere.

På grunn av en feil med kretskortet til utskytningsenheten som gjør at Raspberry Pi Compute Module 4 ikke booter, blir datamaskinen i utskytningsenheten simulert ved å kjøre noden på samme PC som bakkestasjonen.

8.1 Simulering av systemet

8.1.1 Formål

Formålet med denne testen er å verifisere at programvaren laget for bakkestasjonen, utskytnings-enheten og dronen fungerer som planlagt.

8.1.2 Utførelse

Først startes alle ROS 2-nodene på datamaskinen simuleringen skal foregå på. Så startes simuleringen i henhold til simulerings-eksempelet vist i dokumentasjonen til PX4 [76]. Deretter skal det sendes en posisjon fra bakkestasjonen som dronen skal fly til. I kommando-vinduet på bakkestasjonen skrives først 01 01 arm for å armerer dronen. Så sendes den ønskelige posisjonen ved å skrive 01 01 z-1, som vil si at dronen skal fly opp til 1 m i forhold til utgangsposisjonen.

8.1.3 Test-kriterier

Denne testen godkjennes dersom dronen flyr til posisjonen sendt fra bakkestasjonen. *Se kapittel 9.3.1 for resultatet av testen.*

8.2 Fly en drone med radio

8.2.1 Formål

En drone skal flys ved hjelp av en radio-kontroller. Formålet med denne testen er å kontrollere at alle de grunngjende funksjonene til dronen fungerer som planlagt. Komponentene i dronen vil for første gang bli satt sammen til ett system og bli testet. Det skal sjekkes at systemet starter opp og responderer til endringer som gjøres med radio-kontrolleren. Deretter skal testen brukes til å tune dronen slik at stabiliseringssystemet til Pixhawk fungerer optimalt. Under testen vil det bli gjort endringer i PID-parameterene til dronen for å øke stabiliteten ved flyging i stabiliseringsmodus. Dette blir gjort på grunn av at dronen må være så stabil som mulig til senere ved flyging i utenbordskontroll modus.

8.2.2 Utførelse

Dronen plasseres ut på et testområde, et åpent område hvor det er minst mulig fare for skade på liv eller materiale. Deretter sjekkes det for at radio kontrolleren er tilkoblet, slik at dronen kan armeres, det vil si klargjort slik at propellene begynner å snurre. Det sjekkes også om QGroundControl mottar telemtri-data fra dronen. Deretter skal dronen forsøkes å fly, dersom dronen ikke klarer å fly på grunn av ustabilitet, må dronens PID-regulator justeres og flyging forsøkes på nytt.

8.2.3 Test-kriterier

Ved bruk av radio-kontrolleren skal dronen kunne armeres, disarmeres, regulere motorhastighet, lette fra bakken og fly kontrollert. Om dronen gjennomfører alle kravene vil testen betraktest som godkjent. *Se kapittel 9.3.2 for resultatet av testen.*

8.3 Fly drone ut av utskytningsenhet med radio

8.3.1 Formål

Formålet med denne testen er å kontrollere at dronen klarer å fly ut av utskytningsenheten ved bruk av en radio-kontroller. Hensikten er å se at føringsstolpene i utskytningsenheten fungerer som planlagt. Det skal også sjekkes om propellene til dronen treffer føringsstolpene.

8.3.2 Utførelse

Først plasseres én drone i utskytningsenheten, videre plasseres utskytningsenheten på et åpent område. Deretter, ved bruk av radio-kontrolleren, armeres dronen og den flys ut av utskytningsenheten.

8.3.3 Test-kriterier

Dronen skal ved hjelp av en radio-kontroller fly ut av utskytningsenheten i én bevegelse uten å sette seg fast. Propellene bør helst ikke treffe føringsstolpene. Om dronen gjennomfører kontrollert flyging ut av utskytningsenheten vil testen betraktes som godkjent. *Se kapittel 9.3.3 for resultatet av testen.*

8.4 Fly drone i utenbordskontroll modus

8.4.1 Formål

Formålet med testen er å fly én drone i utenbordskontroll modus ved bruk av posisjons- eller hastighetskontroll. Det skal kontrolleres at dronen utfører gitte handlinger og mottar korrekt informasjon sendt fra bakkestasjon via utskytningsenheten. Deretter vil det bli testet å fly én drone ut av utskytningsenheten i utenbordskontroll modus.

8.4.2 Utførelse

I denne testen skal det først kontrolleres om dronen kan armeres og disarmes fra bakkestasjonen. Dette vil bli gjort ved å skrive 01 01 arm og 01 01 disarm i kommando-vinduet til bakkestasjonen. Deretter vil det bli testet å fly dronen ved bruk av posisjons- og hastighetskontroll. Dette blir gjort ved å henholdsvis skrive 01 01 z-1 og 01 01 vz-0.2, for så å launche ved å skrive 01 01 launch i kommando-vinduet. For å lande dronen vil kommandoen 01 01 land bli skrevet. Det blir skrevet 01 01 før hver kommando da drone 1 i utskytningsenhet vil bli testet. I tillegg vil komandoene reset og velocity som henholdsvis resetter alle parameterene og gjør dronen klar til flyging med hastighetskontroll også bli testet.

Når det er verifisert at dronen oppfører seg som planlagt vil det bli testet å fly ut av utskytningsenheten ved bruk av hastighetskontroll. Hastighetkontroll brukes for å få høy nok hastighet på dronen da det i test 8.3 ble erfart at dronen måtte fly raskt ut for å unngå problemer. Hastigheten vil da bli satt til vz-2.

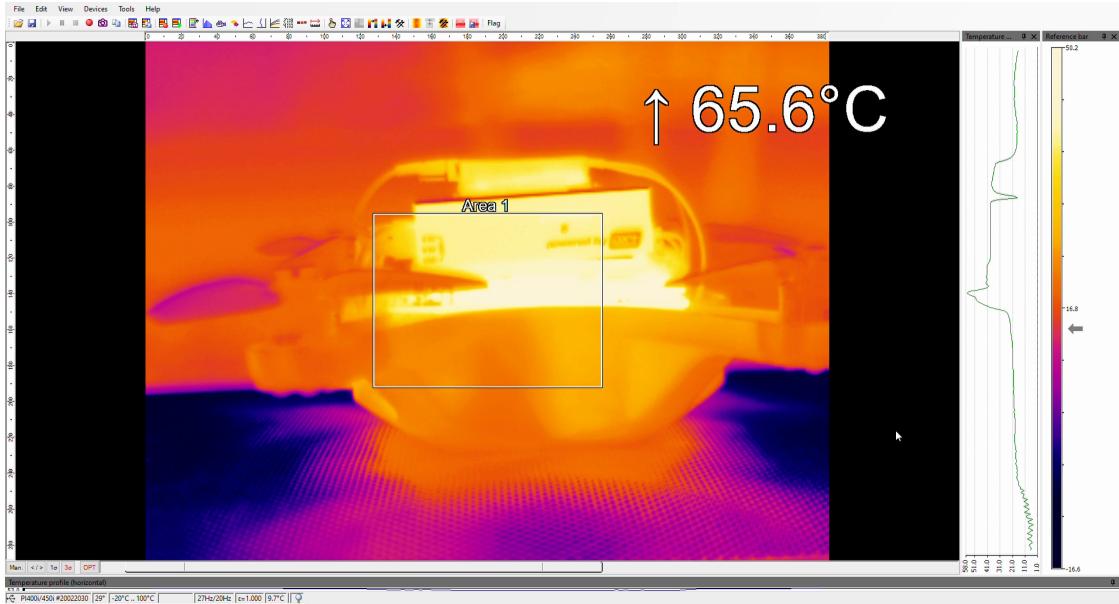
8.4.3 Test-kriterier

For at testen skal bli betraktet som velykket må dronen motta all informasjon sendt fra bakkestasjon via utskytningsenheten og utføre alle gitte oppgaver. Dronen må også kunne fly stabilt i utenbordskontroll modus og fly ut av utskytningsenheten med likt resultat som i test 8.3. *Se kapittel 9.3.4 for resultatet av testen.*

8.4.4 Forbedringer etter test

Som nevnt i resultat-kapittel 9.3.4 og diskusjon-kapittel 10.1.3 hadde dronen problemer under testing av utenbordskontroll modus. Det ble som beskrevet antatt å være et varmeproblem.

Ved å bruke et termisk-kamera, ble temperaturen på autopilot-maskinvaren og ettkortsdata-maskinen målt. Disse målingene, som vist i Figur 8.1, ser ut til å være innenfor kravene om maksimal driftstemperatur, som for begge enhetene er på 85 °C [58] [5]. Denne målingen er ikke nødvendigvis helt korrekt, siden den ble gjennomført utendørs i skyggen med en normaltemperatur på rundt 10 °C ifølge Yr [77]. I tillegg hadde dronen lokket av under måling, dette kan også ha påvirket resultatet.



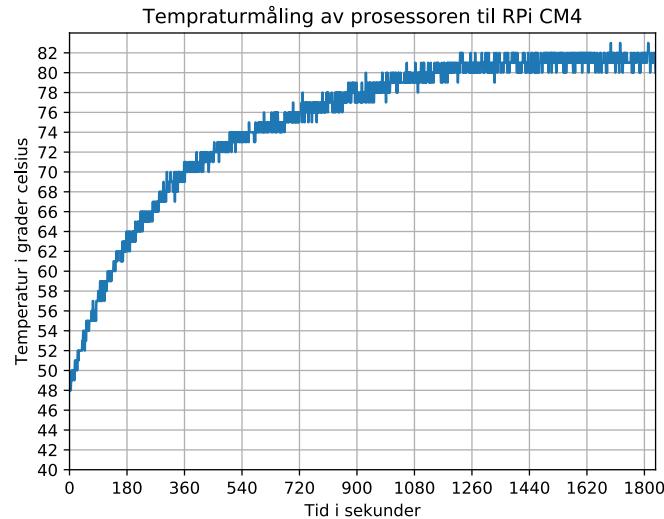
Figur 8.1: Termisk måling av drone.

Ut i fra bildene til det termiske kameraet var det tydelig at RPi CM4 var den varmeste kilden, dette ble derfor undersøkt nærmere. Ved å kjøre kommandoen `top` i terminalen, ble det observert at micrortps-agenten brukte mer enn 100% av prosessor-kapasiteten, dette er vist i figur 8.2 (a). Det antas at dette var grunnen til den høye temperaturen. Ved å lage et Python-skript (C.4.5) som logget prosessor-temperaturen hvert sekund og automatisk kjørte ved oppstart, kunne temperaturutviklingen fra oppstart logges. RPi CM4 ble kjølt helt ned, før den ble skrudd på igjen, deretter ble datamaskinen stående på i 30 minutter. Etter 30 minutter ble loggingen avsluttet og temperaturutviklingen fremstilt grafisk, som vist i Figur 8.2 (b).

I grafen kan det observeres at prosessor-temperaturen stiger opp til over 80 °C og at den deretter flater ut. Grunnen til dette er at prosessoren har en feilsikkerhet som struper klokke-hastigheten slik at temperaturen ikke skal fortsette å stige og til slutt ødelegge prosessoren [58]. Når denne strupingen foregår vil det ta lengre tid for oppgaver og fullføres. Dette er kritisk siden systemet er designet for å sende meldinger med en gitt rate.

ubuntu@ubuntu: ~/dev/ds_scripts													
top - 07:23:32 up 38 min, 4 users, load average: 1.83, 2.26, 2.29													
Tasks: 140 total, 3 running, 137 sleeping, 0 stopped, 0 zombie													
%Cpu(s): 7.1 us, 35.6 sy, 0.0 ni, 56.8 id, 0.0 wa, 0.0 hi, 0.5 si, 0.0 st													
MiB Mem : 3793.3 total, 3011.4 free, 376.6 used, 405.2 buff/cache													
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 3336.3 avail Mem													
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND		
2157	ubuntu	20	0	4671628	223952	126236	R	102.3	5.8	38:48.54	micrortps		
2477	root	20	0	0	0	0	I	26.2	0.0	6:19.59	kworker+		
3011	root	20	0	0	0	0	I	24.2	0.0	2:35.47	kworker+		
3034	root	20	0	0	0	0	I	16.2	0.0	0:46.49	kworker+		
2174	ubuntu	20	0	774568	68416	27332	R	8.9	1.8	3:23.15	px4_off		

(a) Hvor mange prosent av maksimalkraft prosessoren kjører på.



(b) Graf av temperaturen til prosessoren.

Figur 8.2: Original micrortps-agent.

Programmet som brukte mest prosessor-kraft, microRTPS agenten, lager 22 emner ved oppstart. I denne oppgaven er kun 5 av disse emnene brukt. Det ble forsøkt å fjerne de emnene som ikke er brukt av det automatiske utskytningssystemet. De ubrukte emnene ble fjernet fra ROS 2-pakken til microRTPS agenten. Etter dette ble pakken bygget og RPi CM4 startet på nytt. Datamaskinen hadde høy temperatur etter byggingen, så den ble kjølt ned før ny temperatur-logging. *Se kapittel 9.3.4 for resultat av utbedringen.*

8.5 Fly to droner ut av utskytningsenhet

8.5.1 Formål

Formålet med testen er å teste det automatiske utskytningssystemet for dronesvermen ved å fly to droner ut av utskytningsenheten med utenbordskontroll modus.

8.5.2 Utførelse

To droner plasseres i en utskytningsenhet og blir gjort klare til å fly i utenbordskontroll modus. Først blir dronene armert samtidig ved å skrive **999 999 arm** i kommando-vinduet til bakkestasjonen. Deretter blir begge dronene satt til å ha en vz hastighet på 2 m/s oppover. Når dronene da er armert og satt til den gitte hastigheten vil de bli launchet én og én etterhverandre så fort det lar seg gjøre ved kommandoene **01 01 launch** og **01 02 launch**.

Når dronene har tatt av ut av utskytningsenheten vil to godkjente droneoperatører ta over kontrollen på dronene for å prøve å lande de.

8.5.3 Test-kriterier

Om begge dronene tar av kontrollert ut av utskytningsenheten uten problemer etter launch kommando er sendt vil testen betraktes som godkjent. *Se kapittel 9.3.5 for resultatet av testen.*

Kapittel 9

Resultater

9.1 Drone

Dronen som er bygget er en videreutvikling av en modulær quad-rotor som ble påbegynt av BSc.-oppgaver våren 2018 og 2019. Den har under denne perioden flydd kontrollert for første gang på grunn av inkludering av autopilot-maskinvaren Pixhawk 4 mini.



Figur 9.1: To komplette droner.

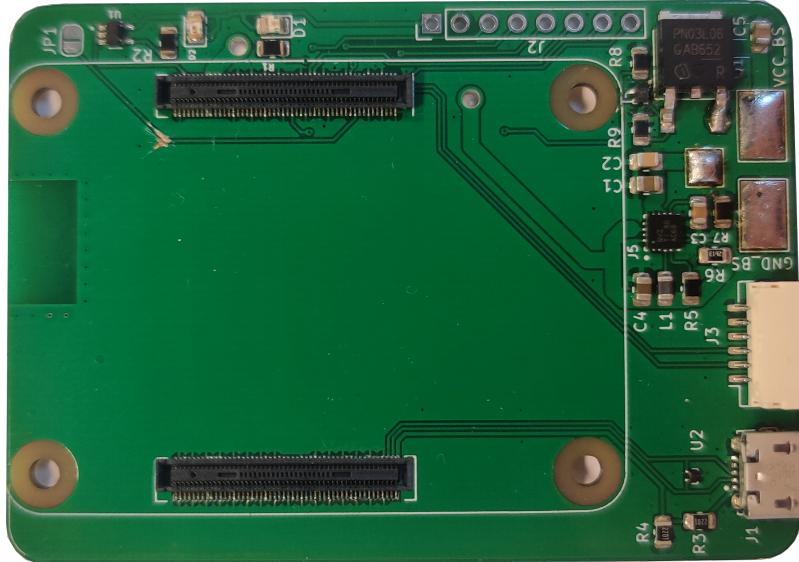
Dronens konstruksjon har holdt samme fasong som ved de tidligere iterasjonene, men har nå blitt tilpasset i henhold til utskytningsenheten, autopiloten og andre komponenter. Det har blant annet ført til at dronen blitt flat under, slik at flere droner kan stå ovenpå hverandre i utskytningsenhetene. I tillegg er de buete sidene på droneskroget formet om og gjort rettere uten kanter slik at dronen glir bedre langs føringsstolpene i utskytningsenheten. Posisjonen til motorinnfestningene er ikke endret, men hoveddelen av skroget, altså der all elektronikken ligger, blitt 6 mm bredere. Som vil si at totalbredden der er 60 mm. På grunn av at den har blitt bredere har det vært mulig å senke høyden på dronen med 4 mm, og den er derfor 58 mm høy uten GPS montert. Droneskroget er designet for å 3D-printes, men kan med små forandringer tilpasses til å bli vakuumformet eller sprøytestøpt. Tilpasningene måtte da har vært å tegne inn slippvinkler på alle rette flater. Samtidig måtte innfestnings-stolpene til strømmodulen blitt fjernet om droneskroget skulle blitt vakuumformet, disse kunne blitt ettermontert.

Motorregulatorene, motorene og batteriet som sitter i dronen er uforandret fra tidligere iterasjoner, resten av komponentene er enten oppgradert eller nye. Ved inkludering av Pixhawk 4 mini krevde den å ha en strømmodul mellom batteriet, motorregulatorer og seg selv, derfor er strømmodulen sammen med Pixhawk 4 mini en ny komponent i dronen. Videre er ettkortsdatamaskinen oppgradert fra en 32-bit Raspberry Pi Zero W til en 64-bit Raspberry Pi Compute Module 4. Den nye ettkortsdatamaskinen er blant annet utstyrt med en prosessor som har fire 1.5 GHz kjerner, kan ha opptil 8 GB RAM og 32 GB eMMC Flash minne. Med bytte av ettkortsdatamaskin måtte det lages et bærekort til Raspberry Pi Compute Module 4.



Figur 9.2: Raspberry Pi Compute Module 4.

Bærekortet har for øyeblikket tre funksjoner; gi strøm til RPi CM4, bistå med kommunikasjon mellom RPi CM4 og Pixhawk 4 mini, og være strømbryter for all elektronikken i dronen. Alle de interne kretsene på bærekortet fungerer som planlagt, som vil si at RPi CM4 kan skru av og på autopiloten og motorregulatorene. Dersom den fysiske bryteren som er koblet til bærekortet blir brutt skrur hele systemet seg av. Konnektorene som står for UART koblingen mellom autopiloten og ettkortsdatamaskinen, og USB OTG-porten som gjør det mulig å skrive til eMMC lagringen til RPi CM4 er testet og fungerer som planlagt.



Figur 9.3: Komplett bærekort til drone.

Med inkludering av autopilot, og oppgradering av ettkortsdatamaskinen har totalvekten på dronen med GPS, telemetri-modul og radio-mottaker, økt til 324 g.

Dronen består nå av en elektronikkpakke som er modularer siden alle komponenter, utenom strømmoden, motorene og motorregulatorene enkelt kan monteres uten bruk av verktøy eller skruer. Komponentene som enkelt kan monteres blir montert ved bruk av innfestningssmetoden som nevnt i kapittel 6.2.1 og blir koblet sammen ved bruk av konnektorer. Strømmoden, motorene og motorregulatorene er loddet sammen, og er derfor fast monterte i dronen.

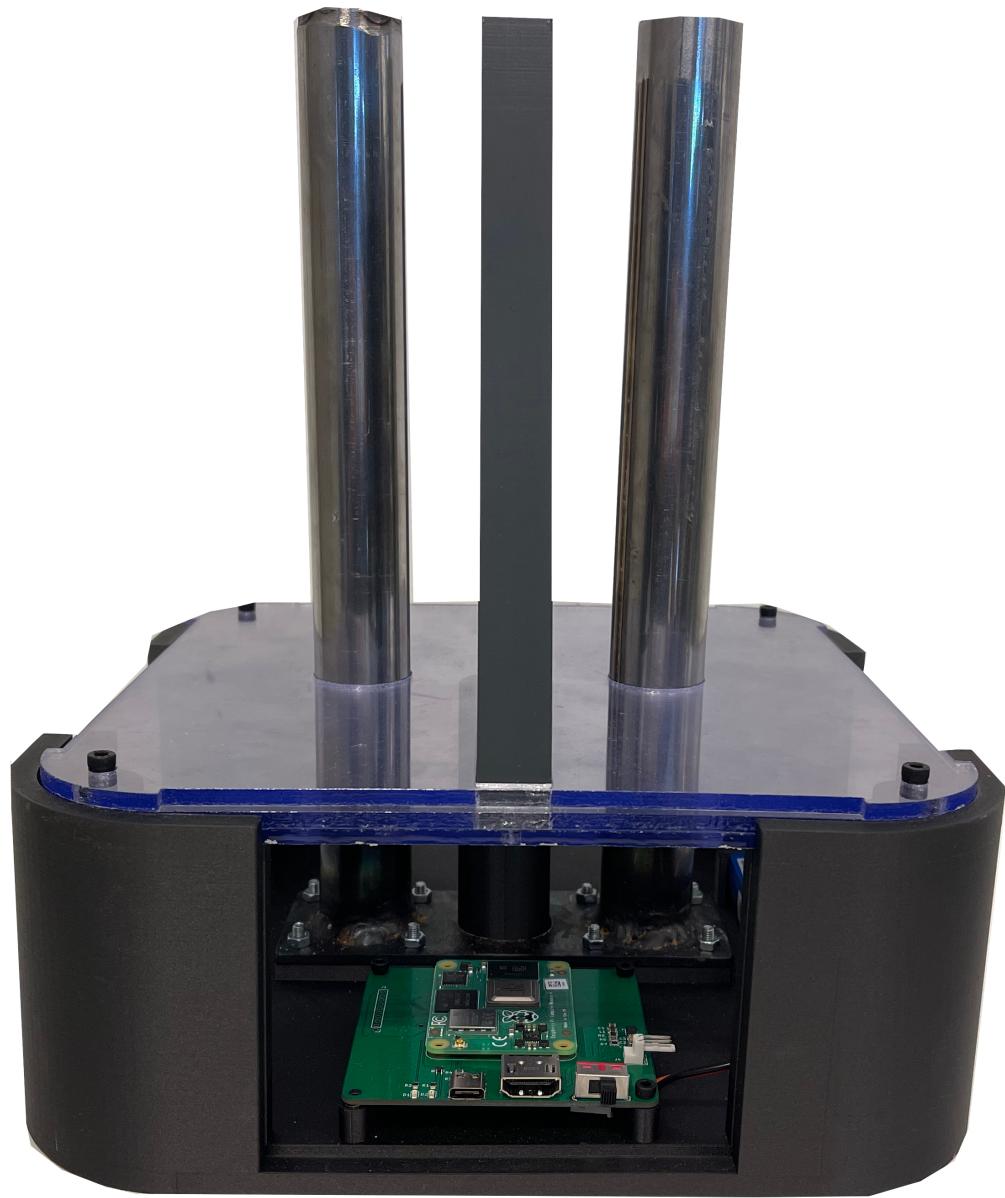
Komponenter som har blitt brukt, men som nødvendigvis ikke vil bli brukt i fremtiden som har vært av nytte er GPS, telemtri-modul og radio-mottaker. Under testing måtte lokket gjøres 7

mm høyere for å få plass til telemetri-modul og radio-mottaker som lå ovenpå Pixhawk 4 mini. I tillegg ble det tegnet og 3D-printet en skål som GPS-en ligger i ved flyging i utenbordskontroll modus.

I perioden er det produsert to fungerende droner som er blitt brukt til å teste det automatiske utskytningssystemet.

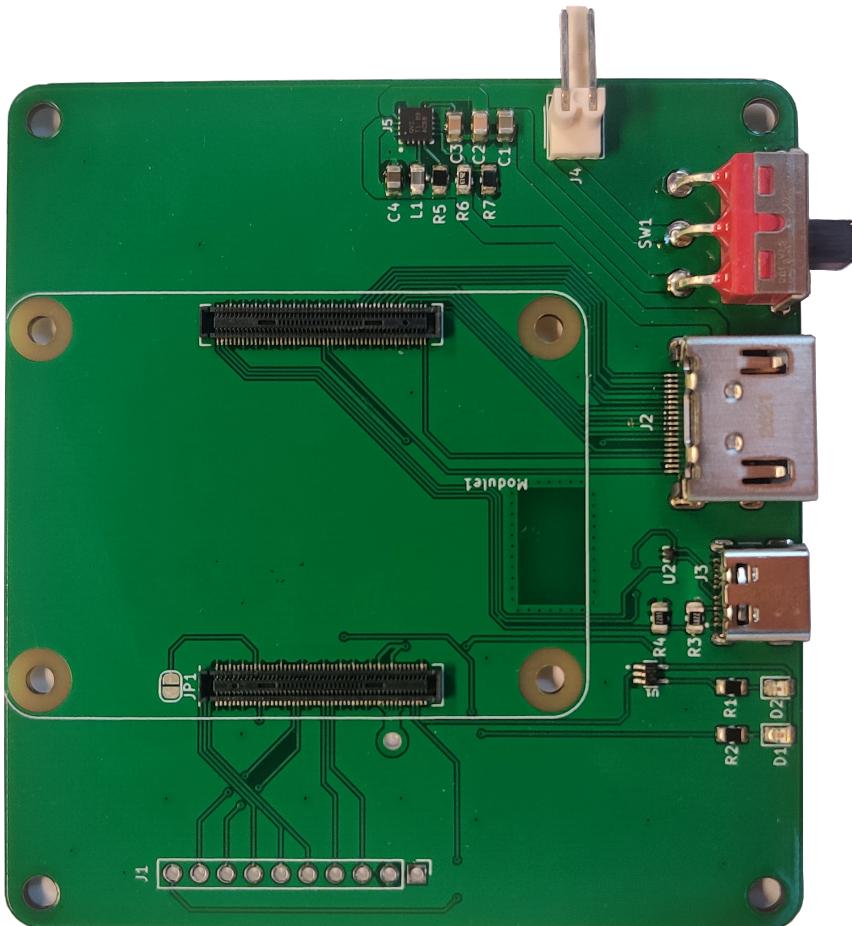
9.2 Utskytningsenhet

Utskytningsenheten er en ny tilføyelse til universitets dronesverm-prosjekt. Som sett i Figur 9.10 kan det stables to droner oppå hverandre i den produserte prototypen av utskytningsenheten, men den er designet til at en ferdig versjon vil romme opptil 10 droner. Stablingen av dronene foregår ved å ha fire føringstolper plassert mellom armene på dronene. Føringstolpene har også som oppgave å geleide dronene stabilt og kontrollert ut av utskytningsenheten ved utskyting. Ved utskyting vil dronene fly én og én ut av utskytningsenheten.



Figur 9.4: Utskytningsenhet uten droner.

Prototypen til utskytningsenheten er produsert uten lokk. Elektronikkboksen er 3D-printet og satt sammen av pleksiglass-plater. De sirkulære føringsstolpene på er produsert i stål og de trekant-formede føringsstolpene er 3D-printet. I utskytningsenhetens elektronikkboks er det for øyeblikket god plass og rommer kun ett batteri og en Raspberry Pi Compute Module 4 med et egenprodusert bærekort. Til sammen veier utskytningsenheten med alle komponenter 2633 gram.



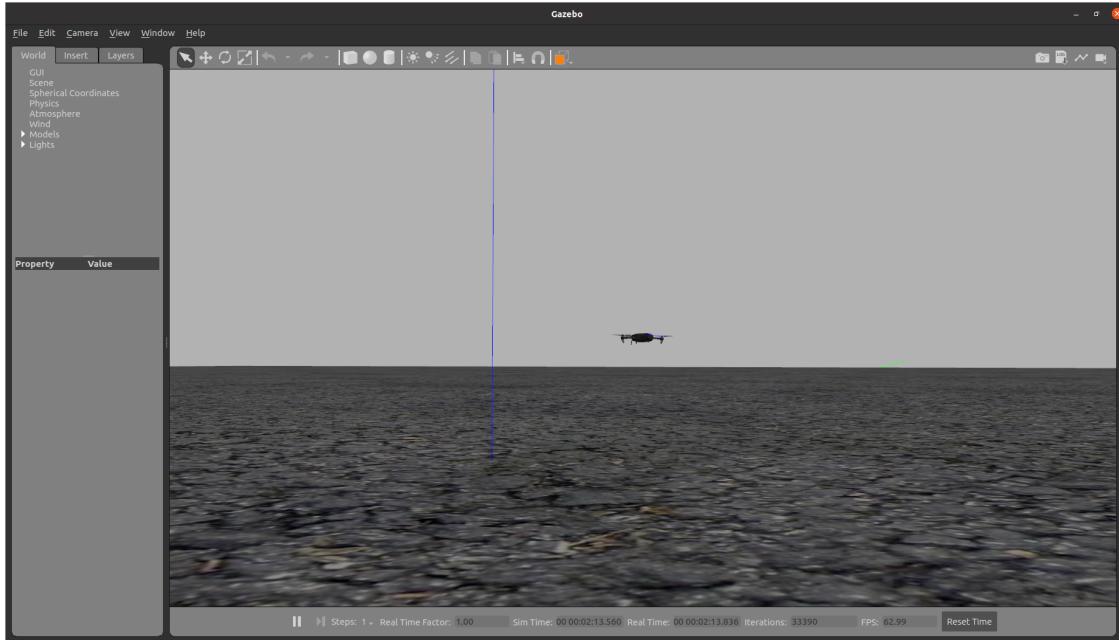
Figur 9.5: Komplett bærekort til utskytningsenheten.

Bærekortet som ble laget til utskytningsenheten var planlagt at skulle være ét mellomledd mellom dronene og bakkestasjonen som skulle forhindre at det i lengden vil bli krevet for mye av bakkestasjonen. I tillegg var det planlagt at brukeren bør ha mulighet til å koble seg på bærekortet med HDMI og USB for å lese av data og informasjon etter endt flyging. Bærekortets spenningsregulator-krets fungerer ikke selv om den er identisk til dronens krets. Derfor har det ikke vært mulig å teste bærekortets funksjoner.

9.3 Testing

9.3.1 Simulering av systemet

I denne testen ble programvaren til alle delene av systemet testet. Alle ROS 2-nodene ble startet på datamaskinen og simuleringen startet i henhold til dokumentasjonen. Ved å sende kommandoene nevnt i utførelsen flyg dronen opp til den ønskelige posisjonen og testen betraktes som godkjent.



Figur 9.6: Simulering av systemet.

9.3.2 Fly en drone med radio

I denne testen ble dronen flydd for første gang med radio. For at testen skulle betraktes som godkjent måtte dronen kunne armeres, disarmeres, motorhastigheten reguleres, løfte fra bakken og fly kontrollert. Dronen kunne vellykket armeres og disarmeres, i tillegg kunne motorene hastighetsreguleres. Ut fra dette kunne PID-kontrolleren justeres til det punktet at dronen ble stabil på bakken ved økt pådrag.

