NTNU
Kunnskap for en bedre verden

Automatisk detektering og flytting av drone

Plassering i batteribyttemaskin ved hjelp av robotarm

Per Forester
Adrian Holmeset
Hans Einar Skinnarland

Bachelor i elektronikk
Innlevert: 20. mai 2019
Hovedveileder: Knut Wold

Norges teknisk-naturvitenskapelige universitet
Institutt for elektroniske systemer

NTNU
Kunnskap for en bedre verden

Oppgavens tittel: Automatisk detektering og flytting av drone - Plassering i batteribyttemaskin ved hjelp av robotarm	Dato: 20. mai 2019 Antall sider: 81			
	Masteroppgave:		Bacheloroppgave:	X
Navn: Per Forester, Adrian Holmeset, Hans Einar Skinnarland				
Veileder: Knut Wold og Pål Erik Endrerud				
Eventuelle eksterne faglige kontakter/ veiledere: Jonas Moen				

Sammendrag:
Forsvarets forskningsinstitutt (FFI) har en visjon om å lage en autonom dronesverm. Den skal bestå av droner opp til 250 gram og operere over lengre tid uten behov for menneskelig interaksjon. Tidligere har det blitt laget et forslag til drone og batteribyttemaskin for denne svermen.

Denne oppgaven videreutvikler dette konseptet ved å implementere en servicearm som skal kunne forflytte dronen fra en landingsplattform til batteribyttemaskinen. Servicearmen som implementeres er et av de første stegene på veien mot en servicestasjon som både kan utføre batteribytte og et begrenset vedlikehold av dronene.

Løsningen prosjektet har resultert i er et sammensatt system i stand til, helt uten menneskelig interaksjon, å lokalisere en drone på en landingsplattform for deretter å plukke den opp og plassere den i en batteribyttemaskin. Systemet kommuniserer direkte med dronen for å redegjøre orientering etter landing. Stasjonen har implementert datasyn som jobber i det infrarøde spekteret for å lokalisere dronen. Basert på denne informasjonen kan servicestasjonen selv regne ut og initiere den nødvendige bevegelsen for robotarmen til å gripe dronen og forflytte den til batteribyttemaskinen.

Stikkord:

Robot Operating System (ROS)
Robotarm
Drone
Datasyn

Per Forester Adrian Holmeset Hans Einar Skinnarland

Abstract

The Norwegian Defence Research Establishment (FFI) has a long-term vision of creating an autonomous drone swarm. Consisting of 250 gram drones, it should be able to operate over a longer period of time without the need of human interaction. Previous bachelor's theses have designed a suggested drone, and battery exchanger device for the aforementioned swarm vision.

This project expands on the previous work by developing a service arm capable of picking up a drone from a landing platform, and placing it into the battery exchanger device. This service arm is one of the first steps towards a service station which is able to swap batteries, and also performing limited maintenance of the drones.

The developed solution has the ability to, without any form of human interaction, detect a drones location on a landing platform, pick it up, and place it into a battery exchanger device. The system communicates with the drone directly to get its orientation after landing. The system also employs computer vision in the infrared spectrum to locate the drone within the landing area. Based on this input the service station can calculate the necessary movement needed for the robot arm to pick up the drone, and move it to the battery exchanger device.

Forord

I dette bachelorprosjektet har tre bachelorstudenter ved NTNU utviklet en løsning for å automatisk detektering og forflytting av en 250 gram drone, og installert denne i tilknytning til en batteribyttemaskin.

Studentene er fordelt over studieretningene elektronikk, henholdvis i Gjøvik og Trondheim, og automasjon i Trondheim. Motivasjonen bak oppgaven ligger i Forsvarets forskningsinstitutt sin visjon om en autonom dronesverm. En slik dronesverm må ha evnen til å lade opp dronenes batterier for å ha en tilstrekkelig operasjonstid. Det er utviklet en batteribytte- og lademaskin tidligere, men oppdragsgiver har ingen drone som er i stand til å lande med nøyaktigheten denne maskinen krever. Forsvarets forskningsinstitutt har på bakgrunn av dette utstedt en oppgave som søker å plassere dronen i maskinen.

Studentene vil gjerne rette en spesiell takk til følgende personer som har gitt veiledning underveis i arbeidet:

- Jonas Moen og Olav-Rune Nummedal ved Forsvarets forskningsinstitutt
- Torleif Anstensrud for veiledning i Trondheim
- Knut Wold og Pål Erik Endrerud for veiledning i Gjøvik
- Åsmund Stavdahl - for god hjelp til fresing av kretskort
- Glenn Angell - for god hjelp til 3D-printing

Kildekoden til det totale systemet prosjektet har resultert i kan lastes ned i sin helhet fra:
`https://bitbucket.org/Adrianho1/bach_ws/`

Innhold

Figurer

Tabeller

Ordliste

Definisjoner

Autonom	Selvstyrende
Driver	En driver er en samling filer som muliggjør kommunikasjon mellom maskinvare og et operativsystem.
Full dupleks	To enheter er i stand til å kommunisere med hverandre samtidig
Halv dupleks	To enheter er i stand til å kommunisere med hverandre, men ikke samtidig
IC	Integrated Circuit. Integrert krets på halvledermateriale. Ofte en avansert krets som oppnår én funksjon
Plug and play	Utrykker at utstyr er klart til å brukes etter at tilkoblingen er utført
QR-kode	Quick Response kode. Horisontale og vertikale linjer utgjør en todimensjonal strekkode.
Seriell kommunikasjon	Kommunikasjon der bit sendes etterhverandre.
Servicearm	En robotarm med mulighet for å inneha flere funksjoner
SiP	System in Package. Et system i en pakke som består av flere integrerte kretser. Ikke nødvendigvis kretser på samme halvledermateriale.
SoC	System on Chip. En integrert krets bestående av flere kretser på samme halvledermaterial. SoC'en kan således inneha flere funksjoner.
Wiki	En samarbeidsbasert kunnskapsdatabase.

Forkortelser

FD	Forsvarsdepartementet
FFI	Forsvarets Forskningsinstitutt
IR	Infrarød
LAN	Local Area Network
LED	Lysemitterende diode
NTNU	Norges teknisk-naturvitenskapelige universitet
ROS	Robot Operating System
UART	Universal asynchronous receiver-transmitter
UR	Universal Robots
USB	Universal Serial Bus
XML	eXtensible Markup Language

1 Innledning

1.1 Bakgrunn og motivasjon

Forsvarets forskningsinstitutt (FFI) er et tverrfaglig institutt underlagt Forsvarsdepartementet (FD), som gjennom samarbeid med andre ledende forskningsinstitusjoner har til hensikt å drive innovasjon innen moderne høyteknologi. Utover dette yter FFI også et betydelig bidrag til Forsvarets langtidsplanlegging gjennom analyser og utviklingsprosjekter etter Forsvarets behov.

Det er identifisert et behov for sensorsystemer som muliggjør autonom overvåking og informasjonsinnhenting av land- og maritime områder gjennom luftdomenet. I denne forbindelse er det ønskelig å anvende droner i sverm som kan operere selvstendig over lengre tid. Det er behov for utstyr som kan utføre bytte av slitasje- og forbruksmateriell, eksempelvis propeller, sensorinstrumenter eller batterier, med minimalt behov for menneskelig interaksjon.

1.2 Problemstilling og avgrensning

Problemstilling

FFI har en visjon om en autonom dronesverm bestående av 250 gram droner som skal kunne operere med et minimalt behov for menneskelig interaksjon. Instituttet har derfor utstedt flere bachelor- og masteroppgaver som sammen er med på å løse denne oppgaven. I fjor ble det laget en batteribyttemaskin for 250 g droner som foretar et automatisk batteribytte etter at dronen er plassert i maskinen. Dagens situasjon er at dronen ikke er i stand til å lande med den nøyaktigheten som denne maskinen krever($\pm$6,0 mm), og det er derfor utstedt en oppgave med følgende problemstilling, som skal løse dette:

Det skal installeres en servicearm som gjør det mulig a plassere en drone i batteribyttemaskinen. Servicearmen skal kunne gripe dronen uavhengig av dronens landingsposisjon og -orientering.

Avgrensniger

Gruppen vil benytte seg av droneutkastet som foreligger, og som ble benyttet i tidligere bacheloroppgave. Små modifikasjoner kan forventes å bli gjort på dronen.

Gruppen vil ikke ta for seg noen viderutvikling av batteribyttemaskinen utviklet i nevnte oppgave.

Gruppen, i samråd med oppdragsgiver, vurderer oppgaven til å være av en art som har til hensikt å demonstrere og drøfte et mulig løsningskonsept fremfor å lage en robust løsning for den spesifikke situasjonen som foreligger.

1.3 Prosjektmål

Dette delkapittelet tar for seg de ulike målene som er knyttet til oppgaven. Det sammensatte målet er oppdelt i effekt-, resultat- og prosessmålene som følger.

1.3.1 Effektmål

Effektmål for denne oppgaven er å komme et steg videre på veien mot FFI sin visjon om en autonom dronesverm som behøver minimal interaksjon fra mennesker. Oppgaven bygger på tidligere utført arbeid og forskning, og vil derfor utbedre en begrensning fra foregående arbeid.

1.3.2 Resultatmål

Resultatmålet av denne oppgaven vil være å implementere nødvendig utstyr for å kunne detektere dronen, og installere en servicearm som kan flytte denne fra en landingsplattform til batteribyttemaskinen som ble utviklet i et bachelorprosjekt året før.

1.3.3 Prosessmål

Prosessmålet for gruppen er å benytte seg av ferdigheter opparbeidet seg i løpet av 3 år med høyere utdanning, samtidig som de tilegner seg ny, relevant kunnskap. Studentene som danner gruppen går ulike retninger innfor elektroingeniørstudiet, og et mål er å utnytte den tverrfaglige gruppesammensetningen for å løse oppgaven.

Gruppen er spredt over forskjellige universitetsområder, og det er et mål å lære seg å samarbeide over avstand, med nye teknikker enn hva et tradisjonelt gruppearbeid krever.

1.4 Oppbygning av rapporten

Prosjektrapporten er bygd opp som følgende. Kapittel 2 er et teorikapittel som introduserer teori som er relevant for alle deler av oppgaven. Teorien omhandler operativsystemet, rammeverket for programmering av roboter og program som ble brukt for å lage fysiske deler til prosjektarbeidet. Kapittel 3 bryter ned problemstillingen og beskriver løsningskonseptet for oppgaven. Kapittel 4, 5 og 6 omhandler moduler for forflytting, finne plassering og orientering til dronen i dette prosjektet. Disse modulene inneholder relevant teori, introduksjon av utstyret som blir brukt, metoden, resultat og drøfting av disse modulene hver for seg. I tillegg vil hver av modulene ha et delkapittel som utdyper alternative løsningsmetoder for de ulike modulene. Kapittel 7 beskriver resultatet av det sammensatte systemet og dets virkemåte. I kapittel 8 blir prosjektets helhet drøftet og det blir presentert gruppens syn på fremtidig arbeid knyttet til prosjektet. Kapittel 9 reflekterer rundt etiske problemer tilknyttet oppgaven, tanker rundt gruppearbeidet og gangen i arbeidet. Kapittel 10 konkluderer oppgaven. Vedlagt ligger forprosjektet for denne oppgaven og egenutviklet kildekode.

1.5 Risikovurdering

Risiko tilknyttet denne oppgaven kan i sin helhet underlegges en av to kategorier: Risiko som fører til redusert gjennomføringsevne av oppgaven, og risiko som medfører personell- eller materiellskader.

Risiko som medfører redusert gjennomføringsevne

Gruppen har identifisert flere potensielle risikoer assosiert med redusert gjennomføringsevne av oppgaven. Utenliggende årsaker vurderes som de mest sannsynlige. En risiko som utmerker seg her er faren for å ikke ha nødvendige komponenter eller verktøy til å kunne løse oppgaven i tide. For å redusere sannsynligheten for dette vil gruppen tilstrebe å oppdrive komponenter i god tid fra lokale leverandører, eller leverandører med korte leveringstider.

Innenliggende omstendigheter som interne konflikter av en slik karakter at det hemmer gruppens evne til å gjennomføre oppgaven kan forekomme, men vurderes som lite sannsynlig, blant annet på grunn av gruppemedlemmenes tidligere erfaring som et team. Gruppen vil benytte seg av dialog for å redusere sannsynligheten ytterligere.

Risiko som medfører personell- eller materiellskader

Som det fremkommer av kapittel 1.3.1 er den overordnede visjonen med dette produktet å i størst mulig grad eliminere behovet for menneskelig interaksjon med systemet. På bakgrunn av dette vil ikke oppgaven studere den sikkerhetsmessige problematikken tilknyttet menneskelig interaksjon med det endelige produktet.

Under tilvirkningen av produktet vil det derimot være nødvendig å etterkomme oppdukkende sikkerhetsmessige hensyn. Dette i den hensikt å redusere risikoen for skade på personell eller utstyr. Gruppen har identifisert slag- og klemfarer tilknyttet bruken av robotarmen, som i sin tur kan skade personell og materiell. For å redusere sannsynligheten for slike skader vil nødstoppbryteren på robotarmen alltid være bemannet under bruk.

Gruppen har ikke identifisert flere risikoelementer med høy grad av sannsynlighet og konsekvens utover dette i arbeidet sitt.

1.6 Ressurser

Gruppen har under prosjektet hatt tilgang til en robotlab på Gløshaugen i Trondheim. Her har gruppen disponert en robotarm med tilhørende griper under hele prosjektperioden.

Gruppen har disponert datamaskiner tilhørende fakultetene i oppgaveløsingen. Gruppen har også benyttet personlige Arduinoer supplert av NTNU under første året av utdanningsløpet.

Tabell 1.1 lister innkjøpsprisen til komponentene gruppen har kjøpt for å løse oppgaven.

IR-filter	249,-
Webkamera x 2	258,-
Komponenter til dronen	200,-

Tabell 1.1: Oversikt over innkjøpt utstyr med tilhørende pris

2 Teoretisk bakgrunn

Dette kapittelet tar for seg teori som vil være relevant gjennom hele oppgaven. Teorien vil dekke programvare som er relevant for løsing av alle deler av oppgaven.

2.1 Linux - Ubuntu

Ubuntu er et Linux-operativsystem som bygger på Debian. Operativsystemet har åpen kildekode (eng. *open-source*) og utvikles av Canonical sammen med frivillige utviklere fra hele verden [1]. Det utgis nye versjoner av Ubuntu hver sjette måned, og hvert andre år blir det utgitt en versjon med lengre tids støtte (eng. *long term support*).

Det som skiller brukeropplevelsen i Linux-baserte operativsystem og programmer fra andre operativsystemer, er at de oftere er laget for å operere i et terminalvindu. Altså at man kaller programmer med tekstparametere i stedet for å manipulere grafiske grensesnitt.

En av de største fordelene ved Ubuntu/Linux er det debian-baserte pakkesystemet. Pakkekontrollprogrammet som følger med Ubuntu er *Advanced Package Tool* (apt). Med dette programmet kan du enkelt installere nye programmpakker, og *apt* vil automatisk installere programvare som pakken du vil installere trenger for å fungere (eng. *dependencies*). På denne måten kan man gjennom et terminalvindu enkelt hente pakker man vil bruke, uten at man trenger å laste ned alle avhengigheter og kompilere binærfilene selv. Dette forenkler jobben som programvareutvikler om prosjektet man jobber med er avhengig av annen programvare. *Apt* kan også brukes til automatisk pakkeoppdatering.

2.2 Robot Operating System (ROS)

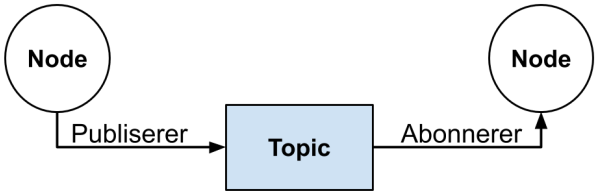
Robot Operating System (ROS) er et fleksibelt rammeverk for programmering av roboter. ROS er ikke et tradisjonelt operativsystem, men en ramme for sammensetting av systemmoduler utviklet for roboter. ROS er hovedsakelig utviklet for Ubuntu, men med eksperimentell støtte i mange andre operativsystemer [2]. Sammen med utvidelsesprosjektet ROS-Industrial, som inneholder utvidelsespakker for populære industriroboter [3], har man tilgang til ferdiglagde programvarebiblioteker for robotstyring. ROS-Industrial er støttet av et internasjonalt konsortium hvor mange store aktører innen robotikk og automasjon er medlemmer:

Figur 2.1: Medlemmer av ROS-Industrial konsortiumet

ROS ble designet for å være så desentralisert og modulært som mulig [4]. Derfor egner det seg til prosjekter som består av flere undersystemer som skal sys sammen. ROS distribueres i form av pakker som inneholder forskjellig funksjonalitet. Dermed kan man laste ned ferdiglagde pakker som inneholder funksjonalitet man ønsker og bruke, og sy disse sammen med annen egenlagd funksjon.

ROS' kjernefunksjonalitet er publisert under den åpne kildekode-lisensen BSD [4], og mange av pakkene til ROS er publisert under denne samme lisensen. Det betyr at man vil ha tilgang til å se kildekoden som ligger bak pakkene man laster ned, og kan modifisere disse pakkene ved behov.

Et ROS-system består av noder som kommuniserer med hverandre ved hjelp av det som i ROS kalles *topics* og *services*. All kommunikasjonen i et ROS-system går via en *master*, men håndteres i bakgrunnen slik at man som utvikler av systemet ikke trenger å tenke på hvordan kommunikasjonen skjer. I figur 2.2 er en enkel illustrasjon av kommunikasjonen mellom to noder, hvor noden til høyre abonnerer (eng. *subscriber*) på den andres *topic*.



Figur 2.2: Et enkelt eksempel på kommunikasjon mellom noder

2.2.1 Noder

En enhet i et ROS-system kalles en node. Hver enkelt node opererer helt uavhengig av hverandre, men kan kommunisere mellom seg ved hjelp de ovennevnte *topics* og *services*. Nodene skrives som separate filer, og kan uten problem kodes i forskjellige språk i samme system [5]. Én av styrkene med ROS er at nodene lett kan skiftes ut. Dette er en fordel ved at for eksempel en kan simulere en node som ikke er ferdigskrevet enda. Da kan man gjøre ferdig alle andre elementer i systemet som avhenger av den ikke ferdig utviklede noden, uten å måtte vente på at den blir ferdig utviklet. Man kan også manuelt styre de signalene man ønsker, for å teste at systemet rundt er robust nok til å håndtere alle verdier som skal kunne komme fra en node. Denne funksjonen kan også brukes for å feilsøke systemet.

2.2.2 Topics

Noder kommuniserer med hverandre via kommunikasjonskanaler som heter Topics. En topic har en definert meldingstype som er kjent for alle sendere og mottakere. All kommunikasjon over den valgte topicen kan kun skje i det definerte meldingsformatet. Kommunikasjonskanalene er veldig fleksible. Nodene kan i teorien lytte og publisere til et ubegrenset antall noder og topics. I praksis begrenses størrelsen på dette nettverket av minne, klokkehastigheter og prosessorkraft. Ved at all kommunikasjonen styres av en *master*-node slipper man som utvikler å spesifisere hvor informasjon skal sendes i nettverket, da det bare er å sette opp nodene til å publisere eller abonnere på de kanalene man ønsker [6].

2.2.3 Messages

Messages benyttes for å beskrive og standardisere dataverdier som sendes fra ROS-noder på topics. ROS har standardiserte messages for et flertall ulike datatyper, som eksempelvis *boolske-* eller *float*-verdier.

Utover å allerede kunne tilby en rekke standardiserte message-typer, tilbyr også ROS muligheten for generering av egne, spesiallagde message-typer.

Det er som tidligere nevnt et krav at det benyttes en definert message-type på en topic, slik at det kan genereres kildekode som muliggjør sending eller mottak av en verdi på en enhet [7].

2.2.4 Programmeringsspråk

ROS-rammeverket er ferdig implementert i programmeringsspråkene Python, C++ og Lisp, og skal være enkel å implementere i andre moderne programmeringsspråk [8]. Python er et høyere nivås programmeringsspråk enn C++, og egner seg godt til programmer som skal kjøres fra kraftigere datamaskiner. Python egner seg også bedre for nybegynnere. C++ er derimot nærmere maskinkode, og egner seg bedre på mindre avanserte prosessorer som mikrokontrollere, eller i tilfeller som stiller høye krav til hastighet og responstid.

2.3 Dataassister konstruksjon (DAK)

Dataassistert konstruksjon (*eng. Computer-Aided Design, CAD*) er konstruksjon og teknisk tegning som utføres ved hjelp av datamaskinbaserte programvarer og redskaper. DAK-tegninger benyttes for å øke produktiviteten for designeren, bedre designkvaliteten, bedre kommunikasjonen ved dokumentasjon, og skape en database for framstilling [9].

2.3.1 Fusion 360

Fusion 360 er et DAK-program laget for å bistå i alle ledd av utformingen av komponenter. Fusion 360 er et gratisprogram for studenter, hobbybrukere og små bedrifter [10]. Programmet er skybasert, noe som tilrettelegger for utviklingsforhold der flere aktører er med i designprosessen.

Fusion 360 lar brukeren tegne 3-dimensjonale tegninger, og tilbyr eksportering av disse i en rekke filformater. Programmet kan fremstille arbeidstegninger for manuell maskinering, eller modeller til bruk i 3D-printere, freser og dreiemaskiner.

3 Løsningskonsept

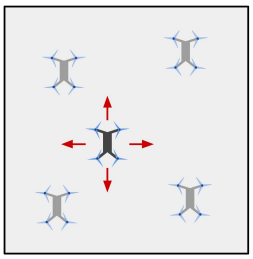
Hovedutfordringer

Fra problemstillingen har gruppen utledet de tre følgende utfordringene:

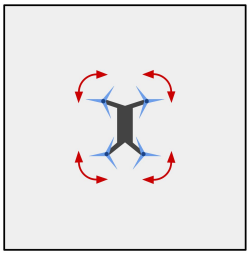
- Identifisere dronens plassering etter landing
- Identifisere dronens orientering etter landing
- Fatte dronen og forflytte den til batteribyttemaskinen

Ansvaret for å gjennomføre disse utfordringene vil bli fordelt på forskjellige systemer, som sammen løser oppgaven i helhet.

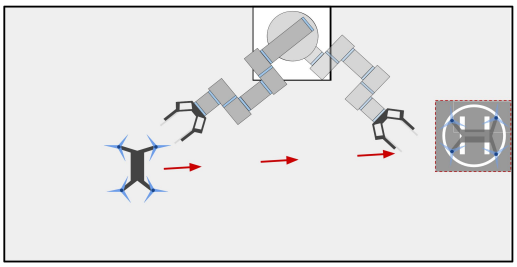
Et system skal kunne kartlegge dronens plassering etter landing. Dronen vil måtte utformes på en måte som gjør at den vil tåle påkjenningene assosiert med å bli flyttet av robotarmen, så vel som å utgjøre det systemet som redegjør for dens retning etter landing. Robotarmen skal implementeres, og det er derfor naturlig at denne utgjør den siste delen av systemet ansvarlig for å flytte dronen fra landingsposisjonen til batteribyttemaskinen.



Figur 3.1: Plassering



Figur 3.2: Orientering



Figur 3.3: Forflytting

Sammenføying av løsning

For å lage et automatisk system som kan løse hovedutfordringene er det behov for et verktøy som kan knyte sammen de forskjellige systemene. Basert på oppdragsgivers ønske om at systemet utvikles så generelt som mulig, falt valget på å løse oppgaven ved hjelp av Robot Operating System (ROS).

4 Forflytting

Dette kapittelet omhandler den delen av prosjektet knyttet til utfordringen å fatte og forflytte dronen. Det vil gå nærmere inn på robotarmen som ble brukt, hvordan den fungerer samt hvordan modifikasjoner som gjorde dronen gripbar. Kapittelet vil utdype om relevant teori, utstyr og framgansgmåte i metode, presentere resultatet, drøfte hvorvidt resultat er nådd og utfordringer pluss et delkapittel om alternative løsningsmetoder for denne modulen.

I problemstillingen er det gitt at gruppen skal implementere en servicearm i løsning av oppgaven. Dette er grunnet at servicearmen kan brukes til flere oppgaver enn forflytting i fremtidig arbeid på prosjektet. Gruppen har derfor valgt å løse denne oppgaven med en robotarm, og tilhørende utstyr som gjør den egnet til forflyttingen av dronen.

4.1 Teori

4.1.1 Robotarmteori

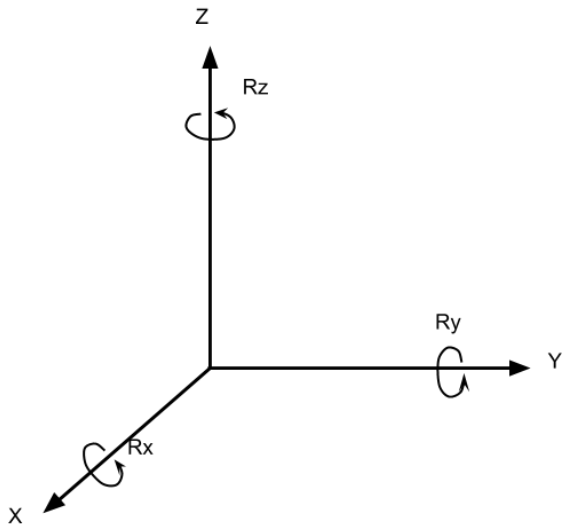
Industriroboter kommer i mange ulike varianter og størrelser. Det som de fleste industrirobotene har til felles er at de består av: manipulatorarm, kraftforsyning, redskap, styresystem og sensorer [11]. En vanlig form for industrirobot er antropomorfiske robotarmer. Denne typen manipulatorarm ligner på en menneskearm i leddsammensetningen, og har derav fått navnet antropomorf, som betyr menneskelignende [11]. Antropomorfiske robotarmer består utelukkende av roterende ledd. Det motsatte av et roterende ledd er et prismatisk ledd, og fungerer et slags stempel. Denne typen ledd er også vanlig i industriroboter, og kan kombineres med roterende ledd for å lage andre type robotarmer, med andre egenskaper.

En robotarm består som oftest av mange ulike ledd som til sammen utgjør hele armen, pluss et verktøy på enden. Verktøyet er det redskapet som er plassert ytterst på robotarmen, og har til hensikt å gi armen dens fullverdige funksjon. Antropomorfe robotarmer består gjerne av 3 store ledd, og et håndledd, som sørger for korrekt plassering av verktøyet. De 3 store leddene sørger for posisjoneringen av verktøyet. Håndleddet er som regel delt opp i 3 roterende ledd, og har til hensikt å orientere verktøyet. Sammen vil armen sørge for en ønsket plassering og rotasjon på verktøyet [12].

På grunn av den fleksible plattformen armen tilbyr kan verktøy i stor grad tilpasses den gitte arbeidsoppgaven. Det finnes standardverktøy, som gripere, men de fleste industriroboter

er også åpne for å tillegge spesiallagde verktøy eller instrumenter. Robotarmer kan lages for å tilrettelegge rask utskifting av verktøyet, og det vil gi robotarmen mulighet til å utføre et enormt antall ulike oppgaver [13].

Objekter i det 3-dimensjonale rommet kan beveges på 6 ulike måter. Det kan beveges på 3 måter translatorisk, altså langs koordinatrammene, og rotere om de samme koordinatrammene slik figur 4.1 viser. Vi kaller det da at objektet har seks frihetsgrader [11]. For en antropomorf robot vil den som oftest også operere innenfor alle de 6 frihetsgradene. Arbeidsområdet til en robotarm defineres som det totale området armen kan plassere verktøyet, ved hjelp av alle mulige kombinasjoner av leddutslag [12].



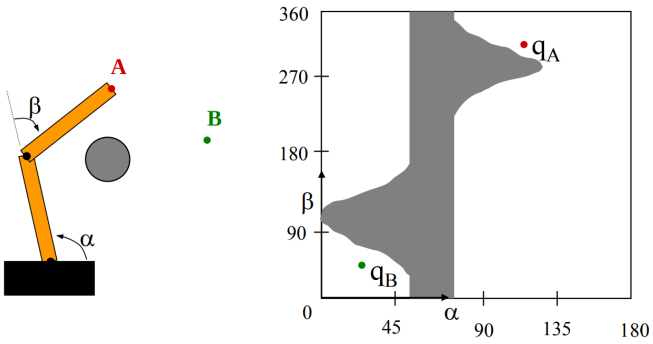
Figur 4.1: De seks frihetsgradene i et koordinatsystem. 3 translasjoner, X, Y, Z og 3 rotasjoner, Rx, Ry, Rz

Det er vanlig for industriroboter å inkludere en berøringsskjerm (eng. *teach pendant*) for styring av roboten via et grafisk grensesnitt. Her kan man vanligvis også programmere komplekse rutiner for operasjon av roboten. Et alternativ til lokal programmering med berøringskjermen er å ha et kommunikasjonsgrensesnitt for ekstern styring av roboten [12]. En fordel som et kommunikasjonsgrensesnitt muliggjør, er at man kan knytte roboten til et sentralt system. Man kan da knytte mange roboter sammen i et avansert system hvor de forskjellige robotene samhandler.

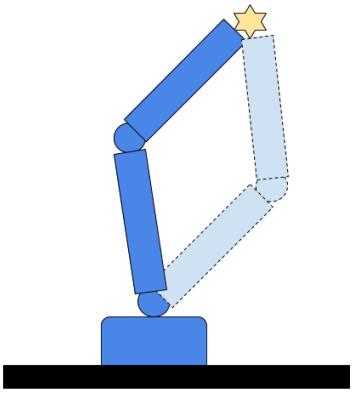
4.1.2 Baneplanlegging

Baneplanlegging handler om å få robotarmen til å bevege seg fra positur A til positur B uten å kollidere med objekter i arbeidsområdet. Dette problemet er blant de vanskeligste problemene å løse innen informatikk [12]. Kompleksiteten, og vanskelighetsgraden, øker med antallet frihetsgrader roboten har.

En komplett spesifikasjon av lokasjonen av hvert enkelt punkt på roboten blir kalt konfigurasjonen. Settet av alle mulige konfigurasjoner roboten kan ha heter konfigurasjonsrommet [12]. Figur 4.2 består av et bilde med en enkel robot med to roterende ledd og et hinder, og et bilde med det tilhørende konfigurasjonsrommet. Vinkelutslag på de roterende leddene danner konfigurasjonsrommets to akser, α og β. Det grå området i figuren representerer konfigurasjoner der roboten vil kollidere med det grå sirkulære hinderet.



