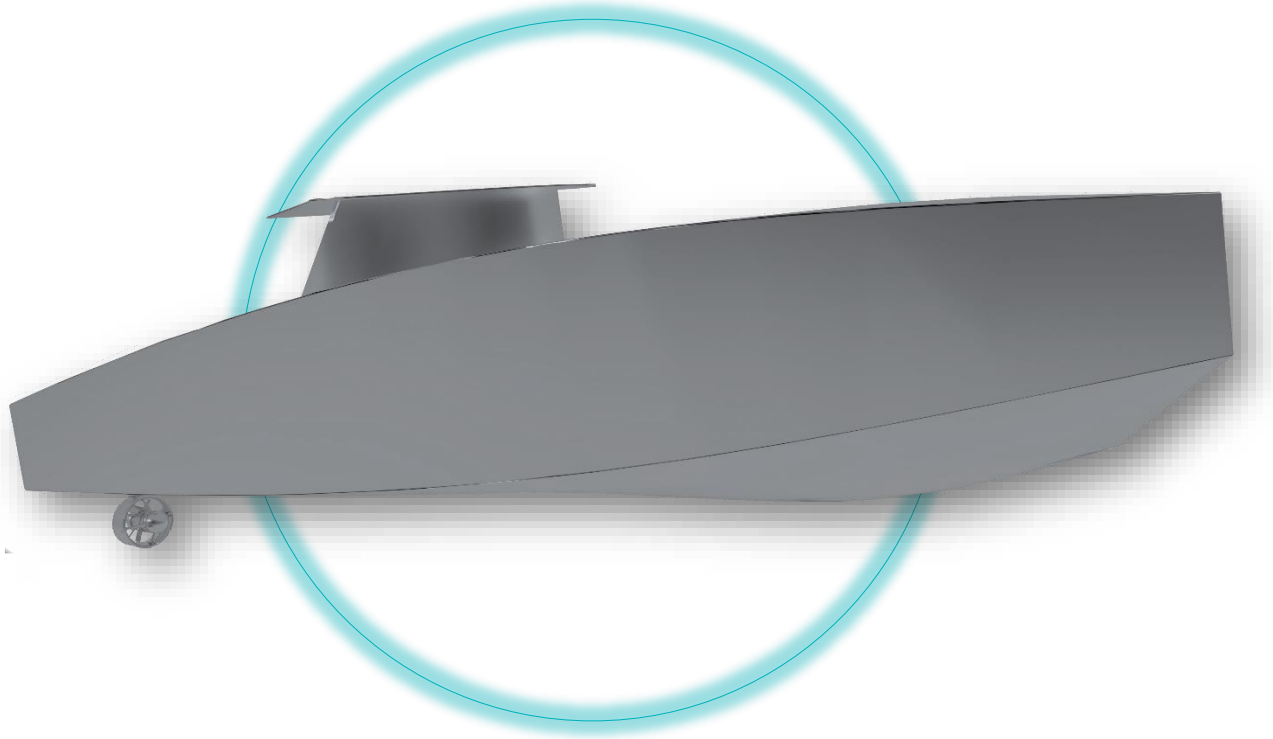
Høgskulen på Vestlandet

BACHELOROPPGAVE

BO22EH-02 AUTONOM SJØDRONE



Linn Jeanette Myhren **Kandidatnummer: 471**

Magnus Eide **Kandidatnummer: 419**

1. mars 2022

Forord

Denne interne bacheloroppgaven ble vi informert om på vårsemesteret 2021. Med informasjonen om oppgaven fikk vi også tilbudet om å reise ned til Horten for å se på Autodrone konkurransen 2021. Dette var for å bli inspirert og vurdere om dette var noe vi ville skrive bacheloroppgave om. En av gruppemedlemmene tok tilbudet og på høstsemesteret 2021 ble gruppen dannet, og vi valgte oppgaven.

Bacheloroppgaven ble gjennomført på vårsemesteret 2022, som slutten på en treårig bachelorgrad innen automatisering med robotikk, Y-vei. For å få best mulig forståelse av rapporten er det en fordel om man har forkunnskaper innenfor instrumentering, robotikk og programmering.

Under oppgaven har vi møtt mange utfordringer og lært mye nytt som er nyttig å kunne, da autonome fartøy blir mer og mer relevant med tiden.

Vi ønsker å takke alle som har bidratt og gitt oss støtte gjennom bachelorprosjektet, og spesielt vil vi takke:

- **HVL (Høgskulen på Vestlandet)** for at vi fikk muligheten til å velge en så spennende og utfordrende bacheloroppgave.
- Veilederne våre **Olav Skjølingstad** og **Torkel Bjarte Larsson** for god veiledning og oppfølgning gjennom bachelorprosjektet.
- PhD-kandidatene **Gizem Ates** og **Laurenz Elstner** fra HVL i Førde for god hjelp med oppgaven.
- Foreleser **Harald Spångberg** for god hjelp med diverse spørsmål knyttet til bacheloremnet.
- **USN (Universitetet i Sørøst-Norge**) som utførte forarbeidet på sjødronen.
- Foreleser **Marius Stian Tannem** fra USN for god kommunikasjon og villighet til å svare på spørsmål angående sjødronen og konkurransen.
- Foreleser **Runald Walter Meyer** fra maskin som har bistått med sjødronens ballast og forankring av bøyene. Dette kan leses mer om i delkapittel 6.1.
- **Styret for Smedasundet Amfi** som lot oss låne bryggen deres til sjøsetting og testing av sjødronen.
- **MARKOM2020** *[1]* som har finansiert sjødronen.
- **Medstudentene våre** for gode diskusjoner og avbrekk.

Sammendrag

Denne bachelorrapporten tar for seg oppgaven å gjøre en fjernstyrt sjødrone om til en autonom sjødrone, som skal representere HVL for første gang i Autodrone konkurransen 2022. Dette innebærer at sjødronen må lære seg å behandle sensordataene sine og ta fornuftige avgjørelser basert på dem.

Skroget til sjødronen og algoritmer for fjernstyring er laget av USN, i tillegg til at de har valgt ut sjødronens komponenter. På grunn av dette måtte gruppen sette seg inn i hvordan komponentene kunne brukes og hvordan USN sine algoritmer fungerte.

Gruppen har også satt seg inn i oppdragene i Autodrone konkurransen og gjort sjødronen klar til å utføre Speed Gate oppdraget. Speed Gate er et hastighetsoppdrag hvor sjødronen skal navigere mellom en rød og en grønn bøye, rundt en gul bøye og tilbake.

For å gjøre sjødronen klar til konkurransen, ble objekt detekterings algoritmen YOLO (**Y**ou **O**nly **L**ook **O**nce) lagt til. Gruppen har blitt kjent med hvordan YOLO kan brukes og trent algoritmen opp til å gjenkjenne bøyene som benyttes i konkurransen. Dette, i tillegg til at det måtte lages et eget datasett for treningen, ble en større del av oppgaven enn forventet. Under installasjonen og treningen av algoritmen møtte gruppen en rekke utfordringer som måtte løses.

Gruppen har skrevet egne algoritmer til sjødronen på programmeringsspråket Python. Disse algoritmene har hvert sitt formål, blant annet å finne ut hvilket oppdrag som skal utføres, lokalisere objektene sjødronen ser, sette veipunkter og navigere sjødronen. Disse algoritmene er utviklet for å passe inn med egenskapene til sjødronen og bruker matematiske utregninger til å utføre oppgavene sine.

Det er også utført en rekke tester, for å teste funksjonaliteten til sjødronen og algoritmene som ble laget. Blant annet ble sjødronen sjøsatt for å teste flyteevne og manøvrerbarhet, og Speed Gate oppdraget ble testet på land.

I starten av prosjektet hadde gruppen ambisjoner om å få til flere av konkurranseoppdragene. Dette ble ikke tilfellet da utfordringene som ble møtt, tok lang tid å løse. Av den grunn fokuserer denne rapporten bare på å gjøre sjødronen klar til å utføre Speed Gate oppdraget. Gruppen skal fortsatt delta i Autodrone konkurransen, så det er planlagt å fortsette å jobbe med sjødronen, etter innlevering av bachelorrapporten.

Summary

This bachelor thesis addresses the task of turning a remote-controlled sea drone into an autonomous sea drone, which will represent HVL for the first time in the Autodrone competition 2022. This means that the sea drone must learn to process its sensor data and make reasonable decisions based on them.

The hull of the sea drone and algorithms for remote control are made by USN, in addition to that they have selected the components for the sea drone. Because of this, the group had to understand how the components could be used and how USN's algorithms worked.

The group has also familiarized themselves with the missions in the Autodrone competition and made the sea drone ready to perform the Speed Gate mission. Speed Gate is a speed mission where the sea drone must navigate between a red and a green buoy, around a yellow buoy and back.

To make the sea drone ready for the competition, the object detection algorithm YOLO (**Y**ou **O**nly **L**ook **O**nce) was added. The group has become familiar with how YOLO can be used and trained the algorithm to recognize the buoys used in the competition. This, and that a custom data set had to be created for the training, became a larger part of the project than expected. During the installation and training of the algorithm, the group met several challenges that had to be solved.

The group has written their own algorithms for the sea drone in the programming language Python. These algorithms each have their own purpose, which involves finding out which mission to perform, locating the objects seen by the sea drone, setting waypoints and navigating the sea drone. These algorithms are designed to fit within the properties of the sea drone and use mathematical calculations to perform their tasks.

There have also been several tests performed, to test the functionality of the sea drone and the algorithms that were created. Among other things, the sea drone was launched to test buoyancy and manoeuvrability, and the Speed Gate mission was tested on land.

At the beginning of the project, the group had ambitions to complete several missions in the competition. This did not occur as the challenges faced took a while to solve. For this reason, this report only focuses on preparing the sea drone to perform the Speed Gate mission. The group are going to participate in the Autodrone competition, and therefore have planned to continue working on the sea drone, after submitting their bachelor thesis.

Innhold

Ordliste

BPS	Bilde Per Sekund
EKF	Extended Kalman Filter
GPS	Global Positioning System
IMU	Inertial Measurement Unit
HVL	Høgskulen på Vestlandet
KOLT	Known Object Localization and Tracking
Led	et spesifisert vannområdet for skipstrafikk
LiDAR	Light Detection And Ranging
MARKOM2020	Maritim Kompetanse mot 2020
NTNU	Norges Teknisk-Naturvitenskapelige Universitet
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
SSD	Single Shot Detection
UIT	Universitetet i Tromsø
USN	Universitetet i Sørøst-Norge
Veipunkt	Et punkt på veien til målet *[2]*
YOLO	You Only Look Once, en objekt detekterings algoritme

Figurliste

Tabelliste

Formelliste

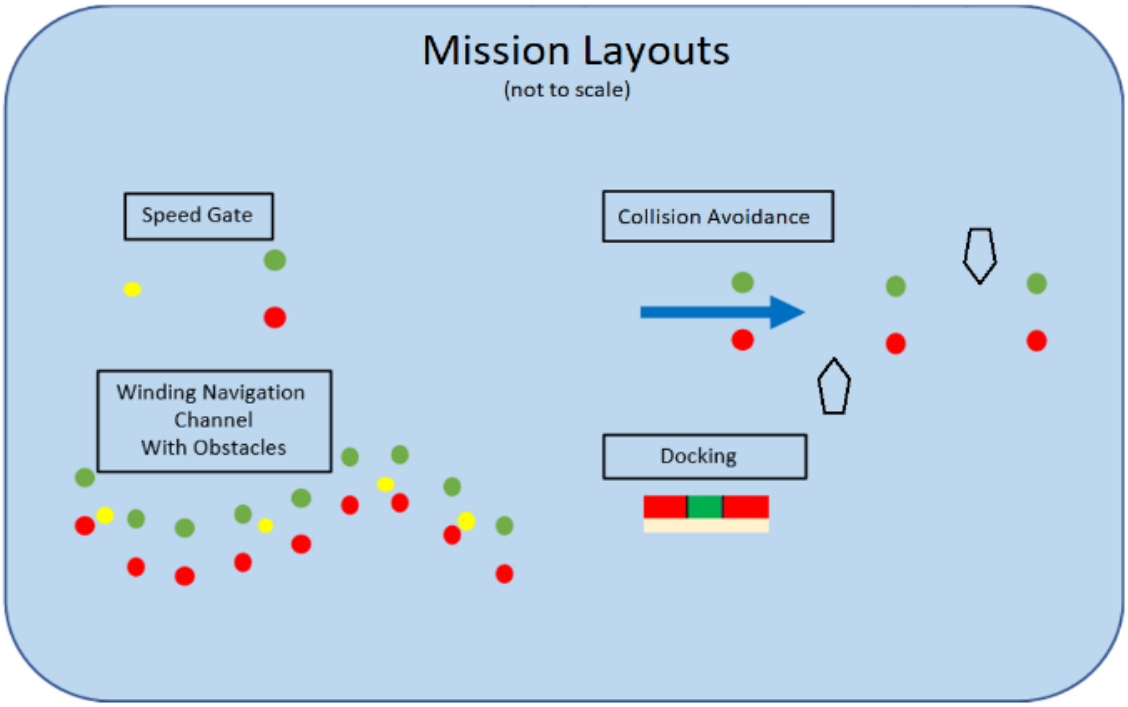
1 Innledning

Havet har alltid vært viktig for Norge. Det gir folket mat, energi, arbeidsplasser og mye mer. I 2010 ble det nasjonale prosjektet MARKOM2020 *[1]* etablert for å styrke kvaliteten og strukturen i norsk maritim profesjonsutdanning frem mot 2020. På grunn av økt interesse for automatisering av maritime operasjoner ble konkurransen Autodrone *[3]* etablert.