Dronen klarte derimot ikke, selv med maksimalt pådrag, å fly høyere enn rundt 40 cm. Dette betraktes ikke som kontrollert flyging og testen godkjennes ikke. *Se kapittel 10.1.1 for diskusjon av resultatet.*

Etter endringer på dronen i henhold til kapittel 10.1.1, ble samme test gjennomført på nytt. Alle test-kriteriene ble godkjent og testen betraktes som godkjent.



Figur 9.7: Flyging med radio-kontroller.

9.3.3 Fly drone ut av utskytningsenhet med radio

I denne testen skulle dronen fly ut av utskytningsenheten for å verifisere at føringsstolpene fungerte som planlagt. Dronen klarte å fly ut av utskytningsenheten i én bevegelse uten å sette seg fast, men det ble observert at propellene såvidt kom i kontakt med føringsstolpene. Ut ifra test-kriteriene kan testen betraktes som godkjent, fordi propellene ikke bør treffe føringsstolpene, men det er ikke kritisk at de gjør det. *Se kapittel 10.1.2 for diskusjon av resultatet.*

9.3.4 Fly drone i utenbordskontroll modus

I denne testen skulle én drone fly i utenbordskontroll modus. Først ble det forsøkt å armere dronen fra bakkestasjonen via utskytningsenheten, resultatet av dette var at dronen kun armerte ved noen få anledninger. I tillegg til at dronen var ustabil ved arming, virket det også tilfeldig når dronen lot seg kontrollere med posisjon- eller hastighetskontroll. Testen er derfor ikke godkjent på grunn av ustabilitet. *Se kapittel 10.1.3 for diskusjon av resultatet.*

Resultat av utbedring

Som beskrevet i kapittel 8.4.4 ble feilen forsøkt utbedret. Etter oppstart av RPi CM4 ble kommandoen `top` kjørt i terminalen, og som vist i Figur 9.8 (a) brukte microRTPS agenten betydelig mindre prosessor-kraft. Etter 30 minutter ble temperatur-loggingen avsluttet og fremstilt grafisk, se Figur 9.8 (b). I denne grafen er det tydelig at temperaturen stabiliserte seg på rundt 62 °C, som er godt innenfor anbefalt driftstemperatur i databladet [58]. *Se kapittel 10.1.3 for diskusjon av resultatet.*

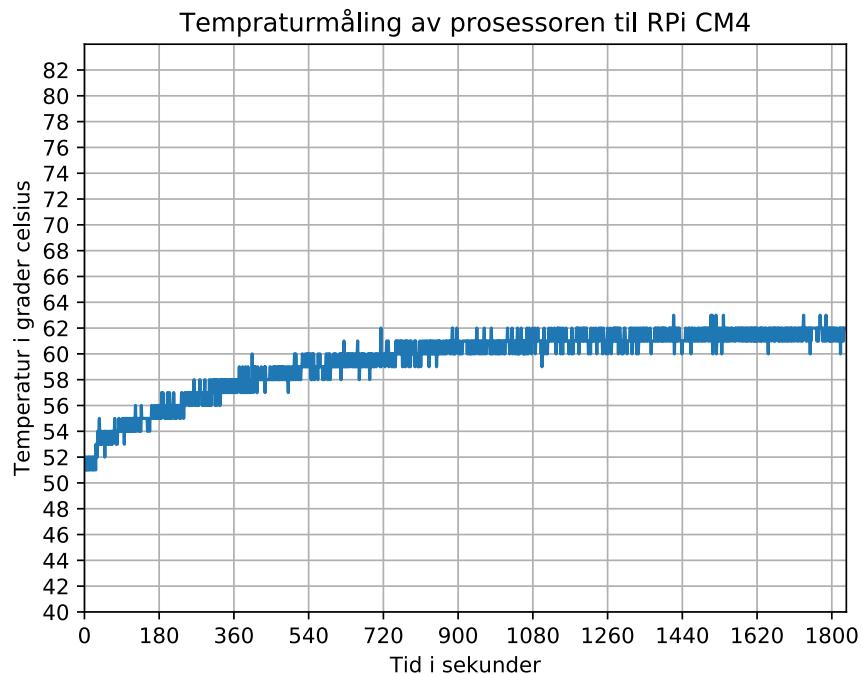
```

ubuntu@ubuntu:~ top - 09:08:31 up 5 min,  3 users,  load average: 0.12, 0.23, 0.12
Tasks: 147 total,  1 running, 146 sleeping,  0 stopped,  0 zombie
%Cpu(s):  3.2 us,  1.9 sy,  0.0 ni, 94.8 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
MiB Mem : 3793.3 total, 3162.6 free,   268.5 used,   362.1 buff/cache
MiB Swap:    0.0 total,      0.0 free,      0.0 used. 3460.5 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM TIME+ COMMAND
2175 ubuntu    20   0 760440 64548 24856 S  9.9  1.7 0:31.14 px4_of+
2161 ubuntu    20   0 2452416 25496 13820 S  7.3  0.7 0:21.82 micrort+
2424 ubuntu    20   0  9248  3260  2684 R  0.7  0.1 0:00.61 top
 7 root       20   0      0      0      0 I  0.3  0.0 0:00.74 kworker+
13 root       20   0      0      0      0 I  0.3  0.0 0:00.34 kworker+

```

(a) Hvor mange prosent av maksimalkraft prosessoren kjører på.



(b) Graf av temperaturen til prosessoren.

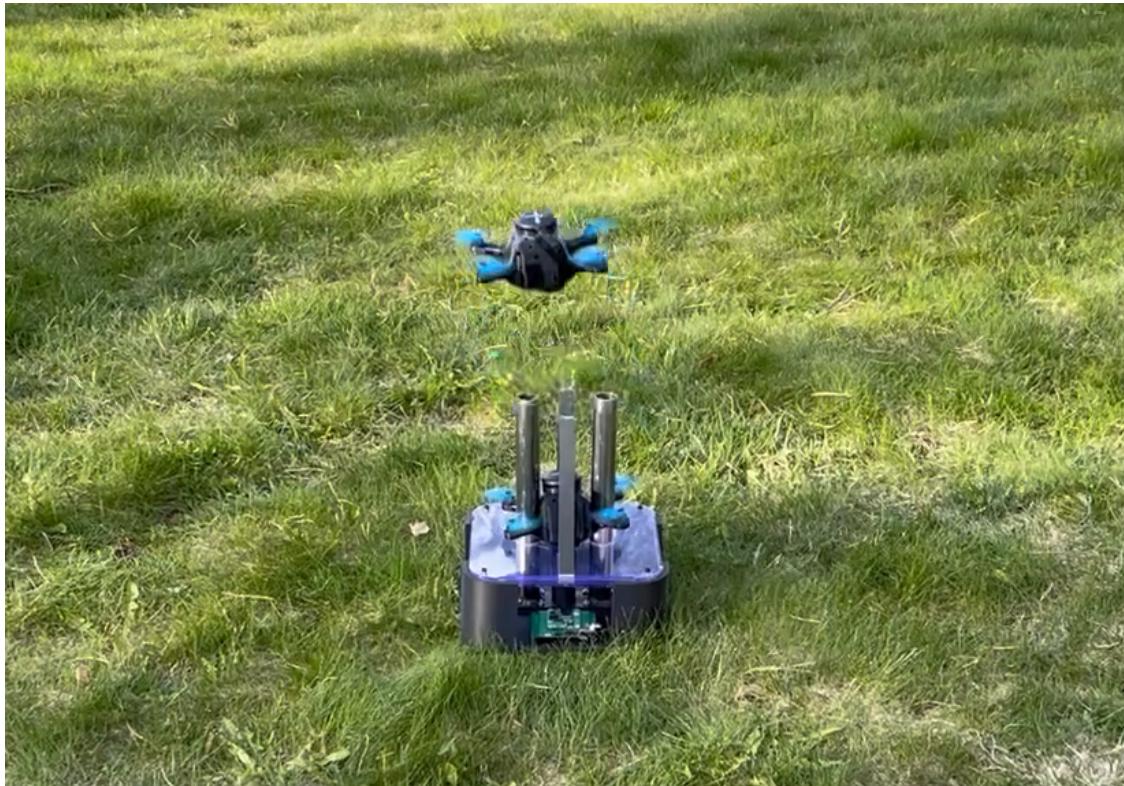
Figur 9.8: Etter endringer gjort på micrortps-agenten.

Ny test etter utbedring

Testen ble gjennomført på nytt etter endringene forklart i Testing-kapittel 8.4.4. Etter disse endringene ble alle test-kriteriene oppfylt og testen betraktes som godkjent.

9.3.5 Fly to droner ut av utskytningsenhet

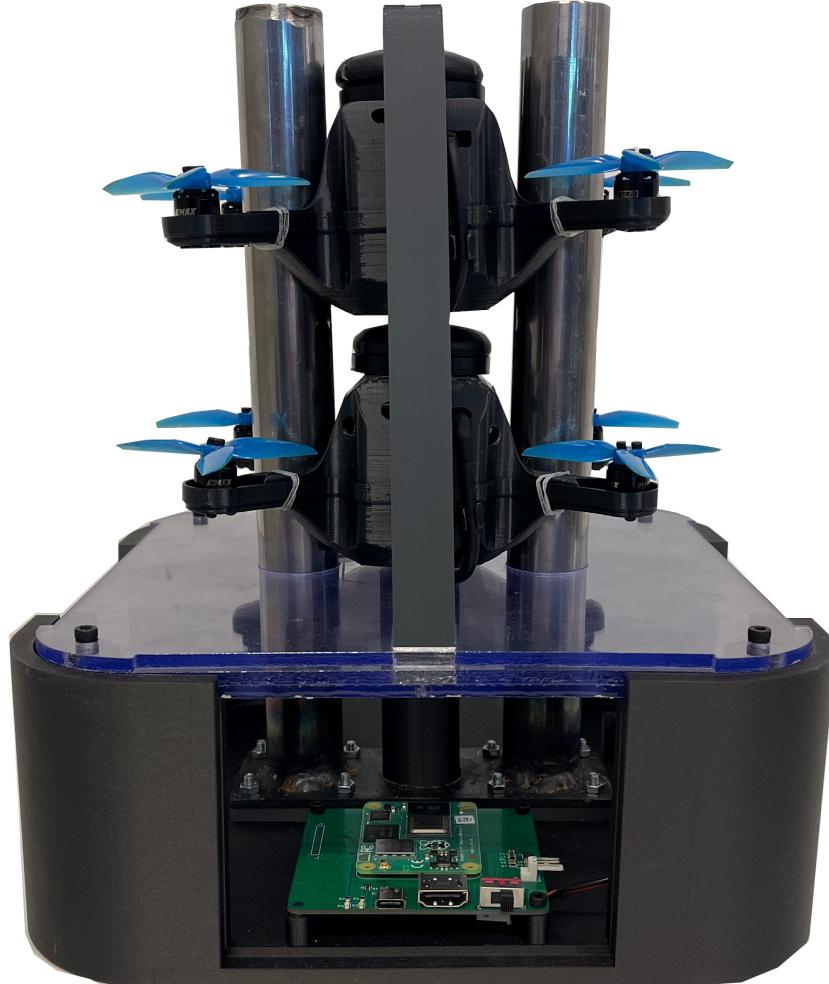
Test-kriteriene for denne testen var at begge dronene skulle ta av fra utskytningsenheten uten problemer, etter å ha sendt launch kommandoer fra bakkestasjonen. Begge dronene ble armet ved å skrive 999 999 arm i kommando-vinduet, deretter ble den første dronen launchet ved å skrive 01 01 launch og rundt 1,5 s etterpå ble den andre dronen launchet ved å skrive 01 02 launch. Siden begge dronene klarte å ta av uten problemer betraktes testen som godkjent.



Figur 9.9: Fly to droner ut av utskytningsenhet i utenbordskontroll modus.

9.4 Det automatiske utskytingssystemet

Første iterasjon av et automatisk utskytingssystem for dronesverm er laget og testet. Fra testene som er utført er det verifisert at konseptet som ble valgt har vist seg å fungere som planlagt. Fra testingen av det automatiske utskytingssystemet for dronesverm vil dronene kunne lette ut fra utskytningsenhetene med mindre enn 1,5 sekunders mellomrom, som vil si at det tar under 15 sekunder å tømme en full utskytningsenhet. Det vil også si at 100 droner kan ta av i løpet av 15 sekunder ut fra 10 utskytningsenheter. Opprigningen av systemet har også vist seg å foregå raskt, siden utskytningsenheten er omrent like enkel å plassere ut som én drone. Det tar derfor omrent 10 ganger så kort tid å plassere ut 10 utskytningsenheter i forhold til å plassere ut 100 droner.



Figur 9.10: Utskytningsenhet med to droner.

Det automatiske utskytingssystemet er satt opp som er hierarkisk modell for å spre datakraften brukt av systemet utover flere enheter, bakkestasjonen, utskytningsenhetene og dronene. Dette er blitt gjennomført gjennom bruken av ROS 2. Systemet består nå av totalt tre ROS 2-noder, hvor nodene i utskytningsenheten og dronen kan brukes flere ganger ved bruk av navnerom. Det er gjort ved å ha Python-skript som henter informasjon om hvilken enhet den er på, og starter noder med navnerom ut i fra den gitte informasjonen. For å kontrollere systemet blir bakkestasjonen brukt som hovedsentral, fra bakkestasjonen er det mulig å kontrollere alle dronene, som vil si at hele systemet kan styres av én person.

All kode og programvare som blir brukt av det automatiske utskytningssystemet har åpen kildekode. For at systemets enheter skal kommunisere trådløst blir Wi-Fi benyttet. Ved testing av systemet har et hotspot blitt satt opp og har derfor hatt en maks rekkevidde på omtrent 20 meter.

For demonstrasjonsvideo av det automatiske utskytningssystemet, se linken nedenfor:

- <https://youtu.be/gGmkMiGq30I>

Kapittel 10

Diskusjon

10.1 Testing

10.1.1 Dronen slet med å fly

I den første fysiske testen slet dronen med å fly høyere enn rundt 40 cm over bakken. Det antas at problemet er relatert til den økte vekten på dronen, 324 g. Dette er 74 g mer enn den planlagte vekten. Grunnen til denne ekstra vekten er at dronen har autopilot-maskinvare, ettkortsdatamaskin, GPS, radio-mottaker og telemetri-modul. Autopiloten betraktes i denne oppgaven som testutstyr. For det overordnede prosjektet er det i lengden ønskelig å kritte seg med autopilot-maskinvaren, og heller kombinere denne med ettkortsdatamaskinen.

Det kan også for det overordnede prosjektet være interessant å se på muligheten til å fly uten GPS, eller eventuelt bruke en som er mindre. Radio-mottaker og telemetri-modul er ikke nødvendig å ha i dronen, men er hendig med tanke på sikkerhet og justeringer underveis i testingen.

Siden problemet mest sannsynlig er vekt, og mange overflødige komponenter sett fra det overordnede prosjektet sitt perspektiv, blir vektøkningen godtatt. Derfor blir det sett på alternativer som kan gjøre at testingen av systemet kan fortsette med de nevnte komponentene ombord. Siden dronen flyr med 2-blads propeller, ble det som vist i Figur 10.1 forsøkt å bytte til propeller med 3 blader. Dette er en løsning som er enkel å gjennomføre, og dersom det er vellykket kan testingen fortsette uten at det er mistet for mye tid.



Figur 10.1: Gamle og nye propeller.

10.1.2 Fly ut av utskytningsenhet

Når dronen skulle fly ut av utskytningsenheten ble det observert at propellene såvidt var i kontakt med fôringstolpene. Dette er ikke kritisk, siden propellene ikke tok nevneverdig skade, men det er ikke optimalt. Det bør derfor i de neste iterasjonene utarbeides en løsning for dette problemet. *Dette er diskutert videre i kapittel 12.1.1.*

Siden hastigheten til dronen ikke er spesifisert i utførelsen av testen, ble det forsøkt med ulike hastighet under testing. Det ble observert at dronen fløy lettare ut av utskytningsenheten med høyere hastigheter i motsetning til lave hastigheter. Dette er tatt i betrakning videre i testingen.

10.1.3 Ustabilt system ved flyging i utenbordskontroll-modus

Systemet var svært ustabilt under testen. I ustabilt ligger det at det var vilkårlig når dronen hørte på kommandoene sendt fra bakkestasjon. Det ble også forsøkt å arme dronen fra radio-kontrolleren, dette også uten respons.

Ved bruk av RQt verktøyet ble det verifisert at meldingene ble sendt fra RPi CM4 til autopiloten. Siden programvaren fungerte uten problemer i simuleringen, ble det antatt å være et fysisk problem. Det har ved flere anledninger blitt lagt merke til at autopilot-maskinvaren har vært varm, i den grad at den ikke kan holdes på. Det er ønskelig å undersøke dette videre. *Se kapittel 8.4.4 for utbedring av feil.*

Utbedring av varmeproblem

Resultatet av utbedringen viser en stor forbedring. Prosessoren brukte 90 % mindre prosessor-kapasitet og temperaturen ser ut til å ha stabilisert seg på litt over 60°C. Denne utbedringen kan ha positive konsekvenser også for autopiloten. *Se kapittel 9.3.4 for resultat av påfølgende test.*

10.2 Drone

Dronen er videreutviklet fra tidligere oppgaver, posisjonene til motorinnfestningene er beholdt og hoveddelen av skroget er tilpasset årets komponenter. Komponentene som sitter i dronen er valgt med bakgrunn i kravet om at dronen skal være flygbar (3.1.6). Det er essensielt at dronen kan fly for å kunne teste det automatiske utskytningssystemet.

Det er valgt å inkludere en autopilot som forenkler flyging og stabilisering av dronen. Valget av autopilot-maskinvare falt på Pixhawk 4 mini, siden det var best kjennskap til Pixhawk-standarden i gruppen og blant ressurser på UiA. I forbindelse med denne maskinvaren var det naturlig å bruke PX4 Autopilot som programvare, ettersom disse komplementerer hverandre. På denne måten sparte gruppen mest sannsynlig en del tid på å unngå utfordringer med kompatibilitet mellom maskinvare og programvare. Å bruke Pixhawk 4 mini i kombinasjon med PX4 Autopilot har vist seg å være et godt valg. Det har i løpet av prosjektet vært enkelt å løse utfordringer som har dukket opp på grunn av god dokumentasjon og gode ressurser på UiA. Ved enkelte tilfeller har det også vært nødvendig å kontakte utviklerne direkte for å få svar på mer spesifikke utfordringer. Gjennom chatte-applikasjoner og forum har det vært enkelt å opprette kontakt med utviklerne, og andre bidragsytere til prosjektet der gruppen har fått svar på alle spørsmål.

Oppgraderingen av ettkortsdatamaskinen til 64-bit gjorde det mulig å benytte seg av ROS 2 til programvareutvikling. ROS 2 gjør det enklere å endre eller legge til funksjoner i systemet senere for det overordnede prosjektet. Dette gjør systemet svært modulært og møter kriteriet om å benytte ROS (3.1.4). Til ettkortsdatamaskinen ble det i prosjektet designet og laget bærekort. Bærekortet er utstyrt med de nødvendige komponenter for å kommunisere med autopilot-maskinvaren. Ettkortsdatamaskinen har innebygd Wi-Fi som gjør det mulig å kommunisere med utskytningsenheten. Ved å benytte Wi-Fi er kriteriet om å benytte ISM-bånd til kommunikasjon godkjent (3.1.2). Bærekortet fungerer som forventet og har et stort potensiale videre i prosjektet på grunn av muligheten til å kunne tilpasse bærekortet etter behov. *Bærekort og elektronikk for videre arbeid er diskutert grundigere i kapittel 12.1.2.*

De tidligere gruppene har gjort en grundig jobb med valg av komponenter som motorer, motorregulatorer, batteri og propeller, så disse ble brukt videre, foruten propellene. Som diskutert i kapittel 10.1.1 slet dronen med å stige. Dette ble løst ved å bytte til propeller med 3 blader i stedet for 2. Videre i prosjektet er det ikke nødvendigvis disse 3-blads propellene som er de mest egnede. Dersom vekten på dronen møter vekt-kriteriet kan propellene med 2 blader fungere og gi lengre flytid.

Inkludert alle komponentene som ble brukt i forbindelse med testing, ble totalvekten på dronen 324 g. Vekt-kriteriet for dronen er på 250 g (3.1.2). Med dronens 324 g er ikke vekt-kriteriet godkjent. Vekten er i forhold til Luftfartstilsynet sitt regelverket fortsatt innenfor, men den ønskede underkategorien er ikke møtt. Med sine 324 g klassifiseres dronen i underkategori A3. To av gruppemedlemmene har sertifikat til å fly droner i denne kategorien. Det vil si at testingen har vært innenfor regelverket. Det er i lengden ønskelig å være innenfor underkategori A1, da må dronens vekt være mindre enn 250 g. I denne kategorien er det mindre krav til pilot, og den kan anses som leketøy. Dersom dronen anses som leketøy kan den flys uten sertifikater.

Når det gjelder den høye vekten på dronen kan det argumenteres med at inkluderingen av autopilot-maskinvaren gjorde terskelen for flyging lavere. Dette førte til at det automatiske systemet kunne testes, forklart i kapittel 8. For det overordnede prosjektet sin del kan et fungerende system være et godt utgangspunkt for videre utvikling av dronen. Det er tenkt at flere av komponentene i dronen kan kombineres eller fjernes fullstendig i fremtiden. Blant annet kan autopilot-maskinvaren og/eller -programvaren kombineres med bærekortet til ettkortsdatamaskinen. Dette kan i seg selv spare rundt 37 g, i tillegg til å redusere størrelsen på hoveddelen av skroget.

Overflødige komponenter i dronen er radio-mottakeren og telemetri-modulen. Disse ble kun benyttet under testing, og er ikke nødvendige for systemets funksjon. GPS-mottakeren er også unødvendig stor, det kunne blitt brukt en mindre enhet, men denne fulgte med Pixhawk 4 minien og ble valgt på grunn av kompatibilitet. Vektmessig veier GPS-en 32 g, radio-mottakeren 6 g og telemetri-modulen 4 g, totalt blir dette 42 g. Ved å kombinere autopilot-maskinvaren (37 g) og GPSen med bærekortet, redusere størrelsen på skroget og fjerne de overflødige komponentene er det rimelig å anta en vektbesparelse på rundt 90 g. Dronen ender da opp på 234 g som er godt innenfor vekt-kriteriet. *Kombineringen av komponenter blir diskutert videre i kapittelet om videre arbeid 12.1.2.*

Når det gjelder videre produksjon og produksjonsmetoder av dronene, bør det etter hvert begynne å se på andre metoder enn 3D-printing. I de tidligere rapportene har vakuumforming (2.3.3) blitt nevnt. Med noen tilpasninger til det nye skroget for innfestningen av strømmodulen, vil vakuumforming også være en metode å vurdere. En annen metode som kan være interessant å kikke på er sprøytestøping (2.3.4). Begge metoder er egnet for produksjon i større skala.

Kostnaden å produsere én drone er ikke veldig representativt med tanke på at dette er en prototype. Ved fullskala-produksjon måtte det blitt utarbeidet avtaler med ulike produsenter for reduserte priser på komponentene. Andre faktorer er at autopilot-maskinvaren koster svært mye i forhold til resten av komponentene, inkludert GPS og strømmodul havner den på rundt kr. 1.500. Produksjonskostnaden på dronen slik den er i dag kan anslås til å ligge på mellom kr. 2.500 og kr. 3.000 regnet uten arbeid. Ved å lage 100 droner på denne måten vil totalkostnaden i verste fall havne på kr. 300.000.

Siden autopiloten betraktes som testutstyr, og det er planlagt å fjerne denne fra dronen senere i prosjektet, kan regnestykke også gjøres uten autopiloten. Da havner totalkostnaden til en drone på mellom kr. 1.000 og kr. 1.500, regnet uten arbeid. Dette er mer representativt for det overordnede prosjektet. Ved å fjerne autopiloten må bærekortet utvikles til å ta over oppgavene autopiloten har hatt. Dette gjør at kostnaden på bærekortet øker, men sammenlignet med autopiloten er det sannsynligvis ikke nevneverdig.

Kostnadsliste for alle deler til å konstruere en drone ligger i appendiks G.1.

10.3 Utskytningsenhet

Utskytningsenheten er satt sammen av 3D-printede deler, pleksiglassplater og føringsrør i stål. Gjennom ulike manuelle tester underveis i utviklingen, har enheten vist seg å fungere som planlagt. Siden dette er en prototype har det ikke blitt lagt mye arbeid i optimalisering av design, det er i stedet for valgt å fokusere på å lage en fungerende prototype. Selv om utskytningsenheten er en tilsvarende statisk del av systemet, er den ansvarlig for å vidersende meldinger fra bakkestasjonen til sine 10 droner ved å bruke ettkortsdatamaskinen som ligger i elektronikkboksen.

Ettkortsdatamaskinen trenger et bærekort. Ved design av dette ble det lagt vekt på å oppfylle alle de ønskede funksjonalitetene. Blant annet skulle ettkortsdatamaskinen forsynes med strøm fra et LiPo-batteri og ha porter for enklere tilgang til ettkortsdatamaskinen. LiPo-batteriet er koblet rett i bærekortet og spenningsregulator-kretsen, uten noen form for overvåking av kapasitet. Dette kan bli et problem dersom batteriet utlades til under anbefalt minimum kapasitet. Derfor bør det utarbeides en tryggere løsning på å forsyne datamaskinen med strøm. Spenningsregulator-kretsen på bærekortet er identisk til kretsen brukt på bærekortet til dronen, alle komponentene ble også bestilt samtidig fra samme leverandør. Produksjonen foregikk på samme måte som beskrevet i kapittel 5.3.4. Derimot ble det oppdaget problemer under testing av kortet da RPi CM4 ikke ville starte opp korrekt.

Spenningen inn til datamaskinen ble målt til 4.84 V, som er for lite til at datamaskinen kan starte opp korrekt. Det er usikert hvorfor dette problemet eksisterer, siden det er brukt identisk krets, komponenter og produksjonsmetode som på bærekortet til dronen. Det ble produsert tre eksemplarer dette kretskortet i løpet av prosjektet, av den grunn kan en feil i produksjon mest sannsynlig utelukkes. Av hensyn til problemet ble ROS 2-noden til utskytningsenheten i stedet for kjørt på samme datamaskinen som ROS 2-noden til bakkestasjonen. Det er ikke sikkert at å lage bærekort i utskytningsenheten var nødvendig, for eksempel kunne det i stedet for blitt benyttet en standard Raspberry Pi 4. *Dette er diskutert mer i videre arbeid kapittel 12.2.2.*

Elektronikkboksen har en del tilgjengelig plass. Den er designet slik for at det skal være enkelt å legge til nye funksjonaliteter. Det kan for eksempel i lengden være ønskelig å implementere lading til dronene, for å gjøre dette kreves det mest sannsynlig en balanserings-krets og en måte å få nok strøm inn til enheten. Slik elektronikkboksen er designet kan det relativt enkelt implementeres uten å måtte lage en helt ny boks. Dette er tids- og kostnadsbesparende for den videre utviklingen av enheten. *Dette er diskutert mer i videre arbeid kapittel 12.2.1.*

Vektmessig havnet hele utskytningsenheten på rundt 2600 g. I forhold til vekt-kriteriet i kapittel 3.1.2 er enheten lastet med 10 droner på maksimalt 250 g hver innenfor kravet til Arbeidstilsynet [49]. Kravet om å være innenfor regelverket er derfor oppfylt.

Produksjonsmessig for utskytningsenheten bør det kikkles på tidsbesparende produksjonsmetoder for elektronikkboksen og lokket. Elektronikkboksen ble for dette prosjektet 3D-printet siden det er en prototype. Størrelsen gjorde at det tok underkant av en uke å få den printet, som er altfor lenge med tanke på masseproduksjon. Lokket til utskytningsenheten ble som nevnt tidligere ikke produsert, det er fordi det ikke ble sett på som kritisk, men også fordi den mest sannsynlig ville tatt over en uke å få produsert. Det bør derfor utarbeides nye produksjonsmetoder som kan senke produksjonstiden.

Ettersom dette er en prototype er det ikke representativt å beregne produksjonskostnad basert på denne utskytningsenheten. Før masseproduksjon av enhet bør designet optimaliseres for billigst mulig produksjon. I tillegg bør det utarbeides avtaler for rimeligere innkjøp av komponenter. Det kan anslås at kostnaden på én utskytningsenhet slik den ser ut i dag havner på mellom kr. 3.500 og kr. 4.000, regnet uten arbeid. Dersom det produseres 10 stykker blir det maksimalt kr. 40.000.

Kostnadsliste for alle deler til å konstruere en utskytningsenhet ligger i appendiks G.2.

10.4 Det automatiske utskytingssystemet

Det automatiske utskytingssystemet har gjennom ulike tester i kapittel 8 vist seg å fungere som planlagt. I henhold til kravet om at dronen skal kunne fly, kan det betraktes som oppfylt, siden samtlige tester er vellykket (3.1.6). Systemet er effektivt i den forstand at det kan utplasseres på kort tid, og at det kan opereres av én person. Dette er mulig ved at systemet blant annet styres fra en sentral som møter kriteriet til kravet om at systemet skal være modulært og skalerbart (3.1.4).

Utplasseringen av en utskytningsenhet har vist seg å være like enkel som utplasseringen av en drone. Det vil si at opprigg av systemet er 10 ganger så raskt som å plassere ut dronene manuelt. Tids-kriteriet for at systemet skal være effektivt å rigge opp er at det skal kunne gjøres på under 15 minutter (3.1.1). Det er rimelig å anta at én person klarer å plassere ut 10 utskytningsenheter på 15 minutter.

Dronene kan lette fra utskytningsenheten med mindre enn 1,5 s mellomrom. Det vil si at alle dronene kan lette fra enheten innen 15 s. Landingen av dronene foregår manuelt, men det er ingenting som står i veien for at dronene, med forhåndsdefinerte baner, kan lande automatisk på et dedikert landingsområde. Med tanke på at det er en prototype som ble produsert har det vært vanskelig å definere hvor lang tid landing av en dronesverm på opptil 100 droner vil ta. Derimot har testingen av systemet vist at tids-kriteriet om at landing må skje på under 2 minutter mest sannsynlig vil oppfylles. Nedrigningen går også svært hurtig, som følge av at det er enkelt å legge dronene tilbake i utskytningsenheten. Med dette godkjennes kriteriene til underkravet om et tidseffektivt system. Dette oppfyller toppkravet om at systemet må være enkelt og effektivt å bruke (3.1.1).

Det automatiske utskytingssystemet benytter seg av Wi-Fi til kommunikasjon. Dette gjør at kriteriet om ISM-bånd er godkjent, og kravet om å benytte ulisensiert radio-kommunikasjon er oppfylt. Videre er hoveddelen av programvaren til systemet utviklet med Python og ROS 2, automatiserings-skript er skrevet i Python og autopilot-programvaren er åpen kildekode. Med dette er både kriteriet om å benytte seg av ROS (3.1.4) godkjent, og underkravet om at all programvaren bør ha åpen kildekode oppfylt (3.1.2).