Figur 4.2: Enkel robot med et hinder i arbeidsområdet, til venstre, og tilhørende konfigurasjonsrom, til høyre [14]



Figur 4.3: Enkel robotarm der to ulike konfigurasjoner oppfyller arbeidskravet

En arbeidsoppgave krever å flytte verktøyet til den nødvendige posisjonen med den nødvendige orienteringen. I figur 4.2 ville dette ha vært å flytte roboten fra punkt A, til punkt B. Dette flyttet må skje innenfor konfigurasjonsrommet til roboten, noe som betyr at det må kunne trekkes en sammenhengende linje mellom punktene i konfigurasjonsrommet, q_A og q_B. I figur 4.2 er ikke dette mulig, på grunn av det grå området er heltrukket parallelt med β-aksen. Det skyldes at det første leddet har en lengde som gjør det umulig å rotere forbi hinderet. Dermed ser vi at roboten ikke ville ha kunnet utført oppgaven sin, fordi det ikke finnes en bane mellom q_A og q_B i konfigurasjonsrommet. Hindringer i arbeidsområdet vil innskrenke bevegelsesfriheten, og som regel gjøre konfigurasjonsrommet mindre. Ved å øke antall ledd på roboten, øker antall konfigurasjoner og det fremkommer flere muligheter for å komme seg rundt et hinder i arbeidsrommet.

Innfor det gjenstående arbeidsområdet er det ofte flere konfigurasjoner en robot kan ha, og oppfylle arbeidskravet, slik figur 4.3 viser. Det er også flere baner en robot kan bevege seg i, og ende i en godkjent konfigurasjon. Det finnes flere ulike tilnærminger til å planlegge banen på som vil by på ulike utfordringer. Noen av disse tilnærmingene er *planlegging ved bruk av arbeidsområdet og potensielle felt*, *bruk av tilfeldige bevegelser for å unngå lokale minima* og *probabilistisk veikart metoden* [12]. Ettersom det er så komplisert å beregne de ulike banene er det laget flere programmer som kan generere banen for roboten. Ulike programmer bruker ofte ulike tilnærminger.

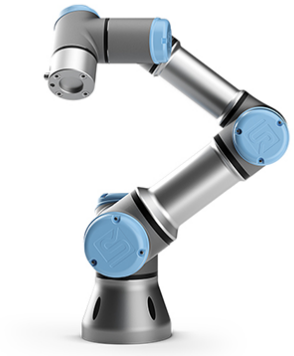
4.2 Metode

I dette prosjektet ble det benyttet en UR3-robotarm for å flytte selve dronen mellom landingsplattformen og batteribyttemaskinen. NTNU hadde to ulike robotarmer som gruppen hadde til rådighet å nytte i prosjektet. Valget av robotarmer stod mellom UR3 og Nachi MZ07. Nachi-armen veide omtrent 30 kg, stod fastmontert, og har en nyttelast på 7 kg [15]. UR3-robotarmen stod fastmontert på et bord og var lett flyttbar, veide 15 kg, og hadde en nyttelast på 3 kg [16]. Ut ifra informasjonen om nyttelast var begge robotarmene fullt kapable til å utføre flyttejobben av en 250 g drone. Andre prosjekter har brukt UR3 med ROS, og dette arbeidet var åpen kildekode. Basert på flyttbarhet og tilgang på tidligere arbeid, falt valget på å bruke UR3 videre i prosjektet.

4.2.1 Utstyr

UR3

Universal Robots (UR) har laget roboter som skal være enkle å sette opp, enkle å bruke og universale [17]. UR har en serie lignende robotarmer, der hovedforskjellen mellom armene er størrelsen. UR3, avbildet i figur 4.4, er en av de minste av disse robotarmene, og tallet 3 indikerer at armen har en maks nyttelast på 3 kg. Den antropomorfiske robotarmen er bygget opp av 6 roterende ledd, der endeleddet har uendelig rotasjon. Alle andre ledd kan vris $\pm 360°$ [16]. Base-, skulder og albue-ledd har en topphastighet på $\pm 180°$/sek [16]. De resterende leddene som ut-



Figur 4.4: Universal Robots UR3-robotarm [17]

gjør håndleddet har en topphastighet på ±360°/sek [16]. UR3 har et maksimumsrekkevidde på 500 mm, med en repetisjonsnøyaktighet på ±0,1 mm [16] innenfor rekkevidden. Ettersom dronen må plasseres med en nøyaktighet på ±6,0 mm vil UR3 være godt egnet til jobben.

I tillegg til selve armen følger det med en kontrollboks som fungerer som robotarmens kraftforsyningsenhet, og et styresystem i form av en trykkskjerm. Boksen forsynes med 240 VAC og forsyner videre armen med 24 VDC, verktøyet og trykkskjermen [16]. Boksen har en Ethernet-tilkobling som kan brukes til ekstern tilgang og kontroll [18].

Griper - Robotiq 2F-85

Verktøyet som robotarmen bruker er en griper fra selskapet Robotiq. Griperen, se figur 4.5, har navnet Robotiq 2F-85, og indikerer at gripemekanismen består av to fingre med et spenn fra 0-85 mm. Gripestyrken kan stilles mellom 20-235 N, alt etter skjørbarhet og tyngde på objektet som skal gripes. Griperen har en egenvekt på 900 g og kommer klar som *plug and play* sammen med blant annet roboter fra Universal Robots. Griperen har en oppløsning på 0,4 mm og hastigheten kan variere fra 20 til 150 mm/sek [20].

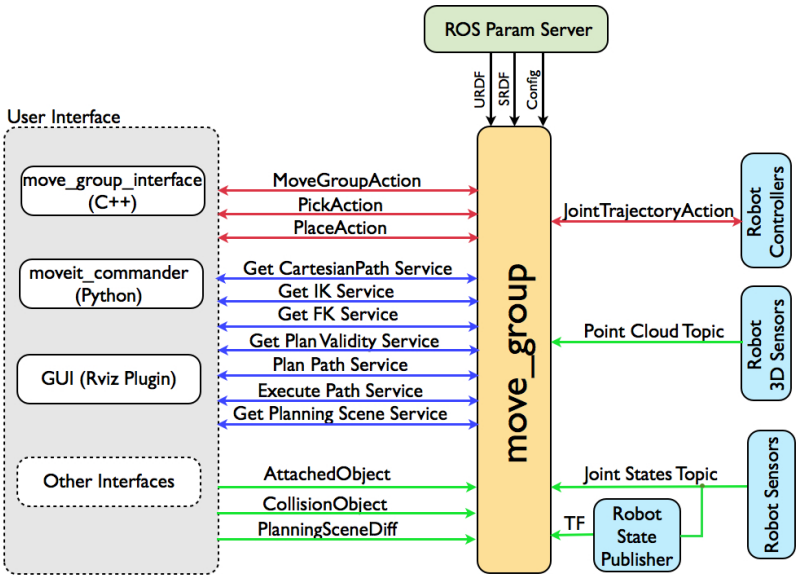
Figur 4.5: Robotiq 2f-85 griper [19]

MoveIt

MoveIt er den største åpen-plattform programvaren for manipulering av roboter [21]. Programvaren bygger på det siste innen kinematikk, baneplanlegging, 3D-persepsjon, manipulasjon, kontroll og navigasjon. Programvaren er brukt i over 100 roboter som operer i alt fra det ytre rom til langt under havoverflaten [21].

MoveIt er laget for å fungere i ROS, og bruker derfor noder og topics. Figur 4.6 viser systemarkitekturen til MoveIt. *move_group*-noden fungerer som en integrator som drar alle individuelle komponenter sammen og skaper et sett med aksjoner som brukeren kan utføre [22]. MoveIt fungerer slik at brukeren gjennom brukergrensesnittet vil be roboten om å flytte seg til en posisjon, og verktøyet med den ønskede orienteringen. *move_group*-noden vil deretter generere en bane som oppfyller brukerens ønske. I tillegg til å generere banen lager også noden en plan for hastigheten som hvert ledd skal bevege seg i for å være effektiv, men uten å overstige maksgrensene for hastighet og akselerasjon. MoveIt benytter flere baneplanleggere, men den vanligste er OMPL (Open Motion Planning Library) [22]. OMPL er en pakke som benytter flere måter å regne baner på, blant annet *probabilistisk veikart metoden* som ble nevnt

i kapittel 4.1.2. Ved hjelp av OMPL kan MoveIt utføre en kollisjonssjekk i de planlagte rutene, for å forhindre at den kolliderer med seg selv. Det er også mulig å legge inn flater, områder og figurer i de digitale planleggingsomgivelsene på en slik måte at de representerer virkeligheten. Dette gjør at robotarmen er i stand til å unngå objekter i sine fysiske omgivelser.



Figur 4.6: Oversikt over systemarkitekturen til MoveIt [23]

4.2.2 Maskinvarekonfigurering

Kommunikasjon med datamaskin

For å styre UR3-armen fra datamaskinen med ROS ble kontrollboksen til UR3 koblet på et Ethernet-nettverk med datamaskinen som hadde ROS installert. Dette nettverket bør ha lav forsinkelse ettersom kontrollboksen forventer en oppdateringsrate på instruksene på 8 ms [24, p. 10]. Foruten å sørge for at kontrollboksen og datamaskinen var koblet på samme LAN var det ingen videre nettverkskonfigurasjon som trengtes.

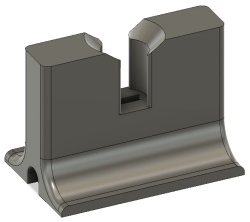
Installasjon av griper

Griperen *Robotiq 2F-85* kommuniserer over ModBus RTU [20, p. 56]. I tillegg til griperen medfulgte det en overgang direkte til USB. Denne koblingen gikk direkte til datamaskinen som opererte med ROS. ROS-pakken *robotiq* som er utviklet av griperens produsent, Robotiq,

gir oss en ROS-node vi enkelt kan samhandle med ved å publisere på en gitt *topic*. Dette gjør det enkelt å stille inn kraft, hastighet eller posisjon med en 8-bits oppløsning. I tillegg publiserer driveren informasjon fra griperen, slik at en kan bruke denne informasjonen som en tilbakekobling i ROS for å verifisere at en bevegelse er fullført.

4.2.3 Gripefinne

Utfordringen knyttet til det å forflytte dronen handler også om samspillet mellom den fysiske dronen og griperen. For å imøtekomme kravet om å flytte dronen uten at den tar skade, ble det designet en brakett, se figur 4.7, som oppfyller dette kravet.



Figur 4.7: Gripefinne

I tillegg til gripefinnens primærfunksjon ble det besluttet å plassere IR-dioden i sentrum av finnen. Diodens funksjon er knyttet til detektering av dronen. Arbeidet med å plukke opp dronen forenkles ved at dronens posisjon er direkte tilknyttet gripepunktet på dronen.

Gripefinnen ble designet i *Fusion 360* og deretter 3D-printet i ABS (Akrylnitril-butadien-styren), en robust plastikk med forholdsvis høyt smeltepunkt[25]. Det er designet egne kanaler for de nødvendige lederene til dioden. Området designet for å gripes har en tykkelse på 6,0 mm, og måler 13 mm x 23 mm i høyde og bredde. Dette har vist seg å være tilstrekkelig areal for at *Robotiq 2F-85*-griperen får et godt grep, og gjør at IR-dioden ikke skades ved løft. Gripefinnen har en totale høyde på 19 mm, når man inkluderer festeflaten og den strukturelle krumningen opp mot gripeflaten. Det arealet som utgjør festeflaten til dronen måler 16 mm x 25 mm.

Åpningene på siden av dioden eksisterer fordi det ikke er mulig for en 3D-printer å printe så tynne vegger. Disse åpningene vil også bedre kjølingen av dioden, da denne i henhold til databladet [26] på et lite område omsetter effekten:

$$1{,}5\,\mathrm{V} \cdot 100\,\mathrm{mA} = 0{,}15\,\mathrm{W} \tag{4.1}$$

4.2.4 Programvarekonfigurering

For å styre UR3-armen i ROS ble ROS-pakken *universal_robot* benyttet. Pakken inneholdt alt som trengtes for å styre armen. Deriblant modeller og beskrivelser av leddene for MoveIt, driver for nettverkskommunikasjon, og kontroller for styring av bevegelser via MoveIt. Kontrolleren i denne pakken var utdatert, og laget for eldre systemversjoner av roboten [27]. Derfor

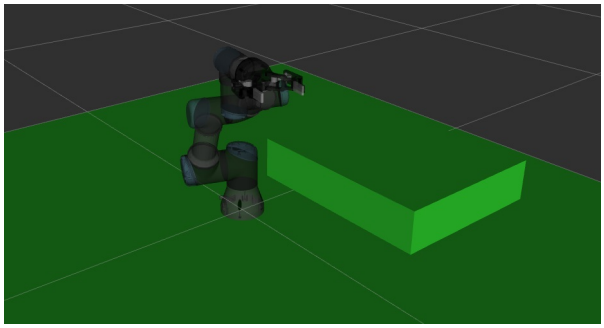
brukte vi også ROS-pakken *ur_modern_driver*, som er en videreutvikling av kontrolleren som støtter nyere fastvare, og har mer funksjonalitet.

For å bruke MoveIt-pakken trenger pakken å vite hvordan roboten den skal planlegge bevegelser for ser ut. Dette gjøres ved hjelp av en stardard XML formatering for robotbeskrivelsen som håndteres av *urdf*-pakken (Unified Robot Description Format) i ROS [28].

Ferdige URDF-filer for både *UR3* og *Robotiq 2F-85* finnes som deler av ferdige pakker laget for ROS. Vi trenger egentlig kun modellen til *UR3* for å kunne planlegge bevegelser og gjennomføre de med MoveIt, men da vil ikke MoveIt kunne ta hensyn til at armen har en griper på enden. Man risikerer da at MoveIt planlegger bevegelser hvor griperen kolliderer med enten armen selv, eller omgivelsene rundt. For å unngå dette må vi sy sammen de to modellene slik at MoveIt tar hensyn til griperen når bevegelser planlegges. Denne sammensyingen gjøres ved hjelp av en ROS-pakke som heter *xacro* (XML Macro) som lar oss kalle på de to separate URDF-filene og sy de sammen med spesifiserte parametre [29].

Til sist gjenstår det å sørge for at MoveIt ikke planlegger bevegelser hvor gjenstanden som har blitt plukket opp, altså dronen, kolliderer med armen eller omgivelsene. Dette har MoveIt støtte for å gjøre mens programmet kjører, slik at man kan kalle en funksjon som kobler en boks av definert størrelse på det leddet som er blitt definert som robotens verktøy i modellfilene som er lastet av MoveIt.

Man kan også legge inn modeller for mer detaljert kollisjonsplanlegging, men en enkel boks er bra nok for dronen vi bruker. Dette er simulert i figur 4.8, der kollisjonsobjektene er de grønne boksene rundt UR3. Ved å bruke denne funksjonaliteten kunne man planlegge raske og effektive bevegelser før dronen ble plukket opp, for så å planlegge roligere baner med større marginer etter at dronen hadde blitt hentet.



Figur 4.8: UR3 simulert med kollisjonsobjekter

For å forsikre oss at MoveIt fant gode løsninger på bevegelsene justerte vi noen parametre for å gi litt større feilmarginer og slingringsmonn i utregningene:

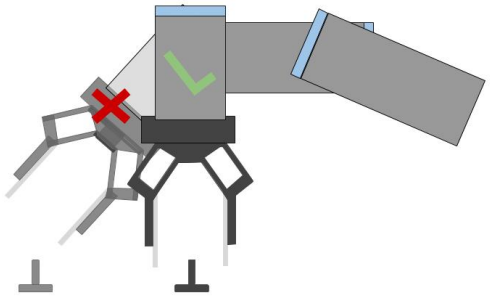
```
MoveGroup.allow_replanning(True)
MoveGroup.set_goal_position_tolerance(0.002)
MoveGroup.set_goal_orientation_tolerance(0.02)
MoveGroup.set_planning_time(3)
MoveGroup.set_num_planning_attempts(50)
MoveGroup.set_max_acceleration_scaling_factor(.1)
MoveGroup.set_max_velocity_scaling_factor(.1)
```

Her stilte vi inn at MoveIt får lov til å forsøke å planlegge ruten flere ganger, og gir den en ekstra toleranse på målposituren. Å gi MoveIt en maksimal planleggingsramme på 50 forsøk à 3 sekunder, i tillegg til den ovennevnte toleransen i positur, ga oss bedre resultater i at MoveIt oftere fant fram til en gyldig løsning på bevegelser. Uten denne ekstra marginen i positur, eller med mindre planleggingsramme, opplevde vi at MoveIt i blant kjørte gjennom baneutregningene uten å finne noen gyldig løsning. I programkoden må dette da enten tas høyde for enten ved å avbryte hele programmet, eller å sette en gitt tid man er villig til å la MoveIt forsøke å finne bevegelsesbaner. Denne begrensningen har man fordi man ikke er garantert at posituren som blir sendt til MoveIt som mål er innenfor robotens konfigurasjonsrom. For noen positurer finnes det ingen gyldige løsninger, mens det for andre ikke finnes mulige bevegelser som tar dem dit uten kollisjon. Man må dermed gjøre en avveing mellom hvor lenge man er villig til å la MoveIt forsøke, og muligheten å ikke finne en eksisterende gyldig løsning i løpet av den rammen som er gitt.

4.3 Resultat

Gjennom å koble robotarmen og griperen til ROS på datamaskinen ender vi med en fullstendig løsning hvor dronen kan gripes innenfor store deler av armens arbeidsområde. Arbeidsområdet blir noe begrenset i forhold til UR3 sitt potensielle arbeidsområde, fordi angrepsvinkelen verktøyet skal ha, er normalt på planet det skal gripe i.

I tillegg tar vi utgangspunkt i at verktøyet skal gripe fra en orientering pekende rett ned i planet under. Dette begrenser arbeidsområdet noe videre. Resultatet er at vi kan gripe dronen i en radius 20-45 cm fra armens base på bordet den er montert.

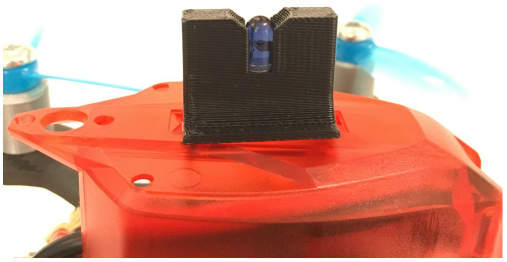


Figur 4.9: Demonstrasjon av angrepsvinkel

Planleggingen av bevegelsene gjøres av MoveIt. ROS-pakken er et robust verktøy, som når riktig innstilt, planlegger effektive bevegelser for robotarmen. Ved å bruke ROS-pakkene for drivere av UR3-armen og Robotiq-griperen, kan vi styre roboten fra datamaskinen. Den kan plukke opp gjenstander og sette de ned hvor som helst innenfor rekkevidden sin, forutsatt at den får oppgitt koordinatene den skal flytte seg til.

4.3.1 Gripefinnen

Gripefinnen endte med å være utformet for å være lett gripbar for *Robotiq 2F-85*-griperen. Det er tilrettelagt for at IR-dioden plasseres i sentrum av gripefinnen, se figur 4.10. Den ble 3D-printet i en robust plastikk, som muliggjør flytt uten at dronen eller IR-dioden tar skade av dette. Finnen er også selvrettende om griperen ikke skulle være helt parallell med finnen.



Figur 4.10: Gripefinnen med IR-diode på dronen

4.4 Drøfting

4.4.1 Robotarm og bevegelse

Noen oppdagelser og erfaringer ble gjort underveis. I teorien skal roboten ha et arbeidsområde som tilsvarer 500mm fra robotens kjerne i basen. Dette betyr i praksis at endeleddet kan nå så langt i en utstrakt posisjon. Ettersom robotarmen skal bevege seg langs z-aksen for å plukke opp dronen blir det virkelige arbeidsrommet innskrenket. For at roboten skal opprettholde bevegelsen i z-aksen som kreves, klarer den derfor ikke å gjøre dette i en avstand på 500mm fra basen. Landingsområdet viste seg derfor å være mindre i praksis enn først antatt.

Det er mulig å tenke at Nachi-robotarmen ville ha vært et bedre valg. En større arm ville ha utvidet det potensielle arbeidsområdet, men problematikken med innskrenkning av dette området ville fortsatt ha vært like reellt. En større arm ville uansett også ha vært nødt til å bevege seg langs z-aksen for å gripe dronen trygt. Akkurat dette problemet var ikke noe som spilte inn på løsningen av oppgaven. Selv med et mindre praktisk landingsareal vil robotarmen kunne utføre oppgaven den er ment å gjøre. I praksis når dronen blir plassert i landingsarealet er det tatt høyde for at robotarmen kan rekke til dette punktet. Innskrenkelsen kan være et problem som må taes med ved utviklingen av et optimalt system, men for løsning i denne oppgaven var det neglisjerbart.

Et annet punkt en bør tenke på ved utvikling av dette systemet er problematikken knyttet til konfigurasjonsrommet. Om landingsområdet er plassert midt mellom robotarmens arbeidsområde vil armen ha god forutsetning for å beregne banen til og fra gripeposisjonen. Ved å ha størst mulig konfigurasjonsrom er det optimalt for banegenereringsprogrammet å finne en egnet bane. Det er da mindre sjanse for at programmet ikke finner en løsning, og stopper opp. Ved å plassere landingsområdet lenger ut i robotarmens arbeidsområde vil konfigurasjonsrommet bli mindre, og baneplanlegging kan bli dårligere.

Resultatmessig beveger roboten seg mellom punktene som den skal. Den beveger seg fra bildetagningsposisjon til henting av drone, og bort til batteribyttemaskin uten kollisjon. Fra observasjon er ikke alltid selve bevegelsene mellom disse punktene den raskeste. MoveIt genererer ofte baner med veldig komplekse bevegelser sett utenfra. Ulike ledd roterer ofte mer enn 360° for å nå et punkt. Når leddet roterer mer enn 360° har leddet rotert unødvendig mye. Når et ledd roterer 360° er det reelt tilbake til start.

Det er usikkert om det er pakken MoveIt bruker, OMPL, som fører til de uortodokse rutene, eller om det er parametersettingen av MoveIt som har skylden. Det har vært testet med ulike

parametere, men resultatet har enten vært likt, eller ikke ført fram, som beskrevet i kapittel 4.2.4. Samtidig kan det hende at MoveIt egentlig planlegger rutene veldig godt, og den roteringen på mer enn 360° gjøres mer som en forberedelse til videre flytt. Dette er usikkert, men det er verdt å merke seg. Det finnes andre pakker i MoveIt for banegenerering som kanskje ville ha gitt andre og smartere resultater. På grunn av manglende erfaring og tid, ble det ikke mulighet til å teste flere pakker en OMPL. I helhet har ikke dette så mye å bety, ettersom roboten fortsatt gjør det den skal, selv om alltid rutevalget er den raskeste.

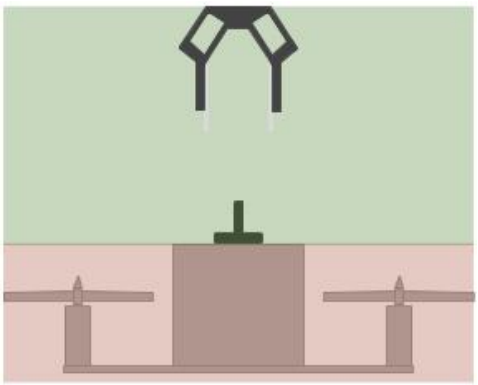
For å utføre det nødvendige flyttet av dronen fra landingsplattformen til batteribyttemaskinen er det ikke nødvendig med en robotarm som operer med 6 frihetsgrader. I realiteten ville denne oppgaven kunne utføres like bra, eller kanskje bedre av en robot som opererte med de 4 nødvendige frihetsgradene, X, Y, Z og Rz. Dronen lander på et horisontalt plan, og skal flyttes til en batteribyttemaskin i det samme horisontale planet. Orienteringen til dronen vil variere, og det vil derfor kreves å rotere dronen rundt z-aksen. Etter at dronen er rotert riktig, kan den løftes uten videre rotasjon til batteribyttemaskinen. En mer spesifisert robot ville hatt lettere for å planlegge gode baner, og spare mye datakapasitet. På en annen side er en robot som opererer i de 6 frihetsgradene bedre egnet til videreutvikling av konseptet, da denne vil muliggjøre flere komplekse operasjoner eksempelvis servicefunksjoner vil kreve.

4.4.2 Gripefinne

Intensjonen med denne løsningen har vært å muliggjøre fatting av dronen uten at den tar skade av det samt tilrettelegge for posisjonsobservering.

Størrelsen på gripefinnen har blitt dimensjonert på en god måte for å fungere sammen med *Robotiq 2F 85*-griperen, også når denne finnen huser IR-dioden. Det har vært flere fordeler ved å benytte seg av løsningen gruppen valgte.

For det første sikrer løsningen vår at dronen ikke blir skadet av griperen. Dette fordi griperen nå kun leter i et område (altså i en høyde over resten av dronen) hvor det eneste den kan berøre er gripefinnen, som igjen er laget for å tåle den slags type berøring. Dersom griperen skulle ha fattet selve dronen kunne dette i tilfeller der posisjonen var feilberegnet ført til at griperen fattet deler av dronen som ikke er beregnet for dette.



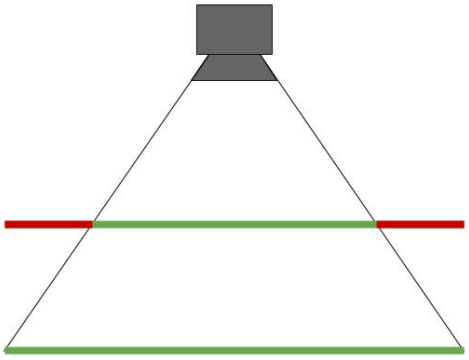
Figur 4.11: Trygt området for å behandle dronen

En annen fordel med denne løsningen er den selvrettende effekten man får når gripefinnen er montert fast på dronen. Dersom dronen skulle stå noe vridd i forhold til griperen, vil dronen dreies parallelt med griperen i det denne lukkes. Dette gir en økt toleranse for feil, eksempelvis i tilfeller der magnetometeret har en unøyaktig avlesning. Tester vi har utført viser at dronens retning kan avvike med $\pm 30°$, og fortsatt bli plukket opp på en tilfredsstillende måte.

Ved at observasjonspunktet er montert i senter av gripepunktet spares også algoritmen for å beregne gripeposisjonen for mellomregninger. Dersom observasjonspunktet ikke var montert samme sted som gripepunktet ville det vært nødvendig med funksjoner som tok høyde for dronens retning kombinert med den observerte posisjonen. Det er ikke nødvendig å foreta denne omregningen med gjeldende løsning, da observasjonspunktet og gripepunktet alltid vil sammenfalle.

Dog medfører denne gripefinneløsningen også noen utfordringer. Eksempelvis tillegger den noe mer vekt på dronen, og den tilfører noe aerodynamisk motstand. Dog kan det argumenteres for at gripefinnen på den ferdig utviklede dronen kan inneha flere funksjoner utover det å være en gripefinne; eksempelvis kan den huse nødvendige antenner eller andre nødvendige sensorer, og dernest veie opp for den økte vekten og luftmotstanden. IR-Dioden i seg selv utgjør et forholdsvis lite, nærmes neglisjerbart, fotavtrykk.

En annen ulempe ved å montere observasjonspunktet høyere enn dronen er det noe reduserte observasjonsområdet. Når kameraet er plassert direkte over landingsområdet minsker det praktiske synsfeltet som funksjon av høyden over planet. Dette fører til at det er et mindre område å observere IR-dioden i når den plasseres høyt enn lavt.



Figur 4.12: Demonstrasjon av redusert synsfelt ved høy montering av observasjonspunkt

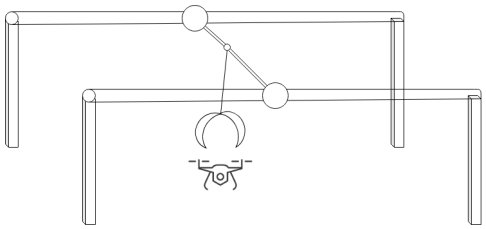
4.5 Alternative løsningsmetoder

4.5.1 Alternativ til robotarm

Alternative løsningsmetoder vil være løsninger som skiller seg veldig fra den valgte robotarmen. Av oppgavens natur fremkommer det at det skal være en servicearm, og det derfor ikke kan brukes noe annet enn en robotarm i prosjektet. Dog finnes det alternativer som kunne ha utført løftet som robotarmen utførte.

Gripeklo i stativ

Et alternativ til fatting og forflytting av dronen kunne være en innretning som har kjører på skinner i x-og y-retning i en stasjonær høyde over landingsplattform og batteribyttemaskin. Innretningen kunne hatt en gripeklo som lar seg heves og senkes i z-retning. Fordelen med en slik innretning ville være at denne hadde hatt færre frihetsgrader å jobbe med, og dermed enklere å programmere. En ulempe med løsning hadde vært at visse deler av landingsområdet ville ha vært dekket av stativet, og kanskje ha gjort landingen vanskeligere for dronen. En annen ulempe med denne løsningen er muligheten for å implementere flere arbeidsoppgaver. Stativet ville nok ha optimalisert oppgaven med å flytte dronen, men å utføre enda flere oppgaver ville kanskje være vanskeligere å implementere.



Figur 4.13: Enkel konsepttegning for gripeklo i stativ

4.5.2 Alternativer til gripefinne

Gripe dronen i skroget

Et alternativ til å gripe dronen i en egen gripefinne, vil være å gripe i selve skroget. En fordel ved dette vil være at man slipper den økte vekten og luftmotstanden nevnt tidligere. På den andre siden vil dette kunne stille noe høyere krav til retningsoppløsningen til griperen. Dersom denne ikke er nøyaktig nok kan en risikere at griperen berører komponenter som ikke er ment å berøres, og i værste fall ødelegge disse. Det er nødvendig å legge til at en slik løsning uansett vil kreve et skrog som har tilfredsstillende strukturell styrke for å kunne bli løftet i.

Skyve dronen til ønsket plassering

En annet alternativ vil kunne være å skyve dronen til ønsket sted, i steden for å løfte den. Man slipper da en innretning på dronen dedikert til forflytting, og det er således ikke nødvendig å gjøre strukturelle endringer på skroget. Denne løsningen vil på den andre siden kreve vesentlige endringer på den maskinen som finnes i dag, slik at det blir mulig å skyve dronen fra landingsstedet til batteribytteroboten.

5 Lokalisering

Dette kapittelet tar for seg den delen av prosjektet som skal identifisere dronens plassering etter landing. Det første delkapittelet inneholder relevant teori om datasyn og lys. Deretter følger metode-kapittelet som tar for seg utstyr, tilhørende programmer, maskinvarekonfigureringen, kalibrering og programmering av modulen. Videre går kapittelet inn på resultat, drøfting og alternative løsningsmetoder knyttet til å finne dronens lokasjon.

Gruppen har valgt å bruke datasyn i løsningen for å finne dronens plassering på landingsplattformen. ROS inneholder måter å implementere datasyn på, som vil gagne prosjektarbeidet. I tillegg vil datasyn være fleksibelt og gi gode muligheter for videre arbeid på dette prosjektet.