Gruppen skal med denne sjødronen delta i Autodrone 2022 og representere Høgskulen på Vestlandet (HVL) for første gang i konkurransen.

2 Autodrone 2022

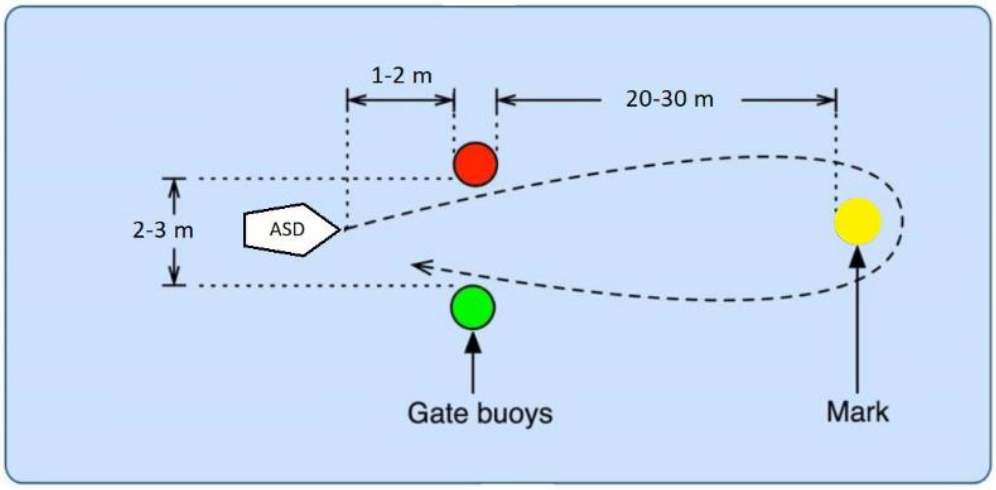
Autodrone er en årlig nasjonal konkurranse mellom utvalgte universiteter og høgskoler i Norge. Disse utvalgte universitetene og høgskolene er for øyeblikket HVL, Universitetet i Sørøst-Norge (USN), Norges Teknisk-Naturvitenskaplige Universitet (NTNU) og Universitetet i Tromsø (UiT). I 2020 ble det utarbeidet regler og oppdrag for konkurransen og i 2021 ble konkurransen holdt for første gang, hvor USN, UiT og NTNU deltok. Hovedmålet med konkurransen er å styrke autonom teknisk kompetanse innen maritim høyere utdanning i Norge. Konkurransen består av oppdragene vist i *Figur 1* som blir nøyere forklart i kommende delkapitler:



Figur 1 Konkurranseoppdrag *[3]*

2.1 Speed Gate

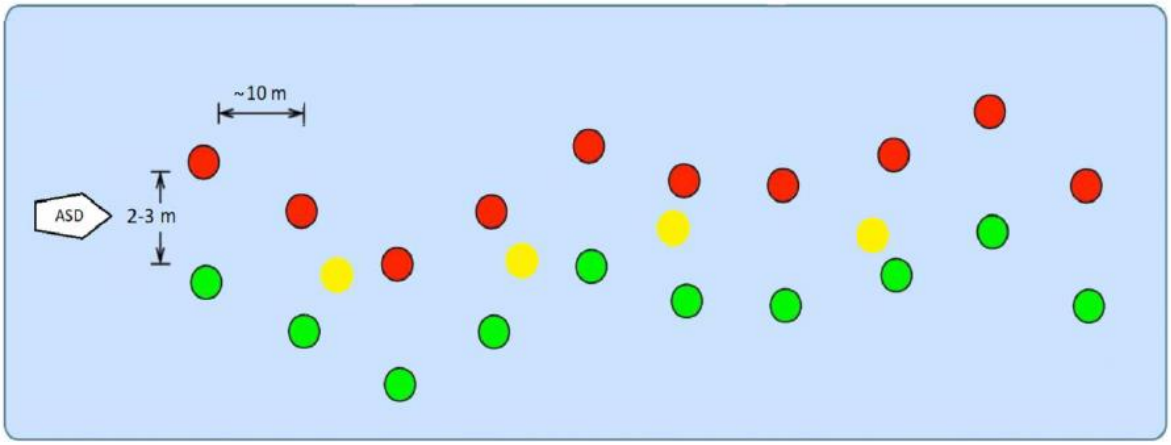
Speed Gate er et hastighetsoppdrag som går ut på at sjødronen skal kjøre mellom en rød og grønn bøye, rundt en gul bøye og tilbake som vist i *Figur 2*. Oppdraget skal gjennomføres så raskt som mulig, da flest poeng gis til den raskeste sjødronen.



Figur 2 Speed Gate *[3]*

2.2 Led med hindringer

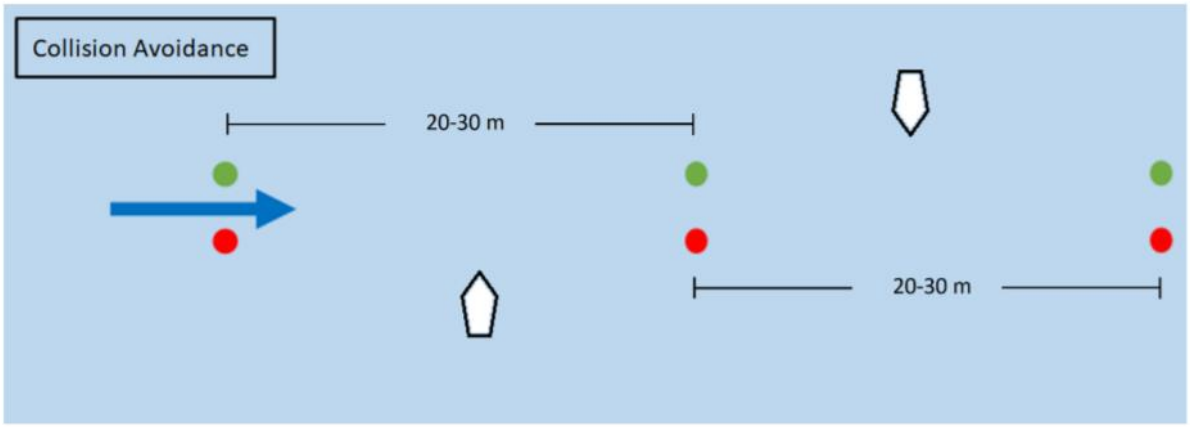
Led med hindringer går ut på at sjødronen skal manøvrere seg gjennom en led som består av røde og grønne bøyer, som vist i *Figur 3*. I tillegg vil det være gule bøyer plassert i leden som skal være til hinder, og som sjødronen må unngå å kollidere med.



Figur 3 Led med hindringer *[3]*

2.3 Kollisjonsunngåelse

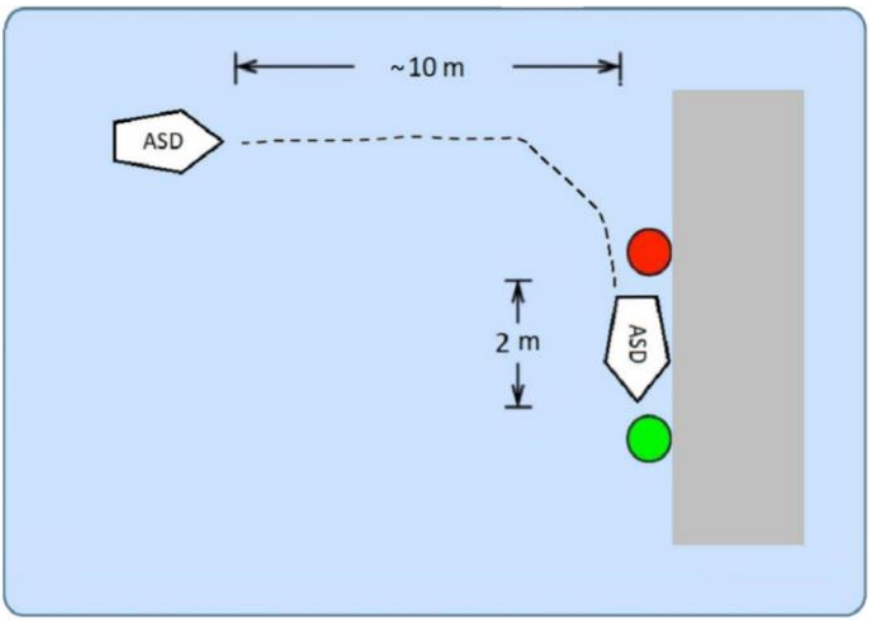
Dette oppdraget går ut på at sjødronen skal komme seg gjennom en led som består av røde og grønne bøyer, uten å kollidere med tilfeldig kryssende trafikk. Dette er illustrert i *Figur 4*. Under oppdraget skal sjødronen ha en fart på under 4 knop, og komme seg gjennom leden på under 10 minutter.



Figur 4 Unngå kollisjon med kryssende trafikk *[3]*

2.4 Dokking

Under dokking oppdraget skal sjødronen lokalisere det definerte dokkeområdet og utføre selve dokkingen. Det definerte dokkeområdet vil bestå av en grønn og rød bøye med cirka 2 meter mellomrom, som illustrert i *Figur 5*. Sjødronen skal dokke slik at den står med en side inntil bryggen og kan bruke fremdriftssystemet sitt for å holde seg på plass, om nødvendig.



Figur 5 Dokking *[3]*

2.5 Krav

For at sjødronen skal kunne delta i Autodrone må den tilfredsstille diverse krav.

Disse kravene *[3]* er:

- **Autonomi:** Sjødronen må være helt autonom og ta autonomibeslutningene ombord på sjødronen.
- **Kommunikasjon:** Sjødronen skal ikke kunne sende eller motta kommandoer fra operatørene når den er i autonom-modus.
- **Plasserbar:** Sjødronen skal kunne plasseres manuelt.
- **Energikilde:** Sjødronen må være batteridrevet. Batteriene skal være forseglet for å redusere faren for lekkasje av syre eller kaustiske elektrolytter. Den åpne kretsspenningen til ethvert batteri (eller batterisystem) skal ikke overstige 60 V DC.
- **Nødstoppbryter:** Sjødronen må minst ha én rød nødstopp plassert på seg. Når denne blir aktivert, må den øyeblikkelig koble fra all strøm til motorene og aktuatorene.
- **Trådløs nødstopp:** Sjødronen må ha en trådløs nødstopp som i en nødsituasjon gjør det mulig for operatørene til å koble fra all strøm til motorene og aktuatorene fra land.
- **Fremdrift:** Ethvert fremdriftssystem kan brukes, eksempelvis trustere eller årer.
- **Fjernstyrbarhet:** Sjødronen skal være fjernstyrt for operatørene på land.
- **Sikkerhet:** Alle skarpe, spisse og bevegelige deler må dekkes til og merkes.
- **Slepbar:** Sjødronen må kunne slepes.
- **Visuell tilbakemelding:** Det er pålagt å implementere et visuelt tilbakemeldingssystem som indikerer statusen til sjødronen.
- **Vekt:** Hele sjødronen skal veie mindre enn 70 kg.
- **Nyttelast:** Sjødronen må ha et sted å montere et kamera med en uhindret utsikt til forsiden av sjødronen.