Systemet er designet for å være skalerbart. Testingen har foregått med to droner, men programvaren er skrevet slik at den enkelt kan ekspanderes til og med 100 droner. Med 100 droner begynner systemet å bli vanskeligere å kontrollere. Det er fordi brukeren må manuelt sette ønsket posisjon eller hastighet, men med små endringer i programvaren til bakkestasjonen kan dette automatiseres. Brukeren kan via bakkestasjonen kontrollere dronene hver for seg, eller alle samtidig. Kriteriet om at systemet må fungere med mange droner og være brukervennlig er godkjent, dette fører til at kravet om at systemet skal være modulært og skalerbart er oppfylt (3.1.4).

Systemet har totalt en vekt på rundt 5100 g, dette er innenfor kriteriet, og vekt-kravet godkjennes. Toppkravet om lisenser og sertifikater godkjennes siden underkravene om ulisensiert radio-kommunikasjon, vekt i forhold til Luftfartstilsynet og Arbeidstilsynet, og åpen kildekode er oppfylt (3.1.2).

Når det gjelder totalkostnad for produksjon av systemet, er kriteriet at den skulle være under kr. 300.000. Ut i fra kostnadene på dronene og utskytningsenhetene ender totalkostnaden maksimalt på kr. 340.000. Som diskutert er dette prototyper. Kostnaden for å produsere enheter i liten skala er stort sett høyere enn i stor skala på grunn av samarbeidsavtaler ved innkjøp av komponenter. Dersom prosjektet videre også klarer å kombinere autopilot-maskinvaren med bærekortet kan kostnaden på dronen kuttes betraktelig, og kriteriet om totalkostnad vil bli godkjent. Siden det er lagt opp til at prosjektet kan redusere kostnaden, betraktes toppkravet om at kostnadene for å produsere systemet bør være så lave som mulig som oppfylt (3.1.3).

Det er i kravet om kommunikasjon satt kriterier på rekkevidde og datarate (3.1.5). Disse kriteriene har blitt nedprioritert på grunn av tid. Under testing ble det benyttet et mobilt hotspot for Wi-Fi kommunikasjon. Dette fungerte greit, siden testingen foregikk med én utskytningsenhet og to droner. Ved større svermer over større områder bør det utarbeides en bedre løsning, siden et mobilt hotspot har begrenset rekkevidde. Kravet om at systemet må kunne sende data med høy rate over et stort område kan derfor ikke betraktes som oppfylt.

På ettkortsdatamaskinene er oppstartsprosessen automatisert. Ved oppstart blir brukeren automatisk logget inn og oppstarts-skriptet til den aktuelle enheten kjøres fra ".bashrc". Å logge inn automatisk er ikke nødvendigvis den beste løsningen. Det kan være uheldig dersom ettkortsdatamaskinen havner i gale hender. Dersom dette skjer er det mulig å sette RPi CM4 på et IO-brett, koble til en skjerm og tastatur. Brukeren vil automatisk bli logget inn og få tilgang til alle filer. En bedre løsning kunne vært å kjørt oppstarts-skriptene uten å måtte logge inn.

Kapittel 11

Konklusjon

Det automatiske utskytningssystemet for dronesverm er designet for å fungere med opptil 100 droner. Systemet kan opereres av én person, og samtlige droner kan lette på under 15 sekunder. For å oppnå dette er systemet delt inn i tre delsystemer; bakkestasjonen, utskytningsenheten og dronen. Bakkestasjonen er brukerens grensesnitt til utskytningssystemet. Fra bakkestasjonen kan brukeren både kommunisere med og kontrollere samtlige droner. Kommunikasjonen foregår gjennom utskytningsenhetene som mottar, analyserer og videresender informasjon fra bakkestasjonen til sine respektive droner. Hver utskytningsenhet rommer opp til 10 droner, og den kommuniserer kun med disse. En slik arkitektur gjør systemet brukervennlig og skalerbart.

Dronen som er brukt i systemet er en videreutvikling fra tidligere oppgaver. Det er valgt å implementere en autopilot for å kunne teste løsningen av det automatiske utskytningssystemet. For å kontrollere autopiloten, er dronens elektronikk-pakke oppgradert. Utskytningsenheten er et nytt tilskudd til det overordnede prosjektet. Den fungerer både som oppbevaring og utskytningsplattform for dronene. For å tilpasses de nye komponentene og utskytningsenheten, er dronens skrog re-designet.

Programvaren til det automatiske utskytningssystemet er bygget opp på ROS 2. Dette gjør systemet modulært og danner et godt grunnlag for god kommunikasjon mellom delsystemene. ROS 2 gjør også systemet enklere å videreutvikle.

Det er produsert en prototype av det automatiske utskytningssystemet som er brukt til å teste og verifisere løsningen. Prototypen består av én utskytningsenhet og to droner. Resultatet av testingen har vist at det automatiske utskytningssystemet fungerer som planlagt. Dronene flyr automatisk ut av utskytningsenheten på en kontrollert måte, og kan landes på et dedikert landingsområde. Derfor anses det utviklede systemet som godt egnet til bruk i sverm-eksperimenter.

Kapittel 12

Videre arbeid

I løpet av prosjektperioden er det bygd opp mye erfaring med droner og dronesvermer. Derfor er det reflektert rundt hva som bør, og kan jobbes med for å videreutvikle det automatiske utskytningssystemet. I dette kapittelet blir det anbefalt hva som bør inngå i det videre arbeidet på systemet.

12.1 Drone

12.1.1 Tilpasser droneskrog

Under testingen tippet dronen nok til at propellene traff føringsstolpene når den fløy ut av utskytningsenheten, se kapittel 9.3.3. Derfor bør det arbeides med å tilpasse droneskroget bedre til føringsstolpene. Det kan bli gjort ved å undersøke om føringsstolpene kan ha mindre diameter, eller om droneskroget kan tilpasses. En annen endring som kan forhindre at propellene treffer føringsstolpene er å re-designe armene til dronen. Ved å gjøre dette kan propellene ha større avstand til føringsstolpene.

12.1.2 Kombinere elektroniske komponenter

Ved montering av komponentene på de produserte test-dronene er komponentene blitt loddet eller festet med konnektorer. Siden det er mange komponenter i dronen har det resultert i mange ledninger. Dette har gjort det vanskelig å plassere komponentene inn i dronen og få på lokket. For å løse dette problemet er funnet ut av at den enkleste monteringen mest sannsynlig ville vært å ha de fleste komponentene plassert på ett eller flere kretskort som var tilpasset droneskroget.

På grunn av at alle komponenter og all programvare som er brukt i dronen er lisensfrie og har åpen kildekode, kan en mulighet være å kontakte produsentene for å få tilgang til komponenter og kretsskjema for å integrere disse inn i et eget kretskort. Pixhawk har mye informasjon tilgjengelig på nettsidene. De har blant annet publisert en standard på hvordan en Pixhawk kompatibel autopilot er satt oppⁱ. Det samme gjelder for motorregulatorene som er brukt i dronen. De er designet av nordmannen Steffen Skaug, som kan være interessert med å hjelpe til. BLHeli har allerede mye tilgjengelig på Githubⁱⁱ. For å gi strøm til autopiloten og motorregulatorene en krets lik strømmodulen blitt designet. For å enkelt bytte ut motorene kan de fortsatt være festet på skroget, men ha konnektorer som går direkte inn på kretskortet. I tillegg kan andre ønskede funksjoner som for eksempel kamera og/eller gimbal bli integrert i kretskortet.

En annen løsning kan være å fortsette med dronen fra de tidligere bachelorprosjektene. De elektriske kretsene de laget kan implementeres på bærekortet som ble designet i denne oppgaven. Dette vil resultere i vekt- og plassbesparelse. Kretskortet kan bli tilpasses droneskroget i stedet

ⁱ<https://github.com/pixhawk/Pixhawk-Standards/blob/master/DS-011%20Pixhawk%20Autopilot%20v5X%20Standard.pdf>

ⁱⁱhttps://github.com/bitdump/BLHeli/tree/master/BLHeli_32%20ARM

for å designe skroget etter komponentene. Ved å kontakte et firma som driver med produksjon av kretskort kan komponentene bli plassert på begge sider av kretskortet. Dette for å ha en mer kompakt drone.

En av disse løsningene bør være mulig å få til om de rette ressursene på universitetet blir brukt. Om mekatronikk-linjen lager en plan, og designer et fungerende system. Så kan elektronikk-linjen designe skjematiske og kretskortutlegget med hensyn på støy og andre faktorer som kan påvirke komponentene.

12.2 Utskytningsenhet

12.2.1 Videreutvikle funksjonaliteter

Utskytningsenheten har potensiale til å ha en rekke funksjoner som vil være behjelpeelige ved bruk i en dronesverm. Utviklingen av utskytningsenhetens funksjonaliteter vil gå i mindre steg, og det er derfor vurdert hvilke funksjonaliteter som bør legges til i de neste iterasjonene.

Det ville vært interessant å sett på muligheten til å lande dronene i utskytningsenheten etter endt flyging. Ved å enten implementere et kamera som ser nedover i dronene, eller et kamera som ser oppover i utskytningsenheten bør det være mulig å kontrollere posisjonen til dronen, og da lande i utskytningsenheten.

Utskytningsenheten bør også oppgraderes så den er mer automatisert. Det kan bli gjort ved å utstyre lokket med motorer som åpner utskytningsenheten før utskyting

For å få et mer automatisk system ville det også vært interessant å jobbet med en ladeløsning til dronene. I begynnelsen av oppgaven ble det jobbet med en ladekrets, men da dette er første iterasjon av utskytningsenheten ble det bestemt å teste hovedfunksjonaliteten først. Det ble sett på muligheter til induksjons- og kontakt-lading. Begge har fordeler og ulemper. Induksjonslading gjør det enklere å vantette dronen, mens kontaktlading vil mest sannsynlig lade dronene forttere.

12.2.2 Ettkortsdatamaskin og nettverk

Ettkortsdatamaskinen som er montert i utskytningsenheten er nå av samme type som sitter i dronen. Valget ble tatt da det ville vært uheldig å ha for mange systemer å sette seg inn i. Sett tilbake på ble valget tatt på helt feil grunnlag. Om valget skulle bli tatt igjen, ville en annen datamaskin blitt valgt. Fra undersøking som er gjort ville det mest sannsynlig valgt en standard Raspberry Pi 4 eller en Nvidia Jetson Nano i stedet for.

Det vil også være interessant å se på mulighetene til å opprette et lokalt kommunikasjonsnettverk, og bruke føringssstolpene som antenner. Om noe slikt blir gjort vil det være lurt å opprette et mesh-nettverk. På den måten kan utskytningsenhetene og dronene være enheter på det store nettet, og sørge for god rekkevidde.

12.3 Det automatiske utskytningssystemet

For å ha et mer komplett system til bruk i dronesverm-eksperimenter bør kontrollen av dronene jobbes videre med. I denne oppgaven er det jobbet med utskytning av dronene, men det er lagt til rette for at et program kan kontrollere dronen ved å publisere til `use_drone_setpoint` emnet. Derfor bør det jobbes med å lage et program som enten kan lage forhåndsbestemte baner, eller kontrollere dronen med et autonomt system som for eksempel UiA Motion Lab, eller ved hjelp av kamera.

Programvaren til utskytningsenheten og dronen ble skrevet nokså dynamisk i dette prosjektet, men kunne vært enda mer dynamisk ved å navngitt dronene smartere. I stedet for å navngi de med en tekstfil som legges inn på systemet, burde de blitt navngitt etter informasjon som allerede finnes. Eksempelvis kan dronene navngis ved å finne posisjonen til dronen i utskytningsenheten. Løsninger for dette kan være ved å bruke ladekretsen til å finne posisjonen til dronene i utskytningsenheten, eller bruke et nøyaktig barometer til å finne høyden.

Ordliste

COM En ettkortsdatamaskin uten innganger og utganger.

ESC Elektrisk krets som kontrollerer og regulerer motorhastighet.

I2C En seriell buss kommunikasjonsprotokoll.

ISM-bånd Lisensfrie radiobånd til industriell, forskning og medisinsk bruk.

KiCad Schematic Capture and PCB Design Software.

kretskort Kort bestående av elektroniske komponenter og elektriske kretser.

MAVLINK Er en lettvekts kommunikasjonsprotokoll.

SPI En synkron seriell buss kommunikasjonsprotokoll.

Utskytningsenhet Oppbevaring- og kommunikasjonmellomledd for dronene.

Akronymer og forkortelser

ADC Analog-to-digital converter.

API Application Programming Interface.

BPI Banana Pi Zero M.2.

CAD Computer Aided-Design.

CAN Controller Area Network.

CPU central prosessing unit.

DC-DC DC-to-DC converter.

DDS Data Distribution Service.

DDSI-RTPS DDS Interoperability Wire Protocol.

eMMC embedded Multi-Media Controller.

EU Den europeiske union.

EØS Det europeiske økonomiske samarbeidsrådet.

GPIO General-purpose input/output.

GPS Global Positioning System.

I/O Input/Output.

LED Light Emitting Diode.

LiPo Lithium Polymer.

MISO Master In Slave Out.

MOSFET Metal Oxide Semiconductor Field Effect Transistor.

MOSI Master Out Slave In.

OSI Open Systems Interconnection.

PWM Pulse-width modulation.

PX4 PX4-Autopilot.

RAM random access memory.

ROS Robot Operating System.

ROS 2 Robot Operating System 2.

RPi Raspberry Pi.

RPi 4 Raspberry Pi 4.

RPi CM4 Raspberry Pi Compute Module 4.

RPi OS Raspberry Pi OS.

RPi Zero W Raspberry Pi Zero W.

RTOS Real Time Operating System.

RTPS Real Time Publish and Subscribe.

SBC Single-board Computer.

SD Secure Digital.

SoC system on a chip.

SSH Secure Shell.

STL Stereolithography.

UiA Universitetet i Agder.

UML Unified Modeling Language.

VM virtuell maskin.

Bibliografi

- [1] Federico Augugliaro Markus Waibel Bill Keays. «Drone shows: Creative potential and best practices». I: ETH Zürich (2017). DOI: <https://doi.org/10.3929/ethz-a-010831954>.
- [2] Guinness World Records. Biggest drone display ever! - Guinness World Records. URL: <https://www.youtube.com/watch?v=44KvHwRHb3A>. (Lastet ned: 25.05.2021).
- [3] Jan Petter Ottesen Ola Christoffer Våge Jørgen Mikal Benum Simon Benum. «Modulbasert drone tilpasset autonom bruk i Motion Capture system». I: UiA, 2018.
- [4] Marius Kruithof Kristoffer Hansen Egeland. «Modulær drone for svermekspimenter i UiA Motion Lab». I: UiA, 2019.
- [5] PX4. Pixhawk 4 Mini. URL: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4_mini.html. (Lastet ned: 08.02.2021).
- [6] Dronecode. Offboard Mode. URL: https://docs.px4.io/master/en/flight_modes/offboard.html. (Lastet ned: 12.05.2021).
- [7] Luis M Soria-Morillo et al. «Single-Board-Computer Clusters for Cloudlet Computing in Internet of Things». I: Switzerland: MDPI AG (2019). DOI: <https://doi.org/10.3390/s19133026>.
- [8] Alexander Bachmutsky. System Design for Telecommunication Gateways. John Wiley Sons, 2011. ISBN: 9781119956426.
- [9] Hassan Gomaa. Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures. New York: Cambridge University Press, 2011. ISBN: 9780521764148.
- [10] Sams Development Team ; [written by UNIX Wizards ; foreword by Mike Azzara]. UNIX unleashed. Sams Publ., 1994. ISBN: 0672304023.
- [11] Marco Cesati Daniel P. Bovet. Understanding the Linux Kernel. O'Reilly Media, Inc, 2005. ISBN: 9780596554910.
- [12] Canonical. About the Ubuntu project. URL: <https://ubuntu.com/about>. (Lastet ned: 24.02.2021).
- [13] Canonical. Ubuntu for desktops. URL: <https://ubuntu.com/desktop>. (Lastet ned: 24.02.2021).
- [14] Canonical. Scale out with Ubuntu Server. URL: <https://ubuntu.com/server>. (Lastet ned: 24.02.2021).
- [15] Canonical. Ubuntu Core 20. URL: https://assets.ubuntu.com/v1/b2c770ea-Ubuntu+Core20+Datasheet.pdf?_ga=2.39090470.465564440.1614163601-956640221.1614163601. (Lastet ned: 24.02.2021).
- [16] Arch Linux. Arch Linux. URL: <https://archlinux.org/>. (Lastet ned: 01.04.2021).
- [17] Arch Linux. Arch Linux - ArchWiki. URL: https://wiki.archlinux.org/title/Arch_Linux. (Lastet ned: 19.05.2021).
- [18] The Apache Software Foundation. NuttX Documentation. URL: <https://nuttx.apache.org/docs/latest/>. (Lastet ned: 18.05.2021).
- [19] The Apache Software Foundation. About Apache NuttX. URL: <https://nuttx.apache.org/docs/latest/introduction/about.html>. (Lastet ned: 18.05.2021).

- [20] Cameron Newham Bill Rosenblatt. Learning the Bash Shell: Unix Shell Programming. O'Reilly Media, Inc, 2005. ISBN: 9780596009656.
- [21] Ajmal Beg. Dynamic Graphical User Interface. Ajmal Beg, 2009. ISBN: 9780980561043.
- [22] Peter Seebach. Beginning Portable Shell Scripting: From Novice to Professional. Apress, 2008. ISBN: 9781430210436.
- [23] K. C Wang. Embedded and Real-Time Operating Systems. Cham: Springer International Publishing AG, 2017. ISBN: 9783319515168.
- [24] Gerardus Blokdyk. Data Distribution Service a Complete Guide - 2020 Edition. Emereo Pty Limited, 2020. ISBN: 9781867301219.
- [25] Open Management Group. About different ROS 2 DDS/RTPS vendors. URL: <https://www.omg.org/spec/DDS1-RTPS/2.3/PDF>. (Lastet ned: 23.02.2021).
- [26] Angelo Corsaro. DDS and OPC UA Explained. URL: <https://image.slidesharecdn.com/2016-160530080732/95/dds-and-opc-ua-explained-20-638.jpg?cb=1485337176>. (Lastet ned: 23.02.2021).
- [27] Paolo Bellavista et al. Middleware Solutions for Wireless Internet of Things. Multidisciplinary Digital Publishing Institute, 2019. ISBN: 9783039210374.
- [28] Open Robotics. About different ROS 2 DDS/RTPS vendors. URL: https://docs.ros.org/en/ros2_documentation/foxy/Concepts/About-Different-Middleware-Vendors.html. (Lastet ned: 23.02.2021).
- [29] Matt Hansen. ROS 2 Foxy Fitzroy: Setting a new standard for production robot development. URL: <https://d2908q01vomqb2.cloudfront.net/a9334987ece78b6fe8bf130ef00b74847c1d3da6/2020/06/10/image-2.png>. (Lastet ned: 23.02.2021).
- [30] ROS2 Community. ROS 2 Documentation. URL: <https://index.ros.org/doc/ros2/>. (Lastet ned: 17.02.2021).
- [31] ROS Community. rqt - ROS Wiki. URL: <http://wiki.ros.org/rqt>. (Lastet ned: 11.05.2021).
- [32] Dronecode. Software overview. URL: <https://px4.io/software/software-overview/>. (Lastet ned: 23.02.2021).
- [33] PX4. Software overview. URL: https://docs.px4.io/master/en/getting_started/px4_basic_concepts.html#qgc. (Lastet ned: 23.02.2021).
- [34] PX4. RTPS/DDS Interface: PX4-Fast RTPS(DDS) Bridge. URL: <https://docs.px4.io/master/en/middleware/micrortps.html>. (Lastet ned: 23.02.2021).
- [35] PX4. PX4-ROS 2 Bridge. URL: https://docs.px4.io/master/assets/img/architecture_ros2.f0ce052c.png. (Lastet ned: 23.02.2021).
- [36] eProsimma. Micro CDR. URL: <https://github.com/eProsima/Micro-CDR>. (Lastet ned: 12.05.2021).
- [37] eProsimma. Micro XRCE DDS. URL: <https://www.ros.org/index.php/products-all/eProsima-Micro-XRCE-DDS>. (Lastet ned: 12.05.2021).
- [38] PX4. microRTPS Bridge. URL: https://docs.px4.io/master/assets/img/architecture_94eff761.png. (Lastet ned: 23.02.2021).
- [39] Dronecode. PX4-Autopilot. URL: <https://github.com/PX4/PX4-Autopilot.git>. (Lastet ned: 14.05.2021).
- [40] Dronecode. Building PX4. URL: https://docs.px4.io/master/en/dev_setup/building_px4.html. (Lastet ned: 12.05.2021).
- [41] Dronecode. px4_msgs. URL: https://github.com/PX4/px4_msgs. (Lastet ned: 13.05.2021).
- [42] Jeremy Zheng Li. Magnetic materials and 3D finite element modeling. Springer International Publishing : Imprint: Springer, 2015. ISBN: 9783319059211.

- [43] Solidworks. SOLIDWORKS. URL: <https://www.solidworks.com/>. (Lastet ned: 18.05.2021).
- [44] Melba Kurman Hod Lipson. Fabricated: The New World of 3D Printing. John Wiley Sons, 2013. ISBN: 9781118416945.
- [45] William D Callister David G. Rethwisch. Materials Science and Engineering, 9th Edition, SI Version. John Wiley Sons Inc, 2014. ISBN: 9781118319222.
- [46] The Open University. Vacuum Forming (Thermoforming). URL: <https://www.open.edu/openlearn/science-maths-technology/engineering-technology/manupedia/vacuum-forming-thermoforming>. (Lastet ned: 22.05.2021).
- [47] Ariel Cornejo. Incetion molding diagram. URL: https://upload.wikimedia.org/wikipedia/commons/3/32/Injection_molding_diagram.svg. (Lastet ned: 27.04.2021).
- [48] Luftfartstilsynet. Regler for droner. URL: <https://luftfartstilsynet.no/droner/>. (Lastet ned: 25.01.2021).
- [49] Arbeidstilsynet. Tungt Arbeid. URL: <https://www.arbeidstilsynet.no/tema/ergonomi/manuelt-arbeid/tungt-arbeid/>. (Lastet ned: 27.01.2021).
- [50] Raul Sidnei Wazlawick. Object-Oriented Analysis and Design for Information Systems: Modeling with UML. Elsevier, 2014. ISBN: 9780124172937.
- [51] Arch Linux Forums. URL: <https://bbs.archlinux.org/>. (Lastet ned: 18.05.2021).
- [52] Ubuntu Community Support. URL: <https://discourse.ubuntu.com/t/community-support/709>. (Lastet ned: 18.05.2021).
- [53] Arch Linux Community. Installation guide. URL: https://wiki.archlinux.org/index.php/installation_guide. (Lastet ned: 10.02.2021).
- [54] Arch Linux Community. Building ROS 2 from source. URL: https://wiki.archlinux.org/index.php/ROS#ROS_2. (Lastet ned: 10.02.2021).
- [55] Canonical Ltd. Install Ubuntu Server. URL: <https://ubuntu.com/tutorials/install-ubuntu-server#1-overview>. (Lastet ned: 15.02.2021).
- [56] Canonical Ltd. Installing ROS 2 via Debian Packages. URL: <https://index.ros.org/doc/ros2/Installation/Foxy/Linux-Install-Debians/>. (Lastet ned: 15.02.2021).
- [57] Banana Pi. BPI-M2 Zero. URL: <http://www.banana-pi.org/m2z.html>. (Lastet ned: 19.02.2021).
- [58] Raspberry Pi Ltd. Raspberry Pi Compute Module 4. A Raspberry Pi for deeply embedded applications. URL: <https://datasheets.raspberrypi.org/cm4/cm4-datasheet.pdf>. (Lastet ned: 15.03.2021).
- [59] Texas Instruments. TPS6213x datasheet. URL: https://www.ti.com/lit/ds/symlink/tps62133.pdf?ts=1614957176833&ref_url=https%5C%253A%5C%252F%5C%252Fwww.ti.com%5C%252Fproduct%5C%252FTPS62133t. (Lastet ned: 27.04.2021).
- [60] Microship. MIC5019 Datasheet. URL: <https://ww1.microchip.com/downloads/en/DeviceDoc/MIC5019.pdf>. (Lastet ned: 27.04.2021).
- [61] John Gowar Duncan A. Grant. Power MOSFETs: Theory and Applications. Wiley, 1989. ISBN: 9780471828679.
- [62] JLCPCB. JLCPCB kapasitet. URL: <https://jlcpcb.com/capabilities/Capabilities>. (Lastet ned: 04.05.2021).
- [63] KiCAD. PCB Calculator. URL: https://docs.kicad.org/master/en/pcb_calculator/pcb_calculator.pdf. (Lastet ned: 18.05.2021).
- [64] Hirose. Hirosde DF40C-100DS0.4V Datasheet. URL: [https://www.hirose.com/en/product/document?clcode=CL0684-4033-4-51&productname=DF40C-100DS-0.4V\(51\)&series=DF40&documenttype=SpecSheet&lang=en&documentid=0000427780](https://www.hirose.com/en/product/document?clcode=CL0684-4033-4-51&productname=DF40C-100DS-0.4V(51)&series=DF40&documenttype=SpecSheet&lang=en&documentid=0000427780). (Lastet ned: 27.04.2021).

- [65] CHIPQUIK. CHIPQUIK TS391SN Datasheet. URL: <http://www.chipquik.com/datasheets/TS391SNL.pdf>. (Lastet ned: 27.04.2021).
- [66] Raspberry Pi Foundation. USB boot code. URL: <https://github.com/raspberrypi/usbboot>. (Lastet ned: 20.05.2021).
- [67] Raspberry Pi Foundation. Raspberry Pi 4 Product Brief. URL: <https://datasheets.raspberrypi.org/rpi4/raspberry-pi-4-product-brief.pdf>. (Lastet ned: 11.05.2021).
- [68] Raspberry Pi Foundation. firmware-nonfree repository. URL: <https://github.com/RPi-Distro/firmware-nonfree>. (Lastet ned: 24.05.2021).
- [69] Open Robotics. Distributions. URL: <https://docs.ros.org/en/foxy/Releases.html>. (Lastet ned: 12.05.2021).
- [70] Open Robotics. Installing ROS 2 via Debian Packages. URL: <https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html>. (Lastet ned: 12.05.2021).
- [71] getpv. Holybro Pixhawk 4 Mini Combo (w/ NEO-M8N GPS, PM06 V2). URL: <https://cdn.getfpv.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/f/c/fc64ffebaf.jpg>. (Lastet ned: 14.05.2021).
- [72] Get FPV. EMAX RS1106 4500KV. URL: <https://www.getfpv.com/emax-rs1106-4500kv-micro-brushless-motor-6563.html>. (Lastet ned: 12.05.2021).
- [73] PX4 User Guide. Holybro Micro Power Module (PM06). URL: https://docs.px4.io/master/en/power_module/holybro_pm06_pixhawk4mini_power_module.html. (Lastet ned: 12.05.2021).
- [74] Dronecode. microRTPS_agent.cpp. URL: https://github.com/PX4/micrortps_agent/blob/master/src/microRTPS_agent.cpp. (Lastet ned: 17.04.2021).
- [75] Dronecode. QGroundControl. URL: <http://qgroundcontrol.com/>. (Lastet ned: 18.05.2021).
- [76] Dronecode. Gazebo Simulation. URL: <https://docs.px4.io/master/en/simulation/gazebo.html>. (Lastet ned: 18.05.2021).
- [77] YR. Grimstad, Laurdag 8. Mai. URL: <https://www.yr.no/nn/historikk/tabell/1-7667/Noreg/Agder/Grimstad/Grimstad?q=2021-05-08>. (Lastet ned: 18.05.2021).

Tillegg A

Fremgangsmåte for bruk av systemet

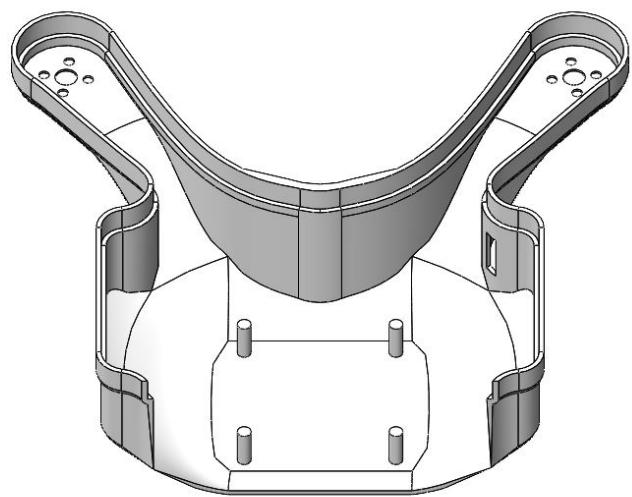
A.1 Montering

A.1.1 Drone

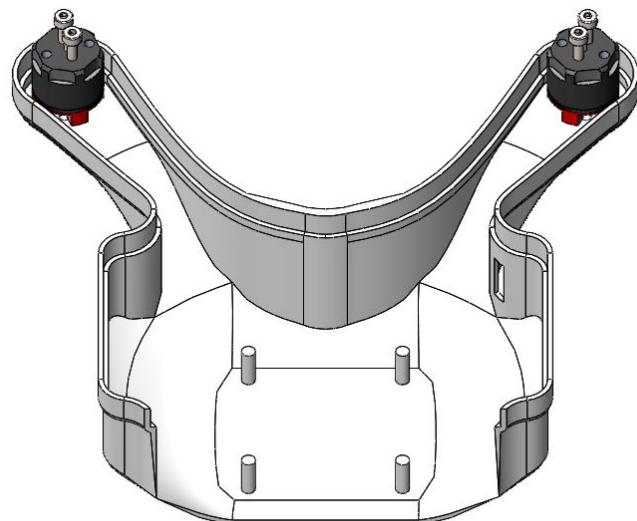
Figurene nedenfor er en grafisk visning gjort i Solidworks, som viser steg for steg hvordan en drone blir montert. Alle deler utenom ESCer, ledninger og konnektorer er tatt med.