5.1 Teori

5.1.1 Datasyn

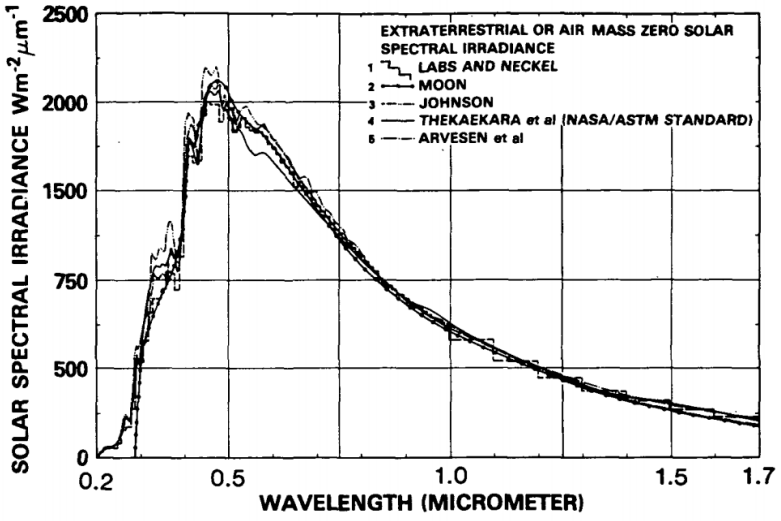
Datasyn er et tverrfaglig fagfelt som har som mål å gi datamaskiner en evne til å tolke visuell data på en måte som ligner det vi mennesker gjør. Fagfeltet påvirkes av en rekke andre fagfelt, som fysikken som ligger bak lysets bølgeforplanting, elektronikken bak måling av lys, og matematikken og informatikken bak digital behandling av bilder.

Et digitalt bilde er en stor matrise med verdier som representerer de forskjellige lysverdiene (piksler), ofte i flere fargekanaler. Det å hente informasjon ut fra bilder innebærer å manipulere disse matrisene. Da digitale bilder ofte består av millionvis av piksler er operasjoner på disse matrisene veldig beregningskrevende, og vil være mer krevende jo større matriser du har.

5.1.2 IR-lys og filter

Infrarød stråling defineres som elektromagnetisk stråling hvor bølgelengden er mellom 700 nanometer og ca. 1 millimeter [30]. Det menneskelige øyet kan se stråling med bølgelengder fra 380 nm til 740 nm, og det meste av IR-strålingen vil derfor være usynlig for mennesket [31]. Veldig mange elektriske apparater og dyr benytter seg av IR-spekteret med ulike formål. Varmekilder med ulik temperatur utstråler ulik IR-stråling. Noen IR-kameraer brukes til å lete etter mennesker ut fra at varmeutstrålingen vil skille seg ut fra omgivelsene. Som

graf 5.1 viser har det meste av dagslyset en bølgelengde på 350 nm - 700 nm, og mengden minker ved høyere bølgelengder.



Figur 5.1: Bølgefordelingen av solens utstråling [32]

Lysfilter lages for å ha en av to funksjoner: evnen til å blokkere eller slippe gjennom bestemt lys. Filter som slipper gjennom lys med større bølgelengder enn knekkfrekvensen kalles lavpassfilter. Filter som kan slippe gjennom lys med bølgelengder i et gitt spekter kalles båndpassfilter. Ved riktig bruk av sliker filter kan man stenger for eller slippe inn det lyset man ønsker. I denne oppgaven benyttes begrepet IR-filter om filter som slipper inn lys i det infrarøde spekteret.

5.2 Metode

I denne modulen ble det brukt et enkelt webkamera for å observere omgivelsene. Grunnen til dette var basert på enkelte gruppemedlemmers erfaring med kameraet fra før, og muligheten for å gjøre fysiske inngrep i kameraet ble vurdert svært viktig. Det var også ønskelig med et enkelt og relativt billig kamera om det skulle bli skadet eller ødelagt underveis i arbeidet. Enkelheten er også en fordel å ha som utgangspunkt for den nødvendige programmeringen.

Utgangspunktet for gjenkjenning av dronens landingsposisjon skulle være en lysemitterende diode plassert i dronens gripeinnretning. På grunn av at landingsområdet er flatt, ble dette vurdert som tilstrekkelig for å kunne identifisere dronens posisjon i et todimensjonalt plan.

5.2.1 Utstyr

Trust Exis Webcam

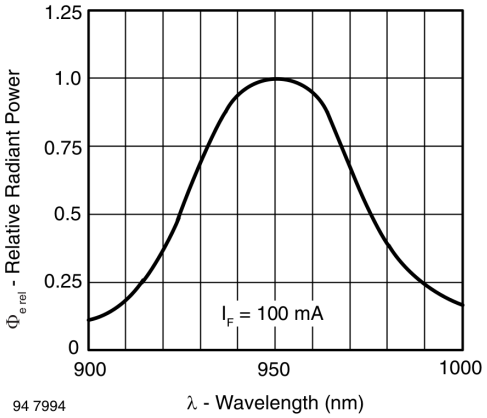
Prosjektet brukte et enkelt webkamera til å fungere som plattformens øye. Figur 5.2 viser webkameraet som ble brukt. Det var av typen Trust Exis webcam, og har en sensoroppløsning på 640 x 480. Kameraet har et manuelt justerbart fokus, selvjusterende feste til datamaskin og USB-A-tilkobling med USB-versjon 2.0 [34]. Kameraet veier 91 g, har en høyde på 77 mm, bredde på 45 mm og dybde på 66 mm. Linsen gir kameraet et vidsyn på ca. 30 grader. Kameraet er kompatibelt med drivere i Linux, det er er klart til å brukes ved tilkobling.

Figur 5.2: Trust Exis Webkamera [33]

IR-diode: TSUS5402

Prosjektet har brukt en IR-diode av typen TSUS5402, se figur 5.3, som sender ut elektromagnetiske bølger i det infrarøde spekteret. Figur 5.4 viser at dioden stråler mest med en bølgelengde på 950 nm [26], lengre enn det menneskelig syn kan oppfatte. TSUS5402 lyser med en intensitet på 20 mW/sr (milliwatt per steradian) [26]. Diodens lysintesitet halveres ved en vinkelendring på $\pm 22°$ fra pekeretning [26].

Figur 5.3: TSUS5402 IR-diode [35]



Figur 5.4: IR-diodens utstråling ut fra bølgelengde [26]

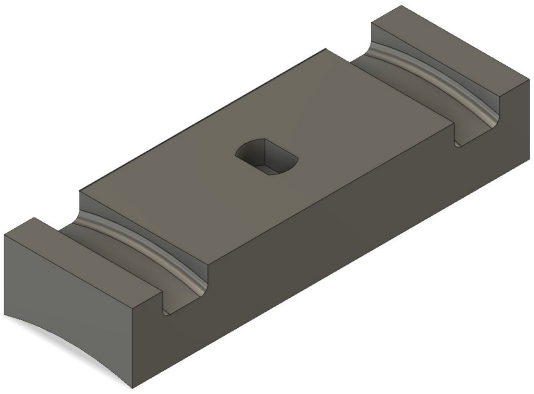
Lee Polyester 87 Infrared

IR-filteret som er brukt i prosjektet er *Lee Polyester 87 infrared*. Dette filteret er laget for å filtrere bort alt lys med en bølgelengde på mindre enn 730 nm [37]. Filteret slepper ikke inn synlig lys, bare lys fra det infrarøde spekteret. Filteret er laget i en type fleksibel polyester som det er mulig å kutte med kniv, noe som har gjort det enkelt å tilpasse kameralinsens utforming. Dette gjør at det enkelt kan skjæres ut et filter med passform til kameralinsen.

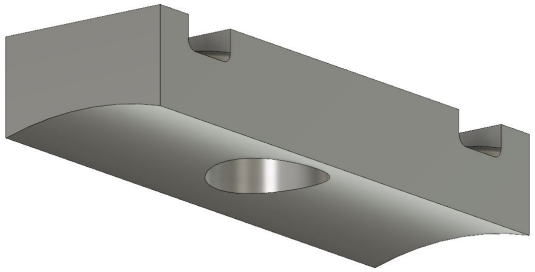
Figur 5.5: Lee 87 Infrared IR-filter [36]

Kamerafeste

Det ble designet og 3D-printet et kamerafeste som gjorde at kameraet enkelt kunne festes på og tas av robotarmen.



Figur 5.6: Festebrakett



Figur 5.7: Festebrakett sett fra undersiden

Det originale kamerahuset har en liten superelliptisk plastikktapp som ble utgangspunktet for festet mellom selve kameraet og den printede delen. Det printede festet var en rektangulær boks med dimensjonene: 52,6 mm x 17 mm x 9,2 mm. På den ene breddesiden av boksen var det et hull til den plastikktappen og et par stroppespor. Undersiden var avrundet til å passe robotarmens krumning ved håndleddet. For å feste den printede boksen ble det brukt strips rundt håndleddet, og et tynt lag gummi under selve festet. Gummilaget økte friksjonen mot robotarmen, og forhindret forskyvning av kamerabildet underveis i testingen. Det var praktisk å kunne enkelt montere kameraet på og av, ettersom modifikasjoner til kameraet ble gjort underveis i prosjektet.

OpenCV

OpenCV (Open Source Computer Vision Library) er et programvarebibliotek for datasyn og maskinlæring [38]. Biblioteket innholder ferdige funksjoner for bildebehandling som har implementert ferdige optimaliserte algoritmer. En bruker av biblioteket trenger dermed ikke å sette seg inn i de underliggende algoritmene for bildebehandling, men kan benytte seg av ferdige funksjoner basert på etablert kunnskap innen datasyn og bildebehandling.

5.2.2 Maskinvarekonfigurering

For at kameraet skulle fungere som ønsket, var det noen modifikasjoner som måtte gjøres. Det første som ble gjort var å demontere kamerahuset fra det originale festet. Deretter ble kamerahuset åpnet, og man kunne demontere kameralinsen fra sensorbrikken. På baksiden av linsen lå det originale anti-IR-filteret som fjerner IR-lys fra kamerabildet. Dette filteret bestod av en kvadratisk 2 mm X 2 mm glassbrikke med et tynt lag filterfilm på. Filteret ble fjernet forsiktig uten å gjøre skade på linsen eller sensorbrikken, som begge ble brukt videre.

Etter fjerningen av anti-IR-filteret var neste steg å sette inn det valgte IR-filteret. For enkelhetensskyld ble det valgt å plassere IR-filteret inne i kamerahuset, foran kameralinsen. Dermed ville den sitte fast i kamera, være enkel å ettersjekke og det var mindre sjanse for at filteret ville endre positur og slippe inn feil lys. Denne modifikasjonen førte til at kameraet nå filmet IR-lys istedenfor synlig lys.

Det neste steget var å feste kameraet på robotarmen. Fra en analyse av robotarmen viste det seg at robotarmens første håndledd hadde en flate som ikke kunne kollidere med andre ledd i robotarmen. Her ble kamerafestet stripset fast, og kameraet ble plassert. Dermed unngikk prosjektet å ta hensyn til faren for å ødelegge kamerat ved bruk av robotarmen.

Etter testforsøk med gjentatte av- og på-plassering av kameraet oppdaget vi at kamerabildet hadde forskjøvet seg. Dette skyldtes at kamerabrikken inne i kapselen hadde noen millimeter margin til å flytte seg på. Dette er ikke et problem når kameraet er stasjonært plassert, men det ble et problem i prosjektet. Kamerabrikken ble derfor limt fast og kamerahuset fylt med skum for å holde komponentene på plass.

5.2.3 Kalibrering

Forvrengning

Etter maskinvarekonfigureringen var det behov for å kalibrere kameraet slik at vi fikk best mulig kvalitet på bildestrømmen. På grunn av linsen og den enkle kodingen som ligger i kameraet fra før, har det digitale bildet ut en forvrengning kjent som fiskeøyeeffekt. Det er laget en pakke i ROS som heter *camera_calibration* som tar seg av en slik forvrengningen i et monokulært kamera. Kalibreringen skjer ved at kameraet tar mange bilder av et sjakkbrettmønster med kjente dimensjoner fra ulike avstander og vinkler. Det ble derfor skrevet ut et 9 x 7 sjakkbrettmønster på et A4-ark med rutestørrelse 24,4 mm x 24,4 mm for å fjerne webkameraets forvrengning.

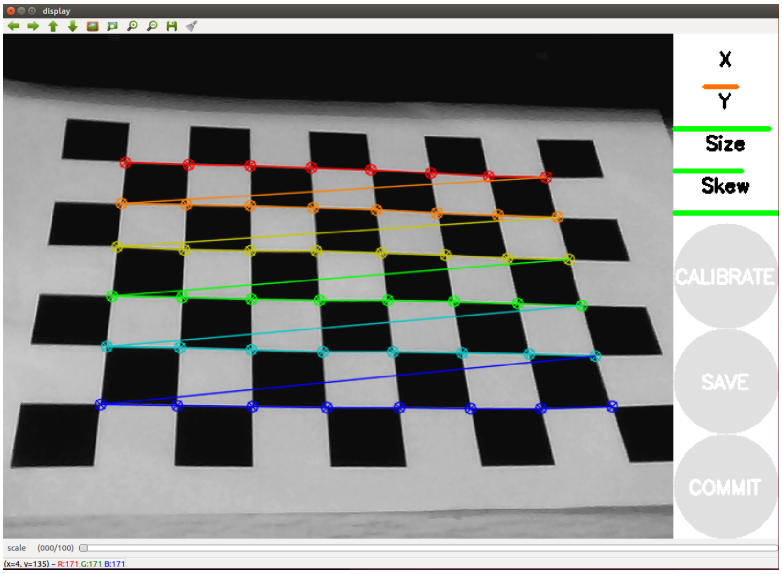
For at kalibreringen skal kunne utføres er det noen forutsetninger som må ligge til rette. Kamerabildet må publiseres i ROS som en egen node. Vi bruker ROS-pakken *usb_cam*, som med den følgende kommandoen setter opp kameraet som en egen node:

```
rosrun usb_cam usb_cam_node _pixel_format:=yuyv
```

Når bildestrømmen publiseres i ROS kan følgende kode kjøres:

```
rosrun camera_calibration cameracalibrator.py --size 8x6
--square 0.0244 image:=/usb_cam/image_raw camera:=/usb_cam
```

Denne koden informerer om rutemønsterets dimensjoner og starter programmet som kjører kalibreringen i tillegg til å informere om sjakkbrettets dimensjoner. Merk at koden bruker 8x6 og ikke 9 x 7. Dette skyldes at programmet trenger å vite antall indre hjørner på brettet langs x-aksen og y-aksen. Figur 5.8 viser programmet underveis i kalibreringsprosessen. Det må bli tatt bilder med brettet i nok ulike posisjoner til at *X*, *Y*, *Size* og *Skew* i figur 5.8 blir helt grønne, og kalibrasjonen kan lagres i kameraet ved å trykke *COMMIT* [39].

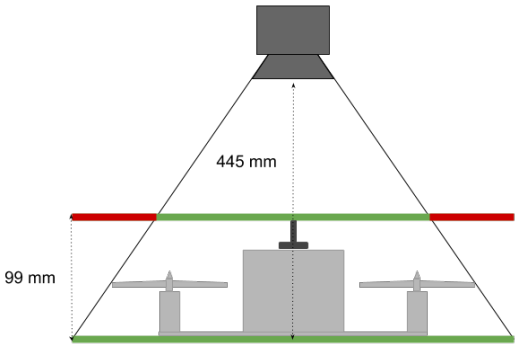


Figur 5.8: Gjenkjenning av sjakkbrettet underveis i kalibreringen

Koordinattransformasjon

Etter at kameraet var kalibrert for å få vekk forvrengningen som linsen skapte, var det neste steget å sørge for at kamerakoden var riktig kalibrert i forhold til kameraets synsfelt. For at robotarmen skal plukke opp dronen nøyaktig er det viktig at koden er stilt inn etter kameraets plassering og orientering. Det ble derfor valgt ut èn positur som robotarmen skulle ha, for observasjon av dronens lokasjon. Denne posituren var utgangspunkt for å lokalisere dronen, og var en positur robotarmen ville vende tilbake til. Den valgte posituren var et av områdene kameraet dekket mest potensielt landingsareal, og var derfor et godt utgangspunkt videre i prosjektet.

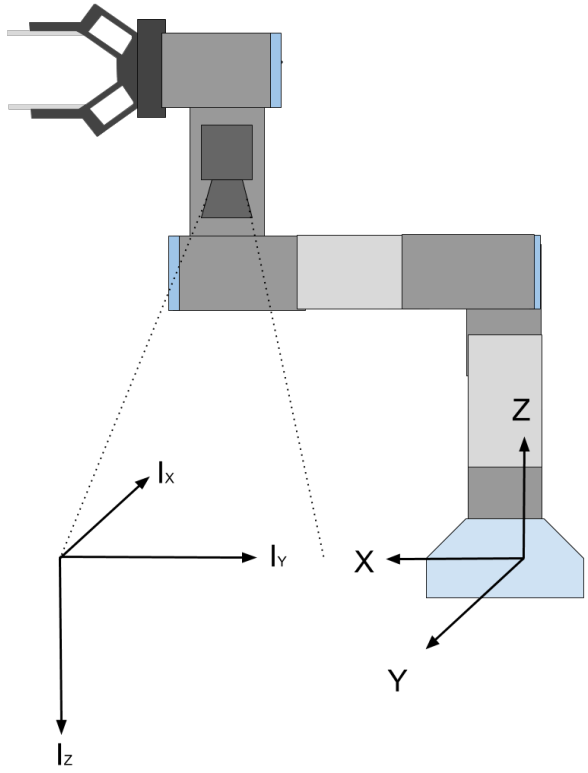
Da bildetagningsposisituren var funnet kunne kameraet bli plassert på armen, og kameraets synsfelt bli tolket. For at kameraet skal kunne finne dronen riktig, var kameraets og dronens høyde viktig å ta med i beregningen. I bildetagningsposituren stod kameraet i en høyde på 445 mm over landingsplattformen. Kameraet observerte et rektangulært område med målene 278 mm x 209 mm. Med batteri og gripefinne har dronen en høyde på



Figur 5.9: Visuell fremstilling av effektivt landingsområde

99 mm. Figur 5.9 viser at det er en forskjell mellom størrelsen på landingsområdet i kamerabildet, mot det effektive landingsområdet med dronens høyde tatt i betraktning. For at den videre kalibreringen av kameraet skulle bli riktig ble robotarmen instruert til å senke det første håndleddets høyde med 99 mm. Dermed havnet kameraet i en høyde på 346 mm over landingsplattformen, som tilsvarer høyden fra gripefinnen og opp til kameraet. Ved å ta dette med i beregningen ble kalibreringen videre gjort riktig.

Nye fysiske målinger på landingsplattformen ble gjort, og de viste at det effektive landingsområdet ble et rektangulært område med målene 216 mm x 162 mm. Dette er da målene på området kameraet observerer i høyden til gripefinnen fra kameraets bildetagningspositur.



Figur 5.10: Koordinatsystemene til kameraet og robotarmen

For at kameraet og robotarmen skal kunne virke sammen var det viktig å samkjøre begges koordinatrammer. Origo for et kamerabilde ligger oppe i venstre hjørne [40], mens origo for robotarmen er midt i basen. Z-aksen pekte motsatt vei i de to koordinatrammene, som vist i figur 5.10. I tillegg var ikke x- og y-aksene parallelle, så vinkelen mellom dem måtte måles og tas høyde for. Vi måtte også vite hvor lang avstand pikselverdiene i bildet representerte. Det var altså tre størrelser som måtte måles: Koordinatet til bildets origo i robotens xy-plan,

vinkelen mellom y-aksene (eller x-aksene), og skaleringen på bildet.

Koordinatet fra bildet kan oversettes til robotens koordinatramme ved hjelp av en transformasjon. I dette tilfellet finnes koordinatet i robotens koordinatramme ved utregningen som vist i formel 5.1. Her er I_x og I_y koordinatet funnet fra bildet, og C er et konstant forhold mellom pikselverdi og avstand. θ er rotasjonen om z-aksen. d_x, d_y og d_z representer forskyvningen mellom de to origoene.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} -cos\theta & sin\theta & 0 & d_x \\ sin\theta & cos\theta & 0 & d_y \\ 0 & 0 & -1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I_x \cdot \mathrm{C} \\ I_y \cdot \mathrm{C} \\ 0 \\ 1 \end{bmatrix} \tag{5.1}$$

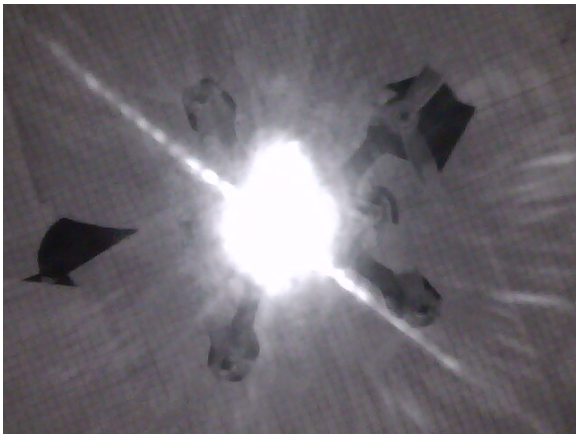
Transformasjonen ble hardkodet i camNode.py, som vist i kodeutsnittet i figur 5.11. Forskyvingen mellom de to origoene ble funnet ved å markere hvor origo i bildet var når kameraet var i bildetakingsposisjonen, for så å måle avstanden til robotens x- og y-akser. Som vi ser av linje 46 i kodeutsnittet var det 356 mm i x-retning, og 297 mm i y-retning. Forskjellen i z-retning er 0, fordi kameraet ser på bordplaten som utgjør robotens xy-plan. Rotasjonen som trengtes om z-aksen ble målt til å være $-71°$. Til sist ble skaleringen på pikslene funnet ved å bruke bredden på synsfeltet, 216 mm, og dele denne avstanden på antall piksler i bredden, 640.

```
35 #coordinate transform:
36 #rotation matrices:
37 rotY = [[np.cos(np.pi),0,np.sin(np.pi)],
38         [0, 1, 0],
39         [-np.sin(np.pi), 0 ,np.cos(np.pi)]]
40 rotZ = [[np.cos(-71*np.pi/180),-np.sin(-71*np.pi/180), 0],
41         [np.sin(-71*np.pi/180), np.cos(-71*np.pi/180), 0],
42         [0, 0, 1]]
43 rot = np.dot(rotY, rotZ)
44
45 #translation
46 displacement = [[0.359],[0.297], [0]]
47 translationMat = np.concatenate((rot, displacement),1)
48 translationMat = np.concatenate((translationMat, [[0,0,0,1]]),0)
```

Figur 5.11: Kodeutsnitt fra camNode.py

Eksponering

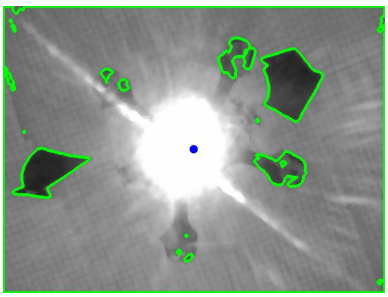
For at kameraet skulle kunne skille IR-dioden fra annet støy måtte kameraets eksponering justeres ned. Støy kunne være gjenskinn fra blanke overflater som rotorblader og droneskallet eller andre ting rundt som ble for lyst. Figur 5.12 viser dronen med dioden slått på fra bildetagningsposisjonen. Her står dronen i posisjon rett under kameraet, og det sterke lyset fører til at kameraet får et veldig diffust bilde av dronen. Figur 5.13 viser det ønskede bildet av dioden. I dette bildet er eksponeringen stilt ned, og dioden fremkommer tydelig på kamera. Et slikt bilde kan brukes av OpenCV for å regne nøyaktig posisjon.



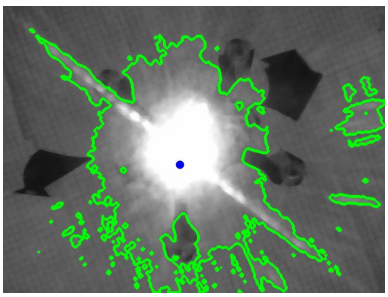
Figur 5.12: Dronen sett fra bildetagningsposisjon uten parameterendring

Figur 5.13: Drone sett fra bildetagningsposisjon etter parameterendring

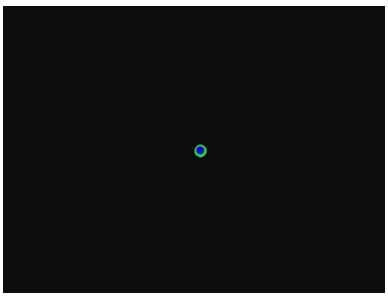
Driveren som snakker med kameraet, *v4l2* (Video 4 Linux), lar brukeren se og stille parametre for kameraet. For å unngå refleksflekker måtte vi stille ned kameraets eksponering, dette gjøres i *v4l2* med kommandoene *exposure_auto*=1 og *exposure_absolute*=0. Figurene 5.14, 5.15 og 5.16 viser konturene som finnes ved hjelp av OpenCV med forskjellige innstillinger. Disse innstillingene fører til at det blir enklere for kameraet å lokalisere dronen og mindre sjanse for at annet støy forveksles med dioden. I figur 5.14 er nesten hele kamerabildet over terskelen for kontraster. Dette gjør at den blå prikken som er midtpunktet for det grønne området/rammen, ikke treffer nøyaktig på dioden. OpenCV er programmert til å finne midtpunktet i den grønne rammen. Ettersom den grønne rammen er langs hele bildet ender den blå prikken med å havne ca. midt i bildet. Figur 5.16 viser hvordan lav eksponering gjør at kun selve dioden overstiger terskelen for kontraster, og det er denne som blir gjenkjent. Den grønne rammen er bare rundt dioden, og den blå prikken ender derfor i sentrum, og midt på dioden. Med eksponeringen justert ned til det minimale nivået, er det kun dioden som blir gjenkjent på kameraet.



Figur 5.14: Automatisk eksponering



Figur 5.15: Eksponeringsverdi satt til 625



Figur 5.16: Eksponeringsverdi satt til 0 (minimum)

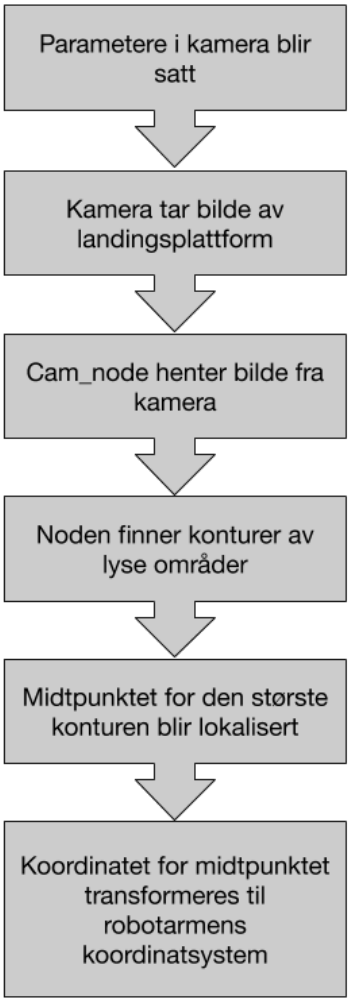
5.2.4 Programmere ROS-node

Vi endte med en egen node i ROS for bildebehandling. Vi bruker en ferdiglaget pakke som heter *usb_cam* for å få videostrømmen fra et USB-kamera over til ROS. Noden tar inn bildedata fra *usb_cam*-pakken og publiserer et koordinat relativt til roboten på en egen ROS-topic. Noden vil kun sende ut et koordinat om den finner en sterk verdi i bildet, altså om vi har en IR-LED i bildet. Kildekoden til noden, *camNode.py*, er vedlagt i sin helhet.

Slik det er implementert i praksis er at vi først plukker ut den røde kanalen fra RGB-bildet som kommer fra kameraet. Vi velger den røde kanalen fordi vi ikke trenger alle tre fargekanalene, og den røde delen av sensoren er mest sensitiv til bølgelengden på dioden vår. Deretter gjøres en terskeloperasjon på bildet, slik at vi setter alle verdier over en viss terskel til maksimal verdi, og alle cellene i bildet under terskelen til 0. Neste steg er å finne konturen til IR-dioden i bildet. Til dette brukes funksjonen *findContours*. Den gir oss alle konturene i det tersklede bildet. Optimalt vil det kun være IR-dioden som er over terskelen i bildet, men i tilfelle refleksflekker eller annen støy i bildet velger vi ut den største konturen i bildet. Vi finner bildekoordinatet til midtpunktet av denne konturen, og matrisemultipliserer det med matrisen beskrevet ovenfor for å få omsatt bildekoordinatet til koordinatsystemet til roboten.

5.3 Resultat

Denne delen tar for seg resultatet av modulens maskinvare og programvarens virkemåte og funksjon. Resultatet for denne modulen ble et system som bruker et IR-kamera for observering av det effektive landingsområdet og en programvare som tolker bildet og transformerer IR-diodens bildekoordinat til robotarmens koordinatsystem. Figur 5.17 viser handlingsmønsteret for modulen.

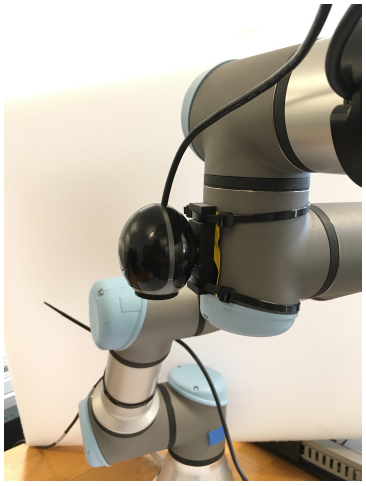


Figur 5.17: Handlingsmønster for lokaliseringsmodulen

5.3.1 Maskinvare

Resultatet av maskinvarekonfigureringen er et IR-kamera som fanger opp IR-lys med bølgelengde på 730 nm og over, se figur 5.18. Filteret som slipper inn IR-lyset ligger foran linsen på innsiden av kamerahuset. Kameraets kretskort er limt fast i kamerahuset og er robust festet for eventuelle støt og slag mot kamerahuset. På baksiden av kameraet er USB-kabelen som må fysisk kobles til datamaskinen som kjører ROS. Kameraet kobles på robotarmens første håndledd ved hjelp av kamerafestet som er permanent låst fast, se figur 5.19.

Figur 5.18: Kamera med IR-filter



Figur 5.19: Kamera montert på første håndledd på robotarm

Når robotarmen står i bildetagningspositur dekker kameraet et rektangulært område på landingsplattformen med dimensjonene 278 mm x 209 mm. Med å ta dronens høyde i betraktning får det effektive landingsområdet målene: 216 mm x 162 mm.