3 Teknisk introduksjon til sjødronens komponenter og programvarer

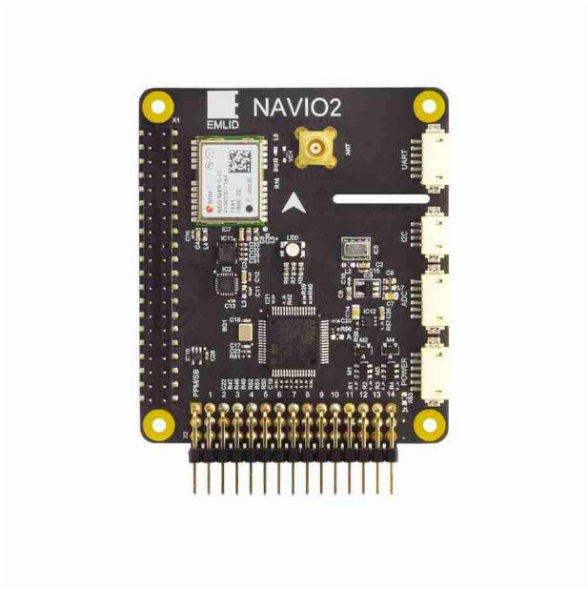
3.1 Raspberry Pi

Raspberry Pi er en mikrokontroller som i denne oppgaven brukes som hovedenhet for Navio2 HAT, som ikke ville fungert uten en Raspberry Pi. Sjødronen er utstyrt med en Raspberry Pi 3 B+, som er avbildet på *Figur 6*.

Figur 6 Raspberry Pi *[4]*

3.2 Navio2 HAT

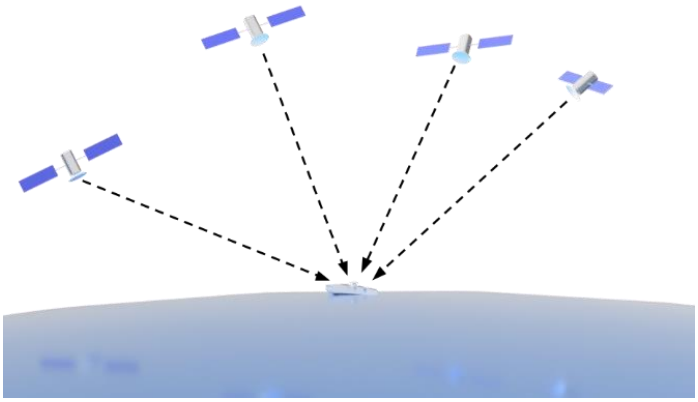
Navio2 HAT (Hardware Attached on Top) som er avbildet på *Figur 7*, er et tilleggskort som monteres oppå Raspberry Pi-kortet. Dette kortet kan brukes til å hente inn data fra blant annet GPS-en (Global Positioning System), radiokontrolleren og den innebygde IMU-en (Inertial Measurement Unit). Dette tilleggskortet kan også brukes til å styre aktuatorene.



Figur 7 Navio2 HAT *[5]*

3.3 GPS (Global Positioning System)

En GPS bruker satellittsignaler for å finne posisjonen sin på jordens overflate. Dersom antennen har klar sikt til minst fire satellitter, som skissert på *Figur 8*, kan man lese posisjonen sin med nøyaktighet ned mot 3 meter.



Figur 8 GPS sin virkemåte *(privat)*

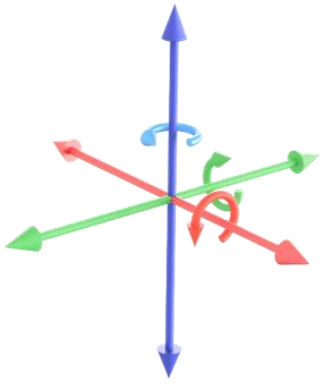
3.4 IMU (Inertial Measurement Unit) *[6]*

IMU er en samlebetegnelse for akselerometer, gyroskop og magnetometer. IMU-er kan brukes til å måle hastighet, akselerasjon, vinkelhastighet, vinkelakselerasjon og magnetiske felt.

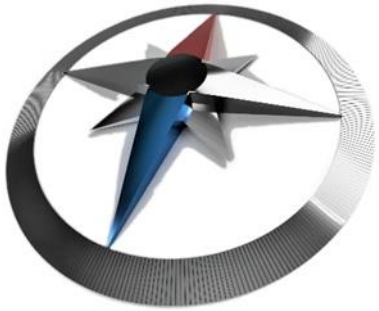
Akselerometeret brukes til å måle lineær hastighet og akselerasjon, som kan brukes til å regne ut posisjon. Gyroskopet brukes for å måle vinkelhastighet og vinkelakselerasjon, som kan brukes til å beregne orientering. Disse målingene er illustrert i *Figur 9*.

Magnetometeret brukes for å måle hvilken retning fartøyet er orientert i forhold til kloden, ved hjelp av magnetfelt, akkurat som et kompass, illustrert med *Figur 10*.

Sjødronen er utstyrt med to IMU-er, hvor både stereokameraet og tilleggskortet Navio2 HAT, har en innebygd IMU.



Figur 9 Illustrasjon av mulige målinger med akselerometer og gyroskop *(privat)*



Figur 10 Kompass/magnetometer *(privat)*

3.5 Stereokamera

Sjødronen har et stereokamera av typen ZED2, som er avbildet på *Figur 11*. Et stereokamera er et kamera som ser verden på en lignende måte som mennesker. Det har to linser som er forskjøvet sideveis, noe som gjør det mulig å beregne avstand. Som nevnt i forrige delkapittel, har dette stereokameraet en innebygd IMU, som kan brukes til å måle sjødronens bevegelser. I tillegg har det et barometer for å måle atmosfæretrykket og en synsvinkel på 120° horisontalt.

Figur 11 Stereokamera *[7]*

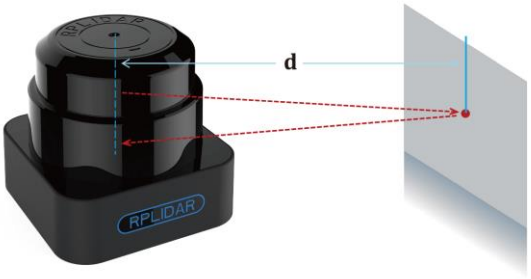
3.6 NVIDIA Jetson *[8]*

Sjødronen har en NVIDIA Jetson Xavier NX mikrokontroller, som er avbildet på *Figur 12*. NVIDIA Jetson er en serie av avanserte mikrokontrollere, som er designet for å utføre krevende regneintensive oppgaver, som for eksempel maskinlæring og kunstig intelligens. Den er dermed ypperlig for objekt detektering, som den hovedsakelig brukes til i denne oppgaven.

Figur 12 NVIDIA Jetson *[9]*

3.7 LiDAR (Light Detection And Ranging)

En LiDAR bruker lysbølger til å måle avstanden til omgivelsene rundt seg. Sjødronen er utstyrt med en Slamtec RPLidar S1, som har en dokumentert rekkevidde på 10 meter for mørke objekter som reflekterer lite lys. Denne LiDARen roterer kontinuerlig, som gjør at den kan måle avstander 360° rundt seg selv. *Figur 13* illustrerer utseende og virkemåten til LiDARen.



Figur 13 RPLidar S1 beregner avstanden d ved hjelp av hvor lang tid lyset bruker frem og tilbake *[10]*

3.8 Servomotorer

En servomotor er en type elektrisk motor, som er nyttig når man skal styre retningen til trusterne, for å manøvrere sjødronen. Sjødronen er utstyrt med to servomotorer av typen BLS-12V7146 fra JX, som er avbildet på *Figur 14*. Disse kan roteres 180°, altså 90° til hver side.

Figur 14 Servomotor *[11]*

3.9 Trustere

På hver av de to servomotorene er det montert på trustere, som danner fremdriftssystemet til sjødronen. Trusterne er fra BlueRobotic og er avbildet på *Figur 15*. Disse er vanntette og populære i forbindelse med mindre fartøyer som opererer i vann.

Figur 15 Truster *[12]*

3.10 Radiokontroller

Radiokontrolleren er av typen FRSKY Tarantis X9D, som er avbildet på *Figur 16*. Radiokontrolleren benyttes for fjernstyringen av sjødronen i denne oppgaven.

Figur 16 Radiokontroller *[13]*

3.11 ROS (Robot Operating System) *[14] [15] [16]*

ROS er et operativsystem for roboter som bidrar til å gjøre utviklingen av nye roboter enklere. Det er et åpent-kildekode-prosjekt, som inkluderer verktøy og biblioteker for å skaffe, bygge, skrive og kjøre kode på tvers av flere datamaskiner og komponenter.

I denne oppgaven bruker sjødronen ROS for å danne kommunikasjon mellom komponentene og algoritmene. De fleste av komponentene på sjødronen er ROS-vennlige, som vil si at man kan bruke ROS-kommandoer oppgitt i manualene nevnt nedenfor, for å hente inn sensordata.

Navio2 ROS-manual: https://docs.emlid.com/navio2/ros/ *[14]*

Stereokamera ZED2 ROS-manual: https://www.stereolabs.com/docs/ros/ *[17]*

LiDAR ROS-manual: https://github.com/slamtec/rplidar_ros *[18]*

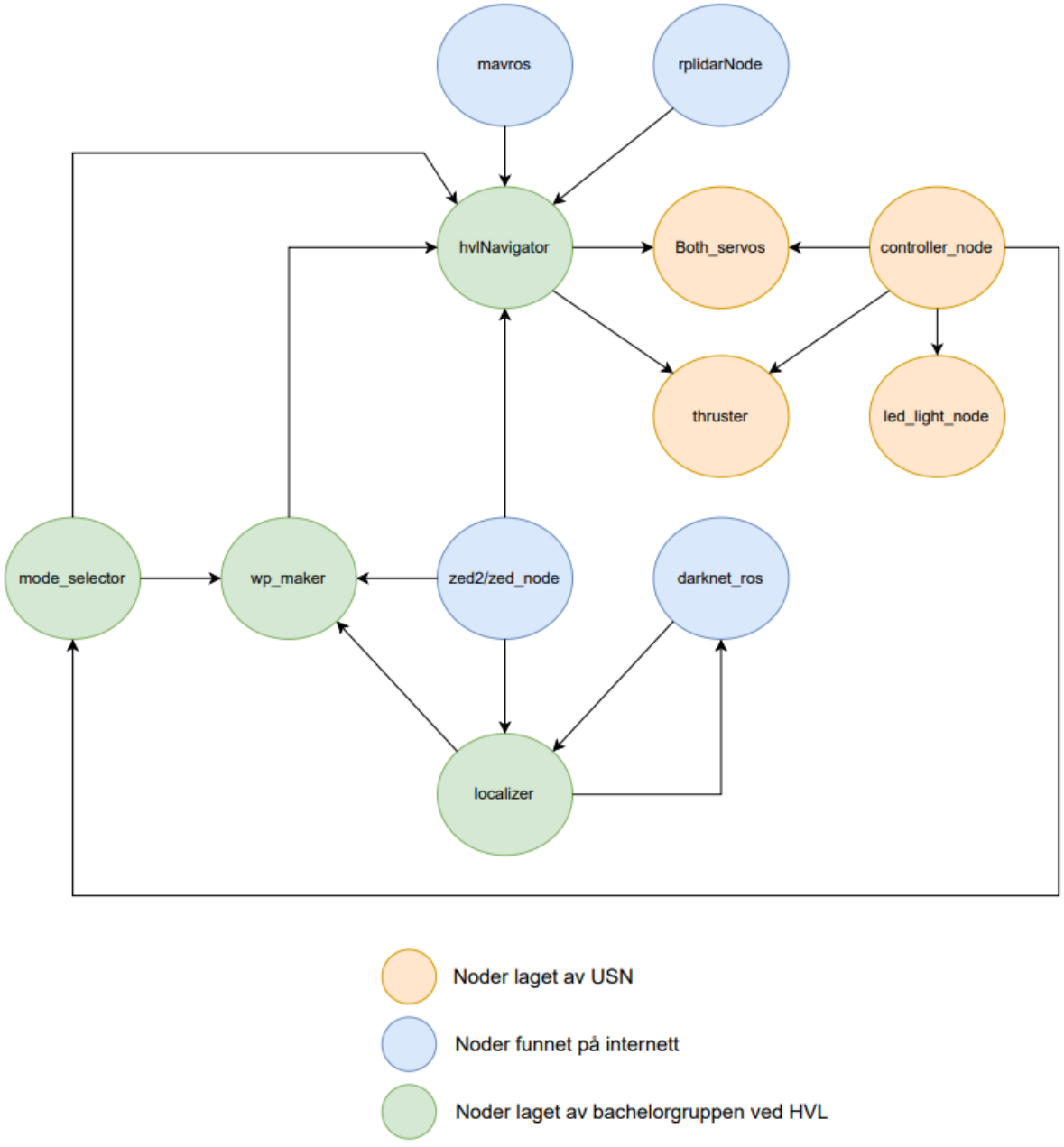
ROS består hovedsakelig av blokkene og kommandoene som blir forklart i påfølgende delkapitler. Disse blir nyttig å kunne til kapittel 5 hvor gruppens algoritmer gjennomgås.