Fremgangsmåte:

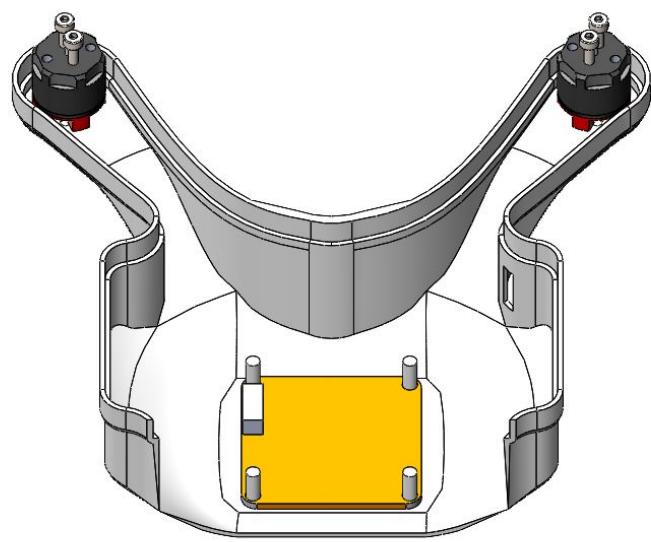
1. Begynner med tom skrogbunn.
2. Monterer motorer i hver arm.
3. Strømmodul plasseres nederst i dronen.
4. Innfestning til batteriet settes over strømmodul ved hjelp av stolper i skroget.
5. Batteriet settes inn.
6. Innfestning for RPi CM4 tres over batteri ved hjelp av stolpene.
7. RPi CM4 og bærekort blir skrudd på plass med fire skruer.
8. Innfestning for Pixhawk 4 mini settes oppå skruene til RPi CM4 og bærekort.
9. Pixhawk mini 4 plasseres mellom hjørnekanter.
10. Propeller blir skrudd på motorene.
11. Lokk settes på.
12. Ferdig montert drone.



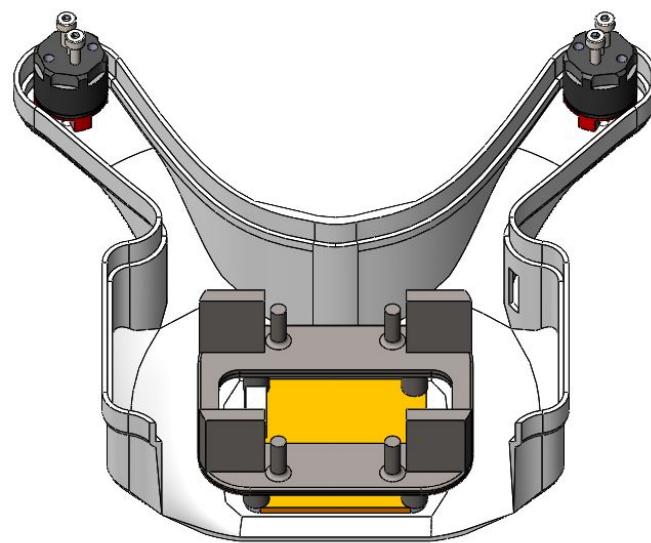
Figur A.1: Begynner med tom skrogbunn



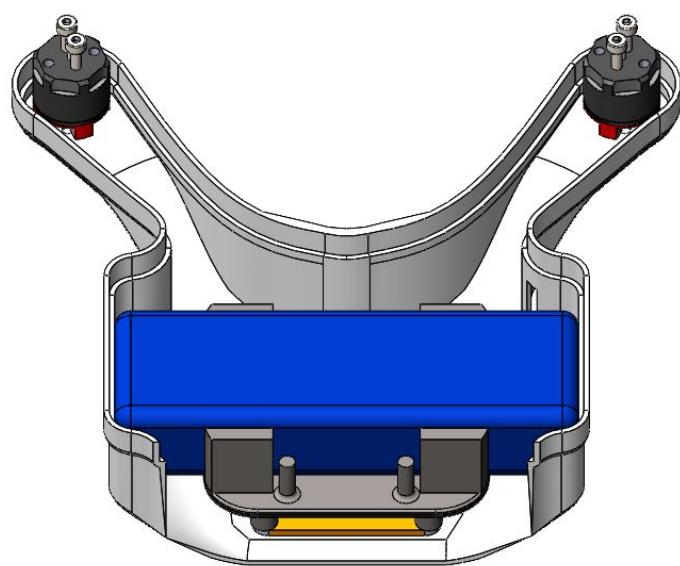
Figur A.2: Monterer motorer i hver arm.



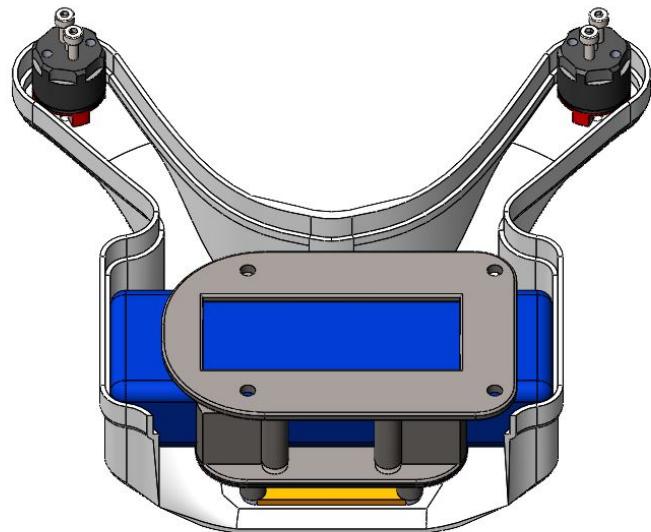
Figur A.3: Strømmodul plasseres nederst i dronen.



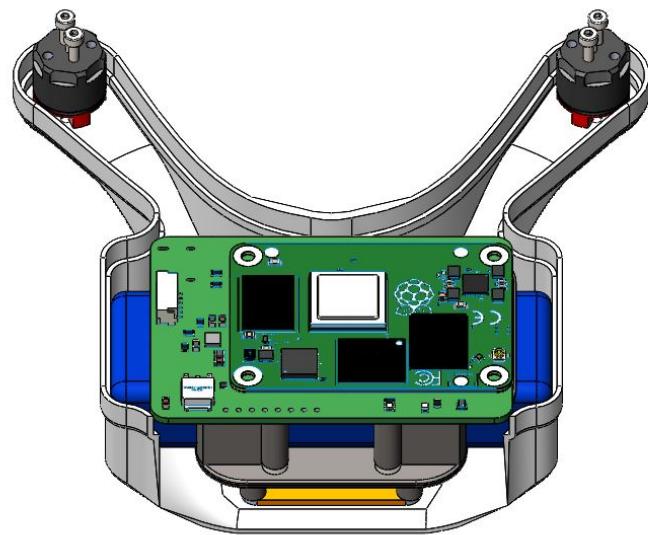
Figur A.4: Innfestning til batteriet settes over strømmodul ved hjelp av stolper i skroget.



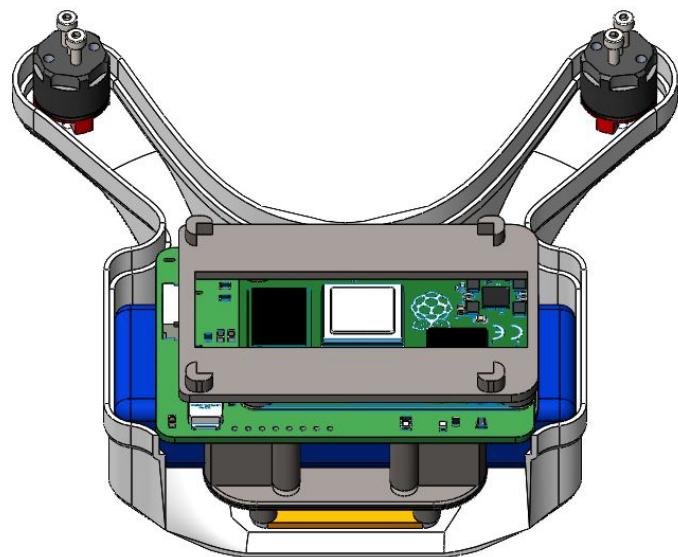
Figur A.5: Batteriet settes inn.



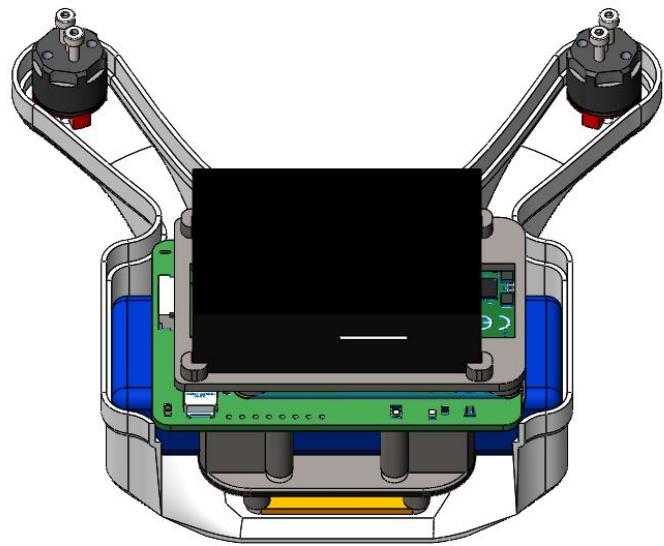
Figur A.6: Innfestning for RPi CM4 tres over batteriet ved hjelp av stolpene.



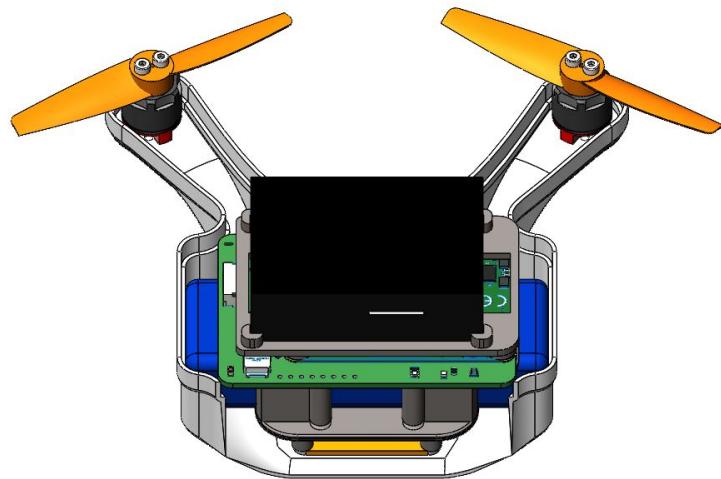
Figur A.7: RPi CM4 og bærekort blir skrudd på plass med fire skruer.



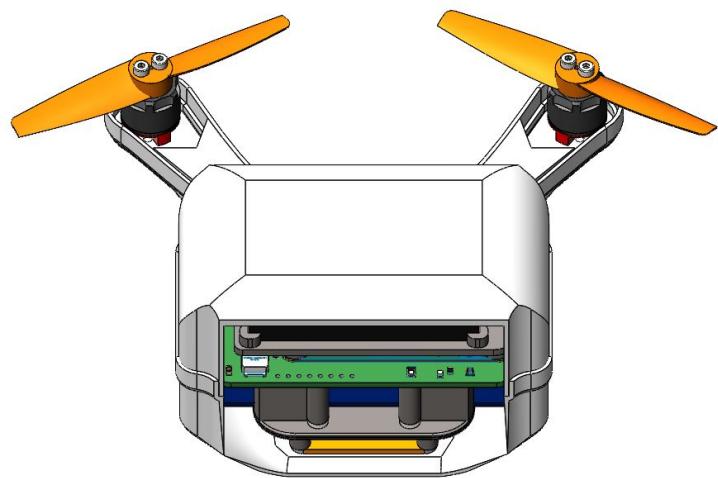
Figur A.8: Innfestning for Pixhawk 4 mini settes oppå skruene til RPi CM4 og bærekort.



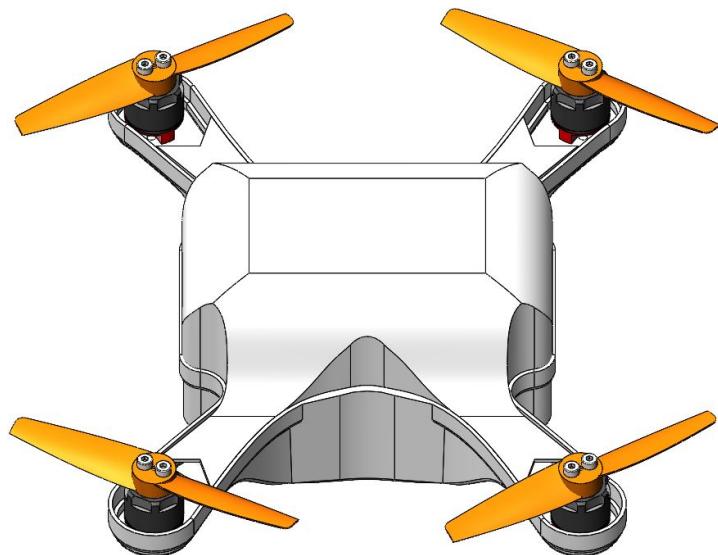
Figur A.9: Pixhawk 4 mini plasseres mellom hjørnekanter.



Figur A.10: Propeller blir skrudd på motorene.



Figur A.11: Lokk settes på.



Figur A.12: Ferdig montert drone.

A.1.2 Utskytningsenhet

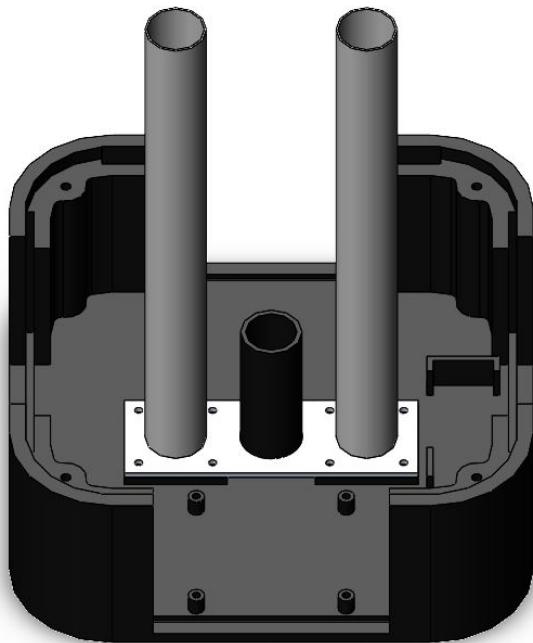
Figurene nedenfor er en grafisk visning gjort i Solidworks, som viser steg for steg hvordan en utskytningsenhet blir montert.

Fremgangsmåte:

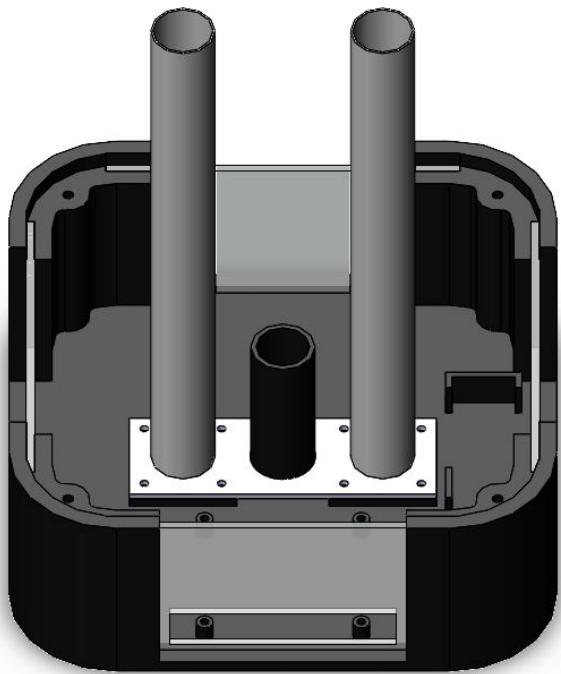
1. Begynner med elektronikkboksen.
2. Føringsstolpene til langsiden av dronene settes inn og skrus fast med 8 skruer under elektronikkboksen.
3. Pleksivegger skvyes på plass inn i ferdiglagde spor.
4. Bærekort med RPi CM4 skrus fast med fire skruer. Batteriet blir lagt på plass mellom egne sporvegger.
5. Tre pleksigulvplater blir lagt oppå og skrudd fast i hvert hjørne på elektronikkboksen.
6. Føringsstolper til kortsidene av dronene blir plassert i utkuttede spor og skrudd fast i nederste pleksigulv.
7. Utskytningsenhet ferdig montert med tre droner.



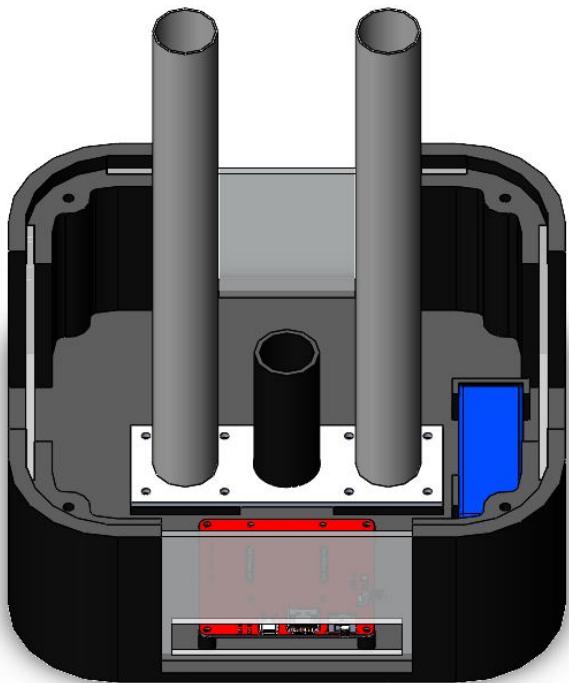
Figur A.13: Begynner med elektronikkboksen.



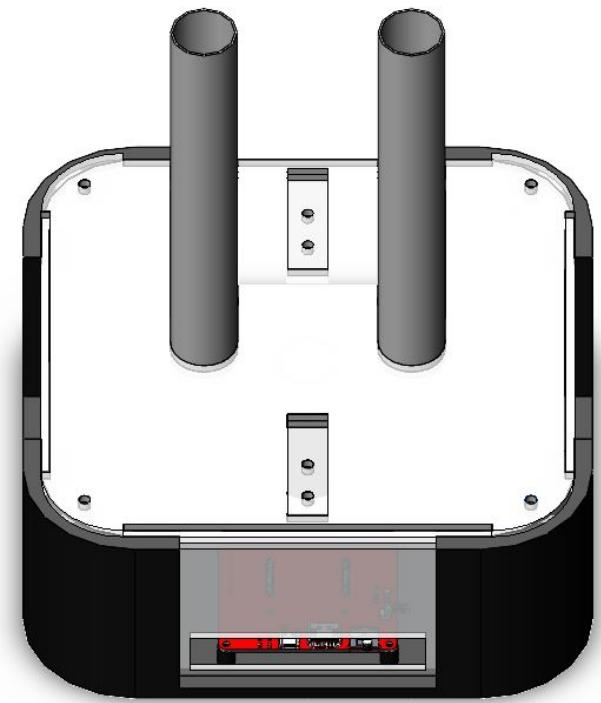
Figur A.14: Føringsstolpene til langsiden av dronene settes inn og skrus fast med åtte skruer under elektronikkboksen.



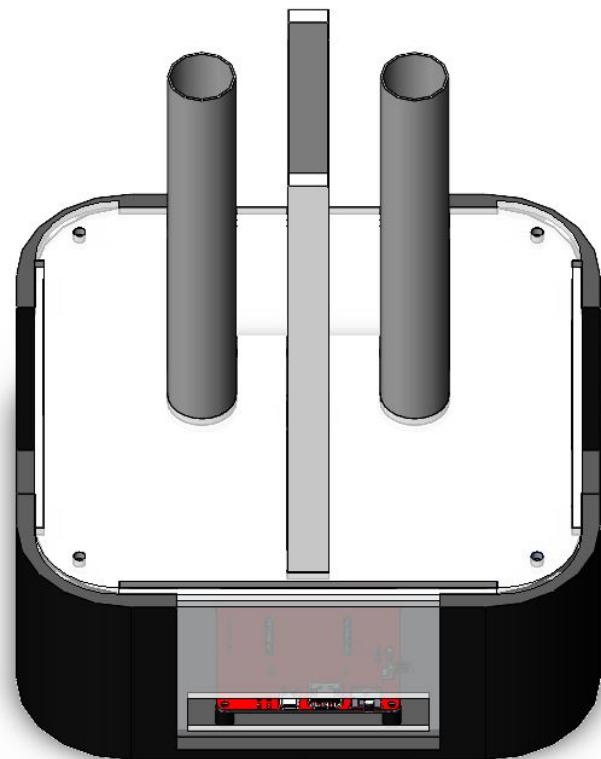
Figur A.15: Pleksivegger skyves på plass inn i ferdiglagde spor.



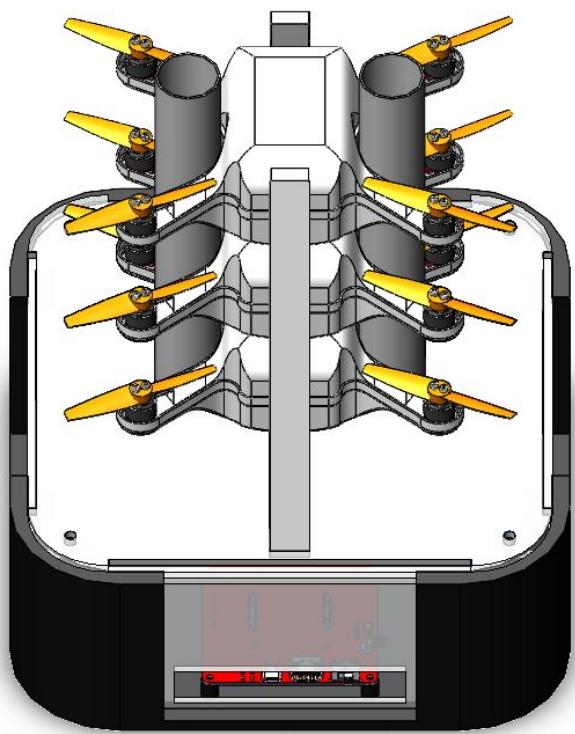
Figur A.16: Bærekort med RPi CM4 skrus fast med fire skruer. Batteriet blir lagt på plass mellom egne sporvegger.



Figur A.17: Tre pleksigulvplater blir lagt oppå og skrudd fast i hvert hjørne på elektronikkboksen.



Figur A.18: Føringsstolper til kortssiden av dronene blir plassert i utkuttede spor og skrudd fast i nederste pleksigulv.



Figur A.19: Utskytningsenhet ferdig montert med tre droner.

A.2 Hvordan bruke systemet

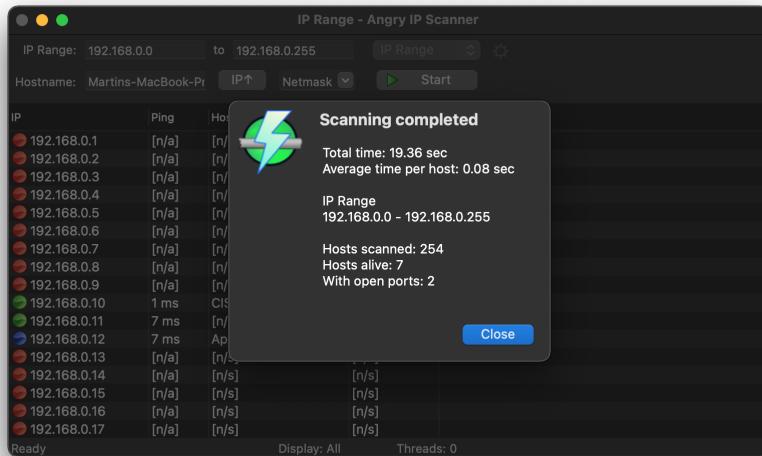
Videre følger instruksjoner for hvordan systemet brukes.

A.2.1 Forhåndskonfigurasjon

Før systemet kan brukes, må det konfigureres. Dette trengs kun å gjøres første gang systemet settes opp eller om nye enheter skal introduseres.

Enhettene settes opp med ulik identifikasjon. Med identifikasjon menes det tallet som representerer enhettene. For utskytningsenhettene starter denne identifikasjonen på 1 og øker til antall utskytningsenheter i systemet. For dronene starter identifikasjonen på 1 og øker opp til og med 10. Dronene har også en tall på hvilken utskytningsenhet de hører til.

Angry IP Scanner. Gjør det enkelt å finne ip-adressene til enheter på samme Wi-Fi. Figur A.20 viser brukergrensesnittet til Angry IP Scanner.



Figur A.20: Brukergrensesnitt Angry IP Scanner.

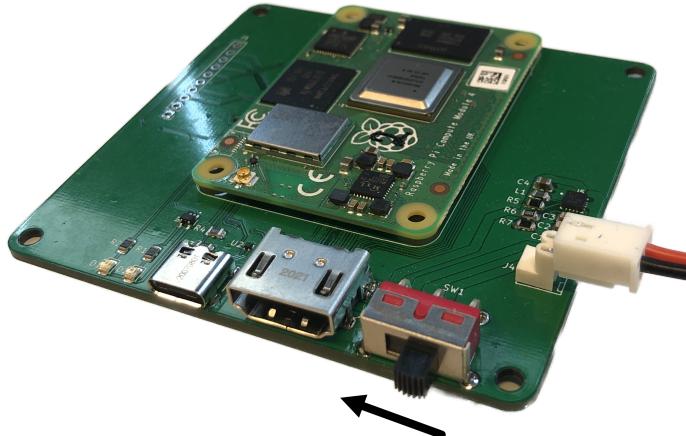
Wi-Fi

1. Sett opp et Wi-Fi med følgende SSID og passord.

- SSID: Dronesverm
- Passord: Dronesverm2020

Utskytningsenhet

1. Utskytningsenheten skrus på ved å koble til et fulladet LiPo-batteri og skyve bryteren til venstre, som vist i Figur A.21.

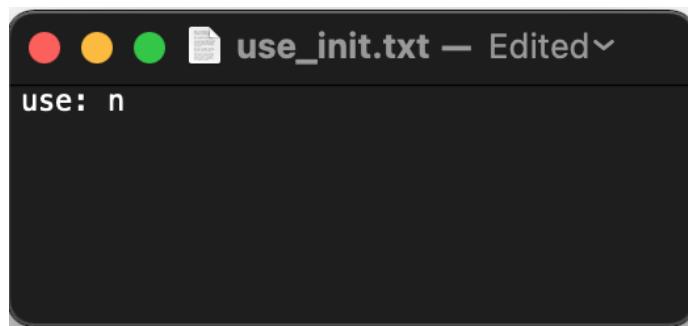


Figur A.21: Skru på utskytningsenheten.

2. Bruk SSH til å fjernstyre utskytningsenheten. For en Ubuntu-maskin gjøres dette ved å skrive følgende i terminalen:

```
$ ssh ubuntu@ip-adresse
```

3. Naviger så til mappen ”/home/ubuntu/ds_scripts/use/”, her ligger filen ”use_init.txt”. I denne filen må n i ”use: n” erstattes med identifikasjonen til utskytningsenheten. Innholdet i filen er vist i Figur A.22.



Figur A.22: Tekstfil for oppsett av identifikasjon for utskytningsenhet.

NB: Det kan bli problemer dersom flere utskytningsenheter har samme identifikasjon, så unngå dette.

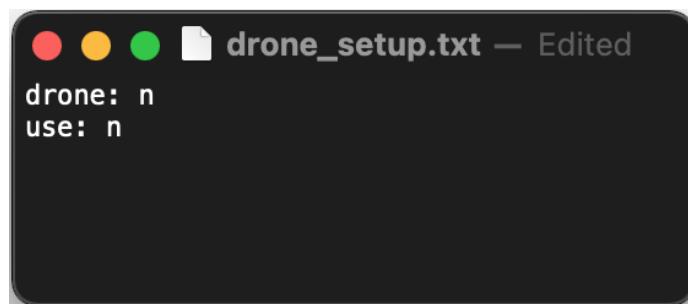
4. Skru av utskytningsenheten ved å skyve skyverbryteren til høyre, motsatt av hva som ble gjort i steg 1.
5. Gjenta prosessen for hver utskytningsenhet.

Drone

1. Dronen skrus på ved å koble til et fulladet LiPo-batteri og skyve bryteren opp.
2. Bruk SSH til å fjernstyre dronen. For en Ubuntu-maskin gjøres dette ved å skrive følgende i terminalen:

```
$ ssh ubuntu@ip-adresse
```

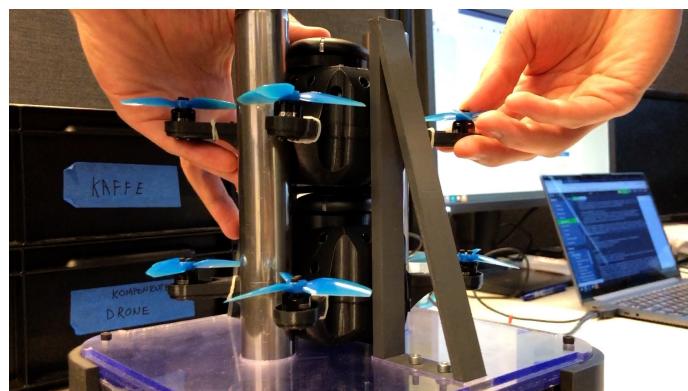
3. Naviger så til mappen ”/home/ubuntu/ds_scripts/drone/”, her ligger filen ”drone_setup.txt”. I denne filen må n i ”drone: n” erstattes med identifikasjonen til utskytningsenheten. Innholdet i filen er vist i Figur A.23.



Figur A.23: Tekstfil for oppsett av identifikasjon for drone.

NB: Det kan ikke være flere droner i én utskytningsenhet med samme identifikasjon.

4. Skru av dronen ved å skyve skyverbryteren ned.
5. Plasser dronen i den bestemte utskytningsenheten fra steg 3, som vist i Figur A.24.



Figur A.24: Plasser dronen i den bestemte utskytningsenheten.

6. Gjenta prosessen for hver drone.

Bakkestasjon

- På pc-en som skal brukes som bakkestasjon må det lages en fil i hjem-mappen. Denne må kalles ”system_setup.txt”, og inneholde ”use: n”, der n er antallet utskytningsenheter i systemet.

Systemet er nå konfigurert og klart til flyging.

A.2.2 Flyging

1. Plasser utskytningsenheten på et åpent område, som vist i Figur A.25.



Figur A.25: Plassering av utskytningsenhet på åpent område.

2. Skru på utskytningsenheten(e) og dronen(e).
3. Start ROS 2 noden ”bs_droneControl” på bakkestasjonen. Denne kan startes ved å åpne et nytt terminalvindu og skrive kommandoene vist under.

```
$ ros2 run ds_ros2_bs bs_droneControl
```

NB: ROS 2 dronesverm arbeidsområdet må sources før bruk.

Etter å ha startet ROS 2 noden vil brukeren få opp en ny kommandolinje i terminalen. Fra denne kommandolinjen kan samtlige droner kontrolleres. Figur A.26 viser kommandolinjen.