5.3.2 Programvare

Ved oppstart av kameramodulen justeres eksponeringen av bildet ned, slik at kun lyset fra dioden er synlig. Bildet fremstår som ganske mørkt med få kontraster. Programmet i ROS henter inn bildet fra kameraet og analyserer dette. Analysen leter etter objekter i kamerabildet med høy kontrastverdi. Dioden gir fra seg et såpass sterkt lys at den veldig enkelt skiller seg fra dronen og landingsplattformen. Dette objektet blir lokalisert og det regnes ut et midtpunkt. Videre får punktet pikselkoordinater. Gjennom kalibreringen blir pikselkoordinatene omgjort til robotarmens koordinater, og et punkt for dronen er funnet.

5.4 Drøfting

Resultatmessig fungere kamermodulen slik den var tiltenkt, men det er noen oppdagelser som ble gjort underveis.

Det ble gjort erfaringer i løpet av prosjektet knyttet til kameraets mangel på kvaliteter. En del tid kunne nok ha vært spart om man hadde valgt et bedre kamera tidlig i fasen. Samtidig har arbeidet med et mangelfullt kamera også gitt gode erfaringer som vi kanskje ikke ville ha

fått uten. For eksempel kalibrering ble enda viktigere. Som nevnt tidligere ble Trust Exis webkameraet valgt på grunn av kjennskapen til kameraet og dets enkelhet i oppbygning. Derimot hadde kameraet noe dårlig byggekvalitet ettersom databrikken flyttet seg internt i kamerahuset. Dette ble løst ved å feste kamerabrikken til det originale kamerahuset. Det ble også vurdert å 3D-printe et eget kamerahus som ville være plasseffektivt, og minske faren for at kameraet skulle kollidere med noe.

Kvaliteten på det digitale bildet var også noe mangelfullt. Oppløsning på 640x480 ga noe uklare bilder, og ved presisjonsarbeidet rundt kalibrering var dette en ulempe. Bildet hadde så lite skarpe detaljer at det var vanskelig å se om kalibreringen av forvrengingen var god nok. Den ble utført flere ganger, men det var ikke så merkbart bedre resultat. En annen ulempe ved kameraets uskarpe bilde var at oppmåling av det faktiske synsfeltet ikke ble like nøyaktig som ønsket. De få millimetrene som eventuelt ble feil har ikke hatt den store betydning for deteksjonen, ettersom gripefinnen er laget for å korrigere denne feilen.

Kameraet har en synsvinkel på ca. 30° og med kameraet i en høyde på 445 mm så dekker dette et område i landingsarealet på ca 278 mm x 209 mm. Det effektive landingsområdet blir enda mindre igjen på grunn av dronens høyde. Og dette området tilsvarer et rektangulært område på 216 mm x 162 mm. Dette området er ikke like stort som ønsket, og det kunne ha vært løst ved innkjøp av et bedre kamera med større synsvinkel. Det ble vurdert, men oppdragsgiver så det ikke hensiktsmessig ettersom det originale kameraet gav prosjektet et resultat som fungerte. Det ville også ha medført en del ekstra arbeid med å ordne IR-filtrering, kalibrere på nytt og lage et fungerende feste til robotarmen. Oppgaven fokuserer mer på det å lage en fungerende løsning, mer enn å lage den optimale løsningen. Å lage den optimale løsningen hadde krevet mer tid og ressurser enn det dette prosjektet hadde til rådighet.

At kameraet ble plassert på robotarmens første håndledd har medført enklere programmering. Robotarmen er laget på en måte som gjør det umulig å kollidere med seg selv i den flaten kameraet er plassert. Det var dermed ikke nødvendig å ta høyde for å legge inn kameraet som et kollisjonsobjekt i ROS. Dette medført enklere kode, og det ble mindre sjanse for skade på utstyret.

Kameraet har også mulighet til å observere nærliggende omgivelser. Ved en endring i paramaterene vil kameraet gi samme type bilder som et ordinært overvåkningskamera. Det er da mulig å overvåke landingsplattformen og oppdage andre objekter enn en IR-diode. Ulempen med dette er at det må skrives ekstra kode som justerer kameraets eksponering hver gang en drone skal foreta en landing. Samtidig vil denne effekten gi kameraet en ekstra funksjon.

Denne ekstra funksjonen var ikke ansett nødvendig i dette prosjektet, og det ble dermed ikke laget.

Prosessering av bilder med full oppløsning i OpenCV trenger ganske mye beregningskraft. En datamaskin med mindre kapasitet ville ha brukt lenger tid på å finne dronens koordinat på landingsplattformen. Hadde kameraet hatt enda høyere oppløsning ville OpenCV ha krevd enda større beregningskraft. Det må derfor gjøres en avveining mellom oppløsning på kamera og beregningens kompleksitet. I dette prosjektet har datamaskinen hatt såpass god kapasitet, og kameraet så lav oppløsning at dette ikke har vært et problem. Det er derfor ikke tatt hensyn til denne problemstillingen i metoden.

Fordelen med å jobbe i OpenCV er at det er mye brukt datasynpakke. Det finnes mye åpen kildekode på programmer som har benyttet OpenCV sammen med ROS. I prosjektarbeidet var det en god hjelp å kunne se andres måter å implementere OpenCV på, som forenklet dette arbeidet og sparte oss mye tid, og programmering.

5.5 Alternative løsningsmetoder

5.5.1 Krysspeiling

En annen måte å finne dronens lokasjon på kunne være å benytte seg av krysspeiling. Det kunne da ha vært plassert to avstandsmålere i hvert sitt hjørne av landingsplattformen, med mulighet til å rotere. Ved å sveipe landingsområdet med to avstandsmålere fra ulike posisjoner vil kryssproduktet av disse indikere hvor dronen står. Dette ville ha krevet at dronens gripefinne var utstyrt på en måte som lot seg oppdage av avstandsmålerene. Systemet kan implementeres på en måte som gjør at dronen utgjør en passiv rolle i arbeidet knyttet til å finne posisjonen.

5.5.2 Sensor i landingsplattform

En annen metode for å finne dronens lokasjon kunne være å implementere en sensor i landingsplattformen som merker at dronen har landet, finner dens posisjon, og videreformidler dette til robotarmen. En slik landingsplattform ville ha fungert på samme måte som en berøringsmatte (eng *touchpad*) på en datamaskin. Løsningen ville ikke nødvendigvis ha krevet noen inngrep på dronen, avhengig av berøringsmattens virkemåte. Systemet kunne dog ha vært sårbart for omgivelsene.

6 Orientering

Dette kapittelet tar for seg utfordringen knyttet til å finne dronens orientering.

Under *Teori* blir det redegjort for nødvendige forkunnskaper, samt noen grunnleggende teoretiske prinsipper. I *Metode* blir det beskrevet hvorfor og hvordan løsningen vi valgte ble produsert og satt sammen, før *Resultat* redegjør for funksjonen til produktet. *Drøfting* vurderer hvorvidt resultatet har innfridd de målene som ble satt, hva som fungerte godt, og hva som kunne ha vært gjort annerledes. Til slutt vil *Alternative løsningsmetoder* ta for seg hva vi som gruppe vurderer som andre måter å løse utfordringen på.

6.1 Teori

6.1.1 Mikrokontroller

En mikrokontroller er en programmerbar prosessor [41] med tilhørende minne på en integrert krets. Den integrerte kretsen kan ved hjelp av en rekke periferienheter fungere som en elektronisk måle- eller kontrollenhet, eller en kombinasjon av disse. Mikrokontrollere egner seg til oppgaver der det stilles høye krav til responstid, størrelse og strømforbruk. Mikrokontrollere er mindre egnet til å utøve komplekse kalkulasjoner, da de ofte har få kjerner, lav klokkehastighet og lite internt minne.

Lav pris, gode spesifikasjoner og tilgjengelighet har ført til stor utbredelse og omfattende bruk av mikrokontrollere i forskjellige applikasjoner. Dette har ført til utbredt og tilgjengelig åpen kildekode, som igjen har gitt opphav til et stort felleskap for brukerstøtte og fildeling. Dette felleskapet har et stadig voksende utvalg av programvarebiblioteker for forskjellige funksjoner og tredjepartskomponenter.

6.1.2 Kommunikasjonsprotokoller

For å forsikre seg om at to eller flere digitale enheter er i stand til å kommunisere sammen er det nødvendig å etablere et sett med regler for informasjonsutveksling mellom dem. [42].

SCLK	Serial Clock
MOSI	Master Out Slave In
MISO	Master In Slave Out
SS	Slave Select

Tabell 6.1: Nødvendige ledere i SPI-protokollen

Serial Peripheral Interface

Serial Peripheral Interface (SPI) er en synkron full dupleks seriell kommunikasjonsprotokoll. Fysisk består protokollen av fire eller flere ledere, beskrevet i tabell 6.1, avhengig av hvor mange enheter masteren skal kunne kommunisere med.

For å sende en melding genererer først masteren et felles klokkesignal for nettverket *SCLK*, for så å velge hvilken slave det skal kommuniseres med. Dette gjøres via *SS*, ofte ved å sette denne lav hos slaven det ønskes å kommunisere med. Kommunikasjon mellom de to enhetene gjøres så på henholdsvis *MOSI*, dersom master skal skrive til slaven, og *MISO*, dersom slaven skal skrive til masteren [43].

I teorien vil det være mulig å kommunisere med et uendelig antall individuelle slaver i et nettverk, men i praksis er ikke dette gjennomførbart grunnet behovet for da å ha et uendelig antall ledere ut fra masterne, til hver individuelle slave.

Inter-Integrated Circuit

Inter-Intergrated Circuit (I^2C) er en synkron halv-dupleks seriell kommunikasjonsprotokoll.Fysisk består protokollen av to ledere, uavhengig av hvor stort nettverket er. Individuell kommunikasjon mellom enhetene oppnås ved å benytte 7- eller 10-bits adressering, som gir en master 127 eller 1023 potensielle slaver å kommunisere med.

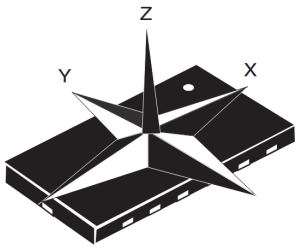
SDA	Serial Data
SCL	Serial Clock

Tabell 6.2: Nødvendige ledere i I^2C-protokollen

Kommunikasjon gjøres ved at masteren etablerer et klokkesignal på *SCL*, for så så sende en startbetingelse på *SDA*. Deretter følger adressen til den aktuelle slaven, og en instruksjon på hvorvidt det skal skrives til eller leses fra slaven. Den ønskede kommunikasjonen foretas etter dette [44].

6.1.3 Magnetometer

Et magnetometer er et instrument som er i stand til å måle styrken i et magnetfelt. Disse instrumentene kan produseres som integrerte kretser, og har ofte grensesnitt for SPI- eller I^2C-kommunikasjon. Dette gjør at de kan kommunisere med mikrokontrollere eller annen maskinvare som er i stand til å lese målingene. Mange moderne magnetometer i integrerte kretser gjennomfører ofte tre



Figur 6.1: Magnetiske målinger foretas i de tre planene X, Y, Z

målinger samtidig, i tre forskjellige plan som vist i figur 6.1.

Målinger av magnetiske feltstyrker oppgis i *Tesla*. Dersom et plan settes vinkelrett på et magnetfelt vil det avleses en positiv verdi. Dersom planet roteres 180°rundt sin egen akse, vil verdien bli negativ. Settes planet parallelt med magnetfeltet vil den magnetiske feltstyrken i måleområdet bli 0.

Figur 6.2: Positiv magnetisk feltstyrke gjennom planet

Figur 6.3: Ingen magnetisk feltstyrke gjennom planet

Figur 6.4: Forskjellig feltstyrke i plan X og Y

Ved å benytte seg av to eller flere målinger av et magnetfelt vil det være mulig å beregne magnetfeltets relative retning til måleinstrumentet. På denne måten kan informasjonen hentet fra et magnetometer med to eller flere målinger, brukes til å regne magnetometerets kompasskurs relativt til jordens magnetfelt.

Ved å ta inverstangens til to av planenes målte magnetiske feltstyrke, og omgjøre dette fra radianer til grader, får man planenes retning i forhold til magetfeltet (formel 6.1).

$$\text{Magnetisk kurs} = \arctan\left(\frac{\text{PlanX}}{\text{PlanY}}\right) \cdot \frac{180°}{\pi} \tag{6.1}$$

6.1.4 Kretskort

Kretskort brukes for å feste elektroniske komponenter på en slik måte at de innfrir et ønsket sett med spesifikasjoner etter montering. Kretskort tilrettelegger for å holde elektroniske komponenter samlet, så vel som å opprette elektronisk forbindelse mellom dem.

Det finnes en rekke forskjellige måter å fremstille slike kretskort på, og et flertall av disse benytter seg av DAK-verktøy. Ved hjelp av feilsøking og revisjonskontroll bidrar disse verktøyene til at fremstillingsprosessen blir raskere, enklere og billigere.

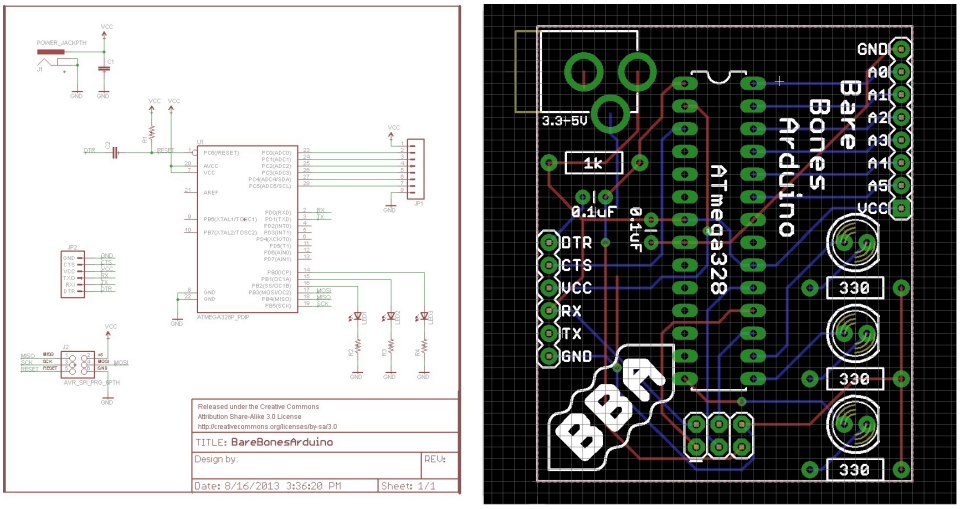
Ved bruk av DAK er fremstillingsprosessen i hovedsak todelt: *Skjematisk tegning* (eng. *schematic*) og *Utlegg* (eng. *board layout*).

Fase 1: Skjematisk tegning

Den skjematiske tegningen har til hensikt å representere de elektroniske forbindelsene mellom komponentene. Den ønskede kretsen tegnes rent symbolsk, uten å ta stilling til fysiske dimensjoner. Etter at kretsen er tegnet ferdig, tillegges fysiske dimensjoner til de forskjellige symbolene, også kalt *fotavtrykk*, slik at denne dataen kan benyttes i andre fase av tegningen.

Fase 2: Utlegg

Den skjematiske tegningen (se eksempel i figur 6.5), med nå tilhørende fysiske dimensjoner lagret i sine *fotavtrykk*, blir så videreført til et program som har til hensikt å generere en datafil som beskriver de endelige dimensjonene på det ferdige kortet. I dette programmet spesifiseres komponentplassering og den endelige utformingen på kortet, og det tegnes hull og baner ved hjelp av informasjonen fra den skjematiske tegningen. Det er i denne fasen en rekke faktorer som må ivaretas. Eksempelvis må banebreddene dimensjoneres etter hvor mye strøm som skal gå i dem, komponenter som omsetter mye effekt må få termisk lettelse (eng. *thermal relief*), og signalbaner må utformes slik at de er mest mulig skjermet mot støy.



Figur 6.5: Et eksempel på en skjematisk tegning og utlegg i Autodesk Eagle [45]

6.2 Metode

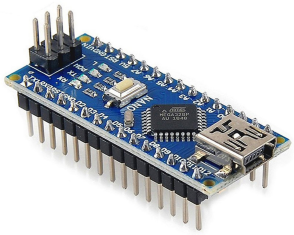
6.2.1 Utstyr og programmvare

Dette er en redegjørelse av utstyret gruppen valgte å bruke i tilknytning til dronen. I dette delkapittelet ligger instrumenter eller annet materiell som enten er brukt direkte i dronen, eller på servicestasjonen for å muliggjøre interaksjon med dronen.

Figur 6.6: Arduino Uno for ROS-noden

ATmega328P

Mikrokontrollerne gruppen har benyttet seg av er av typen ATmega328P fra Atmel i 32-pin-konfigurasjon. Denne mikrokontrolleren har blant annet 32 kilobyte programmerbart *flash minne*, 23 inn/ut-pinner, 8-kanals 10-bit *analog-digital konversjon*, samt grensesnitt for I^2C- og SPI-kommunikasjon. [46]

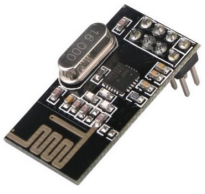


Figur 6.7: Arduino Nano for avlesning av magnetometer

Kontrollerene gruppen har benyttet seg av har vært plassert i to åpen kildekode-utviklingskort. Det ene har vært av typen Arduino Uno (Figur 6.6), det andre av typen Arduino Nano (Figur 6.7). Gruppen valgte å benytte seg av utviklingskort fordi de har fordelen av å ha integrerte *programmerere*, som muliggjør kommunikasjon til chipene ved bruk av UART over USB uten å måtte benytte seg av ekstra maskinvare. Under utvikling har dette kunnet redusere tiden brukt på prototyping vesentlig.

nRF24L01

For trådløs kommunikasjon har gruppen i denne oppgaven benyttet seg av *SoC*-radiosendere av typen *nRF24L01*, som opererer i 2.4GHz ISM-båndet. Disse integrerte kretsene har integrerte forsterkere, krystalloscillatorer, frekvensfremstiller og et SPI-grensesnitt for programmering, skriving og lesing. Kretsen er rimelig og brukt i en rekke applikasjoner, noe som har ført til tilgjengelige og omfattende biblioteker og brukerstøtte. Kretsen har vært montert på et åpen kildekode-utviklingskort av typen *REES52 nRF24L01*.



Figur 6.8: Utviklingskort for nRF24L01

Magnetometer

Magnetometeret som er brukt var av typen *LSM303DLHC*. Dette er en *SiP* med tredimensjonale lineære akselerometer og magnetometer. Kretsen er rimelig, har gode åpen-kildekodebiblioteker og har grensesnitt for I^2C-protokollen. Kretsen har vært montert på et utviklingskort.

Et magnetometer av typen *MPU9255* har også blitt benyttet i arbeidet, men er ikke med i sluttproduktet. Den har dog hatt stor innvirkning på sluttproduktet, og blir derfor gjort rede for. *MPU9255* er et magnetometer svært likt *LSM303DLHC*, også med tredimensjonale lineære akselerometer og magnetometer. Kretsen har til sammenligning med *LSM303DLHC* ikke like mye god åpen kildekode, men det mulig å oppdrive nødvendige biblioteker. Den største forskjellen mellom kretsene er *MPU9255*'s grensesnitt for SPI-kommunikasjon, i motsetning til I^2C. Denne kretsen har i oppgaven også vært montert på et åpen kildekode-kort.

Arduino 1.8.9

Det var nødvendig å installere en IDE (eng. *Integrated Development Environment*), et program som gjør høyere nivåkode som C eller C++ og kompilerer dette til maskinkode for mikrokontrolleren, slik at den kan yte ønsket arbeid. Gruppen benyttet seg her av Arduino sin egen IDE, *Arduino 1.8.9*, da denne kunne tilby rask tilgang til nødvendige biblioteker for de andre utviklingskortene som skulle brukes. Dette var også en IDE som var tilgjengelig for Ubuntu 16.04, hvilket ble ansett som en fordel. Programmet ble lastet ned og installert fra Arduino sim hjemmeside [47]

Autodesk Eagle

Autodesk Eagle er et DAK-program for kretskortdesign. Det er et kraftig verktøy som bistår brukeren i alle fasene av kretskortproduksjonen, ved å integrere sømløse grensesnitt mellom de forskjellige fasene av fremstillingen. Autodesk Eagle har utover sine store samlinger av guider og læreprogrammer, store biblioteker med *fotavtrykk* til komponenter fra en rekke fabrikanter.

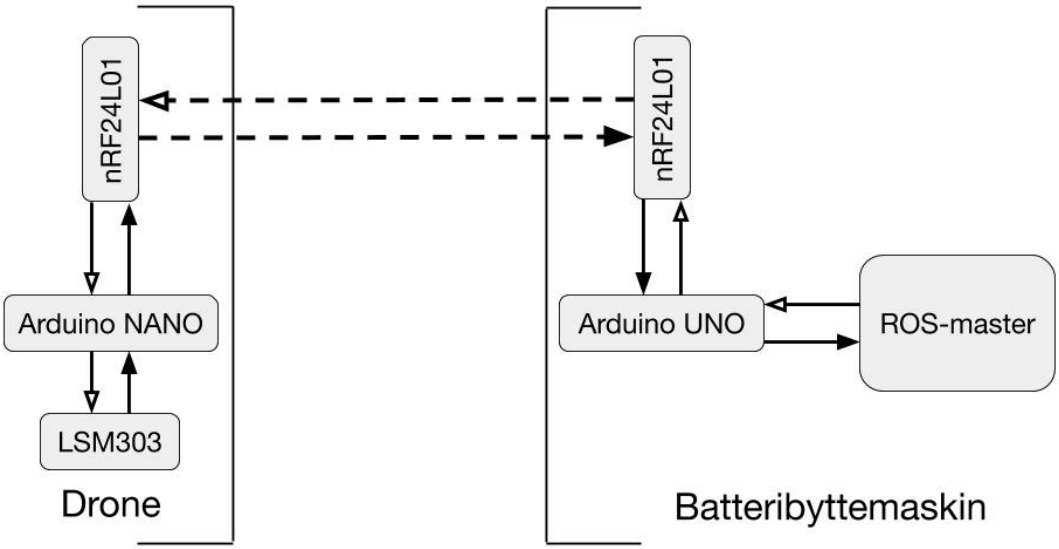
Autodesk Eagle er tilgjengelig i gratisversjoner [48], men da med begrenset funksjonalitet. Vår oppgave hadde ikke behov utover det gratisversjonen kunne tilby.

ROS-serial

ROS-serial er en protokoll for ROS som multiplekser *ROS-topics* og *ROS-services* og derav gjør ROS-kommunikasjonen seriell, primært for kunne å integrere mikrokontrollere i et ROS-nettverk [49].

6.2.2 Kodedesign

Intensjonen med kodedesignet tilknyttet dronen var å få mikrokontrollerne til å kunne redegjøre for dronens status og orientering når dette ble forespurt fra ROS-masteren. Grunnet oppgavens natur ble det besluttet å benyttet sekvensiell kode og maskinvare for å løse dette problemet. De endelige kodene som ble fremstilt fikk derfor en funksjonsflyt lik det gitt av Figur 6.9.



Figur 6.9: Flytskjema ved orienteringsforespørsel

Som det fremkommer av Figur 6.9 blir det sendt en forespørsel fra ROS-masteren via en topic til Arduino Unoen, som da forspør via radioene en avlesning av magnetometeret hos Arduino Nanoen. Denne avlesningen sendes så i retur, og publiseres til slutt på en egen topic fra Arduino Unoen til ROS-nettverket. Dette er en topic ROS-masteren abonnerer på. Det ville ikke være mulig å kommunisere trådløst direkte fra dronen til ROS-nettverket, da den trådløse senderen ikke benytter seg av et grensesnitt datamaskinen har støtte for. For å løse dette problemet ble det besluttet å benytte en mikrokontroller som kunne kommunisere med både dronen, trådløst, og datamaskinen serielt.

Det ble derfor skrevet en kode til hver av de to mikrokontrollerne. *Dronekoden*, som tok seg av det arbeidet dronen måtte utføre, og *stasjonskoden*, som ville gjøre det mulig for det større systemet å interagere med dronen. Det vil her bli redegjort for de forskjellige kodene som er blitt produsert.

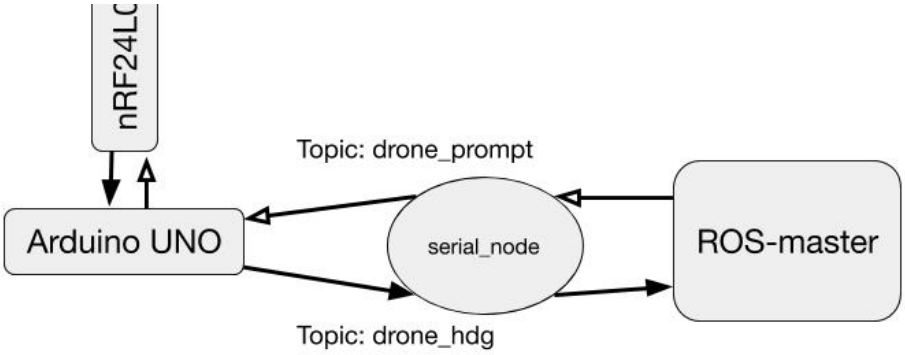
Stasjonskoden

Stasjonskoden uten biblioteker er vedlagt.

Stasjonskoden er den koden som muliggjør kommunikasjon mellom ROS-nettverket og dronen. Den kommuniserer til ROS-nettverket ved at det etableres en node, *serial_node*, for å innhente og svare på forespørsler fra ROS-nettverket. Det var egentlig ønskelig å lage en *ROS-service*, da dette er designet for å håndtere forespørsel-svar [50], men *ROS-serial* har foreløpig dårlig støtte for denne funksjonen [51].

Selv om vi programmerer Arduinoen til å fungere som en node, er det egentlig ROS-nettverket som oppretter denne noden. Denne noden får i oppgave å "oversette" fra ROS-protokoll til seriell kommunikasjon med mikrokontrolleren.

ROS-serial har som nevnt foreløpig dårlig støtte for dupleks-kommunikasjon mellom mikrokontrollere ved bruk av *ROS-services*, og det lar seg egentlig ikke gjøre for mikrokontrolleren å benytte seg av denne funksjonen. For å komme oss rundt dette problemet ble det heller opprettet en node som kommuniserer på to ulike topics: én mikrokontrolleren indirekte lytter til, og én mikrokontrolleren indirekte skriver til. På den måten oppnår vi store deler av funksjonen en *ROS-service* har.



Figur 6.10: Faktisk flytskjema som viser kommunikasjon mellom Arduinoen og ROS-nettverket

I *stasjonskoden* linje 1-8 importeres nødvendige biblioteker for *ROS-serial*, *SPI-protokollen*, *nRF24L01* og to ROS-messages: `UInt8` og `Bool`. Baud raten settes kun for å kunne monitorere prosessen underveis og bistå feilsøking, men er ikke nødvendig for å oppnå funksjonaliteten til koden man ønsker.

Videre settes kanalene radioen skal sende og lytte på:

```
10 RF24 radio(9,10); // CE, CSN
11 const byte addresses[][6] = {"sNode","dNode"};
```

I linje 14 etableres det så en `NodeHandel`. En `NodeHandel` er en C++-funksjon som muliggjør oppstart og avslutning av noder i C++. Det gjør også at det blir lettere å benytte dette sammen med underkomponenter i C++[52].

Deretter gis en *standard message* navnet `hdgMsg`, og det etableres en topic denne publiseres på. Dette vil si at når vår nye node *serial_node* publiserer, publiseres det en `UInt8`-verdi med navn `hdgMsg` til topicen *drone_heading*.

```
16 std_msgs::UInt8 hdgMsg;
17 ros::Publisher drone_heading("drone_heading", &hdgMsg);
```

Topicen det lyttes til i påvente av en forespørsel om å finne dronens orientering, settes også opp noe lenger ned i koden. Av koden ser vi at det lyttes etter en `Boolsk`-verdi med navn `promptHeading` på topicen *drone_prompt*.

```
28 ros::Subscriber<std_msgs::Bool> drone_prompt("drone_prompt", &promptHeading);
```

Å lytte på en ROS-topic fungerer i koden ved at man definerer en funksjon som kalles hver gang en melding mottas på topicen, en såkalt *callback*-funksjon. *Callback*-funksjonen vi har laget (linje 19-26) sjekker først om det vi mottar er `Boolsk` *sann* eller *usann*. Dersom det er *sann* kjøres resten av koden som ved hjelp av funksjonen `getHeading()` henter inn dronens orientering, og publiserer denne `UInt8`-verdien til topicen *drone_heading* under navnet `hdgMsg`.

```
19 byte promptHeading (const std_msgs::Bool& prompt_msg){
20   if(prompt_msg.data){
21     heading = getHeading();
22     hdgMsg.data = heading;
23     Serial.println(heading);
24     drone_heading.publish( &hdgMsg );
25   }
26 }
```

`byte getHeading()`-funksjonen er en funksjon som benyttes etter at det er kommet en forespørsel fra *ROS-masteren* på topicen *drone_prompt*. Funksjonen har som mål å forespørre dronen om orientering, og returnere dette som en `UInt8`-verdi (0-255).

Funksjonen starter ved at den stopper å lytte på mottakerkanalen. Deretter initierers radioen og det sendes en `Int` med verdi 1. Straks denne er sendt bytter den til mottakerfrekvensen. For å unngå å lese av en eldre verdi som fortsatt ligger på båndet, venter den i 20 ms. Etter dette leser den av og returnerer avlest verdi fra *nRF24L01*-radioen.

```
63byte getHeading(){                                    ///Prompts drone for heading
64    radio.stopListening();
65    int pendingRequest = 1;
66    Serial.println("Sending...");                              //Free to comment
67    radio.write(&pendingRequest, sizeof(pendingRequest));
68
69    radio.startListening();
70    delay(20);                                                      //needs 20ms
71    if (radio.available()) {
72        int headingRx;
73        radio.read(&headingRx, sizeof(headingRx));
74        Serial.print("Drone HDG: ");                           //Free to comment
75        Serial.println(headingRx);                             //Free to comment
76        return headingRx;
77
78  }
79}
```

Under `void setup()`-delen av koden initialiseres først radioen (nRF24L01) og de nye topicene *drone_prompt* og *drone_heading*. Disse topicene abonnerer og publiserer fra *serial_node*. Etter initialiseringen formidles det til ROS-masteren at disse nye topicene er opprettet, og grensesnittet de skal interagere med.

```
35void setup() {
36  initSerial();
37  initNRF();
38
39  nh.initNode();
40  nh.advertise(drone_heading);
41  nh.subscribe(drone_prompt);
42
43}
```

Merk at radioen settes opp til å skrive til `sNode`-adressen, og lytter til `dNode`-adressen. Disse adressene ble som nevnt satt opp i linje 11.

```
56 void initNRF(){
57   radio.begin();
58   radio.openWritingPipe(addresses[0]);          //Write on sNode
59   radio.openReadingPipe(1,addresses[1]);        //Listen on dNode
60   radio.setPALevel(RF24_PA_MIN);                //Set min.TX-power
61 }
```

`spinOnce()`-funksjonen sjekker så oppdateringer på relevante topicer, her hvert sekund. Dette utgjør `void loop()`-funksjonen i koden.

```
45 void loop() {
46   nh.spinOnce();
47   delay(1000);
48 }
```

Dronekoden

Dronekoden uten biblioteker er vedlagt.

