Nr. _____ av _____

Sjøkrigsskolen

Bacheloroppgave

Autonom kollisjonsunnvikelse på maritim dronesverm

– Bruk av AIS og radar til operativ bruk på ubemannede fartøy –

av

Mats Riis Asdahl, Julian Ugedal Riis og Petter Antoni Nilsen Wandas

Levert som en del av kravet til graden:

BACHELOR I MILITÆRE STUDIER MED FORDYPNING I LEDELSE OG MARINEINGENIØR VÅPEN, ELEKTRONIKK OG DATA

Antall ord: 21 887

Innlevert: 3. desember 2023

Godkjent for offentlig publisering

Publiseringsavtale

En avtale om elektronisk publisering av bachelor/prosjektoppgave

Kadettene har opphavsrett til oppgaven, inkludert rettighetene til å publisere den.

Alle oppgaver som oppfyller kravene til publisering vil bli registrert og publisert i Bibsys Brage når kadettene har godkjent publisering.

Oppgaver som er graderte eller begrenset av en inngått avtale vil ikke bli publisert.

Vi gir herved Sjøkrigsskolen rett til å gjøre denne oppgaven tilgjengelig elektronisk, gratis og uten kostnader	[x] Ja	[ ] Nei
Finnes det en avtale om forsinket eller kun intern publisering? (Utfyllende opplysninger må fylles ut)	[ ] Ja	[x] Nei
Hvis ja: kan oppgaven publiseres elektronisk når embargoperioden utløper?	[ ] Ja	[ ] Nei

Plagiaterklæring

Vi erklærer herved at oppgaven er vårt eget arbeid og med bruk av riktig kildehenvisning.

Vi har ikke nyttet annen hjelp enn det som er beskrevet i oppgaven.

Vi er klar over at brudd på dette vil føre til avvisning av oppgaven.

Dato: 03.12.2023

Mats Riis Asdahl — Mats Riis Asdahl
Kadett navn — Kadett, signatur

Petter Antoni Nilsen Wandas — _____
Kadett navn — Kadett, signatur

Julian Ugedal Riis — _____
Kadett navn — Kadett, signatur

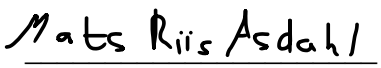
Forord

Tidligere er det skrevet to bacheloroppgaver ved FHS Sjøkrigsskolen som omhandler maritime dronesvermer som har relevans til vår oppgave. Oppgaven bygger videre på Hellesnes og Lyssand sin bacheloroppgave fra 2019: «Plattform for sverm - Fra store avanserte plattformer til mange små». I tillegg så er det fire år siden sist all koden, komponentene og sjødronene ble brukt så det er jo en fare for at det kan oppstå problemer, for teknologien er i konstant utvikling.

Ved å bruke deres ferdige plattform er det mange muligheter med å videreutvikle sjødroner for å gjøres deres ferdige sjødroner mer bevisste på omgivelsene rundt seg ved å legge til sensorer og annen data vi mener vil forbedre denne plattformen og vise hvilke muligheter man kan ha med begrensede ressurser.

Vi ønsker å spesielt å rette en takk til førsteamanuensis Aleksander Sauter for veiledning, konstruktive diskusjoner og god hjelp når det virkelig har vært utfordrende. Takk til førsteamanuensis Christophe Massacand for hjelp til feilsøking og nyttige diskusjoner. En takk til avdelingsingeniør Frode Wikne for støtte til 3D-printing samt teknisk støtte ved problemer på modellbåtene. Takk til førstelektor Terje Fykse for korrekturlesing. Til slutt ønsker vi å takke tidligere kadett ved FHS Sjøkrigsskolen Andreas Hellesnes for å ta tiden til å gi introduksjon til sin bacheloroppgave.

Bergen, Sjøkrigsskolen, 03-12-2023



Mats Riis Asdahl

Petter Antoni Nilsen Wandas

Julian Ugedal Riis

Sammendrag

I en tid med krig i Europa er den geopolitiske stabiliteten på kontinentet avhengig av sikker og stabil gassimport fra Norge som aldri før (DNV, 2022, s. 2). Samtidig har hendelsene på Nordstream i 2022 og Balticconnector i 2023 vist hvor utsatt infrastrukturen på havets dyp er. Forsvarskommisjonen av 2021 har konkludert med at dagens sjøforsvarsstruktur ikke har tilstrekkelig evne til å møte norske strategiske interesser i nordområdene (Forsvarskommisjonen, 2023, s. 54-55). Små ubemannede fartøyer kan bidra til å minke gapet mellom evne og ambisjon. Dronesvermer bestående av tre eller flere enheter vil kunne utvide bruksområdet og redusere sårbarheten sammenlignet med et tilsvarende antall fartøyer som løser oppdrag alene. En testplattform for en slik maritim dronesverm ble utviklet ved Sjøkrigsskolen i 2019, tilrettelagt for å kunne utvide funksjonaliteten til systemet (Hellesnes & Lyssand, 2019). Det er en klar fordel at dronesvermer har en stor grad av autonomi og evner å løse oppdrag uten kontinuerlig oppsyn og styring av en operatør. For å realisere dette må hver enkelt drone være bevisst omgivelsene sine, samt kunne navigere trygt både blant båter på havet og mellom øyer og holmer langs kysten. Dermed stiller denne oppgaven seg følgende problemstilling:

Hvordan kan AIS og radar implementeres på dronesverm for å drive kollisjonsunnvikelse, og hvilke utfordringer ligger det i å implementere dette i en eksisterende testplattform for dronesverm?

Dette prosjektet har utviklet et konsept for kollisjonsunnvikelse samtidig som den belyser problemstillingen rundt mangel på oppdatering og vedlikehold av aldrene systemer. Gjennom kompatibilitetstesting kan det konkluderes at testplattformen fra 2019 bør moderniseres for å sikre fremtidig mulighet for videreutvikling. Innhenting av live AIS-data fra Kystverket muliggjør unnvikelse av fartøy før en kollisjon, men må for øyeblikket simuleres grunnet inkompatibilitet mellom ny og gammel programvare. Radaren er ikke fullstendig implementert, men viser lovende resultater og skal med få utbedringer kunne integreres direkte inn i dronesystemet.

Modernisering av testplattformen vil ikke bare gjøre integrering av AIS og radar mer sømløst, men også flere systemer vil kunne kombineres og dermed skape en dronesverm med mer komplett funksjonalitet som evner å løse komplekse oppdrag.

Innholdsfortegnelse

Figurer

Tabeller

Nomenklatur

Begrep/forkortelse	Beskrivelse
AIS	Automatic Identification System
Aktuator	En mekanisk bevegelse av et teknisk organ som styres av strøm
Algoritme	En presis beskrivelse av hvordan operasjoner skal utføres for å løse ett eller flere problemer
API	Application Programming Interface
Autonomi	Selvstyre/selvbestemmelse
Avstandsnøyaktighet	Hvor nøyaktig avstandsmåling radaren evner å gi
Avstandsoppløsning	Radarens evne til å skille mellom to nærliggende objekter
ECDIS	Electronic Chart Display and Information System
FMCW	Frequency Modulated Continous Wave (radar)
GCS	Ground Control Station, kontrollstasjonen som gir brukeren bilde over egne ubemannede fartøy sin posisjon og bevegelser
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
Hastighetsnøyaktighet	Hvor nøyaktig hastighetsmåling radaren evner å gi
Hastighetsoppløsning	Radarens evne til å skille farten til to nærliggende objekter
IMO	International Maritime Organization
Legacy-system	Et utdatert datasystem, programmeringsspråk eller programvare som av ulike årsaker brukes i stedet for tilgjengelige oppgraderte versjoner.
Millimeterbølgeradar	Radar som anvender bølger med bølgelengde 1-10 mm, dermed frekvenser 30-300 GHz
Open Source Software	Kode som er tilgjengelig for alle, kan endres og redistribueres av «alle»
OS	Operativsystem

ROS	Robot Operating System
RPi	Raspberry Pi
Syntaks	Forhåndsdefinerte regler som styrer oppbygging av fraser og setninger i et språk, f.eks. Python i denne oppgaven.
Teknisk Gjeld	Problemer som oppstår etter en mangel på oppdatering av system og/eller programvare.
UAV	Unmanned Aerial Vehicle

1 Introduksjon

I dette kapittelet vil først den tematiske bakgrunnen for prosjektet bli lagt frem, for så å introdusere tidligere produserte bacheloroppgaver som er relevante innenfor fagområdet. Introduksjonen fortsetter med å presentere problemstillingen, og hvordan den konkretiseres gjennom definerte mål og begrensninger, før den avslutningsvis går igjennom strukturen til oppgaven.

1.1 Bakgrunn

Norge er en maritim nasjon. Kystlinjen strekker seg over hundre tusen kilometer fra nord til sør, med tilhørende havarealer over fem ganger så store som landarealet (Thorsnæs, 2023). Den maritime næringen er sammensatt og består av blant annet petroleumsvirksomhet, shipping og fiskeri, hvor disse til sammen står for om lag 70 prosent av nasjonens eksportinntekter (Regjeringen, 2021). «Europe is dependent on secure gas import from Norway» skriver DNV i rapporten *Energy Transition Norway* fra 2022, og understreker videre at geopolitisk stabilitet i Europa er avhengig av stabil energiimport fra Norge. Hendelsene på Nordstream i 2022 og Balticconnector i 2023 har tydeliggjort hvor sårbar infrastrukturen i Nordsjøen er, og Forsvarskommisjonen av 2021 har konkludert at dagens vedtatte sjøforsvarsstruktur ikke har tilstrekkelig evne til å verne om nordområdene våre (Korme, 2023). Løsningen er derimot ikke så enkel som å produsere et stort antall fartøy, ettersom det også kreves personell for å bemanne fartøyene. En slik styrkeproduksjon vil ta mange år, og kreve store endringer i styrkestrukturen (Forsvarskommisjonen, 2023).

Det amerikanske forsvarsdepartementet har satt i gang «The Replicator Initiative», et initiativ som har som mål «to field attainable autonomous systems at scale of multiple thousands, in multiple domains, within the next 18-to-24 months» (Hicks, 2023). Kina har verdens største marine, og har som mål å bli en verdensledende militærmakt innen midten av århundret (U.S. Department of Defence, 2023). Det amerikanske initiativet har som hensikt å "[…] counter the [Chinese People's Liberation Army's] mass with mass of our own" (Hicks, 2023), noe som er en interessant tankegang også i norsk sammenheng.

Den norske marinen vil ikke kunne måle seg med marinen til Russland rent tallmessig, for selv om investeringsbudsjettene hadde økt betraktelig så hadde bemanning fortsatt vært en stor utfordring (Grønning & Dimmen, 2022). En satsning på autonome systemer kan derfor være en måte å kontre den tallmessige underlegenheten.

Fremtidens minemottiltak er tiltenkt å bli en realitet rundt 2030, og dette med stort fokus på autonome fartøy som kan gjennomføre ubemannede minesveip og mineryddinger. «Hovedhensikten med systemet er å finne og uskadeliggjøre miner uten å dra personell inn i det minelagte området.» (Nakjem, 2023) Dette kan bli et tiltak som kan forstørre rekkevidden og tilstedeværelsen på det personellet som allerede finnes, samtidig som de settes i mindre fare. Noe som kan få personellmangelen til å være mindre pressende.

«Anskaffelsen danner grunnlag for å innføre autonome, modulære systemer også i andre deler av Forsvaret.» (Andersen, 2023) Som regjeringen sier i statsbudsjettet 2024: «Teknologier som for eksempel stordata, kunstig intelligens, autonomi, […] vil bli viktig for norsk sikkerhet og forsvarsevne framover» (Regjeringen, 2023). Dette gir muligheter for nye løsninger ved at næringslivet har stor interesse i fagområdene og er en sterk pådriver i utviklingen av teknologiene. Utviklingen av autonome systemer er tett knyttet til utviklingen av droner.

«Drone er et ubemannet luftfartøy som kan kontrolleres med fjernstyring eller fly autonomt ved hjelp av programvare, sensorer og GPS» (Tandberg, Engerengen, & Jarslett, 2023). Det samme gjelder for en sjødroner, bare at disse beveger seg på eller under vann istedenfor i luften – med fordelen i at de kan ligge i ro over lengre tid uten å bruke betydelige ressurser. For at (sjø)droner skal kunne operere autonome må en rekke av forutsetninger være på plass. Prinsippene for autonomi er noe mer komplisert enn f.eks. fjernstyring. Autonomi er avhengig av at både sensorsystemer, styringssystemer og beslutningssystemer, samt kommunikasjon mellom disse og eventuelt andre, må være tilstrekkelig langt utviklet (Hareide, et al., 2018). Et sensorsystem kan for eksempel i det enkleste tilfellet bestå av bare en GPS-mottaker og et kamera, som gir data om egen posisjon, fart, retning og bilde av hva som skjer rundt. Styringssystemet er mellomleddet som binder inputene fra sensorene sammen med handlingene til aktuatorene, delvis direkte og delvis avhengig av situasjonstolkningen til beslutningssystemet.

Kommunikasjon er viktig og kan deles opp i to deler, intern og ekstern kommunikasjon. Den eksterne delen er for å kunne sende og motta data mellom fartøy og en basestasjon, som enten er landbasert eller på et annet fartøy. Selv om dette ikke er like viktig for autonome fartøy som for fjernstyrte og semi-autonome fartøy så vil det fortsatt være nyttig informasjon som det autonome fartøyet kan innhente for basestasjonen. Den interne kommunikasjonen handler om den kontinuerlige kommunikasjonen et autonomt system trenger for å opprettholde samarbeidet for bærebjelkene som ligger til grunne for autonomi. Derfor er det lurt med robuste og sikre kommunikasjonsbærere slik at autonomien kan nå høyere nivå. (Hareide, et al., 2018)

Til slutt er det beslutningssystemet som kan være autonomt, men kun i den grad rammene er satt av overordnede målsetninger. «Kjernen i autonomien er et beslutningssystem som kan ta avgjørelser og prioritere, basert på informasjon og et rasjonale som ligger i programvaren» (Hareide, et al., 2018). Samtidig er beslutningssystemet avhengig av informasjon om omgivelsene fra sensorsystemene for å kunne tilpasse seg den reelle situasjonen slik at oppdragene kan gjennomføres. Men hvordan vil dette fungere når flere droner skal delta i en felles målsetning?

Ved å ha flere droner som jobber mot samme mål kan det øke rekkevidde og effektivitet i f.eks. søk- og redningsoppdrag. Det å ha flere enheter aktive samtidig vil øke antall sensorer som observerer ett området. Eneste problemet er at disse ikke kommuniserer eller kan dele informasjon om hva som skjer med hverandre. Dette kan føre til at de jobber alene og dobbelt opp på områder, til tross for å være flere i antall. Dersom dronene fungerer sammen fra start så vil de kunne jobbe mer systematisk med oppdrag sammen.

Med dronesverm vil 3 eller flere droner bli brukt sammen. Disse blir ikke styrt én og én slik vanlig lysshow med droner blir gjort på, men dronene skal følge en gitt adferd som gjenskapes fra naturen. Eksempler på adferder de kan følge er gjeter-adferden, hvor en gjeterdrone leder flokken etter sine premisser. Dersom den endrer retning eller fart skal resterende «følge-droner» tilpasse seg dette. En annen adferd er flokk-adferden som fugler har der det er en frastøtingssone rundt hver drone samtidig som de skal bevege seg i samme retning. I motsetning til gjeter-adferden så er dronene likestilt i denne «flokken» av droner. Her styres alle av samme kommandoer som blir gitt, ikke bare en drone. Dette

samvirke fjerner rom for uklarheter i en gruppe med droner, og krever at alle har samme bilde. Dette er det en tidligere bachelor fra 2019 har basert seg på.

Tidligere er det skrevet to bacheloroppgaver ved FHS Sjøkrigsskolen som omhandler maritime dronesvermer. Konsepter, produkter og erfaringer fra begge disse bacheloroppgavene danner mye av grunnlaget for denne oppgaven.

I 2019 skrev Andreas Handal Hellesnes og Kim André Lyssand oppgaven «Plattform for sverm», hvor det ble utviklet en testplattform for en sverm av maritime overflatedroner. Disse overflatedronene er plastbåter som får sin fremdrift av tre motorer som påvirker hver sin propell. Totalt er det 3 ror, som består av ett hovedror som styrer retningen, med to mindre ror som har mulighet til å løfte båten for å minske vannmotstanden. På båten er det montert en Pixhawk 4 sensorpakke, en arduino og en Raspberry pi. Systemet består av en GPS-mottaker, «autopilot», kommunikasjonssystemer og beslutnings-takere om bord svermenhetene, samt en basestasjon for overvåkning og sentralisert styring. Produktet er en sverm bestående av tre droner som kan kjøre til en gitt posisjon, samtidig som de forholder seg til hverandre med tilsvarende atferd som en fugleflokk (Boids). Denne svermen kan skaleres opp med flere droner, men for å få frem poenget er det nødvendig med minst 3 for at det skal kunne defineres som en sverm. De løfter frem økt miljøoppfatning som mulig videreutvikling av svermsystemet, ettersom dronene ikke har systemer for å oppfatte land eller fartøyer utenfor systemet, bare sin egen og andre svermenhetenes GPS-posisjon. Denne plattformen fungerer fint på åpent havområder med mangel på andre fartøy, og med kommunikasjon til en basestasjon for å kunne motta oppdrag eller posisjonsområdet. Foreløpig er ikke plattformen utstyrt med effektorer som overvåkningsutstyr eller våpen. Men om man skal bruke disse til f.eks. kamikazedroner mangler det kun funksjonaliteten til å oppdage motstandere.

Figur 1-1: Overflatedrone (Sauter, 2019)

For å løse sistnevnte utfordring utviklet Magnus Røsand og Olav Hollup i 2022 et system gjennom oppgaven deres «Implementering av objektdeteksjon på maritim dronesverm».

Der anvender de kameraer montert på dronene for å detektere og identifisere fartøy gjennom bruk av kunstig intelligens, og viser det potensialet som ligger i denne utvidelsen for å gi dronene en større miljøoppfatning enn tidligere. – f.eks. for å oppdage enkeltfartøy i et svermforband. Røsand og Hollup anbefaler å videreutvikle systemet til å kunne benytte deteksjonsmodellen til å drive kollisjonsunnvikelse, men selv om bilde-gjenkjenning ved hjelp av optiske sensorer har flere mulige bruksområder så ligger det store muligheter i å bruke flere datakilder for å oppnå en økt situasjonsforståelse. For å få til dette må slike sensorsystemer være mer integrert i selve svermplattformen.

Med kompliserte systemer som svermplattformen fra 2019 er det en kjent sak at det er stor fare for at programvarer blir utdatert. Dette kan føre til at deler av systemet ikke er kompatible med hverandre, noe som videre kan føre til at deler av systemet ikke evner å dele data med eksterne enheter eller forstå data som blir sendt internt på enheten. Programvare er ofte under kontinuerlig utvikling, og hvis programvaren om bord svermplattformen ikke holdes oppdatert vil det være stor risiko for økende teknisk gjeld med tiden som går. Med større teknisk gjeld vil det bli mer utfordrende å implementere nye funksjoner, og derfor kan det være en fordel å undersøke mer tradisjonelle navigasjonshjelpemidler som øker situasjonsforståelsen. Med flere systemer vil man kunne oppnå en mer komplett situasjonsforståelse, og er dermed rustet til å kunne unngå hindringer av alle slag. Ved siden av optisk navigering er radar og AIS blant de viktigste navigasjonshjelpemidlene.

1.2 Problemstilling

Oppgaven har som mål å implementere AIS-innhenting og radar på systemet Hellesnes og Lyssand produserte i 2019. Gjennom AIS kan systemet detektere fartøy i nærheten av svermen, radar vil kunne detektere land eller ytterligere hindringer på sjøen, og sammen vil disse potensielt kunne øke miljøoppfatningen til svermen betraktelig. Videre vil dette kunne integreres i beslutningstakingen og autopiloten til dronene, slik at den nye sensorpakken blir et tilskudd og ikke en erstatning for adferden som allerede eksisterer.

Å implementere den nye sensorpakken i et allerede eksisterende system kommer ikke nødvendigvis uten utfordringer. Det er fire år siden Hellesnes og Lyssands plattform ble utviklet. Mye av programvaren de brukte kan være utdatert og foreldet, hvor nyere versjoner av programmer ikke nødvendigvis er kompatible med hverandre. Dette har de forsøkt å ta høyde for gjennom et modulært systemdesign, men vesentlig for oppgaven vil likevel være å undersøke i hvilken grad det er mulig å ta et eldre system og videreutvikle det med nye sensorer og utvidet funksjonalitet.

For å utforske mulighetene og utfordringene ved implementering av nye sensorsystemer på testplattformen stiller vi følgende problemstilling:

Hvordan kan AIS og radar implementeres på dronesverm for å drive kollisjonsunnvikelse, og hvilke utfordringer ligger det i å implementere dette i en eksisterende testplattform for dronesverm?

1.3 Avgrensninger

Oppgavens første begrensning er at koden som allerede eksisterer skal endres minimalt. Dette gjøres fordi mye av den koden er basert på *open source software* som gjør at den oppdateres ofte. I tillegg vil mange med mye kompetanse legge ut løsninger på eventuelle problemer som oppstår med denne koden som allerede ligger ute. Noe som også vil forebygge mengden teknisk gjeld som samles opp med tiden.

En annen begrensning er at programmer blir valgt ut ifra hva som fremhever konseptet best mulig fremfor det mest operasjonelle. Man kan ta tilpasninger for å ivareta operasjonelle behov.

Tredje begrensing er at oppgaven er avhengig av å skulle bruke så mye som mulig av plattformen slik den var i 2019. Dette gjøres fordi det vil ta for mye tid å utvikle alt på nytt, samtidig som alle målene skal bli oppnådd.

For det fjerde vil oppgaven forenkle operasjonsmiljøet ved å ha klare skiller mellom hva AIS skal detektere, og hva radaren skal detektere. Ettersom AIS brukes for å hente data om fartøyer i nærheten, vil det forutsettes at alle fartøy rapporterer på AIS. Dermed vil radaren kun behøve å detektere hindringer som er stasjonære eller tilnærmet stasjonære. Dette vil kunne omfatte land, sjømerker, kajakkpadlere med mer.

Dronenes bevegelsesretning er også begrenset. Å hensynta om dronene kjører forover eller akterover vil komplisere både fysisk implementering og kode, og det forutsettes derfor at dronene kun kjører fremover.

Oppgaven vil avslutningsvis begrense seg til implementering på én enhet. Å sette unnvikelsesalgoritmen i system med resten av svermen, altså at svermens kollektive adferd endres basert på én enhets endrede adferd, vil kreve en betydelig om-programmering av koden, noe som ikke vil fokuseres på i oppgaven.

1.4 Mål

Å implementere kollisjonsunnvikelse kan gjøres på flere metoder, og kompatibilitets-testingen av testplattformen kan ha ulike vinklinger og fokusområder. Det kan være nyttig å fortsette med et modulært systemdesign, og gjennom kompatibilitetstesten vil det kartlegges hvor komplisert implementeringen av kollisjonsunnvikelse kan være. Et modulært systemdesign muliggjør at systemene produsert i oppgavene om dronesverm, og eventuelt andre relevante systemer, slås sammen slik at de har sensorsystemer som komplementerer hverandre, eller et mer komplekst system som evner å løse flere og vanskeligere typer oppdrag. For at oppgaven skal gi ønsket resultat er problemstillingen brutt ned til spesifikke mål, som vises i *Tabell 1-1*:

Tabell 1-1: Oppgavens mål

	Mål	**Beskrivelse**
1	Systemforståelse	Få forståelse av hvordan plattformen fungerer og hva som skal til for å få den til å kjøre
2	Få plattform operasjonell	Forberede teknisk utstyr og kartlegge teknisk gjeld
3	Implementere radar	Enheten skal kunne innhente data fra radaren
4	Implementere AIS	Enheten skal kunne innhente AIS-data
5	Lokalisering av hindring	Enheten skal kunne anvende dataen for å detektere om objekter er på kollisjonskurs eller ikke
6	Unnvikelse	Enheten skal kunne endre egen bevegelsesretning i den hensikt å unngå kollisjon med lokalisert hindring

1.5 Struktur

Oppgaven er delt opp i 6 kapitler. Innledningen setter lys på bakgrunnen bak valg av oppgave, konkretisering av oppgaven og linjene vi ønsker å holde oss innenfor. Videre vil det tas opp nødvendig teori som skal bygge opp forståelse for løsning og fremgangsmåte som er blitt valgt. Teorien tar for seg prinsippene for autonomi, droner, dronesverm som bygger opp til relevante svermoppgaver, Radar, Automatic Identification System (AIS), Robot Operating System (ROS) og kompatibilitetstesting.

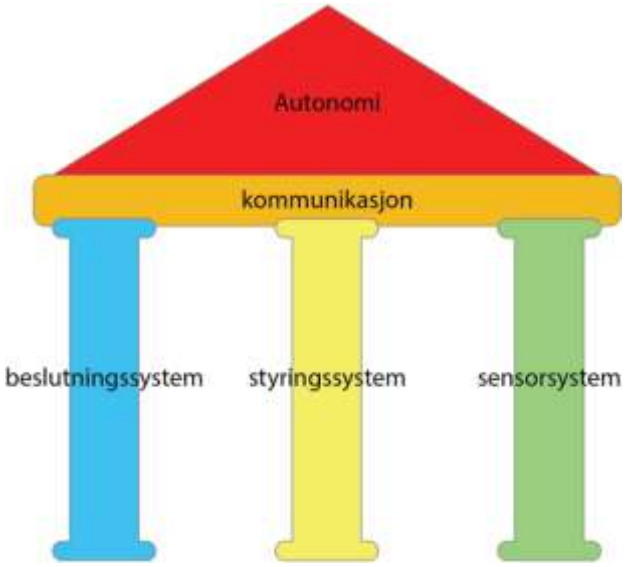
Tredje kapittel handler om implementeringen, og denne er delt inn i fire deler. Første del tar for seg det rent fysiske i og på båtene, komponentene båten består og sensorene som er festet til den. Neste underkapittel omhandler eksterne programmer pakker som har blitt utnyttet. De siste delene omhandler tankegangen og fremgangsmåten i implementeringen av radar og AIS for så å se på hvordan dette kunne implementeres i oppførselen til dronesvermen. Testingen er tredelt og består av kompatibilitet, radar og AIS, som deretter danner grunnlaget for drøftingen som er neste kapittel. Avslutningsvis er det konklusjoner med anbefalinger til videre arbeid.

2 Teori

Dette kapitlet vil fremheve nødvendig kunnskap for å ha forståelse for hvordan vi har gått frem for å videreutvikle svermplattformen og grunnlaget for å implementere AIS og radar. Det startes med en innføring av tidligere sverm-oppgave av Hellesnes og Lyssand fra 2019 (Hellesnes & Lyssand, 2019). Dette er for å gi en forståelse for startpunktet for denne oppgaven og videre forstå hva det snakkes om når det refereres til forskjellige deler av systemet. Det vil også forklares kort fra en videre bygging på svermplattformen i oppgaven til Hollup og Røsand fra 2022 (Hollup & Røsand, 2022) for å vise at det er flere mulige retninger å gå i på svermplattformen. Videre vil det gis informasjon om sentrale tekniske konsepter som gir grunnlaget for å forstå begreper og konsepter som blir presentert i oppgaven.

2.1 Autonomi

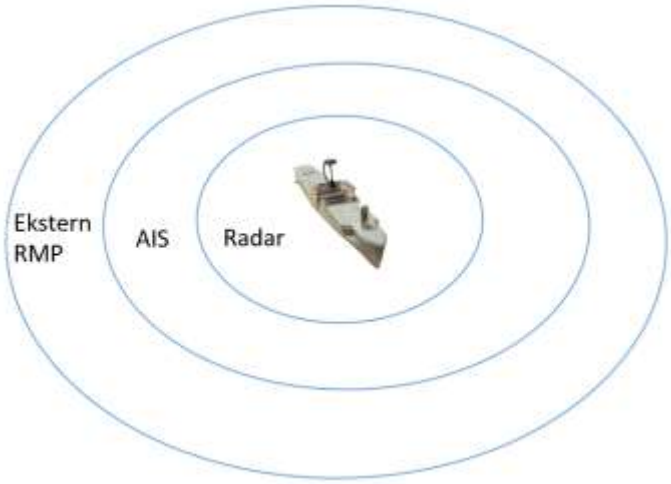
Autonomi er en del av den nære fremtiden, og derfor er det viktig å ikke forveksle autonomi med automatisering og fjernstyring. Som vist i Tabell 2-1***Ulike nivåer av maritim autonomi*** så inngår disse begrepene i autonomi, men forbi dette må også autonome systemer kunne ta beslutninger med menneskelig påvirkning i ulik grad eller ingen påvirkning i det hele tatt. Grunnlaget for å kunne oppnå autonomi består av et sensorsystem, styringssystem, beslutningssystem og kommunikasjon som vist i Figur 2-1Figur 2-1: Bærebjelkene for realisering av Autonomi .



Figur 2-1: Bærebjelkene for realisering av Autonomi (Hareide, et al., 2018)

2.1.1 Sensorsystem

Sensorsystemet består av fartøyets totale sensorpakke som sender inputs til systemet. Ideelt har fartøyet komplimenterende sensorer som overlapper hverandres rekkevidde som fremstilt i Figur 2-2. Med ulike sensorer ønskes data for å kartlegge egen absolutte eller relative posisjon, fart og retning. Mens fra eksterne forhold bør data inneholde eventuelle hindringer som båter, skjær og land for å skaffe et reelt situasjonsbilde. Dette for å gi et bedre grunnlag for å ta gode beslutninger innenfor målsetningenes rammer. (Hareide, et al., 2018)



Figur 2-2: Modell som viser de ulike rekkeviddene til radar som har kortest, AIS som har noe lenger, og Recognized Maritime Picture (RMP) med den største.

2.1.2 Styresystem

Styresystemet er mellomleddet som mottar input fra sensoren for så å sende det videre slik at de riktige aktuatorene utfører korrekte aksjoner. Innenfor navigasjonsstyring blir aktuatorene for styringssystemet retning og pådrag. Selv om tradisjonelt så er det mennesker som styrer det som skjer så har det blitt integrert sikkerhetsfunksjoner der menneskenes kontroll blir overstyrt om de gjør en «feil», som å sette ned farten slik at eventuell farlig situasjon kan bli nedskalert. Denne styring fra mennesker kan være fjernstyring fra land eller overvåking, uansett så vil dette kreve en form for kommunikasjonsbærere for å opprettholde. (Hareide, et al., 2018)

2.1.3 Kommunikasjon

Kommunikasjon i denne konteksten er evnen til å opprettholde en forbindelse mellom fartøy og land, sending og mottakelse av data og datapakker. Samtidig trengs det kontinuerlig kommunikasjon innad i systemet for at de tre bærebjelkene skal kunne samvirke, og for at autonomi på et høyere I kommunikasjon og de to foregående avsnittene, sensorsystem og styresystem, så er redundans viktig for militær robusthet. Samtidig så ønskes det økning av sensorsystem som deretter øker til større mengde data som skal håndteres, noe som fører høyere krav til kommunikasjonsbærerne. Dersom ets systems autonomi blir større vil dette være mindre avhengig av kommunikasjon og kun operere med forhåndsprogrammering og målsetninger. Det som vil ta vurderingene under oppdraget frem til systemet er tilbake vil være beslutningssystemet. (Hareide, et al., 2018)

2.1.4 Beslutningssystem

«Kjernen i autonomien er et beslutningssystem som kan ta avgjørelser og prioritere, basert på informasjon og et rasjonale som ligger i programvaren." (Hareide, et al., 2018)

Som nevnt tidligere er autonomibegrepet relativt og dette er viktig å huske på når det er snakk om systemer som har byttet ut operatøren for datamaskin, fordi uansett hvor autonom systemet er så vil den fortsatt jobbe etter menneske satte målsetninger som definerer oppdraget og innenfor hvilke rammer. Beslutningssystemet vil også kunne ta vurderinger underveis om revurdering av plan eller å avbryte operasjonen, dette på grunnlag av hva sensorsystemet sender av data, og om dette endrer gjennomføringen eller graden av måloppnåelse. Det er viktig å poengtere at autonomi ikke er binært, enten så har du det eller ikke, det er flere nyanser av hvor autonomt et system er. (Hareide, et al., 2018)

2.1.5 De ulike gradene av maritim autonomi

«Autonomi kommer fra det greske ordet «autos» (selv, egen) og «nomos» (lov, regel eller styre), og betyr delvis eller fullstendig selvstendighet, selvstyre eller selvbestemmelse.» (Hareide, et al., 2018). Autonomi er et ord som har hatt en dårlig felles definisjon som det blir sagt i «Fremtidens autonome ubemannede kapasiteter i Sjøforsvaret», og mangelen på definisjon i forsvars sammenheng har også blitt tatt opp til debatt av Gine Rønne

Bolling i 2021 med innlegget «Autonomi – Hva betyr det?». Her fremmer hun at denne mangelen på en felles definisjon åpner rom for misforståelser på hvor stor grad av autonomi man faktisk snakker om, og dette kan hemme utviklingen av autonome systemer og det militær-sivile samarbeidet på denne fronten. (Bolling, 2021)

Denne oppgaven forholder seg til følgende definisjon på autonomi: «Et autonomt system er adaptivt og kan tilpasse seg situasjonen. Det skal kunne håndtere uforutsette hendelser, og er fristilt fra mennesket i utførelsen av operasjonen» (Hareide, et al., 2018). Denne definisjonen kan samtidig brytes ned til flere gradsnivåer av autonomi. *Tabell 2-1* er et forslag på å utdype de ulike nivåene av maritim autonomi laget av NFAS (Norsk Forum for Autonome Skip), noe som kommer til å bli brukt som et verktøy i denne oppgaven.