A screenshot of a terminal window titled "martin@martin-ubuntu:~". The window shows the command "\$ ros2 run ds_ros2_bs bs_droneControl" being typed. After the command, there is a prompt "#-----# DRONESWARM CONTROL CENTER #-----#" and then a cursor at ">>".

Figur A.26: ROS 2 noden på bakkestasjonen.

Kommandoer for flyging

I kommandovinduet kan dronene kontrolleres. Dette gjøres ved å skrive ulike kommandoer. Hver kommando må ha to tall etterfulgt av en tekst, som vist i Figur A.27. Det første tallet forteller hvilken utskytningsenhet meldingen skal til (1). Det andre tallet forteller hvilken drone i den utskytningsenheten meldingen skal til (2). Teksten som står etter forteller hva du ønsker at den dronen skal gjøre (3). Figur A.28 viser en komplett oversikt over tilgjengelige kommandoer og noen eksempler.

```
#-----#
DRONESWARM CONTROL CENTER
#-----#
>>01 01 arm
1 2 3
```

Figur A.27: Eksempel på kommando.

Utskytningsenhet	Handling
n	Velge utskytningsenhet n. Støttet n opptil 10.
999	Velge alle utskytningsenheter.

Drone	Handling
n	Velge drone n. Støttet n opptil 10.
999	Velge alle droner.

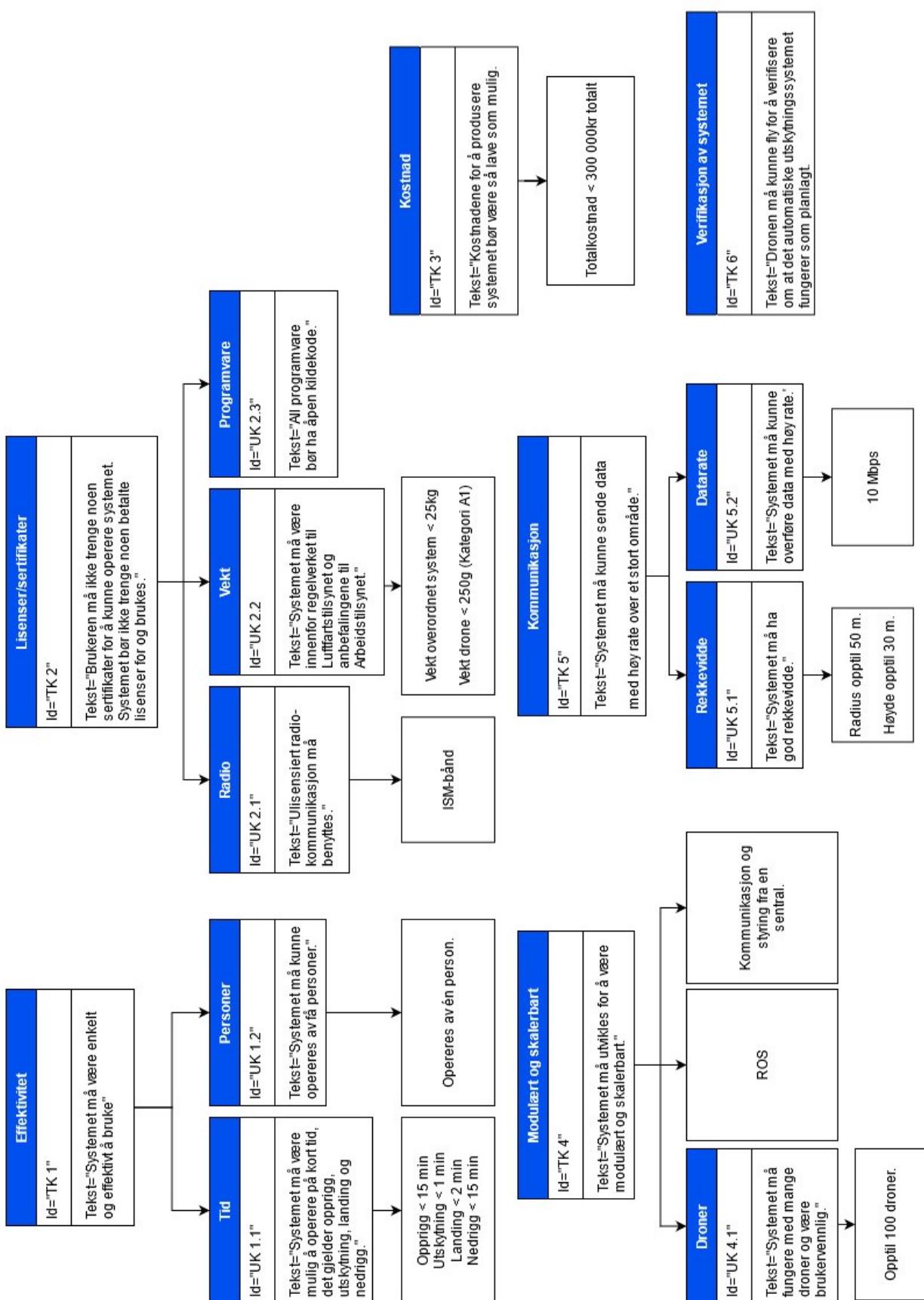
Kommando	Handling
arm	Armere dronen.
disarm	Disarmere dronen.
launch	Lette dronen.
land	Lande dronen.
velocity	Sette dronen til å motta ønsket hastighet.
x<n>, y<n>, z<n>	Sette ønsket posisjon. Bytt ut <n> med en verdi.
vx<n>, vy<n>, vz<n>	Sette ønsket hastighet. Er avhengig at velocity er satt først. Bytt ut <n> med en verdi

Eksempler	Handling
01 01 arm	Armere drone 01 i utskytningsenhet 01.
02 04 disarm	Disarmere drone 04 i utskytningsenhet 02.
01 999 velocity	Sette alle dronene i utskytningsenhet 01 til å motta ønsket hastighet.
999 01 z-1	Sette ønsket posisjon til én meter opp for drone 01 i alle utskytningsenhetene.

Figur A.28: Liste over kommandoer.

Tillegg B

UML-skjema



Figur B.1: Fullt UML-skjema av krav og kriterier for drone og utskytningsenhet konsepter.

Tillegg C

Programvare

I dette tillegget er all programvare brukt vedlagt.

Programvaren kan også finnes på GitHub:

- https://github.com/mdherman/automatic_takeoff

C.1 Bakkestasjon

C.1.1 ROS 2 node i bakkestasjonen

```
51 import rclpy
52 from rclpy.node import Node
53 import getpass
54
55
56 # Import msg
57 from std_msgs.msg import String
58 from ds_ros2_msgs.msg import DroneControl
59 from ds_ros2_msgs.msg import TrajectorySetpoint
60
61
62 # Create setpoint publisher class
63 class DroneControlNode(Node):
64
65     def __init__(self):
66         # Init publisher
67         super().__init__('bs_droneControl')
68
69         #Variables
70         self.uses = 0
71         self.land = False
72         self.use = 0
73         self.drone_nr = 0
74
75         self.setup()
76
77         # Create publishers
78         self.control_publisher = [0]
79         self.setpoint_publisher = [0]
80         for i in range(1, self.uses+1):
81             self.control_publisher.append(self.create_publisher(DroneControl, '/use_0' + str(i) + '/bs_use_control', 10))
82             self.setpoint_publisher.append(self.create_publisher(TrajectorySetpoint, '/use_0' + str(i) + '/bs_use_setpoint', 10))
83
84         # Run prompt function
85         self.prompt_user()
86
```

```

87
88     def setup(self):
89         username = getpass.getuser()
90         system_setup = open("/home/" + username + "/system_setup.txt")
91         for line in system_setup:
92             if "use: " in line:
93                 space = line.find(" ", 0, len(line))
94                 self.uses = int(line[space:])
95
96     # Prompt user for commands
97     def prompt_user(self):
98
99         # Welcome message
100        print("#-----#")
101        print("DRONESWARM CONTROL CENTER")
102        print("#-----#")
103
104    # Create DroneControl message instance
105    control_msg = DroneControl()
106    setpoint_msg = TrajectorySetpoint()
107
108    # Keep prompt going
109    while True:
110        send_control_input = "not enter"
111        send_control_input = input(">>")
112
113        # If input is blank, catch error
114        if (send_control_input == ""):
115            send_control_input = "blank"
116
117        # Check which use and drone to control
118        first_space = send_control_input.find(" ", 0, len(send_control_input))
119        if ( (send_control_input[:first_space].isdigit()) and (first_space != -1) ):
120            self.use = int(send_control_input[:first_space])
121            send_control_input = send_control_input[first_space+1:]
122
123            second_space = send_control_input.find(" ", 0, len(send_control_input))
124        if ( (send_control_input[:second_space].isdigit()) and (second_space != -1) ):
125            self.drone_nr = int(send_control_input[:second_space])
126            control_msg.drone = int(send_control_input[:second_space])
127            setpoint_msg.drone = int(send_control_input[:second_space])
128            send_control_input = send_control_input[second_space+1:]
129
130        # Invalid input if drone nr or use nr is out of range
131        if ((self.use == 999 or self.use in range(1, self.uses+1)) and (self.drone_nr == 999 or self.drone_nr in range(1, 11))):
132            None
133        else:
134            send_control_input = "invalid"
135
136        # Check command input
137        if (send_control_input.upper() == "DISARM"):
138
139            # Disarm
140            print("Disarm command sent..")
141            control_msg.arm = False
142            self.publish_(0, control_msg, setpoint_msg)
143
144        elif (send_control_input.upper() == "ARM"):
145
146            # Disarm before arm
147            control_msg.arm = False
148            control_msg.land = False
149            self.publish_(0, control_msg, setpoint_msg)

```

```

150
151     # Arm and offboard control true
152     print("Arm command sent...")
153     control_msg.arm = True
154     self.publish_(0, control_msg, setpoint_msg)
155
156 elif (send_control_input.upper() == "LAUNCH"):
157     control_msg.launch = True
158     self.publish_(0, control_msg, setpoint_msg)
159     print("Launch command sent...")
160
161 elif (send_control_input.upper() == "LAND"):
162     control_msg.land = True
163     self.publish_(0, control_msg, setpoint_msg)
164     print("Land command sent...")
165
166 elif (send_control_input.upper() == "TURN PX ON"):
167     control_msg.switch_px = True
168     self.publish_(0, control_msg, setpoint_msg)
169     print("Switched on pixhawk...")
170
171 elif (send_control_input.upper() == "TURN PX OFF"):
172     control_msg.switch_px = False
173     self.publish_(0, control_msg, setpoint_msg)
174     print("Switched off pixhawk...")
175
176 elif (send_control_input.upper()[0] == "X"):
177     print("Setpoint x sent...")
178     setpoint_msg.x = float(send_control_input[1:])
179     self.publish_(0, control_msg, setpoint_msg)
180
181 elif (send_control_input.upper()[0] == "Y"):
182     print("Setpoint y sent...")
183     setpoint_msg.y = float(send_control_input[1:])
184     self.publish_(0, control_msg, setpoint_msg)
185
186 elif (send_control_input.upper()[0] == "Z"):
187     print("Setpoint z sent...")
188     setpoint_msg.z = float(send_control_input[1:])
189     self.publish_(0, control_msg, setpoint_msg)
190
191 elif (send_control_input.upper()[0:2] == "VZ"):
192     print("Setpoint vz sent...")
193     setpoint_msg.vz = float(send_control_input[2:])
194     self.publish_(0, control_msg, setpoint_msg)
195
196 elif (send_control_input.upper() == "VELOCITY"):
197     setpoint_msg.x = float("NaN")
198     setpoint_msg.y = float("NaN")
199     setpoint_msg.z = float("NaN")
200     setpoint_msg.yaw = float("NaN")
201     setpoint_msg.yawspeed = float("NaN")
202     setpoint_msg.vx = 0.0
203     setpoint_msg.vy = 0.0
204     setpoint_msg.vz = 0.0
205     setpoint_msg.acceleration = [float("NaN"), float("NaN"), float("NaN")]
206     setpoint_msg.jerk = [float("NaN"), float("NaN"), float("NaN")]
207     setpoint_msg.thrust = [float("NaN"), float("NaN"), float("NaN")]
208
209     control_msg.arm = False
210     control_msg.launch = False
211
212     print("Ready for velocity setpoints..")
213
214     self.publish_(0, control_msg, setpoint_msg)
215

```

```

216
217     elif (send_control_input.upper() == "RESET"):
218         setpoint_msg.x = 0.0
219         setpoint_msg.y = 0.0
220         setpoint_msg.z = 0.0
221         setpoint_msg.yaw = 0.0
222         setpoint_msg.yawspeed = 0.0
223         setpoint_msg.vx = 0.0
224         setpoint_msg.vy = 0.0
225         setpoint_msg.vz = 0.0
226         setpoint_msg.acceleration = [0.0, 0.0, 0.0]
227         setpoint_msg.jerk = [0.0, 0.0, 0.0]
228         setpoint_msg.thrust = [0.0, 0.0, 0.0]
229
230         control_msg.arm = False
231         control_msg.launch = False
232
233         print("All reset..")
234
235         self.publish_(0, control_msg, setpoint_msg)
236
237     else:
238         print("invalid command..")
239
240 def publish_(self, msg, control_msg, setpoint_msg):
241     if (self.use == 999):
242         for publisher in self.setpoint_publisher:
243             if (publisher != 0):
244                 publisher.publish(setpoint_msg)
245         for publisher in self.control_publisher:
246             if (publisher != 0):
247                 publisher.publish(control_msg)
248     elif (msg == 0):
249         self.setpoint_publisher[self.use].publish(setpoint_msg)
250         self.control_publisher[self.use].publish(control_msg)
251     elif (msg == 1):
252         self.setpoint_publisher[self.use].publish(setpoint_msg)
253     elif (msg == 2):
254         self.control_publisher[self.use].publish(control_msg)
255
256 def main(args=None):
257     rclpy.init(args=args)
258
259     bs_droneControl = DroneControlNode()
260
261     rclpy.spin(bs_droneControl)
262
263     bs_droneControl.destroy_node()
264
265     rclpy.shutdown()
266
267
268 if __name__ == '__main__':
269     main()

```

C.2 Drone

C.2.1 ROS 2 node i drone

```

51 from px4_msgs.msg import OffboardControlMode
52 from px4_msgs.msg import TrajectorySetpoint
53 from px4_msgs.msg import Timesync
54 from px4_msgs.msg import VehicleCommand

```

```

55 from px4_msgs.msg import VehicleControlMode
56
57 from ds_ros2_msgs.msg import DroneControl
58 from ds_ros2_msgs.msg import TrajectorySetpoint as TrajectorySetpointDS
59
60 import rclpy
61 from rclpy.node import Node
62 import os
63
64
65 class PX4OffboardControl(Node):
66     def __init__(self):
67         # Init node with name
68         super().__init__("drone_offboard_control")
69
70     # Creating publishers
71     self.offboard_control_mode_publisher_ = self.create_publisher(
72         OffboardControlMode, "OffboardControlMode_PubSubTopic", 10)
73     self.trajectory_setpoint_publisher_ = self.create_publisher(TrajectorySetpoint,
74         "TrajectorySetpoint_PubSubTopic", 10)
75     self.vehicle_command_publisher_ = self.create_publisher(VehicleCommand,
76         "VehicleCommand_PubSubTopic", 10)
77
78     # Creating subscribers
79     self.timesync_subscriber_ = self.create_subscription(Timesync,
80         "Timesync_PubSubTopic", self.timesync, 10)
81     self.use_drone_setpoint_subscriber_ = self.create_subscription(
82         TrajectorySetpointDS, "use_drone_setpoint", self.fetch_trajectory_setpoint, 10)
83     self.control_subscriber_ = self.create_subscription(DroneControl,
84         "use_drone_control", self.drone_control, 10)
85
86     # Member trajectory setpoint message
87     self.trajectory_msg_ = TrajectorySetpoint()
88
89     # Member variables
90     self.timestamp_ = 0
91
92     # Flags
93     self.arm_ = False
94     self.land_ = True
95     self.launch_ = False
96     self.armed_ = False
97     self.switch_px_ = False
98     self.px_status_ = False
99
100    # Timer running at 20 Hz
101    timer_period = 0.05
102    self.timer = self.create_timer(timer_period, self.timer_callback)
103
104    # Spinning function
105    def timer_callback(self):
106        # This arms the drone
107        if self.arm_ == True and self.armed_ == False:
108            self.arm_vehicle()
109            self.armed_ = True
110
111        # This disarms the drone
112        if self.arm_ == False and self.armed_ == True:
113            self.disarm_vehicle()
114            self.set_stabilized_mode()
115            self.armed_ = False
116            self.launch_ = False
117
118        # This launches the drone
119        if self.launch_ == True and self.armed_ == True:
120            self.set_offboard_mode()

```

```

116     self.publish_offboard_control_mode()
117     self.trajectory_setpoint_publisher_.publish(self.trajectory_msg_)
118
119     # This lands the drone
120     if self.land_ == True:
121         self.trajectory_msg_.z = float("NaN")
122         self.trajectory_msg_.vz = 0.2
123
124     # This switches the pixhawk
125     if self.switch_px_ == True and self.px_status_ == False:
126         os.system("sudo sh -c 'echo '1' > /sys/class/gpio/gpio27/value'")
127         self.px_status_ = True
128     elif self.switch_px_ == False and self.px_status_ == True:
129         os.system("sudo sh -c 'echo '0' > /sys/class/gpio/gpio27/value'")
130         self.px_status_ = False
131
132     # System control
133     def drone_control(self, control_msg):
134         self.arm_ = control_msg.arm
135         self.land_ = control_msg.land
136         self.switch_px_ = control_msg.switch_px
137         self.launch_ = control_msg.launch
138
139     # Fetch setpoints
140     def fetch_trajectory_setpoint(self, use_msg):
141         self.trajectory_msg_.x = use_msg.x
142         self.trajectory_msg_.y = use_msg.y
143         self.trajectory_msg_.z = use_msg.z
144         self.trajectory_msg_.yaw = use_msg.yaw
145         self.trajectory_msg_.vx = use_msg.vx
146         self.trajectory_msg_.vy = use_msg.vy
147         self.trajectory_msg_.vz = use_msg.vz
148         self.trajectory_msg_.acceleration = use_msg.acceleration
149         self.trajectory_msg_.jerk = use_msg.jerk
150         self.trajectory_msg_.thrust = use_msg.thrust
151
152     # Fetch timestamp
153     def timesync(self, px4_time):
154         self.timestamp_ = px4_time.timestamp
155
156     # Sets mode to offboard control.
157     def set_offboard_mode(self):
158         self.publish_vehicle_command(VehicleCommand.VEHICLE_CMD_DO_SET_MODE, 1, 6) # Control modes..
159         self.get_logger().info("Drone set to offboard mode")
160
161     # Set mode to stabalized control.
162     def set_stabilized_mode(self):
163         self.publish_vehicle_command(VehicleCommand.VEHICLE_CMD_DO_SET_MODE, 1, 2) # Control modes..
164         self.get_logger().info("Drone set to stabalized mode")
165
166     # Send a command to Arm the vehicle
167     def arm_vehicle(self):
168         self.publish_vehicle_command(VehicleCommand.VEHICLE_CMD_COMPONENT_ARM_DISARM, 1.0, 0.0)
169         self.get_logger().info("Arm command send")
170
171     # Send a command to Disarm the vehicle
172     def disarm_vehicle(self):
173         self.publish_vehicle_command(VehicleCommand.VEHICLE_CMD_COMPONENT_ARM_DISARM, 0.0, 0.0)
174         self.get_logger().info("Disarm command send")
175
176     # Publish the offboard control mode.
177     def publish_offboard_control_mode(self):
178         msg = OffboardControlMode()

```

```

179     msg.timestamp = self.timestamp_
180     msg.position = True
181     msg.velocity = True
182     msg.acceleration = False
183     msg.attitude = False
184     msg.body_rate = False
185
186     self.offboard_control_mode_publisher_.publish(msg)
187
188 # Publish vehicle command
189 def publish_vehicle_command(self, command, param1, param2):
190     msg = VehicleCommand()
191
192     msg.timestamp = self.timestamp_
193     msg.param1 = float(param1)
194     msg.param2 = float(param2)
195     msg.command = command
196     msg.target_system = 1
197     msg.target_component = 1
198     msg.source_system = 1
199     msg.source_component = 1
200     msg.from_external = True
201
202     self.vehicle_command_publisher_.publish(msg)
203
204
205
206
207 def main(args=None):
208     rclpy.init(args=args)
209
210     px4_offboard_control = PX4OffboardControl()
211
212     rclpy.spin(px4_offboard_control)
213
214     px4_offboard_control.destroy_node()
215
216     rclpy.shutdown()
217
218
219 if __name__ == "__main__":
220     main()

```

C.2.2 Oppstarts-skript for drone

```

51 import getpass
52 import os
53 import time
54 import sys
55 username = getpass.getuser()
56
57 try:
58     # Get current file path
59     file_path = sys.path[0]
60
61     # Reading system_setup file
62     system_setup = open(file_path + "/drone_setup.txt")
63     for line in system_setup:
64         if "use: " in line:
65             space = line.find(" ", 0, len(line))
66             use = int(line[space:])
67         elif "drone: " in line:
68             space = line.find(" ", 0, len(line))

```

```

69         drone = int(line[space:])
70
71     # Checking if drone and use is within range
72     if (drone in range(1,10) and use in range(1,10)):
73         string = "use_0" + str(use) + "/drone_0" + str(drone)
74     elif (use >= 10 and drone == 10):
75         string = "use_" + str(use) + "/drone_" + str(drone)
76     elif (use >= 10 and drone in range(1,10)):
77         string = "use_" + str(use) + "/drone_0" + str(drone)
78     elif (drone == 10 and use in range(1,10)):
79         string = "use_0" + str(use) + "/drone_" + str(drone)
80     else:
81         string = "none"
82
83     # Starting ROS2 node and micrortps_agent if drone and use is in range
84     if (string != "none"):
85         os.system("source /opt/ros/foxy/setup.bash")
86         os.system("source /home/ubuntu/dronesverm_ws/install/local_setup.bash")
87         os.system("micrortps_agent start -t UART -b 921600 -d /dev/ttyS0 -n '" +
88         string + "' &")
89         time.sleep(10)
90         os.system("ros2 run ds_ros2_drone_pkg drone_offboard_control --ros-args -r
91         _ns:=/" + string)
92     else:
93         print("Something went wrong...")
94         print("Have you remembered to select drone- and use nr in ~/drone_setup.txt
95         ?")
96     except:
97         print("Something went wrong...")
98         print("Have you remembered to select drone- and use nr in ~/drone_setup.txt?")

```

C.3 Utskytningsenhet

C.3.1 ROS 2 node i utskytningsenhet

```

51 import rclpy
52 from rclpy.node import Node
53
54 # Import correct msg
55 from ds_ros2_msgs.msg import TrajectorySetpoint
56 from ds_ros2_msgs.msg import DroneControl
57
58 class Transfer(Node):
59
60     def __init__(self):
61         # Init parent-class
62         super().__init__('use_transfer')
63
64         # Get namespace this node is started in
65         self.my_namespace_ = super().get_namespace()
66
67         # Trajectory setpoint transfer subscriber and publishers
68         self.trajectory_subscriber_ = self.create_subscription(TrajectorySetpoint,
69         'bs_use_setpoint', self.recv_setpoints, 10)
70         self.trajectory_publishers = [0]
71         for drone in range(1,10):
72             self.trajectory_publishers.append(self.create_publisher(
73             TrajectorySetpoint, self.my_namespace_ + '/drone_0' + str(drone) + '/'
74             'use_drone_setpoint', 10))
75             self.trajectory_publishers.append(self.create_publisher(TrajectorySetpoint
76             , self.my_namespace_ + '/drone_10' + '/use_drone_setpoint', 10))
77
78         # Drone control transfer subscriber and publishers

```

```

75         self.droneControl_subscriber = self.create_subscription(DroneControl, 'bs_use_control', self.recv_control, 10)
76         self.droneControl_publishers = [0]
77         for drone in range(1,10):
78             self.droneControl_publishers.append(self.create_publisher(DroneControl, self.my_namespace_ + '/drone_0' + str(drone) + '/use_drone_control', 10))
79             self.droneControl_publishers.append(self.create_publisher(DroneControl, self.my_namespace_ + '/drone_10' + '/use_drone_control', 10))
80
81     # Number of messages sent variables
82     self.i_trajectory = 0
83     self.i_droneControl = 0
84
85
86     def recv_setpoints(self, msg):
87         # Transfer setpoint to correct drone
88         if (msg.drone == 999):
89             for publisher in self.trajectory_publishers:
90                 if (publisher != 0):
91                     publisher.publish(msg)
92
93         elif (msg.drone > 0 and msg.drone < 11):
94             self.trajectory_publishers[msg.drone].publish(msg)
95
96         # Send log
97         self.i_trajectory += 1
98         self.get_logger().info('Transferred TrajectorySetpoint: %d' % self.i_trajectory)
99
100
101     def recv_control(self, msg):
102         # Transfer control to correct drone
103         if (msg.drone == 999):
104             for publisher in self.droneControl_publishers:
105                 if (publisher != 0):
106                     publisher.publish(msg)
107
108         elif (msg.drone > 0 and msg.drone < 11):
109             self.droneControl_publishers[msg.drone].publish(msg)
110
111         # Send log
112         self.i_droneControl += 1
113         self.get_logger().info('Transferred DroneControl: %d' % self.i_droneControl)
114
115
116     def main(args=None):
117         rclpy.init(args=args)
118
119         use_transfer = Transfer()
120
121         rclpy.spin(use_transfer)
122
123         use_transfer.destroy_node()
124         rclpy.shutdown()
125
126
127     if __name__ == '__main__':
128         main()

```

C.3.2 Oppstarts-skript for utskytningsenhet

```
51 import os
```

```

52 import sys
53
54 try:
55     # Get current file path
56     file_path = sys.path[0]
57
58     # Reading use_init file
59     system_setup = open(file_path + "/use_init.txt")
60     for line in system_setup:
61         if "use: " in line:
62             space = line.find(" ", 0, len(line))
63             use_number = int(line[space:])
64             use_namespace = str(use_number)
65
66     # Source ros 2
67     os.system("source /opt/ros/foxy/setup.bash")
68     os.system("source /home/ubuntu/dronesverm_ws/install/local_setup.bash")
69
70     if (use_number > 10):
71         os.system("ros2 run ds_ros2_use use_transfer --ros-args -r __ns:=/use_" +
72               use_namespace)
73     elif (use_number in range(1,10)):
74         os.system("ros2 run ds_ros2_use use_transfer --ros-args -r __ns:=/use_0" +
75               use_namespace)
76     else:
77         print("Invalid number of use..")
78 except:
79     print("Could not read 'use_init.txt'..")