Dronekoden er den koden som håndterer mottak av den trådløse forespørselen om orientering, genereringen av denne og retur transmisjonen. Det er denne koden som programmeres på mikrokontrolleren som sitter i dronen.

Tidlig i prosjektet ble koden skrevet for å benytte seg av magnetometeret *MPU9255*. Det var fordi dette magnetometeret hadde grensesnitt for SPI-kommunikasjon, og koden blir lettere å designe. Alle komponentente (radio og magnetometer) brukte SPI fra før. Dette medførte også at første versjon av koden brukte mindre programminne, fordi det ikke var nødvendig med et ytterligere bibliotek for I^2C, `wire.h`.

Testing viste at *MPU9255* ikke ga stabile resultater, og brikken ble skrotet til fordel for *LSM303DLHC*. Dette medførte at koden måtte skrives om, og fikk en noe mer kompleks struktur. Den kompilerte maskinkoden ble også større enn initielt, men ikke så stor at det overskred minnet på *ATmega328P*-kontrolleren.

Dette kapittelet vil, som resultat av at *MPU9255* var en blindvei, kun ta for seg den siste versjonen av koden brukt i prosjektet.

Først inkluderes nødvendige biblioteker for *nRF24L01*, *LSM303DLHC*, *SPI- og I²C*-protokollen. Etter dette får magnetometeret en unik ID for I²C-kommunikasjon, og kanalene radioen skal sende og lytte på settes opp:

```
15/* Assign a unique ID to this sensor at the same time */
16Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);
```

og

```
27RF24 radio(radioEnable,csNRF); // CE, CSN
28const byte addresses[][6] = {"sNode","dNode"}; //Drone writes dNode. vice-versa
```

Etter dette gjennomføres initialiseringen. En forskjell verdt å nevne her er at funksjonen `initNRF()` setter opp skriving og lesing motsatt av hva *stasjonskoden* gjør. Slik unngår de å snakke på samme kanal. Dronen settes opp med lavest mulige sendereffekt, slik at den har en minimal rekkevidde. Dette gjør at en annen drone ikke vil være i stand til å oppgi sin egen retning istedenfor den som har landet på plattformen.

```
66void initNRF(){
67  radio.begin();
68  radio.openWritingPipe(addresses[1]);      //Write on dNode
69  radio.openReadingPipe(1,addresses[0]);    //Listen to sNode
70  radio.setPALevel(RF24_PA_MIN);            //Set min.TX-power
71}
```

Deretter starter `void loop()`-funksjonen. Mikrokontrolleren starter her å lytte på radioen. Dersom den mottar en `Boolsk` *sann*-verdi, slik den skal fra *stasjonen*, kjører den funksjonen `void respond()`.

```
39void loop() {
40  radio.startListening();
41  delay(5);
42
43  if (radio.available()) {
44    radio.read(&pendingRequest, sizeof(pendingRequest));
45    delay(10);
46  }
47  if (pendingRequest == 1){     //Reads heading and sends it
48    respond();
49  }
50} //-------End LOOP
```

Funksjonen `void respond()` vil først slå av radioen, før den leser av verdiene til magnetometeret. Disse verdiene føres så inn i ligningen beskrevet i formel 6.1, som da returnerer magnetometerets retning relativt til et magnetfelt $\pm 0{,}7°$. Dette er en oppløsning langt bedre enn hva systemet behøver, og derfor konverteres dette tallet om til en `UInt8`. Dette formatet har fortsatt en god nok oppløsning av retningen med verdier fra 0-255, tar mindre plass enn en `Float`, og egner seg bedre på tvers av systemer. Etter at dette er gjort vil funksjonen til slutt legge dette tallet ut på senderkanalen, slik at **stasjonen** har mulighet til å plukke det opp.

```
74 void respond(){
75   radio.stopListening();
76
77   sensors_event_t event;
78   mag.getEvent(&event);
79   float heading = (atan2(event.magnetic.y,event.magnetic.x) * 180) / PI;
80
81   if (heading < 0){
82     heading = 360 + heading;
83   }
84   byte sendHdg = map(heading, 0,360,0,255);
85   radio.write(&sendHdg, sizeof(sendHdg));
86   pendingRequest = 0;
87   //delay(50);
88 }
```

6.2.3 Kretskortdesign

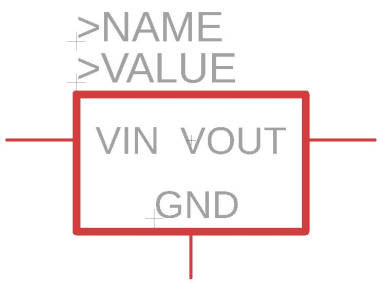
Det fremkommer av Figur 6.9 den tenkte interaksjonen mellom komponentente i dronen. Det ble besluttet å utforme et kretskort som kunne innfri denne samhandlingen mellom de forskjellige utviklingskortene som skulle bli brukt i dronen. Før den skjematiske tegningen ble utformet, ble det besluttet at batteriet ombord i dronen skulle forsyne kretskortet og de nødvendige komponentene. Dette ville gi en mer kompakt og brukervennlig løsning.

Skjematisk tegning

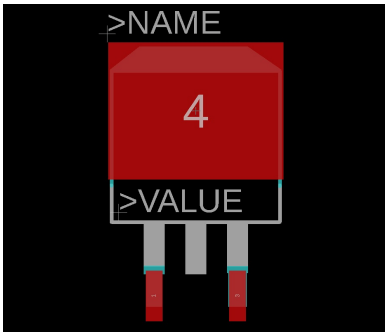
Først ble symbolene for de forskjellige komponentene fremstilt. Dette ble gjort ved å først gi dem et grafisk symbol (til den skjematiske tegningen), og et tilhørende fotavtrykk (til bruk i fremstillingen av kortet). Deretter ble pinnene på det grafiske symbolet koblet sammen med de forskjellige hull- og loddeflatene på fotavtrykket. Figur 6.12 og 6.13 er eksempler på slike tegninger til komponenten i Figur 6.11.

Etter at det var laget representative tegninger inkluderende grafiske symboler og fotavtrykk, ble det laget en skjematisk tegning som inneholdt alle komponentene. Disse komponentene

Figur 6.11: Spenningsregulatoren 7805CD2T



Figur 6.12: Symbol av 7805CD2T til bruk i skjematisk tegning



Figur 6.13: Fotavtrykk til 7805CD2T til bruk i utlegg

ble til slutt koblet sammen på tegningen på korrekt måte. Det er her verdt å merke seg at det ble tilrettelagt for at IR-dioden kunne trekke strøm rett fra kortet, forutsatt montering av en motstand som begrenser strømmen. Motstanden kan regnes ut etter formel 6.2.

$$\text{Motstand} = \frac{V_{cc} - V_{LED}}{I_{LED}} \tag{6.2}$$

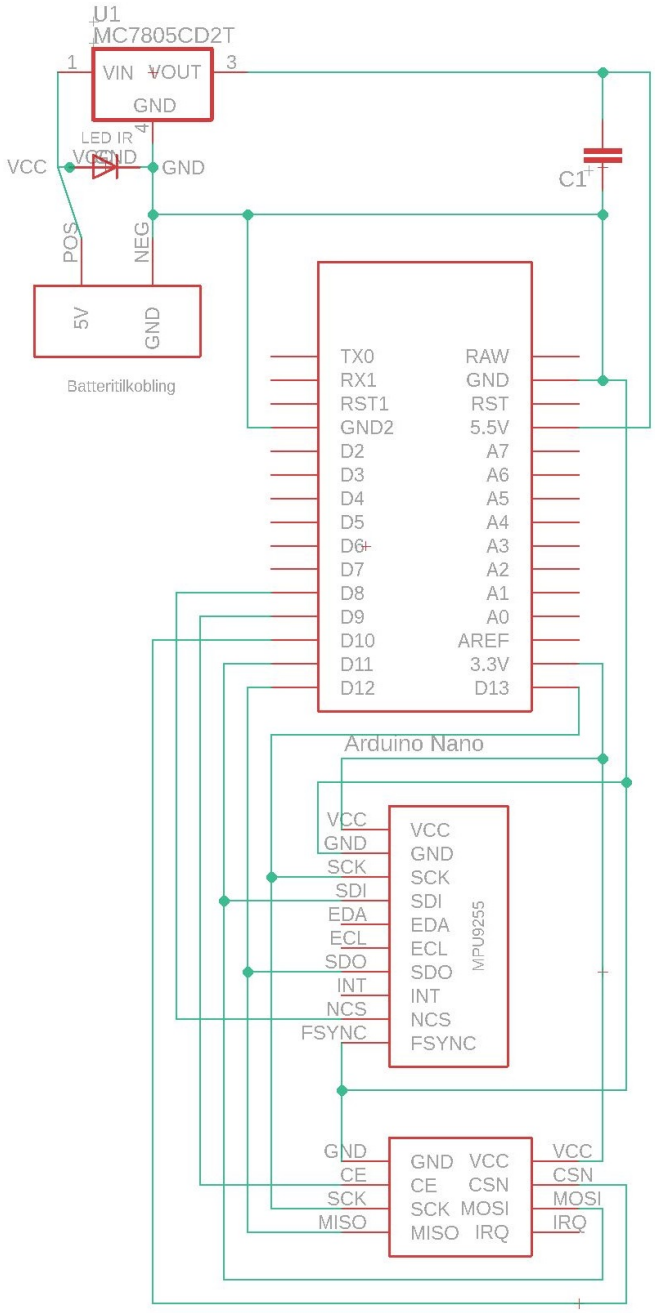
Fremstilling av kretskort

Den skjematiske tegningen ble overført til neste fase: tegningen av det endelige kortet. Kortet fikk først en fysisk dimensjon som ville passe i dronen. Deretter måtte komponentene plasseres på en slik måte at de ikke ville kollidere i hverandre. Komponenter som ville omsette mye effekt måtte også få avstand til andre komponenter for å kunne kjøles.

Etter at plassering var gjort ble de nødvendige strømlederne tegnet opp. Det er viktig at baner med stor strøm dimensjoneres for å unngå varmgang. Dette medfører unødig effekttap, og kan i verste fall være skadelig for kortet. Det finnes tabeller man kan bruke til dimensjonering, som regner banebredde som funksjon av strøm og kobbertykkelse. På grunn av kortets relativt store utforming ble banene på kortet fremstilt her overdimensjonert.

Deretter ble signalbanene mellom komponentene tegnet. Det var utfordringer knyttet til dette, da det ikke var mulig å legge samtlige baner i ett plan uten at de måtte krysse hverandre. På grunn av dette ble det benyttet hull gjennom kortet, som muliggjorde ruting av baner på andre siden også.

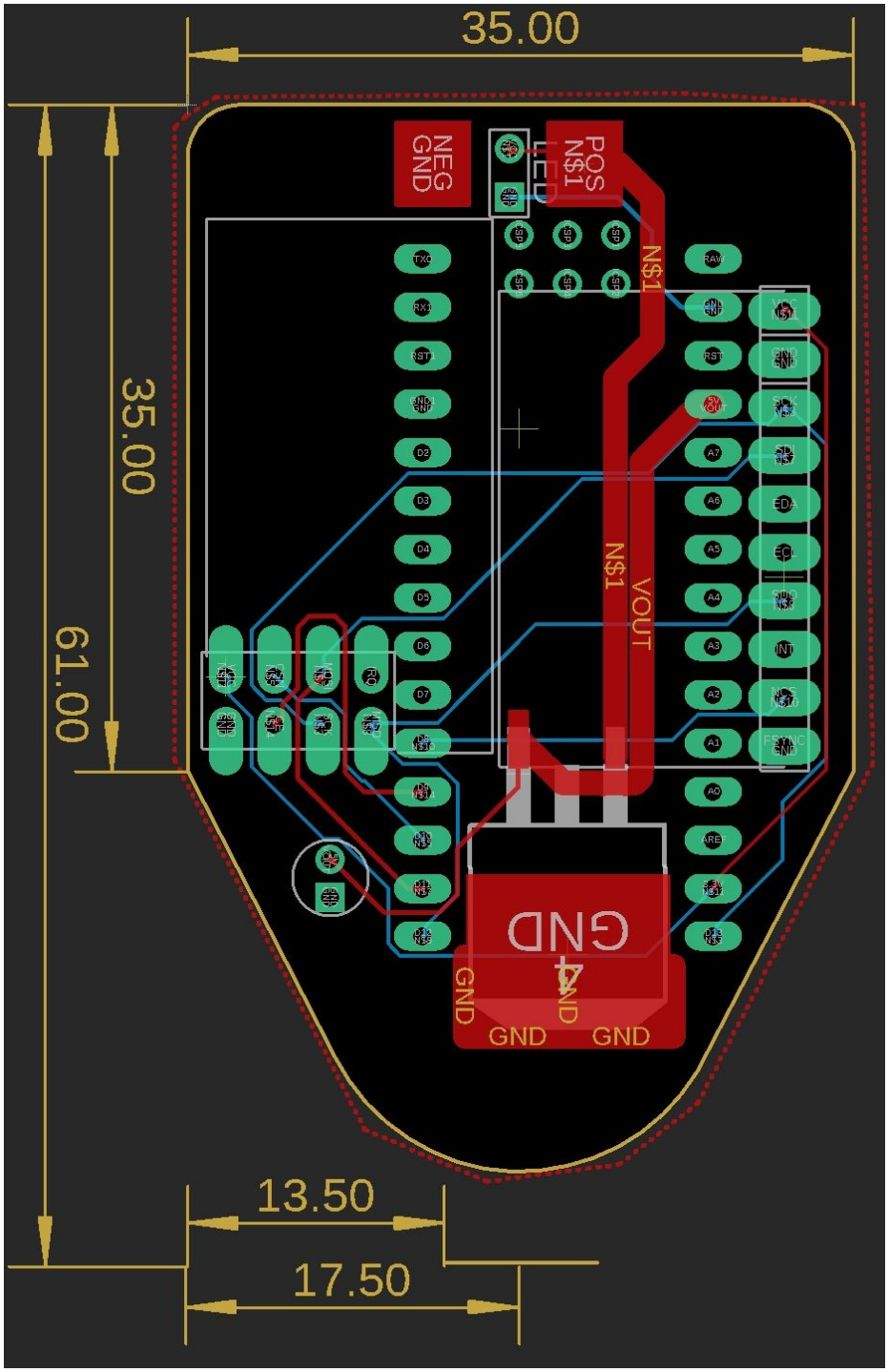
Den ferdige filen ble så eksportert til en fres.



Figur 6.14: Den fullstendige kretstegningen. Nederste komponent er av typen nRF24L01. Merk at dette er tegnet for MPU9255, og derfor ikke kompatibelt med den endelige koden som ble skrevet for dronen.

Komponent	Beskrivelse
Arduino Nano	Mikrokontroller
MPU9255	Magnetometer
nRF24L01	Radio
7805CD2T	Spenningsregulator
1μF	Avlastningskondensator
LED-IR	Strømforsyning til IR

Figur 6.15: Komponentliste



Figur 6.16: Utlegg for kretskortet til dronen

6.3 Resultat

Denne seksjonen vil redegjøre for den endelige funksjonaliteten til dronen, og det vil bli gitt en innføring i det ferdige produktets virkemåte.

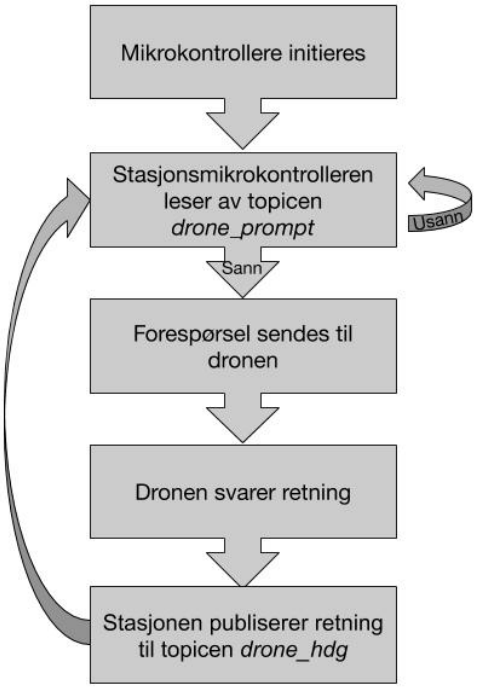
Kodene og kretskortet fremstilt i denne seksjonen kan sammen formidle dronens orientering til ROS-nettverket, ved å benytte seg av et magnetometer og trådløse radiosendere.

6.3.1 Kode

Koden produsert for hver av mikrokontrollerene fungerer nå på en måte som innfrir de krav satt av oppgaven, og i henhold til Figur 6.9. Figur 6.17 beskriver flyten i koden som er blitt produsert.

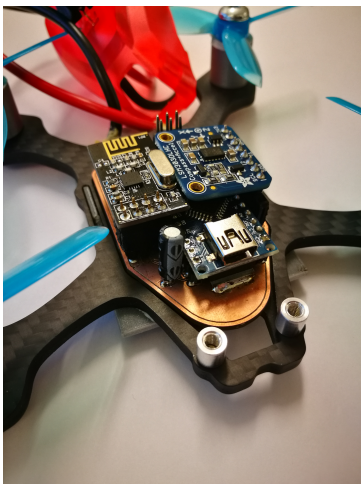
Ved å laste opp de to kodene til hver sin mikrokontroller, og koble mikrokontrolleren med **stasjonskoden** i en USB-port med forbindelse til ROS-nettverket, vil det være mulig å initiere sekvensen ved å publisere en Boolsk *sann* på topicen *drone_promt*.

Dersom dronen er innen rekkevidde når dette gjøres, vil dronen svare sin retning med en feilmargin på $\pm 0,7°$ representert som en `UInt8`. Dette er altså en verdi mellom 0-255, der 0 vil tilsvare magnetisk nord og 127 magnetisk sør. Verdien publiseres av *serial_node* til topicen *drone_hdg*.



Figur 6.17: Den resulterende kodeflyten

6.3.2 Kretskortet



Figur 6.18: Kretskortet plassert i dronen med påmonterte komponenter

Denne modulen har produsert et kretskort som er tilpasset dronens størrelse og er i stand til å finne og videreformidle dronens orientering. Det ferdig produserte kretskortet er et tosidig kretskort, og har blitt sammenføyd med ønskede komponentene. Kretskortet lar seg plassere i dronen, og dronens batteri er via kretskortets spenningsregulator, i stand til å forsyne ønsket strøm og spenning til komponentene.

Det er nødvendig å nevne at det ferdige kortet er designet for å benytte seg av et magnetometer av typen MPU9255. Magne-

tometerene vi har av denne typen har vist seg å ikke fungere, slik redegjort i kapittel 6.4.1. Vi er i stedet for blitt nødt til å benytte oss av et magnetometer av typen LSM303DLHC.

Grunnet forskjeller i pinneoppsettet til denne komponenten er ikke LSM303DLHC kompatibelt med kretskortet som ble fremstilt. Det resulterende kretskortet benyttes nå kun til å forsyne komponentene med strøm. Magnetometeret LSM303DLHC er nå tilkoblet mikrokontrolleren via ledere utenom kretskortet, og er således fortsatt i stand til å redegjøre for retningen. Disse lederen er ikke synlig på figur 6.18.

6.4 Drøfting

I dette kapittelet vil det bli fokusert på de løsninger som har vært gjort i tilknytning til dronen. Funksjonaliteten og samspillet mellom de forskjellige elementene vil bli drøftet.

Fordi det ikke foreligger en spesifikk drone oppgaven skal forholde seg til, har løsningen blitt mer konseptuell. Løsningen er derfor ment å tolkes som mer veiledende, enn å sette spesifikke krav til det fremtidige systemet.

6.4.1 Løsningen

Intensjonen med løsningen implementert på dronen er å få denne til å formidle sin orientering til ROS-nettverket. Det er ikke til å legge skjul på at løsningen som ble valgt ikke har vært den letteste å implementere, og dette er utvilsomt den største ulempen forbundet med denne løsningen. Derimot mener gruppen dette er den mest relevante løsningen for oppdragsgiver, sett systemet i en helhet, av flere grunner.

For det første mener gruppen det er rimelig å anta at en autonom drone er nødt til å ha navigasjonsinstrumenter for å kunne returnere til en ladestasjon. Derfor vurderes det dit hen at et eller flere av disse navigasjonsinstrumentene må kunne redegjøre for dronens retning. For det andre mener gruppen at det er en forutsetning for systemet i sin helhet at det er mulig å kommunisere direkte med dronen. Foruten dette ville det ikke vært mulig å gi den instrukser eller hente ut sensorinformasjon.

Fordi dronen både må kunne gjøre rede for sin orientering og kunne kommunisere med andre enheter, mener gruppen at løsningen som er valgt her vil være best egnet for å implementere i sluttproduktet.

Kommunikasjonen denne løsningen tilrettelegger for ved bruk av nRF24L01-radioen gjør det svært enkelt å implementere flere sikkerhetfunksjoner i sluttproduktet. Eksempelvis vil

dronen kunne, utover det å sende sin orientering, orientere batteribyttemaskinen om at landingen var velykket og at det vil være trygt å gripe den, formidle status på interne komponenter eller identifisere seg selv blant et større antall droner.

Slike funksjoner har ikke blitt implementert i sluttproduktet, fordi nødvendige kontrollere som styreenhet (eng. *flight controller*) og motorstyrere (eng. *Electronic Speed Control*) ikke har vært tilgjengelig. Gruppen har vurdert at det å implementere dette i dronen, har vært utenfor oppgavens omfang. Sluttproduktet er heller bygget for å lett kunne implementere dette i fremtiden.

Det kan argumenteres for at det er potensielt sårbart å basere seg på å innhente retning ved hjelp av trådløs kommunikasjon. Manglende innhenting av retning blir dog en triviell problematikk sett opp mot de problemene som følger av å miste kommunikasjon med en drone som flyr. Gruppen vurderer derfor dette som robust nok til å kunne implementeres.

6.4.2 Mangelfull støtte for ROS-serial

Til tross for at ROS-services ikke er godt støttet av ROS-serial mener gruppen de klarte å løse problemet på en god måte, slik beskrevet i kapittel 6.2.2. God forståelse av ROS gjorde det mulig å jobbe seg rundt et problem snarere enn å bli hindret av det. Gruppen mener en forutsetning for at dette var mulig var det omfattende arbeidet som hadde blitt brukt på å sette seg inn i ROS før arbeidet med å bygge systemet begynte.

6.4.3 Magnetometeret

Den største utfordringen gruppen møtte ved implementeringen av løsningen de valgte, var det defekte MPU9255-magnetometeret. Grunnet den lange leveringstiden fra siden det ble bestilt fra, oppdrev vi i mellomtiden et magnetometer fra Ascend NTNU. Disse hadde en brukt chip av den typen vi bestilte, og de lot oss låne denne frem til vi mottok vår egen.

Magnetometeret vi fikk låne klarte vi derimot ikke å hente ut forståelige resultater fra. Det var lite støtte for denne brikken tilgjengelig på Internett, men det lille vi fant tilsa at vi brukte den korrekt og at vi burde ha fått riktige verdier ut. Problemet var at retningen vi fikk ut ikke samsvarte med den virkelige retningen, og at det tilsynelatende ikke var noen proporsjonale forhold vi kunne forholde oss til. Eksempelvis ga en 30° dreining av magnetometeret et utslag på alt fra 10°- 120° avlest. Etter nærmere feilsøking klarte vi se at noen av de magnetiske målingene aldri ble negative, hvilket ikke skal være mulig basert på et magnetometers

virkemåte. Vi antok at magnetometeret vi hadde fått låne av Ascend NTNU var ødelagt, og ventet på vårt eget.

Fordi vi klarte å kommunisere med det første magnetometeret hadde vi fått påvist at kretstegningen var korrekt, og vi designet og fremstilte et kretskort som passet til magnetometeret av typen MPU9255.

Da kretskortet vi hadde bestilt omsider ankom, var ikke dette magnetometeret heller i stand til å gi oss fornuftige verdier. Vi opplevde symptomer tilsvarende det vi hadde på det andre, men simultane avlesninger av magnetfeltene (med kortene i identisk orientering) ga oss ikke identiske data. Dette forstod vi svært lite av, og vi var ikke i stand til å oppdrive svar som ga mening i litteraturen. På bakgrunn av dette ble det besluttet å gå til innkjøp av et LSM303DLHC.

Da LSM303DLHC ble levert, sent i prosjektet, fungerte funksjonene vi hadde skrevet uten problemer. Vi var nå i stand til å utføre kompassavlesninger, men var dessverre ikke i stand til å plassere det på kretskortet vi hadde designet. Kortet demonstrerer i dag konseptet på en god måte, men det hadde vært givende å få brukt hele kretskortet vår i det endelige produktet. Prosjekttiden er den begrensende faktoren for at det ikke har blitt utarbeidet et nytt kort som ivaretar LSM303DLHC.

Alt i alt ble det brukt svært mye ressurser på å få magnetometeret til å fungere. Det defekte magnetometeret ble testet forskjellige steder, både ute- og innendørs. Det ble også koblet opp på forskjellige måter, for å i størst mulig grad forhindre at ledere i nærheten induserte magnetfelter. De samme testene ble kjørt på begge magnetometrene av typen MPU9255. Til sammen har magnetometerene vi ikke klarte å bruke lagt beslag på flere arbeidsdager.

6.4.4 Emulering av dronen

Mens gruppen ventet på det siste magnetometeret ble *Python*-koden *drone_emu* skrevet. Denne koden emulerte de funksjonene det var tiltenkt å gi dronen, så det videre arbeidet som måtte gjøres på systemet kunne fortsette. *drone_emu*, i likehet med *Stasjonskoden*, lytter til topicen *drone_prompt* og publisererer til *drone_hdg*. `UInt8`-verdien som publiseres blir dog tastet inn fra brukeren. Koden er ikke i bruk i det endelige resultatet, men var av vesentlig betydning for resten av arbeidet som ble gjort i perioden gruppen ventet på LSM303DLHC. Vi tror genereringen av denne spilte en avgjørende rolle for å muliggjøre sammensyingen av systemet i sin helhet.

6.4.5 Oppløsning og responstid

Løsningen som nå foreligger har en mer enn god nok oppløsning på orienteringen med sine 1,4°/bit. Responstiden på dronens avlesning av retning utgjør ingen tidsmessig flaskehals i systemet. Innhenting av orienteringen tar under ett sekund, og kan uansett kjøres i paralell med fysiske flytt av armen.

6.4.6 Kretskortet

Kretskortet har hatt som formål å sammenføye de nødvendige komponentene i dronen. Kretskortet har til dels vært i stand til å gjøre dette, grunnet bytte til andre komponenter som følge av defekter. Kretskortet brukes fortsatt i dronen, blant annet til å tilføre IR-dioden strøm, men bistår i mindre grad enn først tiltenkt i avlesningen av magnetometeret.

Til tross for å være designet for et annet magnetometer, har kretskortet til en viss grad latt seg modifisere til det nye magnetometeret. En erfaring vi har gjort oss her er at det kan være fordelaktig å legge ut lodde-puter (eng. *solder pads*) og hull selv for pinner man ikke regner med å bruke. Dette gir stor økning i fleksibilitet og anvendelighet veid opp mot liten ekstra innsats under designprosessen.

Ettersom det originale kretskortet lot seg modifisere ble det ikke fremstilt noe nytt kretskort. Dette fordi det første kortet nå innfrir sin hensikt, og fordi gruppen vurderer det dit at det uansett ikke vil utgjøre en endelig komponent i sluttproduktet. Designet av kretskortet i denne oppgaven har som nevnt vært i den hensikt å få sammenføyd de nødvendige utviklingskortene til demonstrasjon av vår løsning, og ikke som bruk i et tenkt sluttprodukt.

I retrospekt ville fortsatt gruppen ha designet et kretskort, grunnet dets nødvendighet for å demonstrere resultatet, men benyttet seg av mer omfattende prototyping i forkant av fremstillingen for å forhindre feilprodusering.

6.5 Alternative løsningsmetoder

6.5.1 Alternativer til kretskort

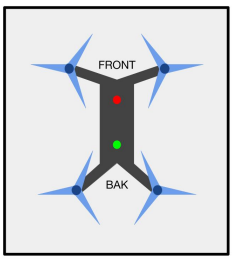
Det ble lenge vurdert å ikke designe eget kretskort, til fordel for å benytte egenbygde utviklingskort (eng. *bread board*). Dette viste seg å ikke være mulig, grunnet dronens begrensede plass, og det ble derfor nødvendig å fremstille et egetdesignet kretskort. Det foreligger ingen realiserbare alternativer til denne problemstillingen foruten å velge et orienteringsverktøy som ikke vil kreve elektronisk inngrep i dronen.

6.5.2 Alternative måter å finne orientering

Det finnes en rekke alternative løsninger for å fremskaffe dronens orientering. Dette delkapittelet vil redegjøre for noen metoder som benytter dronens utforming sammen med datasyn for å finne orienteringen. En modifikasjon som er mer generell, og som gruppen ikke mener trenger en omfattende redegjørelse, vil være å benytte seg av refleksive flater i steden for lyskilder, i de tilfellene det er aktuelt. Disse to alternativene ansees å ha lik effekt i konseptuelle utforminger.

Benytte 2 dioder

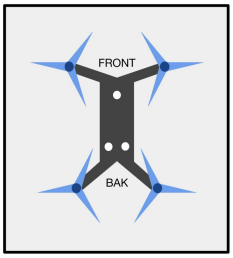
En alternativ løsning vil være å benytte to dioder for å finne retningen, slik vist i figur 6.19. Ved bruk av et kamera vil dronen kunne lese av retningen til dronen ved hjelp av de to punktene som da befinner seg i planet. En slik strek vil dog gi en feilmargin på $\pm 180°$, hvilket vil utlede et behov for å plukke opp dette. Et alternativ kan være å benytte seg av ulike farger på diodene for å skille mellom dem. Å kunne skille forskjellige farger fra hverandre kan utgjøre et problem. Denne løsningen bruker også et større område enn den valgte løsningen gjør, og er utsatt dersom en diode skulle svikte.



Figur 6.19: To dioder for å beregne orientering.

Benytte 3 dioder

Et annet alternativ vil være å benytte 3 dioder i en form som gjør den asymmetrisk over minimum 1 akse, slik figur 6.20 viser. Ved å benytte et mønster med eksempelvis 2 dioder på den ene siden og en i den andre, vil datasyn kunne gjenkjenne denne formen, og derav vite retningen på objektet den er plassert på. En ulempe med dette er det større arealet man trenger på dronen. Systemet er også sårbart dersom en eller flere dioder skulle svikte.