3.11.1 Node

En node representerer hvert enkelt program og det anbefales å lage en node for hvert formål. Gruppen har laget nodene:

- `localizer` for lokaliseringsalgoritmen
- `wp_maker` for algoritmen som setter veipunktene
- `mode_selector` for algoritmen som finner hvilken modus sjødronen er i
- `hvlNavigator` for navigasjonsalgoritmen

Noder bruker topics for å kommunisere sammen, hvor topics kan leses om i delkapittel 3.11.5. Nedenfor på *Figur 17* er det en oversikt over alle nodene sjødronen bruker og hvilke av disse nodene som kommuniserer sammen og deler data med hverandre.

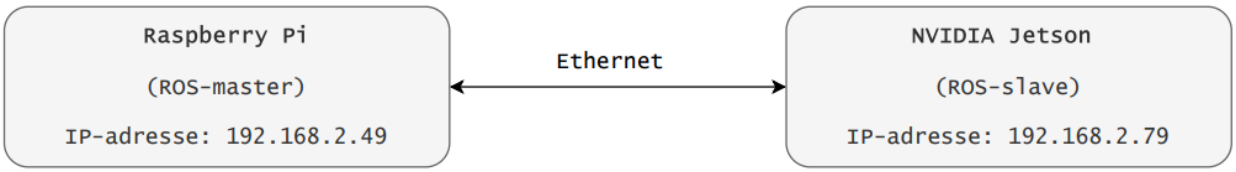


Figur 17 Hvilke noder som brukes og kommuniserer sammen på sjødronen *(privat)*

3.11.2 Master

En master er hoved-noden i et ROS system, som gjør det mulig for de andre nodene å kommunisere med hverandre. Masteren må kjøre hele tiden når man bruker ROS.

På sjødronen er det lagt opp til master-slave kommunikasjon, hvor Raspberry Pi-kortet fungerer som master og NVIDIA Jetson-kortet fungerer som slave. *Figur 18* illustrerer dette.



Figur 18 ROS Master-Slave *(privat)*

3.11.3 Publisher

En publisher publiserer data til et spesifisert topic, med struktur definert i en message. Les mer om topic og message i delkapittel 3.11.5 og 3.11.6.

3.11.4 Subscriber

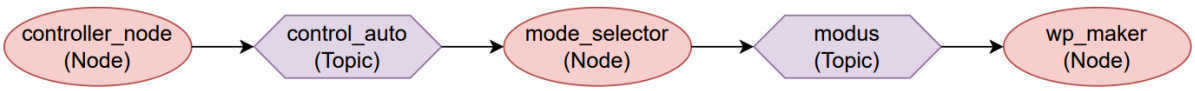
En subscriber henter inn data fra et spesifisert topic til en node, og lar noden bruke verdiene topicet inneholder.

3.11.5 Topic

Et topic er som emnet i en samtale og brukes for å kommunisere mellom nodene. Topic-kommunikasjonen er en asynkron enveis kommunikasjon, som baserer seg på publishere og subscribere. Denne typen kommunikasjon er nyttig for å overføre visse data, siden topicet kontinuerlig mottar messages og brukes ofte for sensorer som periodisk publiserer data. Forskjellige topics kan bruke samme message, men må da ha forskjellig navn.

Nedenfor på *Figur 19* er en grafisk fremstilling over hvordan topics brukes.

- Noden `controller_node` publiserer til topicet `control_auto`.
- Noden `mode_selector` subscriber på topicet `control_auto` og publiserer på topicet `modus`.
- Noden `wp_maker` subscriber på topicet `modus`.



Figur 19 Grafisk fremstilling over hvordan topics brukes *(privat)*

3.11.6 Message

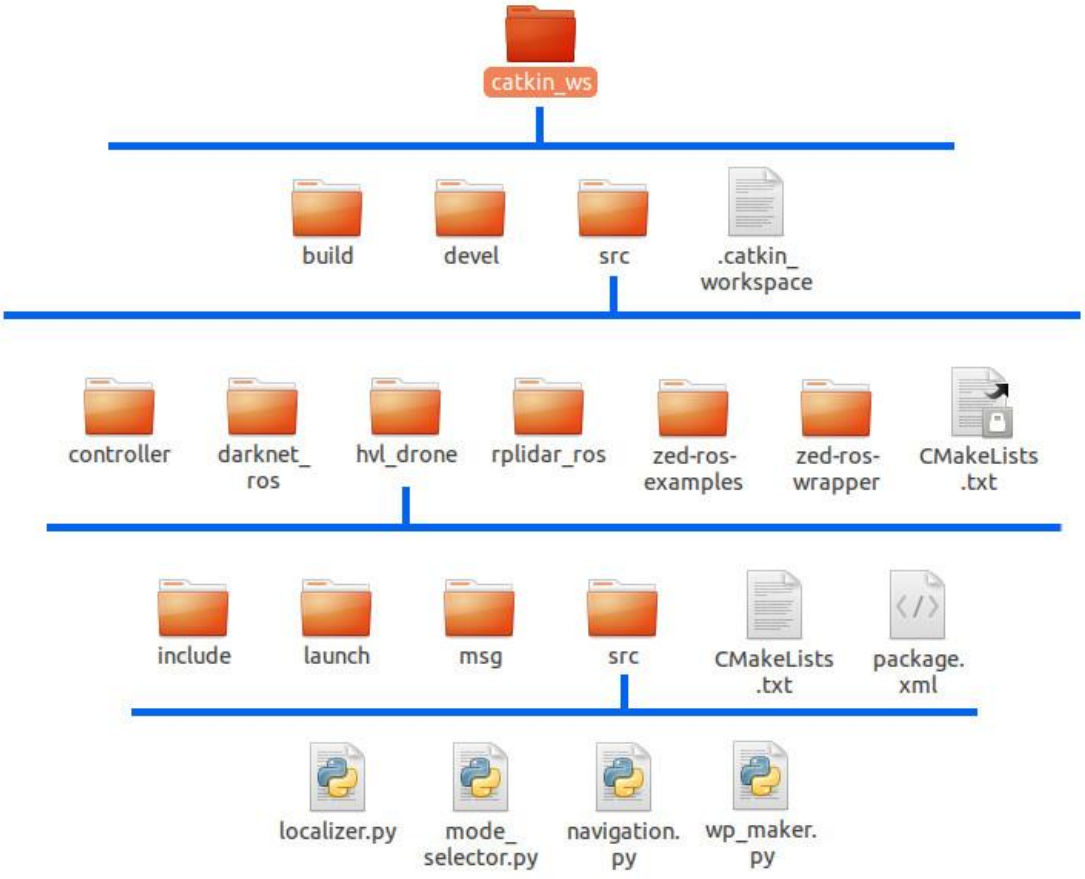
En message er informasjonen om strukturen i et topic og består av navngitte variabler. På *Figur 20* vises et skjermbilde av en av message som gruppen laget til sjødronen. For å lage en message, måtte gruppen følge den spesifikke prosedyren fra hjemmesiden til ROS *[19]*.

Figur 20 ObjectPosition.msg *(privat)*

De egendefinerte messagene laget av bachelorgruppen ved HVL er vedlagt som **Vedlegg 1 - Vedlegg 3** og messagene laget av USN er vedlagt som **Vedlegg 4 – Vedlegg 8**.

3.11.7 Catkin

Catkin er byggesystemet til ROS og er der alle de ROS-relaterte mappene ligger. Se *Figur 21* for sjødronens Catkin og hva den inneholder.



Figur 21 Sjødronens mappestruktur i ROS *(privat)*

3.11.8 Package

En package er en mappe, som ligger i mappen Catkin/src, som inneholder flere noder og konfigurasjonsfiler utviklet for et bestemt formål. På *Figur 21* vises mappestrukturen til sjødronen, der man ser hvor gruppens package (hvl_drone) ligger. Dette er packagen som inneholder alle algoritmene og messagene gruppen har laget til bacheloroppgaven.

3.11.9 rosrun

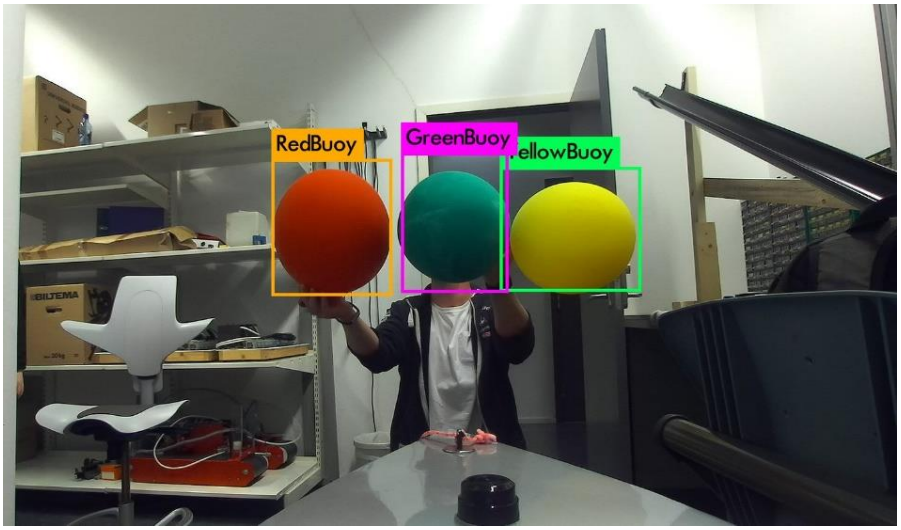
rosrun er en grunnleggende ROS-kommando som brukes for å kjøre en enkelt node. Dette kan være et python script (.py) eller et C++ script (.cpp).

3.11.10 roslaunch

roslaunch er en grunnleggende ROS-kommando som brukes for å kjøre flere noder. For å kunne bruke kommandoen må man ha en launch-fil (.launch) der man definerer hvilke noder som skal kjøres.

3.12 Objekt detektering

For at sjødronen skal kunne gjenkjenne og lokalisere bøyene i konkurransen, må den først vite hva en bøye er og hvordan en bøye ser ut. For å løse dette ble det tatt i bruk en algoritme for objekt detektering, hvor *Figur 22* viser resultatet etter at algoritmen var trent opp til å gjenkjenne bøyene som benyttes i konkurransen.



Figur 22 Objekt detekterings algoritmen etter trening *(privat)*

3.12.1 Hva er objekt detektering? *[20] [21]*

Objekt detektering er en avansert form for bildebehandling og datasyn, der et nevralt nettverk benyttes. Det nevrale nettverket oppdager hvilke objekter som befinner seg i et digitalt, todimensjonalt bilde og lokaliserer de med avgrensede bokser, som *Figur 22* viser. Objekt detektering er et stort tema innenfor datasyn (Computer Vision), fordi det er med på å gjøre maskiner og roboter mer menneskeligvennlige og menneskelignende.

Sjødronens oppgaver muliggjøres med objekt detektering, da sjødronen kan gjenkjenne og lokalisere objektene rundt seg og dermed samhandle bedre med omgivelsene sine.

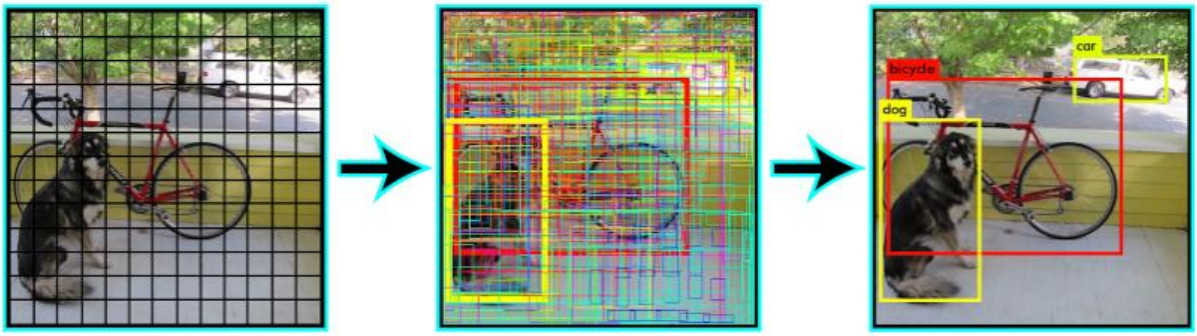
Det finnes mange algoritmer for objekt detektering, hvor de mest populære i dag er YOLO (**Y**ou **O**nly **L**ook **O**nce) og SSD (**S**ingle **S**hot **D**etector). De forskjellige algoritmene fungerer på tilnærmet lik måte, men gruppen valgte å ta i bruk YOLO, da YOLO algoritmen hadde flere fordeler og var best egnet for sjødronens bruk.