Tabell 2-1: Ulike nivåer av maritim autonomi **(Hareide, et al., 2018)**

Nivå	Operatørens rolle	Automatisering	Kompleksitet
0	Besetning på broen	Ingen: Direktekontroll av menneske	Få statiske objekter
1	Tidvis ubemannet bro overvåket av kontrollstasjon påland som kan påkalle besetning	Veiledning til operatør, ingen automatisk kontroll	Mange statiske objekter
2	Ubemannet fartøy, kontinuerlig overvåket fra kontrollstasjon på land	Menneskelig tilsyn, automatisk og deterministisk kontroll ved bruk av enkle grenseverdier	Mer dynamisk miljø, ingen begrensninger påfartøyets manøvrerbarhet
3	Periodisk ubemannet bro uten overvåkning fra kontrollstasjon. Systemet kan mønstre bemanningen på fartøyet ved behov.	Automatisk og deterministisk system med lengre og mer komplekst beslutningssystem	Forholdsvis dynamisk miljø, noen begrensninger på fartøyets manøvrerbarhet.
4	Ubemannet med tilsyn fra kontrollstasjon, kontrollstasjon gir ordrer ved behov	Begrenset autonomi – flere, men begrensede valgmuligheter for fartøyets beslutningssystem	Dynamisk miljø, noen begrensninger på fartøyets manøvrerbarhet
5	Ubemannet, ingen overvåkning fra kontrollstasjon på land	Fullt autonomt – ingen begrensninger på beslutningssystem	Dynamisk miljø, begrensninger på fartøyets manøvrerbarhet.

2.2 Droner

Som nevnt i innledningen er droner ubemannede fartøy som kan enten kontrolleres med fjernstyring eller autonomt ved hjelp av forhåndsprogrammert programvare, og

sanntidsdata fra sensorer og GPS. I tillegg er det ikke enten eller med operatørens rolle og automatisering som er vist i tabell 2-1. Droner kommer i mange ulike størrelser, med bredt spekter av komponenter som kamera, radar, IR-sensor og navigasjonssystem er noen eksempler. Med alle disse mulighetene og modifikasjonene kan droner brukes på mange forskjellige bruksområder. Selv om en drone kan ta inn mye data med disse sensorene så vil en drone alene bli noe snaut for å dekke et større område.

2.3 Dronesverm

«En dronesverm kan sammenliknes med et fotballag: Treneren kan lede mye fra sidelinjen, men han kan ikke fjernstyre. De øyeblikkelige beslutningene må tas av dem som er ute på banen.» (Aarønæs, 2022) I dette sitatet gjenspeiler treneren operatørens rolle, og fotballag referansen belyser viktigheten av beslutningssystemet når dronene skal samhandle. Dronesverm-intelligens er bygget på en form for kunstig intelligens (AI), denne bruker algoritmer for å identifisere mønstre i miljøet, slik at dronene kan handle deretter. En dronesverm må også handle etter forhåndsprogrammerte regler slik som en enkelt drone, men i tillegg må en drone ha adferds-programmering som forteller hvordan dronene i svermen skal koordinere oppførselen sin, med hverandre og samhandle med miljøet. Deling av sanntidsdata innad i svermen som hverandres posisjon, og data om omgivelsene legger grunnlaget for beslutningsprosessen. (Frąckiewicz, 2023)

2.4 Relevante oppgaver

I 2019 skrev Andreas Handal Hellesnes og Kim André Lyssand oppgaven «Plattform for sverm», hvor de utviklet et svermsystem for maritime overflatedroner. Systemet består av en autopilot, kommunikasjonssystemer, beslutningstakere og en basestasjon for overvåkning. I 2022 utvidet Magnus Røsand og Olav Hollup dronesystemet gjennom oppgaven deres «Implementering av objektdeteksjon på maritim dronesverm». Der anvender de kameraer montert på dronene for å detektere og identifisere fartøy gjennom bruk av kunstig intelligens.

2.4.1 Svermoppgaven fra 2019

Testplattformen til Hellesnes og Lyssand (2019) er basert på sjødroner konstruert ved FHS SKSK. Disse sjødronene er modifiserte modellbåter der fremdriften styres av 3 motorer som er markert med gul sirkel, disse går til hver sin propell som er på undersiden av båten i rød sirkel. Styringen består hovedsakelig av et hovedror som blir kontrollert av 2 servoer, disse har også kontroll på 2 mindre ror som kan løfte båten opp fra vannet, markert med rød sirkel. Alle disse følger signaler som blir sendt fra en Arduino, som deretter er koblet til en Raspberry Pi (Begge plassert i grønn sirkel). I tillegg er det en Pixhawk 4 som er tilkoblet Raspberry Pi, som var hele sensorpakken. En Pixhawk 4 har blant annet GPS-modul, akselerometer og kompass som gjør at den greier å innhente bevegelsesdata og egen posisjon. Denne innhentede informasjonen blir først bearbeidet av dronen, for så å distribuere det internt til resterende sjødroner i svermen, samt å bli brukt for styring av sjødronen selv.

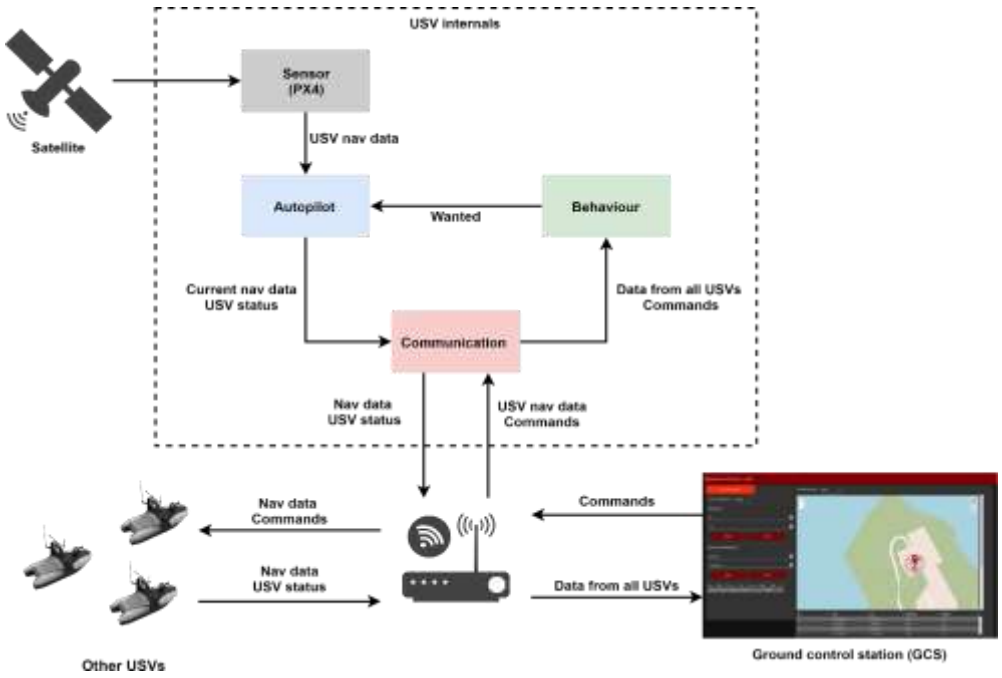
Figur 2-3: Drone fra 2019 med åpent dekk

Ved at disse dronene greier å drive beslutningstaking på grunnlaget av egen og hverandres GPS-informasjon gjør at disse enhetene kan fungere som en sverm. Derfor er kommunikasjonen sentral for at dette skal være mulig, og dette er en av tre deler som utgjør prosessen for hvordan svermplattformen fungerer som system, som vist Figur 2-5: Systemstruktur av delsystemer i svermen . Systemet bygger på teorien om autonomi dele består av PX4 som er sensorsystemet, *Autopilot* som styresystemet, *Communication* som kommunikasjon og *Behaviour* som beslutningssystemet. Etter kommunikasjonsprosessen har hentet ut GPS-data første gang vil den gjennomføre en loop der den konstant oppdaterer dataen. *Autopilot* er en annen hoveddel som også er avhengig av samme GPS-informasjon som *Kommunikasjon*, men vil også trenge ønsket posisjon på dronen som *Behaviour* beregner. Når *Autopilot* har mottatt disse nødvendige dataene så vil den sette nye bevegelsesvektorer ment for aktuatorene, ror og motorer.

Figur 2-4: Sjødrone fra 2019

Bevegelsesvektoren blir satt på grunnlag av beregninger gjort i PID-regulatoren. Denne prosessen vil også fungere i en loop. Dronenes posisjon og bevegelse kan overvåkes i Ground Control Station (GCS). Her er det også mulig å gi kommandoer for setting av oppførsel, fence og punkt det skal seiles mot. GCS er et node red program. Node red er et nettleser-basert kodeverktøy som fungerer med å sette «noder» sammen til et program. Nodene i denne konteksten er som funksjoner i vanlige programmer og kan modifiseres.

I motsetning til de to andre prosessene så fungerer *Behaviour* litt annerledes ved at den allerede har et sett default-data den kan bruke som er lagt inn i programmet. Denne default-dataen fastsetter et sett med startverdier slik at enheten kam samhandle med svermen uten å ha fått kommando fra GCS. Dersom det eventuelt kommer en kommando fra GCS så vil denne overskrive behaviouren som kjørte før kommandoen.



Figur 2-5: Systemstruktur av delsystemer i svermen (Hellesnes & Lyssand, 2019)

2.4.1.1 Behaviour

Som nevnt over er det behaviour som tar seg av utregning av ønsket styringsretning. Derfor vil det her forklare litt grundigere hvordan den fungerer. Behaviour har to forskjellige oppførsler som kan kjøres: BOID og PSO. Disse algoritmene er allerede forklart grundig i oppgaven til Hellesnes og Lyssand så dette blir en kort oppsummering. BOID behaviour går ut på at droner i svermen ønsker å holde seg på en viss avstand fra hverandre og ønsker å gå i samme retning. Dette gjøres ved å kalkulere tre vektorer: samstilling, tiltrekning og frastøting. Samstillings-vektoren er den som får svermen på samme kurs. Tiltreknings- og frastøtnings-vektorene styrer dronene til og bort fra svermen respektivt. Størrelsen på disse vektorene er avhengige av distansen til de andre dronene i svermen og en individuell forsterkelseskonstant som bestemmer hvilket vektorer som skal vektlegges mest.

PSO algoritmen er mer kommunikasjonsbasert. Den går ut på at svermen prøver å finne det beste punktet å være ved at dronene beveger seg basert på hvor dens personlig kjente beste punkt er og hvor svermens kjent beste punkt er i søken etter en ny beste posisjon. Slik fortsetter svermen til et tilfredsstillende punkt er funnet. I dette tilfelle vil personlig beste og svermens beste være basert på signalstyrke, men ettersom ingen sensorer for måling av signalstyrke er montert blir dette simulert. PSO tar også i bruk vektorer for å bestemme retningen den skal gå: nåværende, mot personlig beste, mot svermens beste. Disse har hver sin forsterkelseskonstant som bestemmer hvor store vektorene skal være. Svermens beste vektlegges tyngst fordi der er det kjent sterkest signal, deretter vektes personlig beste litt mindre og til slutt den nåværende retningen vektlegges minst.

2.4.2 Svermoppgaven fra 2022

I 2022 ble dronesvermen videreutviklet av Magnus Røsand og Olav Hollup. Oppgaven deres «Implementering av objektsdeteksjon på maritim dronesverm» anvender kameraer montert på dronene sammen med kunstig intelligens for å detektere og identifisere fartøy og lage en heading til det identifiserte fartøyet. Hovedfokuset deres ligger på implementering av deteksjons-modellen på enhetsnivå og deling av deteksjonsdata over kommunikasjonsnettverket, men noe grad av videre agering fra dronesvermen på individ-

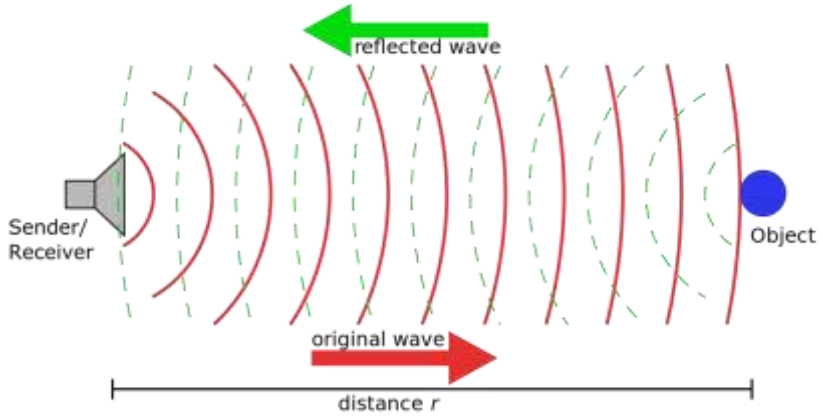
eller systemnivå er ikke implementert. Oppgaven til Røsand og Hollup presenterer et interessant konsept for hvordan svermplattformen kan brukes til innhenting av data til bruk i etterretning. Denne bruken av kunstig intelligens kunne mulig også brukes for kollisjonsunnvikelse, men det har sine begrensninger. Det er mer krevende å trene en kunstig intelligens til dette enn det er å implementere radar og AIS dersom man ønsker samme pålitelighet.

2.5 Radar

Deler av algoritmen for kollisjonsunnvikelse bruker en radar levert av Hi-Link Electronic Company. Dette delkapittelet vil ta for seg grunnleggende radarteknologi og ulike prinsipper som radaren anvender, og derifra brukes for å belyse radarens egenskaper og begrensninger. Delkapittelet vil ikke ta for seg temaer som generering av radarbølger, bølgeledere eller antennefysikk.

2.5.1 Grunnledende om radar

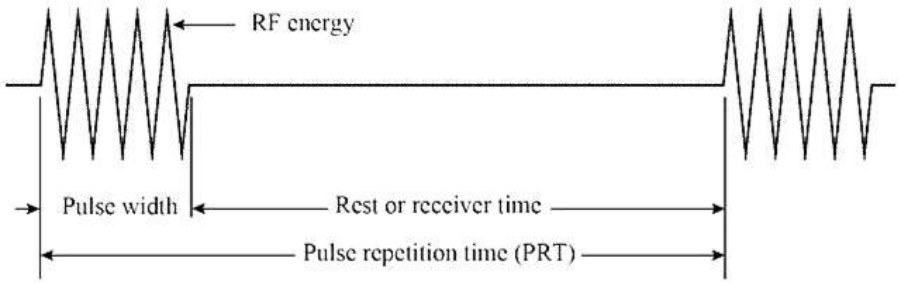
Radar står for Radio Detection And Ranging, og benytter seg av radiobølger for å detektere ulike mål og finne avstanden til målet, samt målets hastighet. En radar består av både en senderantenne og en mottakerantenne, men i praksis vil de benytte seg av samme antenne. Radarprinsippet (Figur 2-6) beskriver hvordan radaren fungerer ved å sende ut en radiobølge, for så å lytte etter et retursignal (ofte kalt radarekko) (Fausa, 2014, s.51).



Figur 2-6: Radarprinsippet (Wikipedia, 2023)

Ved å bruke lysets hastighet *c* og tiden *t* mellom man sender og mottar signalet kan man enkelt regne ut avstanden *L* til et detektert objekt med formelen $L = c\frac{t}{2}$. Det er også mulig å bruke dopplereffekten til å beregne objektets relative hastighet i forhold til en selv (Fausa, 2014, s.51).

Frekvensen til bølgen som sendes ut heter radarfrekvens, hvor signalfrekvensområdet er i mikrobølgespekteret mellom 1 og 40 gigahertz. Pulsbredden τ er varigheten av en utsendt puls, og sammen med hviletiden mellom pulsene utgjør de pulsrepetisjonstid *PRT*, som er den totale tiden fra starten av en puls til starten av neste puls. Sammenhengen mellom pulsbredde, hviletiden mellom pulsene og PRT er vist i Figur 2-7.



Figur 2-7: Pulsrepetisjonstid, pulsbredde og hviletid (Engineering Projects, 2023)

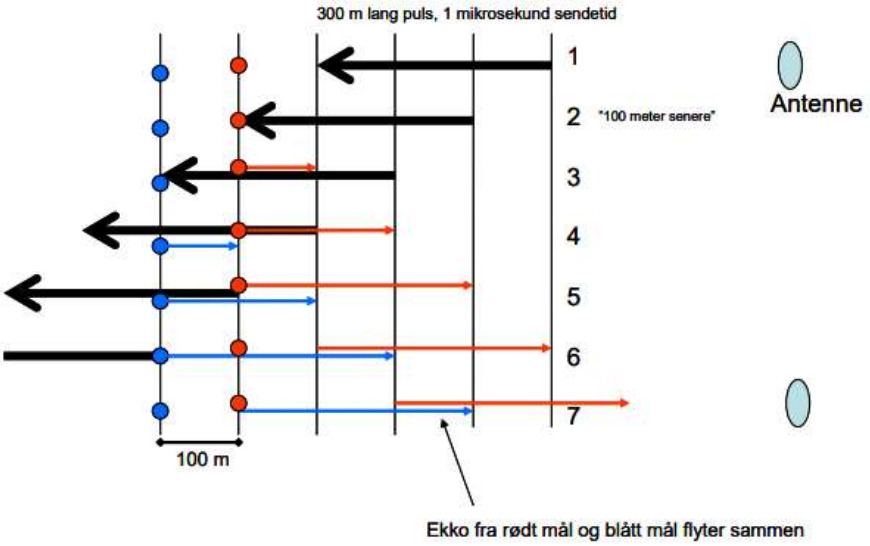
Ut ifra dette kan man også beregne duty cycle, som er hvor stor andel av PRT hvor radaren utsender et signal, og pulsrepetisjonsfrekvens, som er frekvensen på radarpulser (Fausa, 2014, s.53-54).

Radaregenskaper som direktivitet, størrelse på antenner, effekt på utsender og rekkevidde vil alle påvirkes av hvilke verdier man gir de tidligere nevnte variablene. Ikke alle egenskapene er like kompatible, og i flere tilfeller motvirker de hverandre. En radar vil derfor ha følgende begrensninger:

1. **Maksimal avstand uten sammenblanding av pulser** begrenses av tiden et signal bruker fra det sendes ut og returnerer, og pulsrepetisjonstiden. Hvis et signal B sendes ut før signal A har returnert kan ekkoet til A tolkes som om B ble returnert fra et nærliggende objekt. Dette mitigeres blant annet ved å sørge for at tiden

radarsignalet bruker på å returnere er mindre enn PRT. Andre metoder er å la enten bølgefrekvensen eller PRT variere (Fausa, 2014, s. 53-54).

2. **Minste avstand uten sammenblanding av puls og ekko** begrenses når det er kort avstand mellom utsenderen og objektet, hvor det kan være at deler av radarsignalet returnerer før resten av signalet er sendt ut. Da kan det være at radarekkoet tolkes som et objekt veldig langt unna (som i punkt 1), eller at radarekkoet ikke kan «leses» ettersom antennen fortsatt er i sendemodus. For å unngå dette må man sørge for at tur-retur tiden til radarsignalet er større enn pulsbredden.
3. **Rekkeviddeoppløsningen kan begrenses av pulsbredden** ettersom størrelsen på pulsbredden vil begrense radarens oppløsningsevne, altså den minste avstanden mellom to ulike objekter hvor begge kan detekteres. Hvis tur-retur avstanden mellom to objekter med samme peiling relativt til radarantennen er kortere enn lengden på pulsen, vil radarekkoene sammenblandes og objektene vil kunne tolkes som ett større objekt. Dette problemet kan løses ved å ha en lavere pulsbredde, men vil samtidig resultere i en lavere rekkevidde (Fausa, 2014).



Figur 2-8: Illustrasjon av virkningen av pulsbredden (Fausa, 2014)

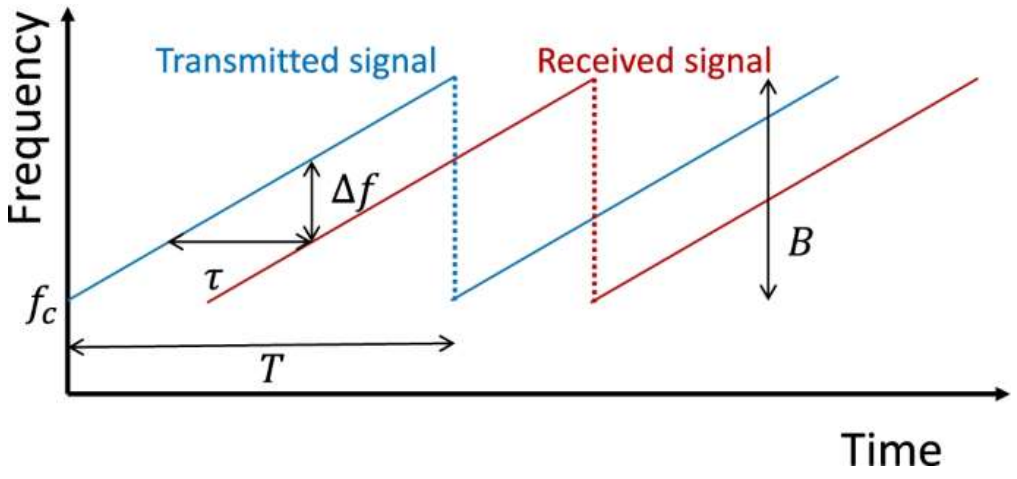
4. **Rekkeviddeoppløsningen kan også begrenses av strålebredden.** Som med pulsbredde avhenger det av avstanden mellom to objekter, men nå i forhold til den horisontale bredden til signalet. Ettersom radaren sender ut signaler med en viss

strålebredde, vil radarekko fra to forskjellige objekter kunne sammenblandes hvis de samtidig berøres av strålen til radaren. Radarutstrålingen er radiell, og dermed vil oppløsningsevnen bli dårligere når avstanden til objektene øker (Fausa, 2014).

5. **Radarfrekvensen** begrenser egenskapene til radaren. Lavfrekvente bølger har mindre energi, vil dempes mindre i atmosfæren og få lengre rekkevidde. Eksempelvis vil en høy radarfrekvens blant annet resultere i en smal stråle, bedre måloppløsning og en mindre antenne, men samtidig få kortere rekkevidde. Radarparameterne må altså balanseres nøye for å ha de egenskapene man trenger (Fausa, 2014).
6. **Andre begrensinger** som er mindre vesentlige i vårt tilfelle. Disse innebærer minste målbare effekt til radaren, jamming av radarsignalet og ingen refleksjon fra målet.

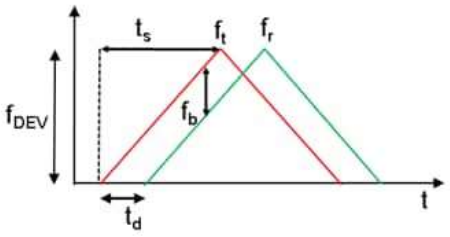
2.5.2 Frekvensmodulert kontinuerlig bølgeradar

Radaren som benyttes i prosjektet er en såkalt frekvensmodulert kontinuerlig bølgeradar (FMCW radar). Ved å frekvensmodulere signalet kan radaren kontinuerlig sende og motta signaler, for frekvensforskjellen gjør det mulig å skille mellom hvilke radarekko som tilhører hvilke utsendte signaler. Dermed omgår man problemene med største og minste avstand uten sammenblanding av puls og ekko, og kan ha en radar som kan brukes på både kort og lengre hold (Radartutorial, 2023). Figur 2-9 viser sending og mottak av FMCW signaler.



Figur 2-9: Eksempel på utsendt og mottatt FMCW-signal

Rekkeviddeoppløsningen til radarer som anvender FMCW er ikke avhengig av pulsbredden slik som beskrevet i forrige delkapittel, men heller båndbredden til signalet (Radartutorial, 2023). Formelen for dette er vist i Figur 2-10, hvor en mindre rekkeviddeoppløsning (R_{RES}) er ønskelig slik at man kan skille mellom to objekter som ligger nære hverandre. I formelen ser man at rekkeviddeoppløsningen har et omvendt proporsjonalt forhold til båndbredden (f_{DEV}). Figur 2-11 viser de omvendt proporsjonale egenskapene til formelen:



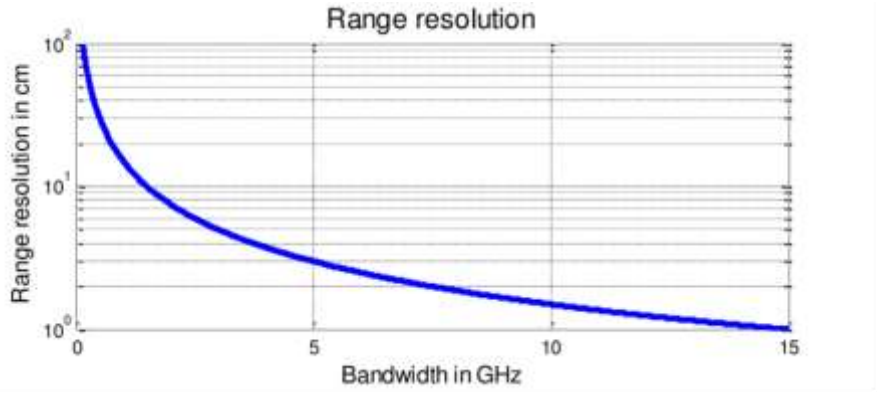
$$R_{RES} = \frac{c}{2f_{DEV}}$$

R_{RES} = *Range Resolution*

f_{DEV} = *Sweep Frequency Deviation*

c = *Speed of Light*

Figur 2-10: Rekkeviddeoppløsningen til FMCW radar



Figur 2-11: Rekkeviddeoppløsning avhengig av båndbredde

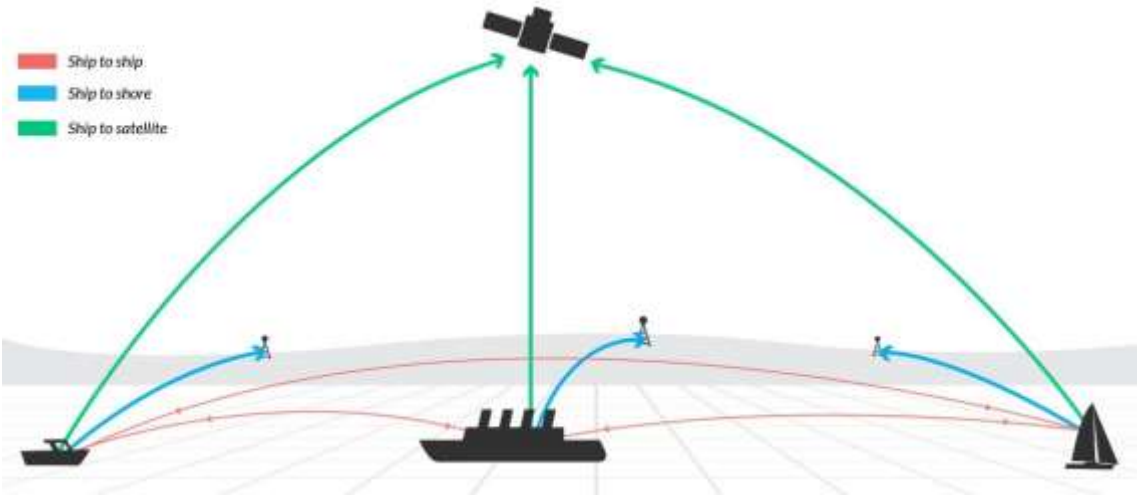
2.5.3 Millimeterbølgeradar

HLK-radaren benytter også millimeterbølger, en relativt ny radarteknologi. Kommersielt anvendes teknologien blant annet i førerstøttesystemer i biler, værradarer og overvåkning av medisinske pasienter (Cadence Design Systems, 2023). Millimeterbølgeradarer opererer som regel med en radarfrekvens på 24, 60 eller 77 gigahertz, en del raskere enn en typisk navigasjonsradar mellom 2 og 12 gigahertz. Begrensning fem i kapittel 2.2.1 nevner at en høyere radarfrekvens vil resultere i smalere stråle, bedre oppløsning og en mindre antenne, og dette har ført til at millimeterradarer ofte er små i størrelsen og brukes til oppgaver som krever relativt kort rekkevidde, men større rekkeviddeoppløsning (Cadence Design Systems, 2023).

2.6 Automatic Identification System (AIS)

«Etter krav fra IMO skal fartøyer over 300 brutto registertonn i internasjonal fart, og/eller fartøy som fører farlig eller forurensende last, ha utstyr for sending og mottak av AIS-signaler» (Kystverket, u.d.).

AIS er et system som er mye brukt i sjøfarten også på mindre fartøy som ikke har dette som krav. Med dette implementert gir det mulighet for utvidelse av sensorsystem, som videre kan bedre evnen til kollisjonsunnvikelse. Dette systemet brukes til å automatisk identifisere fartøyet og dets bevegelse, som inkluderer posisjon, fart og kurs. Andre mulig info som kan opptas via AIS er fartøystype, last, destinasjon og annet, men dette kommer an på fra fartøyet selv ønsker å sende ut av informasjon. AIS mottakeren blir ofte koblet sammen med radar og elektronisk kartsystem (ECDIS) slik at det blir mulig å se alle fartøyene i sanntid med navn. Informasjon mottas fra alle som har AIS sendere, som er fartøy og basestasjoner. Systemet er avhengig av GNSS (Global Navigation Satellite System) og gyrokompass for å fungere så optimalt som mulig. Systemet opererer i VHF-båndet med en rekkevidde på 20-30 nautiske mil. (Kjerstad, 2022)



Figur 2-12: Visuelt hvordan AIS fungerer (NATO Shipping Centre, 2021)

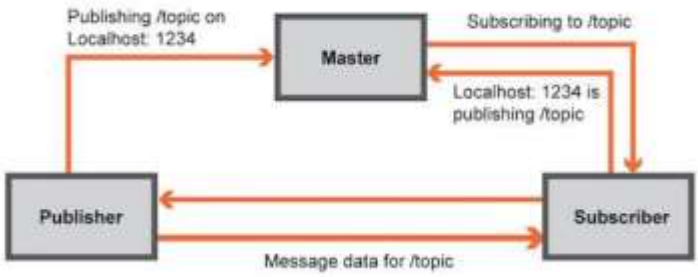
2.7 Robot Operating System

ROS (Robot operating system) er det programmet som brukes for å kjøre mye av koden i dronesvermen. ROS er et «open source»-operativsystem som er laget spesielt for roboter. Det fungerer som de fleste operativsystemer gjør, men de viktigste

funksjonalitetene for dette prosjektet er meldingsoverføring mellom noder og mulighetene til å starte og kjøre kode.

«Nodes are processes that perform computation» (Azab, 2022), altså kan en node være behandling av informasjon fra en sensor mens en annen node kan utregning av retning styreretning. Naturligvis trengs det da kommunikasjon mellom disse nodene for at for eksempel noden som regner ut retning skal få dataen den trenger for å beregne fra sensor noden. Til å få dette til å fungere så trengs det en master. En Master ligner på mange måter en DNS-server. Masteren sin funksjon er å ha oversikt over node og deres plassering og fortelle nodene som skal kommunisere hvor de finner hverandre. Masteren er også ansvarlig for parameter serveren, men den er ikke relevant for oss i denne oppgaven.

Kommunisering mellom noder skjer via publisering og lytting på porter. Dette kan være TCP- eller UDP-porter. «*På bakgrunn av hva msg sier om hvilken data som skal sendes bygger ROS en tunnel som passer akkurat for den dataen … Det tillater også ROS å tilpasse hvordan dataen sendes, enten som UDP eller TCP. Som regel er det UDP …*» (Hellesnes & Lyssand, 2019). Disse portene tildeles en topic som sier noe om hva som finnes på porten. Si for eksempel at du har en radar node og en node for utregning av vektor for denne kontakten. Da publiserer radar noden en melding på en port med en topic som da utregnings noden lytter på og henter inn dataen for å gjøre sine utregninger.