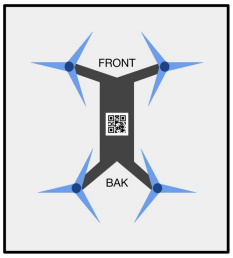


Figur 6.20: Tre dioder for å beregne orientering.

QR-kode

Et noe lignende alternativ vil være å benytte seg av et annet grafisk gjennkjenningsmerke. En QR-kode vil kunne utformes på en slike måte at det kan indikere droneorienteringen. Grunnet de mange utformingskombinasjonene QR-koden kan ha, vil en slik kode også kunne tillegges informasjon om objektet den er festet på. Dette er også en helt passiv løsning, som vil fungere uavhengig av om dronen er funksjonell.

En ulempe med denne løsningen vil være at den krever et større dedikert område på dronen, slik figur 6.21 viser. Det må vurderes hvorvidt systemet kan bli sårbart over tid, da operasjonsmiljøet kan skitne til QR-koden og derav svekke funksjonaliteten.

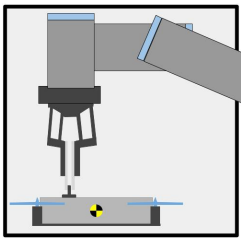


Figur 6.21: Benytte QR-kode for å beregne orientering.

Benytte tilbakemeldinger fra robotarmen

En måte å bestemme orienteringen av dronen i griperen på vil være å benytte seg av vekt-informasjon armen kan gi. Ved å fatte dronen utenfor tyngdepunktet er det mulig for armen å beregne hvor tyngdepunktet ligger relativt til griperen. En løsning som dette vil ikke kreve noen inngrep på dronen.

Dog er en forutsetning for å klare dette er at den endelige utformingen, inklusivt vekten, på dronen er kjent. Dronen må også veies under forholdsvis kontrollerte omgivelser, så eksempelvis kraftig vind vil kunne påvirke utfallet i tilfeller der ikke dette er tilstrekkelig skjermet for.



Figur 6.22: Benytte vekt for å finne orientering.

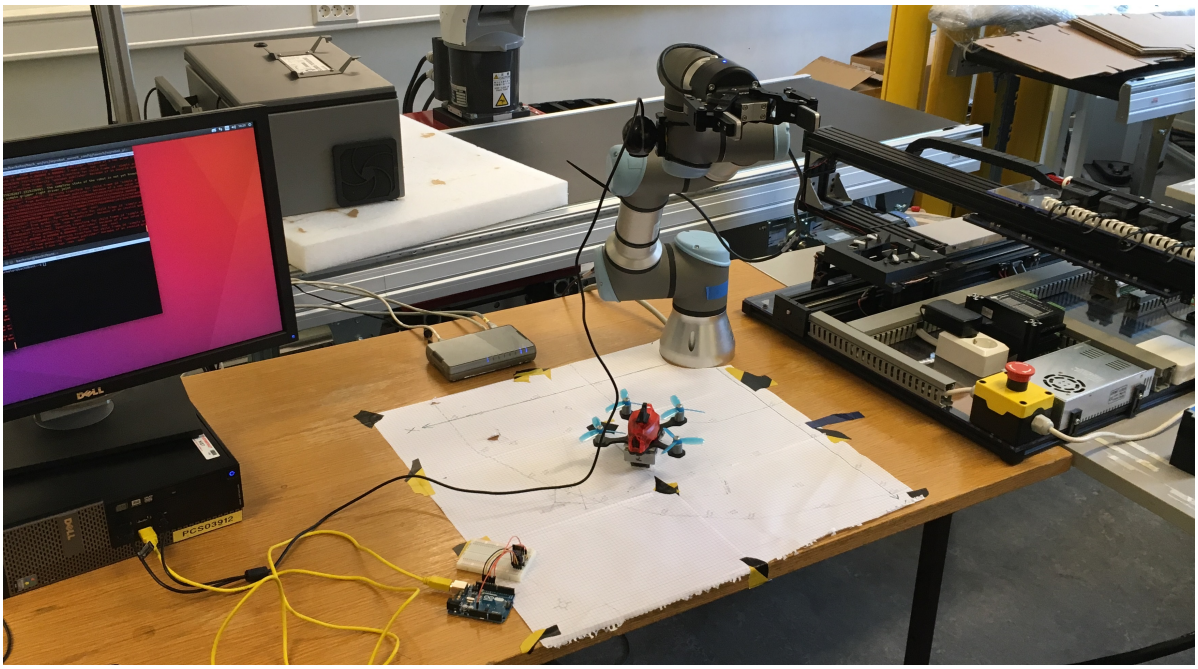
Visuell gjenkjenning

En annen måte som ikke krever inngrep på dronen vil være å bruke datasyn til å sammenligne dronens positur med en datamodell. En løsning som dette krever at dronens design er kjent, og at gjenkjenningsalgoritmen kan ta høyde for at eksempelvis propeller stanser i forskjellige stillinger fra gang til gang.

7 Resultat

Dette kapittelet vil ta for seg det overordnede resultatet av prosjektet. Ved å sy sammen de tre modulene produsert får vi en løsning som er i stand til å detektere en drone visuelt, ta hensyn til orienteringen når den plukkes opp og plassere den i en batteribyttemaskin. Vi vil først gå gjennom den fysiske sammensettningen av komponentene, for så å se på det resulterende ROS-nettverket. Til slutt vil systemets handlingsmønster bli presentert.

7.1 Konstruksjon



Figur 7.1: Konstruksjonen med robotarmen, kamera, drone og batteribyttemaskin

De forskjellige elementene er blitt satt sammen på følgende måte (se figur 7.1):

Robotarmen er plassert på et fast punkt relativt til batteribyttemaskinen, og har blitt koblet til en datamaskin via en ethernettkabel for å kunne kommunisere med ROS-nettverket.

Kameraet er koblet til samme datamaskin via en USB-kabel, og lar seg feste og løsne fra kamerafestet som er montert på robotarmen. For å unngå kabelbrudd festes dette kameraet når armen er i bildetagningspositur, og fjernes før armen flyttes videre.

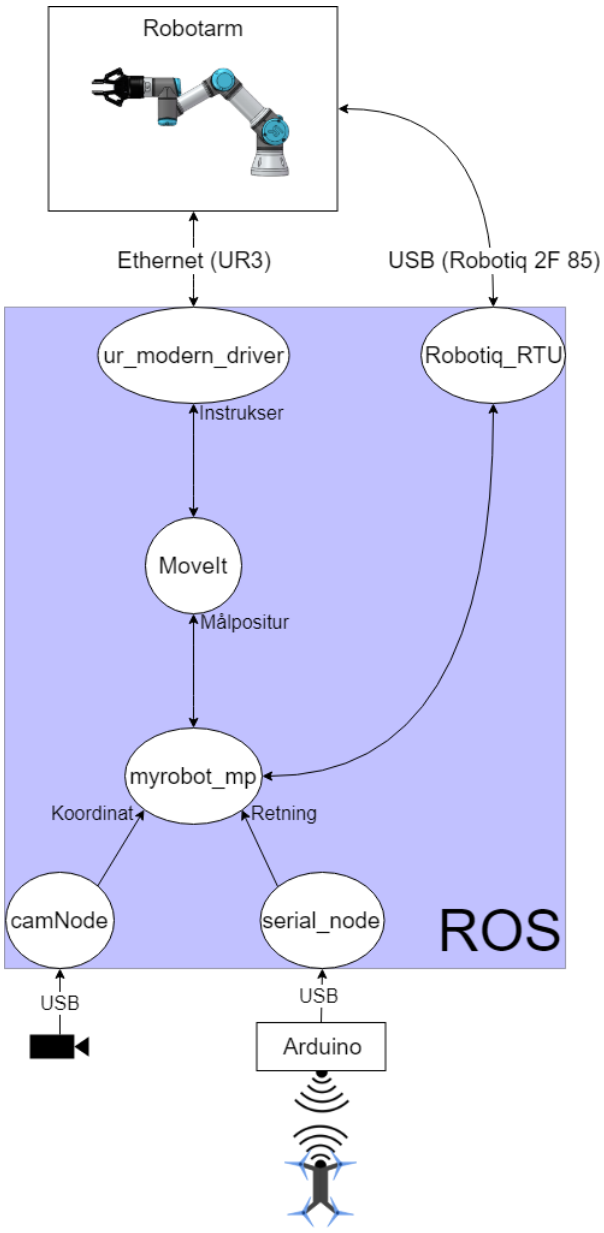
Griperen er festet på endeleddet til robotarmen. I likhet med kameraet er griperen også tilknyttet ROS-nettverket via en USB-kabel som går til datamaskinen.

Det effektive landingsområdet utgjør det området av griperens rekkevidde kameraet ser fra bildetagningspositur. Dronen plasseres i dette området før den skal plukkes opp.

Dronen har fått installert gripefinnen og det designede kretskortet med tilhørende elektronikk. Dronen kommuniserer trådløst til en mikrokontroller som er tilkoblet ROS-nettverket ved hjelp av en USB-kabel.

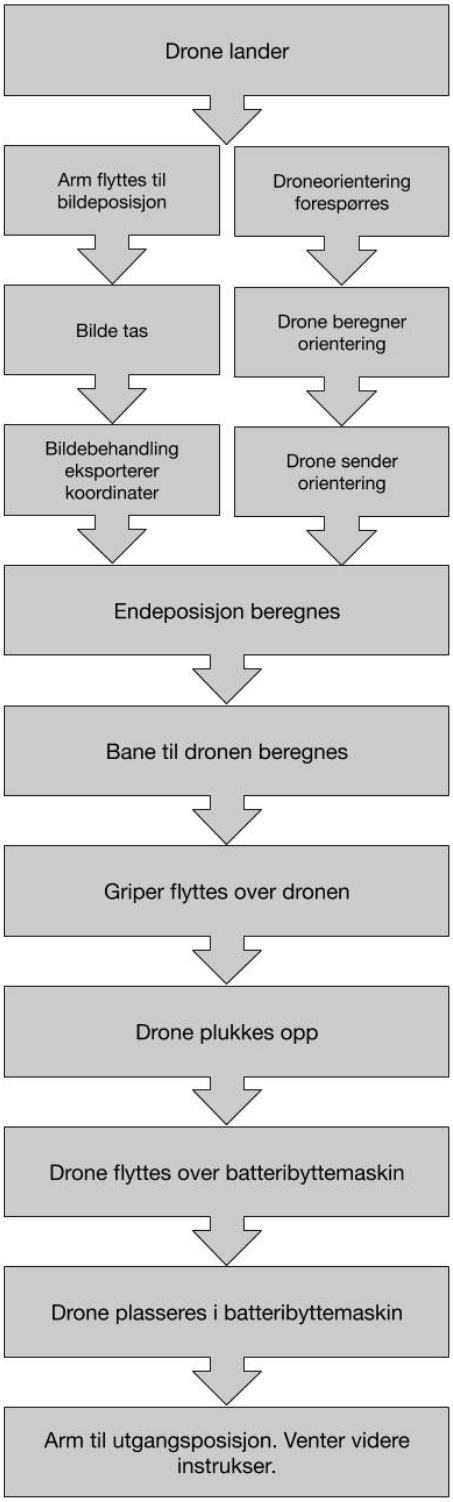
7.2 Nodeskjema

Det resulterende systemet, se figur 7.2, består av 6 ROS-noder. Drivverket i systemet er noden *myrobot_mp*. Kildekoden, *myrobot_mp.py*, er vedlagt i sin helhet. Denne noden tar inn koordinatet beregnet av kameranoden, *camNode*, og orienteringen sendt fra dronen, via *serial_node*. MoveIt får tilsendt målpositurer og regner seg fram til en gyldig bevegelsesbane mellom nåværende positur og målposituren. Det er MoveIt som sender bevegelsesinstruksen til driverpakken som styrer robotarmen over Ethernet. MoveIt kommuniserer tilbake, slik at Myrobot_mp venter til MoveIt har fullført bevegelsen. Myrobot_mp kommuniserer også med *Robotiq_RTU*, som styrer griperen via USB, og sender tilbake informasjon om griperens status.



Figur 7.2: Systemdiagram

7.3 Handlingsmønster



Figur 7.3: Komplett handlingsmønster for systemet

Sekvensen starter med at alle nodene initieres i ROS. Det videre handlingsmønsteret er beskrevet i figur 7.3.

Det første som skjer etter man starter *myrobot_mp* er at armen beveger seg til posisjonen for bildetagning, samtidig som den starter sekvensen beskrevet i kapittel 6.2.2 for å innhente dronens orientering. Programmet vil vente til kameranoden har detektert IR-LED-en, og har fått en retning fra dronen.

Den mottatte lokasjonen og orienteringen brukes til å definere et punkt 40 mm over dronen som armen beveges til. Dette gjøres for å unngå kollisjon med dronens finne i det armen beveger seg i posisjon for å gripe dronen.

Når griperen har blitt senket og lukket om gripefinnen, planlegges banen til batteribyttemaskinen. Denne bevegelsesbanen planlegges med parametre som unngår kollisjon mellom dronen og omgivelsene. På samme måte som før, beveger armen seg først over batteribyttemaskinen slik at den endelige bevegelsen blir vertikal.

Når dronen har blitt plassert i batteribyttemaskinen og griperen har åpnet seg flyttes armen tilbake til utgangsposisjonen.

Om det i løpet av programmets kjøring skjer at MoveIt ikke klarer å planlegge en gyldig bane, eller griperen ikke åpner eller lukker seg som den skal, vil programmet avbrytes.

8 Drøfting og framtidig arbeid

Drøftingen har til hensikt å vurdere det helthetlige resultatet opp mot målene det skulle innfri. Her vil det bli trukket frem styrker og svakheter knyttet til løsningssystemet som ble valgt. Dette vil deles opp i drøfting av maskinvare og programvare.

8.1 Drøfting av resultatet

Løsningen viser at konseptet fungerer, men vil trenge videre tilpassing ved en reell implementasjon. Utvikling av et fungerende system vil trenge videre arbeid både med maskinvare og programvare. Under fremtidig arbeid vil det bli redegjort for arbeid gruppen anbefaler å vurdere, med bakgrunn i erfaringer de har gjort seg under prosjektarbeidet.

8.1.1 Maskinvare

Maskinvaren er i stand til å innfri kravene satt av oppgaven; finne dronens posisjon, finne dronens orientering og forflytte denne fra landingsområdet til batteribyttemaskinen. Den løser dette problemet på en tilfredsstillende måte ved at den ikke skader dronen under forflytning.

Kabelproblematikk

Grunnet kameraets- og griperens kabel, kombinert med den tidvis intrikate baneplanleggingen, har den nåværende løsningen behov for ettersyn når den foretar flytt av armen. Dette fordi kombinasjonen av nevnte faktorer utgjør en risiko for kabelbrudd.

En midlertidig løsning på kabelbruddproblematikken vil være å installere lenger kabler, men dette løser ikke faren for at kabelen over tid vikler seg inn i armen og ryker.

En mer permanent løsning på dette vil være å enten integrere kabelen i robotarmen, eller benytte seg av trådløs kommunikasjon med griper og kamera.

Redusert praktisk landingsareal

En annen utfordring som oppstår som et resultat av maskinvaren er det reduserte området dronen kan lande på. Det smale synsfeltet til kamera, verktøyets behov for å gripe dronen vertikalt, robotarmens begrensede rekkevidde og observasjonspunktets plassering en høyde over dronen vil hver for seg bidra til å innsnevre arealet dronen kan bli plukket opp fra. Når disse kombineres blir det praktiske landingsarealet vesentlig minimert. Dette problemet kan

løses ved å utbedre et eller flere av de nevnte problemene.

Mikrokontrollere som ledd i kommunikasjonslinje

Som nevnt i kapittel 6.4.1 har gruppen vurdert det som fordelaktig å ha trådløs kommunikasjon med dronen. Måten dette ble innført på har vært tilstrekkelig, men kunne kanskje vært implementert på en måte som ikke gjorde det nødvendig å benytte seg av et ytterligere mikrokontroller for å kunne kommunisere med ROS-nettverket.

En bedre måte å gjøre dette på kunne ha vært å benytte seg av kommunikasjonsprotokoller flere datamaskiner har støtte for, som Bluetooth og Wi-Fi. Dette kunne på den ene siden ha bidratt til en mer allsidig løsning, men på den andre siden ville det ikke har skissert den konseptuelle løsningen bedre enn hva det som blir gjort nå.

Modularitet

Løsningen er bygget på en måte som tilrettelegger for at moduler og komponenter i systemet enkelt kan byttes ut. Robotarm, griper, kamera og drone kan hver for seg utføres endringer på uten at dette hemmer systemets funksjonalitet.

Dette kan bidra til at systemet kan bygges av rimelige komponenter når dette blir tilgjengelig, eller som i denne oppgaven, utføres med de verktøyene man har til rådighet.

8.1.2 Programvaredrøfting

Høyt krav til opplæring

Løsningen har krevet at gruppen har brukt mye ressurser på å lære seg en rekke nye dataferdigheter. Dette har vært tidkrevende, og gjort det vanskelig å gjennomføre tenkte løsninger på en rask og god måte.

I et arbeidsmiljø der det er ønskelig å levere et produkt innen kort tid kan dette by på utfordringer, da opplæring spiser arbeidstimer som kunne ha blitt brukt til å utvikle et produkt.

ROS

Det fungerte godt å bruke ROS som et nav i systemet. Etter en bratt læringskurve opplevde gruppen at det var relativt enkelt å sy systemet sammen i ROS, på grunn av dets intuitive oppbygning med noder, topics og ferdigbygde pakker. Fordi ROS er laget fra bunnen av for roboter finnes veldig mye av det som er relevant å bruke i et robotsystem som ferdiglagde pakker. Utførelsen av dette prosjektet hadde krevet vesentlig mer programmeringskompetanse

om ikke drivere for robotarm og griper hadde vært tilgjengelig ferdig implementert i ROS.

En annen opplevd fordel ved utvikling i ROS var at det var enkelt å finne svar på problemer underveis. Det var mange gode læringsressurser tilgjengelig på Internett, og man fant ofte at andre hadde stått overfor samme problem som en selv.

ROS fremstår som et kraftig verktøy fordi det blant annet har muligheten til å simulere hendelser ved å bygge noder som emulerer programmer eller fysiske komponenter. Dette gjør at arbeid knyttet til å bygge systemet kan fortsette selv om enkelte elementer ikke er ferdige. Noders modulære natur gjør det også lett å bygge på systemet i ettertid.

Til sist er det viktig å trekke frem at ROS har forenklet prosessen knyttet til å få forskjellige komponenter til å kommunisere med hverandre. I vår oppgave har dette bidratt til sømløs interaksjon mellom enkle mikrokontrollere og høyteknologiske robotarmer.

8.2 Tverrfaglig utbytte

Utformingen av løsningen, det utøvede arbeidet og det endelige resultatet har gitt gruppen en rekke erfaringer. Oppgaven har krevet forskjellig kunnskap innenfor fagfelt som *datateknikk*, *elektronikk*, *datasyn*, *kybernetikk*, *dataassistert konstruksjon*, *trådløs- og kabelbasert kommunikasjon* og *prosjektvirksomhet*. Gruppen har også gjort en rekke mindre fagspesifikke erfaringer, som gjerne kan sies å være mer generelle. Dette er erfaringer knyttet til *møtevirksomhet*, *prosjektledelse*, *beslutningstaking*, *gruppedynamikk* og *samarbeid over avstand*.

Erfaringene gruppemedlemmene har gjort seg, bidrar til å gjøre dem bedre rustet på å løse prosjektoppgaver i fremtiden, og gir dem et fortrinn i fremtidig arbeid kyttet til disse fagfeltene.

8.3 Fremtidig arbeid

Dette kapittelet tar for seg fremtidig arbeid som kan gjøres for å tillegge funksjoner eller øke kvaliteten på produktet og systemet gruppen har jobbet på. Gruppen skiller på arbeid som kan gjøres på dronen og på servicestasjonen.

8.3.1 Forslag til framtidig drone

Systemoppbygning

Gruppen har flere forslag som burde vurderes før den endelige fremstillingen av en drone til prosjektet. Den viktigste er at til tross for at dronen kun skal utgjøre en del av et større system, må den få en svært sentral posisjon i utarbeidelsen av en endelig løsning. Gruppen har erfart at implementeringen av en servicearm gir stor fleksibilitet, og anbefaler derfor at systemet i sin helhet designes nedenfra-og-opp med utgangspunkt i dronen.

Grensesnitt for fysisk håndtering

Dronen burde utformes på en måte som gjør den robust nok til å tåle gjentakende håndtering av en servicearm. Dronen burde inneha punkter eller områder med tilstrekkelig strukturell styrke til å muliggjøre slik håndtering. De nevnte områdene bør integreres i skroget eller andre komponenter på en fordelaktig måte. Eksempelvis kan antenner, sensorer eller andre ekstremiteter styrkes på en måte som øker deres bruksområdet. Å gjøre slike områder selvrettende slik det var gjort med gripefinnen i oppgaven vurderes som en fordel, fordi det gjør sluttproduktet mindre komplekst.

Trådløs kommunikasjon

Videre har gruppen, som nevnt i kapittel 6.4.1 identifisert flere fordeler ved å implementere trådløs kommunikasjon i dronen. Halvdupleks-kommunikasjon mellom dronen og resten av systemet, i vårt tilfelle servicearmen, medfører en vesentlig forenkling systemets kompleksitet. Det er svært rimelig å anta at kommunikasjon med dronen vil kunne tillegge det endelige systemet en lang rekke nyttige funksjoner også.

8.3.2 Forslag til framtidig servicestasjon

Forbedre gjeldende kameraløsning

Det kan med fordel utforskes bedre måter å etablere kommunikasjon mellom kameraet på servicearmen og ROS-nettverket, da nåværende løsning utgjør fare kabelbrudd. Gruppen vil oppfordre til å benytte seg av trådløs kommunikasjon, eller å rute kabelen på innsiden av armen.

Trådløs kommunikasjon

Gruppen henviser til kapittel 8.3.1 og presiserer igjen anbefalingen om å tilrettelegge for trådløs kommunikasjon med dronen.

Integrering av kamera i servicefunksjoner

I en framtidig servicestasjon vil en visuell overvåkning være svært nyttig. Kameraet kan brukes svært allsidig, og kan tillegges flere oppgaver enn kun deteksjon av dronen. Det monterte kameraet kan brukes til visuell inspeksjon av dronen, ettersom det er montert på en robotarm med muligheter for å endre positur. Kameraet kan også brukes til visuell inspeksjon av landingsområdet, samt observasjon av nærområder.

Redusere arealmessig fotavtrykk

Videre anbefaler gruppen at servicestasjonen i sin helhet bygges så kompakt som mulig for gi flest mulig realistiske bruksområder. Dette kan oppnås ved at servicemodulen benytter seg bedre av høyden enn det den nåværende versjonen gjør. Eksempelvis kan selve batteribyttet utføres under landingsområdet.

Bistå landinsprosessen

Ved å etterstrebe et så lite arealmessig fotavtrykk som mulig kan det oppstå problemer tilknyttet å lande små droner nøyaktig nok. Det bør vurderes hvorvidt servicestasjonen kan tillegges funksjonalitet som bistår denne landingsprosessen. Gruppen anbefaler å se nærmere på bevegelsesopptaksteknologi (eng. *motion capture*) i forbindelse med dette.

Redusere kollisjonsfarer

For å trygge landing på et arealmessig avgrenset området anbefaler gruppen å vurdere en løsning som tillater fjerning eller skjuling av servicearm under landing, slik at kollisjonssannsynlighet reduseres.

Økt utvalg av verktøy

For å i best mulig grad kunne fungere som en servicestasjon burde den endelige maskinen evne å benytte seg av forskjellige verktøy. Et godt utvalg av verktøy gjør det mulig at flere operasjoner kan utføres som første linjes vedlikedhold. Dette vil kunne gi økt operativ evne i sluttproduktet.

Økt interoperabilitet

Stasjonen kan med fordel designes på en måte som gjør den anvendbare på forskjellige droner. Dette vil også kunne stimulere foroverkompatibilitet for systemet.

9 Refleksjon

Dette kapittelet er mer knyttet til prosessen enn resultatet. Kapittelet går nærmere inn på refleksjoner i etikken knyttet til prosjektet, gruppearbeidet, arbeidsflyt og erfaringer som er gjort.

9.1 Etiske refleksjoner

Det er viktig å ha et bevisst forhold til etiske spørsmål knyttet til arbeidet man gjør eller sluttproduktet man fremstiller. Sluttproduktet må i sin helhet underbygge etiske prinsipper, holdinger og verdier, og arbeidet som gjøres for å nå dette må heller ikke utføres på en måtes som trosser dette.

Eksempelvis kan det kreves at arbeidet skal underbygge positive utviklinger som ivaretar samfunn, mennesker og miljø. I tilfeller der etiske retningslinjer kan være motstridende eller ute av stand til å la seg etterfølge, er det viktig å benytte seg av etiske verktøy for å treffe de riktige beslutningene.

I vårt tilfelle er det viktig å gjøre en etisk vurdering av produktet som skal leveres. Hva *skal* produktet brukes til? Hva *kan* produktet brukes til? Hva kan arbeidet gjort her brukes til i fremtiden?

Produktet utviklet i denne bacheloroppgaven er en brikke i et større system som skal drive autonome sensorfunksjoner i luftdomenet. Dette kan være alt fra innhenting av informasjon om skogbranner, våtmarksområder og smelterater på isbreer, til overvåkning av eiendom og personer. For tiden er debatten knyttet til autonom overvåkning relevant, og det er viktig å følge med på denne selv om man ikke tar et standpunkt i den.

Videre er det viktig å ta stilling til hvorvidt systemet i sin helhet har etisk hjemmel. Intensjonen er å benytte seg av droner lettere enn 250 g, fordi det pr. dags dato ikke finnes lovverk som regulerer bruken av disse på samme måte som større droner. Det må tas stilling til hvorvidt dette underbygger intensjonen til de lover og regler vi omgir oss med, eller om dette utnytter et smutthull som ikke burde eksistere.

9.2 Tanker rundt gruppearbeid og -dynamikk

Majoriteten av bachelorgruppene på NTNU ble satt sammen av 4 studenter, gjerne fra samme fakultet. I dette tilfellet bestod gruppen av studenter fra to ulike universitetsområder, Gjøvik og Trondheim. Denne fordelingen hadde potensialet til å bli problematisk, men har grunnet god kommunikasjon vist seg å være veldig overkommelig.

Fordi samtlige gruppemedlemmer har vært tilknyttet NTNU har det latt seg gjøre å koble seg på hverandres datamaskiner via nettverk. Til tross for at robotarmen har vært plassert i Trondheim, har dette gjort det mulig styre og jobbe med den fra Gjøvik. Denne muligheten vært en forutsetning for å kunne lykkes med arbeidet.

Gruppen har jevnlig benyttet seg av Skype, Discord og andre programmer for å kunne kommunisere over avstand. Dette har bidratt til at gruppen har hatt en bedre felles forståelse av arbeidet som er gjort underveis, enn hva de ville ha hatt uten slike verktøy. Gruppen har dog erfart at telefon- og videokonferanser er underlegne fysisk tilstedeværelse. Gruppen understreker hvilken fordel det er å kunne benytte seg av fysiske hjelpemidler som peking, tegning og demonstrering, fremfor å være begrenset til verbale eller visuelle hjelpemidler.

Det har vært både fordeler og ulemper tilknyttet det at gruppen har bestått av 3 studenter i motsetning til 4. Den første, innlysende, ulempen er den reduserte arbeidskraften gruppen har. Til tross for at forventningene til sluttproduktet justeres, er det ikke til å komme unna at en større gruppe ofte vil ha flere ideer og flere forkunnskaper, og derav ha et bedre grunnlag for løse oppgaver som krever dette.

Derimot er en liten fordel med å være 3 kontra 4 muligheten til lettere kunne ta demokratiske beslutninger. I arbeidet utført av gruppen har dette blitt brukt som beslutningsverktøy de gangene avgjørelser har blitt langtekkelige eller stagnert.

Under skriveperioden har det eksempelvis vært en ulempe å befinne seg på ulike plasser. Saker som ellers hadde blitt diskutert fortløpende som de dukket opp, ble noen ganger utsatt eller ikke håndtert. Dette skyldtes at terskelen for å diskutere slike saker blir høyere når det krever en telefonsamtale, i motsetning til å snu seg til sidemannen. Gruppen har erfart at det var mer omfattende enn antatt å samkjøre rapporten.

Oppsummert har gruppearbeidet i sin helhet gått bedre enn forventet. Det har vært hyppige telefonmøter, og terskelen for å ta opp personlige saker har vært lav. Dette grunnes også at studentene kjenner hverandre fra før, og gjennom tidligere arbeid har etablert gode rutiner og holdninger tilknyttet gruppearbeid.

9.3 Tanker om arbeidsflyten

Gruppen har erfart at planlegging er viktig. I tillegg til god planlegging er det også viktig at alle gruppemedlemmer har samme forståelse av planen.

I forprosjektet ble det utarbeidet en tidsplan for bacheloroppgaven. Denne omfattet både forprosjekt, hovedprosjektarbeidet og rapportskrivingen. Videre var denne planen delt opp i arbeidspakker med en antatt plan for timeverk per pakke. Deler av prosjektet har tatt mer tid enn hva som ble planlagt i pakkene.

Undersøking av løsningsmetoder tok kortere tid enn forventet, men opplæring og bruken av ROS har tatt mer tid en antatt. På de store tidsfristene har gruppen vært innenfor. Prosjektarbeidet rundt selve løsningen varte som antatt, og arbeidet med skriving har gjort det samme.

9.4 Erfaringer tilknyttet arbeidet

De første erfaringene gruppen ønsker å trekke frem er de gjort i tilknytning til operativsystemet Ubuntu 16.04. Gruppens medlemmer hadde lite til ingen erfaring med dette operativsystemet fra før, og brukte en full arbeidsuke på å sette opp maskinvare og lære seg enkel håndtering av programvaren. Gruppen erfarte at det ikke var tilstrekkelig å kjøre Ubuntu i virtuelle maskiner da de skulle bruke ROS, og måtte derfor installere dette på fysiske maskiner. Til tross for å ha et mindre intuitivt brukergrensesnitt enn mer grafiske operativsystemer som Windows og iOS, ble gruppen over tid mer komfortable med å benytte seg av et Linux-basert operativsystem. Gruppen vil i større grad vurdere Linux i fremtidig arbeid.

Gruppen har også gjort seg flere erfaringer med ROS. Den første erfaringen gruppen gjorde seg, var å forstå det omfattende arbeidet nødvendig for å tilegne seg de grunnleggende kunnskaper det var en forutsetning å ha for å kunne bruke ROS med den tiltenkte hensikten. Gruppen hadde ingen forkunnskaper, og brukte i overkant av en arbeidsuke på å lese brukermanualer og gjennomføre innføringskurs i programvaren. Det var også nødvendig med ytterligere tilegning av kunnskap under hele prosjektarbeidet. Gruppen undervurderte i forkant av arbeidet hvor mye tid de ville bruke på å lære seg ROS.