```

C.4 Annet

C.4.1 Automatisk innlogging

```

# Endre #NAutoVTs som ligger i /etc/systemd/logind.conf fra '6' til '1'

# Lag mappen og filen /etc/systemd/system/getty@tty1.service.d/override.conf ved
# kjøre kommandoen;

systemctl edit getty@tty0

# Legg inn følgende i override.conf filen;

[Service]
ExecStart=
ExecStart=/sbin/agetty --autologin root --noclear %I 38400 linux

# Innvilg getty@tty0.service, for så reboote;

systemctl enable getty@tty1.service
reboot

```

C.4.2 Shell-skript for ROS 2 installasjon

```

# Shell script for installing ROS 2
# Works with: foxy, eloquent
# -----
# Prompt user for ROS 2 version

```

```

## Prompt user for distribution
echo "Distribution"
echo "* foxy"
echo "* eloquent"
read -p "Select: " ros2_distribution
echo ""

## Prompt user for version
echo "Version"
echo "* desktop"
echo "* base"
read -p "Select: " ros2_version
echo ""

# Check if valid version
if [ $ros2_distribution != 'foxy' ] && [ $ros2_distribution != 'eloquent' ]
then
    echo "Distribution invalid: $ros2_distribution"
    exit 0
fi

# Update locales
sudo apt update && sudo apt install locales -y
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8

# Add key to apt
sudo apt update && sudo apt install curl gnupg2 lsb-release -y
curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -

# Install correct package for system
sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)] http://packages.ros.org/
ros2/ubuntu $(lsb_release -cs) main" > /etc/apt/sources.list.d/ros2-latest.list
'

# Update apt
sudo apt update

## Install
if [ $ros2_distribution == 'foxy' ] && [ $ros2_version == 'desktop' ]
then sudo apt install ros-foxy-desktop -y
fi

if [ $ros2_distribution == 'foxy' ] && [ $ros2_version == 'base' ]
then sudo apt install ros-foxy-ros-base -y
fi

if [ $ros2_distribution == 'eloquent' ] && [ $ros2_version == 'desktop' ]
then sudo apt install ros-eloquent-desktop -y
fi

if [ $ros2_distribution == 'eloquent' ] && [ $ros2_version == 'base' ]
then sudo apt install ros-eloquent-ros-base -y
fi

# Install some neat dependencies
sudo apt update && sudo apt install -y \
python3-rosdep \
build-essential \
cmake \
python3-colcon-common-extensions \
python3-flake8 \
python3-pip \
python3-pytest-cov \

```

```

python3-setuptools \
python3-vcstool \
wget

sudo rosdep init
rosdep update

echo "Finished!"
echo "Your $ros2_distribution distribution ($ros2_version version) should be
located in /opt/ros/"
exit 0

```

C.4.3 Shell-skript for FastDDS installasjon

```

# Shell script for installing FastDDS.
# -----
# Install JAVA JDK 8.
sudo apt install openjdk-8-jdk -y

# Change JAVA JDK version to JDK 8.
sudo update-alternatives --config java

sudo update-alternatives --set java /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java

# Install ZIP.
sudo apt install zip -y

# Install GRADLE via SDKMAN.
cd ~
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"

mkdir ~/FastRTPS
cd ~/FastRTPS

# Install the Foonathan Memory dependency.
git clone https://github.com/eProsima/foonathan_memory_vendor.git
cd foonathan_memory_vendor
mkdir build && cd build
sudo cmake ..
sudo cmake --build . --target install

# Install Fast-RTPS (DDS)
git clone --recursive https://github.com/eProsima/Fast-DDS.git -b v2.0.0 ~/FastRTPS/FastDDS-2.0.0
cd ~/FastRTPS/FastDDS-2.0.0
mkdir build && cd build

sudo cmake -DTHIRDPARTY=ON -DSECURITY=ON ..
sudo make
sudo make install

# Install Fast-RTPS-Gen
git clone --recursive https://github.com/eProsima/Fast-DDS-Gen.git -b v1.0.4 ~/FastRTPS/Fast-RTPS-Gen
cd ~/FastRTPS/Fast-RTPS-Gen
sudo ./gradlew assemble
sudo ./gradlew install

```

C.4.4 Shell-skript for nedlasting og bygging av arbeidsområde

```
# Shell script for creating ROS 2 workspace for PX4 bridge with companion computer

# Works with: foxy, eloquent
# ----

## Prompt user for distribution.
echo "Distribution"
echo "* foxy"
echo "* eloquent"
read -p "Select: " ros2_distribution
echo ""

# Check if valid version.
if [ $ros2_distribution != 'foxy' ] && [ $ros2_distribution != 'eloquent' ]
then
    echo "Distribution invalid: $ros2_distribution"
    exit 0
fi

# Create ws directory.
mkdir -p ~/dronesverm_ws/src

# Clone PX4 repos.
git clone https://github.com/PX4/px4_ros_com.git ~/px4_ros_com_ross2/src/
px4_ros_com
git clone https://github.com/PX4/px4_msgs.git ~/px4_ros_com_ross2/src/px4_msgs
git clone https://github.com/mdherman/ds_ross2_drone_pkg.git
git clone https://github.com/martinmaeland/ds_ross2_msgs

cd ~/dronesverm_ws

# Install some neat dependencies
sudo apt update && sudo apt install -y \
python3-rosdep \
build-essential \
cmake \
python3-colcon-common-extensions \
python3-flake8 \
python3-pip \
python3-pytest-cov \
python3-setuptools \
python3-vcstool \
wget \

# Rosdep init and update.
sudo rosdep init
rosdep update --include-eol-distros

# Source ROS 2 environment and check packages..
if [ $ros2_distribution == 'eloquent' ]
then
    source /opt/ros/eloquent/setup.bash
    rosdep install -i --from-path src --rosdistro eloquent -y
fi

if [ $ros2_distribution == 'foxy' ]
then
    source /opt/ros/foxy/setup.bash
    rosdep install -i --from-path src --rosdistro foxy -y
fi

# Install packaging.
```

```

sudo pip3 install packaging
sudo pip3 install pyros-genmsg

# Set one available core for system when running colcon build.
cpu=$(nproc)
cpu_in_use="$((($cpu-1)) "
export MAKEFLAGS="-j $cpu_in_use"

# Build packages.
colcon build --cmake-args -DCMAKE_BUILD_TYPE=RELWITHDEBINFO --executor sequential
--symlink-install --event-handlers console_direct+

```

C.4.5 Python script som logger temperatur av prosessoren til Raspberry Pi Compute Module 4.

```

import time
import os
import os.path as path

temp = 0
time_s = 0
print_time = 1

if path.exists("templog.txt"):
    os.system("rm templog.txt")

while True:
    os_temp = open("/sys/class/thermal/thermal_zone0/temp", "r")
    templog = open("templog.txt", "a")
    temp = int(int(os_temp.read())/1000)
    templog.write(str(temp) + " " + str(time_s) + "\n")
    time_s = time_s+print_time
    time.sleep(print_time)

```

Tillegg D

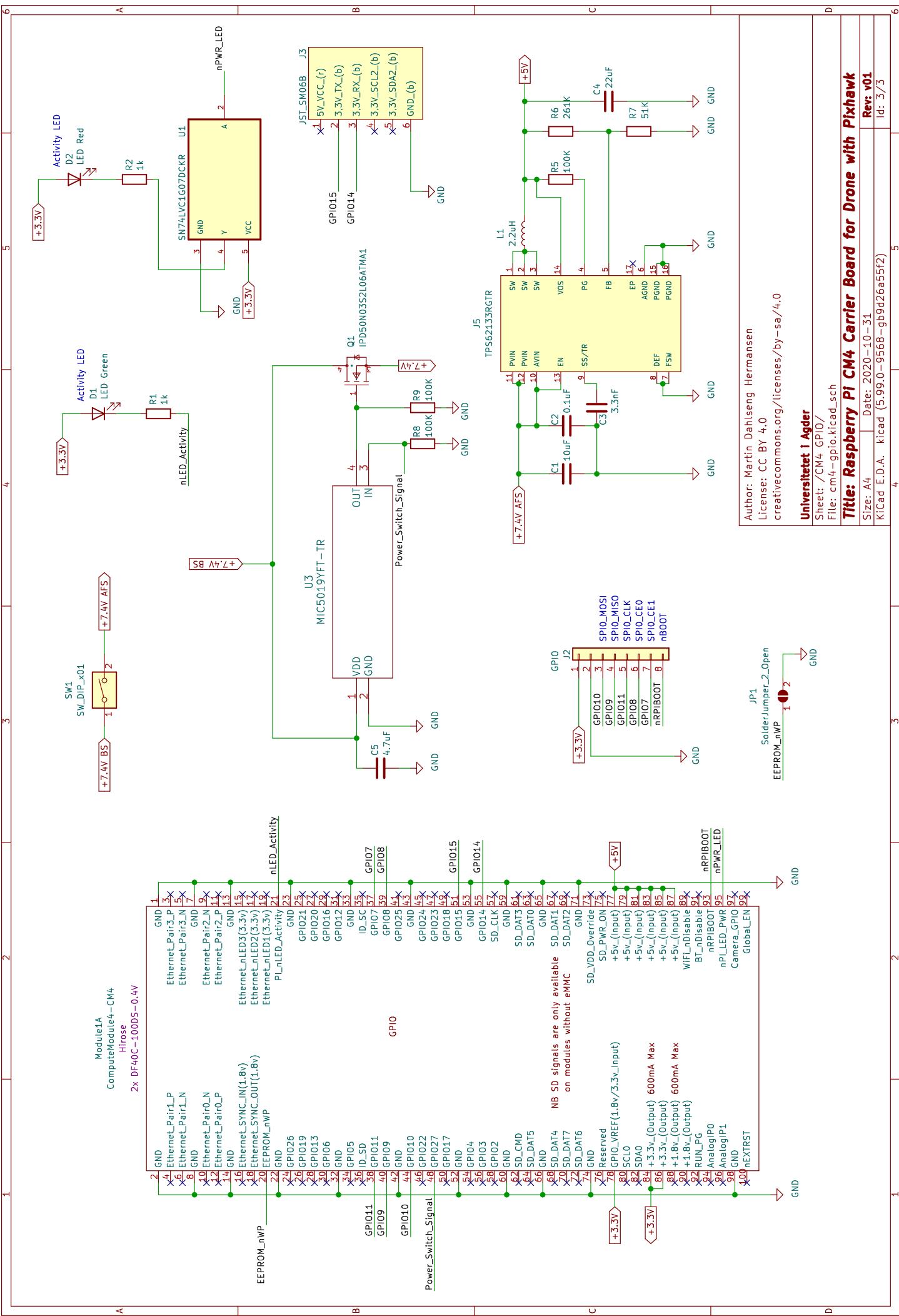
Kretsskjema

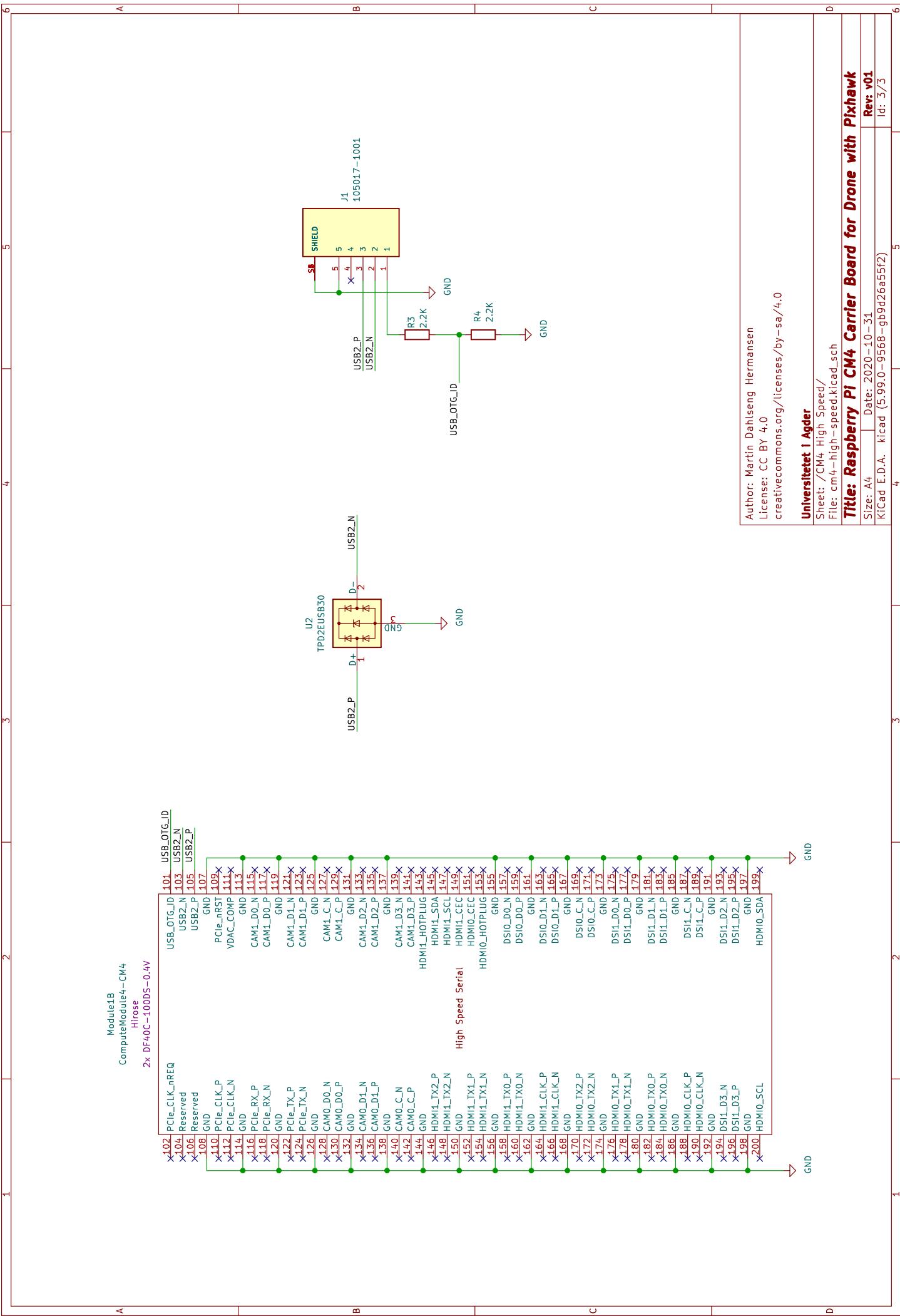
I dette tillegget er alle kretsskjema til bærekortene vedlagt.

Filene til bærekortene kan også finnes på GitHub:

- Drone: https://github.com/mdherman/cm4_cb_drone
- Utskytningsenhet: https://github.com/martinmaeland/cm4_cb_use_v01

D.1 Drone





D.2 Utskytningsenhet

This figure shows a detailed technical drawing of a CM4 Carrierboard Utstytningshet (Module 4). The board is rectangular with a central component labeled "CM4 High Speed". On the left side, there is a large blue rectangle labeled "CM4 GPIO". To the right of the central component, there is another blue rectangle labeled "File: cm4-high-speed.kicad_sch". At the top edge of the board, there are several labels: "A" at the far left, "B" in the middle, "C" on the right, and "D" at the far right. The vertical edges of the board have numerical labels: "6" at the top, "5" below it, "4" further down, "3", "2", and "1" at the bottom. The entire drawing is enclosed in a red border.

Author: Martin Mæland
License: CC BY-SA 4.0
creativecommons.org/licenses/by-sa/4.0/

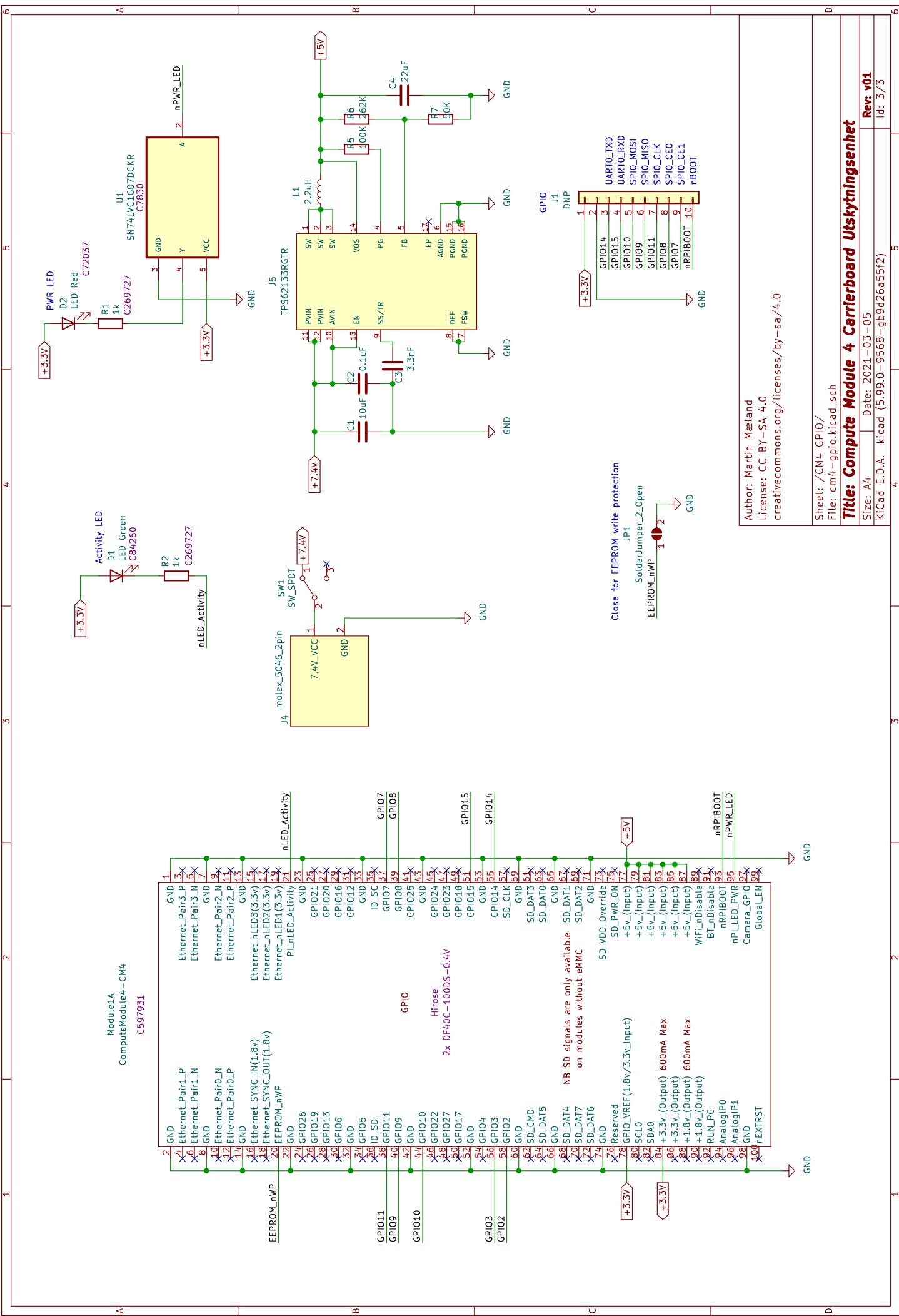
Sheet: /
File: cm4_cb_use_v01.kicad_sch

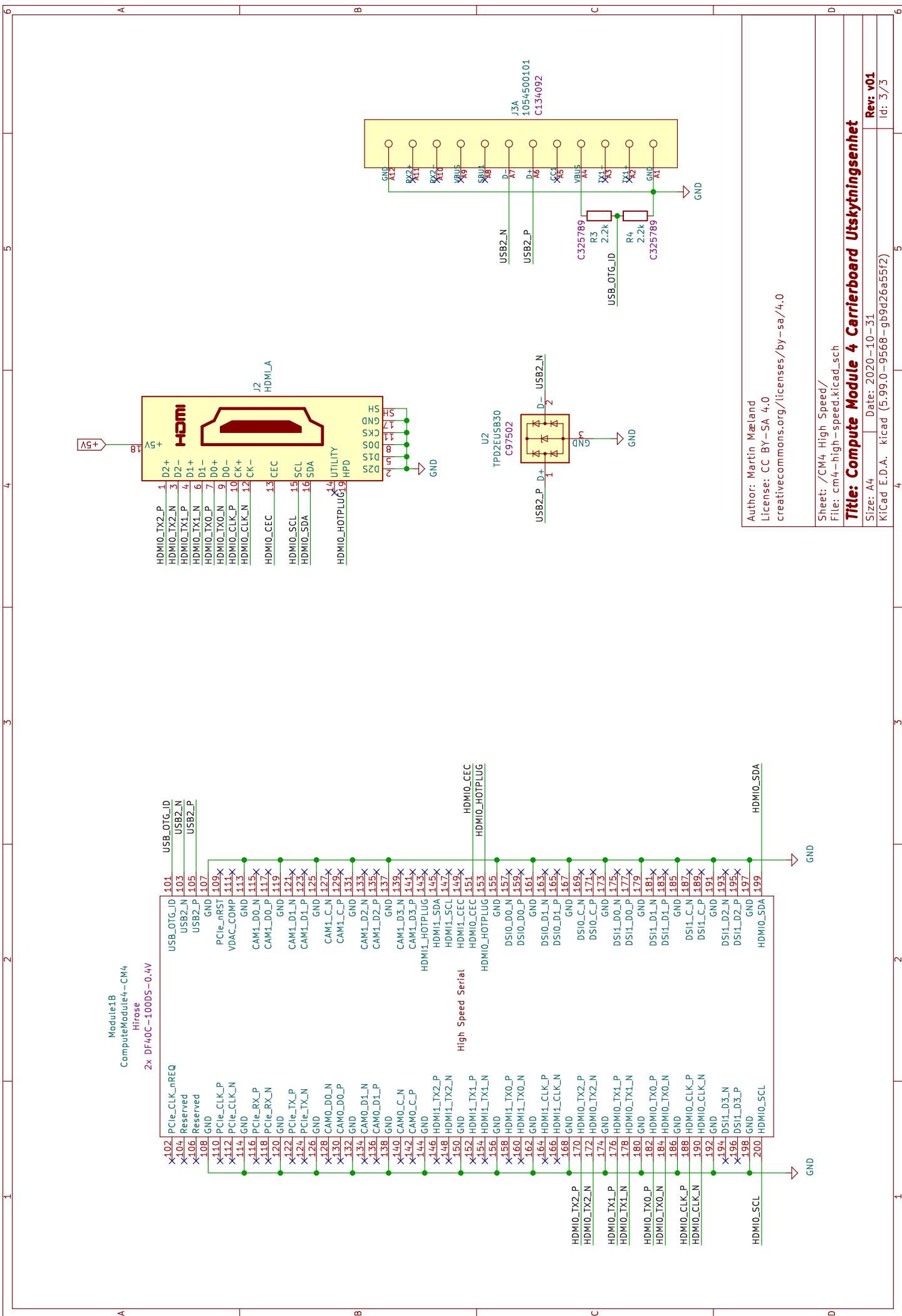
Title: Compute Module 4 Carrierboard Utstytningshet

Rev: v01
Id: 1/3

Size: A4 Date: 2020-10-31
KiCad E.D.A. kicad (5.99.0-9568-gb9d26aa55f2)

4 3 2 1 5 6





Tillegg E

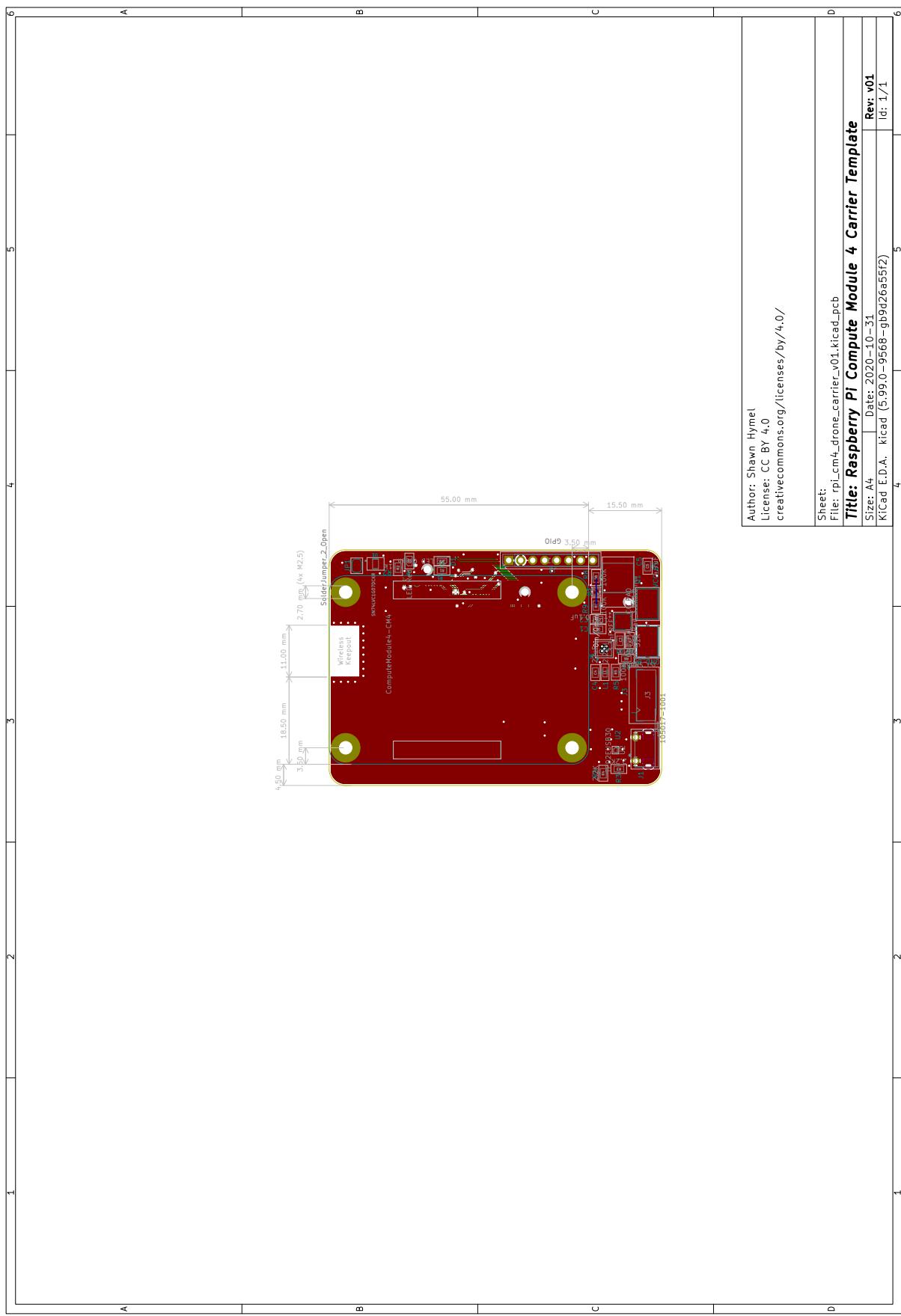
Kretsutlegg

I dette tillegget er alle kretsutleggene til bærekortene vedlagt.

Filene til bærekortene kan også finnes på GitHub:

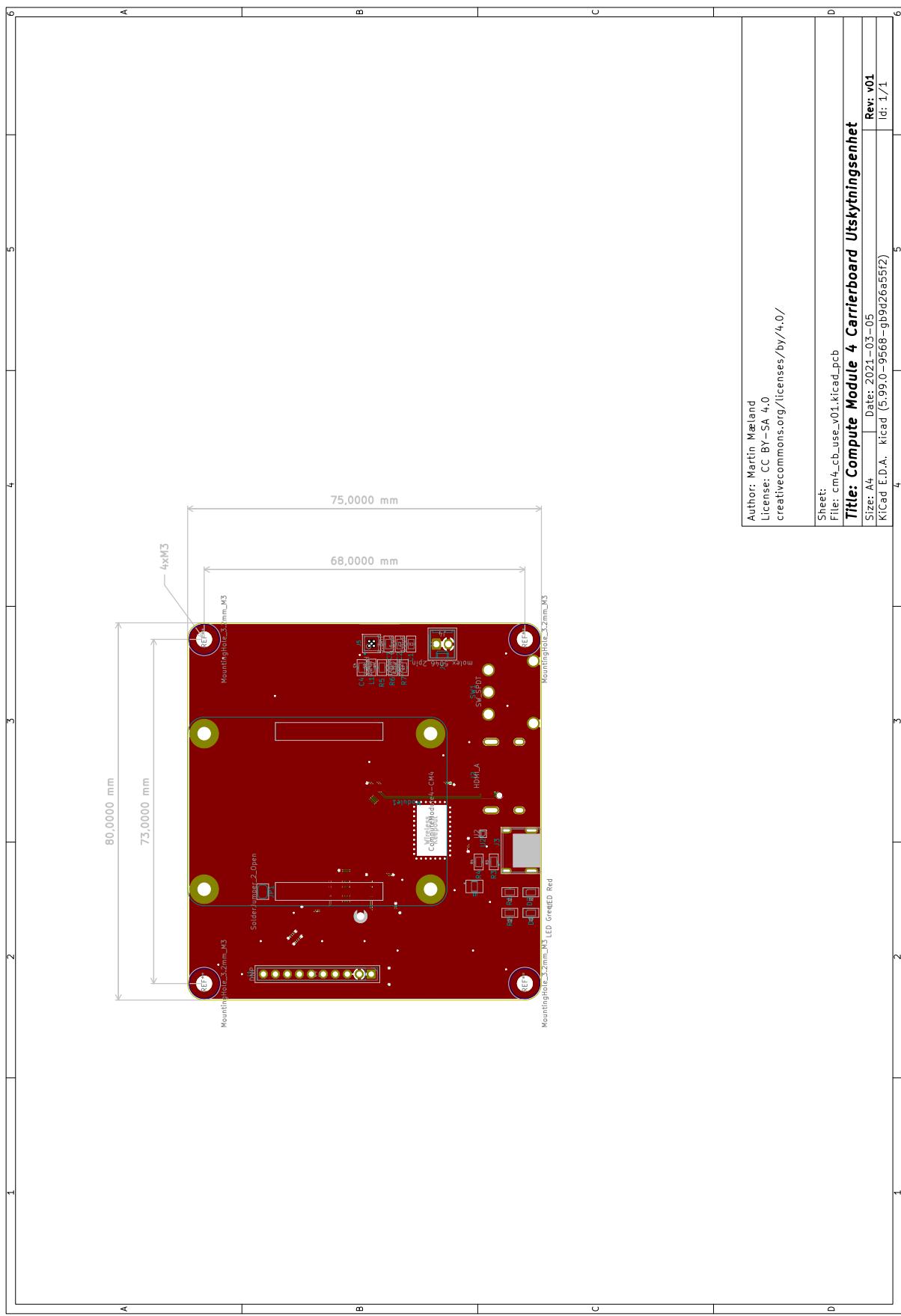
- Drone: https://github.com/mdherman/cm4_cb_drone
- Utskytningsenhet: https://github.com/martinmaeland/cm4_cb_use_v01

E.1 Drone



Figur E.1: Drone, kretskortutlegg

E.2 Utskytningsenhet



Figur E.2: Utskytningsenhet, Kretskortutlegg

Tillegg F

Komponentliste bærekort

F.1 BOM - Bærekort drone

Tabell F.1: BOM - Bærekort drone.

Antall	Navn	Verdi	Komponent	Delenummer	Produsent
1	C1	10uF	Kondensator	08053D106KAT2A	AVX
1	C2	0.1uF	Kondensator	08051C104K4T2A	AVX
1	C3	3.3nF	Kondensator	C0805C332K5RACTU	Kemet
1	C4	22uF	Kondensator	08056D226MAT2A	AVX
1	C5	4.7uF	Kondensator	08053D475KAT2A	AVX
1	D1		LED	KP-2012SGC	Kingbright
1	D2		LED	KP-2012SRC-J4	Kingbright
1	J1		Micro USB	MOLEX_105017-1001	Molex
1	J3		JST Connector	SM06B-GHS-TB(LF)(SN)	JST
1	J5		Halvleder	TPS62133RGTR	Texas Instruments
1	L1	2.2uH	Spole	LQM21PN2R2MC0D	Murata
2	Module1		Hirose DF40C-100DS konnektor	DF40C-100DS-0.4V(51)	Hirose
1	Q1		N-channel enhancement MOSFET	IPD50N03S2L06ATMA1	Infineon
2	R1,R2	1kΩ	Resistor	CRCW08051K00FKEA	Vishay
2	R3,R4	2.2kΩ	Resistor	CRCW08052K20FKEAHP	Vishay
3	R5, R8, R9	100kΩ	Resistor	CRCW0805100KFKEA	Vishay
1	R6	261kΩ	Resistor	CRCW0805261KFKEA	Vishay
1	R7	51kΩ	Resistor	CRCW080551K0FKEA	Vishay
1	U1		Buffer/Line driver	SN74LVC1G07DCKR	Texas Instruments
1	U2		ESD protection device	TPD2EUSB30DRTR	Texas Instruments
1	U3		MOSFET driver	MIC5019YFT-TR	Microchip
1	SW1		Fysisk bryter	S201031MS02Q	C&K Components

F.2 BOM - Bærekort utskytningsenhet.

Tabell F.2: BOM - Bærekort utskytningsenhet

Antall	Navn	Verdi	Komponent	Delenummer	Produsent
1	C1	10uF	Kondensator	08053D106KAT2A	AVX
1	C2	0.1uF	Kondensator	08051C104K4T2A	AVX
1	C3	3.3nF	Kondensator	C0805C332K5RACTU	Kemet
1	C4	22uF	Kondensator	08056D226MAT2A	AVX
1	D1		LED	KP-2012SGC	Kingbright
1	D2		LED	KP-2012SRC-J4	Kingbright
1	J2		HDMI konnektor	2086588131	Molex
1	J3		USB C	105450-0101	Molex
1	J5		Halvleder	TPS62133RGTR	Texas Instruments
1	L1	2.2uH	Spole	LQM21PN2R2MC0D	Murata
2	Module1		Hirose DF40C-100DS konnektor	DF40C-100DS-0.4V(51)	Hirose
2	R1,R2	1k	Resistor	CRCW08051K00FKEA	Vishay
2	R3,R4	2.2k	Resistor	CRCW08052K20FKEAHP	Vishay
1	R5	100K	Resistor	CRCW0805100KFKEA	Vishay
1	R6	261k	Resistor	CRCW0805261KFKEA	Vishay
1	R7	61k	Resistor	CRCW080551K0FKEA	Vishay
1	U1		Buffer/Line driver	SN74LVC1G07DCR	Texas Instruments
1	U2		ESD protection device	TPD2EUSB30DRTR	Texas Instruments
1	J4		2-pin Molex konnektor	22-05-1022	Molex
1	SW1		Fysisk bryter	1101M2S3AQE2	C&K Components

Tillegg G

Kostnadslister

Tabellene under er et estimat på hvor mye koster å produsere en drone og en utskytningsenhet.