Figur 2-13: Oversikt over kommunikasjonsprosessen i ROS (Lyssand & Hellesnes, 2019)

Figuren over viser hvordan kommunikasjonen mellom publisher-node, Master og subscriber-node. Noden som publiserer meldingen, *Publisher*, sier ifra til *Master* hvilken port den publiserer på og hvilken *topic* dette er. *Master* legger dette til i sitt register.

Subscriber noden trenger informasjonen i denne *topic*'en og spør *Master* hvor den finner det. Den sier til *Subscriber* hvilken port meldingene publiserer på. Da kan Subscriber ta kontakt med Publisher på denne porten og få den nødvendige informasjonen.

2.8 Eksterne programpakker

I denne oppgaven brukes det noen støtteverktøy som har til hensikt å effektivisere og forenkle koden.

Tilnærmet all kode har blitt utviklet i programmeringsspråket Python. Python er blant de mest populære programmeringsspråkene de siste årene. Det har et vidt bruksområde med open source software som et sentralt konsept. På grunn av dette finnes det mange biblioteker som kan gjøre koding i python mer effektivt. Et bibliotek er en samling med kode noen har laget som du selv kan bruke for å utføre komplekse funksjoner med kun noen enkle linjer. I oppgaven har spesielt to biblioteker vært sentrale i koden; geopy og pyais. Geopy er et bibliotek som tilbyr diverse funksjoner tilknyttet lokalisering og utregninger med koordinater. De funksjonene som benyttes fra denne er *distance* som regner ut distansen mellom to koordinater, og funksjonen destination som regner ut koordinaten man ender opp når du har et startpunkt, seilingsretning og distanse. Det andre biblioteket som brukes er pyais. Pyais er et bibliotek som kan hente inn og dekode AIS meldinger. «This library has been used and tested extensively in representative real-world scenarios. This includes tests with live feeds from Spire, the Norwegian Coastal Administration and others» (Richter, 2023).

2.9 Kompatibilitetstesting

Oppgaven denne bacheloren bygger videre på er fra 2019. I løpet av en periode på 4 år er det stor sjanse for at teknologien har utviklet seg, samme med programvarer. Og ved at denne droneplattformen ikke har blitt jobbet med siden da så kan den sees på som et *legacy system*, «et utdatert datasystem, programmeringsspråk eller programvare som av ulike årsaker brukes i stedet for tilgjengelige oppgraderte versjoner.» (Sandberg, 2023) «Legacy» er brukt for å vise til at et system er avhengig av tidligere versjoner eller støtte

som ikke lenger er der i like stor grad. I noen tilfeller av legacy system kan en underliggende årsak være forbundet med alder, men hovedsakelig ikke. For å identifisere integrasjonsmulighetene med nye systemer eller komponenter til et legacy system sjekke kompatibiliteten med disse. For å gjøre dette finnes det to metoder, bakover- og forover-kompatibilitetstesting, og innunder disse er det flere mulige tester å kjøre som versjon-testing, nettleser-testing, maskinvare-testing, programvare-testing, nettverk-testing, enhet-testing, mobil-testing og OS-testing. Noen av disse kan avgjøre om noe kan lett implementere med små modifikasjoner eller om en full modernisering av systemet må til.

Vanligvis blir kompatibilitetstestingen gjort når et produkt er ferdig utviklet og stabilt for å kunne teste og utbedre funksjoner, bredden, sikkerheten og eventuelt finne mulige problemer. Et eksempel på dette er en nettleser som f.eks. Mozilla Firefox og utviklerne ønsker at denne skal se og fungere like godt på alle versjoner av Windows. Hensikten med kompatibilitetstesting er for å sjekke stabiliteten, funksjonene og allsidigheten til plattformen på ulike versjoner av operativsystem, programvare og kode. Både eldre og nyere. (Chernyak, 2023)

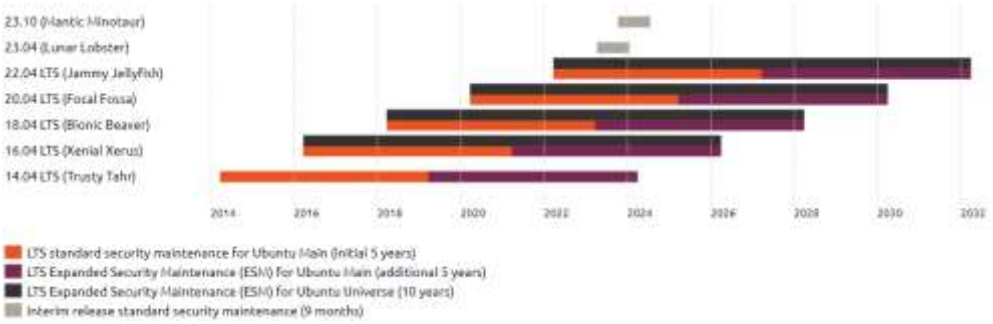
2.9.1 Bakover- og Forover-kompatibilitetstesting

Med bakover-kompatibilitetstesting ser man på hvor bra systemet fungerer med eldre operativsystem, programvare og kodeversjoner utover det stabile utkastet av systemet først var basert på og utviklet for. Kvaliteten på samspillet mellom de moderne og legacy systemene. Det hjelper med å holde eldre systemer relevante lenger og det er ikke nødvendig å bygge et system fra null.

Med forover-kompatibilitet foregår testingen på hensyn av operativsystem, programvare og kode som er nyere eller kanskje ikke har blitt utgitt enda. Ofte er dette en jobb som er ønskelig skal ta så kort tid som mulig når et system er avhengig av for eksempel en nettleser som akkurat har fått en ny oppdatering, og denne nye oppdateringen kan føre til feil i funksjoner eller stabilitet. Ved at tidligere systemer er utstyrt med APIer eller tilleggskode som støtter nye funksjoner uten at det påvirker systemytelsen og ikke må implementeres i hovedprogram eller annet sentralt, vil det være lette med bakover-

kompatibilitet. Dette betyr også at eldre systemer vil være forward-kompatible for nye produkter i tillegg. (Kanade, 2023)

Et annet aspekt som må tas hensyn til i kompatibilitetstesting er livssyklusen til OS og programvare. For eksempel har ROS en tradisjon på å gi ut en årlig utgivelse 23. mai. Disse utgivelsene er tilpasset en spesifikk Ubuntu utgivelse og følger livsløpet til denne.



Figur 2-14: Oversikt over livsløpet til Ubuntu operativsystemer (Ubuntu, u.d.)

Ubuntu sine utgivelser kommer stort sett hvert andre. Disse livsløpene pleier å vare i fem år standard sikkerhet, men de har også 5 tilleggsår med nedsatt støtte. ROS utgivelsen er ment for de fem første årene før den får status som EOL (End of Life). Python har også et femårs livsløp på versjonene sine, der seks versjoner er i ulike prosesser. De tre eldste, men fortsatt aktive versjonene blir passet på sikkerhetsmessig uten at de fikser feil. De to neste som er aktive og offentlige blir passet på samtidig som å bli rettet opp i feil. Til slutt er det siste versjonen som er i utviklingsfasen, og ikke enda offentlig.

Figur 2-15: Oversikt over livsløpet til Python versjoner (Python Software Foundation, 2023)

3 Implementering

Teoridelen har lagt grunnlaget for å forstå konseptene AIS og radar, samt en grunnleggende innføring i relevante begreper. Videre vil dette kapittelet først ta for seg de eksisterende delene av dronene som er vesentlige for videreutviklingen av systemet. Videre vil oppgaven redegjøre for hvordan nye komponenter og programmer er integrert i systemet, og dertil hvordan data fra radaren og AIS hentes, behandles og resulterer i unnvikelsesvektorer. Til slutt forklares det hvordan disse vektorene er innlemmet i *Behaviour*-algoritmene for å kunne endre adferden til dronene.

3.1 Skrog med tilhørende komponenter

Dronene som blir videreutviklet er opprinnelig fra en tidligere bacheloroppgave som omhandlet plattform for dronesverm (Hellesnes & Lyssand, 2019). En av modellbåtene er vist i figur 3-1. Disse dronene er plastbåter som får sin fremdrift av tre motorer markert med gul sirkel, som går til tre propeller som ligger på undersiden av båten i området rød sirkel. Totalt er båten innebygd med 3 ror. Disse består av ett hovedror som styrer, med to mindre ror som har mulighet til å løfte båten slik at vannmotstanden minsker, men dette blir ikke brukt i vår oppgave. Videre har denne modellen en Raspberry Pi, Arduino og Pixhawk 4 med et oppladbart batteri til å gi komponentene strøm. Raspberry Pi og Arduino er innebygd i skroget, vist med grønn sirkel. Pixhawk 4 derimot montert sentralt oppå båten (under dekselet i bildet). Dette er uforandret fra den tidligere bacheloroppgaven. Videreutviklingen med hensyn til maskinvare som har blitt gjort på dronene er å konstruere et hus til å holde radaren på skroget, slik at den skal best mulig sende og motta for å kartlegge omgivelsene rundt (se avsnitt 3.1.4).

Figur 3-1: Drone med åpent dekk

3.1.1 Raspberry Pi

Det benyttes kun én Raspberry Pi av modellen Raspberry Pi 3b+ per drone i dette bachelorprosjektet, for å både kjøre svermprogrammene og behandle data som kommer fra den implementerte radaren og AIS.

Figur 3-2: Raspberry Pi 3 B+ (Raspberry PI, 2023)

3.1.2 Pixhawk 4

Sensorpakken vist i figur 3-3, er gjenbrukt fra tidligere bacheloroppgave grunnet at den fortsatt er god nok til å løse oppgaven som kreves av den, hovedsakelig GPS-posisjonen og kompass for å bestemme orienteringen til båten.

Figur 3-3: Pixhawk 4 med GPS modul (Amazon, 2023)

3.1.3 Arduino

Hver drone har også en mikrokontroller av typen Arduino Uno Rev3 SMD, som vist i Figur 3-4Figur 3-4: Arduino Uno Rev3 SMD . Opprinnelig er Arduinoens eneste funksjon å overføre digital styrings-informasjon til pulsviddemodulasjon (PWM) for motorer og ror, men i oppgaven utvides denne ved å konfigurere to digitale porter for å kunne håndtere seriell kommunikasjon med radaren. Mikrokontrolleren benyttes også til å dele opp dataen radaren sender, slik at de lettere kan anvendes av Raspberry Pien. Dette vil forklares videre i

Figur 3-4: Arduino Uno Rev3 SMD (Arduino, 2023)

3.1.4 Overbygg for radar

For å kunne benytte radaren er det 3D-printet et overbygg som skal huse radar-brikken. Overhuset er vist i Figur 3-5 og Figur 3-6, og er festet i baugen på dronen. Radaren er orientert med en relativ peiling på 0˚ i forhold til kjøreretning og dekket på dronen for å

Figur 3-5: Overbygg sett ovenfra

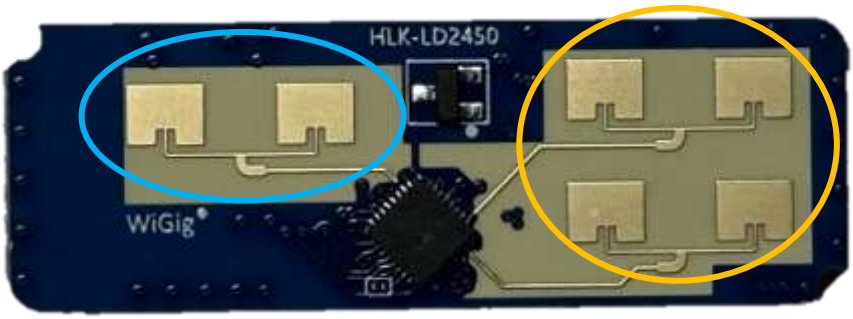
Figur 3-6: Overbygg med radar sett forfra

best mulig kunne detektere potensielle hindringer dronen møter på. Radaren er løftet to centimeter over dekket for å ha bedre sensordekning i området rett foran baugen. Foran radaren er det festet en beskyttelsesskjerm i plast for å skjerme mot elementene. Plastskjermen er så tynn at den vil ha minimal innvirkning på radarens egenskaper.

3.2 HLK-LD2450 radar

For å detektere land og andre hindringer på sjøen benytter oppgaven seg av en radar produsert av Hi-Link Electronic Company. Radaren anvender frekvensmodulerte kontinuerlige bølger (delkapittel 2.5.2) og millimeterbølger (delkapittel 2.5.3). Figur 3-7 og Figur 3-8 viser forsiden og baksiden av radaren. På forsiden er mulig å koble seg til med en seriell plugg (markert rødt) eller med en mer primitiv pin-tilkobling (markert grønt), hvor. På baksiden ser man to senderantenner (markert blått) og fire mottaker-antenner (markert oransje).

Figur 3-7: Forsiden av HLK-LD2450



Figur 3-8: Baksiden av HLK-LD2450

3.2.1 Funksjon og behandling av verdier

Radaren opererer i 24GHz ISM-båndet. Dette betyr at den har en radarfrekvens på 24,05 gigahertz og båndbredde på 200 megahertz, som resulterer i et frekvensbånd fra 24,05 til 24,25 gigahertz. Radaren har en horisontal strålebredde på 120 grader, og en vertikal strålebredde på 90 grader, med maksimal deteksjonsavstand seks meter (Hi-Link, 2023).

Radaren analyserer mottatt retursignal og evner å identifisere opptil tre mål. Så sendes måldataen videre til Arduinoen gjennom en seriell port. Dataen sendes som *rammer*, hvor hver ramme består av 32 bytes. Med en *baud rate* (tilsvarer i dette tilfellet bits/sekund) på 256000 resulterer det i en hastighet på 10 rammer per sekund. Tabell 3-1 viser oppsettet til en ramme, eksemplifisert med Figur 3-9Figur 3-9: Eksempel på en ramme sendt fra radaren Figur 3-9. Hver ramme har en fast sekvens bytes kalt *header* og *tail*, som brukes for å markere starten og slutten av rammen. I figuren er disse farget grønt. I midten av rammen er det 24 bytes som brukes for informasjon om detekterte objekter, hvor det er tre delrammer med åtte bytes for hvert objekt. I figuren er delrammene for objekt 1, 2 og 3 farget henholdsvis blått, rødt og **svart**.

Tabell 3-1: Oppsettet av en ramme sendt fra radaren (Hi-Link, 2023)

Frame header	Intra-frame data	End of frame
AA FF 03 00	Goal 1 information Goal 2 information Goal 3 information	55 CC

Figur 3-9: Eksempel på en ramme sendt fra radaren (Hi-Link, 2023)

Radaren bruker little-endian byte rekkefølge, som vil si at den sender den minst signifikante byten først, for så å sende dem i stigende rekkefølge. Motsatt av little-endian er big-endian, som er den «vanlige» måten å lese bytene på hvor den største verdien kommer først (Scheel, 2023). Ved detektering av *frame header* og *tail* vil ikke dette ha

noe påvirkning, men ved konvertering fra bytes til numeriske verdier vil rekkefølgen på bytene ha avgjørende rekkefølge for å få riktig resultat. Som vist i Tabell 3-2 består delrammen av fire ulike deler som alle er datatypene int16 eller uint16. Disse dataypene består av to bytes, og hvis delramme 1 (merket blått) i Figur 3-9 deles i par med to bytes for så å bytte fra little- til big-endian vil delrammen være «**03 0E 86 B1 00 10 01 40**».

Tabell 3-2: Oppsettet av en delramme fra radaren (Hi-Link, 2023)

Del av delrammen	**Infotype**	**Datatype**	**Fra eksempel (little-endian)**	**Bytes i big-endian**	**Numerisk verdi**	**Enhet**
X-koordinat	Posisjon	Int16	0E 03	03 0E	-782	mm
Y-koordinat	Posisjon	Int16	B1 86	86 B1	1713	mm
Hastighet	Hastighet	Int16	10 00	00 10	-16	cm/s
Avstands-oppløsning	Usikkerhet	Uint16	40 01	01 40	320	mm

Nullreferansen til posisjon og hastighet er radarens egen posisjon og fart. Avstandsoppløsningen er den minste avstanden mellom to objekter må ha for å ikke oppfattes som ett mål, og oppgis som radiell avstand fra objektet som er detektert. Figur 3-10 er et utklipp fra radarmanualen, og viser hvordan produsenten konverterer fra int16 og uint16 til de numeriske verdiene til posisjon, fart og avstandsoppløsning:

The process of converting the data of target 1 into relevant information is demonstrated as follows:

Objective 1 x-coordinate: 0x0E + 0x03 * 256 = 782

0 - 782 = -782 mm

Objective 1 y-coordinate: 0xB1 + 0x86 * 256 = 34481

34481 - 2^15 = 1713 mm

Goal 1 speed: 0x10 + 0x00 * 256 = 16

0 -16 =-16 cm/s

Target 1 distance resolution: 0x40 +0x01* 256 = 320 mm

Figur 3-10: Produsentspesifikk konvertering av int16 til numerisk verdi (Hi-Link, 2023)

Verdt å merke seg er at radaren bruker en unormal metode for konvertering fra int16 datatype til numeriske verdier. Manualen oppgir at den numeriske verdien er positiv når den mest signifikante biten (MSB) er 1, og at den er negativ når MSB er 0 (Hi-Link, 2023). Dette strider stikk imot vanlig praksis, og det virker derfor som radaren bruker en produsentspesifikk koding for håndtering av negative verdier. Ettersom uint16 ikke har noe fortegnsbit er metoden for å konvertere denne normal. I Figur 3-10: Produsentspesifikk konvertering av int16 til numerisk verdi er utregningen gjort vanskeligere enn nødvendig, ettersom den numeriske uint16 verdien regnes ut først før det tas høyde for MSB og fortegn. En enklere metode vil være å skrive ut bytene binært i big-endian rekkefølge, så ta høyde for MSB og til slutt regne ut de resterende 15 bitene til en numerisk verdi. Metoden minner mest om «signed magnitude», og er eksemplifisert med x- og y-koordinatene til *Objekt 1* i Figur 3-11:

Objekt 1 x-koordinat: 0E 03 (little-endian) => 03 0E (big-endian)

$03\ 0E_{16} = 0000\ 0011\ 0000\ 1110_2 = -782_{10}$

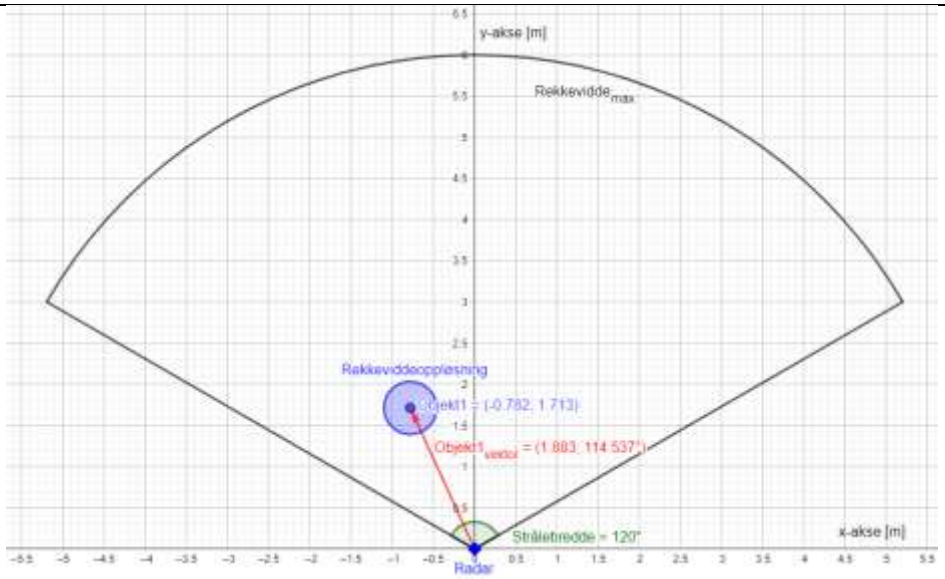
Negativ 782

Objekt 1 y-koordinat: B1 86 (little-endian) => 86 B1 (big-endian)

$86\ B1_{16} = 1000\ 0110\ 1011\ 0001_2 = 1713_{10}$

Positiv 1713

Figur 3-11: Intuitiv konvertering fra int16 til numerisk verdi

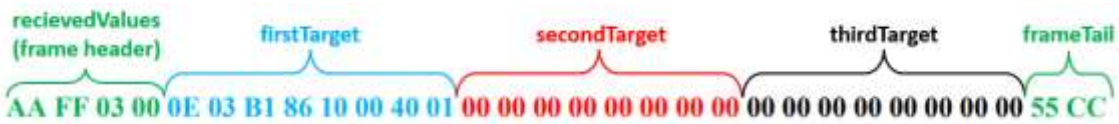


Figur 3-12: Visualisering av eksempeldata for radar

Figur 3-12 er en fremstilling av *Objekt 1* fra eksempeldataen til produsenten. Radaren er plassert i origo ettersom den er dens egen nullreferanse, og eksempelobjektets numeriske verdier fra Tabell 3-2 er plottet innenfor radarens horisontale strålebredde og maksimale rekkevidde. Radaren har også en vertikal strålebredde på 90 grader, men ettersom den ikke gir noen z-koordinater for objektene den detekterer er det rimelig å anta at den komprimerer et tredimensjonalt bilde ned til det todimensjonale horisontalplanet. På figuren er også objektets posisjonsvektor ($Objekt1_{vektor}$) tegnet inn, og denne vil benyttes senere i delkapittel 3.2.3 for å konstruere en unnvikelsesvektor. Før det kan gjøres, må rådataen som kommer fra radaren struktureres i delrammer.

3.2.2 Prosessering av seriell informasjon

Den serielle informasjonen blir sendt fra radaren til Arduinoen gjennom en digital port konfigurert for seriell kommunikasjon. Arduinoens oppgave er å prosessere den serielle informasjonen og strukturere den i delrammer. Programmet som kjøres på Arudinoen ligger i Vedlegg H – Radartest 3 Arduinokode: header, tre mål og tailog forklares ytterligere i delkapittel 4.2, men resultatet er følgende:



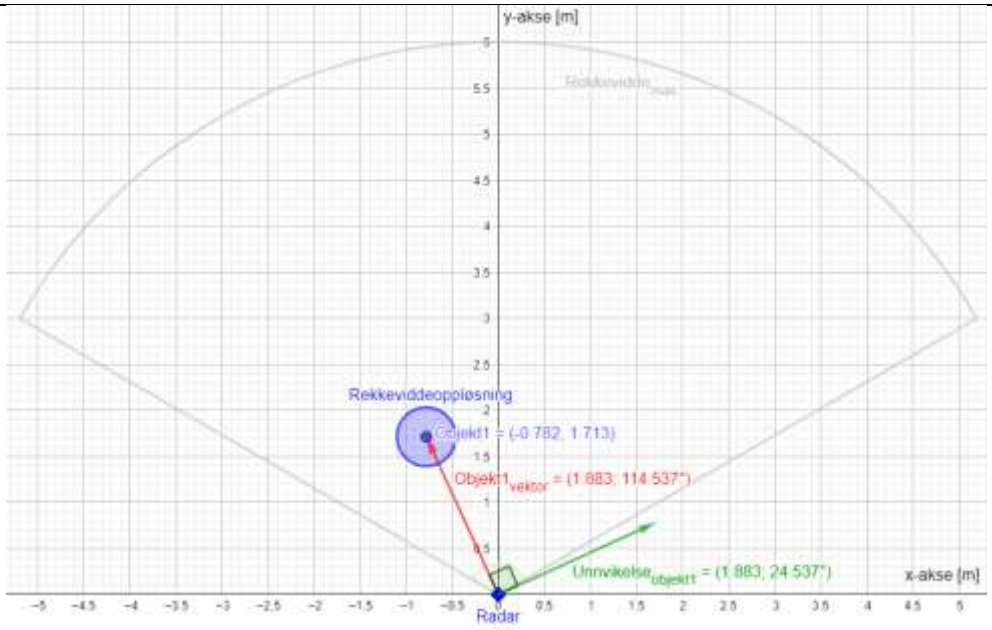
Figur 3-13: Oppdelingen av datarammen etter programmet har kjørt

Figur 3-13 viser hvordan datarammen er delt opp i delrammer for hvert objekt etter programmet har kjørt. For å oppnå denne oppdelingen starter programmet starter med å opprette en array med navn *targetSequence*, som inneholder verdiene til *frame headeren*. *TargetSequence* sammenlignes med arrayen *recievedValues*, som inneholder verdiene som sendes fra radaren, og oppdateres fortløpende. Når *recievedValues* inneholder en sekvens på fire bytes som er lik *targetSequence* vil programmet ha funnet rammens *frame header*, som betyr at de påfølgende verdiene er informasjon om detekterte objekter. Programmet vil derfor gå videre til neste del, hvor det bruker arrayene *firstTarget*, *secondTarget* og *thirdTarget*. Disse tilsvarer delrammene til hvert objekt som radaren kan detektere. Videre vil programmet legge de påfølgende verdiene inn i arrayene for henholdsvis første, andre og tredje mål. Når dette er gjort er syklusen endt, og programmet begynner på nytt med å lete etter *frame headeren*.

Gjennom USB-tilkoblingen kan Raspberry Pien kontinuerlig hente oppdaterte arrays fra Arduinoen. Hvis en array kun inneholder nullverdier betyr det at ingen objekter er detektert. Informasjon om detekterte objekter vil brukes for å lage unnvikelsesvektorer som påvirker dronenes *Behaviour* algoritme, som gjør at dronene kan unnvike eventuelle hindringer.

3.2.3 Unnvikelsesvektorer basert på detekterte hindre

For å kunne implementere kollisjonsunnvikelse på dronene er det nødvendig å tilpasse dronenes *Behaviour*. Dette gjøres ved å implementere unnvikelsesvektorer basert på detekterte objekter og AIS-kontakter, som vil henholdsvis behandles i dette kapittelet og i delkapittel 3.3.4. For å konstruere vektorene henter Raspberry Pien arrayene med objekt-informasjon fra Arduinoen, hvor det er spesielt er x- og y-koordinatene til hindringene som er mest vesentlige ettersom radaren er tiltenkt å kun detektere stasjonære objekter (som nevnt i kapittel 0). Potensielt kan det være tre unnvikelsesvektorer av gangen, en for hvert detekterte objekt.



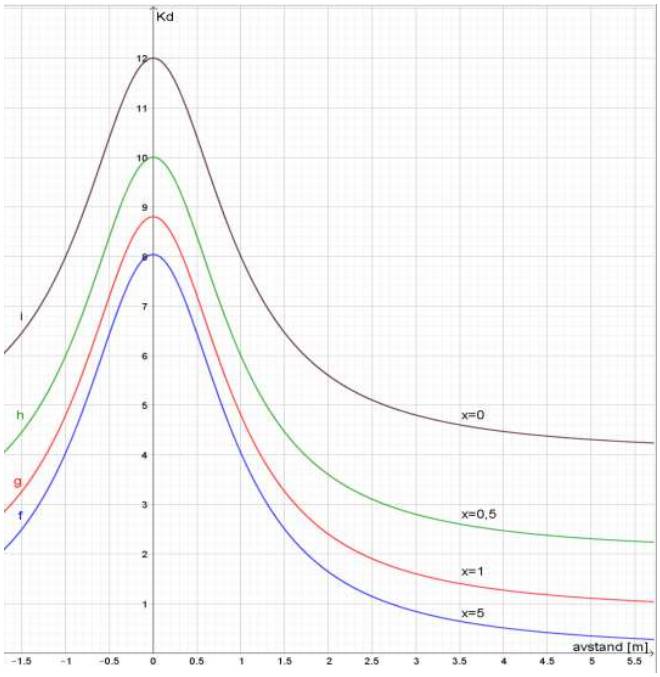
Figur 3-14: Dronens unnvikelsesvektor 90 grader vekk fra objektet

Ettersom posisjonskoordinatene gitt av radaren er relative til egen posisjon, vil koordinatene også gi verdiene for posisjonsvektoren til objektet, som vist i Figur 3-14: Dronens unnvikelsesvektor 90 grader vekk fra objektetFigur 3-14. Videre vil det lages en unnvikelsesvektor i dronens bevegelsesretning som står vinkelrett på objektets posisjonsvektor, og dermed vil lede dronen bort fra kollisjonsfaren. Verdt å merke seg er at vektoren vist i Figur 3-14: Dronens unnvikelsesvektor 90 grader vekk fra objektetFigur 3-14 kun er ment for å illustrere retningen til vektoren, ikke størrelsen.

Størrelsen på vektoren bestemmes av avstanden mellom dronen og objektet, samt hvor nær dronens kurslinje objektet er. Faktoren som bestemmer størrelsen på vektoren, er gitt ved formel 3.1:

$$Kd = \frac{2}{0.25l^2 + 0.25} + \frac{1}{0.25 + x^2} \quad (3.1)$$

I formelen er variabelen *l* lengden på posisjonsvektoren til objektet ($l = \sqrt{x^2 + y^2}$) og *x* er avstanden fra objektet til dronens kurslinje. Formelen har to ledd med omvendt kvadratiske egenskaper for at mindre verdier av *l* og *x* skal gi større utslag samt for å forhindre negative verdier. Figur 3-15 eksemplifiserer hvordan formel 3.1 avhenger av *l* og *x*:



Figur 3-15: Faktoren Kd avhengig av avstand til objekt. Forskjellig farge markerer forskjellige avstander fra objektet til dronens kurslinje

X-aksen på figuren er avstanden *l*, og y-aksen er verdien til Kd, faktoren som bestemmer størrelsen på unnvikelsesvektoren. De forskjellige fargene viser funksjonen ved forskjellige verdier av *x*. Blå linje viser et objekt i randsonen til radaren, rød og grønn linje viser et objekt henholdsvis én menter og en halv meter unna kurslinjen, og sort linje viser funksjonen når et objekt er rett forut for dronen. Sett i sammenheng ser man at

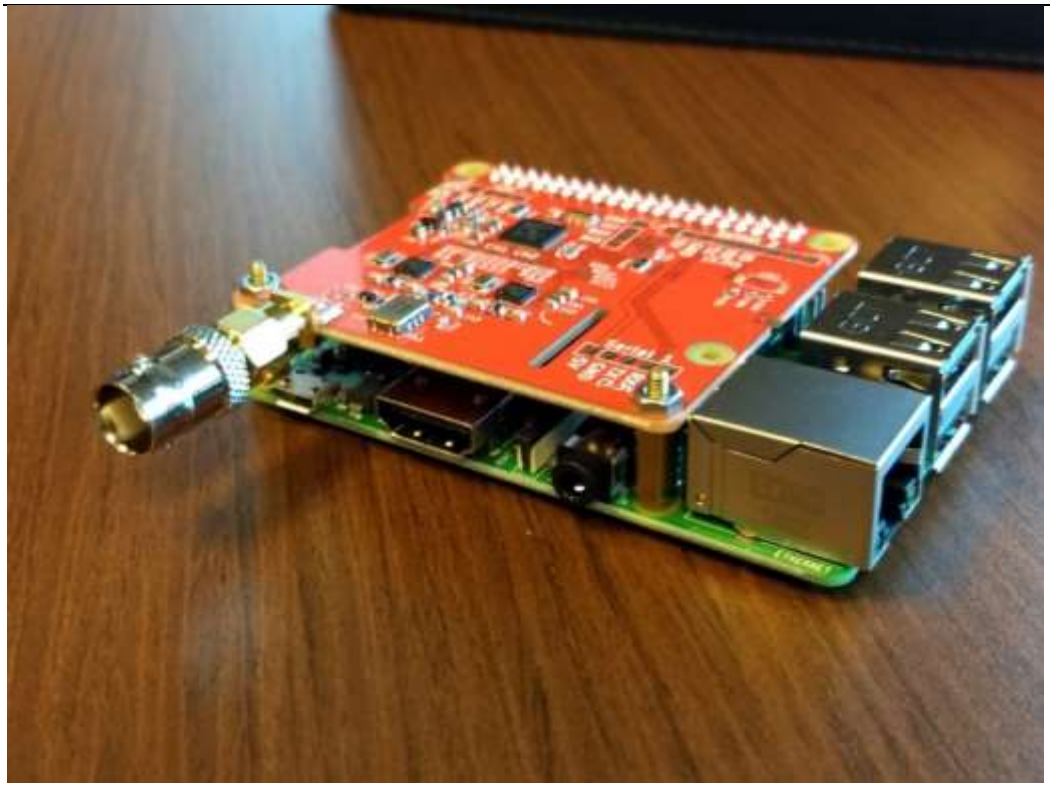
objekter som generelt sett er nære dronen vil gi en sterkere unnvikelsesvektor, mens objekter lengre unna vil ha liten påvirkning. I tillegg ser man at objekter i nærheten av kurslinjen også vil gi en sterkere vektor, og at dette leddet gir særlig utslag når avstanden til kurslinjen er under én meter.