Gruppen har dog erfart nødvendigheten av grundig opplæring i ROS før arbeidet begynte. Gruppens gode forståelse av systemet før arbeidet begynte, gjorde det lettere å kommunisere godt om arbeidet som ble gjort, og resulterte i tidsbesparing og forhindret en rekke dårlige avgjørelser.

Videre har de erfart hvor fleksibelt ROS er som verktøy på tvers av plattformer. Gruppen

mener ROS kan benyttes i en rekke tenkelige scenarioer, fra enkle til mer komplekse systemer slik som i dette prosjektet. Gruppen mener ROS-kunnskapen de har opparbeidet seg under prosjektperioden kan bli svært anvendbar i fremtidig ingeniørarbeid.

En annen god erfaring vi gjorde underveis spesifikt i ROS, var det å lete etter pakker som var laget og distribuert av produsenten selv. Disse pakkene var ofte veldig gode, og ryddige forklaringer medfulgte. Ved fremtidig arbeid i ROS anbefales det veldig å benytte arbeid som kommer fra produsenten selv.

Flere av gruppens medlemmer har gjort seg erfaringer tilknyttet versjonskontrollsystemer for programvare. Til tross for å være kraftige verktøy, og i tilfeller helt nødvendige, har gruppen jobbet lite med slike programmer under sitt utdanningsløp. Gruppen mener de erfaringene de har gjort seg rundt dette har vært avgjørende for å kunne levere produktet som foreligger.

10 Konklusjon

Denne rapporten hadde som mål å løse problemstillingen knyttet til å gripe og forflytte en 250 grams drone fra et landingsområde til en batteribyttemaskin. Prosjektet utbedret en avdekket mangel på tidligere prosjekt, og har tatt FFI et steg nærmere sin visjon om en autonom dronesverm med 250 grams droner. Prosjektet har innfridd effekt- og resultatmålene som ble definert i kapittel 1.3, på en måte som tilbyr konseptuelle løsninger på utfordringene knyttet til å plassere dronen i batteribyttemaskinen.

Løsningen for denne oppgaven bruker en robotarm av typen UR3 til forflytningen av dronen. Installert på armen er en griper fra selskapet Robotiq som kan gripe en selvdesignet og -produsert gripefinne, som er festet på droneskallet. I denne gripefinnen er det plassert en IR-diode som blir detektert av et IR-kamera festet til robotarmen. Ombord i dronen er det blitt plassert nødvendige instrumenter for å detektere og transmittere dronens orientering. Systemet benytter seg av Robot Operating System (ROS) for å knytte sammen modulene som finner dronens plassering og orientering, og planlegger banen for robotarmen, slik at dronen blir flyttet fra sin vilkårlige landingsposisjon til batteribyttemaskinen.

Gruppen har brukt programvare som enten er gratis eller åpen kildekode i løsningen av dette prosjektet. Robotarm, griper og datamaskin har vært lånt av NTNU, hvilket har medført lave prosjektkostnader til tross for bruk av avansert teknologi.

Oppgavens prosessmål om å benytte seg av eksisterende kunnskap så vel som å tilnærme seg ny, har i høyeste grad blitt innfridd. En del av løsningen bygget på kjente kunnskaper fra utdanningen, mens en stor del av prosjektet har krevet omfattende kompetanseheving på flere områder innen programmering, Linux og ROS. Gruppen har i svært stor grad tilegnet seg ny kunnskap de anser som relevant for fremtidig arbeid som elektroingeniører.

Bibliografi

[1] Ubuntu. *The story of Ubuntu*. URL: https://www.ubuntu.com/about (sjekket 07.05.2019).

[2] ROS.org. *Installation*. URL: http://wiki.ros.org/Installation/ (sjekket 16.05.2019).

[3] ROS-Industrial. *Description*. URL: https://rosindustrial.org/about/description (sjekket 12.05.2019).

[4] ROS.org. *Is ROS For Me?* URL: http://www.ros.org/is-ros-for-me (sjekket 16.05.2019).

[5] ROS. *Nodes*. URL: http://wiki.ros.org/Nodes (sjekket 15.05.2019).

[6] ROS. *Topics*. URL: http://wiki.ros.org/Topics (sjekket 15.05.2019).

[7] ROS.org. *ROS Messages*. URL: http://wiki.ros.org/msg (sjekket 18.05.2019).

[8] Ken Conley. *ROS Introduction*. URL: http://wiki.ros.org/ROS/Introduction (sjekket 03.05.2019).

[9] Wikipedia. *Dataassistert konstruksjon*. URL: https://no.wikipedia.org/wiki/Dataassistert_konstruksjon (sjekket 10.05.2019).

[10] Autodesk. *Fusion 360*. URL: https://www.autodesk.com/campaigns/fusion-360-for-hobbyists (sjekket 10.05.2019).

[11] Dag Aune. *Industriell Automatisering: Robotteknikk*. 1.1. Trondheim: Institutt for teknisk kybernetikk, automatisering, 2017.

[12] M. Vidyasagar Mark W. Spong Seth Hutchinson. *Robot Dynamics and Control*. 2. utg. 2004.

[13] Jim Camillo. *Does Your Robot Need a Tool Changer*. URL: https://www.assemblymag.com/articles/93146-does-your-robot-need-a-tool-changer (sjekket 18.05.2019).

[14] Howie Choset. *Robotic Motion Planning: Configuration Space*. URL: https://www.cs.cmu.edu/~motionplanning/lecture/Chap3-Config-Space_howie.pdf (sjekket 18.05.2019).

[15] Nachi Robotics. *MZ07*. URL: http://www.nachirobotics.com/product/mz07/ (sjekket 29.04.2019).

[16] Universal Robots. *Technical details, UR3*. English. Universal Robots. 1 s.

[17] Universal Robots. *UR3*. 2019. URL: https://www.universal-robots.com/no/produkter/ur3-robot/ (sjekket 03.05.2019).

[18] Universal Robots. *Brukerveiledning UR3*. Norsk. Versjon Versjon 3.3.0. Universal Robots. 2016. 179 s.

[19] Robotiq. *Products*. 2019. URL: https://robotiq.com/products (sjekket 16.05.2019).

[20] Robotic Inc. *Robotiq 2F-85 & 2F-140 Instruction Manual*. Robotic Inc. 13. mar. 2019. 144 s. Under utgiv.

[21] Ioan A. Sucan og Sachin Chitta. *MoveIt*. URL: http://moveit.ros.org (sjekket 01.05.2019).

[22] MoveIt. *System Architecture MoveIt*. 2019. URL: moveit.ros.org/documentation/concepts/ (sjekket 01.05.2019).

[23] MoveIt. *System Architecture MoveIt*. 2019. URL: moveit.ros.org/documentation/concepts/ (sjekket 01.05.2019).

[24] Thomas Timm Andersen. *Optimizing the Universal Robots ROS driver.* Technical University of Denmark, Department of Electrical Engineering, 2015.

[25] Wikipedia. *ABS-plastikk*. URL: https://en.wikipedia.org/wiki/Acrylonitrile_butadiene_styrene (sjekket 06.05.2019).

[26] Vishay Semiconductors. *Infrared Emitting Diode, 950 nm, GaAs*. English. Versjon Versjon 1.8. Vishay. 2017. 5 s.

[27] *universal_robot*. URL: http://wiki.ros.org/universal_robot (sjekket 08.05.2019).

[28] *urdf*. URL: http://wiki.ros.org/urdf (sjekket 07.05.2019).

[29] *xacro*. URL: http://wiki.ros.org/xacro (sjekket 07.05.2019).

[30] Trygve Holtebekk. *Infrarød stråling*. URL: https://snl.no/infrar%C3%B8d_str%C3%A5ling (sjekket 03.05.2019).

[31] Inge Christ. *Lys*. URL: https://www.uis.no/getfile.php/13329205/Studietilbud/Vedlegg/Lys.pdf (sjekket 03.05.2019).

[32] J. C. Richmond A. T. Mecherikunnel. ≪Spectral Distribution of Solar Radiation≫. I: *NASA Technical Memorandum 82021* (1980), s. 5.

[33] Trust. *Trust Exis Webcam*. 2019. URL: https://www.trust.com/en/product/17003-exis-webcam-black-silver (sjekket 03.05.2019).

[34] Trust. *Trust-webcam*. URL: https://www.trust.com/en/product/17003-exis-webcam-black-silver (sjekket 26.04.2019).

[35] Elektrokit Sweden. *LED IR TSUS5402 5mm 15mW 44gr*. URL: https://www.electrokit.com/en/product/led-ir-tsus5402-5mm-15mw-44gr-2/ (sjekket 19.05.2019).

[36] Japan Photo. *Lee 100x100mm 87 Infrared Polyester*. URL: https://www.japanphoto.no/lee-100x100mm-87-infrared-polyester-5055782209399-007 (sjekket 19.05.2019).

[37] LeeFilters. *Infrared*. URL: http://www.leefilters.com/index.php/camera-directory/camera-dir-list/category/infrared (sjekket 26.04.2019).

[38] OpenCV. *About*. URL: https://opencv.org/about/ (sjekket 15.05.2019).

[39] Adam Allevato. *How to Calibrate a Monocular Camera*. URL: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration (sjekket 03.05.2019).

[40] Robyn Owens. *Camera Calibration*. URL: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT9/node2.html (sjekket 03.05.2019).

[41] Wikipedia. *Mikrokontroller*. URL: https://no.wikipedia.org/wiki/Mikrokontroller (sjekket 26.04.2019).

[42] Wikipedia. *Kommunikasjonsprotokoll*. URL: https://snl.no/kommunikasjonsprotokoll (sjekket 01.05.2019).

[43] Wikipedia. *Serial Peripheral Interface*. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface (sjekket 01.05.2019).

[44] Wikipedia. *I^2C*. URL: https://en.wikipedia.org/wiki/I2C (sjekket 01.05.2019).

[45] SparkFun. *Using EAGLE:Schematic*. URL: https://cdn.sparkfun.com/assets/7/7/8/e/b/52127573757b7f8a3e8b456c.png (sjekket 20.05.2019).

[46] Wikipedia. *ATmega328P*. URL: https://en.wikipedia.org/wiki/ATmega328 (sjekket 30.04.2019).

[47] Arduino. *Arduino IDE*. URL: https://www.arduino.cc/en/Main/Software (sjekket 07.05.2019).

[48] Autodesk. *Autodesk Eagle*. URL: https://www.autodesk.com/products/eagle/free-download (sjekket 15.05.2019).

[49] ROS.org. *ROSserial*. URL: http://wiki.ros.org/rosserial (sjekket 03.05.2019).

[50] ROS.org. *ROS services*. URL: http://wiki.ros.org/Services (sjekket 10.05.2019).

[51] ROS.org. *ROSserial service-feil*. URL: https://answers.ros.org/question/77681/serviceserver-on-arduino-service-requests-will-eventually-hang/ (sjekket 10.05.2019).

[52] ROS.org. *NodeHandles*. URL: http://wiki.ros.org/roscpp/Overview/NodeHandles (sjekket 06.05.2019).

A Forprosjekt

Forprosjekt

Per Forester
Adrian Holmeset
Hans Einar Skinnarland

Januar 2019

Sammendrag

En av de mange utfordringene knyttet til å holde en dronesverm i kontinuerlig drift er de servicebehovene som må løses uten menneskelig interaksjon. En av de utfordringene man oftest vil støte på er problematikken knyttet til et autonomt batteribytte. For å løse denne problematikken har FFI over lengre tid etterspurt flere oppgaver som har til hensikt å løse dette.

Denne oppgaven bygger på en tidligere bacheloroppgave som så på problematikken knyttet til å utføre selve batteribyttet. Den oppgaven avgrenset seg til at dronen var plassert i maskinen før batteribyttet kunne bli utført. Med bakgrunn i det har FFI i år ønske om å undersøke problematikken knyttet til å få plassert dronen i den maskinen. Hvordan skal man detektere dronens orientering? Hvordan skal man gripe dronen? Hvordan skal man flytte dronen? Kan denne innretningen brukes til noe mer enn å bare flytte dronen?

Dette forprosjektet danner arbeidsgrunnlaget for hovedprosjektet. Den inneholder bakgrunnen for oppgaven, en problemstilling med nødvendige forutsettinger og avgrensninger, prosjektmål, nødvendige spesifikasjoner og avdekkede problemområder. Videre bryter den det forestående arbeidet ned i en fremdriftsplan bestående av håndterlige arbeidspakker, med deklarering og beskrivelse av arbeidsfordeling og myndighetsforhold.

Forord

Denne rapporten omhandler forarbeidet utført av elektroingeniørstudenter i forbindelse med en bacheloroppgave på NTNU. Prosjektgruppen på tre studenter skal våren 2019 installere en innretning som skal hente og plassere droner i en allerede ferdigutviklet dronebatteribyttemaskin, som var en bacheloroppgave året før. Installeringen av denne innretningen på dronebatteribyttemaskinen er en del at et større konsept for en autonom dronesverm som utvikles ved FFI. Dronesvermen skal kunne operere over lengre tid, og for å bytte batteri skal dronene hentes fra en landingsplattform og plasseres i batteribyttemaskinen.

Prosjektgruppen er veldig glade for å få denne bacheloroppgaven som er en del av et større og spennende prosjekt hos FFI. Vi ønsker å takke FFI som gir oss denne muligheten og NTNU som stiller med utstyr og veileder. Gruppen ønsker spesielt å takke Jonas Moen på FFI og Torleif Anstensrud og Knut Wold på NTNU for veiledning og tips.

Innhold

Tabeller

1. Innledning

1.1. Bakgrunn

Forsvarets Forskningsinstitutt (FFI) er et tverrfaglig institutt underlagt Forsvarsdepartementet (FD), som gjennom samarbeid med andre ledende forskningsinstitusjoner har til hensikt å drive innovasjon innen moderne høyteknologi. Utover dette yter FFI også et betydelig bidrag til Forsvarets langtidsplanlegging gjennom analyser og utviklingsprosjekter etter Forsvarets behov.

Det er identifisert et behov for sensorsystemer som muliggjør autonom overvåking og informasjonsinnhenting av land- og maritime områder gjennom luftdomenet. I denne forbindelse er det ønskelig å anvende droner i sverm som kan operere selvstendig over lengre tid. Det er behov for utstyr som kan utføre bytte av slitasje- og forbruksmateriell, eksempelvis propeller, sensorinstrumenter eller batterier, med minimalt behov for menneskelig interaksjon.

Av *Forskrift om luftfartøy som ikke har fører ombord* fremkommer det begrensninger på dronens fysiske dimensjoner. Disse må iakttas for ikke å forringe løsningens evne til å utføre oppgaven. Av Kapittel 7, §51 Sikkerhetsavstander, maksimal flygehøyde:

> *Luftfartøy som har en MTOM på 250 gram eller mindre, kan flys VLOS, EVLOS eller BLOS, men ikke høyere enn 50 meter over bakken eller vannet.*

I praksis vil dette sette en begrensning på dronens vekt til 250g MTOM, for ikke å måtte etterkomme krav og restriksjoner satt av lovverket.

Grunnet dronens lave vekt har den en naturlig kort flyvetid. For å holde det menneskelige interraksjonsbehovet så lavt som mulig har FFI sett behovet for å utvikle et system for autonomt batteribytte på dronene. Oppgaven med å utvikle denne maskinen ble løst av en bachelorgruppe våren 2018. Denne maskinen hadde en begrensning som forutsatte at dronen allerede var plassert i maskinen før den foretok batteribyttet. Det er ikke å forvente at dronen vil være i stand til å plassere seg selv i denne maskinen når den må utføre et batteribytte. Flere faktorer, som vær og vind, flyve- og navigasjonsegenskaper, slitasje på- eller ødelagte komponenter, eller et flertall av andre uforutsette hendelser, bidrar til dette.

På bakgrunn av denne begrensningen har FFI identifisert behovet for en løsning som ivaretar de nevnte faktorene og bygger opp under *visjonen om et autonomt sensorsystem i luftdomenet*. De har derfor utarbeidet denne oppgaven for bachelorstudenter ved NTNU.

1.2. Oppgaveteksten

FFI har som et resultat av denne *visjonen om et autonomt sensorsystem i luftdomenet*, identifisert behovet for en innretning som er i stand til å pålitelig plassere dronen i batteribyttemaskinen, og som på sikt også vil være i stand til å utføre 1.linjes vedlikehold på dronen. Gruppen skal ta utgangspunkt i dronebatteribyttemaskinen som ble laget våren 2018, og ta for seg problematikken knyttet til å implementere en innretning som henter og plasserer dronen i denne.

Fordi det nåværende dronedesignet fortsatt vil være utsatt for endringer, er det ønskelig fra FFI sin side at løsningen som utvikles er så generell som praktisk mulig. Med dette menes at løsningen lar seg implementere på en lignende drone uten å måtte gjennomgå store modifikasjoner. I rapporten er det ønskelig med en analyse av flere løsningskonsepter, og en redegjørelse for den valgte løsningen.

1.3. Ordforklaring

1.3.1. Definisjoner

Autonom - selvstyrende

Datasyn - et system med sensor og prosessering av bilder som gir en datamaskin mulighet til å tolke visuell input.

Drone - et ubemannet luftfartøy

Endurance - utholdenhet oppgitt i tid

Servicearm - multifunksjonell innretning for å utføre et gitt arbeid

Praktisk landingsareal - Det området dronen må lande innenfor for å få utført et batteribytte.

1.linjes vedlikedhold - Brukervedlikedhold. Eksempelvis bytte av propeller, vaske sensorlinser etc.

1.3.2. Forkortelser

FD - Forsvarsdepartementet
FFI - Forsvarets Forskningsinstitutt
NTNU - Norges teknisk-naturvitenskapelige universitet
MTOM - Maximum Take-Off Mass
LOS - Line of Sight
VLOS - Visual Line of Sight
EVLOS - Extended Visual Line of Sight
BLOS - Beyond Line of Sight

2. Teknisk del

2.1. Problemstilling og avgrensning

Dagens situasjon er slik at det er en uoverensstemmelse mellom landingsnøyaktigheten til dagens drone og størrelsen på landingsområdet på maskinen. Dronen har ikke god nok landingsnøyaktighet til å sikre at den lander slik at batteribyttemaskinen får foretatt et batteribytte hver gang. Maskinen har heller ingen innretning som gjør det mulig å bytte eller vedlikeholde komponenter som trenger ettersyn. På bakgrunn av dette har FFI utstedt en problemstilling med følgende formulering:

> *Det skal installeres en servicearm som gjør det mulig å plassere en drone i batteribyttemaskinen. Servicearmen skal kunne fatte dronen uavhengig av dronens landingsposisjon og -orientering.*

Hovedutfordringene her ligger i det å få robotarmen til å gjenkjenne dronen, plukke denne opp uten å ødelegge den, for så å plassere denne korrekt i batteribyttemaskinen. Det gis ingen krav eller retningslinjer til valg av løsning eller utstyr.
På bakgrunn av oppgavens formulering har vi valgt følgende avgrensninger:

- Vi forholder oss til den dronen, herunder dens utforming, som foreligger pr. dags dato. Vi forbeholder oss retten til å utføre små modifikasjoner, så fremt de ikke spiller vesentlig inn på dronens flygeegenskaper, flyvetid eller sensorkapasitet.
- Vi vil ikke studere problematikken rundt servicerutiner. Dette inkluderer å studere hva som er nødvendige servicerutiner, og å kartlegge hvilke behov, arbeid- og utstyrsmessig, disse servicerutinene stiller. Vi vil dog tilstrebe å tilrettelegge for fremtidig implementering av servicerutiner.
- Gruppen vil ikke se på de faktiske kravene til størrelse dronen krever for pålitelig landing. Gruppen vil heller dimensjonere innretningen etter utstyret vi har tilgjengelig. Dette gjelder i hovedsak valget av robotarm.

2.2. Prosjektmål

Det er hensiktsmessig å skille mellom tre typer mål: effektmål, resultatmål og prosessmål.

2.2.1. Effektmål

Effektmålet for denne oppgaven vil være å komme enda nærmere FFI sin visjon om et autonomt sensorsystem bestående av en selvfungerende dronesverm, der det over en lengre periode er et minimalt behov for menneskelig interaksjon. For å oppnå dette på en god og ressurseffektiv måte er det viktig at tidligere arbeid, forskning og erfaringer tas med i arbeidet mot målet. Dette leder derfor til en oppgave som utbedrer en eller flere av de begrensninger foregående arbeid har hatt.

2.2.2. Resultatmål

Et resultatmål av denne oppgaven vil være å ha installert en innretning på den eksisterende batteribyttemakinen som gir dronen et større praktisk landingsareal. Innretningen skal være autonom, og ikke kreve menneskelig interaksjon for å utføre det nødvendige arbeidet. Målet er innfridd når en slik innretning er installert, testet og erklært fungerende. Målet skal innfris innenfor de ressursmessige rammene som settes av oppdragsgiver.

2.2.3. Prosessmål

Gruppen har som prosessmål å videreutvikle de ferdigheter og kompetanse de har opparbeidet seg i løpet av sine 2,5 år med studier, innenfor fagfeltene automasjon og elektronikk. Videre er det forventet at enkeltindividene på lik linje med gruppen i sin helhet videreutvikler sin kunnskap om gruppearbeid og samarbeid.

Gruppen, med tanke på dens geografisk store utspredning, forventes å opparbeide seg en stor erfaringsbank når det kommer til å samarbeide over lange avstander. I dette ligger en viss forventning til å løse oppgavene sammen ved å dra nytte av digitale verktøy på en oversiktlig og strukturert måte, som gir leseren god helhetlig innsikt og forståelse av besvarelsen.

Grunnet medlemmenes fremtidige arbeidssituasjon er det også et prosessmål at prosjektdeltagerne sammen utvikler kompetanse på hverandre som kan bidra til et profesjonelt forhold som tilrettelegger for godt fremtidig arbeid.

2.3. Prosjektbeskrivelse

Et forprosjekt skal utarbeides for å ha et godt grunnlag for arbeidet som skal utføres i hovedprosjektet. Forprosjektet skal i størst mulig grad hjelpe med å planlegge prosjektperioden for å få oversikt over arbeidet som skal utføres, og milepæler som skal oppnås.

Et tidlig arbeidsmål er å undersøke mulige tekniske løsninger på problemet. Deriblant hvilke mulige produkter som finnes på markedet, eller hvilke systemer som kan settes sammen for å løse problemet. Det finnes en rekke løsninger for å bruke kamera med robotarmer, noen mer proprietære og kostbare enn andre. Produkter og løsninger må kartlegges, og vurderes. Ut i fra disse mulighetene må det velges en løsning som skal implementeres i prosjektet.

Et naturlig neste steg er å implementere den valgte løsningen med robotarmen som disponeres. Herunder eventuelt å få ferdige produkter til å fungere med systemet, eller å lage en egen løsning som får mer elementære komponenter til å fungere sammen.

Etter et system for datasyn har blitt implementert med robotarmen, må vi programmere løsningen til å fungere for vårt oppdrag. Robotarmen skal kunne finne en drone innenfor en landingsplass, plukke den opp, og plassere den på batteribytteroboten, uten behov for menneskelig interaksjon.

2.4. Spesifikasjoner

Batteribyttemaskinen er dimensjonert for en drone på 250g. Videre har den et praktisk landingsområde tilsvarende dronens fotavtrykk, +/- 6mm i x og y retning. Dette gir altså en slingring på plasseringen på en drøy kvadratcentimeter.

Det er nødvendig å legge til at oppdragsgiver ikke sitter på spesifikasjoner for dronens nødvendige praktiske landingsareal. På grunn av dette vil løsningen i større grad fokusere på metode og virkemåte enn de endelige fysiske dimensjonene.

2.5. Problemområder

Prosjektet vårt er innen et fagområde ingen av oss har spisskompetanse på, og det er derfor naturlig at vi kan støte på ukjente problemer som i forkant er vanskelige å ta høyde for. Vi tror at fordi denne teknologien er ny for oss, vil det kreves omfattende undersøkelser før hvert arbeid vi foretar oss.

Dermed er vi også sårbare for risikoen å støte på manglende verktøy, utstyr, programvare

og drivere til å løse oppgaven. Fakultetet stiller med en del materiell til disposisjon, men har per dags dato blant annet ingen utstyr for datasyn.

Dette fører til et potensielt problem tilknyttet lang innkjøpstid på komponenter som vurderes kritiske. Denne lange leveringstiden er ikke utelukkende fra leverandørs side, men også fordi interne rutiner på instituttet tilknyttet anskaffelse av dyrt utstyr kan bli langtekkelige.

Problemområde	**Strategi**
Manglende kompetanse på fagområde medfører vesentlig tid- og ressursbruk	Legge inn gode tidsmarginer for undersøkelser
Manglende verktøy, utstyr, programvare og drivere for å kunne løse oppgaven.	Etablere et godt nettverk blant ansatte på instituttet som har kunnskap om utstyr og maskinparker.
Lang anskaffelsestid grunnet omstendige innkjøpsrutiner på fakultetet, eller forsinkelser hos leverandør.	Gjennomføre tidlig kartlegging av rutiner for innkjøp, samt være ute i god tid med nødvendige bestillinger.

Tabell 2.1.: Problemområder

3. Arbeidspakker

Arbeidet vi mener er nødvendig for å gjennomføre bacheloroppgaven har blitt fordelt i arbeidspakker. Arbeidspakkene inneholder en beskrivelse av arbeidsmålet og en arbeidsinstruks. Videre inneholder de informasjon om ansvar- og myndighetsforhold, arbeidsfordeling, forventet arbeidsmengde i antall timer, nøkkeldatoer og frister. Bacheloroppgaven har blitt inndelt i følgende arbeidspakker:

- Forprosjekt

  Levering av en forprosjektrapport som en forstudie av hovedprosjektet.
- Undersøke datasyn.

  Finne den beste løsningen for datasyn til servicearmen
- Implementere datasyn.

  Implementere datasyn på servicearmen
- Gripemekanisme.

  Gjøre det mulig for servicearmen å gripe tak i dronen for å løfte den.
- Programmere og teste.

  Gjennom datasynet vil servicearmen lokalisere, plukke opp og plassere dronen på korrekt måte i batteribyttemaskinen.
- Hovedprosjektrapport

  Skrive og ferdigstille hovedprosjektrapporten
- Muntlig presentasjon

  Forbrede og gjennomføre muntlig presentasjon av hovedprosjektet i Trondheim og Gjøvik.
- Administrativt arbeid

  Administrere arbeid nødvendig for å gjennomføre bacheloroppgaven.

Samtlige arbeidspakker er vedlagt i denne rapporten.

4. Prosjektorganisering

4.1. Prosjektdeltagere

Forester, Per

Alder: 25
Tlf: 993 88 804
E-post: per.forester@gmail.com

Bakgrunn:

Per gikk studiespesialiserende med realfag på videregående. Etter endt videregående utdanning (2012) tok han befalsutdanning i Forsvaret, og jobbet deretter 2 år i basetroppen på Bodø Hovedflystasjon, først som instruktør og deretter NK-tropp. I 2015 startet han på et 4-årig utdanningsløp ved Luftkrigsskolen, der første året ble avholdt på nevnte skole, mens de tre påfølgende benyttes til å ta en Bachelorgrad i elektronikk ved Norges teknisk-naturvitenskapelige universitet, Trondheim. Etter endt utdanning vil han fortsette sin karriere i Luftforsvaret.

Holmeset, Adrian

Alder: 27
Tlf: 948 66 395
E-post: adrianholmeset@gmail.com

Bakgrunn:

Adrian gikk data og elektronikk på videregående. Etter å ha tatt påbygging til studiekompetanse (2010) gikk han inn i førstegangstjenesten og videre til Luftforsvarets Befalsskole (2011-2012). Etter å ha arbeidet i Luftforsvaret i 3 år, hovedsakelig som radaroperatør, begynte han på høyere utdanning ved Luftkrigsskolen. Han har siden 2016 studert til å bli elektroingeniør ved NTNU i Gjøvik.

Skinnarland, Hans Einar

Alder: 24
Tlf: 413 27 403
E-post: hasskinn@gmail.com

Bakgrunn:

Hans Einar sin faglige bakgrunnen begynner med generell studiekompetanse med realfag fra videregående (2013). Deretter tok han grunnleggende befalsutdanning gjennom Luftforsvarets befalskole (2013-2015), før han begynte på Grunnleggende offisersutdanning ved Luftkrigsskolen (2015-) i Trondheim. Der går han retningen sivile studier, med tilhørende elektroingeniørstudiet ved NTNU (2016-). Fra arbeidslivet har han jobbet som Wingops-offiser ved Gardermoen Militære Flystasjon (2014-2015).

4.2. Utstyr og ressurser

Arbeidsrom - Gjennom hele semesteret disponerer prosjektgruppen robotlaben til fakultetet, for å arbeide med bacheloroppgaven
APS UR3 Robotic arm - En universal robotarm fra selskapet APS Robotics, som kan utføre mange ulike oppgaver. UR3 er en nokså liten robotarm og plasseres gjerne på en arbeidsbenk, hvor den vil utføre oppgavene fra. Roboten kan rotere 360 grader, griperen kan byttes ut etter behov.
Dronebatteribyttemaskin - Et tidligere bachelorprosjekt lagde en fungerende batteribyttemaskin som utfører batteribytte på droner
Nachi MZ07 Robotic arm - En industriell robotarm med mulighet for å utføre et mangfold av oppgaver alt etter hva den blir programmert til. Det er den raskeste roboten i sin klasse og kan flytte en distanse på 300mm horisontalt og 25mm vertikalt på 0.31 sekunder. Armen har mulighet for montering på alle tre romflater, gulv, vegg og tak.
Økonomi/budsjett - FFI har, basert på erfaring fra tidligere bacheloroppgaver, gitt dette prosjektet en økonomisk ramme på 20.000 NOK. Dette vil inkludere kjøp av utstyr til datasyn, 3D-printing, ledninger og annet utstyr som ikke kan lånes fra NTNU eller FFI.
3D-printer - Fakultetet på NTNU har 3D-printere som gruppen kan disponere underveis i prosjektarbeidet.

4.3. Prosjektleveranser

Underveis i prosjektperioden er det noen frister som skal overholdes. I tabell 4.1 står det informasjon om leveransen og tid den skal leveres. Prosjektleveransene strekker seg fra oppstart av forprosjektet til siste presentasjon av hovedprosjektet er utført.

Leveranser	Tidsfrist
Holde oppstartsmøte med veileder	18.jan
Levere forprosjektrapport	4.feb
Bestemme tittel på norsk og engelsk	15.apr
Levere hovedprosjektrapport	20.mai
Levere A3-plakat for hovedprosjektet	20.mai
Presentasjon Trondheim	slutten av mai
Presentasjon Gjøvik	5. eller 6. juni

Tabell 4.1.: Tabell for leveranser med tilhørende frister

4.4. Tidsplan

I vedlegg B ligger det et Gantt-diagram som gir en oversikt over tidsplanen. Det er også mulig å se tidsplanen fra arbeidspakkene som er vedlagt og nevnt i kapittel. 3.