3.12.2 Hva er YOLO? *[20] [21]*

YOLO er en forkortelse for **Y**ou **O**nly **L**ook **O**nce og er en algoritme for sanntid objekt detektering. Den er svært rask og nøyaktig, noe som har gjort den veldig populær. Algoritmen har stor anvendbarhet med mange brukstilfeller, spesielt for autonome fartøy og intelligent videoanalyse, nettopp på grunn av hastigheten.

3.12.3 Hvordan fungerer YOLO? *[20] [21]*

YOLO algoritmen fungerer ved at bildet deles inn i N antall rutenett, hvor alle rutenettene er like store. Deretter er hvert rutenett ansvarlig for å detektere og lokalisere objektet de inneholder. Måten de gjør dette på er ved å sette bokser rundt objektene som rutenettet inneholder, sammen med hvilken type objekt det er og hvor sannsynlig det er at rutenettet inneholder objektet. Nedenfor på *Figur 23* er en grafisk fremstilling over hvordan YOLO behandler et bilde.



Figur 23 YOLO sin virkemåte *[20]*

3.12.4 Fordeler med YOLO *[20] [21]*

En av de største fordelene med YOLO algoritmen, er at det er åpen kildekode. Dette innebærer at koden er tilgjengelig for alle og at alle har mulighet til å gjøre endringer slik de selv ønsker. Dette har gjort det mulig å anvende algoritmen i dette prosjektet, ettersom noen har laget YOLO algoritmen ROS-vennlig.

Det er også mye informasjon tilgjengelig om YOLO, som var en stor fordel ettersom ingen i gruppen hadde brukt objekt detektering tidligere. Det finnes også mye informasjon på hvordan man kan trene opp algoritmen til å detektere det man vil og mener er relevant.

I tillegg er YOLO algoritmen veldig rask, som gjør det mulig å detektere objekter nærmest i sanntid. Dette er noe den tidligere nevnte SSD algoritmen fungere dårlig til, da den har et høyere fokus på nøyaktighet enn hastighet.

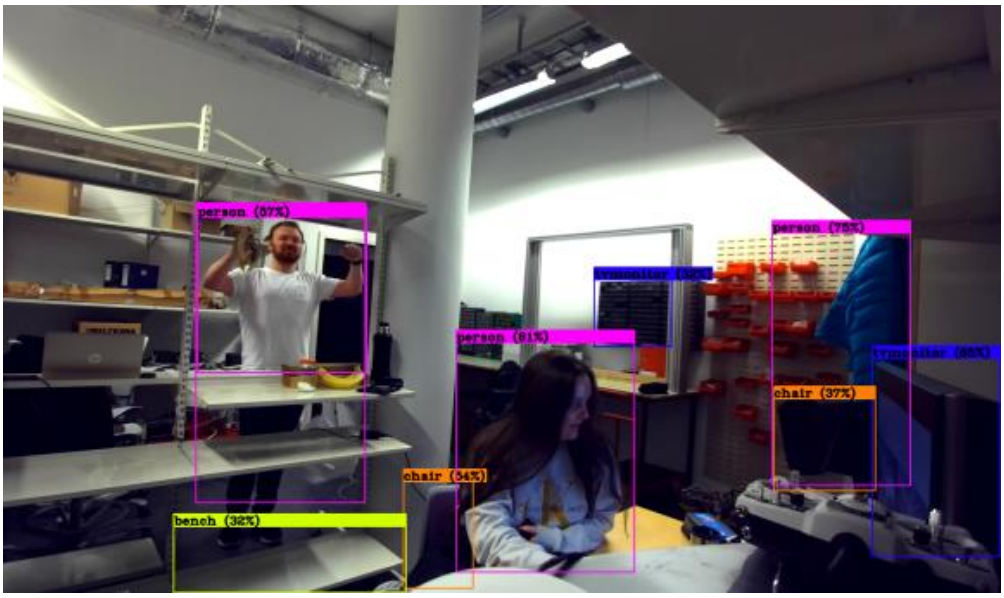
3.12.5 Begrensninger med YOLO *[20] [21]*

Selv om YOLO ser ut til å være den mest populære algoritmen for objekt detektering, så har den flere begrensninger. For eksempel kan YOLO slite med å oppdage små objekter, dersom det er flere på et lite område i bildet. Små objekter som kommer i grupper, for eksempel en linje med maur, vil derfor kunne være vanskelig for algoritmen å oppdage, eller skille fra hverandre som individuelle maur.

I tillegg vil YOLO være mindre nøyaktig sammenlignet med de mye tregere objekt detekterings algoritmene som ikke detekterer i sanntid.

3.12.6 Implementering av YOLO

Det finnes i dag fem versjoner av YOLO algoritmen, som har små forbedringer for hver generasjon. Som nevnt tidligere, finnes det en ROS-vennlig YOLO versjon, som er en YOLOv3 implementasjon for ROS. Denne versjonen ble installert ved å følge prosedyren som ligger ute på GitHub-siden *[22].* Når algoritmen var installert, klarte sjødronen å detektere 80 forskjellige typer objekter, med en hastighet på 8-11 BPS (**B**ilder **P**er **S**ekund). Se *Figur 24* som viser resultatet fra YOLO algoritmen etter installasjon.

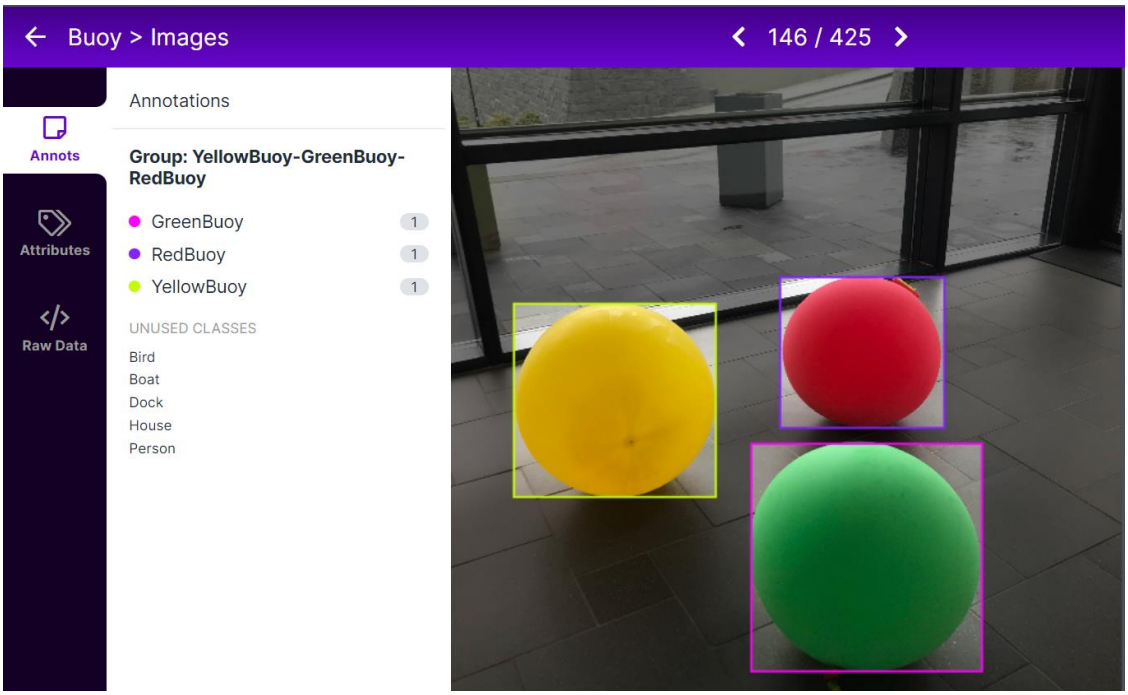


Figur 24 YOLO etter installasjon *(privat)*

3.12.7 Trening av YOLO

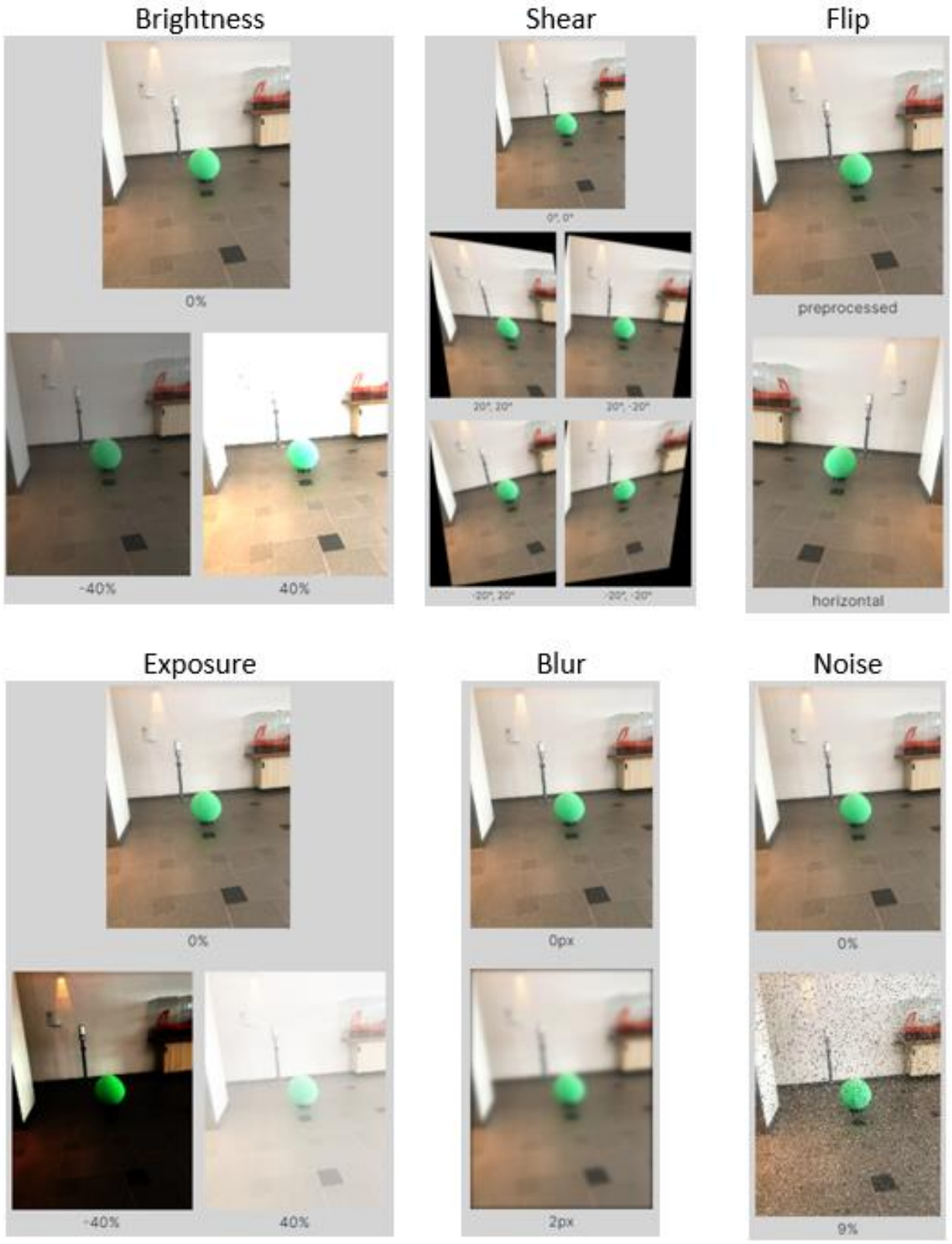
Selv om YOLO kunne detektere 80 forskjellige typer objekter etter installasjonen, klarte den ikke å detekterte bøyene som benyttes i Autodrone konkurransen. Algoritmen måtte dermed trenes opp til å detektere disse bøyene.

For å kunne trene YOLO algoritmen trengtes det et datasett av bøyene. Dette datasettet ble laget på roboflow *[23]*, som er en nettside for å lage datasett for objekt detekterings algoritmer. Det ble tatt mange bilder av bøyene som deretter ble lastet opp til nettsiden. Når bildene lå inne på nettsiden, ble det manuelt satt bokser rundt alle objektene sjødronen skulle kunne detektere. Nedenfor på *Figur 25* vises et skjermbilde når det ble satt bokser rundt objektene på roboflow *[23]*.