3.3 Bruk av AIS for unnvikelse av fartøy

AIS er som beskrevet i delkapittel 2.6 et system større fartøy bruker til å dele blant annet sin posisjon, retning og fart. Systemet er en viktig del av sikker navigasjon på sjøen. Ut ifra AIS kan man enklere planlegge seilingsmønstret sitt for å unngå uønskede hendelser med andre fartøy. For å få inn AIS dataen trenger man en AIS-antenne og mottaker. Disse kommer i mange størrelser og prisklasser. Dessuten er det mulig å hente aktuell AIS-data via internett fra Kystverket. For dette prosjektet ble det sett på to konkrete alternativer som reelle med tanke på budsjett og størrelse relativ til dronen.

3.3.1 AIS-mottaker

Et alternativ var en AIS-mottaker med navn dAISy HAT. «*The dAISy HAT is the perfect AIS reciever for your Raspberry Pi projects and embedded applications*» (Wegmatt, 2023). Ettersom dronene benytter en RPI som til å kjøre systemet, så har denne mottakeren potensiale for å enkelt kunne implementeres i prosjektet grunnet at den kan sende dataen direkte til RPI via en seriell port.



Figur 3-16 dAISy hat montert på en RPI (Kessler, 2023)

Ulempen med denne mottakeren er at den produseres i USA, noe som medfører lang leveringstid. I tillegg ville monteringen av denne kreve et redesign av vanntettpakningen for å få tilgang til en ekstra seriell port, på samme måte som radaren. Antennen som monteres på denne ville også hatt mye å si for rekkevidden for mottatt av AIS, spesielt med tanke på at båten ligger lavt i vannet. «The range of reception can be variable and is dependent on factors including … the height of the transmitting and receiving antenna and the strength of the vessel transmitter» (NATO Shipping Centre, 2021). Det er uvisst om rekkevidden vil være av betydning ettersom det kun er av interesse med fartøy som er ganske nærme dronene. En liten ulempe er også at monteringen av mottakeren og antenna vil kreve et redesign av beskyttelsesdeksel og montering av antenne. En fordel med dette alternativet er at mottakelsen av AIS-data ikke er avhengig av internettdekning.

3.3.2 Data fra Kystverket

Det andre alternativet var å benytte oss av kystverket sin sending som sender ut AIS-data på en adresse via en TCP-protokoll. «Alle kan få tilgang til AIS-data fra fartøy innenfor

norske farvann, dette inkluderer norsk økonomisk sone, fiskevernesonen ved Svalbard og vernesonen ved Jan Mayen.» (Kystverket, 2023). Denne dataen hentes inn fra basestasjoner og satellitter før det distribueres via internett. Den åpne kanalen sender ikke ut AIS-meldinger fra fiskefartøy under 15 meter og ikke fritidsfartøy under 45 meter. (Kystverket, 2023) Disse vil foreløpig måtte bli detektert via radaren. Det er derimot mulig å få tilgang til høyere oppløst AIS-data som inkluderer flere meldering per sekund og som inkluderer også små fartøy. Det kan vurderes for senere bruk.

Denne oppgaven tar for seg to mulige måter å hente inn dataen fra Kystverkt. Den ene handler om å benytte oss av node-RED og en TCP node for å lytte på kanalen og ta imot TCP-meldingene, for så å bruke en ais node for å dekode AIS-meldingen og gjøre den til et JSON format. Deretter kan man bruke multicast for å sende denne meldingen til alle dronene. Ulempen med denne fremgangsmåten er at det potensielt kan være mange meldinger som skal sendes og mottas, ettersom multicast også brukes til å sende ut meldinger fra autopiloten samt å sende meldinger fra basestasjonen ut til dronene. En annen muligheten er å benytte seg av et python bibliotek for å motta og dekode AIS meldingene til lesbare python «dictonaries». Dictonaries er objekter som inneholder flere egenskaper som man kan gi en verdi til. I dette tilfellet egenskapene «msg-type», «latitude», «longitude» og så videre. Fordelen med denne fremgangsmåten kontra node-RED er at da slipper man å sende informasjonen noe sted ettersom vi kan behandle den internt. En ulempe med denne metoden er at python programmet vil bare hente inn de AIS-meldingene som blir sendt ut i det programmet kjører. Dette skaper en mulighet for å gå glipp av meldinger som kan være kritiske å få inn. Felles for disse to handlingsmåtene er ulempen ved at de er avhengig av internett-tilgang for å få inn AIS-data.

Valget falt her til slutt på å ta i bruk kystverket sin utsending til å få tak i AIS-dataen og benytte python biblioteket pyais for å hente inn og dekode den. Dronene er allerede avhengige av et wifi-nettverk for å utveksle meldinger og kommandoer. Dette nettverket har allerede internett så det var ikke nødvendig med å utvide med mer hardware. En annen grunn er at leveringstiden på dAISy HAT ville gjøre at utviklingen av programmene måtte vente til mottakeren har blitt mottatt, for så å se hvordan dataen blir sendt og mottatt.

3.3.3 Prossesering av AIS-data

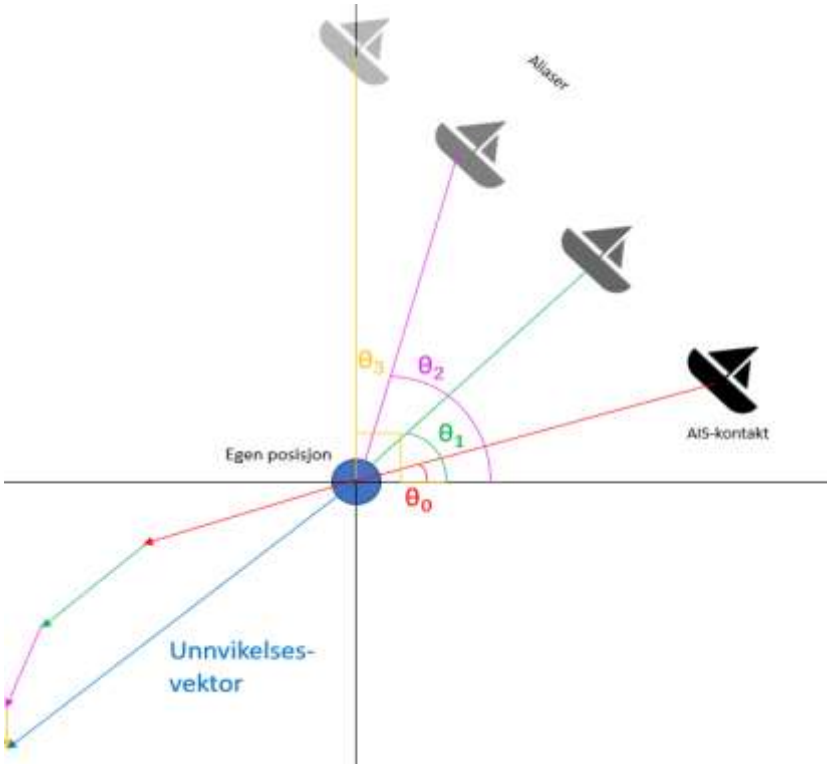
Hele programmet starter med å hente inn AIS-data ved å bruke en TCP-lytter fra biblioteket pyais som hører på IP 153.44.253.27 og port 5631 som er der Kystverket tilgjengeliggjør AIS dataen. Deretter dekodes meldingene ved hjelp av decode() funksjonen som følger med i samme bibliotek. Da har programmet alle meldingene som kommer inn fra Kystverket i form av flere «dictonaries». Ut ifra disse brukes «message type» til å filtrere bort de meldingene det ikke er behov for. Programmet har kun bruk for meldinger av typen 1, 2, 3, 18 og 19 ettersom disse inneholder data som er relevante for hvordan dronen skal forholde seg til fartøyet. Deretter brukes egenskapene «latitude» og «longitude» for å filtrere bort meldinger som kommer så langt borte ifra dronen at de ikke er nødvendig å ta hensyn til. Da gjenstår dataen på de relevante kontaktene og kan brukes til utregning av unnvikelsesvektorer.

3.3.4 Beregning av unnvikelsesvektor

AIS_class er filen som klassen avoidance_vectors blir laget. Denne instansen av klassen trenger input-verdiene posisjon (latitude,longitude) , fart og retning, for å opprettes. Disse verdiene får den fra AIS-meldingen. Når instansen opprettes så kjøres «__call__» funkjonen.

Fra AIS-meldingene hentes kontaktens posisjon, fart og retning ut og oppretter en instans av klassen AIS_class. Instansen av klassen brukes til å kjøre dens egen «__call__» funksjon. Denne funksjonen regner først ut hvor båten kommer til å være ved faste fremtidige tidsintervaller, basert på fartøyets aktuelle kurs (posisjon, fart og retning). Her har vi valgt verdiene 1, 5 og 10 minutter. Dette gjøres ved å først konvertere farten til fartøyet fra knop til m/s for så å gange det med tidsintervallene i sekunder for å få antall meter for så å konvertere til kilometer som brukes som enheten for posisjonsberegninger i programmet. Når vi da har både distansen og retningen fartøyet går i så kan man regne ut koordinatene til der fartøyet kommer til å være. Den nye posisjonen regnes ut ved hjelp av python biblioteket geopy sin funksjon destination som tar startkoordinat, distanse og retning som input verdier og returnerer koordinatene til det nye punktet.

Listen med posisjoner på hvor fartøyet kommer til å være sendes da som input til en funksjon som lager unnvikelses-vektorene. Disse vektorene lages ved å regne ut den relative vinkelen mellom punktene som brukes til å lage en motsatt rettet vinkel fra denne. Det legges også ved en liste med distanser fra eget fartøy til de nye punktene, samt en liste med avtagende verdier [1, 0.5, 0.33, 0.25]. Sistnevnte er for at de punktene som er nærmest i tidsperspektivet skal ha mest å si for hvordan man velger å unnvike.



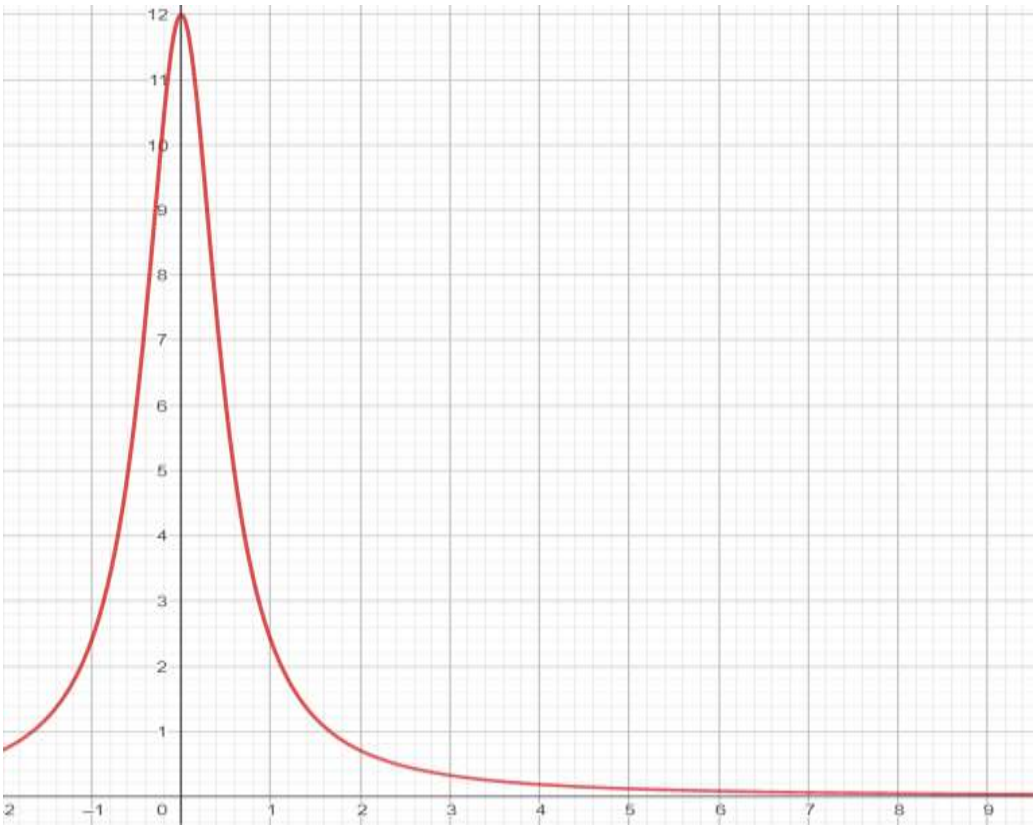
Figur 3-17: Illustrasjon av konseptet for unnvikelsesvektorene. Vinklene θ_n brukes til å gi en motsatt rettet vinkel for vektoren, mens distansen og nærheten i tid bestemmer størrelsen til vektoren.

(3.2)

Størrelsen på vektoren bestemmes av om hvor langt frem i tid punktet vektoren lages ut av er, og hvor lang distanse det er til objektet. Faktoren for tidspunktet er som nevnt listen med avtagende verdier, mens faktoren for distansen er gitt ved formelen:

$$Kd = \frac{3}{d^2 + 0.25}$$

Formel 3.2 er forsterkningsfaktoren Kd for unnvikelsesvektoren der *d* er avstand fra egen posisjon til en koordinat. Figur 3-18 er en grafisk fremstilling av formel 3.2:



Figur 3-18: Funksjon for vektorlengde basert på distanse til fartøyet

Denne funksjonen er valgt for å gi et kraftig utslag for fartøy som er i kollisjonsfare. Denne må også være såpass kraftig slik at den klarer å nesten alene bestemme retningen når den legges sammen med de andre vektorene fra «behaviour». Den gir også veldig lite utslag for fartøyer langt borte, noe som er ønskelig for å unngå unødvendige unna manøvrer.

3.3.5 Betraktninger

Underveis i implementeringen av AIS-unnvikelse så ble det oppdaget at pyais kun er kompatibelt med python3.7 eller nyere, som nevnt i avsnitt 4. Derfor skrives vektorene

til en fil i stedet for å sende de som et topic på ROS fordi ROS bruker Python2 så programmene må være separert. Denne fremgangsmåten løser også et annet problem; levetid på vektorene. For at vektorene skal ha noe effekt på hvor man styrer så må de bidra i behaviour over tid. Behaviour spør ganske hyppig om hvilken vei den skal gå og hvis vi da skulle kjørt programmet som henter ut AIS-dataen og vektorene så hadde ikke vektorene vart lenge før de ble erstattet av nye. Vektoren ville returnert igjen på et tidspunkt, men da ville du bare oppnådd en konstant ror-endring, noe som ikke er ideelt for å seile et sted.

Ved å skrive vektorene til en fil så kan behaviour bare hente vektorer fra den filen hver gang behaviour etterspør ny informasjon. Da kan det heller velges hvor ofte vi oppdaterer den filen.

3.4 Behaviour

Det ble først utviklet en metode der AIS-dataen hentes direkte i de to «behaviour»'ene; boid og PSO. Da kan dataen hentes direkte der bevegelses vektoren blir regnet ut for å unngå mer sending av informasjon enn nødvendig. Dette førte til to problemer: levetid til vektoren og python versjon. Levetid til vektoren vil det kommes tilbake til senere. Problemet med python versjonen er at biblioteket pyais, som brukes til å hente inn AIS-dataen, krever python3.7 eller nyere. Denne python versjonen er ikke kompatibel med den ROS versjonen som systemet bruker. Derfor er ikke denne måten å gjøre det på mulig.

Løsningen det endte med ble da å ha en separat fil som henter inn AIS-meldingen og regner ut unvikelses-vektor for hver kontakt og skriver disse til en fil. Denne filen leses av «behaviour»'ene og legger til vektorene i utregningen sin. Denne ble dog ikke ferdigutviklet.

I denne vektorutregningen måtte det også gjøres justeringer i klassen som er ansvarlig for håndtering av vektorer. Denne klassen definerer vektorene på polar form. Det vil si at de har en størrelse r og en vinkel θ. Det at de er på polar form påvirker i stor grad hvordan man kan behandle de via matematiske operasjoner. Her var det en liten glipp i koden fra prosjektet til Hellenes og Lyssand. Der behandles vektorene som om de var basert på kartesiske koordinater, noe som gjør at regningen med de vil bli feil. Dette løstes ved å

endre adderingsmetoden til det matematiske uttrykket i bilder under. Der r_1 og r_2 er størrelsen på vektorene og θ_1 og θ_2 er vinklene til vektorene.

$$r = \sqrt{r_1^2 + r_2^2 + 2r_1 r_2 \cos(\phi_2 - \phi_1)}$$

Figur 3-19: Utregning av størrelsen til en vektor på polar form (Viola, 2017)

$$\phi = \phi_1 + \arctan2(r_2 \sin(\phi_2 - \phi_1), r_1 + r_2 \cos(\phi_2 - \phi_1))$$

Figur 3-20: Utregning av vinkel til en vektor på polar form (Viola, 2017)

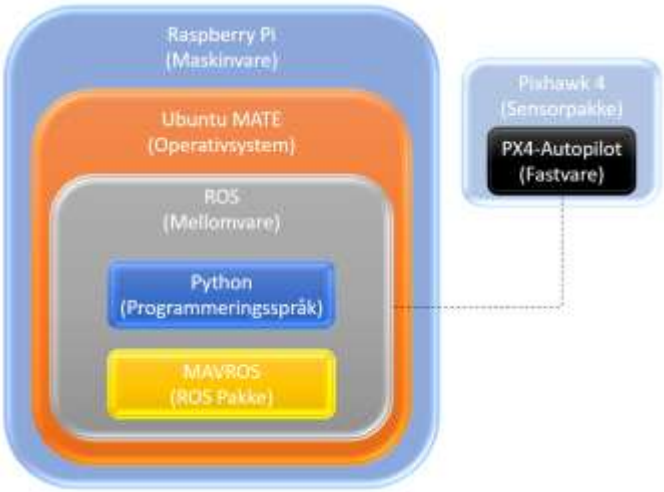
En annen feil som ble funnet var hvordan man ganget vektorer med konstanter. I koden ble både størrelsen på vektoren og vinkelen på vektoren ganget med konstanten. Dette gir ikke mening ettersom at du har ikke lyst til å endre retningen på vektoren ved å gange opp en konstant, kun størrelsen på den så den får en annen vekting. Dette løstes ved å fjerne den delen av funksjonen som ganget vinkelen med konstanten.

4 Testing og resultater

Frem til nå har oppgaven redegjort for bakgrunnen som trengs for å forstå problemstillingens relevans, for så å ta for seg teorien bak teknologien som anvendes, samt hvordan systemet har blitt satt sammen for å løse problemstillingen. For å undersøke om delsystemene svarer på problemstillingen, kommer oppgaven nå til å ta for seg testingen av systemet. Det første som testes er svermplattformen slik vi overtok den, hvor det kommer frem at systemet blant annet ikke er like «plug and play» som først antatt. Deretter vil radar- og AIS-systemene testes for å undersøke i hvilken grad de klarer å hente inn pålitelig data for navigering. Til slutt testes unnvikelsesvektoren, og hvorvidt den påvirker dronens oppførsel på en fornuftig måte.

4.1 Klargjøring av dronene og kompatibilitetstesting

Innledningsvis må dronene klargjøres ved å fremskaffe Pixhawk 4 og minnekortene som ble brukt når plattformen ble satt opp, for så å koble de til RPi-ene som er montert foran i dronene (som vist i Figur 3-1). Operativsystemet med mellomvare, programmeringsspråk og tilleggspakker blir tilgjengelig på RPi-en ved hjelp av minnekortet. Figur 4-1 viser strukturen til dronens system og hvordan alle delene henger sammen. Teoretisk skal svermplattformen starte opp ved å følge Vedlegg Q – Oppstart og sekvensiell sjekkliste for test, noe som ikke skjedde og ledet til spørsmål ved kompatibiliteten. Kompatibilitetstestingen består av tre deler, der første del tar for seg det å få dronene til å kommunisere med svermsystemet. De to siste delene er teoretiske kompatibilitetstester der den ene kartlegger mellom Ubuntu MATE og ROS, mens siste tar for seg hele systemet, med delene som er i vist i Figur 4-1 med hensyn på ROS.

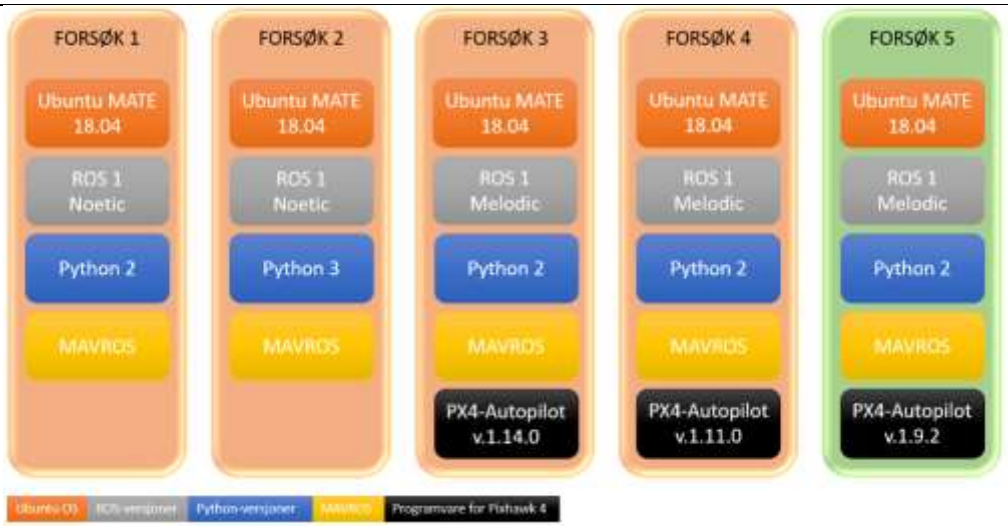


Figur 4-1: Figuren viser strukturen til en drone fra maskinvaren opp til sentrale pakker.

4.1.1 Klargjøring av dronesverm

Denne kompatibilitetstesten er delt opp i 5 forsøk slik den er vist i Figur 4-2, og disse handler om å få oppstarten til svermsystemet til å fungere vist i Vedlegg Q – Oppstart og sekvensiell sjekkliste for test. Før disse kan startes på så må RPi-en kobles til en monitor slik at dronens respons på kommandoene er direkte overvåket. Eneste kommandoen som trengs fra oppstarts-rutinen i denne testen er «roslaunch swarm system.launch», fordi koden er uendret fra sist den ble brukt.

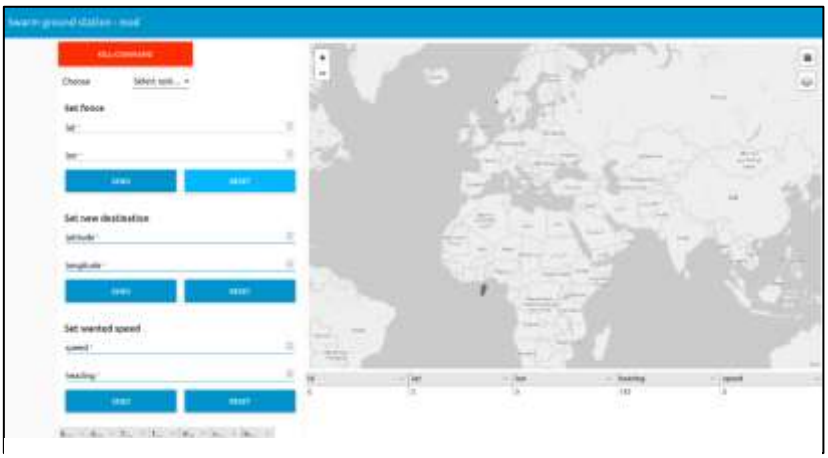
På **forsøk 1** i Figur 4-2 er intensjonen å få svermsystemet til å starte opp med ROS1 som er oppgradert fra Melodic til Noetic slik at systemet kjører på den nyeste versjonen av ROS1. Dette forsøket resulterte i en feilmelding som tilsa at kommandoen «roslaunch swarm system.launch» ikke fungerte, og ved hjelp av ROS sine nettsider ble det funnet at ROS1 Noetic er eksklusivt kompatibelt med Python3 og ikke Python 2. (Pecka, 2023)



Figur 4-2: Fremstilling av forsøkene med ulike kombinasjoner for å klargjøre svermsystemet. Her vises endringene som ble gjort mellom forsøkene og om det fungerte, vist med grønn hvis det funket og rød dersom det ikke gjorde det.

Forsøk 2 baserer seg på å beholde Noetic som mellomvare og oppgradere programmeringsspråket til Python 3, deretter gjenta samme kommando for å starte systemet som i forsøk 1. Etter å ha lastet ned Python 3 og valgt denne som versjonen som skal kjøres, ble det flere problemer med operativsystemet. Da kommandoen ble kjørt kom samme feilmelding som tidligere, som resulterte i at neste forsøk ville foregå med systemet slik det var før forsøk 1.

I **forsøk 3** ble mellomvaren nedgradert til ROS1 Melodic igjen, og programmeringsspråket nedgradert til Python 2 slik at systemet er identisk med slik det var da det greide å starte opp riktig slik det gjorde i 2019. Utfallet etter kommandoen «roslaunch swarm system.launch» ble tastet inn førte til at systemet faktisk startet opp, men alle verdiene som ble hentet fra Pixhawk 4 ble tolket som 0. Dette gjorde at båten sin posisjon ble 0.0ºN, 0.0ºØ i basestasjonen, som tilsvarer Guineagulfen utenfor Afrika som vist i Figur 4-3. Pixhawk 4 GPS modulen tok imot GPS-data, men datapakkene ble ikke mottatt slik de skulle i programmet. Dette var et tegn på av neste problem lå mellom ROS og Pixhawk 4.



Figur 4-3: Skjermbilde fra basestasjonen som viser posisjoneringsfeilen som oppsto på grunn av feil data

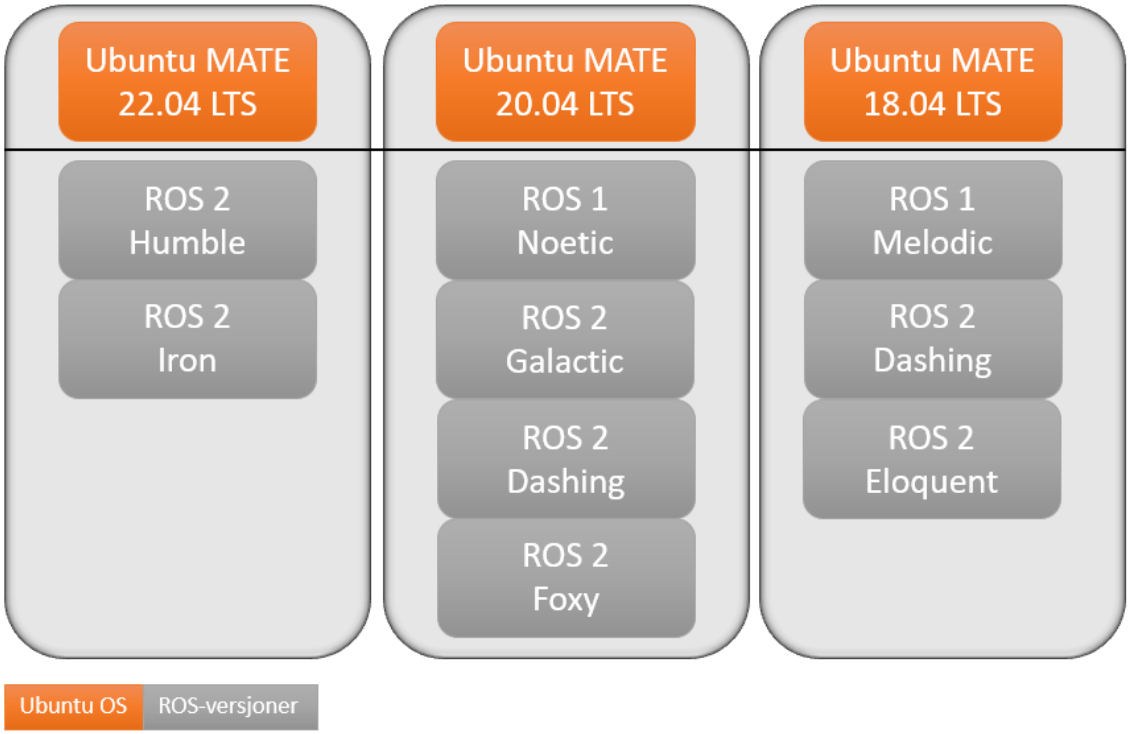
Forsøk 4 handler om å finne ut hva mellom ROS og sensorpakken som kunne være feilkilden til at verdiene ble ukorrekte. Det som ble funnet ut av var at fastvaren på sensorpakken var oppdatert til PX4-Autopilot v.1.14.0, som er den nyeste versjonen. Med erfaringen lært fra de foregående forsøkene så skulle denne fastvaren nedgraderes til versjon som ble brukt når plattformen var sist operativ, men uten kunnskap om hvilken versjon det var. Det ble valgt v.1.11.0 fordi denne inneholder feilrettinger som skulle løse feilmeldingen vi fikk. Dette forsøket endte med samme resultat som i forsøk 3, som førte til at i neste forsøk må fastvaren nedgraderes til versjonen som mest sannsynlig ble brukt i 2019.

I **forsøk 5** ble fastvaren nedgradert til v.1.9.2 som var versjon som ble gitt ut i tidsrommet plattformen ble utviklet. Når oppstarts-kommandoen da ble kjørt endte det opp med å løse problemet og det ble oppgitt riktig GPS-data på sjødronene, som viste seg å være på Sjøkrigsskolen. Disse erfaringene startet fokuset på å kartlegge kompatibiliteten mellom disse og de nyere utgivelsene og versjonene av de ulike delene i Figur 4-1.

4.1.2 Teoretisk kompatibilitet av operativsystem

Intensjonen med denne testingen er å finne ut hvilke ROS utgivelser som er kompatible med hvilke OS. På svermplattformen er operativsystemet Ubuntu MATE 18.04, så det er nødvendig at dette er en av operativsystemene som blir testet. Vi har også tatt for oss de to neste Ubuntu MATE LTS (Long Term Support) utgivelsene. Denne kartleggingen

gjøres ved hjelp av linkene i Vedlegg A – Oversikt over relevante livsløp. I Figur 4-4 er resultatet av denne teoretiske kompatibilitets testingen. Det resultatene av testen viser i er at Ubuntu MATE 20.04 og 18.04 versjonene er i en mellomperiode der de er både kompatible med ROS1 og ROS2 versjoner, mens 22.04 versjonen er kompatibel med kun ROS 2.

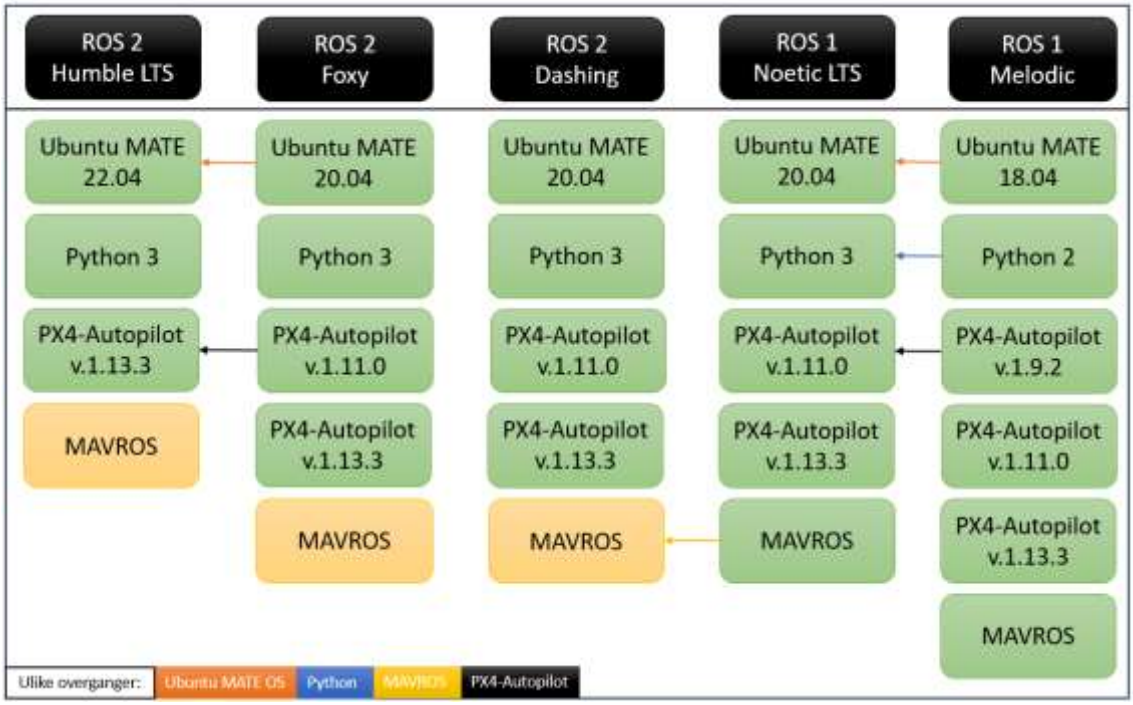


Figur 4-4: Oversikt over kompatibilitet mellom OS og ROS med hensyn på OS på grunnlag av Vedlegg A – Oversikt over relevante livsløp.