G.1 Drone

Tabell G.1: Kostnadsliste, drone.

Navn/Komponent	Produksjonsmetode	Pris
Skrogbunn	3D-printet	kr 7
Lokk	3D-Printet	kr 4,20
Innfestning LiPo-batteri	3D-printet	kr 1,50
Innfestning CM4	3D-printet	kr 1
Innfestning Pixhawk	3D-printet	kr 1,25
Motor x4	Innkjøpt	kr 108
Propeller x4	Innkjøpt	kr 80
ESC x4	Innkjøpt	kr 240
Pixhawk 4 mini	Innkjøpt	kr 1 550
Strømmodul	Innkjøpt	
GPS	Innkjøpt	
LiPo-Batteri	Innkjøpt	kr 74
Bærekort m/komponenter	Innkjøpt/produsert	kr 200
Raspberry Pi Compute Module 4	Innkjøpt	kr 323
SUM		kr 2 589,45

G.2 Utskytningsenhet

Tabell G.2: Kostnadsliste, utskytningsenhet.

Navn/Komponent	Produksjonsmetode	Pris
Elektronikkboks	3D-printet	kr 3 000
Føringsstolper langside	Produsert	kr 64
Føringsstolper kortside	3D-printet	kr 100
Pleksiglass (vegger og gulv)	Innkjøpt	kr 123
Bærekort m/komponenter	Innkjøpt/produsert	kr 200
Raspberry Pi Compute Module 4	Innkjøpt	kr 323
LiPo-Batteri	Innkjøpt	kr 74
SUM		kr 3 884

Tillegg H

Tekniske tegninger

Tegningsoversikt

Tegning-settene er fordelt mellom drone og utskytningsenhet. Det er ikke produksjonstegninger, kun en intensjon for å beskrive dronens og utskytningsenhetens respektive størrelser, samt hvilke komponenter begge innholder. Kabler, kontakter og lignende er ikke tatt med.

H.1 Drone

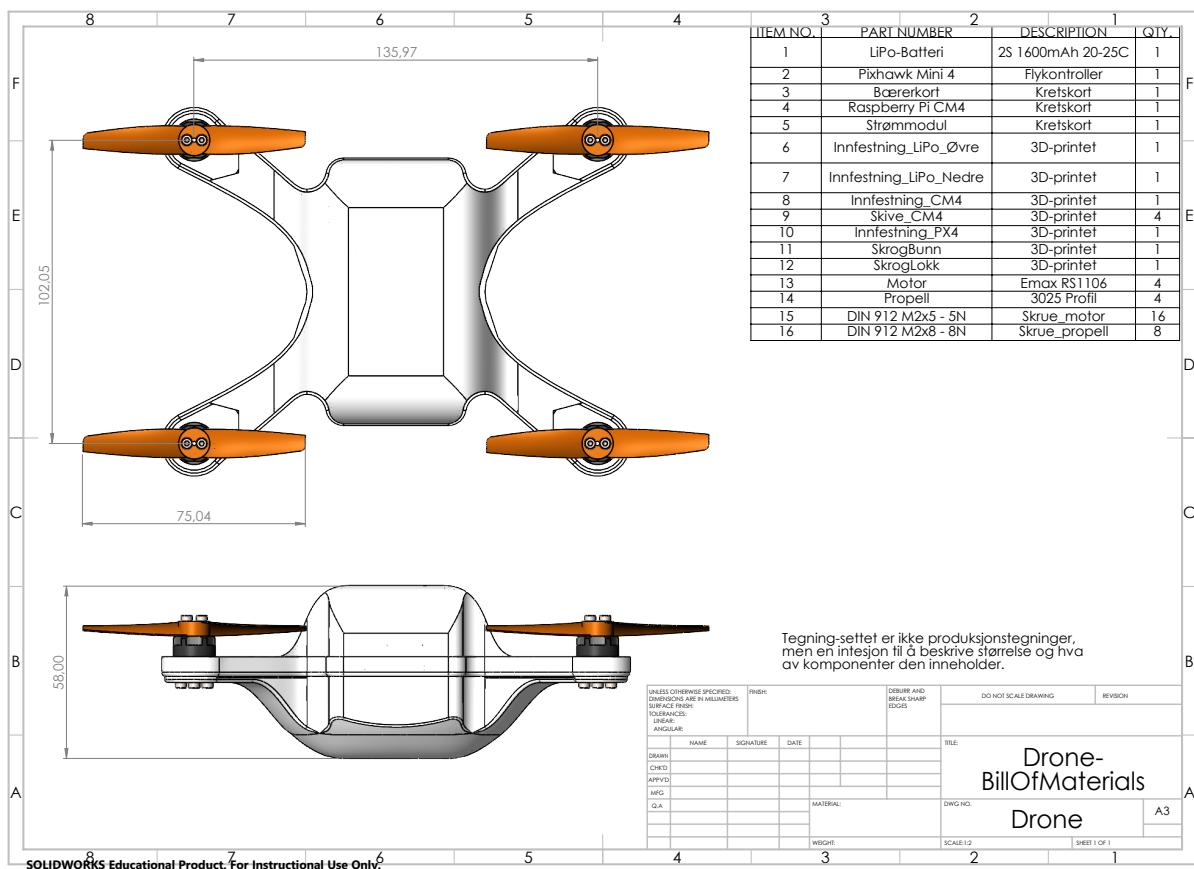
Innhold

1. Drone-BillOfMaterials
2. SkrogLokk
3. SkrogBunn
4. Innfestning-Sammentilling
5. Motor-Sammenstilling

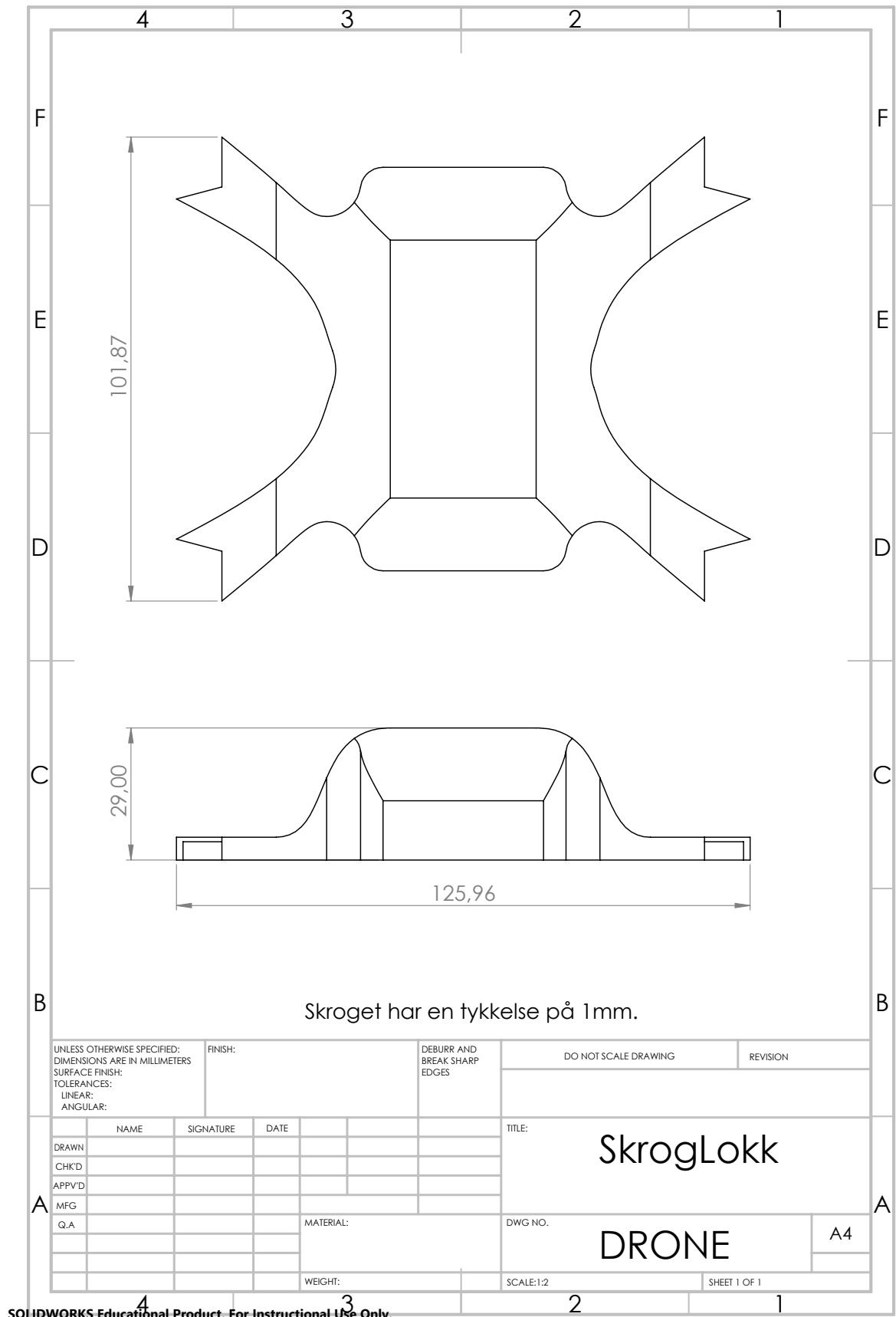
H.2 Utskytningsenhet

Innhold

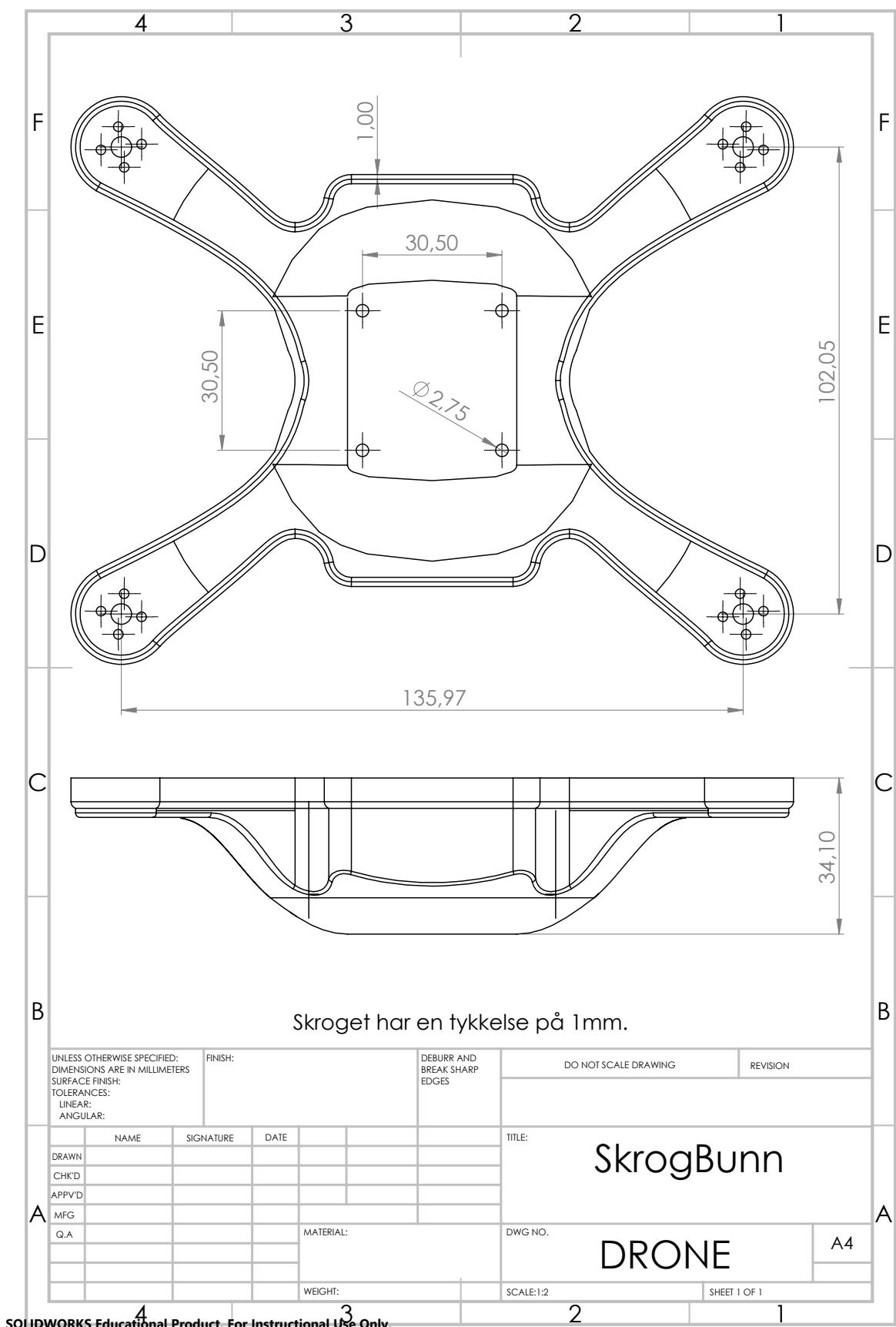
1. USE-BillOfMaterials
2. USE Nedre Konstruksjon
3. Pleksivegg Nede
4. Plexivegg-Nedre-Foran
5. Plexigulv-1
6. Plexigulv-2
7. Plexigulv-3
8. Føringsstolpe-Langside
9. Føringsstolpe-Kortside
10. USE-Nedre-Sammenstilling
11. USE-Lokk
12. Plexivegg-Øvre
13. USE-Øvre-Sammenstilling



Figur H.1: Drone, Bill Of Materials



Figur H.2: Drone, SkrogLokk



Figur H.3: Drone, SkrogBunn

ITEM NO.	PART NUMBER	QTY.
1	Innfestning_PX4	1
2	Pixhawk Mini 4	1
3	Bærerkort m/Raspberry Pi CM4	1
4	Innfestning_CM4	1
5	LiPo-Batteri	1
6	Innfestning_LiPo_Øvre	1
7	Innfestning_Lipo_Nedre	1
8	Strømmodul	1

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:	FINISH:	DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING	REVISION
DRAWN	SIGNATURE	DATE	TITLE:	
CHKD			Innfestning- Sammenstilling	
APPV'D			Drone	
MFG			DWG NO.	A4
Q.A		MATERIAL:	SCALE:1:1	SHEET 1 OF 1
		WEIGHT:		

Figur H.4: Drone. Innfestning-sammenstilling

4	3	2	1
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	Motor	Emax RS1106	1
2	Propell	3025 Profil	1
3	DIN 912 M2x5 -- 5N	Skrue	4
4	DIN 912 M2x8 -- 8N	Skrue	2

F

E

D

C

B

A

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:

FINISH:

DEBURR AND
BREAK SHARP
EDGES

DO NOT SCALE DRAWING

REVISION

NAME: _____ SIGNATURE: _____ DATE: _____

TITLE: Motor-Sammenstilling Drone

DRAWN: _____

CHK'D: _____

APP'D: _____

MFG: _____

Q.A: _____ MATERIAL: _____

DWG NO.: _____

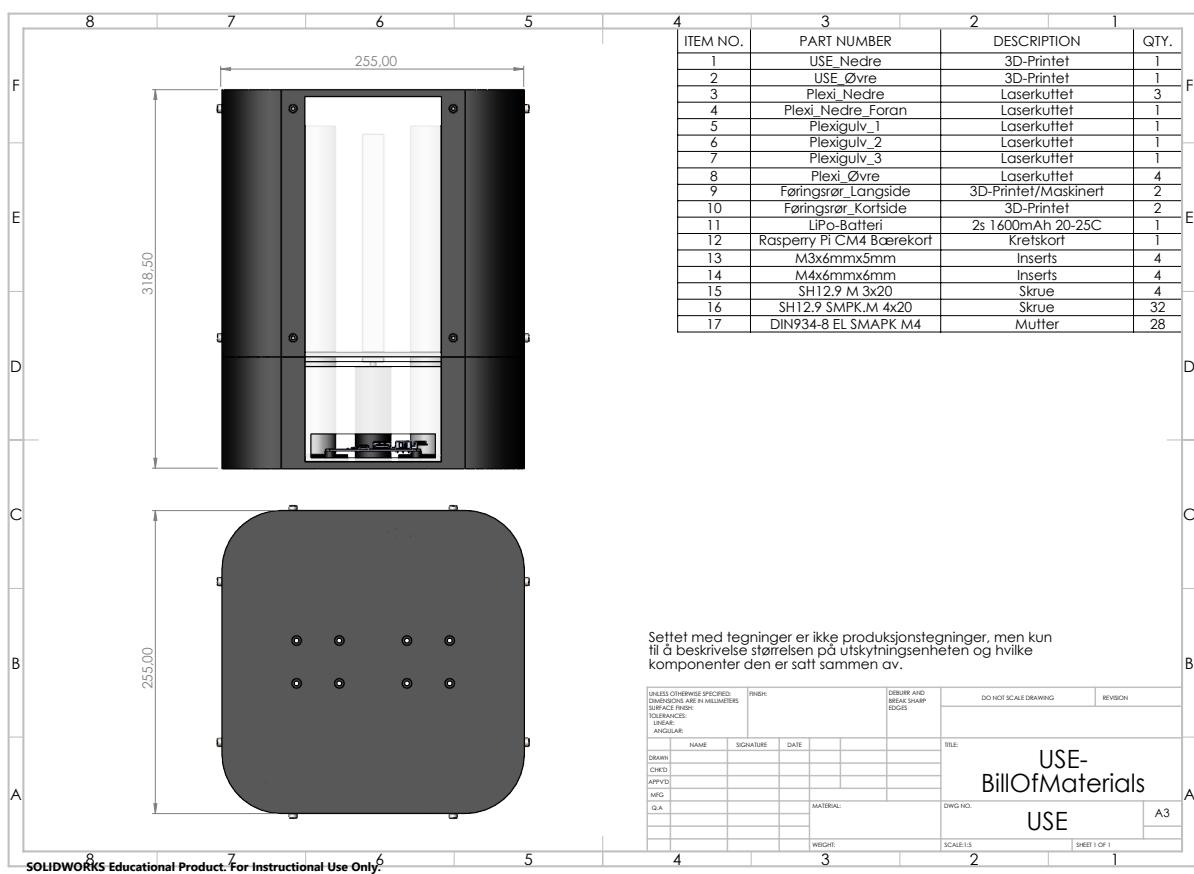
SCALE: 1:1

SHEET 1 OF 1

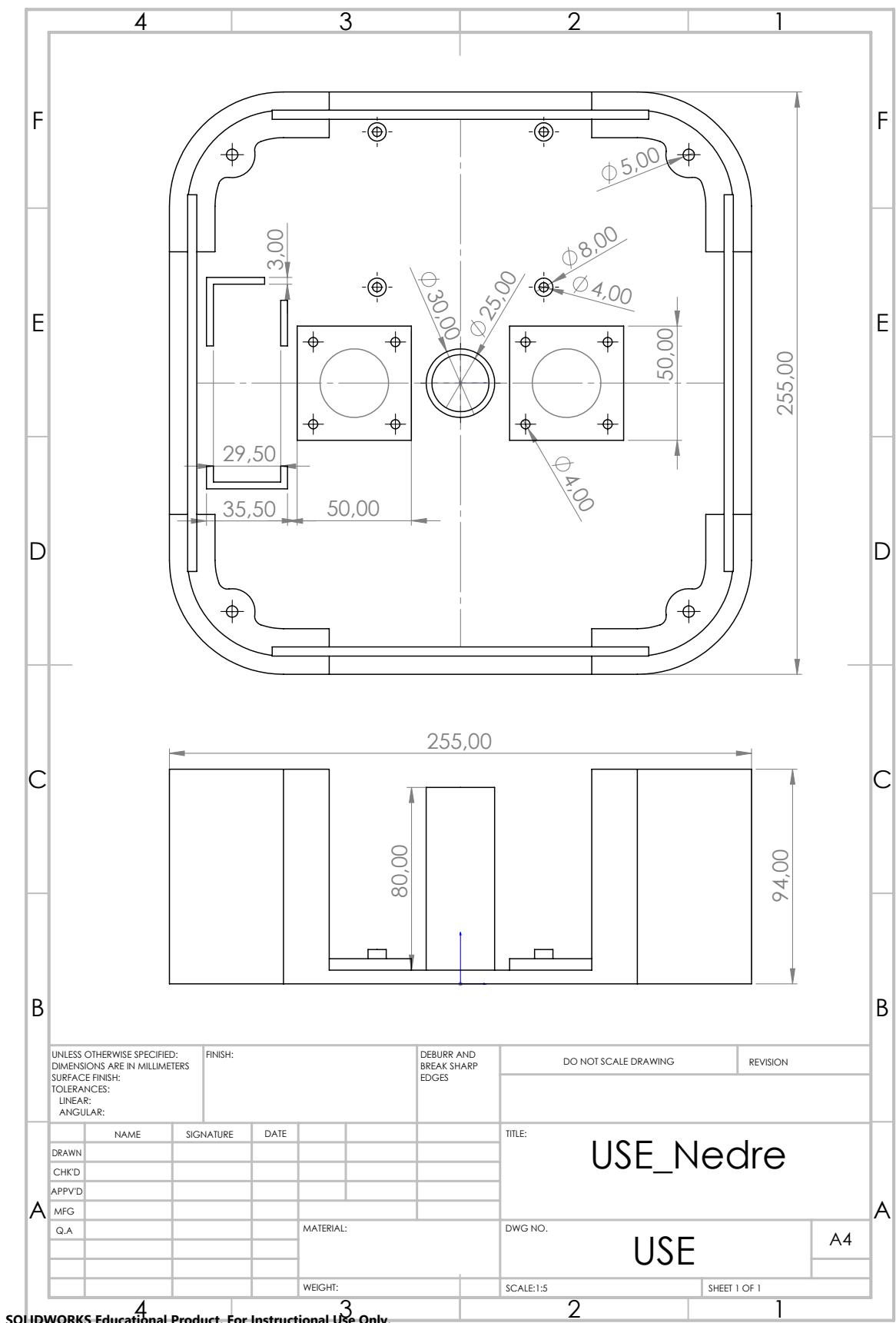
4 **3** **2** **1**

SOLIDWORKS Educational Product, For Instructional Use Only.