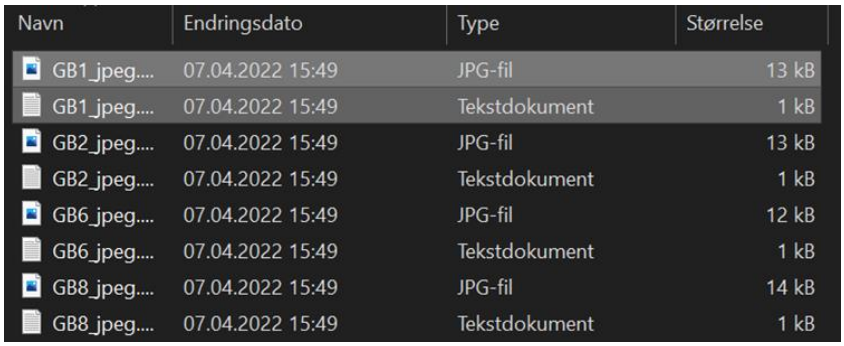
Figur 25 Markering av bøyene på roboflow *(privat)*

På roboflow *[23]* er det også en rekke funksjoner, som redigerer kopier av bildene, for å simulere objektene i forskjellige miljøer. Se *Figur 26* for hvilke funksjoner som ble brukt.



Figur 26 Funksjoner i roboflow *(privat)*

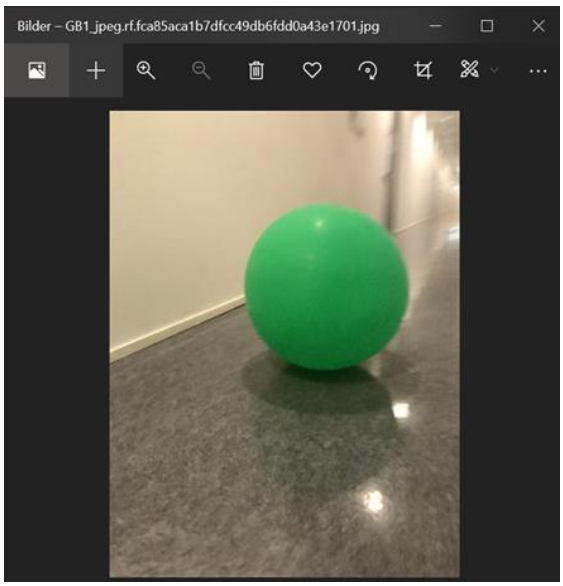
Når datasettet var ferdig, bestod datasettet av bildefiler og tekstfiler som *Figur 27* viser.



Navn	Endringsdato	Type	Størrelse
GB1_jpeg....	07.04.2022 15:49	JPG-fil	13 kB
GB1_jpeg....	07.04.2022 15:49	Tekstdokument	1 kB
GB2_jpeg....	07.04.2022 15:49	JPG-fil	13 kB
GB2_jpeg....	07.04.2022 15:49	Tekstdokument	1 kB
GB6_jpeg....	07.04.2022 15:49	JPG-fil	12 kB
GB6_jpeg....	07.04.2022 15:49	Tekstdokument	1 kB
GB8_jpeg....	07.04.2022 15:49	JPG-fil	14 kB
GB8_jpeg....	07.04.2022 15:49	Tekstdokument	1 kB

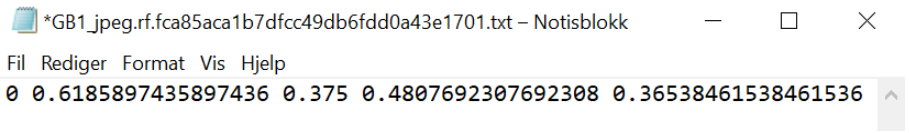
Figur 27 Datasettet fra roboflow *(privat)*

Hvor bildefilene er bildene som ble lagt inn, se *Figur 28*.



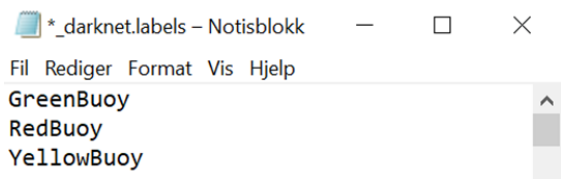
Figur 28 Bildet som ble lagt inn i roboflow *(privat)*

Mens tekstfilene inneholder hvilke type objekter som befinner seg i bildet, og verdiene til boksene som ble satt rundt i roboflow *[23]*, se *Figur 29*.



Figur 29 Tekstfil som generes for bildet *(privat)*

Det generes også en liste over alle objektene som kan detekteres, hvor det første 0-tallet i *Figur 29* refererer til GreenBuoy i *Figur 30*.



Figur 30 Liste over alle objektene *(privat)*

Når datasettet er ferdig laget, kan selve treningen av YOLO algoritmen starte. For å trene YOLO algoritmen ble stegene i en YouTube-video fra The AI Guy *[24]* fulgt. I denne videoen går han gjennom hvordan man kan få trent YOLO algoritmen i Google Colab.

I forbindelse med bacheloroppgaven har YOLO algoritmen blitt trent opp tre ganger.

På **den første treningen** besto datasettet kun av bøyer, som gjorde at algoritmen bare detekterte bøyer. Dette medførte at algoritmen ble veldig unøyaktig og klassifiserte det meste kun etter farge. For eksempel ble hoder og ansikt klassifisert som gule og/eller røde bøyer, siden algoritmen ikke var trent opp til å detektere personer. Dette var forventet, da den første treningen var ment som en test for å se om videoen gruppen fulgte fungerte. Det ble også observert at hastigheten til algoritmen hadde sunket ned til 1-2 BPS. Dette ble undersøkt og kom av at algoritmen var trent som YOLOv3.

Til **den andre treningen** ble datasettet utvidet med enda flere bilder av bøyene. I tillegg ble det lagt til bilder av hus, brygger, båter, fugler og personer som ble funnet på Google. Dette ble lagt til da dette er objekter sjødronen burde kunne detektere og skille mellom. Båter må sjødronen kunne detektere hvis den skal kunne utføre Autodrone oppdraget «Kollisjonsunngåelse». Under den andre treningen ble algoritmen trent som YOLOv3-tiny for å øke hastigheten, hvor YOLOv3-tiny er en raskere versjon av YOLOv3. Etter treningen kom hastigheten opp til 8-11 BPS, men var unøyaktig og slet med å detektere de andre objektene som var lagt til. Grunnen til dette var at det var lagt inn for få bilder av disse objektene.

Til **den tredje treningen** ble datasettet utvidet med enda flere bilder av bøyene. I tillegg ble det også hentet ferdige datasett av fugler, båter, hus og personer fra Google Open Image Dataset *[25]*, ved å følge stegene i en annen YouTube-video fra The AI Guy *[26]*. Etter dette bestod datasettet av totalt 4700 bilder, fordelt som *Tabell 1* viser.

Klasser	Antall bilder
Bøyer	700
Båter	1000
Personer	1000
Fugler	1000
Hus	1000

Tabell 1 Datasettets innhold

Under den tredje treningen ble algoritmen igjen trent opp som YOLOv3-tiny. Etter treningen hadde algoritmen samme hastighet som tidligere, 8-11 BPS, og en mye bedre nøyaktighet.

4 Forarbeid utført av USN

USN har laget selve skroget til sjødronen, valgt ut og montert på komponentene som blir brukt. I tillegg gjorde de sjødronen fjernstyrt som innebærer at radiokontrolleren kan brukes til å styre servomotorene og trusterne. Når sjødronen er fjernstyrt bruker den ikke sensordataene sine til noe, da dette er gruppens oppgave, i forbindelse med å gjøre den autonom. Med sjødronen fikk gruppen også en startprosedyre, som ligger vedlagt som **Vedlegg 9 – Autodrone_Startbeskrivelse_HVL**. Denne startprosedyren ga gruppen en god start med å bli kjent med oppsettet til sjødronen.

4.1 Sjødronens virkemåte

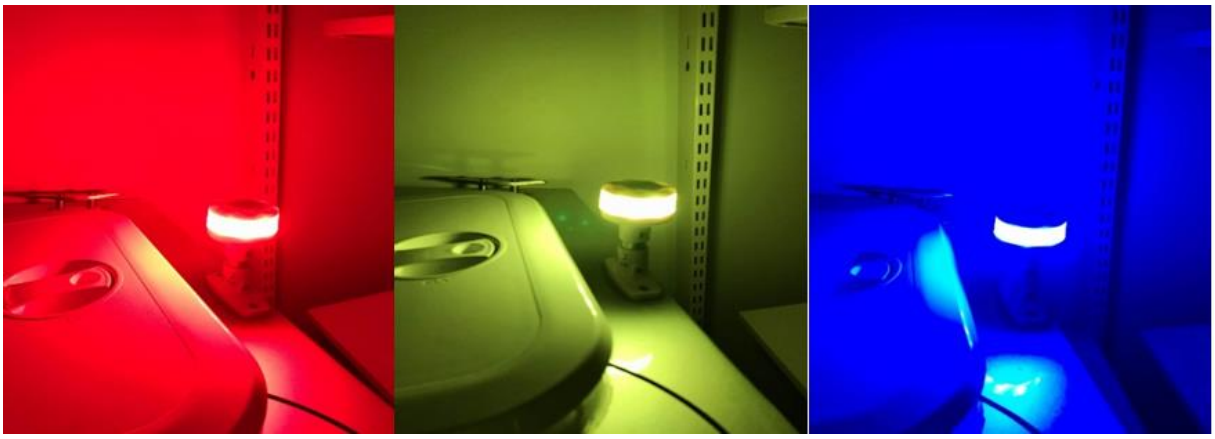
Forarbeidet utført av USN gir gruppen et godt utgangspunkt, da den tilfredsstiller de fleste kravene for å kunne delta i Autodrone konkurransen. Nedenfor er alle kravene *[3]* den allerede tilfredsstilte da den kom til HVL på høstsemesteret i 2021:

- **Kommunikasjon:** Sjødronen skal ikke kunne sende eller motta kommandoer fra operatørene når den er i autonom-modus.
- **Plasserbar:** Sjødronen skal kunne plasseres manuelt.
- **Energikilde:** Sjødronen må være batteridrevet. Batteriene skal være forseglet for å redusere faren for lekkasje av syre eller kaustiske elektrolytter. Den åpne kretsspenningen til ethvert batteri (eller batterisystem) skal ikke overstige 60 V DC.
- **Nødstoppbryter:** Sjødronen må minst ha en rød nødstopp plassert på seg, se *Figur 31*. Når denne blir aktivert, må den øyeblikkelig koble fra all strøm til motorene og aktuatorene.

Figur 31 Nødstopp på sjødronen *(privat)*

- **Trådløs nødstopp:** Sjødronen må ha en trådløs nødstopp som i en nødsituasjon gjør det mulig for operatørene til å koble fra all strøm til motorene og aktuatorene fra land. *Trådløs nødstopp på sjødronen er når radiokontrolleren setter sjødronen i manuell-modus.*
- **Fremdrift:** Ethvert fremdriftssystem kan brukes, eksempelvis trustere eller årer. *Fremdriftssystemet på sjødronen er trusterne.*
- **Fjernstyrbarhet:** Sjødronen skal være fjernstyrt for operatørene på land.
- **Slepbar:** Sjødronen må kunne slepes.

- **Visuell tilbakemelding:** Det er pålagt å implementere et visuelt tilbakemeldingssystem som indikerer statusen til sjødronen. Se *Figur 32*.



Figur 32 LED-lys på sjødronen som lyser rødt, grønt/gult og blått *(privat)*

Sjødronen indikerer at:
-Nødstopp er aktivert ved rødt lys.
-Sjødronen er fjernstyrt ved grønt/gult lys.
-Sjødronen er autonom ved blått lys.

- **Vekt:** Hele sjødronen skal veie mindre enn 70 kg.
- **Nyttelast:** Sjødronen må ha et sted å montere et kamera med en uhindret utsikt til forsiden av sjødronen.
- **Sikkerhet:** Alle skarpe, spisse og bevegelige deler må dekkes til og merkes.

Dette medfører at det er bare et krav igjen, for at sjødronen skal få delta i Autodrone. Dette kravet *[3]* er:

- **Autonomi:** Sjødronen må være helt autonom og ta autonomibeslutningene ombord på sjødronen.

4.2 USN sine algoritmer

I dette delkapittelet forklares kort om algoritmene fra USN som lå inne på sjødronen, og hva de brukes til.

4.2.1 controller_radio.py (Se Vedlegg 10 – controller_radio.py)

Denne algoritmen muliggjør at sjødronen kan fjernstyres via radiokontrolleren. Dette er altså koden som knytter radiokontrolleren til servomotorene, trusterne og LED-lampen på sjødronen, ved å publisere til topicet `cmd_vel`. Algoritmen publiserer også til topicet `control_auto` som forteller om sjødronen er i fjernstyrt eller autonom modus.