4.5. Kvalitetssikring

Gruppen har en spesiell utfordring fordi den er spredd over to campuser. Dette vil kunne medføre ekstra utfordringer med tanke på samarbeid og koordinasjon av arbeidet. Det er viktig at gruppen er bevisste på denne utfordringen, og etablerer gode rutiner for koordinering av arbeidet. Gruppen vurderer det som høyst nødvendig å finne gode elektroniske verktøy som legger til rette for samarbeid over avstand. I perioder vil det gjennomføres reiser slik at gruppen får anledning til å møtes i person, og gi anledning til å bruke det samme utstyret og ressursene.

Utover dette ser gruppen det uansett nødvendig at det gjennom arbeidsperioden gjennomføres daglige oppstart- og avslutningssamtaler. Dette for å koordinere dagens gjøremål, og for å kart-

legge status på arbeidet ved endt arbeidsdag. God delegering av arbeidet er en nødvendighet for at det kan utføres godt og effektivt.

Til sist vurderer gruppen det som svært viktig at individene har rom for å dele sine tanker. Det være seg spørsmål, forslag, innspill eller kommentarer knyttet til arbeidet, men også at det er mulig å dele bekymringer av mer personlig natur. Gruppen er ikke sterkere enn sitt svakeste ledd, og det vurderes derfor som svært viktig å ivareta den enkelte.

Det er planlagt å bruke standardiserte dokumenter for møtevirksomhet og timelister, og godt utarbeidede arbeidspakker med tydelig ansvarsdelegering.

A. Arbeidspakker

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Forprosjektrapport		**Aktivitet nr:** 01
Startdato:	07.01.19	**Sluttdato:** 04.02.19	
Avhengighet:	**Foregående aktiviteter:**		
	Etterfølgende aktiviteter: 02 Løsninger for datasyn		
Mål: Levere en forprosjektrapport som en forstudie av hovedprosjektet			
Arbeidsbeskrivelse: En forstudie av bachelorprosjektet skal utføres og dokumenteres i en prosjektrapport. Dette forprosjektet skal gjennomføre en forstudie for å vurdere prosjektets realisme og verdi. Det skal skrives en utfyllende rapport med problemstilling, avgrensninger, mål, tekniske spesifikasjoner, prosjektorganisering og plan for gjennomføring av hovedprosjektet.			
Timeverk: 105 timer		**Fordeling:**	Forester, Per: 35 timer Holmeset, Adrian: 35 timer Skinnarland, Hans Einar: 35 timer
Kostnader:			
Ressurser: PC			
Risiko: Det er ingen spesiell risiko knyttet til arbeidet med forprosjektrapporten.			

Faglig ansvarlig:

Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

Prosjektmedarbeidere:

Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Undersøke datasyn		**Aktivitet nr:** 02
Startdato:	05.02.19	**Sluttdato:** 15.02.19	
Avhengighet:	**Foregående aktiviteter:**	01 Forprosjektrapport	
	Etterfølgende aktiviteter:	03 Implementering av robotsyn	
Mål: Finne den beste løsningen for datasyn til robotarmen			
Arbeidsbeskrivelse: Undersøke ulike allerede etablerte og uetablerte løsninger på datasyn. Utføre et litteratursøk på datasyn gjennom å lese rapporter, vitenskapelige artikler, lese på produsenters hjemmesider, snakke med erfarne mennesker innenfor fagfeltet.			
Timeverk: 102 timer		**Fordeling:**	Forester, Per: 30 timer Holmeset, Adrian: 40 timer Skinnarland, Hans Einar: 32 timer
Kostnader:			
Ressurser: PC, bibliotek			
Risiko: Det er ingen spesiell risiko knyttet til arbeidet med å undersøke datasyn til robotarm			

Faglig ansvarlig:
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com

Prosjektmedarbeidere:
Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com
Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Implementere datasyn		**Aktivitet nr:** 03
Startdato:	15.02.19	**Sluttdato:** 29.03.19	
Avhengighet:	**Foregående aktiviteter:**	02 Undersøke datasyn	
	Etterfølgende aktiviteter:	05 Programmere og teste	
Mål: Implementere datasyn på robotarmen			
Arbeidsbeskrivelse: Finne en måte for å prossessere datasyn slik at man kan styre robotarmen mot et objekt. Sannsynligvis vil dette innebære å bruke en datamaskin til å prosessere bilder fra en kameraenhet. Datamaskinen vil da måtte kunne kommunisere med robotarmen for å styre den til ønsket posisjon.			
Timeverk: 410 timer		**Fordeling:**	Forester, Per: 140 timer Holmeset, Adrian: 140 timer Skinnarland, Hans Einar: 130 timer
Kostnader:			
Ressurser: PC, robotarm, bibliotek			
Risiko: Det er ingen spesiell risiko knyttet til arbeidet med å implementere datasyn til robotarm			

Faglig ansvarlig:
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com

Prosjektmedarbeidere:
Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com
Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Gripemekanisme		**Aktivitet nr:** 04
Startdato:	25.02.19	**Sluttdato:** 18.03.19	
Avhengighet:	**Foregående aktiviteter:**	02 Undersøke datasyn	
	Etterfølgende aktiviteter:	05 Programmere løsning og testing	
Mål: Gjøre det mulig for robotarmen å gripe tak i dronen, og løfte den			
Arbeidsbeskrivelse: Utvikle en løsning slik at robotarmen kan plukke opp dronen uten å ødelegge den. Det skal undersøkes om gripemekanismen som følger med robotarmen er brukbar, eller om det må brukes en annen. Om det skal brukes en annen skal det gjøres en vurdering på hvilken, og anskaffe denne. Det skal også vurderes om det må gjøres noen fysiske endringer på dronen for at den skal være mulig å gripe tak i. Endringer kan ikke være for store i frykt for å ødelegge dronens flygeevne samt at den kan ende opp med å veie mer enn 250g.			
Timeverk: 98 timer		**Fordeling:**	Forester, Per: 30 timer Holmeset, Adrian: 30 timer Skinnarland, Hans Einar: 38 timer
Kostnader:			
Ressurser: PC, robotarm, griper, drone			
Risiko: Det er en viss fare for å knuse deler av dronen underveis i jobbinga med å få griperen til å fungere optimalt. Det er også noe risiko knyttet til å arbeide med en fysisk robotarm som i verste fall kan skade utstyr og mennesker.			

Faglig ansvarlig:

Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

Prosjektmedarbeidere:

Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Programmere og teste		**Aktivitet nr:** 05
Startdato:	18.03.19	**Sluttdato:** 12.04.19	
Avhengighet:	**Foregående aktiviteter:**	03 Implementere datasyn 04 Gripemekanisme	
	Etterfølgende aktiviteter:	06 Hovedprosjektrapport	
Mål: Gjennom datasyn vil robotarmen plukke opp dronen og plassere den riktig plass			
Arbeidsbeskrivelse: Robotarmen skal kunne hente dronen og plassere den korrekt i batteribyttemaskinen, for så å sette den tilbake på landingsplattformen for en ny take-off. Det skal programmeres slik at robotarmen kan gjenkjenne dronen, vite orienteringen, når det er trygt å plukke den opp, når batteribyttemaskinen er klar for å ta imot, og når batteribytte er utført.			
Timeverk: 260 timer		**Fordeling:**	Forester, Per: 80 timer Holmeset, Adrian: 90 timer Skinnarland, Hans Einar: 90 timer
Kostnader:			
Ressurser: PC, robotarm, griper, drone			
Risiko: Det er en viss fare for skade på dronen og annet utstyr under testing. Det er også noe risiko knyttet til å arbeide med en fysisk robotarm som i verste fall kan skade utstyr og mennesker.			

Faglig ansvarlig:
Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com

Prosjektmedarbeidere:
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com
Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Hovedprosjektrapport		**Aktivitet nr:** 06
Startdato:	1.03.19	**Sluttdato:** 20.05.19	
Avhengighet:	**Foregående aktiviteter:**	05 Programmere og teste	
	Etterfølgende aktiviteter:	07 Muntlig presentasjon	
Mål: Skrive og ferdigstille hovedprosjektrapporten			
Arbeidsbeskrivelse: Alt arbeidet underveis i hovedprosjektet skal dokumenteres og samles i en rapport som omhandler arbeidet underveis. Tidligere rapporter, manualer, tidligere arbeid skal brukes i rapporten. Rapporten skal ta for seg den utarbeidede problemstillingen, gi svar på om dette ble løst, og på hvilken måte gruppa løste problemstillingen. Det skal også lages en A3-plakat om arbeidet, som også skal leveres med rapporten.			
Timeverk: 430 timer		**Fordeling:**	Forester, Per: 140 timer Holmeset, Adrian: 140 timer Skinnarland, Hans Einar: 150 timer
Kostnader:			
Ressurser: PC, bibliotek, kamera			
Risiko: Det er ingen spesiell risiko knyttet til arbeidet med hovedprosjektrapporten			

Faglig ansvarlig:

Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

Prosjektmedarbeidere:

Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com
Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Muntlig presentasjon		**Aktivitet nr:** 07
Startdato:	20.05.19	**Sluttdato:** 06.06.19	
Avhengighet:	**Foregående aktiviteter:**	06 Hovedprosjektrapport	
	Etterfølgende aktiviteter:		
Mål: Forberede muntlig presentasjon			
Arbeidsbeskrivelse: Forberede muntlig presentasjon for fremføring.			
Timeverk: 45 timer		**Fordeling:** Forester, Per: 15 timer Holmeset, Adrian: 15 timer Skinnarland, Hans Einar: 15 timer	
Kostnader:			
Ressurser: PC, bibliotek			
Risiko: Det er ingen spesiell risiko knyttet til arbeidet med å forberede fremføring			

Faglig ansvarlig:
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com

Prosjektmedarbeidere:
Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com
Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com

NTNU

Institutt for teknisk kybernetikk,
Studieretning automatiseringsteknikk

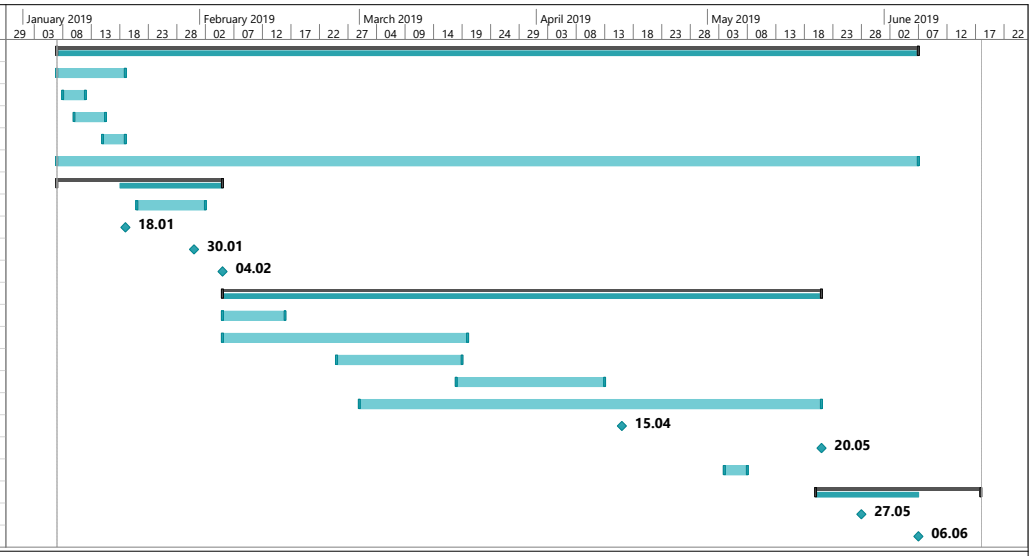
Fag:	**TELE3001 Bacheloroppgave**		**Dato:** 31.01.19
Prosjekt:	**Implementering av servicearm**		
Aktivitet:	Administrativt arbeid		**Aktivitet nr:** 08
Startdato:	07.01.19	**Sluttdato:** 06.06.19	
Avhengighet:	**Foregående aktiviteter:**		
	Etterfølgende aktiviteter:		
Mål: Administrerende arbeid nødvendig for å løse hovedprosjektet			
Arbeidsbeskrivelse: Foreta det nødvendige administrative arbeidet som kreves for å løse prosjektet på en god og ryddig måte. Arbeidet inkluderer utarbeidingen av nødvendige dokumentmaler, administrering av nødvendige dokumenter, som møteinnkallelser, referater, timelister m.m., så vel som anskaffelsen av nødvendige ressurser for arbeidet, som lokaler, IKT-verktøy og tilganger. Arbeidet inkluderer også nødvendige oppstartssamtaler og signering av avtaler.			
Timeverk: 110 timer	**Fordeling:**	Forester, Per: 50 timer Holmeset, Adrian: 30 timer Skinnarland, Hans Einar: 30 timer	
Kostnader:			
Ressurser: IKT-verktøy.			
Risiko: Det er ingen spesiell risiko knyttet til det administrative arbeidet.			

Faglig ansvarlig:
Forester, Per tlf: 993 88 804 e-post: per.forester@gmail.com

Prosjektmedarbeidere:
Skinnarland, Hans Einar tlf: 413 27 403 e-post: haskinn@gmail.com
Holmeset, Adrian tlf: 948 66 395 e-post: adrianholmeset@gmail.com

B. Gantt-diagram

ID	Task Name	Duration	Start
1	**Administrativt arbeid**	**109 days**	**Mon 07.01.19**
2	Planlegge oppstartsmøte	10 days	Mon 07.01.19
3	Skaffe arbeidsrom	4 days	Tue 08.01.19
4	Skaffe arbeidmaskin	3,5 days	Thu 10.01.19
5	Opprette maler og filstruktur	4 days	Tue 15.01.19
6	Utføre administrativt arbeid	109 days	Mon 07.01.19
7	**Fase 1: Forprosjekt**	**21 days**	**Mon 07.01.19**
8	Skrive forprosjekt	10 days	Mon 21.01.19
9	Oppstartsmøte	1 day	Fri 18.01.19
10	Avtalesignering	1 day	Wed 30.01.19
11	Rapportlevering	1 day	Mon 04.02.19
12	**Fase 2: Hovedprosjekt**	**75 days**	**Tue 05.02.19**
13	Undersøke datasyn	9 days	Tue 05.02.19
14	Implementere datasyn	31 days	Tue 05.02.19
15	Gripemekanisme	16 days	Mon 25.02.19
16	Programmere og teste	20 days	Mon 18.03.19
17	Hovedprosjektrapport	57 days	Fri 01.03.19
18	Finne tittel på norsk og engelsk	1 day	Mon 15.04.19
19	Levere Hovedprosjekt	1 day	Mon 20.05.19
20	Lage infoplakat	3 days	Sat 04.05.19
21	**Fase 3: Avslutning**	**21 days**	**Mon 20.05.19**
22	Presentere NTNU Trondheim	1 day	Mon 27.05.19
23	Presentere NTNU Gjøvik	2 days	Wed 05.06.19



Project: Gant for bacheloroppg
Date: Thu 31.01.19

Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress

B myrobot_mp.py

```
#!/usr/bin/env python


import rospy, sys, numpy as np
import roslib; roslib.load_manifest('ur_driver')
import actionlib
from control_msgs.msg import *
from trajectory_msgs.msg import *
from sensor_msgs.msg import JointState
from math import pi

import moveit_commander
from copy import deepcopy
import geometry_msgs.msg

from myrobot.msg import *
from tf.transformations import *

import moveit_msgs.msg
from std_msgs.msg import Header
from robotiq_2f_gripper_control.msg import *

class myrobot_mp:
    def __init__(self):
        moveit_commander.roscpp_initialize(sys.argv)
        rospy.init_node('myrobot_mp')

        rospy.Subscriber('cam_coord', finPoint, self.coordCallback)
        rospy.Subscriber('Robotiq2FGripperRobotInput',
                         Robotiq2FGripper_robot_input, self.gripperCallback)
        rospy.Subscriber('drone_out', drone_status, self.droneCallback)

        self.r2f_pub = rospy.Publisher('Robotiq2FGripperRobotOutput',
                         Robotiq2FGripper_robot_output, queue_size=10)
        self.r2f_out = Robotiq2FGripper_robot_output()
        self.r2f_input = Robotiq2FGripper_robot_input()
        self.idle = True
        self.x = 0
        self.y = 0
        self.status = False #drone status -> True if landed
        self.valid = False #camera found bright spot -> True if found
        self.robot = moveit_commander.RobotCommander()
        self.scene = moveit_commander.PlanningSceneInterface()
        self.group = moveit_commander.MoveGroupCommander("manipulator")
        self.group_names = self.robot.get_group_names()

        self.display_trajectory_publisher =
            rospy.Publisher('/move_group/display_planned_path',
                         moveit_msgs.msg.DisplayTrajectory, queue_size=20)

        rospy.sleep(1)

        #add table to scene model for collision avoidance
        t = geometry_msgs.msg.PoseStamped()
        t.header.frame_id = self.robot.get_planning_frame()
        t.pose.position.x = 0.
        t.pose.position.y = 0.
        t.pose.position.z = -0.051
        #print("adding box")
        self.scene.add_box("table", t, (2,2,0.1))


        #add charging to scene model for collision avoidance
        c = geometry_msgs.msg.PoseStamped()
        c.header.frame_id = self.robot.get_planning_frame()
        c.pose.position.x = -0.45
        c.pose.position.y = 0.17
        c.pose.position.z = 0.065
        #print("adding box")
        self.scene.add_box("chargingStation", c, (0.5,0.6,0.13))


```

```
73          #TODO: Initialize gripper!
74          self.r2f_pub.publish(self.r2f_out) #reset
75          rospy.sleep(0.3)
76          self.r2f_out.rACT = 1
77          self.r2f_out.rGTO = 1
78          self.r2f_out.rSP = 255
79          self.r2f_out.rFR = 150
80          self.r2f_pub.publish(self.r2f_out) #activate
81          rospy.sleep(1)
82
83      def droneCallback(self, msg):
84          self.heading = msg.heading
85          self.status = msg.status
86
87      def coordCallback(self, fp):
88          self.x = fp.x
89          self.y = fp.y
90          self.valid = fp.valid
91
92      def gripperCallback(self, msg):
93          self.r2f_input.gACT = msg.gACT
94          self.r2f_input.gGTO = msg.gGTO
95          self.r2f_input.gSTA = msg.gSTA
96          self.r2f_input.gOBJ = msg.gOBJ
97          self.r2f_input.gFLT = msg.gFLT
98          self.r2f_input.gPR = msg.gPR
99          self.r2f_input.gPO = msg.gPO
100         self.r2f_input.gCU = msg.gCU
101
102     def execute(self):
103
104         # Allow some leeway in position (meters) and orientation (radians)
105         self.group.allow_replanning(True)
106         self.group.set_goal_position_tolerance(0.002)
107         self.group.set_goal_orientation_tolerance(0.02)
108         self.group.set_planning_time(3)
109         self.group.set_num_planning_attempts(50)
110         self.group.set_max_acceleration_scaling_factor(.1)
111         self.group.set_max_velocity_scaling_factor(.1)
112
113
114         P1 = [-1.92, -2.06, -0.585, -0.505, 1.733, 0]
115
116         if (not self.group.go(P1, wait=True)): #stand-by position
117             print("No path found! Stand-by position")
118             return
119
120
121         #waiting for location and heading
122         while not((self.valid and self.idle and self.status) or rospy.is_shutdown()):
123             print("waiting for valid")
124             rospy.sleep(1)
125
126
127         self.idle = False
128
129         pose_goal = geometry_msgs.msg.Pose()
130         pose_goal.position.x = self.x
131         pose_goal.position.y = self.y
132         pose_goal.position.z = 0.29
133
134         #rotate to drone heading
135         #self.heading -> [0-255] 0 is along the arm y-axis
136         if self.heading > 128:
137             radOff = (0-((self.heading*2*np.pi)/256)) % np.pi * -1
138         elif self.heading == 128:
139             radOff = np.pi
140         else:
141             radOff = ((self.heading*2*np.pi)/256) % np.pi
142         q_down = [1, 0, 0, 0]
143         q_rot = quaternion_from_euler(0, 0, radOff)
144         q_new = quaternion_multiply(q_rot, q_down)
```

```
145             pose_goal.orientation.x = q_new[0]
146             pose_goal.orientation.y = q_new[1]
147             pose_goal.orientation.z = q_new[2]
148             pose_goal.orientation.w = q_new[3]
149             self.group.set_pose_target(pose_goal)
150
151             if (not self.group.go( wait=True)): #above pick-up-point
152                 print("No path found! Above pick-up point")
153                 return
154
155             self.group.stop()
156             self.group.clear_pose_targets()
157
158
159             pose_goal.position.z = 0.25
160             self.group.set_pose_target(pose_goal)
161
162             if (not self.group.go( wait=True)): #above pick-up-point
163                 print("No path found! Pick-up point")
164                 return
165
166             self.group.stop()
167             self.group.clear_pose_targets()
168
169
170             #add drone to scene
171             box_pose = geometry_msgs.msg.PoseStamped()
172             box_pose.header.frame_id = self.robot.get_planning_frame()
173             box_pose.pose.position.x = self.x
174             box_pose.pose.position.y = self.y
175             box_pose.pose.position.z = 0.0425
176             box_pose.pose.orientation = pose_goal.orientation
177             box_name = "drone"
178             self.scene.add_box(box_name, box_pose, size=(0.185, 0.16, 0.085))
179
180             self.r2f_out.rPR = 225
181             self.r2f_pub.publish(self.r2f_out)
182
183             #wait for gripper to close
184             while (self.r2f_input.gPO > self.r2f_out.rPR + 15) or
185                     (self.r2f_input.gPO < self.r2f_out.rPR - 15):
186                 rospy.sleep(0.3)
187
188             #attach box to gripper for moveit planning
189             touch_links = self.robot.get_link_names(group="gripper")
190             self.scene.attach_box(self.group.get_end_effector_link(),
191                     box_name, touch_links=touch_links)
192
193             rospy.sleep(0.2)
194
195             pose_goal.position.z = 0.29
196             self.group.set_pose_target(pose_goal)
197             if (not self.group.go( wait=True)): #above pick-up-point
198                 print("No path found! Pick-up point")
199                 return
200             self.group.stop()
201             self.group.clear_pose_targets()
202
203             #end point - drone charging station
204             pose_goal.position.x = -0.414
205             pose_goal.position.y = 0.007
206             pose_goal.position.z = 0.39 #temporaryly increased for card board box
207             pose_goal.orientation.x = np.sqrt(0.5)
208             pose_goal.orientation.y = np.sqrt(0.5)
209
210             self.group.set pose target(pose goal)
211             print("trying to move above drop-off-point")
212             if (not self.group.go(wait=True)): #above charging station
213                 print("No path found! Above drop off")
214                 return
215             self.group.stop()
216             self.group.clear_pose_targets()
```

```
217
218
219             pose_goal.position.z = 0.335
220             self.group.set_pose_target(pose_goal)
221             if (not self.group.go(wait=True)): #above charging station
222                 print("No path found! Lowering to drop off")
223                 return
224             self.group.stop()
225             self.group.clear_pose_targets()
226
227             self.r2f_out.rPR = 0
228             self.r2f_pub.publish(self.r2f_out)
229             while (self.r2f input.gPO > self.r2f out.rPR + 5) or
230                     (self.r2f_input.gPO < self.r2f_out.rPR - 5) and not
231                     rospy.is_shutdown():
232                 rospy.sleep(0.3)
233
234             self.scene.remove_attached_object(self.group.get_end_effector_link(),
235                             name=box_name)
236             self.scene.remove_world_object(box_name)
237
238             self.idle = True
239
240             rospy.sleep(0.3)
241
242
243
244     if __name__ == '__main__':
245         try:
246             myrob = myrobot_mp()
247             myrob.execute()
248         except rospy.ROSInterruptException:
249             pass
250
```

C camNode.py

```python
#!/usr/bin/env python
from __future__ import print_function

import sys
import rospy
import cv2
import numpy as np
from std_msgs.msg import String
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
from myrobot.msg import finPoint

class image_converter:

    def __init__(self):
        rospy.init_node('cam_node')
        self.coord_pub = rospy.Publisher("cam_coord",finPoint , queue_size=5)

        self.bridge = CvBridge()
        self.image_sub = rospy.Subscriber("usb_cam/image_rect"
                          ,Image,self.callback)

    def callback(self,data):
        try:
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            print(e)

        threshVal = 100 #TUNE THIS

        cX = cY = 0
        (rows,cols,channels) = cv_image.shape

        cameraScale = 0.217/640

        #coordinate transform:
        #rotation matrices:
        rotY = [[np.cos(np.pi),0,np.sin(np.pi)],
                [0, 1, 0],
                [-np.sin(np.pi), 0 ,np.cos(np.pi)]]
        rotZ = [[np.cos(-71*np.pi/180),-np.sin(-71*np.pi/180), 0],
                [np.sin(-71*np.pi/180), np.cos(-71*np.pi/180), 0],
                [0, 0, 1]]
        rot = np.dot(rotY, rotZ)

        #translation
        displacement = [[0.359],[0.297], [0]]
        translationMat = np.concatenate((rot, displacement),1)
        translationMat = np.concatenate((translationMat, [[0,0,0,1]]),0)


        if cols > 60 and rows > 60:
            redChan =   cv2.extractChannel(cv_image, 2)
            redChan =  cv2.blur(redChan, (5,5))
            _, maxVal, _, maxLoc = cv2.minMaxLoc(redChan)
            thresh = cv2.threshold(redChan, 100, 255, cv2.THRESH_BINARY)[1]
            contours = cv2.findContours(thresh, cv2.RETR_TREE
                        , cv2.CHAIN_APPROX_SIMPLE)[1]
            if (len(contours) > 0):
                contours = max(contours, key = cv2.contourArea)
                M = cv2.moments(contours)
                cX = int(M["m10"] / M["m00"])
                cY = int(M["m01"] / M["m00"])

                coord = np.dot(translationMat, [[cX*cameraScale]
                            ,[cY*cameraScale], [0], [1]])
                outMsg = finPoint()
                outMsg.x = coord[0]
                outMsg.y = coord[1]
                if maxVal > threshVal:
                    self.coord_pub.publish(outMsg)

```

```
73  def main(args):
74      ic = image_converter()
75      try:
76          rospy.spin()
77      except KeyboardInterrupt:
78          print("Shutting down")
79      cv2.destroyAllWindows()
80
81  if __name__ == '__main__':
82      main(sys.argv)
```

D dronekoden.cpp

```
/*
*Code #3 for the DRONE
*Wait for status request
*if OK; reads magnetometer(heading)
*gives status(heading) if OK
*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303_U.h>

/* Assign a unique ID to this sensor at the same time */
Adafruit_LSM303_Mag_Unified mag = Adafruit_LSM303_Mag_Unified(12345);

#define baud 9600
#define csNRF 10       //ChipSelect for nRF24
#define radioEnable 9  //ChipEable for nRF24

#define magY IMU.getMagY_uT()
#define magX IMU.getMagX_uT()
#define magZ IMU.getMagZ_uT()


RF24 radio(radioEnable,csNRF); // CE, CSN
const byte addresses[][6] = {"sNode","dNode"}; //Drone writes dNode. vice-versa

bool pendingRequest = 0;
//float heading;

void setup() {
 initSerial();// delete later. Initiates SerialCom.
 initLSM();   //Initiates the compass
 initNRF();   //Initiates the radio
} //---End SETUP

void loop() {
  radio.startListening();
  delay(5);

  if (radio.available()) {
    radio.read(&pendingRequest, sizeof(pendingRequest));
    delay(10);
  }
  if (pendingRequest == 1){     //Reads heading and sends it
    respond();
  }
} //-------End LOOP



/**********FUNCTIONS**********/
/****Setup***/
void initLSM(){
  if(!mag.begin()){
    /* There was a problem detecting the LSM303 ... check your connections */
    Serial.println("Ooops, no LSM303 detected ... Check your wiring!");
    while(1);
  }
}
void initSerial(){
  Serial.begin(baud);
}
void initNRF(){
  radio.begin();
  radio.openWritingPipe(addresses[1]);      //Write on dNode
  radio.openReadingPipe(1,addresses[0]);    //Listen to sNode
```

```
70      radio.setPALevel(RF24_PA_MIN);            //Set min.TX-power
71    }
72
73    /***Actions***/
74    void respond(){
75      radio.stopListening();
76
77      sensors_event_t event;
78      mag.getEvent(&event);
79      float heading = (atan2(event.magnetic.y,event.magnetic.x) * 180) / PI;
80
81      if (heading < 0){
82        heading = 360 + heading;
83      }
84      byte sendHdg = map(heading, 0,360,0,255);
85      radio.write(&sendHdg, sizeof(sendHdg));
86      pendingRequest = 0;
87      //delay(50);
88    }
89
90    /*float getHeading(){
91      //float heading;
92
93
94     // heading = atan2(magY,magX)*(180)/PI;
95      if (heading < 0){
96      heading = 360 + heading;
97      }
98      return heading;
99    }*/
100
```

E stasjonskoden.cpp

```
#include <ros.h>
#include <std_msgs/UInt8.h>
#include <std_msgs/Bool.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

#define baud 9600

RF24 radio(9,10); // CE, CSN
const byte addresses[][6] = {"sNode","dNode"};
byte heading;

ros::NodeHandle nh;

std_msgs::UInt8 hdgMsg;
ros::Publisher drone_heading("drone_heading", &hdgMsg);

byte promptHeading (const std_msgs::Bool& prompt_msg){
  if(prompt_msg.data){
    heading = getHeading();
    hdgMsg.data = heading;
    Serial.println(heading);
    drone_heading.publish( &hdgMsg );
  }
 }

ros::Subscriber<std_msgs::Bool> drone_prompt("drone_prompt", &promptHeading);




/*------Setup-------------------------------------------------------------------*/
void setup() {
  initSerial();
  initNRF();

  nh.initNode();
  nh.advertise(drone_heading);
  nh.subscribe(drone_prompt);

}

void loop() {

  nh.spinOnce();
  delay(1000);
}

/*------Initializations---------------------------------------------------------*/

void initSerial(){
  Serial.begin(baud);
}
void initNRF(){
  radio.begin();
  radio.openWritingPipe(addresses[0]);          //Write on sNode
  radio.openReadingPipe(1,addresses[1]);        //Listen on dNode
  radio.setPALevel(RF24_PA_MIN);                //Set min.TX-power
}
/*-------Functions---------------------------------------------------------------*/
byte getHeading(){                                  ///Prompts drone for heading
    radio.stopListening();
    int pendingRequest = 1;
    Serial.println("Sending...");                              //Free to comment
    radio.write(&pendingRequest, sizeof(pendingRequest));

    radio.startListening();
```

```
70        delay(20);                                                    //needs 20ms
71        if (radio.available()) {
72            int headingRx;
73            radio.read(&headingRx, sizeof(headingRx));
74            Serial.print("Drone HDG: ");                          //Free to comment
75            Serial.println(headingRx);                          //Free to comment
76            return headingRx;
77
78      }
79    }
80
```