4.1.3 Teoretisk kompatibilitet i systemet

I denne testen kartlegges kapabiliteten mellom operativsystemet og de underliggende pakkene den har, men fra ROS utgivelsene som styrende faktor. ROS utgivelsene bytter på å være korttidsstøttet (1 år) og langtidsstøttet (flere år) annethvert år, og vi tar kun med de langtidsstøttede i denne testingen. De utgivelsene som ikke er LTS-merket i Figur 4-5 har allerede nådd End of Life (EOL) som vil si at de ikke blir støttet av utgiver lenger for feilretting fra deres side. Resultatet i Figur 4-5 belyser endringen av krav til kompatibilitet hos de ulike ROS utgivelsene med pilene i forskjellige farger.

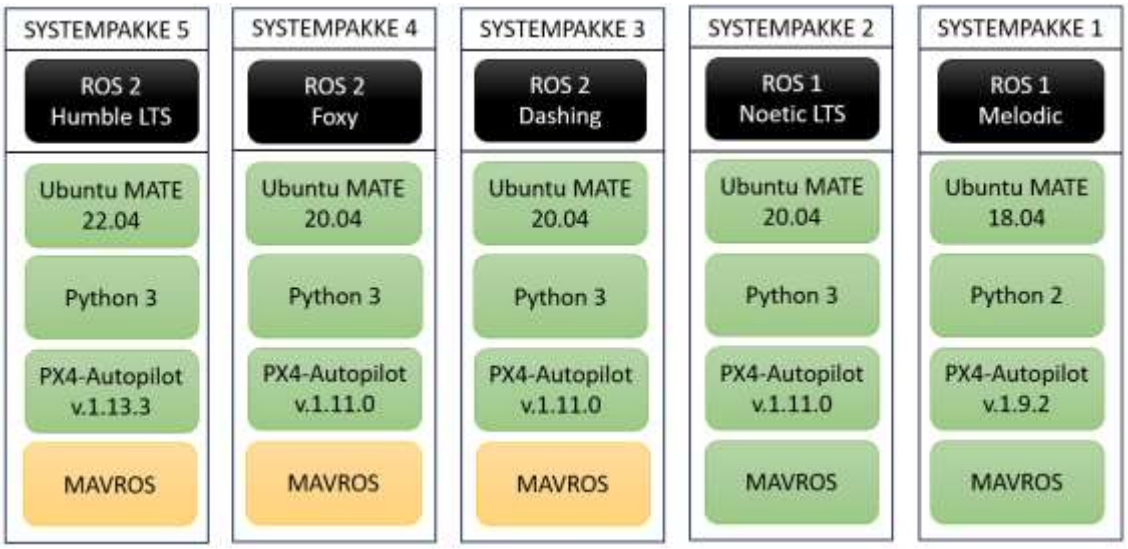
Boksene til MAVROS er endret til gul under ROS2 utgivelsene fordi utgiver ikke gir teknisk støtte for disse pakkene, men det finnes omveier for å kunne få det til å fungere som open-source samfunnet har funnet. I Vedlegg M –µXRCE-DDS kontra µRtps ligger det linker som forklarer endringen fra ROS1 til ROS2 som er grunnen til utfasingen av MAVROS. Når det kommer til PX4-Autopilot så skal alle ROS utgivelsene være forover-kompatible til nyere versjoner, men når det kommer til bakover-kompatibiliteten så er dette avhengig av hvilket OS systemet har, som vist med svart pil i Figur 4-5.



Figur 4-5: Visualisering av kompatibiliteten for ulike ROS utgivelser. Pilene fremhever overganger av krav til utgivelser, fargene deres er forklart i figuren. De grønne boksene tilsier at det er kompatibelt med ROS utgivelsen over, mens gult er inkompatibelt. Grunnlaget for denne modellen er Vedlegg A – Oversikt over relevante livsløp.

Etter disse testene og perioden med kartlegging så har det endt opp med fem systempakker å bygge svermplattformen på som er vist i Figur 4-6. Disse er oppdaterte med utgivelsene som er tilgjengelig på tidspunktet denne oppgaven er skrevet. Systempakkene bør tas hensyn til når plattformene skal videreutvikles, og alle er fungerende med RPi B-modeller, deriblant den som blir brukt i denne oppgaven. I

oppgaven for kollisjonsunnvikelse ble det valgt å fortsette å bruke systempakke 1 som ble brukt i 2019 av Hellesnes og Lyssand. Dette ble valgt fordi systemet funket når de lagde plattformen, som bør redusere mulige problemer som ikke er oppdaget ved de andre kombinasjonene.



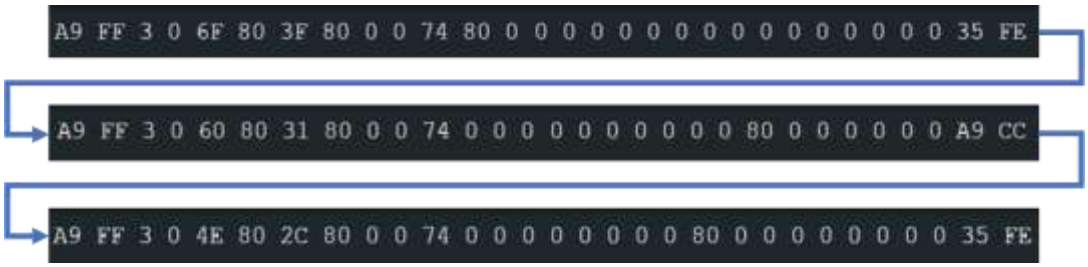
Figur 4-6: Oversikt over mulige systempakker etter gjennomført kompatibilitetstesting

4.2 Radartesting

Radaren skal gjøre det mulig å detektere og unngå hindre i dronens umiddelbare nærhet, eksempelvis holmer, sjømerker og lystige kajakpadlere. Totalt ble det utført tre tester av radaren, hvor hver test hadde som mål å identifisere og strukturere ulike deler av datarammen som sendes til Arduinoen, før det så sendes videre til svermlogikken på Raspberry Pi. Sluttproduktet av testene er arrays for hver delramme slik som beskrevet i delkapittel 3.2.2, som enkelt kan hentes av Raspberry Pi'en. Verdiene i arrayene brukes for å konstruere en av unnvikelsesvektorene som endrer *behaviour*-algoritmen til dronen, som skildret i delkapittel 3.2.3.

Første testen som ble gjennomført hadde som mål å hente ut en kontinuerlig datastrøm fra radaren for å kunne gjenkjenne datarammen manualen oppgir at radaren sender over

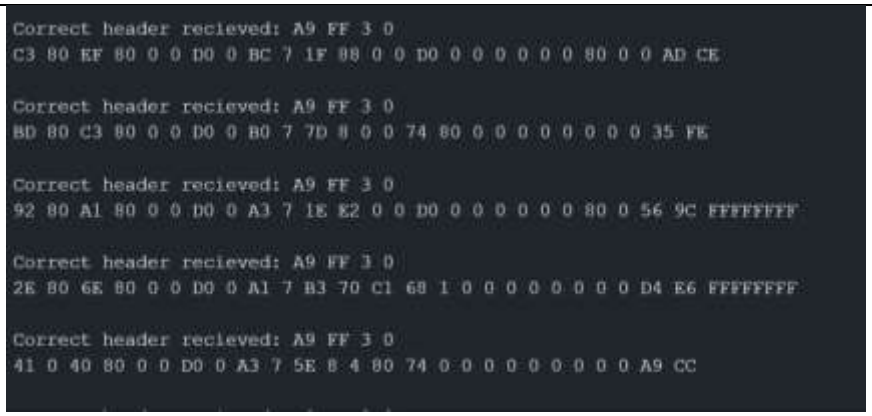
den serielle forbindelsen. Koden som ble lastet opp på Arduinoen (vedlegg F) er rimelig enkel, hvor den først konfigurerer pin 10 og 11 for seriell kommunikasjon, før den leser av disse og fortløpende skriver ut byte-strømmen fra radaren til konsollen i Arduino programvaren. Verdt å merke seg ved koden er at baud raten på den serielle kommunikasjonen mellom radaren og Arduinoen er annerledes enn den mellom Arduinoen og PCen. Arduino programvaren støtter ikke 256000, så derfor ble 250000 valgt som det nærmeste. Likevel gir ikke denne forskjellen noe merkbar effekt, da koden også printer ønsket resultat til konsollen ved baud rate på 9600 (som er standard for programvaren). Resultatet av testen vises i Figur 4-7:



Figur 4-7: Første test gir en kontinuerlig bytestrøm

Figur 4-7 viser en sammenhengende strøm av output fra radaren, oppgitt i bytes. I datastrømmen er det et tydelig gjentakende mønster, tydeliggjort i figuren med linjeskifter. Mønsteret starter alltid med «A9 FF 03 00», som er 1 bit unna å være lik oppgitt *frame header* på «AA FF 03 00». Avviket i mønsteret var konstant selv da radarenheten ble byttet, og dermed kan det konkluderes med at den faktiske *frame headeren* er «A9 FF 03 00». Etter headeren følger det 26 bytes før mønsteret gjentar seg, og dette er i henhold til manualen. I kontrast er *frame tailen* ikke i nærheten av slik den vises i manualen. Ifølge manualen skal den være «55 CC», men i praksis varierer den mellom et utvalg verdier, eksempelvis «35 FE», «A9 CC» og «AD CE», hvor det er tilsynelatende tilfeldig variasjon mellom disse. Dermed er det lite hensiktsmessig å videreutvikle noe kode som avhenger av *frame tailen*, da den ikke kan ventes å gi en forutsigbar verdi.

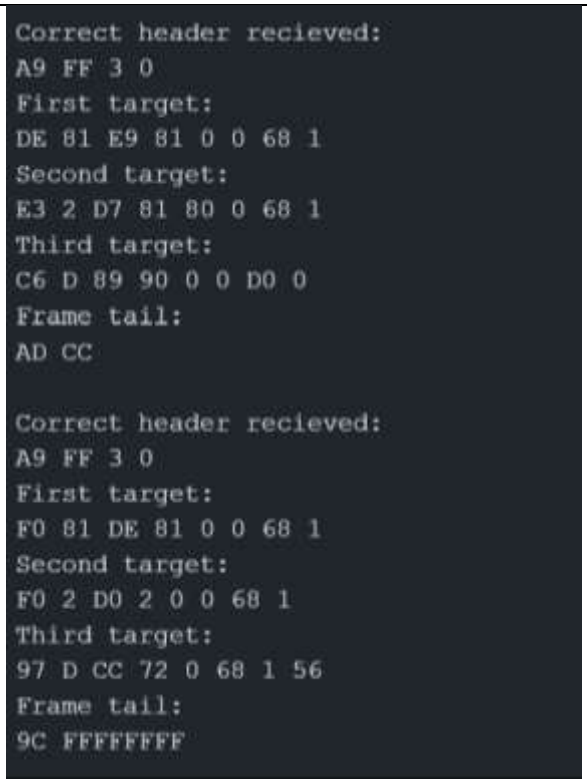
Den andre testen hadde som mål å hente ut *headeren*, for så å skrive ut de påfølgende 26 bytene før en ny *header* kommer. Koden er lagt ved i vedlegg G, og resultatet kan sees i Figur 4-8:



Figur 4-8: Andre test isolerer *frame headeren*, og printer resten av rammen

Figur 4-8 viser innholdet i fem påfølgende datarammer, og gjør det lettere å gjenkjenne strukturen til radaroutputet. Manualen sier at hvert mål har åtte bytes med måldata, og sett i sammenheng med datastrømmen ser man at radaren her har detektert to mål. Videre ser man også den brede variasjonen i *frame tailen*, hvor ingen av de fem rammene har lik *tail*. Tidvis er deler av *tailen* korrekt eller tilnærmet korrekt, men variasjonen er stor og uten noe åpenbart mønster. Et annet problem er at det hender siste byte skrives ut som «FFFFFFFF». Det ser ut til å skje når en byte tidligere i rammen ikke blir skrevet ut, og inntreffer uavhengig om radaren detekterer null, ett, to eller tre mål. Det er derfor vanskelig å si nøyaktig hvorfor problemet inntreffer. Likevel har ikke dette noen innvirkning på programmet, ettersom *frame tailen* ikke brukes til noe.

Tredje og siste test utført på radaren har til hensikt å fordele bytene inn i sine respektive deler av rammen. Her ble det benyttet arrays for hver del av rammen, som fortløpende ble skrevet ut til konsollen i Arduino programvaren. Hensikten med dette er at dataene for hvert objekt skal være isolert før Raspberry Pien henter det ut, og dermed blir uthentingen mer effektiv i motsetning til hvis datastrømmen ble videresendt til Raspberry Pien uten å være behandlet. Koden er i vedlegg H, og resulterte i følgende:

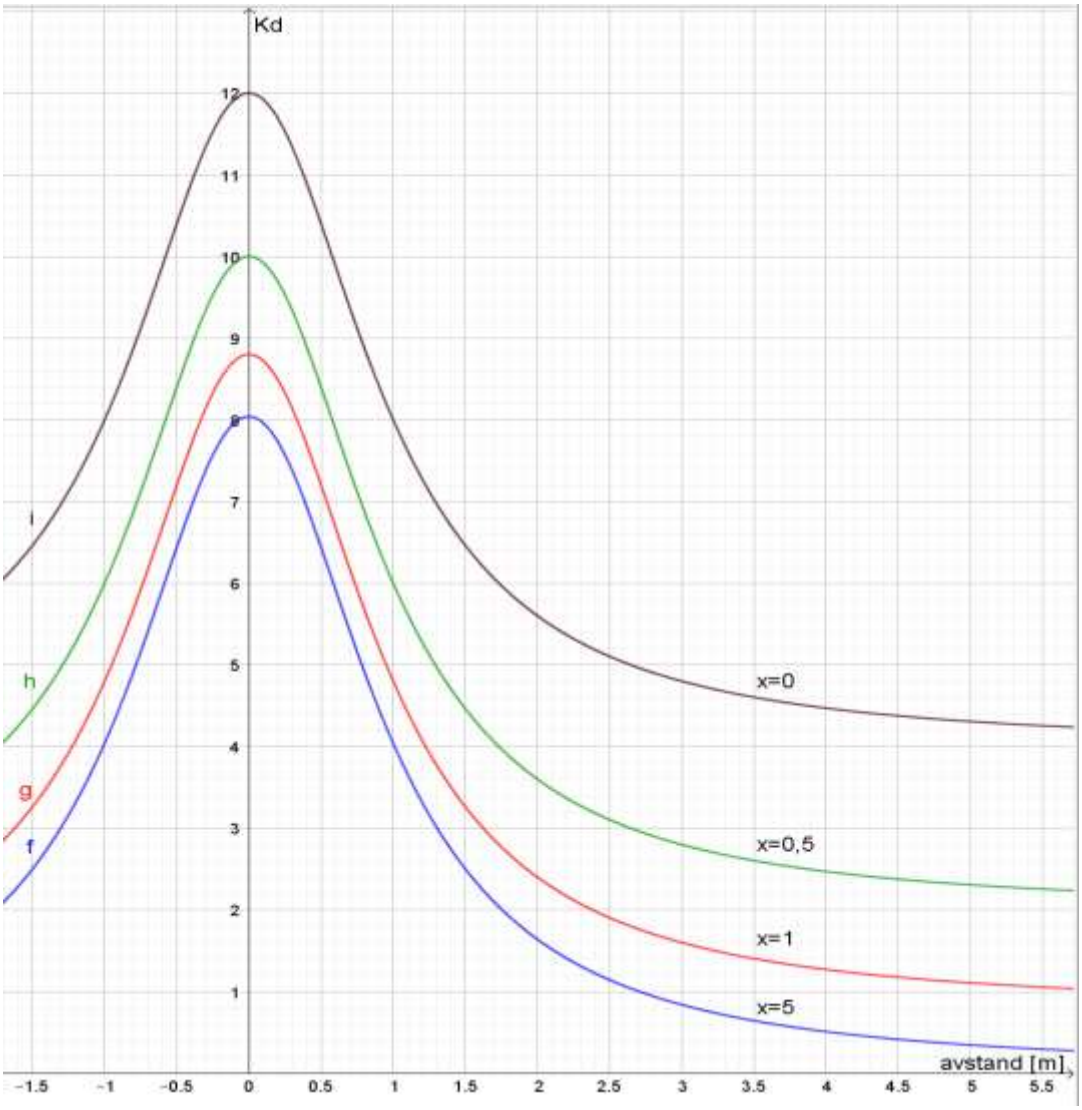


Figur 4-9: Tredje radartest isolerer hver del av rammen, slik at de lett kan hentes av Raspberry Pien

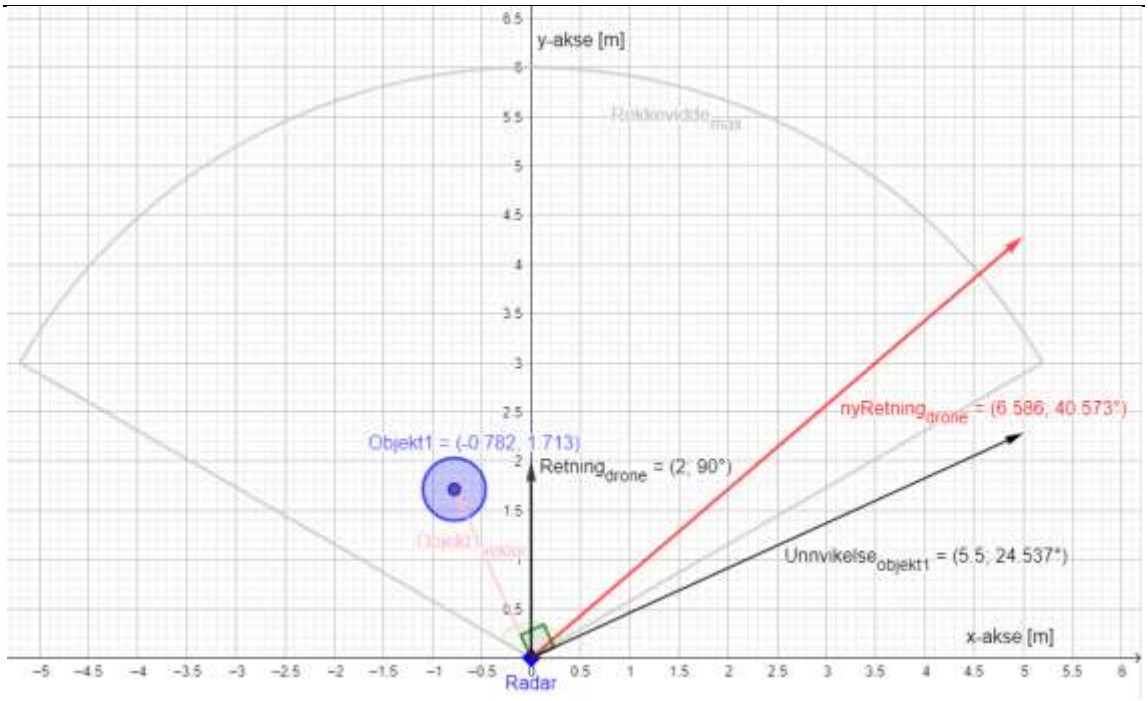
Figur 4-9 viser at programmet deler datarammen opp i hver enkelt del og skriver ut delrammene til Arduino konsollen. Dataene for hvert objekt er dermed klargjort i hver sin array , klare for å bli hentet av Raspberry Pien. Det tidligere problemet med «FFFFFFFF» er fortsatt til stede, uten merkbar endring i hvor hyppig det forekommer. Vurderingen fra første test om å ikke gjøre koden avhengig av *frame tailen* er derfor styrket.

På dette punktet av systemutviklingen begynte det å bli knapt med tid, og en videre test av radaren på vannet eller systemintegrering med Raspberry Pi og resten av sverm-algoritmen er derfor ikke blitt gjennomført. Likevel er det behov for å utføre en teoretisk test av unnvikelsesvektoren, for å undersøke om den oppnår ønsket påvirkning av dronens adferd. Til testen benyttes eksempeldataen gitt i radarmanualen, samt faktoren Kd fra delkapittel 3.2.3. Kd har formelen $Kd = \frac{2}{0.25l^2 + 0.25} + \frac{1}{0.25 + x^2}$, hvor variabelen l er lengden på posisjonsvektoren til objektet ($l = \sqrt{x^2 + y^2}$) og x er avstanden fra objektet

til dronens kurslinje. Figur 4-10 er en grafisk fremstilling av Kd, og Figur 4-11 viser hvordan Kd påvirker unnvikelsesvektoren og med dette dronens adferd.



Figur 4-10: Faktoren Kd avhengig av avstand til objekt samt forskjellige avstander fra objektet til dronens kurslinje (delkapittel 3.2.3)



Figur 4-11: Unnvikelsesvektorens påvirkning på dronens adferd

I Figur 4-11 ser man retningsvektoren til dronen før objektet er detektert, den påfølgende unnvikelsesvektoren og til slutt dronens nye retningsvektor (markert rødt). Den nye retningsvektoren har i både størrelse og retning blitt påvirket av unnvikelsesvektoren, som vil resultere i at dronen vil foreta en kraftig retningsendring for å unnvike objektet. Dette kan tyde på at formelen for Kd ikke er tilstrekkelig optimalisert, ettersom unnvikelses-vektoren har veldig stor påvirkning på dronens retning.

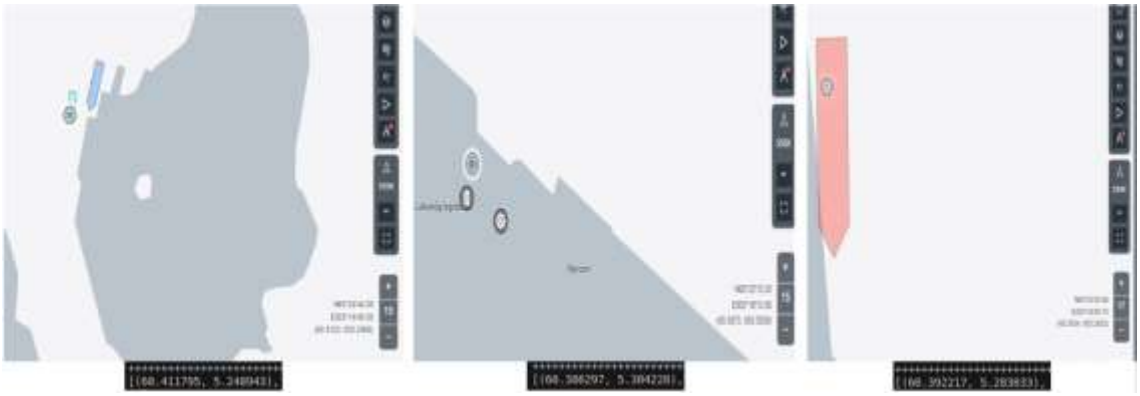
Sluttresultatet av radartestingen er tre arrays på Arduinoen som inneholder informasjon om detekterte objekter. Disse oppdateres fortløpende når nye verdier sendes gjennom den serielle forbindelsen med radaren. Arduinoen kunne alternativt også ha gjort utregningene fra bytes til numeriske verdier, men ettersom produsenten bruker en særegen metode for konvertering fra bytes til desimaltall er disse utregningene ikke gjennomført. Noe videre integrering med Raspberry Pien er ikke gjort, men dataen er klargjort for å kunne anvendes videre på lik måte som AIS-dataene anvendes i neste kapittel.

4.3 AIS-testing

Innhenting av AIS-data skal gjøre det mulig å unngå kollisjon med andre fartøy på lengre hold som har AIS-sender.

Første test for AIS var å se om programmet mottar AIS-meldinger, klarer å dekode disse, og at de meldingene som kommer inn er reelle. At meldingene er reelle, vil si AIS-meldinger fra fartøy det kan verifiseres er i samme posisjon som AIS-meldingen tilsier. Denne verifiseringen gjøres via Marine Traffic. Marine Traffic er en nettside drevet av Kpler som presenterer data for bruk til forskning i det maritime domenet. Den funksjonen testen bruker til verifisering er Live Map funksjonen, som henter inn nåværende AIS-data og presenterer denne grafisk (Kpler, 2023). Dette muliggjør verifisering ved at koordinatene, som programmet for mottak av AIS meldinger skriver til konsollen, kan brukes til å finne frem fartøyet som sendte AIS-meldingen i Marine Traffic sitt Live Map og validere AIS-meldingens autentisitet.

Programmet for innhenting av AIS-meldinger startes opp og det skrives ut tre kontakter i konsollen: (60.411795, 5.24894), (60.386297,5.304228), (60.392217,5.283033). Posisjonene ble brukt til å finne utsenderne av AIS-meldigene i Live Map, se Figur 4-12.



Figur 4-12 Skjermutklipp av Marine Traffic Live Map som viser senderen av AIS-meldingen. Under utklippene i sort er utklipp fra konsoll-vinduet i Python der koordinaten ble skrevet ut. NB, koordinatene på utklipp fra Live Map viser feil koordinat grunnet den viser til hvor musepekeren forlot kartet.

Figur 4-12 viser at koordinatene som programmet henter gjenspeiler posisjonen til fartøyene som sendte AIS-meldingen i Live Map. Dette fyller testens krav for at AIS-

meldingene programmet mottar er reelle. Med dette grunnlaget kan det testes videre om vektoren blir regnet ut korrekt.

Hensikten med denne testen var å sjekke om utregningen av distanse og retning ble korrekt og om vektorutregningen ble korrekt. Med korrekt menes det at vektorer får som samsvarer med forventede verdier gitt distanse og vinkel, samt at addisjon av disse vektorene blir matematisk korrekt. Her er det sentralt å få med seg at output fra en AIS-melding skal ende opp som en enkelt vektor som blir laget av en summering av alle vektorene fra fremtidige aliaser. Metoden for testen er å først sjekke om utregningen av distanse og vinkel var korrekt for så å sjekke vektorutregningen. Dette gjøres ved å kjøre programmet for å hente inn de nåværende AIS-meldingene. Det legges på noen funksjoner for utskriving til terminalvinduet i programmet for å tilgjengeliggjøre: koordinatene, distansene og den endelige unnvikelses-vektoren i terminal-vinduet. Testingen blir utført ved hjelp av tre forskjellige verktøyer på nett. Disse var: igismap sin distanse og vinkel kalkulator (IGISMAP, 2023), sunearthtools sin distanse og vinkel kalkulator (SunEarthTools, 2023) og Marine Traffic sitt Live map (Kpler, 2023). Verdiene fra programmet skal sammenlignes med igismaps sitt verktøy for å sjekke rett implementering av koden for utregning av vinkel. Det er fra igismaps vi har hentet den matematiske formelen for utregning av vinkelen mellom to koordinater så denne burde i teorien bli identisk til den vi regner ut selv.

Bearing from point A to B, can be calculated as,

$\beta = \text{atan2}(X,Y)$,

where, X and Y are two quantities and can be calculated as:

$X = \cos \theta b * \sin \Delta L$

$Y = \cos \theta a * \sin \theta b - \sin \theta a * \cos \theta b * \cos \Delta L$

Figur 4-13: Formel for utregning av vinkel β mellom to koordinatpunkter der θb er breddegrad til punktet du skal ha vinkelen til, θa er i programmets tilfelle egen posisjons breddegrad, og *Δ*L er differansen mellom lengdegradene til koordinatene

Det skal da sjekkes om vinkelen som kommer fra programmet og vinkelen Igismap regner ut blir lik. Distansen som nevnt regnes ut ved hjelp av geopy så det forventes ikke identiske resultater her. For å forhindre å gå i en bekreftelses-felle sjekkes resultatene også opp mot en tredjepart og dette brukes sunearthtools sitt verktøy, som også regner ut avstand og vinkler mellom to punkter. For spesifikasjoner på maskinen brukt i test, se Vedlegg B – «Clean Install» av operativsystem på lab-PC.

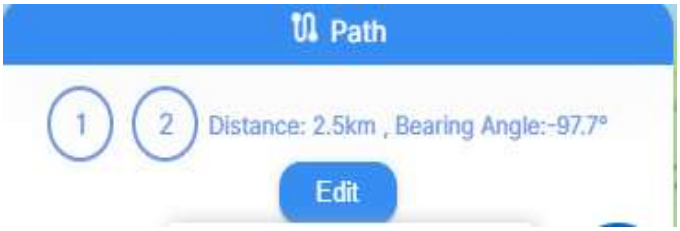
Programmet ble kjørt og fikk inn en AIS-melding med koordinat (60.397753, 5.313375). Dataen i Figur 4-14 viser hvilke utregninger programmet gjorde med dataen fra meldingen.

Figur 4-14: Utklipp av terminal-vinduet som viser resultatene av utregningene for AIS-meldingen. Øverst er koordinatene for aliasene, deretter en liste med vektor objekter, så distanser fra egen posisjon til hver av koordinatene, til slutt er den totale vektorens størrelse og vinkel som lages av programmet.

Øverst i Figur 4-14 ser du en liste med koordinater som tilhører initiell posisjon og fremtidige posisjoner til fartøyet som sender AIS-meldingen. Det første koordinatet er den originale posisjonen som blir hentet ut av AIS-meldingen og de påfølgende er aliaser som programmet lager selv. Du ser her at fartøyet har så og si de samme koordinatene for alle aliasene, som betyr at farten er rapportert som 0. I neste linje i Figur 4-14 ser du en utskrift av vektorene som blir laget ut ifra koordinat-punktene. Disse er ikke formatert i utskriften, så man ser de som objekter. Dette viser at det blir laget en unnvikelsesvektor per punkt ved at antall objekter er likt antall koordinater i linje 1 i Figur 4-14. Videre ser man distansen fra vår egen posisjon til fartøyet som har sendt AIS-meldingen og til de lagde aliasene. Til slutt ser man den ferdige utregnede unnvikelses vektoren sin lengde og vinkel i grader fra -180 til 180. Der 0° er Nord og positive grader er øst for egen posisjon og negative grader er vest for egen posisjon.

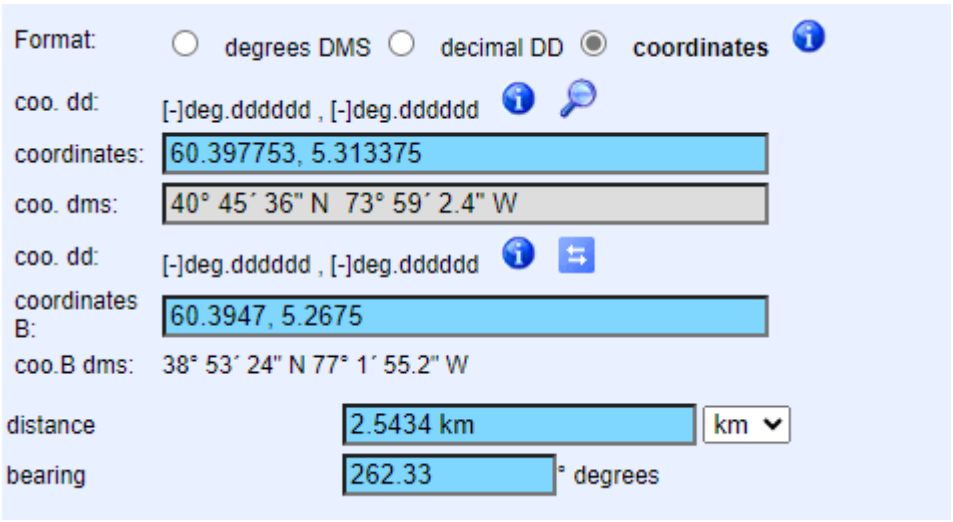
Vi skrev inn punktene i igismap sitt verktøy og fikk følgende:

Figur 4-15: Visualiseringen fra igismap for avstand og vinkel. 2 er egen posisjon og 1 er kontakten (rapportert AIS-fartøy) (IGISMAP, 2023)



Figur 4-16: En mer detaljert fremstilling av dataen fra igismaps (IGISMAP, 2023)

Dette er dataen som blir brukt for å sjekke riktig implementering av formelen i Figur 4-13. Videre skal denne dataen sammenlignes opp mot data fra verktøyet til SunEarthTools. Den samme prosessen brukes som i avsnittet over og får resultatene i Figur 4-17:



Figur 4-17: Sunearthtools sin utregning av dataen (SunEarthTools, 2023). Øverst vises koordinatene som skrives inn og på bunn er outputen til verktøyet: distanse(km) og vinkel(°)

Her ser vi at det er litt forskjell på svarene, spesielt på vinkelen. Dersom resultatene fra programmet sammenlignes med Igismap sine ser vi at vinkelen er nesten helt identisk mens på distansen er det noe mer slingring. Her gjør avrundingen det vanskelig å se hvor stor forskjellen i resultatene egentlig er, men vinkelen kan tyde på at implementeringen av formelen for utregning av vinkel har vært rett. Når vi da sammenligner med dataen fra sunearthtools ser vi at distansen (2,5434km) er ganske lik med den distansen koden regner ut (2,5519km). Ved første øyekast ser vinkelen helt feil ut, men det er fordi formelen som benyttes i koden gir en vinkel fra 0 til 180° for koordinater på østlig side for egen posisjon og 0 til -180° for koordinater som er på vestlig side. For å sjekke den egentlige likheten så kan det regnes ut slik: -97.69 + 360 og får da 262.31 som viser at de er ganske like.