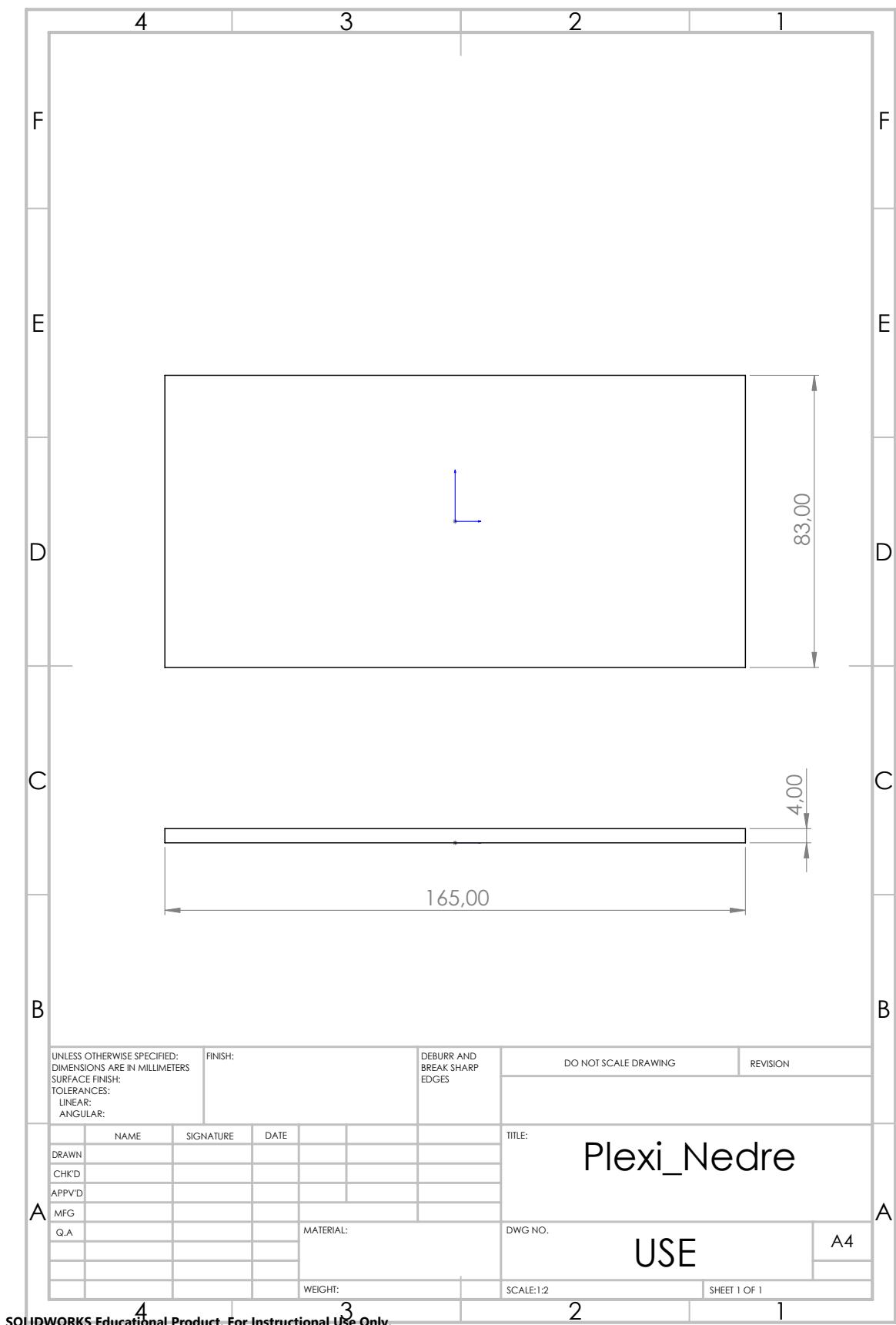
Figur H.5: USE, Motor-Sammenstellung



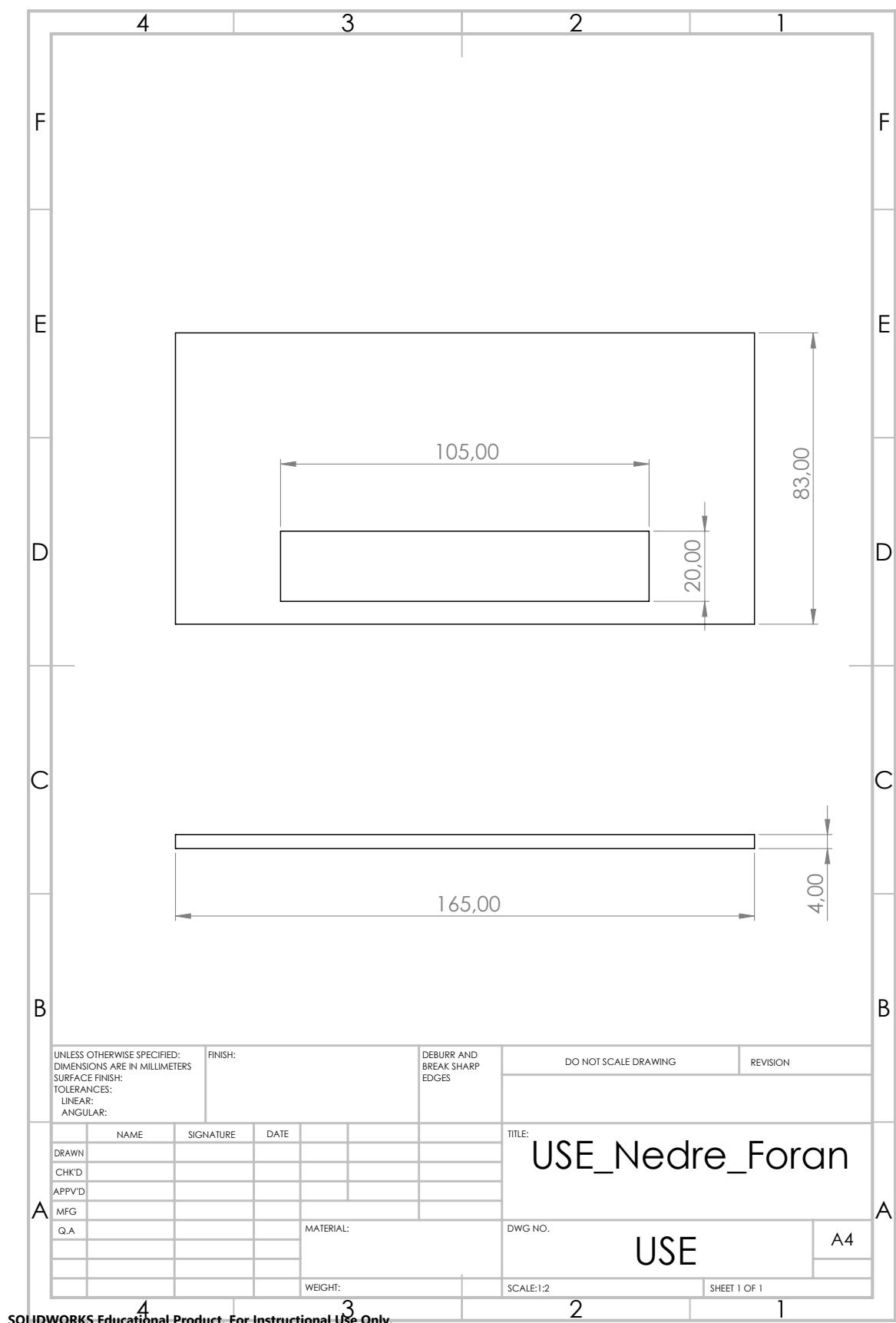
Figur H.6: USE - Bill Of Materials



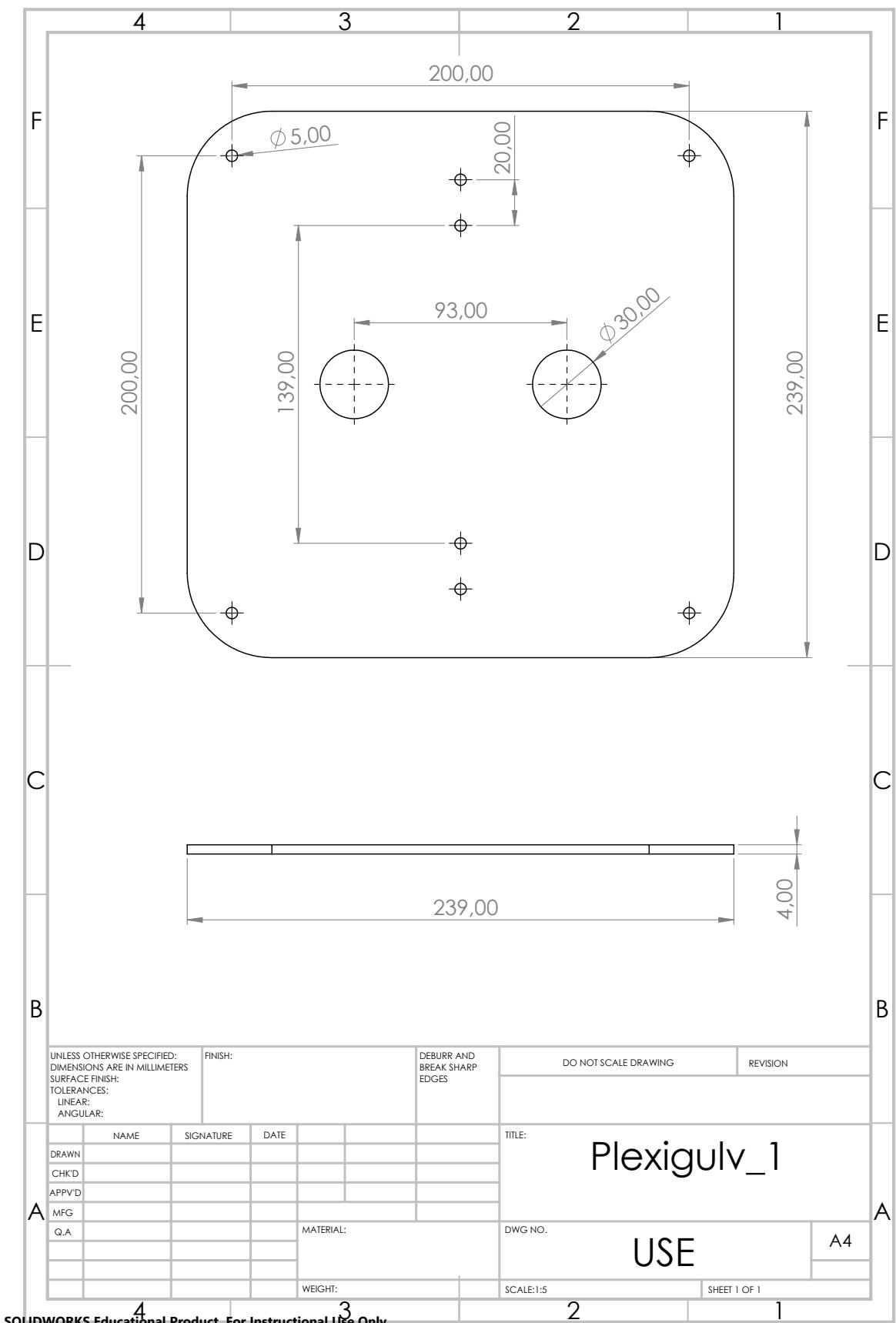
Figur H.7: USE Nedre Konstruksjon



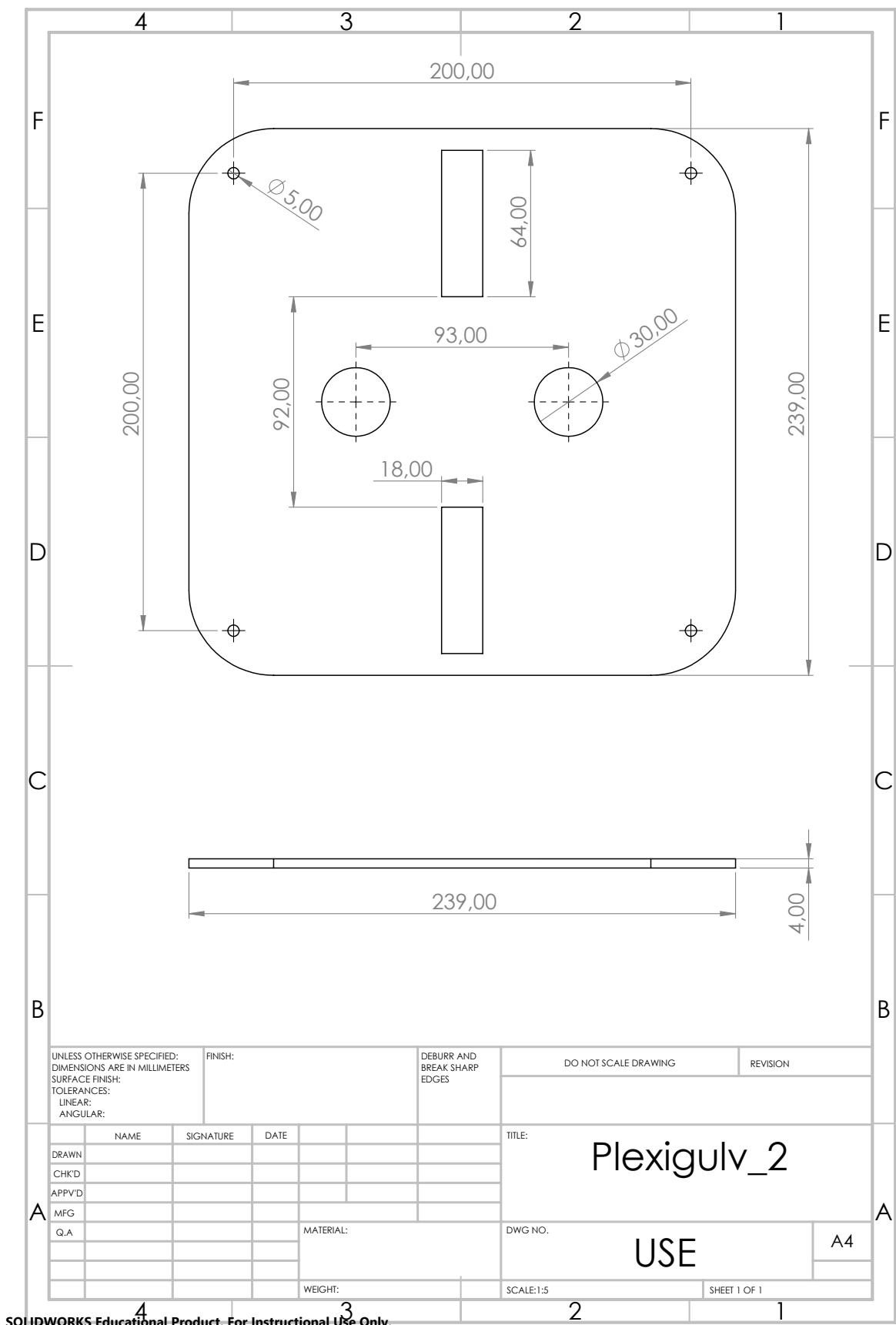
Figur H.8: Pleksivegg Nede



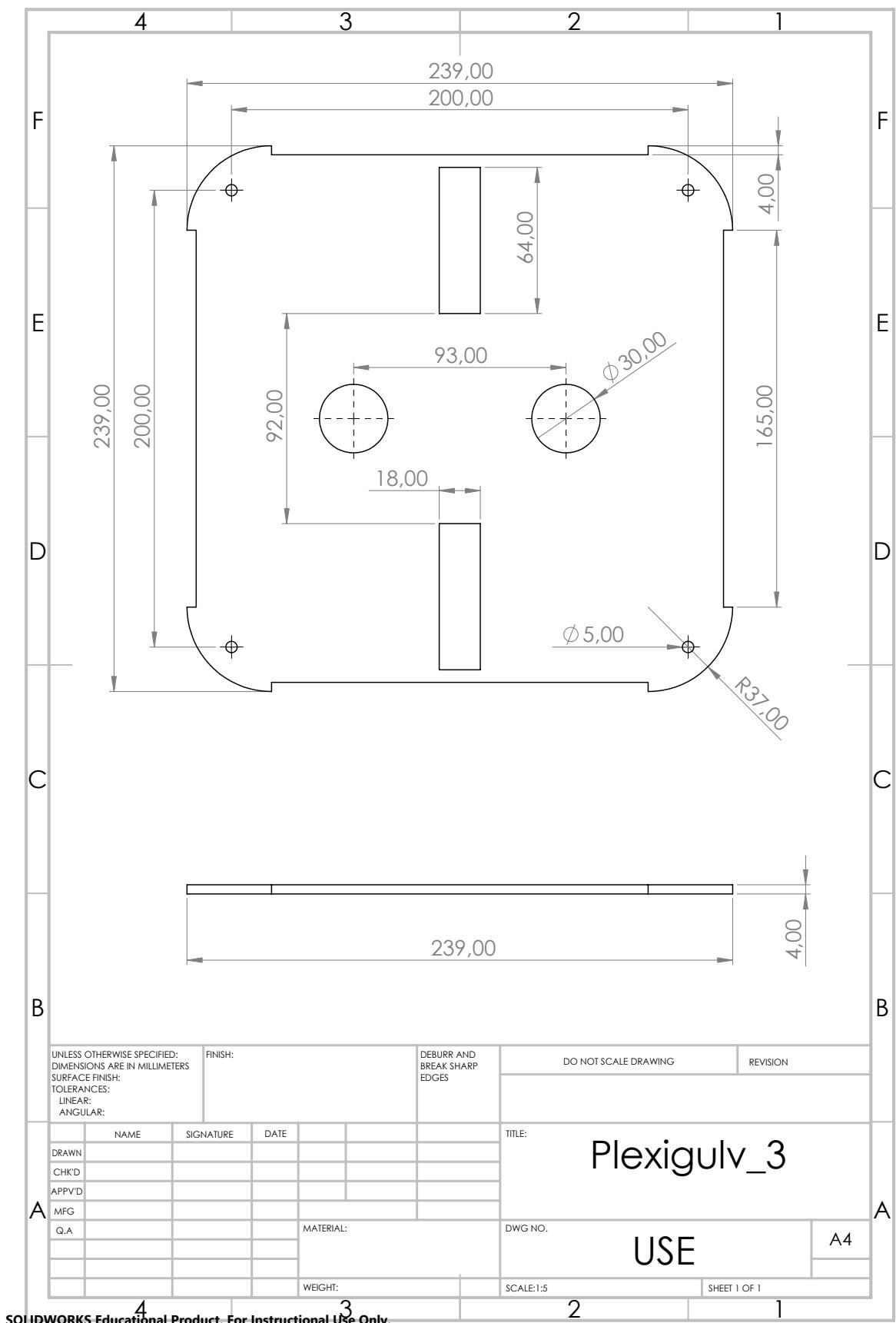
Figur H.9: Pleksivegg nede foran



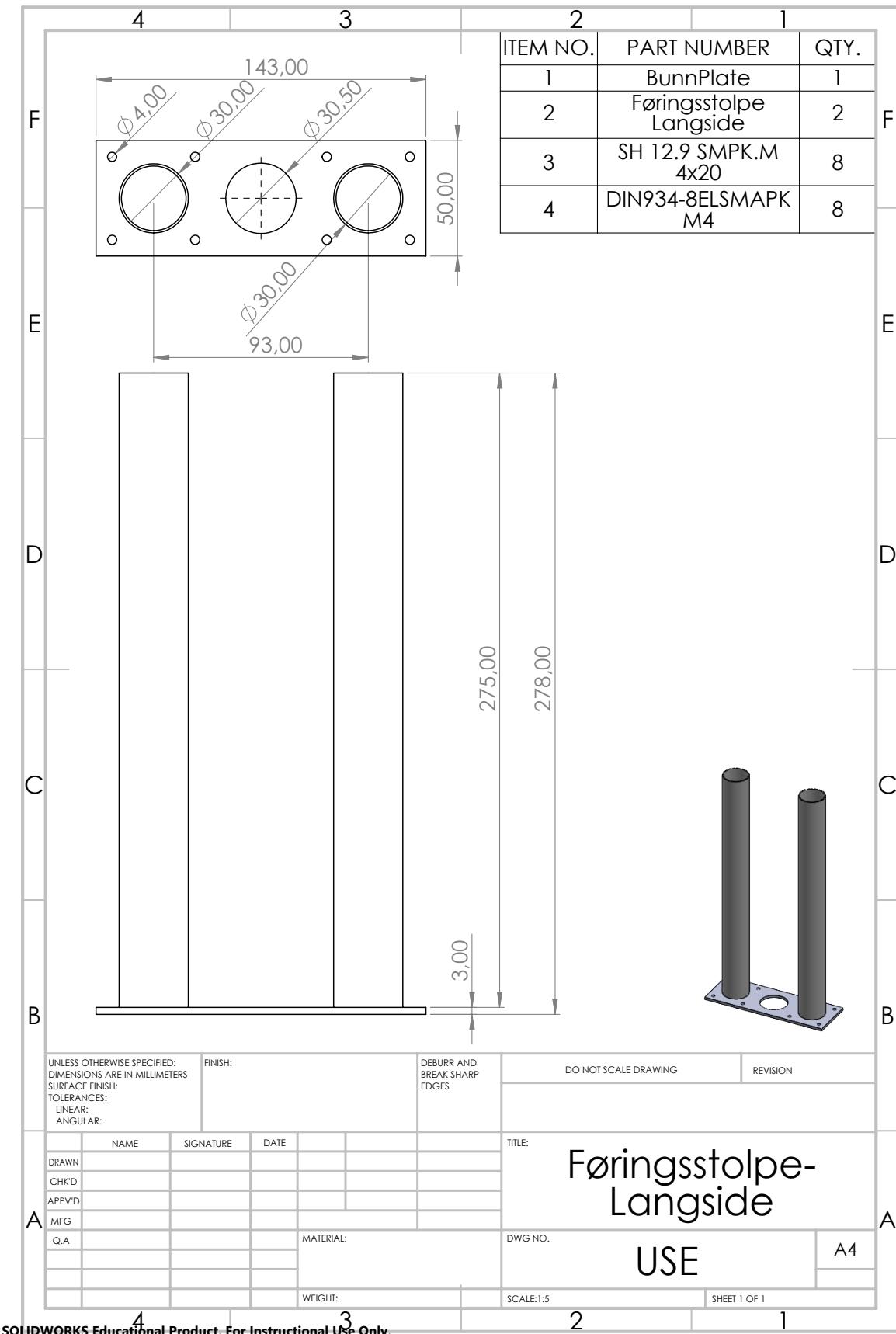
Figur H.10: Pleksiguly 1



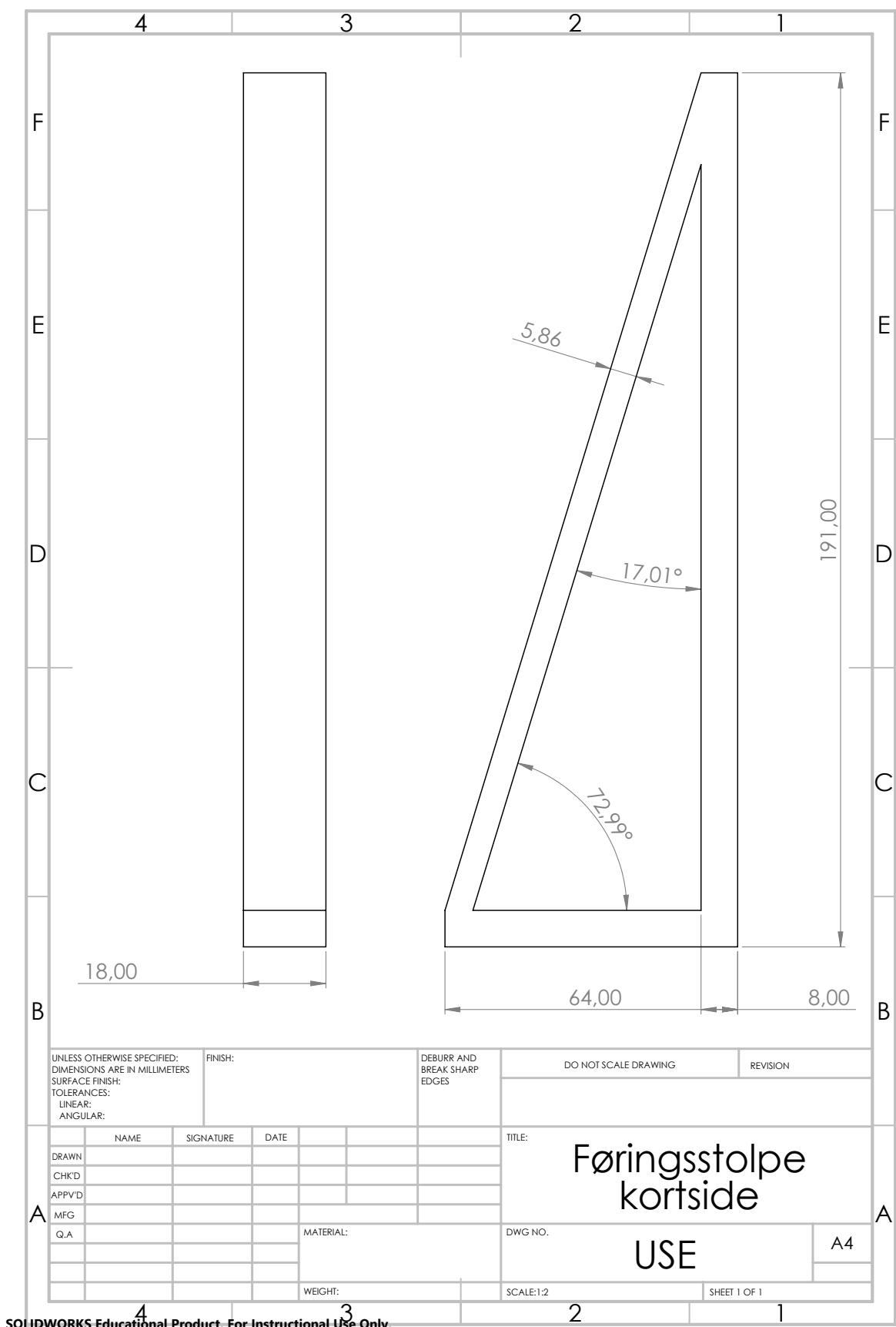
Figur H.11: Pleksigulv 2



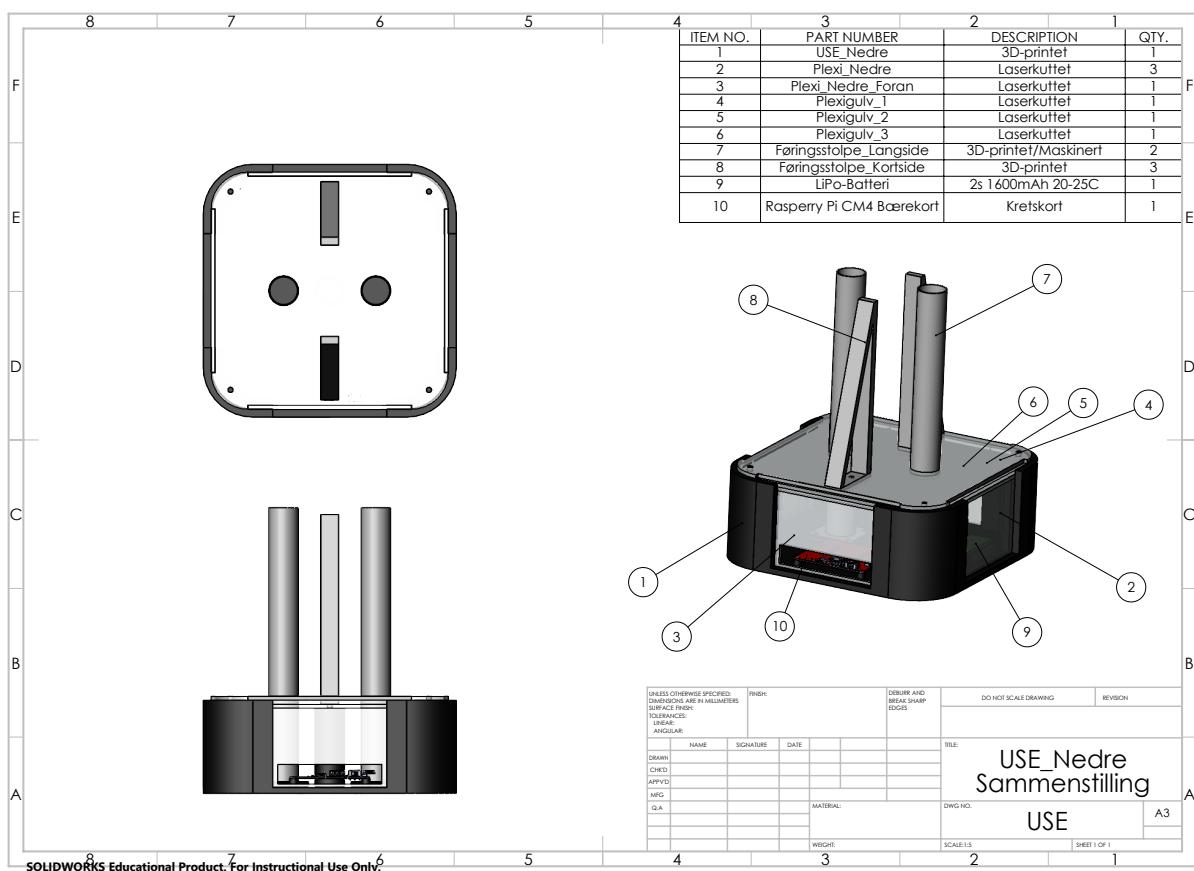
Figur H.12: Pleksigulv 3



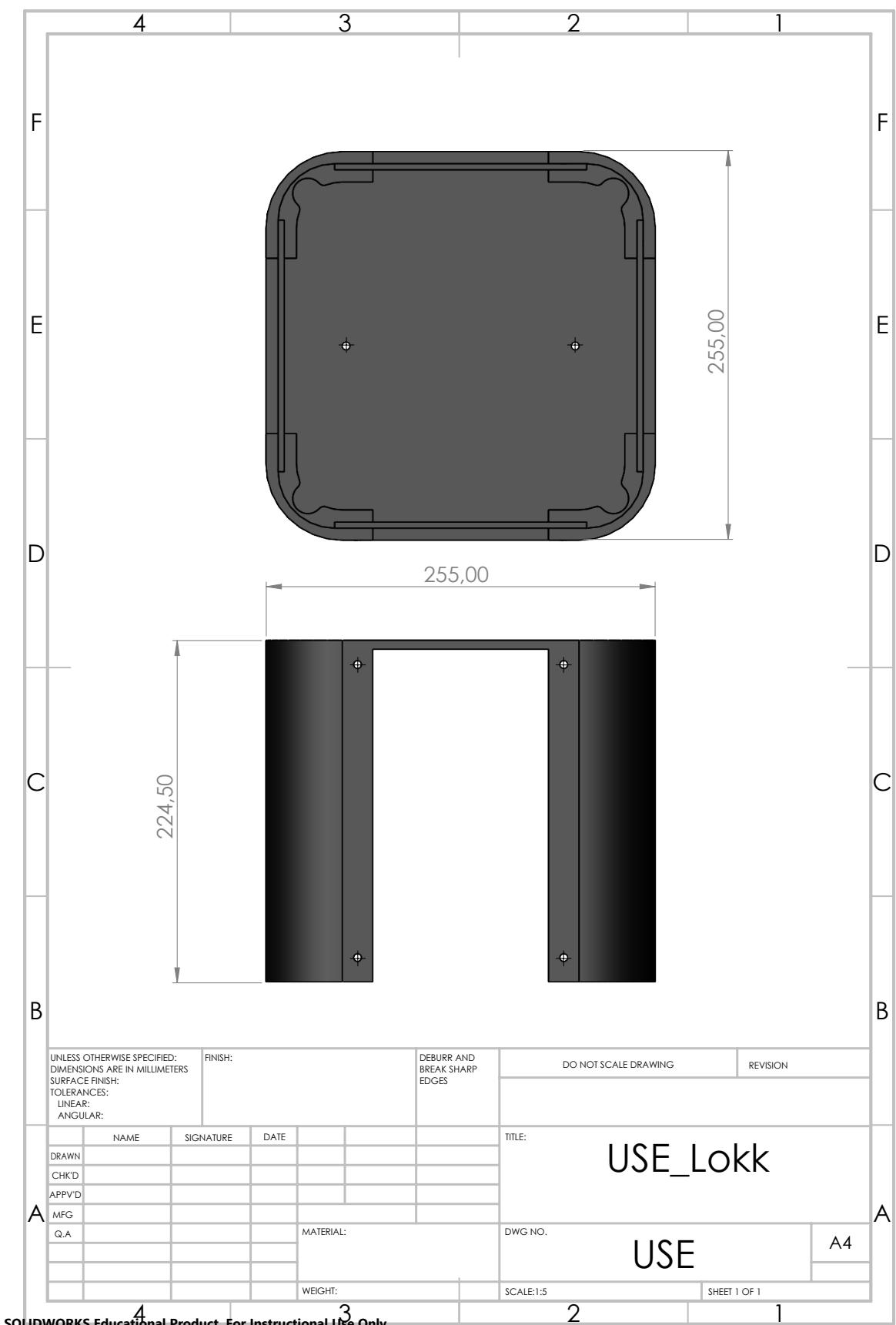
Figur H.13: Føringsstolpe Langside



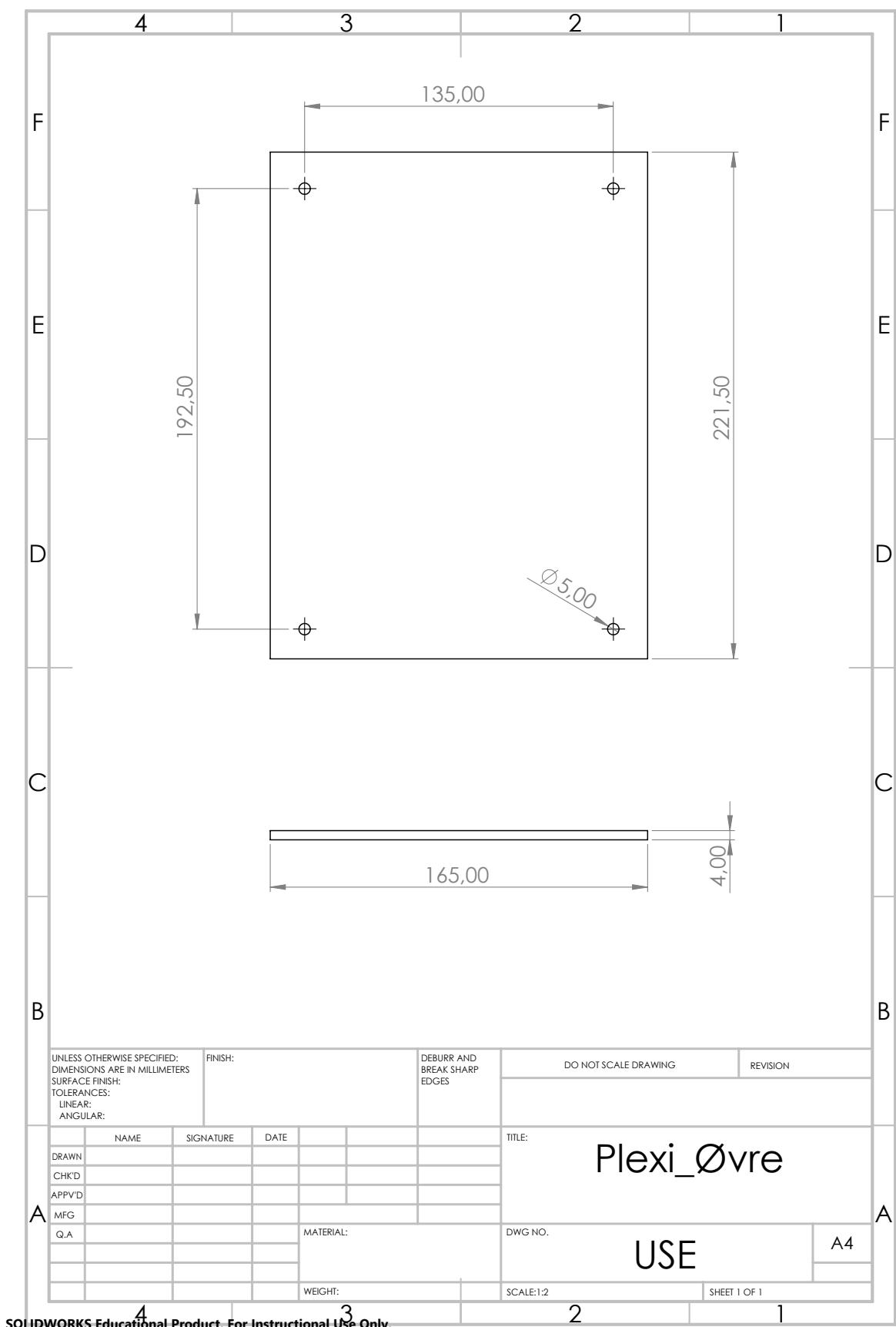
Figur H.14: Føringsstolpe Kortside



Figur H.15: USE Nedre Sammenstilling

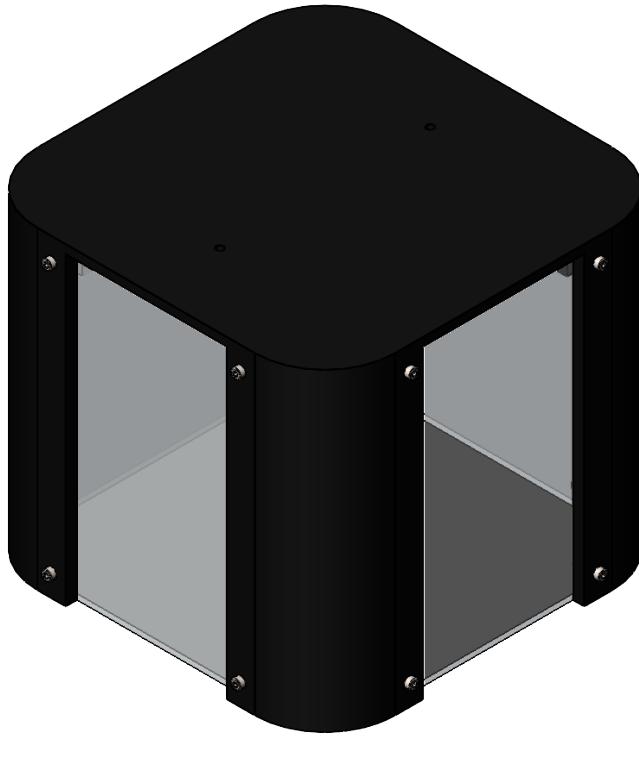


Figur H.16: Lokk



Figur H.17: Lokk Pleksivegg

ITEM NO.	PART NUMBER	QTY.
1	USE_Øvre	1
2	Plexi_Øvre	4
3	M4 Skrue	16
4	M4 Mutter	16



UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN MILLIMETERS SURFACE FINISH: TOLERANCES: LINEAR: ANGULAR:		FINISH:		DEBURR AND BREAK SHARP EDGES	DO NOT SCALE DRAWING		REVISION
DRAWN	NAME	SIGNATURE	DATE		TITLE:		
CHK'D							
APP'V'D							
MFG							
QA					MATERIAL:		DWG NO.
					WEIGHT:		SCALE:1:5
							SHEET 1 OF 1

USE_Øvre
Sammenstilling
USE

A4

Figur H.18: Lokk Sammenstilling