4.2.2 servo.py (Se Vedlegg 11 – servo.py)

Denne algoritmen brukes for å styre servomotorene på sjødronen og er laget av bachelorgruppen fra USN som deltok på Autodrone 2021. Her subscriber algoritmen på topicet `cmd_vel` og roterer servoene basert på styringsvinkelens verdi.

4.2.3 thruster.py (Se Vedlegg 12 – thruster.py)

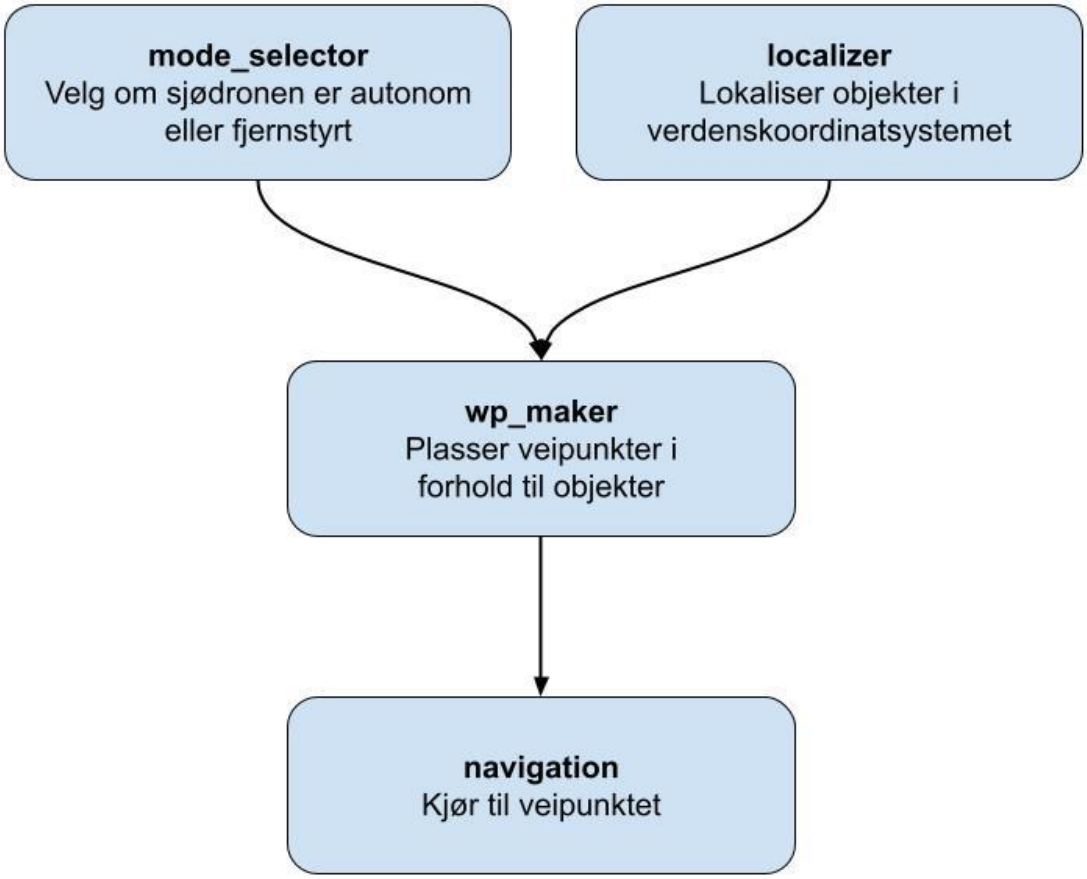
Denne algoritmen brukes for å styre hastigheten og dreieretningen på trusterne til sjødronen, ved å subscribe på topicet `cmd_vel`. Algoritmen er laget av bachelorgruppen fra USN som deltok på Autodrone 2021.

4.2.4 com-LED.py (Se Vedlegg 13 – com-LED.py)

Denne algoritmen brukes for å styre LED-lampen på sjødronen og er laget av bachelorgruppen fra USN som deltok på Autodrone 2021. Algoritmen bruker verdier fra topicet `cmd_vel` og `control_auto`, til å sette fargen på LED-lampen.

5 Algoritmene gruppen har laget selv

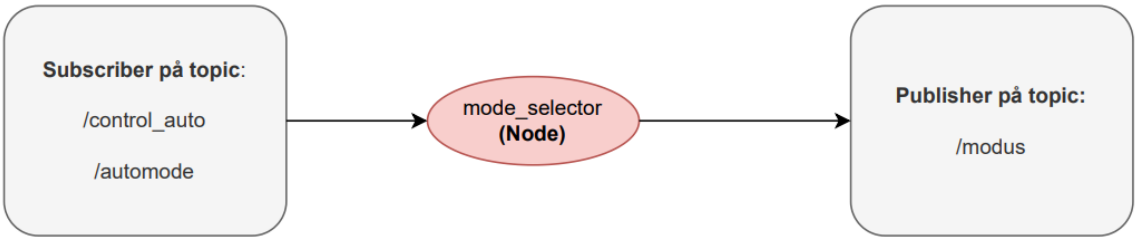
I dette kapittelet forklares det hva algoritmene som gruppen har laget gjør og brukes til. På *Figur 33* er en enkel oversikt over dette og hvordan algoritmene henger sammen, for å gjøre sjødronen klar til å utføre Speed Gate oppdraget.



Figur 33 Flytskjema som viser hva de forskjellige algoritmene gjør, og hvordan de henger sammen *(privat)*

5.1 mode_selector.py (Se Vedlegg 14 – mode_selector.py)

Denne algoritmen brukes for å finne ut om sjødronen er i fjernstyrt- eller autonom-modus, og for å bestemme hvilket konkurranseoppdrag som skal utføres. Se *Figur 34* for hvilke topics noden `mode_selector` subscriber og publisher på.



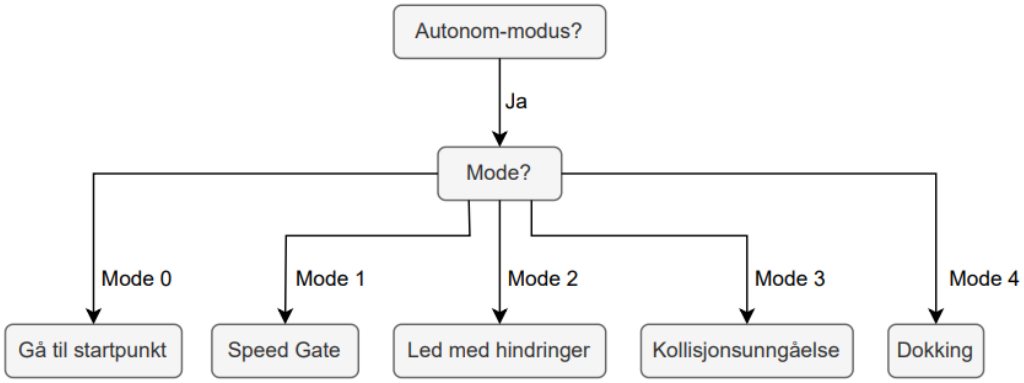
Figur 34 Oversikt over hvilke topics noden mode_selector subscriber og publisher på *(privat)*

Ved hjelp av en boolsk verdi som blir hentet fra topicet `control_auto`, bekreftes det om sjødronen er i autonom-modus eller ikke. Dersom den er i autonom modus, går algoritmen videre med å finne ut hvilket oppdrag sjødronen skal utføre, ved hjelp av en heltallsverdi som blir hentet fra topicet `automode`. Dette topicet er planlagt å bli skrevet til manuelt før start, hvor verdien forteller hvilket konkurranseoppdrag sjødronen skal utføre, som beskrevet i *Tabell 2*. Om sjødronen er i modus 0 skal den kjøre til et gitt punkt, bli der, og vente på ny instruks.

Mode	Oppdrag
0	Gå til startpunkt
1	Speed Gate
2	Led med hindringer
3	Kollisjonsunngåelse
4	Dokking

Tabell 2 Oversikt over hvilket mode som tilsvarer hvilket konkurranseoppdrag

I *Figur 35* er det laget et enkelt flytdiagram, som grafisk viser hvordan denne algoritmen fungerer.

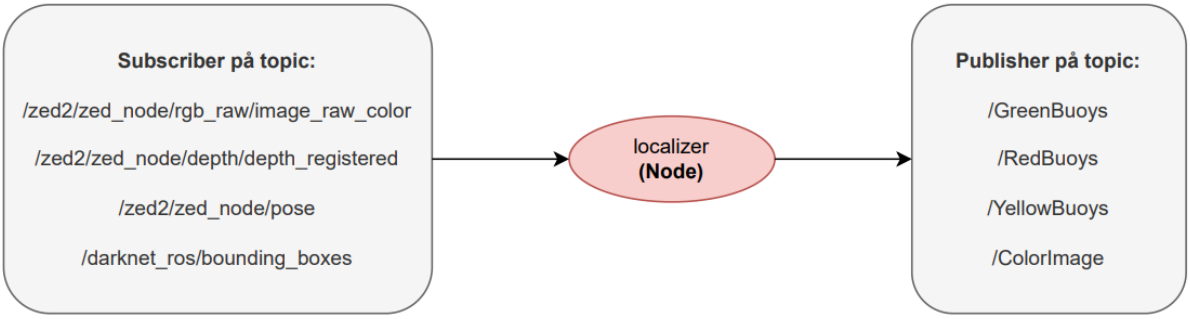


Figur 35 Flytdiagram for mode_selector.py *(privat)*

5.2 localizer.py (Se Vedlegg 15 – localizer.py)

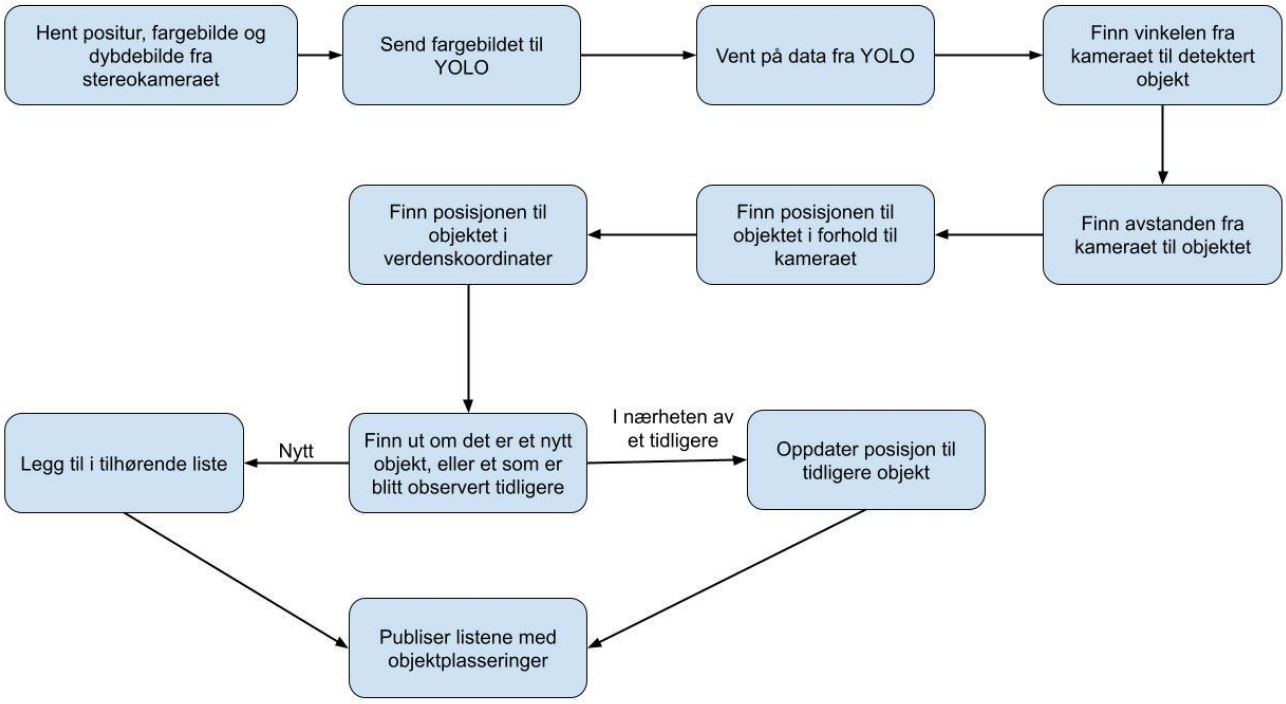
Denne algoritmen brukes for å lokalisere objektene som sjødronen detekterer. Dette gjøres ved å bruke data fra YOLO algoritmen, samt stereokameraets dybdebilde, fargebilde og positur. Posituren til stereokameraet er posisjonen i verdenskoordinatsystemet, på kartesisk form, og orienteringen på kvaternion form ([x, y, z, w]).