Ut ifra disse resultatene så kan det konkluderes med at det er ingen grove feil i utregning av distanse og vinkel til koordinater. Det er litt forskjell i resultatene fra de to verktøyene i forhold til distanse, men dette virkes å skylde en grov avrunding da sunearthtools viser fire desimaler 2.5334 mens Igismap viser kun en 2.5. Nøyaktigheten i distanse til punktet er av med noen titalls meter. Der programmet viser 2.5519 mens verktøyene viser 2.5334 og 2.5. Det er dog tilfredsstillende nok til at det kan fortsettes med neste del av testingen, som er å se på utregningen av vektoren.

I neste del av testen brukes kun sunearthtools sitt verktøy, ettersom at denne gav høyere oppløsning på resultatene og er også mer brukervennlig når det skal måles flere ganger mot samme punkt. Det måles her opp mot et fartøy i bevegelse for å se om den regner ut totalvektoren korrekt når det er blir flere ulike punkter for programmet å ta hensyn til. Fremgangsmåten her ble relativ lik, men ettersom det var mer data så er denne strukturert i et excel ark for å gjøre det mer oversiktlig og håndterbart. Dataen fra testen kan sees i vedlegg O. Disse dataene inneholder distanse fra egen posisjon regnet ut av koden og regnet ut av verktøyet. For hvert fremtidig alias regnes vektorens lengde ut ved bruk av formelen under**Feil! Fant ikke referansekilden.**, vinkelen hentes ut fra verktøyet. Den resulterende unnvikelsesvektoren blir regnet ut ved hjelp av kalkulator og sammenlignes med vektoren programmet regner ved hjelp av formelen:

$$Error_{Prosent} = \frac{Forventet - Faktisk}{Faktisk} * 100\% \qquad (3.1)$$

for å finne hvor stor feilen var. Forventet verdi er den som er regnet ut ifra verdiene gitt fra verktøyet, mens faktisk verdi er den som programmet regnet ut.

Total lengde verktøy	1,00692
Total vinkel verktøy	-100,816
Total lengde kode	0,96459
Total vinkel kode	-100,8215
Feil lengde	4,388 %
Feil vinkel	-0,005 %

Figur 4-18: Resultat for utregning av vektorer. Markert i blått er verdien regnet ut manuelt ved hjelp av verdiene hentet fra verktøyet til SunEarthTools. Markert i rødt er vektoren som programmet regner ut. Under er feilen for lengde og vinkel regnet ut ved hjelp av Formel 4.1

For den totale vektoren så er feilen i vinkelen liten, mens feilen i lengden er noe større. Størrelsen på feilen for vinkel er meget tilfredsstillende og tyder på at denne utregningen ikke trenger noen justeringer. For lengden derimot er det litt mer usikkert. Denne feilen kan også potensielt bli større for en kortere distanser, ettersom formelen for utregningen av lengden for vektoren gror eksponentielt når distansen går mot 0. En mulig feilkilde kan være at koordinatene vi har skrevet inn i verktøyet ikke har vært hatt like høy oppløsning som de koordinatene som programmet regner ut.

Potensialet for en økende feil er viktig å kartlegge. Derfor utføres det ny test på utregningen av vektoren. I denne neste testen simuleres posisjon, retning og fart til kontakten som skal unnvikes fra. Det velges verdier som gir stort utslag både i distanse og i vektoren. Dette er for å få en form for «stresstest» der nøyaktigheten til programmet sjekkes ved sprik i data. Dataen valg kan sees i Figur 4-19:

```
# Simmulering av fartøy
other_lat = 60.39461
other_lon = 5.287204
speed = 7
course = 320
other_pos = (other_lat, other_lon)
```

Figur 4-19: Utklipp fra programmet der den simulerte dataen blir definert.

Denne situasjonen vil se ut som i Figur 4-20:

Figur 4-20: Visualisering av bevegelsen til den simulerte kontakten. 1 er startposisjon og 2 er sluttposisjon (Kpler, 2023)

Den samme prosessen for den tidligere testen blir gjentatt, men denne gang med et ekstra fokus på å ha et likt antall gjellende desimaler for programmets utregning og den manuelle utregning ved hjelp av verdiene fra verktøyet. Resultatene sees i Figur 4-21.

Total lengde verktøy	3,94414
Total vinkel verktøy	-108,664
Total lengde kode	3,93027
Total vinkel kode	-108,8906
Feil lengde	0,353 %
Feil vinkel	-0,208 %

Figur 4-21: Resultat for utregning av vektorer. Markert i blått er verdien regnet ut manuelt ved hjelp av verdiene hentet fra verktøyet til SunEarthTools. Markert i rødt er vektoren som programmet regner ut. Under er feilen for lengde og vinkel regnet ut ved hjelp

Resultatene for denne testen viste større nøyaktighet for distansen enn tidligere. Dette kan tyde på at feilkilden om nøyaktighet i desimaler nevnt i forrige test kan ha vært

utslagsgivende. Derimot er det her en større feil i vinkelen. Dette er ikke helt urimelig, ettersom det her var en mye større varians i vinklene til de forskjellige koordinat-punktene. Det var her 100° forskjell på største og minste vinkel i forhold til 1° på det meste i forrige test.

4.4 AIS-basert vektortesting på plattformen

Denne testens formål var å teste implementeringen av den AIS-baserte unnvikelses-vektoren på svermplattformen. Testen vil bli utført med BOID som oppførsel. Denne oppførselen er mer forutsigbar og er derfor best å bruke for denne testen. Det vil da forsøkes å holde dronene cirka 5 meter fra hverandre. Dette er den ideelle posisjonen dronene søker å oppnå i BOID oppførselen. Da vil seperasjonsvektoren og samlingsvektoren jobbe minimalt, mens samstillingsvektoren vil styre svermen mot en felles gjennomsnittlig retning. Fremgangsmåten for denne testen blir å simulere en kontakt vest for posisjonen til dronen på ganske nært hold. Det forventes da en kraftig unnvikelsesvektor som vil styre fartøyet østover. Kun en av dronene blir utstyrt med programmet for AIS-unnvikelse i den hensikt å simplifisere testen. Dronene blir her båret og beveger seg etter det roret viser, se Figur 4-22.

Figur 4-22: Illustrasjon av testen der dronen pekes etter rorutslaget.

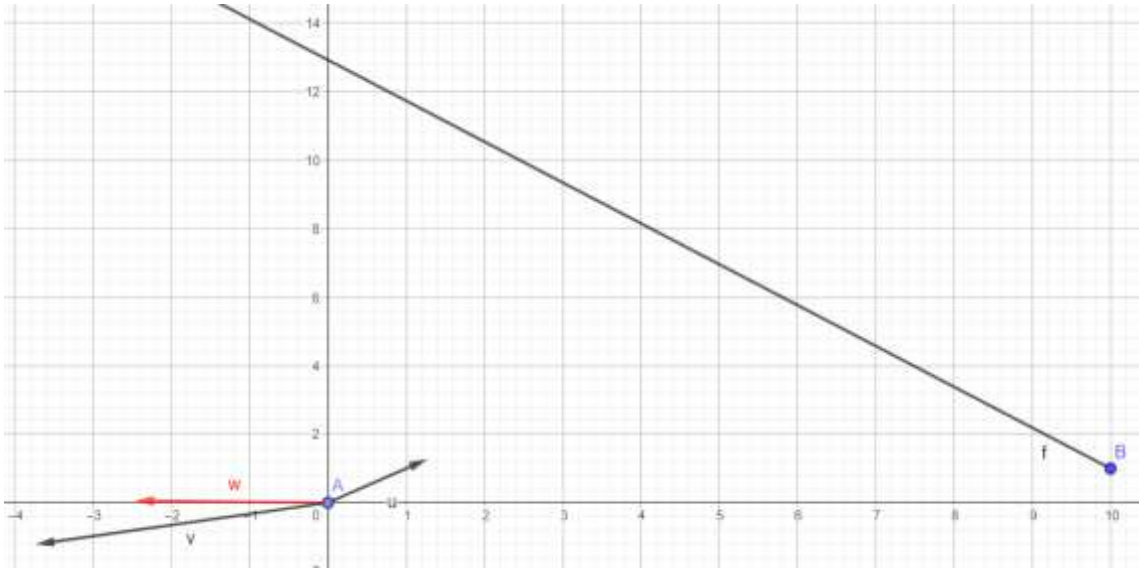
Programmet ble startet opp med dronen pekende vest og det kom inn data til ground station. Dronen ble vist på forventet posisjon og med riktig bearing, som bekrefter korrekt posisjon basert på GPS-modul til Pixhawk. Propellene gikk rundt og rorene beveget seg i styrbod retning. Vi beveget båten etter rorene og fant at den justerte seg mot nord. I terminalen der programmet kjøres derimot kom det feilmeldinger om at behaviour noden hadde dødd. Det kom med dette også en rekke feilmeldinger. Vi utførte en komplett omstart av systemet, men resultatet ble det samme. Dermed måtte testen avsluttes og koden feil søkes før et nytt forsøk kan utføres.

Da feilene nevnt i vedlegg N ble fikset ble testen startet på nytt med samme startpunkt som sist med samme simulerte kontakt, altså forventes en styring mot øst. Det er her sentralt å få med seg at AIS-innhentingen måtte deaktiveres for at programmet skulle kjøre (se vedlegg N). Da dronene ble startet så tørnet rorene på dronen med unnvikelses programmet på en måte motstridene måte, da det virket som om de ikke jobbet i samme retning, dog så det ut til at de stabiliserte seg dersom dronen ble pekt i øst. Det ble forsøkt litt manøvrering av dronenes relative posisjon for å se hvordan dette ville påvirke ønsket posisjon. Det kom lite resultater av dette da det ble publisert feilmeldinger om at Boat_rx prosessen døde. Dette er programmet som mottar meldinger fra de andre dronene i svermen. Da stoppes også propellene. Det ble forsøkt å gjenstarte testen, men det var fortsatt samme problem.

På dette tidspunktet var det ikke tid til å feilsøke og gjennomføre en ny fysisk test på platformen. Da den sistnevnte testen ikke gav resultater som kunne brukes til å si noe om hvorvidt implementering faktisk fungerte, ble det gjennomført en matematisk test av unnvikelse basert på AIS.

Denne testen baserer seg på BOID oppførsel ettersom denne baserer seg rent på utregninger med verdier det kan forutsi hvordan vil se ut. PSO er kun simulert med vilkårlige verdier og er derfor ikke gunstig for en slik test. I scenarioet for testen er dronene i en posisjon med 5 meter avstand til hverandre i formasjon av en likesidet trekant. 5 meter er den ideelle avstanden som dronene er programmert for å forsøke å oppnå. Derfor vil seperasjon- og tiltrekningsvektorene være lik 0. Dronene navigerer mot en posisjon som er 45° i kompassretningen. Den simulerte AIS-meldingen fra Figur 4-19

blir brukt som objekt som skal unnvikes i denne testen. Utregningene for denne testen ligger i vedlegg P. Den resulterende situasjonen kan sees i Figur 4-23:



Figur 4-23: Visualisering og kalkulering av resulterende vektor i Geogebra. A er egen posisjon. u er vektoren svermen styrer etter i øyeblikket AIS-meldinger kommer inn, v er unnvikelsesvektoren som opprettes av AIS-meldingen, w er den resulterende vektoren som droner vil styre i.

5 Drøfting

I forrige kapittel ble det gjennomført testing på tre hovedområder kompatibilitet, radar og AIS. Kompatibilitetstestene resulterte i at systempakken av RPi 3B+, Ubuntu MATE 18.04, ROS1 Melodic, Python 3, MAVROS og PX4-Autopilot v.1.9.2 ble brukt til oppstart av svermsystemet.

Radartestingen endte opp med tre arrays med informasjon om detekterte objekter, som kan hentes av Raspberry Pien. Dog er ikke denne integreringen med resten av dronesystemet gjennom Raspberry Pien er ikke implementert.

Testingen for AIS viste at det fungerer å hente inn sanntidsmeldinger til programmet på en test pc. Programmet klarte videre å regne ut vinkel og avstand til utsenderen av AIS-meldingen og til de aliasene den selv oppretter. Disse verdiene blir laget til vektorer og addert sammen korrekt. I simulerte tester klarte dronen å unnvike, men resultatene for den fysiske testen var ikke god nok til å kunne bekrefte at det fungerer når implementert på svermplattformen.

Dette kapittelet handler om drøfting av de resultatene oppgaven har kommet frem til under videreutviklingen av svermplattformen fra 2019. Disse resultatene skal drøftes opp mot målene som har blitt oppgitt i starten av oppgaven, gjengitt i

Tabell 5-1 under. Slik vil det komme frem i hvilken grad målsetningene ble oppnådd, hvorvidt valg som har blitt tatt underveis var hensiktsmessig og om det finnes bedre

metoder for å gå frem. Videre blir det drøftet om mulighetene denne oppgaven kan gi i en sjømilitær kontekst.

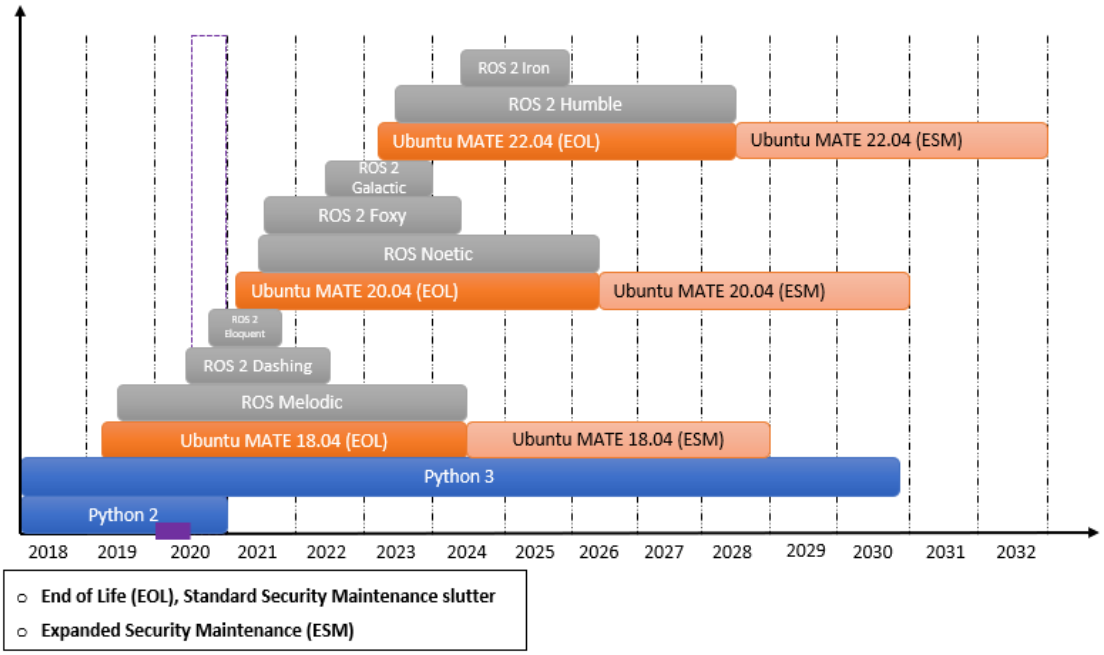
Tabell 5-1: Målsetninger for oppgaven (fra delkapittel 1.4)

	Mål	**Beskrivelse**
1	Systemforståelse	Få forståelse av hvordan plattformen fungerer og hva som skal til for å få den til å kjøre
2	Få plattform operasjonell	Kartlegge teknisk gjeld og forberede teknisk utstyr etter kompatibilitet
3	Implementere radar	Enheten skal kunne innhente seriell data fra radaren
4	Implementere AIS	Enheten skal kunne innhente AIS-data
5	Lokalisering av hindring	Enheten skal kunne detektere om hindringen er på kollisjonskurs eller ikke, relativ til egen bevegelsesretning
6	Unnvikelse	Enheten skal kunne endre egen bevegelsesretning i den hensikt å unngå kollisjon med lokalisert hindring

5.1 Kompatibilitet

Da klargjøringen av dronesvermen ble testet for første gang i denne oppgaven oppsto det uforventede kompatibilitetsproblemer grunnet mangel på systemforståelse, og det ble

mindre «plug and play» enn først antatt. I tillegg ble det antatt at for å oppdatere systemet var det kun nødvendig å finne den nyere utgivelser og versjoner av delene i Figur 4-1. Som vist i avsnitt 4.1.1 skulle dette være en mer omfattende jobb enn forventet, noe som betyr at den opprinnelige programvaren til svermplattformen måtte håndteres som et legacy-system. Denne systemforståelsen var sentral for å kunne bygge videre. Dette systemet er veldig skjørt med alle de ulike delene vist Figur 4-1 som må være kompatible med hverandre. Noe som førte til at det ble nødvendig å hente inn mye informasjon på nett for å forstå hva som skapte problemer mellom programmene i den hensikt å få den operasjonell.



Figur 5-1: Oversikt over livsløpet til Ubuntu MATE, ROS1, ROS2 og Python utgivelser. Lilla markering er utviklingsperioden til svermplattformen. Informasjonen innhentet for å lage denne ligger i Vedlegg A – Oversikt over relevante livsløp.

Det å få plattformen operasjonell igjen ble krevende av en rekke grunner, deriblant tidspunktet svermplattformen ble utviklet på står for flere av dem. Første grunn har med programmeringsspråket Python sitt livsløp som er vist i Figur 5-1. Alle de senere utgivelsene og versjonene bygger på Python3 som gjør at innenfor Python-området var svermplattformen allerede utdatert da Python2.7 (siste versjon av Python2) fikk status

EOL i overgangen til 2020, under én måned etter svermplattformen ble fullført. I overgangen fra Python2 til Python3 skjedde det store endringer innenfor blant annet integer divisjon, unicode support og en del mer. Disse endringene gjør at biblioteker laget for python3 sjeldent er bakover-kompatible med python2. (Luna, 2022) Denne inkompatibiliteten førte til at programmet for unnvikelse ikke fungerte når det ble lagt til på systemet som eksisterte fra før. Det valgte programmeringsspråket ble Python 2, selv om det hadde kommet 9 versjoner av Python 3 innen starten av 2020 (Python Software Foundation, 2023) og det var det valget som var mer fremtidsrettet som vist i Figur 5-1.

Andre grunnen til at det ble krevende har med valget av en ROS1 utgivelse fremfor ROS2. Valget av ROS1 Melodic er et valg som igjen ikke kan begrunnes av forover-kompatibilitet, der dette er den siste ROS utgivelsen som baserer seg på Python 2 som fremvist i Figur 4-6. ROS1 Noetic er den eneste av ROS1-familien som bruker Python 3 og kunne vært et delvis fremtidsrettet valg, men Noetic ble ikke tilgjengelig før mai 2021. Satsningen på ROS2 fra utgiver sin side ble permanent i 2022 da de kun distribuerte nye ROS2 utgivelser som var Ubuntu MATE 22.04-kompatible, for de to foregående OS'ene var det både ROS1 og ROS2 som Figur 5-1 viser.

Et annet argument for at ROS1 Melodic var lite bærekraftig å velge er MAVROS. Denne tilleggspakken gjør det vanskeligere å oppfylle mål 2, å få svermplattformen operasjonell, samtidig som den skal være forover-kompatibel grunnet mangel på utfasing av MAVROS. For ROS1 er MAVROS sentralt for å kommunisere med sensorenheten, altså for å få GPS-posisjonen og heading til dronen, men den er ikke brukt av ROS2 (se Vedlegg M –µXRCE-DDS kontra µRtps). Open-source samfunnet har lagd smutthull som kan få MAVROS og ROS2 til å fungere sammen, men dette har kun ført til halvveisløsninger. Ardupilot er en åpen kildekode som muliggjør autonom kontroll av kjøretøy, og utgiveren har satt opp et prosjekt for å rette lys mot å få et MAVROS alternativ til ROS2 (ArduPilot Dev Team, 2023). Og ved at Melodic er nest siste ROS1 utgivelsen så er dette enda en faktor som gjør svermplattformen innviklet å videreutvikle da alt av kode er bygget på to utdaterte faktorer.

Dersom valget heller hadde falt på ROS2 Dashing ville både Python 3 blitt brukt og MAVROS ville vært utfaset, som hadde skapt mindre kompatibilitetsproblemer i svermsystemet og koden. Som også betyr at oppgaven om kollisjonsunnvikelse med stor

sannsynlighet kunne hatt færre legacy problemer. Eneste som taler for å velge ROS1 Melodic er at livsløpet er mer enn dobbelte av ROS2 Dashing sitt som kan ses i Figur 5-1, og den ble utgitt før så kan ha vært en mer stabil løsning som har blitt feilrettet i større grad. Men selv med disse argumentene så er valget av ROS1 Melodic å utsette et fremtidig problem som bygger teknisk gjeld, sett i retrospekt.

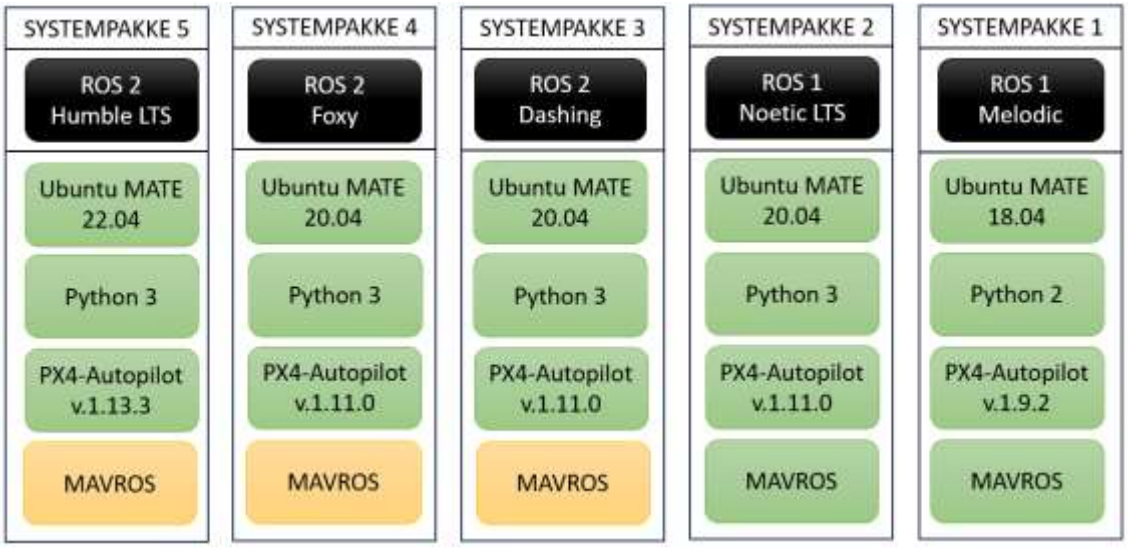
I tillegg til de tidligere nevnte utfordringene så har koden hindret svermplattformen fra å være forover-kompatibel på fastvaren til sensorpakken Pixhawk 4 som den teoretisk skulle vært kompatibel med, som da også ble enda et hinder i å få plattformen operasjonell. Koden har også låst systemet til å kun funke på v.1.9.2 av PX4-Autopilot som ble brukt under utviklingen av plattformen.

Valget av Ubuntu MATE 18.04 derimot er mer forståelig. Dette var eneste utgivelsen som var langtidsstøttet (som vist i Figur 5-1) under utviklingen av svermplattformen og utgivelse havnet i samme situasjon som ROS1 Noetic, at den ble utgitt like etter plattformen ble utviklet. Ubuntu MATE 18.04 er kompatibel med Python 3 og har ingenting med MAVROS å gjøre. Dette viser at dersom det hadde blitt valgt en ROS2 utgivelse, så ville største utfordringen med å modernisere et svermsystem, som er basert på Python 3 og har ingen MAVROS nødvendighet, ville vært å oppdatere operativsystem med tilhørende mellomvare. Noe som følger et fast livsløp og ikke burde by på de største utfordringer, slik som å oppdatere mobilen.

Plattformen består av mange deler som settes sammen til et felles system som vist Figur 5-1 og har et stort volum kode. Dette førte til en tidkrevende prosess med å oppnå mål 1, det å sette seg inn i virkemåten til svermen og skaffe en god nok forståelse av systemet som helhet for å kunne fortsette utviklingen av svermplattformen. En konsekvens av dette er også at inkompatibiliteter og noen problemstillinger blir oppdaget senere i prosjektperioden. For oppgaven førte dette til at omfanget av den tekniske gjelden i dette systemet ble oppdaget for sent. Så fremfor å gjennomføre moderniseringen som utfaser MAVROS helt og implementerer en helt annen måte for svermsystemet å kommunisere på var det mer tidseffektivt å bygge på den utdaterte svermplattformen for få den operasjonell.

Det vil alltid være fare for at et system kan bli foreldet, og denne sannsynligheten øker med tiden den har vært urørt. I tillegg er det klart at denne svermplattformen ble utviklet

på et tidspunkt som har gjort grunnlaget for videreutvikling veldig komplisert. Dette er enda en grunn til at plattformen bør moderniseres til systempakke 5 (Figur 5-2) der svermsystemet blir basert på Python 3 og MAVROS ikke er en del av det, slik at den er mer robust og kapabel til å håndtere fremtidige oppdateringer. Samtidig vil koden trenge en renovasjon der den håndterer overgangen fra MAVROS og oppgraderingen til Python 3, slik at den lettere kan oppgradere Maskinvare, operativsystemet, mellomvaren og fastvaren til sensorpakken. Dette vil også hjelpe til å holde tritt med livsløpene til Ubuntu MATE, Python, ROS og endringer som kommer fra åpenkilde samfunnet.



Figur 5-2: Dette er resultatene fra testingen. Systempakke 1 er den med de eldste utgivelsene, også blir de gradvis nyere der systempakke 5 er den nyeste.

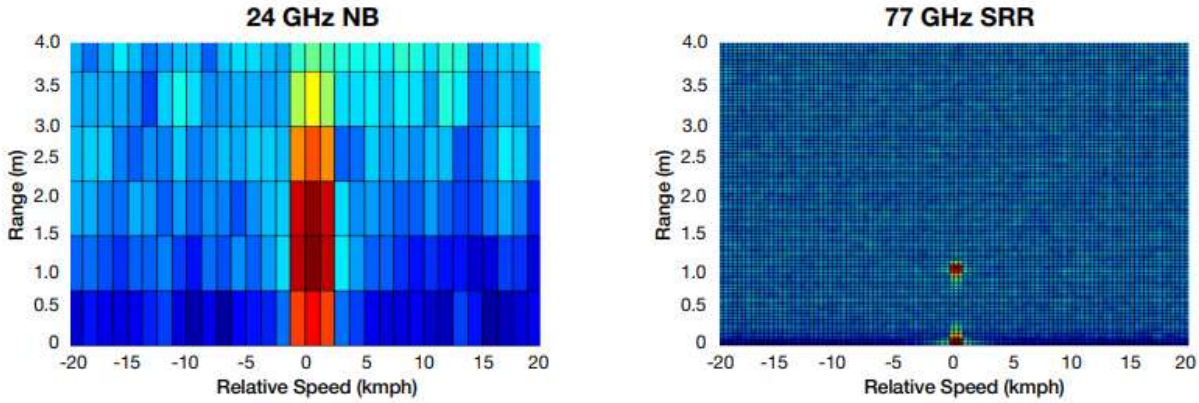
5.2 Radar

Oppgavens tredje mål er at enheten skal kunne innhente data fra HLK-LD2450 radaren, med den intensjon at dronen skal kunne oppfatte hindringer i umiddelbar nærhet og unngå å kollidere med dem. Videre brukes dataen i konstruksjonen av den ene unnvikelsesvektoren. Det endelige resultatet etter testingen var at datarammen var delt opp i forskjellige tabeller, hvor *firstTarget, secondTarget* og *thirdTarget* er arryene som inneholder informasjon om detekterte objekter. Ved første øyekast virker det altså som om målsetningen om å kunne lese av seriell data fra radaren er oppnådd, men hvis en ser

på intensjonen bak målsetningen kan det likevel virke som om det bare er delvis, hvor valg av radar, radarfrekvensen og

Valg av radar er den største grunnen til målsettingen ikke ble nådd. For det første gir datarammen en veldig begrenset mengde detekterte mål. I et komplekst operasjonsmiljø kan det godt tenkes at det finnes mer enn tre objekter i samme område som må unngås, og det er uvisst hvordan radaren vil filtrere eller selektere hvilke mål den registrerer og sender til Arduinoen. Derfor hadde det vært hensiktsmessig med en radar som evner å rapportere på en god del flere mål enn det HLK-LD2450 evner. På den andre siden vil det sette strengere krav til datastrømmen som sendes fra dataen, da det ikke vil være like enkelt å telle seg opp til enden av rammen slik som det gjøres i denne oppgaven. Da kan det i stedet være gunstig å ha identifisert en korrekt og stabil *frame tail* som man kan bruke til å detektere enden på datarammen. Dog kan det være at alle radarer med lik frekvens vil lide av samme problem, nemlig at de heller begrenses av deres fysiske egenskaper.

Den andre grunnen til at valg av radar kan være begrensende er radarfrekvensen. 24GHz er en eldre type millimeterbølgeteknologi, hvor nyere teknologi anvender millimeter-bølgeradarer på 77GHz. Radarer på denne frekvensen fører med seg tre store fordeler. Den første er bedre avstandsoppløsning og -nøyaktighet, hvor disse egenskapene har et inverst proporsjonalt forhold til båndbredden, som nevnt i delkapittel 2.5.2. Der radarer på 24GHz kun har 200MHz båndbredde, har 77GHz radarer en båndbredde på 4GHz, og dette resulterer i at 77GHz radarer har opptil 20 ganger bedre avstandsmåling enn en 24GHz radar (se Figur 5-3). En annen fordel er at hastighetsoppløsningen og -nøyaktigheten på sin side har et inverst proporsjonalt forhold til radarfrekvensen. Ettersom 77GHz er omtrent tre ganger så rask frekvens som 24GHz, vil hastighets-målingen også forbedres med samme faktor (Texas Instruments, 2023).



Figur 5-3: 24GHz radar versus 77GHz radar (Texas Instruments, 2023)

Feil! Fant ikke referansekilden.viser både avstands- og hastighetsmåling fra en 24GHz og 77GHz radar. Figur 5-3 viser tydelig hvor stor forskjell det er på oppløsningen til radarene. Den siste fordelen ved 77GHz radarer er at høyere frekvenser resulterer i mindre radarantenner. Med en 77GHz radar kan radarbrikken være tre ganger mindre enn en 24GHz radar, og dermed kan den integreres i systemer som har mindre marginer når det gjelder plass og vekt (Texas Instruments, 2023). Likevel finnes det ikke bare fordeler ved å endre radartypen, for som nevnt i kapittel 2.5.1 vil en høyere radarfrekvens også resultere i en kortere rekkevidde enn den nåværende på 6 meter. For å kontre dette er det mulig å øke radarens gain eller utsendte effekt, men dette vil igjen føre til at andre komponenter må oppskaleres (Fausa, 2014).

Uavhengig av radartype kan selve programmet som kjører på Arduinoen optimaliseres. Arbeidsoppgavene til programmet er rimelig primitive og utnytter ikke mulighetsrommet til databehandling som Arduinoen gir. Eksempelvis kunne måldataen blitt konvertert fra bytes til numeriske verdier uten særlig komplisert kode. Raspberry Pien kjører allerede store deler av svermsystemet, og vil på et eller annet tidspunkt nå terskelen når det gjelder prosesseringskraft. En slik fragmentering, hvor så mye som mulig gjøres på Arduinoen, vil kunne minke arbeidsmengden til Raspberry Pien til en viss grad. Koden kan i tillegg optimaliseres gjennom å eliminere forekomsten med «FFFFFFFF» i slutten av rammen, slik at *frame tailen* kan brukes for å detektere slutten på en ramme. Det kan virke som at «FFFFFFFF» oppstår når en verdi midt i rammen ikke blir skrevet ut, og det kan tenkes

at forskjellen i baud rate i kommunikasjonen mellom radar og Arduino og mellom Arduino og PC kan være årsaken for dette. Selv om tilsynelatende riktige verdier skrives ut til konsollen ved både 250000 og 9600 baud mellom Arduino og PCen, kan det være at begge kommunikasjonslinjene må kjøre på 256000. Hvis dette ikke lar seg gjøre kan en mulig løsning være at unnvikelsesvektoren baseres på et snitt av de innkommende verdiene over en viss tid. Samtidig vil dette gjøre at objektenes posisjon ikke vil oppdateres like hyppig, som kan minke dronens reaksjonsevne.

5.3 AIS

Det fjerde målet var at enheten skulle innhente AIS-data for å kunne unnvike fra fartøyer som benytter AIS for å oppnå sikrere navigasjon. Testingen viste at det var mulig å få inn AIS-data og behandle denne dataen slik at den kunne bli brukt i de to behaviour-programmene. AIS-databehandling var funksjonelt da det ble kjørt på en test PC som benyttet seg av Python3. Da det ble kjørt på en dronene fungerte ikke programmet grunnet inkompatibiliteten mellom bibliotekene og python versjon nevnt i kapitel 5.1. Resultatet fra den første fysiske testen av implementering viste at AIS-mottakingen måtte deaktiveres for at sverm plattformen skulle kjøre ordentlig. Målet om å kunne hente inn AIS-meldinger og behandle disse på dronen er derfor ikke nådd, ettersom at AIS-meldingene ikke kan hentes inn på dronene og behandles der. Valget om å ta i bruk Kystverket sin tjeneste og dekode meldingene med pyton biblioteket pyais var mulig feil valg. Å bruke en annen dekoder enn pyais biblioteket kunne vært en mulig løsning. Dersom node-red metoden for innhenting og dekoding av AIS-meldinger hadde blitt tatt i bruk kunne enheten mottatt dataen fra den allerede etablere kommunikasjonen mellom GCS og enhetene. Python biblioteket geopy kunne blitt byttet ut med formler som gjør de samme beregningene som funksjonene i biblioteket. Da kunne inkompatibiliteten mellom bibliotekene og python versjonen vært unngått ved å unngå bibliotekene helt. Å bruke dAISy-hat AIS-reciever kunne også vært et interessant alternativ. Da hadde dataen kommet direkte inn til RPI via en fysisk kobling i stedet for å belage seg på internett for innhenting av AIS-meldinger for så sende denne til enheten. Her hadde fortsatt

problemstillingen om hvordan AIS-meldingene skal dekodes fortsatt vært et problem som måtte løses.

5.4 Lokalisering og unnvikelse av hindringer

Resultatene fra testingen har vist at det fungerer å få inn AIS-meldinger fra faktiske fartøyer og behandle dataen i disse meldingene på en test PC. Programmets kalkuleringer med disse dataene gjør det mulig å predikere hvor fartøyet fra meldingen kommer til å være, samt lage unnvikelses-vektorer bort fra disse posisjonene. Det er dog verdt å nevne at troverdigheten til de verktøyene brukt i testen bestemmer hvor gode resultatene fra disse testene er. Det ble brukt to forskjellige uavhengige verktøyer som gav relativt like svar. Dette kan tyde på at troverdigheten til verktøyene er intakt.

Resultatene fra den siste matematiske testen viser at båten vil unnvike kraftig fra fartøyet og forlate sin opprinnelige retning, se Figur 4-23. Dette viser at programmet vil styre unna en kontakt, men styrken av unnvikelse må eventuelt fremdeles kalibreres. Dronene er kun en meter lange og har ikke en ekstremt høy topphastighet. Det vil nok derfor ikke være nødvendig å gi opp kursen helt når kontakten er en kilometer unna. Det kommer naturligvis også an på nødvendigheten til båten for å holde denne kursen. Dersom oppdraget bare er patruljering i nærområdet vil ikke en unnvikelse her, for så å ta opp igjen kursen senere være helt ødeleggende for oppdraget.

Opp mot målet om at enheten skal kunne detektere om den er på kollisjonskurs eller ikke så må det sees i konteksten av hvordan svermplattformen opererer. Svermens navigering følger ikke presise ruter og vil være vanskelig å forutse posisjoner frem i tid. Det kan da heller ikke detekteres kollisjonskurs direkte. Måten det håndteres i oppgaven er at den unnvikes basert på nærhet til egen posisjon, ikke direkte om det hadde blitt en kollisjon i fremtiden. Gitt da oppførselen til svermen kan dette virke som en god metode å «detektere» og unnvike kollisjon. Resultatet for testene av unnvikelsen i praksis var ikke gode nok til at man kan si at unnvikelsen fungerer når den settes på svermplattformen. Det kan derfor sies at strategien for unnvikelsen står til målets krav, men implementeringen i praksis på svermplattformen ikke står til målets krav.

For problemet med at unnvikelsesvektoren overstyrer bevegelsen helt, noen ganger unødig, finnes det alternativer. Det ene alternativet er å kalibrere vektoren bedre. Dette kan gjøres på flere måter, men testing i praksis er nødvendig. Dersom man vet snuradiusen bedre kan funksjonen tilpasses deretter, selvfølgelig med forbehold om handling i tilstrekkelig tid og ikke bare en unnvikelse i siste liten. Denne snuradiusen kan også brukes til å se på en form for «nødstopp». I et tilfelle der en kontakt blir oppdaget først når den er nærmere enn det svingradiusen gir mulighet for å unnvike så vil unnvikelsesvektoren mulig føre til kollisjon. Dette ved at det vil lages en unnvikelsesvektor med en stor størrelse som vil få båten til å sette farten opp for å forsøke å unnvike, men svingradiusen gjør at de treffer uansett. I slike tilfeller kan det tenkes at en full stopp av båten er en bedre løsning, eller eventuelt at den prøver å rygge – som ikke var tiltenkt i autopiloten til svermen så langt.

Et annet alternativ er å legge til en normalisering av unnvikelsesvektoren. På denne måten kan man skalere vektoren slik at det er kun ved de høyeste verdiene av funksjonen for vektorens størrelse at opprinnelig bevegelsesvektor blir overkjørt. Her også kan testing være hjelpsom for å finne en god balanse mellom når det skal overstrides.

Overstyringen er ikke nødvendigvis like brå som den ser ut i Figur 4-23: Visualisering og kalkulering av resulterende vektor i Geogebra. A er egen posisjon. u er vektoren svermen styrer etter i øyeblikket AIS-meldinger kommer inn, v er unnvikelsesvektoren som opprettes av AIS-meldingen, w er den resulterende vektoren som droner vil styre i.. I hvert fall ikke for BOID oppførselen. I BOID-oppførselen så er en av vektorene som bestemmer bevegelsesretning til dronen *alignement* eller samstilling. Denne vektoren jobber for å få fartøyene til å styre i samme retning. Dette gjøres ved at dronene publiserer sin egen nåværende bevegelsesvektor til alle dronene. Ut ifra denne så regnes det ut en gjennomsnitts-vektor for alle dronene som brukes i oppførselen for å bestemme ny bevegelsesretning.

Denne vektoren vil ha noe å si for hvordan bevegelsen ser ut over tid. Dette forbeholder at kun den ene dronen har fått inn AIS-meldingen som unnvikelsesvektoren blir laget av. Dersom alle dronene får denne så vil alle styre i relativ lik retning og gjennomsnitts-vektoren vil da ende i samme retning. Om vi ser bort ifra dette så kan vi ikke si noe om hvordan det spilles ut i realiteten da dette krever testing av systemet. Det kan tenkes at

denne sannsynligvis ikke hadde vært nok til å skape en god bevegelse basert på hvor stor unnvikelsesvektoren er sammenlignet med de andre vektorene. En endring i den nåværende metoden er nok nødvendig.

Før dette blir relevant er det viktig å se på selve implementeringen av denne unnvikelsesvektoren på dronen. Mål 6 spesifiserer at enheten skal kunne endre bevegelsesretning for å unngå kollisjon. Resultatene fra testingen av svermen viste at den tiltenkte måten å implementere AIS-basert unnvikelse på den eksisterende plattformen kun fungerte i simulerte tester. Det funket som nevnt ikke da denne legges til i svermplattformen. Mye av dette fordi det var problematisk å benytte seg av nyere biblioteker som krever nyere Python versjoner. For å fikse dette ble det påbegynt en metode der programmet som tar imot AIS-meldingene og gjør disse om til en vektor opererer på et separat virtuelt miljø. Virtuelle miljøer er en måte å adskille programmer og hvilke pakker og applikasjoner de benytter. På denne måten kan det være et miljø der alt er oppdatert til de nyeste standardene og benytte oss av en form for «mellommann» for å kommunisere vektorene på tvers av miljøer. Denne «mellommannen» var tiltenkt en tekstfil som innehold størrelsen og vinkelen til vektoren. Denne funksjonaliteten kunne vært på hver og en av dronene. Valget av tekstfil til dette er fordi skriving og lesing til fil er en funksjon som både python2 og python3 er kapable til. Dessverre var det her ikke mer tid igjen til programmering og testing. I en simulert setting med utregninger av vektoren vil unnvikelsen i teorien fungere.

Selv om det har vært en del problemer for å bruke radar og AIS i praksis på svermen, så er det ingen prinsipielle problemer som sier at svermen ikke skal kunne navigere trygt og utføre operasjoner. Det skal nå drøftes hvilke muligheter dette presenterer for sjømilitære operasjoner.

5.5 Mulighetene til dronesverm i en sjømilitær kontekst

Droner har i nyere tid vist seg som en ny aktør på slagmarken. Spesielt USA har brukt droner mye i kontraterrorisme-operasjoner, blant annet i Midtøsten. (Savell, 2021). I disse tilfellene har dronene som regel vært brukt mot en fiende med liten evne til å forsvare seg mot luftbaserte enheter. Til nyere tid har droner også vist seg effektive i Ukraina, selv mot en motstander med antiluft-kapabiliteter. Særlig her har droner som et

relevant krigføringsmiddel utviklet seg kraftig, hvor nye innovasjoner gjøres både innen tiltak og mottiltak. Små og relativt billige droner fra det kommersielle markedet kan brukes til informasjonssamling, og modifiseres med sprengladninger for å brukes til angrep mot mål. (Isabelle Khurshudyan, 2022).

Disse har som regel vært luftbaserte, men også i det maritime domenet har droner sett sitt bruksområde. Nå nyligst i krigen i Ukraina har vi sett selvmords droner bli brukt mot russiske fartøy som har ført til at russiske krigsskip har blitt slått midlertidig ut av spill. (Stern, 2023) Dette viser hvordan små relativt billige droner kan brukes til å treffe fiendens større kapabiliteter for å gi et overtak i krigen. Disse dronene har krevd operatører til å styre dem noe som gjør de utsatt for blant annet «jamming», som vil føre til at dronene blir ubrukelige når kommunikasjonen mellom operatør og system forsvinner. I tillegg krever dette kompetent personell som kan operere dronene noe som påvirker skalerbarheten til denne typen krigføring. Å øke autonomiseringen av disse dronene kan være en løsning til dette problemet. Dronene kan operere for seg selv og drive de operasjoner de er tiltenkt uten å trenge input fra en operatør for hver eneste bevegelse. Dersom du setter flere droner sammen til å operere som en enhet i en dronesverm kan du øke arealet med sensordekning, redusere antall operatører samt at du kan ta tap uten at hele operasjonen mislykkes. For svermplattformen i denne oppgaven kunne dette bli gjort ved å utvikle en til oppførsel for svermen. Den kunne for eksempel fungere ved at operasjonsområdet blir delt opp i soner der dronene finner den sonen de er nærmest og begynner patruljering i dette området. Dronene kommuniserer hvilken sone de patruljerer og er overordnet programmert til å være likt fordelt utover. På denne måten kunne da en sone bli overtatt av en annen drone dersom det mistes signal til den første. Det kunne da også prioriteres soner etter viktighet.

Dronesvermer kan ha flere bruksområder i krigføring. De kan spres over et større område for å drive overvåkning av havområder uten å risikere at egne større fartøy selv blir oppdaget. Dronene vil naturligvis ikke kunne ha like lang radarrekkevidde som det et konvensjonelt krigsskip kan ha grunnet størrelsen til dronene begrenser størrelsen på radarsystemet og antenner. Dessuten er rekkevidden til radarer ofte begrenset av jordkrumningen, noe som er et større problem når dronene er i lavere høyde. Dronens størrelse vil dog gi de et lite radartverrsnitt samt at de ligger lavt i vannet. Disse

egenskapene kan hjelpe dronene komme seg nærmere enn det et større fartøy kunne gjort, noe som kan kompensere litt for den kortere sensorrekkevidden. Dessuten er kostnaden for tap av droner betydelig mindre enn kostnaden for tapen av en fregatt.

Når det er snakk om teknologi som denne blir det også naturlig å snakke om etikken rundt dette. Dersom vi går så langt til å ta i bruk autonome droner lastet med sprengstoff, kan vi være tilstrekkelig sikre på at de klarer å finne rett mål at vi kan si det er etisk akseptabelt? Kan de ta hensyn til krigen folkerett? Og hva med de operatørene som starter opp systemet og sender det av gårde? Kommer man til å bli så skjermet fra konflikten at det å ta liv ikke blir tatt seriøst eller vil de kjenne på «sniper's syndrome», altså skyldfølelse ettersom målet ikke utgir noen direkte trussel mot deg. (Henschke, 2019)Her er det naturligvis ikke noen fasit, men det er uansett viktig å poengtere at disse spørsålene bør diskuteres. Det må også bestemmes hva som er målepunktet for påliteligheten her. Automatisk gjenkjenning av mål vil sannsynligvis ikke være feilfritt, men det er ikke mennesker heller. Risikoen man er villig til å ta vil også avhenge av hvor presset situasjonen er. Da må vi heller snakke om hvilken feilrate vi kan akseptere.

Økonomi er også et perspektiv av relevans. Det er også ønskelig å gjøre dette så billig som mulig med såkalte «hyllevarer», som gjør at oppgaven kan vise et interessant punkt med hvor lavterskel et slik prosjekt kan være sett fra et økonomisk ståsted.

5.5.1 Mulighetene oppgaven presenterer

Nå som det har blitt sett på dronesverm som konsept generelt, skal det nå ses på hvordan det som i prinsipp er mulig for svermen kan brukes i en sjømilitær kontekst. Størrelsen til fartøyene vi har brukt i denne oppgaven er en begrensende faktor på hvor de kan operere. Som vi nevnt i innledningen til oppgaven er Norges energi viktig for hele Europa. Det er svært ressurskrevende å overvåke og beskytte disse fasilitetene. Dronene her hadde vært en ypperlig ressurs til å gi støtte i form av overvåkning og muligens beskyttelse. Dessverre ville operasjoner i et slikt område kreve en stor oppskalering i størrelse for stabilitetens skyld i krevende farvann, batteritid og drone infrastruktur. Det å håndtere et stort antall droner er omfattende og krever teknisk kompetent personell. Dersom vi ser for oss at vi enkelt kunne oppskalert størrelsen slik at dronene håndterer det tøffe farvannet rundt

plattformene så kunne en fremtidig løsning være noe som det Portugal går for. De går for en form for moderskip løsning. Med moderskip menes et fartøy som kan frakte dronene, utføre enkelt vedlikehold, lade og kanskje til og med gi kommandoer. Det prosjekterte skippet beskrives slik: “With a total length of 107.6 meters, the "D.João II" will function as an aerial, terrestrial and underwater 'drone carrier' […]” (defense-aerospace.com, 2023). En slik form for infrastruktur kunne gjort det mulig for en form for RAS, noe som hadde økt både fleksibiliteten og utholdenheten til en sverm. Man kunne da spart batteri og tid ved å frakte dronene litt nærmere Area of Operation samt lade dem der de er i stedet for å måtte få de tilbake til land.

Dersom vi ser på dagens teknologi og mindre skala av innvestering i utstyr vil nok operasjoner i tilløp til kai og oljeraffinerier være områder det er fornuftig å operere dronesvermer i. I sin oppgave i 2022 viste Røsand og Hollup som et proof of concept at det er mulig å implementere objekt deteksjon på en dronesverm som kunne gjenkjenne en egendefinert klasse av fartøy. (Hollup & Røsand, 2022). Med gode nok datasett så kan svermen trenes til å gjenkjenne ønskede mål i området. Det å få gode nok datasett er i seg selv en vanskelig oppgave, men den videre kompliseres av at vi kan ikke si for sikkert hvordan en fiende vil se ut i disse områdene. I PST sin nasjonale sikkerhetsvurdering for 2023 ser de det som lite sannsynlig med russiske sabotasjeaksjoner på norsk jord, men dersom Russlands vilje til å eskalere konflikten øker kan sabotasjehandlinger mot strategiske mål i Norge. Her ser de petroleumssektoren som svært utsatt. En slik handling «vil mest sannsynlig bli utført på en måte som gjør det utfordrende å tilskrive handlingen til aktøren som står bak.» (PST, 2023, ss. 6-23) . Dette presenterer et stort problem for bruken av sverm til slike oppgaver. Når det ikke er et klart bilde på hvordan en trussel i området vil se ut vil det være tilnærmet umulig å trene en maskinlæringsalgoritme til å gjøre dette. Det å introdusere en menneskelig kontrollfunksjon kunne muligens vært til hjelp til dette. Dronene kan sende inn det de ser til en kontrollstasjon som kan gi en menneskelig vurdering av trusselbilde i området. Dette kan bli en svært krevende oppgave for en operatør i et trafikkert område, men kan utføres av færre mennesker enn det det ville tatt å operere et fartøy i området.

Dronene kan også ha en avskrekkende effekt på en fiende. Dersom fienden ikke er kjent med kapabilitetene til dronesvermen kan tilstedeværelsen av en slik sverm virke som en

trussel for deres oppdrag. Det viser også vilje til å tildele ressurser til forsvar av infrastruktur.

De to operasjonsområdene beskrevet har omhandlet en defensiv overvåkings oppgave, men det ligger også offensive muligheter i konseptet. Det å sende en sverm lastet med eksplosiver mot et fiendtlig fartøy har potensiale for å påføre fienden skade uten å risikere egen sikkerhet i like stor grad som et direkte engasjement ville gjort. Å angripe med en hel sverm på likt ville også gjort det vanskeligere for en fiende å forsvare seg mot. Oppgaven til Røsand og Hollup viste også at de fikk ut en relativ peiling på et gjenkjent fartøy. Denne peilingen kan brukes til å lage en vektor mot fartøyet man ønsker å ta ut. Ved hjelp av unnvikelse ved hjelp av AIS-meldingene kan dette hjelpe til å øke sannsynligheten for at riktig mål blir valgt. Meldingene kan til og med bli brukt til targeting hvis det er mål som er godkjente i forhold til lov. Denne metodens effektivitet kan senkes betraktelig dersom fienden bruker «GPS-spoofing» for å utgi seg for å være et annet fartøy. Men dette vil ikke fungere dersom det er god nok objektgjennkjenning.

Et annet bruksområde for dronesvermen er å gi måldata. Ved hjelp av kamera gjenkjennelse kan dronene videre programmeres til å sende ut egen posisjon når målet er identifisert. Større enheter kan da bruke denne måldataen til *over the horizion targeting* med for eksempel en MSN-missil som vil kunne bruke sine sensorer til å finne målet i området. *Over the horizion targeting* er et konsept at via måldata fra andre enheter kan en enhet engasjerte fartøy utenfor egen radars horisont. (Simensen, 2023)

Andre muligheter for offensive operasjoner er dersom man videreutvikler fartøyet til å kunne vært å sette på en laser på fartøyet som var programmert til å følge retningen kamera detekterer målet i. Da kunne dronene brukes som en form for JTAC som lyser opp fienden så fienden kan bekjempes med en laser guidet bombe.

6 Konklusjon

Ved å implementere radar og AIS på en eksisterende plattform for dronesverm har oppgaven økt situasjonsforståelsen til svermen i den hensikt at svermen skal kunne drive autonom kollisjonsunnvikelse. Testplattformen som skulle rekonstrueres var et legacy-system, noe som begrenset muligheten til å gjenskape systemet slik det var i 2019. Mulighetene til å implementere nye løsninger ble derfor begrenset grunnet inkompatibilitet mellom operativ-systemet og de ulike delprogrammene.

Systemet har delvis oppnådd målsetningene for oppgaven, og har vist en prinsipiell gyldighet for fremgangsmåten. Svermplattformen er gjort operasjonell gjennom nødvendige oppdateringer og utskiftninger av programvare, og fremtidig kompatibilitet har blitt kartlagt for å tilrettelegge for en fremtidig modernisering av plattformen. Radarsystemet er ikke fullstendig integrert, da program for unnvikelsesvektoren for radar ikke er utviklet ennå. Radarsystemets egnethet må fremdeles testes i praksis. Derimot evner AIS-systemet å innhente data og beregne nødvendige verdier som trengs for beregning av en unnvikelsesvektor i simulerte tester, men må justeres for å kunne legges til på svermplattformen. En kalibrering for hensiktsmessig vekting av hindre i dronenes *behaviour* står fremdeles igjen. Radar og AIS-systemet er dermed prinsipielt brukbare, og tilpasningen av svermoppførselen er skissert. Dermed viser oppgaven hvordan det kan økes dronesvermens evne til å operere i mer komplekse omgivelser.

Det ligger mange muligheter for bruk av droner i en sjømilitær kontekst, både i form av informasjonsinnhenting, defensive operasjoner og offensive operasjoner. Felles for disse er at det vil kreves utdanning av teknikere som kan vedlikeholde plattformene og det krever infrastruktur for at dronene kan vedlikeholdes.

6.1 Muligheter for videreutvikling

Under utviklingen av oppgaven har det blitt møtt på flere hindringer som kan håndteres bedre ved en senere anledning. Under så vil det bli nevnt forslag til videreutvikling og tiltak av denne svermplattformen.

6.1.1 Full modernisering av svermplattformen

I denne oppgaven er det blitt erfart kompleksiteten av svermplattformen, og behovet den har for en grundig modernisering. Dette må starte med nyeste Ubuntu MATE OS, Python3, og kompatibelt og langtidsrettet ROS2 utgivelse på den eksisterende maskinvaren slik at den tekniske gjelden blir tilsvarende borte. Med dette fundamentet så er det mulig å fortsatt bruke mye av koden som er brukt, enten om det er open-source software eller kodet av tidligere oppgaver, slik at samme datainnhenting fungerer og plattformen er mer robust for fremtidige implementeringer og oppdateringer. Dette vil igjen gjøre plattform enklere å videreutvikle.

6.1.2 Utbedring av unnvikelsesvektor

Programmet som regner ut unnvikelsesvektor har mye rom for forbedring. Det er muligheter her for å sette en fast oppdateringsrate på innhenting av AIS-meldinger. På denne måten kan man også bestemme hvor lenge unnvikelsesvektoren for en kontakt påvirker styring, for eksempel bruke posisjons prediktoren til å se hvor lenge man trenger å vike for fartøyet. Som nevnt i drøftingen kan å legge til en skalering av unnvikelsesvektoren også gi dronen en bedre navigasjonsevne. En annen addisjon til systemet kunne vært til å bruke AIS-kontakten sin *course* for å beregne vektoren til å unnvike på en sikrere måte.

6.1.3 Kompatibel AIS behandling

Dersom en ikke går for en modernisering av systemet så må det utvikles en annen måte å hente inn og dekode AIS data. De to metodene nevnt i implementering og drøfting kan her være mulige punkter å jobbe ut ifra. DAISy Hat metoden krever å finne en måte å dekode meldingene i Python2 på mens Node-red metoden krever at man ser nærmere på trafikk på sendingen.

7 Bibliografi

Amazon. (2023, november 23). *PX4 Pixhawk 4 FMUv5 Autopilot with NEO-M8N GNSS and Power Management Board px4combo*. Retrieved from Amazon: https://www.amazon.sa/-/en/Pixhawk-Autopilot-NEO-M8N-Management-px4combo/dp/B07DQDXQSH

Arduino. (2023, november 23). *Arduino Uno Rev3 SMD*. Retrieved from Arduino Store: https://store.arduino.cc/products/arduino-uno-rev3-smd

ArduPilot Dev Team. (2023). *List of Suggested Projects for GSoC 2023*. Retrieved from Ardupilot: https://ardupilot.org/dev/docs/gsoc-ideas-list.html#ros2-mavros-support-for-ardupilot

Atea. (n.d.). *Zyxel WAH7601 Portable Router - Mobilt hotspot - 4G LTE - 150 Mbps - 802.11b/g/n*. Retrieved November 29, 2023, from ATEA: https://www.atea.dk/eshop/product/zyxel-wah7601-portable-router/?prodid=4946818

avXperten. (n.d.). *Kingston FCR-HS4 USB 3.0 Kortleser (CF/SD/MicroSD/USB)*. Retrieved November 29, 2023, from avXperten: https://www.avxperten.no/kortlaeser/kingston-fcr-hs4-usb-3-0-kortlaeser-cf-sd-microsd-usb.asp?gad_source=1&utm_source=daisycon&utm_campaign=daisycon_Onlinepriser.no&utm_medium=affiliate

Azab, A. (2022, 09 20). *ROS/Concepts*. Retrieved from wiki.ros.org: http://wiki.ros.org/ROS/Concepts

Bolling, G. R. (2021, November 25). *Autonomi - Hva betyr det?* Retrieved from Stratagem: https://www.stratagem.no/autonomi-hva-betyr-det/

Cadence Design Systems. (2023, 11 21). *MmWave Radar Applications and Advantages*. Retrieved from Cadence Design Systems: https://resources.system-analysis.cadence.com/blog/msa2022-mmwave-radar-applications-and-advantages

Chernyak, A. Z. (2023). *Kompatibilitetstesting – hva er det, typer, prosesser, egenskaper, verktøy og mer!* Retrieved from ZAPTEST: https://www.zaptest.com/no/kompatibilitetstesting-hva-er-det-typer-prosesser-egenskaper-verktoy-og-mer

defense-aerospace.com. (2023, 11 24). *Portugal Orders New PNM Helicopter and Drone Carrier*. Retrieved from defense-aerospace.com: https://www.defense-aerospace.com/portugal-orders-new-pnm-helicopter-and-drone-carrier/

DNV. (2022). *Energy Transition Norway 2022.* Oslo: DNV.

Engineering Projects. (2023, november 18). *Radar Waveform and Range Determination*. Retrieved from Engineering Projects: https://bestengineeringprojects.com/radar-waveform-and-range-determination/

Fausa, L. (2014). Radar. In L. Fausa, *Halvlederfysikk og elektrooptikk. Mikrobølger/radar* (p. 76). Bergen: Sjøkrigsskolen.

Forsvarets Forskningsinstitutt. (2020). *Teknologiske trender Muligheter og utfordringer for fremtidens forsvar.* Retrieved from FFI: https://www.ffi.no/publikasjoner/arkiv/teknologiske-trender-muligheter-og-utfordringer-for-fremtidens-forsvar

Forsvarets Forskningsinstitutt. (2023, Oktober 6). *Statsbudsjettet: Forsvaret får nytt system for maritim minerydding*. Retrieved from NTB Kommunikasjon: https://kommunikasjon.ntb.no/pressemelding/18006433/statsbudsjettet-forsvaret-far-nytt-system-for-maritim-minerydding?publisherId=17846797&lang=no

Forsvarskommisjonen. (2023). *NOU 2023: 14.* Oslo: Forsvarskommisjonen.

Frąckiewicz, M. (2023, Juni 15). *Hvordan fungerer en drones avanserte svermintelligens og selvorganiserende system?* Retrieved from TS2: https://ts2.space/nn/hvordan-fungerer-en-drones-avanserte-svermintelligens-og-selvorganiserende-system/#gsc.tab=0

Grønning, T., & Dimmen, S. D. (2022, Mars 22). *Sjøforsvaret skal styrkes – har ikke folk nok til å gjøre det*. Retrieved from NRK:

https://www.nrk.no/tromsogfinnmark/sjoforsvaret-mangler-folk-_-styrkes-med-800-millioner_-men-mangler-ansatte-1.15902367

Hareide, O. S., Relling, T., Pettersen, A., Sauter, A., Mjelde, F. V., & Ostnes, R. (2018, Oktober 1). *Fremtidens autonome ubemannede kapasiteter i Sjøforsvaret.* Retrieved from Semantic Scholar: https://www.semanticscholar.org/paper/Fremtidens-autonome-ubemannede-kapasiteter-i-Hareide-Relling/5c8150a75aad425f0aed355e02f2cf98c5a60937

Hellesnes, A. H., & Lyssand, K. A. (2019). *Plattform for sverm. Fra store avanserte plattformer til mange små*. Retrieved from FHS Brage: https://fhs.brage.unit.no/fhs-xmlui/handle/11250/2673330

Henschke, A. (2019, 09 28). *Modern soldiers can kill a target on computer, then head home for dinner — and it's giving them 'moral injury*. Retrieved from ABC News: https://www.abc.net.au/news/2019-09-29/unmanned-combat-drone-pilots-moral-injury-warfare-dissonance/11554058

Hicks, K. (2023, August 28). *Deputy Secretary of Defense Kathleen Hicks Keynote Address: 'The Urgency to Innovate' (As Delivered)*. Retrieved from U.S. Department of Defence: https://www.defense.gov/News/Speeches/Speech/Article/3507156/deputy-secretary-of-defense-kathleen-hicks-keynote-address-the-urgency-to-innov/

Hi-Link. (2023, mai). HLK-LD2450 Motion target detection and tracking module Instruction manual. Shenzhen, Guangdong, Kina.

Hollup, O., & Røsand, M. (2022). *Implementering av objektdeteksjon på maritim dronesverm. Bruk av kunstig intelligens til maritim overvåkning på ubemannede fartøy*. Retrieved from FHS Brage: https://fhs.brage.unit.no/fhs-xmlui/handle/11250/3069283

IGISMAP. (2023, 11 3). *bearing-angle*. Retrieved from map igismap: https://map.igismap.com/bearing-angle/

Isabelle Khurshudyan, M. U. (2022, 12 2). *Russia and Ukraine are fighting the first full-scale drone war*. Retrieved from The washington post:

https://www.washingtonpost.com/world/2022/12/02/drones-russia-ukraine-air-war/

Kanade, V. (2023, Februar 23). *What Is Backward Compatibility? Meaning, Uses, Benefits, and Challenges*. Retrieved from Spiceworks: https://www.spiceworks.com/tech/devops/articles/what-is-backward-compatibility/

Kessler, G. C. (2023, 9 11). *AIS RESEARCH USING A RASPBERRY PI (2022 Update)*. Retrieved from garykessler.net: https://www.garykessler.net/library/ais_pi.html

Kjerstad, N. (2022, November 2). *AIS*. Retrieved from Store Norske Leksikon: https://snl.no/AIS#-Litteratur

Korme, C. (2023, mai 5). *Forsvarskommisjonen: Behov for omfattende maritim satsing*. Retrieved from Norges Rederiforbund: https://www.rederi.no/nyheter/forsvarskommisjonen-behov-for-omfattende-maritim-satsing/

Kpler. (2023, 11 3). *Live map*. Retrieved from Marine Traffic: https://www.marinetraffic.com/en/ais/home/centerx:5.314/centery:60.398/zoom:18

Kystverket. (2023, 11 14). *Tilgang på AIS-data*. Retrieved from Navigasjonstjenester/AIS: https://www.kystverket.no/navigasjonstjenester/ais/tilgang-pa-ais-data/

Kystverket. (n.d.). *Hva er AIS?* Retrieved from Kystverket: https://havbase.kystverket.no/havbase_report/doc/AIS.pdf

Luna, J. C. (2022, 8). *Python 2 vs 3: Everything you need to know*. Retrieved from datacamp: https://www.datacamp.com/blog/python-2-vs-3-everything-you-need-to-know

MathWorks. (n.d.). *What Is Object Detection?* Retrieved from MathWorks: https://se.mathworks.com/discovery/object-detection.html

NATO Shipping Centre. (2021, 10 22). *AIS (Automatic Identifaction System) overview*. Retrieved from NATO Shipping Centre:

https://shipping.nato.int/nsc/operations/news/2021/ais-automatic-identification-system-overview

Naval Air Warfare Centre Weapons Division. (2013). RF Atmospheric Absorption / Ducting. In N. A. Division, *Electronic Warfare and Radar Systems Engineering Handbook* (pp. 5-1 - 5-8.9). Washington: Naval Air Systems Command.

NFAS. (n.d.). *Norsk Forum for Autonome Skip*. Retrieved from https://nfas.autonomous-ship.org/?lang=no

Pecka, M. (2023, Juni 13). *Migration*. Retrieved from ROS.org: http://wiki.ros.org/noetic/Migration

PST. (2023). *Nasjonal Trusselvurderng 2023.* Oslo: PST.

Python Software Foundation. (2023). *Status of Python versions*. Retrieved from Pythons Developer's Guide: https://devguide.python.org/versions/?fbclid=IwAR0-00HNivAI-VRa8UMlmSFRDz4RYIffiy6gN3OUDJ_SuaNhYg6SM4NdhxA

Radartutorial. (2023, november 29). *Frequency-Modulated Continuous-Wave Radar (FMCW Radar)*. Retrieved from Radartutorial: https://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html

Radartutorial. (2023, november 29). *Range Resolution*. Retrieved from Radartutorial: https://www.radartutorial.eu/01.basics/Range%20Resolution.en.html

Raspberry PI. (2023, november 23). *Raspberry Pi 3 Model B+*. Retrieved from Raspberry Pi: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

Regjeringen. (2021, oktober 4). *Havnasjonen Norge*. Retrieved from Regjeringen: https://www.regjeringen.no/no/tema/hav/innsiktsartikler/havnasjonen-norge/id2605291/

Regjeringen. (2023, Oktober 6). *Statsbudsjettet 2024 -Moderne førerløse mineryddere til Sjøforsvaret*. Retrieved from Regjeringen.no: https://www.regjeringen.no/no/aktuelt/moderne-forerlose-mineryddere-til-sjoforsvaret/id2998910/

Richter, L. M. (2023, 10 2). *pyais*. Retrieved from Github: https://github.com/M0r13n/pyais

Sandberg, S. (2023, Januar 31). *Hva er et "Legacy system"? -og hvorfor er det så problematisk?* Retrieved from documaster: https://www.documaster.com/blogg/hva-er-et-legacy-system-og-hvorfor-er-det-s%C3%A5-problematisk

Savell, S. (2021). *United states Counterterrorism Operations 2019-2020.* Providence: Watson Institute for International and Public Affairs.

Scheel, J. (2023, desember 1). *Just the FAQs about Little Endian.* Retrieved from IBM: https://www.ibm.com/support/pages/just-faqs-about-little-endian

Simensen, I. (2023, 6 6). *KONGSBERG receives new Naval Strike Missile order for the U.S. Navy*. Retrieved from Kongsberg: https://www.kongsberg.com/no/newsandmedia/news-archive/2023/kongsberg-receives-new-nsm-order-for-the-u.s.-navy

Stern, D. L. (2023, 10 13). *Ukraine hits Russian navy ships with sea drones*. Retrieved from The Washington Post: https://www.washingtonpost.com/world/2023/10/13/russia-ukraine-boats-drones-sea/

SunEarthTools. (2023, 11 3). *Distance*. Retrieved from SunEarthTools: https://www.sunearthtools.com/tools/distance.php

Tandberg, E., Engerengen, L., & Jarslett, Y. (2023, Januar 25). *drone*. Retrieved from Store Norske Leksikon: https://snl.no/drone

Texas Instruments. (2023, november 29). *Moving from legacy 24 GHz to state-of-the-art 77 GHz radar*. Retrieved from Texas Instruments: https://www.ti.com/lit/wp/spry312/spry312.pdf?ts=1700485888338&ref_url=https%253A%252F%252Fwww.ti.com%252Ftool%252FAWR1642BOOST

Thorsnæs, G. (2023, juni 26). *Norges geografi*. Retrieved from Store norske leksikon: https://snl.no/Norges_geografi

U.S. Department of Defence. (2023). *Military and Security Developments Involving the People's Republic of China.* Washington: U.S. Department of Defence.

Ubuntu. (n.d.). *The Ubuntu lifecycle and release cadence*. Retrieved from Ubuntu: https://ubuntu.com/about/release-cycle

Viola, M. (2017, 11 9). *adding-two-polar-vectors*. Retrieved from Mathematics Stack Exchange: https://math.stackexchange.com/questions/1365622/adding-two-polar-vectors

Wegmatt. (2023, 11 14). *Daisy hat reciever for the Raspberry PI*. Retrieved from Products: https://shop.wegmatt.com/products/daisy-hat-ais-receiver

Wikipedia. (2023, november 18). *Radar.* Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Radar

Aarønæs, L. (2022, Juni 24). *Droner utfordrer NATOs radarer*. Retrieved from FFI: https://www.ffi.no/aktuelt/feature-artikler/droner-utfordrer-natos-radarer

Vedlegg

Vedlegg A – Oversikt over relevante livsløp

Som nevnt i oppgaven finnes det ulike utgivelser med livsløp som er lurt å se på før en starter et prosjekt med disse programmene. I de underliggende linkene ligger oversikt for de spesifikke.

Ubuntu OS nylige utgivelser sine livsløp:

https://ubuntu.com/about/release-cycle

Ubuntu OS oversikt, samtlige utgivelser:

Releases - Ubuntu Wiki

Ubuntu MATE utgivelser (til Raspberry Pi):

Ubuntu MATE Releases - / (ubuntu-mate.org)

Python, nylige utgivelser sine livsløp:

Status of Python versions

ROS1, samtlige utgivelser:

Distributions - ROS Wiki

ROS2, samtlige utgivelser:

Distributions — ROS 2 Documentation: Rolling documentation

Vedlegg B – «Clean Install» av operativsystem på lab-PC

PC-ene ble utlevert fra IKT og er ikke underlagt skolens administrator slik at nødvendige operativsystem og programmer kan fritt lastes ned. Støtte PC ble brukt til å gjennomføre dette for å kunne installere Ubuntu 20.04 LTS på lab PC-ene. De tekniske spesifikasjonene PC-ene har kun for info, ikke noe krav utenom 25GB lagringsplass. Resten vil kun gå utover ytelsen.

	Lab PC 1 (Wiped)	Lab PC 2 (Wiped)	Støtte PC
Modell:	Lenovo Thinkpad E470	Lenovo Thinkpad E470	Surface Pro 4
Lagringsplass:	256,1GB	256,1GB	235GB
RAM:	8GB	8GB	8GB
Prosessor:	Intel® Core™ i5-7200U CPU @ 2.5GHz x 4	Intel® Core™ i7-7500U CPU @ 2.7GHz x 4	Intel® Core™ i5-6300U CPU @ 2.4GHz x 4
Grafikkort:	Mesa Intel® HD Graphics 620	Mesa Intel® HD Graphics 620	Intel® HD Graphics 520
Operativsystem:	Ubuntu 20.04 LTS	Ubuntu 20.04 LTS	Windows 10 Pro

Dersom en PC som skal brukes er allerede blitt «wiped»:

Hva som trengs:

- 1 minnepinne med minst 12GB

Hva som skal gjøres:

Følge denne fremgangsmetoden med valgt støtte PC: Install Ubuntu desktop | Ubuntu

Dersom eldre operativsystem er ønsket kan det finnes her: Releases - Ubuntu Wiki

Velg Desktop Image, ikke Server Install Image.

Dersom en PC som skal brukes ikke er allerede blitt «wiped»:

For å sette opp PC-en for «clean install» når den allerede har Linux går det an å bruke denne kommandoen i terminal-vinduet: **sudo shred -vfz -n 10 /dev/sda2**

Vedlegg C – Kloning SD-kort

Siden vi gjenbrukte plattformen til Hellesnes og Lyssand fra 2019 så valgte vi å klone SD-kortene deres slik at vi ikke risikerte å gjøre de korrupt.

Det som trengs:

- Støtte PC som ikke er under forsvarssektoren.

- SD-kortleser med minst 2 SD-kort innganger (eller 1 SD og 1 microSD), vi brukte Kingston FCR-H24

- Nye SD-kort

- SD-kort som ønskes å klone

Hva som skal gjøres:

1. Last ned BalenaEtcher:
   balenaEtcher - Flash OS images to SD cards & USB drives
2. Start programmet.
3. Sett inn SD-kortleser i støtte PC-en.
4. Sett inn ubrukt SD-kort og originalt SD-kort i kortleseren.
5. Følg programmet, men vær nøye med hvilken inngang du velger på *image* og *drive* før *flash* trykkes, slik at riktig SD-kort blir klonet.

Figur 0-1: Kingston FCR-H24

Vedlegg D – Båtspesifikasjoner og teknisk utstyr

I tabellen står informasjon om hvilke båter vi brukte, og hvilke versjoner delene har:

Båtnavn:	*Båt 0*	*Båt 1*	*Båt 2*	*Ekstra SD-kort*
Faktisk modellbåt:	B04	B01	B05	
IP-adresse:	10.124.224.49	10.124.224.33	10.124.224.58	10.124.224.34
OS på RaspberryPi:	18.04.3 LTS (Bionic)	18.04.3 LTS (Bionic)	18.04.3 LTS (Bionic)	
PX4-nummer:	B01 (står på konsollen)	B06	B01 (står på GPS-hodet)	
Firmware (PX4):	1.9.2	1.9.2	1.9.2	
ROS version:	Melodic	Melodic	Melodic	
Python release:	2.7.15	2.7.15	2.7.15	
Raspberry Pi modell:	3B+ (32-bit)	3B+ (32-bit)	3B+ (32-bit)	

Hvordan sjekke om en Raspberry Pi er 32-bit eller 64-bit:

- Skriv kommandoen «uname -m» i terminalvinduet
- Hvis det oppgis «armv7l» er det en 32-bit, men oppgis «aarch64» så er det en 64-bit.

Ruteren vi har brukt er en Zyxel WAH7601 Portable Router.

Figur 0-2: Bilde av Zyxel-ruter

Ved ny ruter så vil det bli nye IP-adresser som kommer til å bli brukt. Disse nye IP-adressene som blir brukt må skrives inn i koden.

Stien til dokumentene som må endres i filstrukturen er:
catkin_ws/src/swarm/Behaviour/ROS_operators/Boat_ID.py og
catkin_ws/src/swarm/Autopilot/ROS_operators/Boat_ID.py

Koden er vist under, og den grønne markerte boksen viser hvor IP-adressene skal endres.

```
#!/usr/bin/env python
'''
This class is used to set boat_id based on IP-adress

Questions: anhellesnes@fhs.mil.no
'''

import socket

def get_ID():
    '''Helper function to return ID to boat based on current IP'''

    IDs = {"192.168.136.61" : 0,
           "192.168.136.62" : 1,
           "192.168.136.63" : 2,
           "192.168.136.64" : 3,
           "192.168.136.65" : 4}

    IP = _get_ip()

```

Vedlegg E – HLK-LD2450 manual

Radarmanualen ligger vedlagt besvarelsen på WiseFlow.

Vedlegg F – Radartest 1 Arduinokode: rå output fra radar

serial_print.ino

```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // Serial ports for communications with the radar. RX=10, TX=11
int incomingByte = 0;            // For incoming bytes from the radar

void setup() {
  Serial.begin(250000);          // Open serial communications with PC and wait for port to open
  mySerial.begin(256000);        // Open serial communications with radar
}

void loop() {
  // reply only when you receive data:
  if (mySerial.available() > 0) {
    incomingByte = mySerial.read();     // read the incoming byte:
    Serial.println(incomingByte, HEX);  // print the incoming byte
  }
}
```

Vedlegg G – Radartest 2 Arduinokode: header isolert

serial_print_header.ino

```ino
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // Serial ports for communications with the radar. RX=10, TX=11
int incomingByte = 0;            // For incoming bytes from the radar
const int sequenceLength = 4;    // Length of the frame header
uint8_t targetSequence[] = {0xA9, 0xFF, 0x03, 0x00}; // The frame header
uint8_t receivedValues[sequenceLength];   // Array for checking the radar
int currentIndex = 0;   // Current index in the array

void setup() {
  Serial.begin(250000);     // Open serial communications with PC
  mySerial.begin(256000);   // Open serial communications with radar
}

void loop() {
  if (mySerial.available()) {
    // Read the byte recieved by the serial port
    uint8_t receivedValue = mySerial.read();

    // Check if recieved value matches the expected frame header
    if (receivedValue == targetSequence[currentIndex]) {
      receivedValues[currentIndex] = receivedValue;
      currentIndex++;

      // If the whole frame is recieved
      if (currentIndex == sequenceLength) {
        // Print the header
        Serial.print("Correct header recieved: ");
        for (int i = 0; i < sequenceLength; i++) {
          Serial.print(receivedValues[i], HEX);
          Serial.print(" ");
        }
        Serial.println();
        // Print the following bytes (26 bytes between the following header)
        for (int i = 0; i < 26; i++) {
            incomingByte = mySerial.read();
            Serial.print(incomingByte, HEX);
            Serial.print(" ");
        }
        Serial.print("\n\n");
        currentIndex = 0;   // Reset the index for next iteration
      }
    } else {
      // If recieved value doesnt match expected header, reset index and start over
      currentIndex = 0;
    }
  }
}

```

Vedlegg H – Radartest 3 Arduinokode: header, tre mål og tail

serial_print_experimental.ino

```ino
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11);                    // Serial ports for communications with the radar. RX=10, TX=11
const int sequenceLength = 4;                       // Length of the frame header
uint8_t targetSequence[] = {0xA9, 0xFF, 0x03, 0x00}; // The frame header
uint8_t receivedValues[sequenceLength];             // Array for checking the radar
int currentIndex = 0;                               // Current index in the array

const int targetLength = 8;         // Length of the target arrays
uint8_t firstTarget[targetLength];  // Array for first target
uint8_t secondTarget[targetLength]; // Array for second target
uint8_t thirdTarget[targetLength];  // Array for third target
int byteIndex = 0;                  // Current index in the target arrays
int incomingByte = 0;               // For incoming bytes from the radar

void setup() {
  Serial.begin(250000);   // Open serial communications with PC and wait for port to open
  mySerial.begin(256000); // Open serial communications with radar
}

void loop() {
  if (mySerial.available()) {
    // Read the byte recieved by the serial port
    uint8_t receivedValue = mySerial.read();

    // Check if recieved value matches the expected frame header
    if (receivedValue == targetSequence[currentIndex]) {
      receivedValues[currentIndex] = receivedValue;
      currentIndex++;

      // If the whole frame is recieved
      if (currentIndex == sequenceLength) {
        // Print the header
        Serial.println("Correct Header recieved:");
        for (int i = 0; i < sequenceLength; i++) {
          Serial.print(receivedValues[i], HEX);
          Serial.print(" ");
        }

        // Sort the first 8 bytes into the first array and print
        Serial.println("\nFirst target:");
        for (byteIndex; byteIndex < targetLength; byteIndex++) {
          incomingByte = mySerial.read();
          firstTarget[byteIndex] = incomingByte;
          Serial.print(firstTarget[byteIndex], HEX);
          Serial.print(" ");
        }
        byteIndex = 0;  // Reset the index for next iteration

        // Sort the second 8 bytes into the second array and print
        Serial.println("\nSecond target:");
        for (byteIndex; byteIndex < targetLength; byteIndex++) {
          incomingByte = mySerial.read();
          secondTarget[byteIndex] = incomingByte;
          Serial.print(secondTarget[byteIndex], HEX);
          Serial.print(" ");
        }
        byteIndex = 0;  // Reset the index for next iteration

        // Sort the third 8 bytes into the third array and print
        Serial.println("\nThird target:");
        for (byteIndex; byteIndex < targetLength; byteIndex++) {
          incomingByte = mySerial.read();
          thirdTarget[byteIndex] = incomingByte;
          Serial.print(thirdTarget[byteIndex], HEX);
          Serial.print(" ");
        }
        byteIndex = 0;  // Reset the index for next iteration

        // Print the two last remaining bytes into
        Serial.println("\nFrame tail:");
        for (int i = 0; i < 2; i++) {
          incomingByte = mySerial.read();
          Serial.print(incomingByte, HEX);
          Serial.print(" ");
        }
        currentIndex = 0; // Reset the index for next iteration
        Serial.print("\n\n");
      }
    }
    else {
      // Hvis mottatt verdi ikke matcher forventet verdi, nullstill
      currentIndex = 0;
    }
  }
}
```

Vedlegg I – Kode på Github

All kode utviklet i oppgaven er lagt ut på Github. Github er en online tjeneste som lar utviklere samarbeide på prosjekter sammen over nettet. Her kan kode publiseres slik at den er tilgjengelig for andre utviklere til å jobbe med.

All kode i denne oppgaven, samt data fra testing, ligger tilgjengelig for alle på linken:

MRABach/Swarm_final: The final version of the bachelor project. (github.com)

Dersom det er noen spørsmål – ta kontakt med forfatterne direkte.

Alt fra oppgaven til Andreas Handal Hellesnes og Kim Andrè Lyssand ligger også tilgjengelig på linken: AnKIbach/swarm: Bachelor based on making a maritime drone-swarm (github.com)

Vedlegg J – Tilleggsmoduler Python

Vi har tatt i bruk noen biblioteker som ikke er tilgjengelig som standard i Python og krever innstallering. Dette installeres ved hjelp av pip som er en pakkebehandler for python. Tabellen under viser pakkene.

Navn	Kommando	Link	Kommentar
PIP	Kommandoene star I linken	Installation - pip documentation v23.3.1 (pypa.io)	Guiden forklarer installasjon
Geopy	pip install geopy	Welcome to GeoPy's documentation! — GeoPy 2.4.1 documentation	Brukes til utregning av distanser og destinasjoner
Pyais	Pip install pyais	pyais · PyPI	Brukes til innhenting og dekoding av AIS-meldinger

Vedlegg K – AIS Klasse

```
from .GPS_class import GPS
from .Vector_class import Vector
import math as m
#from geopy import distance
import geopy.distance

class avoidance_vectors():
    #Deklarerer startverdier for klassen
    def __init__(self, other, other_speed, other_bearing):

        #self.own = GPS()
        self.own = (60.3947, 5.2675) #Denne brukes ettersom dronen programmet ikke er koblet opp mot dronen enda og har dermed ikke GPS posisjon
        self.other = other
        self.other_speed = other_speed
        self.other_bearing = other_bearing
        self.distance = geopy.distance.distance(self.own, self.other)
        self.placeholder = GPS() # Kommer errormelding hvis denne er borte men brukes ikke til noe
        self.avoidance_vecs = []

        #self.relative = self.calc_bearing(self.own,self.other)


    # Call Funksjonen som returnerer en liste med vektorer og distansene til punktene som vektorene regnes ut ifra
    def __call__(self, own, other, other_speed, other_bearing):

        aliases = self.create_aliases(other, other_speed, other_bearing)
        vec_tuple = self.make_avoid_vec(own,aliases)

        return vec_tuple


    #Regner ut relativ vinkel mellom to koordinater
    def calc_bearing(self, cordA, cordB):
        PointA = cordA
        PointB = cordB
        delta_lon =  PointB[1] - PointA[1]
        X = m.cos(PointB[0]*(m.pi/180))*m.sin(delta_lon*(m.pi/180))
        Y = m.cos(PointA[0]*(m.pi/180))*m.sin(PointB[0]*(m.pi/180))-m.sin(PointA[0]*(m.pi/180))*m.cos(PointB[0]*(m.pi/180))*m.cos(delta_lon*(m.pi/180))
        sigma = m.atan2(X,Y)
        bearing = m.degrees(sigma)
        return bearing
    # Regner ut distansen mellom to punkter
    def calc_distance(self, cord1, cord2):
        own = cord1
        other = cord2
        dist = geopy.distance.distance(own,other).km
        return dist

    # Funksjon som regner ut aliaser
```

```
48
49      # Funksjon som regner ut alliaser
50      def create_aliases(self, cord, speed, rel_angle):
51
52          intervals = [50, 300, 600]
53          lat = cord[0]
54          lon = cord[1]
55
56          speed_mps = (speed * 1.852)*(5/18) #Konverter fra knop til meter per sekund
57          distances = [i * speed_mps for i in intervals]
58          alias_list = []
59          alias_list.append(cord) #legger til original posisjon først i lista
60          #Looper gjennom distanse lista for å regne ut posisjonen den vil tilsvare i koordinatsystemet
61          #Sidekommentar: gatt over til m.radians her grunnet jeg trodde det var en måte på å fikse en feil, men feilen la et annet sted så nå tør jeg ikke endre der
62          for dist in distances:
63              a_d = (dist/1000) #/ 6371 #Angular distance,Hjår om distanse til km og så deler på joras radius i km
64              ---
65              Gamle koden vi brukte for utregning av ny posisjon. Denne ble erstattet av geopy biblioteket sin funksjon. Denne var en del verre på posisjon
66              lat2 = lat + m.asin(m.sin(m.radians(lat))*m.cos(a_d)+ m.cos(m.radians(lat))*m.sin(a_d)*m.cos(m.radians(rel_angle)))
67              lon2 = lon1 + m.atan2(m.sin(m.radians(rel_angle))*m.sin(a_d)*m.cos(m.radians(lat)),m.cos(a_d)-m.sin(m.radians(lat))*m.sin(m.radians(lat2)))
68              lat_deg = m.degrees(lat2)
69              lon_deg = m.degrees(lon2)
70              ---
71              #Geopy funksjon som regner ut punktet man ender opp etter cord antall kilometer
72              point = geopy.distance.distance(kilometers=a_d).destination(cord,rel_angle)
73              #al_pos = (lat2, lon2)
74              alias_list.append(point)
75          return alias_list


76
77      #Funksjon som lager unnvikelsesvektorer
78      def make_avoid_vec(self ,own_position, other_positions):
79          angle_list = []
80          vector_list = []
81          distance_list = []
82          #Funksjon som jobber gjennom alle aliassene og lager en vektor motsatt rettet av vinkelen til punktet
83          for positions in other_positions:
84              ang = self.calc_bearing(own_position,positions)
85              al_dist = geopy.distance.distance(own_position,positions).km
86              #For å passe på at vinkelen er i forhold til et kompass
87              if ang < 0:
88                  ang = 360 + ang
89              avoid_ang = ang + 180
90              if avoid_ang > 360:
91                  avoid_ang = avoid_ang - 360
92              else:
93                  avoid_ang = avoid_ang
94              angle_list.append(avoid_ang)
95              distance_list.append(al_dist)
96          for i in range(len(angle_list)):
97              vec = Vector(1/(i+1),angle_list[i]) # Lager vektorer der fremtidige posisjoner blir svakere vektet
98              vector_list.append(vec)
99          print(vector_list)
100         return_arg = (vector_list,distance_list)
101
102         return return_arg
103
```

Vedlegg L – Program for AIS unnvikelsesvektor

```
"""
Programm for utregning av unnvikelsesvektorer fra AIS-meldinger

Programmet er kodet for å ligge i mappen Saave. Dersom denne filen flyttes må importering av biblioteker endres
"""



# Må sette inn dette når i linux: #!/usr/bin/env python3
from pyais.stream import TCPConnection
from geopy import distance
from geopy import units
#from Behaviour.Behaviours.Classes.GPS_class import GPS
import math as m
from Behaviour.Behaviours.Classes.Vector_class import Vector
from Behaviour.Behaviours.Classes.AIS_class import avoidance_vectors

# IP og port for TCP-strømmen som Kystverket strømmer på
host = "153.44.253.27"
port = 5631

#Simulerert egen posisjon, senere må kan byttes med GPS() dersom man får integrert den i droneplatformen skikkelig
mylat = 60.3947
mylon = 5.2675
own_pos = (mylat,mylon)

buffer = 0.05


# Regner ut avstanden mellom 2 punkter. Brukes til å finne avstand til andre fartøyer
def calc_distance(cord1, cord2):
    own = cord1
    other = cord2
    dist = distance.distance(own,other).km
    return dist

# Åpner filen og sletter gammelt innhold
file_clear = open(r"Vecs_from_ais","w")
file_clear.truncate()
file_clear.close
# Åpner filen vektorene skrives til igjen
file = open(r"Vecs_from_ais","a")
#Lytter på alle TCP meldingene som ligger ute
for msg in TCPConnection(host, port):
    # print(test)
    boat = msg.decode() #Dekoder AIS meldingen
    ais_content = boat
    ais_dict = ais_content.asdict() # Gjør den dekodede meldingen om til en dict så den kan brukes i koden
    tot_vector = Vector(0,0)
    #print(ais_dict)
    # Filter for å få bort AIS meldinger som ikke er relevante for navigering av dronene
    if ais_dict["msg_type"] == 1 or ais_dict["msg_type"] == 2 or ais_dict["msg_type"] == 3 or ais_dict["msg_type"] == 18 or ais_dict["msg_type"] == 19:

        #Filter for å få bort fartøy langt unna
        if ((ais_dict["lon"] < mylon + buffer) and  (ais_dict["lon"] > mylon - buffer)) and  ((ais_dict["lat"] < mylat + buffer) and  (ais_dict["lat"] > mylat - buffer)):
```

```
55                #print(ais_dict)
56                other_pos = (ais_dict["lat"], ais_dict["lon"])
57                dist = calc_distance(own_pos, other_pos)
58
59                #Lager en instans av avoidance_vectors klassen og kaller __call__ funksjonen dens
60                instsans = avoidance_vectors(other_pos, ais_dict["speed"], ais_dict["course"])
61                instans_vectors = instsans(own_pos,other_pos,ais_dict["speed"],ais_dict["course"])
62                #Henter ut de to listene som returneres
63                get_vec_list = instans_vectors[0]
64                get_distances = instans_vectors[1]
65                counter1 = 0
66                #print("*"*80)
67                for vec in get_vec_list:
68                    r = get_distances[counter1]
69                    print(r)
70
71                    weight = (3/(m.pow(r,2)+0.25))
72                    temp_vec = vec * weight
73                    tot_vector += temp_vec
74                    counter1 += 1
75                """
76                Utskrifter for å se hvordan vektoren ser ut
77                print("*"*80)
78                print(tot_vector.magnitude)
79                print(tot_vector.angle)
80                """
81
82                #Lager en string ut av vektoren for så å skrive den til en fil
83                vec_magn_string = str(tot_vector.magnitude)
84                vec_ang_string = str(tot_vector.angle)
85                space = " "
86                tot_vector_string = vec_magn_string + space + vec_ang_string
87                file.write(tot_vector_string + '\n')
88    #Lukker filen
89    file.close
```

Vedlegg M –µXRCE-DDS kontra µRtps

µXRCE-DDS forklart på PX4-Autopilot sin egen hjemmeside:

uXRCE-DDS (PX4-ROS 2/DDS Bridge) | PX4 User Guide (main)

Video fra PX4 Developer Summit 2019 som forklarer µXRCE-DDS, µRtps og sammenligner de:

ROS2 Powered PX4 - PX4 Developer Summit 2019 - YouTube

PowerPoint-presentasjonen som ble vist:

Apresentação do PowerPoint (px4.io)

Diskusjon på PX4-Autopilot sin Github-side om hvordan tilpasse et system fra µRtps til µXRCE-DDS:

Adaptation from microRtps to XRCE-DDS · Issue #21035 · PX4/PX4-Autopilot · GitHub

Vedlegg N - Feilsøking etter første praktiske test

Ved hjelp av feilmeldingene som kom under testen ble feilene fort identifiserte. Den første feilen var at kompilatoren ikke kjente igjen noen av tegnene i koden. Kompilatoren bruker ACII til tolkningen av tegn. Denne standarden inneholder blant annet ikke de norske bokstavene: æ, ø og å. I kommentarer i koden ble det skrevet på norsk med noen av disse bokstavene. Dette ble endret på ved å fjerne bokstavene, men i tillegg legges det til en linje på toppen av koden for å bestemme at standarden UTF-8 skal brukes i stedet. UTF-8 inneholder alle de samme tegnene som ASCII og mer. En ulempe ved å bruke UTF-8 i stedet for ASCII kan være inkompatibilitet med legacy systemer. Dette er en større risiko i systemer som benytter seg av databaser. Basert på dette ser vi det som trygt nok å velge

UTF-8.

Den andre feilen vi kom frem til var at bibliotekene vi tok i bruk ikke var installert på systemet. Python har en pakke håndterer kalt pip. Denne brukes for å installere nødvendige biblioteker og gjør de tilgjengelig for prosjektene. Ved hjelp av denne installerte vi geopy, men fikk ikke gjort det samme for pyais. Etter en feilsøking fant vi ut at dronene brukte python2. Pyais og geopys nyeste versjoner krever python3.7 eller nye. Grunnen til at geopy fungerte selv om den nyeste versjonen krever python3.7 er sannsynligvis at den hentet da geopy versjon 1.x som fortsatt var tilgjengelig. Ettersom pyais ikke var nødvendig for denne testen siden kontakte bare blir simmulert og det ikke hentes inn eller oversettes noen melding så ble det besluttet å fjerne dette fra koden og forske uten.

Vedlegg O - Testresultater for vektorregning

1	Kordinat kontakt	Egen posisjon	Distanse verktøy(km)	Distanse kode(km)	Feil distanse	Vinkel verktøy	Vektor lengde, verktøy	Vektor lengde, kode	Total lengde verktøy		1,00692		
2	(60.39884, 5.31204)	(60.3947,5.2675)	2,490300	2,498625	-0,33 %	259,34	0,465001	0,462027	Total vinkel verktøy		-100,816		
3	(60.39886, 5.31201)	(60.3947,5.2675)	2,489000	2,497380	-0,34 %	259,29	0,232734	0,231235	Total lengde kode		0,856459		
4	(60.39895, 5.31188)	(60.3947,5.2675)	2,483800	2,492466	-0,35 %	259,03	0,154223	0,153194	Total vinkel kode		-100,8215		
5	(60.39906, 5.31172)	(60.3947,5.2675)	2,477600	2,486388	-0,35 %	258,71	0,154966	0,153915					
6									Feil lengde		4,388 %		
7									Feil vinkel		-0,005 %		
8	TEST 2												
9	Kordinat kontakt	Egen posisjon	Distanse verktøy(km)	Distanse kode(km)	Feil distanse	Vinkel verktøy	Vektor lengde, verktøy	Vektor lengde, kode	Total lengde verktøy		3,94414		
10	(60.39461, 5.28720)	(60.3947,5.2675)	1,0827	1,0864	-0,34 %	270,53	2,109350	2,097513	Total vinkel verktøy		-108,664		
11	(60.39609, 5.28468)	(60.3947,5.2675)	0,9562	0,960122	-0,41 %	260,93	1,288307	1,280045	Total lengde kode		3,93027		
12	(60.40203, 5.27460)	(60.3947,5.2675)	0,9047	0,906501	-0,20 %	205,57	0,926548	0,923728	Total vinkel kode		-108,6806		
13	(60.40946, 5.26200)	(60.3947,5.2675)	1,6696	1,672557	-0,18 %	169,58	0,246908	0,246108					
14									Feil lengde		0,353 %		
15									Feil vinkel		-0,208 %		
16													

Vedlegg P - Utregninger for matematisk vektortest

```
for boid in boats:
    if boid['distance'] < self.perception and boid['distance'] != 0.0: #finds number of boids within perception

        dx = boid['speed'] * m.sin(m.radians(boid['bearing'])) #converts vector to x,y components
        dy = boid['speed'] * m.cos(m.radians(boid['bearing']))
        average_temp.set(dx,dy)

        average_vector += average_temp
        total += 1.0

if total > 0.0:
    try:
        alignment.magnitude = average_vector.magnitude / total
    except ValueError:
        pass

    try:
        alignment.angle = average_vector.angle / total
    except ValueError:
        pass

    alignment_tot = m.sqrt(m.pow(average_vector.magnitude, 2.0) + m.pow(average_vector.angle, 2.0))

    if alignment_tot > self.maxForce:
        alignment = (alignment.__truediv__(alignment_tot)) * self.maxForce

return alignment
```

Utklipp fra koden som inneholder formlene som brukes til utregningen der self.max_force er 1.2 (Hellesnes & Lyssand, 2019)

Først regnes det ut hva average_vektor blir totalt.

Dronene går som nevnt i 3 knop med en course på 45°. For hver av dronene blir da utregningen slik:

$$dx = 3 * \sin(45°) = 2.12132$$

$$dy = 3 * \cos(45°) = 2.12132$$

Vektoren for en drone ser da slik ut: (2.12132, 2.12132). Når denne legges sammen for alle fartøyene (3) blir average_vektor: (6.36396 , 6.36396).

Alignement_tot blir da : $\sqrt{6.36396^2 + 6.36396^2} = 9$

Dette overgår self.max_force som fører til en normalisering:

$$alignement = \frac{(6.36396, 6.36396)}{9} * 1.2 = (0.84852, 0.848528)$$

Da har alignement gjort sin utregning, men BOID behaviour har også en vekting på 1.5 på denne vektoren så det endelige resultatet blir da

$$Styringsvektor = (0.84852, 0.848528) * 1.5 = (1.27279, 1.27279)$$

Vedlegg Q – Oppstart og sekvensiell sjekkliste for test

OPPSTART AV BÅT FOR TEST

HVA	**Kommando**	**Annet**
LOGG INN		**brukernavn** b05 **passord** Bachelor
HENT NYESTE KODE:	git pull origin master	
OPPDATER ARBIEDSROMMET MED NYE FILER	catkin_make	innenfor mappen for arbeidsrommet
SOURCE SETUP FILEN	source /devel/setup.bash	gjøres automatisk i .bashrc
LUKK BÅTEN OG GJØR KLAR		
START SYSTEMET	roslaunch <pkg> <launchfil>	For dette systemet: roslaunch swarm system.launch
START ROSBAG	rosbag record -a	Tar opp data på topics på hele ROS
STOPP SYSTEMET	ctrl + c	Stopper hele systemet - kan gjøres fra basen
LASTE NED FILER FRA GITHUB		
HVA	**Kommando**	**Annet**
GÅ INN I ARBEIDSROMMET	cd <arbeidsrom>/src/	For dette er arbeidsrommet: catkin_ws
KLON FILENE FRA GITHUB	git clone https://github.com/AnKiBach/swarm.git	**Brukernavn** AnKiBach **Passord** 1Blirstor
OPPDATER ARBEIDSROMMET	catkin_make	