Topics som noden `localizer` subscriber og publisher på er vist på *Figur 36*.



Figur 36 Oversikt over hvilke topics noden localizer subscriber og publisher på *(privat)*

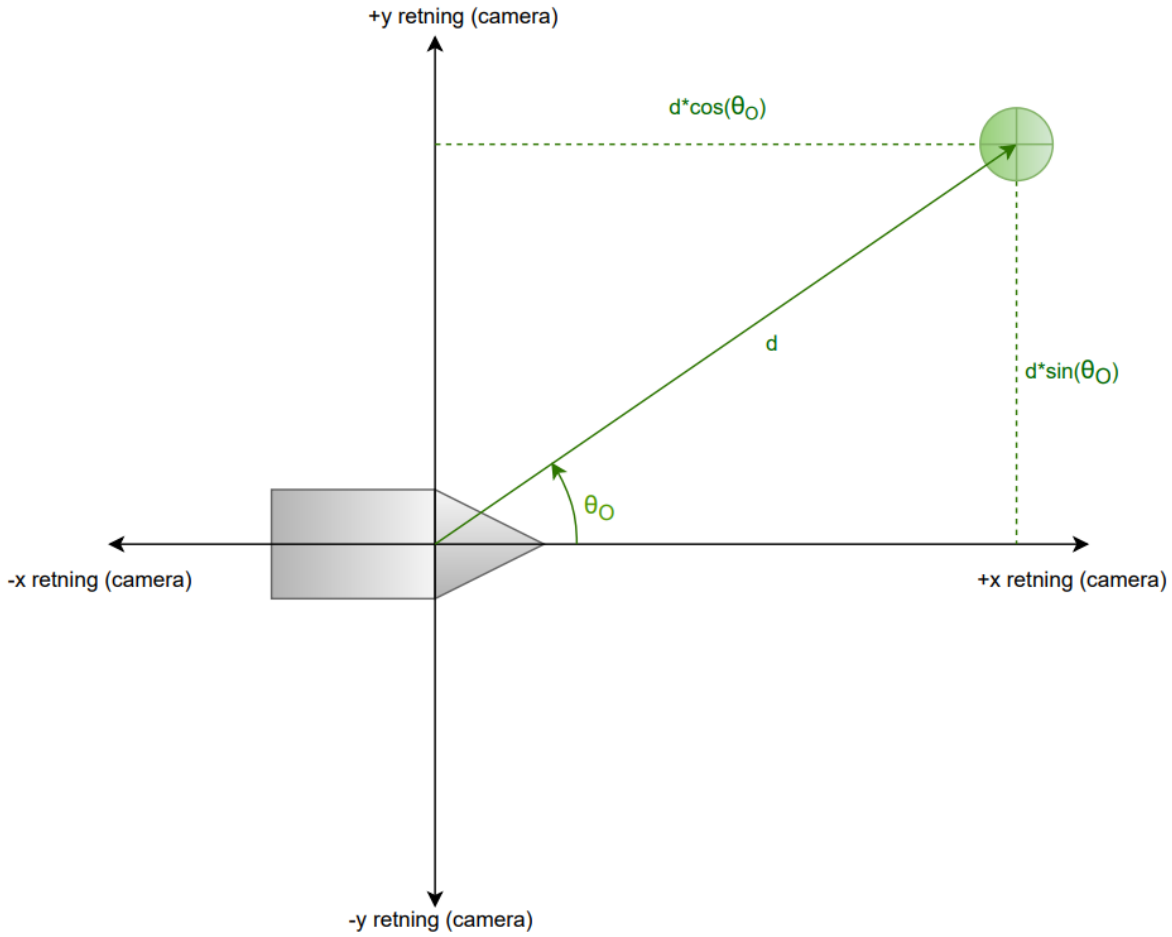
Algoritmen bruker disse dataene til å lokalisere bøyer og båter som er innenfor synsfeltet og rekkevidden til stereokameraet. Når sjødronen vet hvor objektene er i forhold til seg selv, finner den også orienteringen til sjødronen og legger plasseringen av objektene i forhold til verdenskoordinatsystemet inn i en liste. Et enkelt flytdiagram over hvordan algoritmen fungerer er vist på *Figur 37*.



Figur 37 Flytdiagram for localizer.py *(privat)*

Det er også lagt til en filter-funksjon, som skal gjøre at samme bøye ikke blir lagt til flere ganger. Dette filteret fungerer slik at, om det er to bøyer som er nærmere enn 2 meter fra hverandre, legges ikke bøyen til i listen, men posisjonen til den aktuelle bøyen endres til midtpunktet mellom tidligere posisjon og den nye posisjonen, for å holde listen med bøyer mer oppdatert.

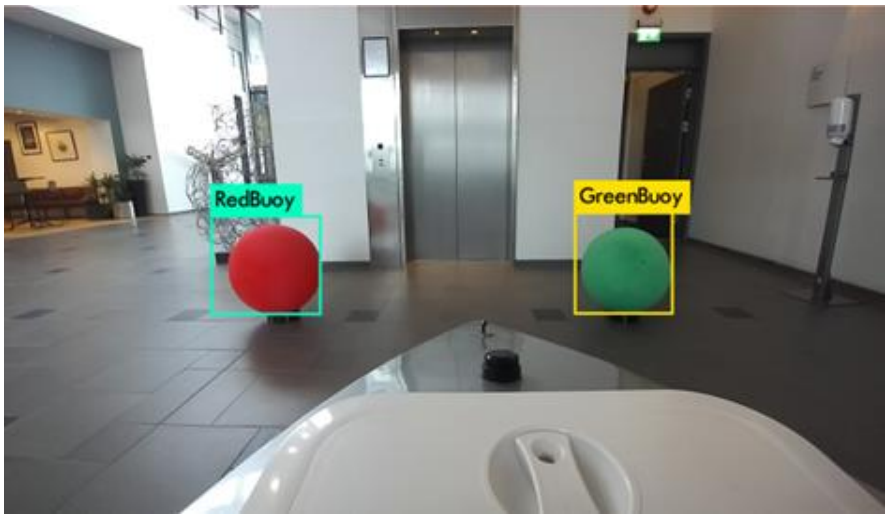
For å finne ut hvor objektene befinner seg i forhold til sjødronens kamera, ble *Figur 38* tegnet, for å få oversikt over nødvendige variabler.



Figur 38 Hvordan 2D koordinatene til objektene i forhold til kameraet på sjødronen *(privat)*

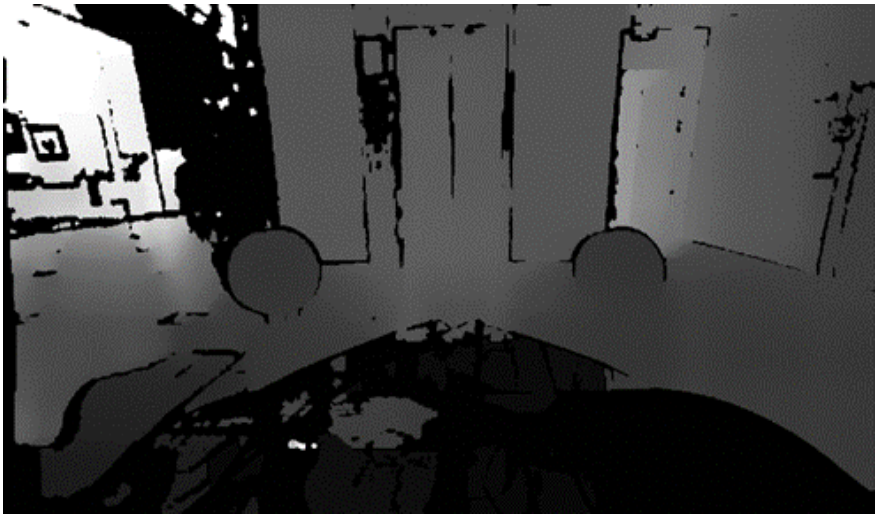
Resultatene fra YOLO, som er avbildet på *Figur 39*, ble brukt til å finne variablene, ved å velge en piksel innenfor boksen rundt objektet. Derfra kan vinkelen θ_O finnes ved hjelp av Formel *1*, hvor 120° og 60° stammer fra kameraets oppgitte synsvinkel på 120°.

$$\theta_{Object} = \theta_O = -\frac{(x\ verdien\ til\ ønsket\ piksel) \cdot 120°}{(bildets\ bredde)} + 60° \qquad (1)$$



Figur 39 Fargebilde med objekt detektering *(privat)*

For å finne avstanden d, brukes verdien fra valgt piksel i dybdebildet, som avbildet i *Figur 40*. Denne verdien er oppgitt i meter, hvor posisjonen til objektet enkelt kan regnes ut ved hjelp av *Formel 2*, der x- og y-verdiene er funksjoner av d og θ_O.



Figur 40 Dybdebildet fra stereokameraet, hvor hver piksel har en tallverdi, som representerer avstanden fra kameraet. I *Figur 38* er denne avstanden betegnet som *d (privat)*

$$\left(d \cdot cos(\theta_O),\quad d \cdot sin(\theta_O)\right) \qquad (2)$$

Med dette kan 2D-transformasjonsmatrisen ${}^{Camera}T_{Object} = {}^{C}T_{O}$ settes opp, ved å bruke tidligere kunnskaper fra robotikkemnet (ELE306) *[27]*.

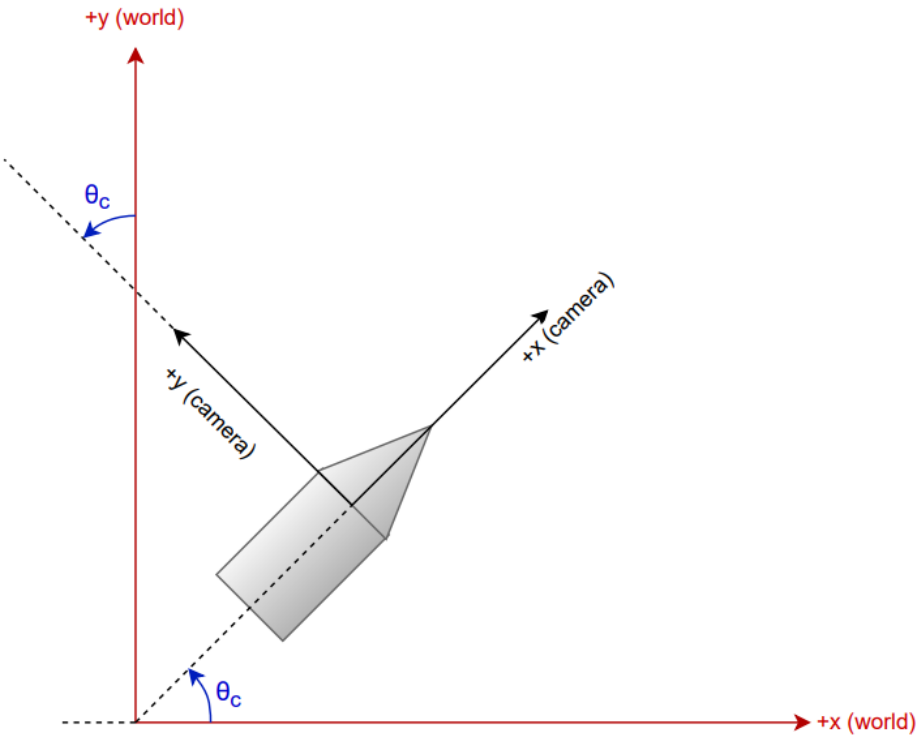
En transformasjonsmatrise forteller noe om posituren til et objekt, i forhold til et annet objekt. Altså hvor noe er posisjonert, og hvor mye det er rotert, i forhold til noe. Transformasjonsmatrisen i *Formel 3* representerer objektene, i forhold til kameraet på sjødronen.

$$ {}^{C}T_{O} = \begin{bmatrix} 1 & 0 & {}^{C}x_{O} \\ 0 & 1 & {}^{C}y_{O} \\ 0 & 0 & 1 \end{bmatrix} \text{ hvor } \begin{matrix} {}^{C}x_{O} = d \cdot sin(\theta_{O}) \\ {}^{C}y_{O} = d \cdot cos(\theta_{O}) \end{matrix} \qquad (3) $$

Deretter måtte transformasjonsmatrisen ${}^{World}T_{Camera} = {}^{W}T_{C}$ settes opp. For å gjøre dette konverteres IMU-dataene fra stereokameraet, som er oppgitt på kvaternion form, om til en vinkel rotert om z-aksen, ved hjelp av *Formel 4*, som er funnet på forumet StackOverflow *[28]*. Denne vinkelen representerer rotasjonen til sjødronen i verdenskoordinatsystemet, som illustrert i *Figur 41*.

$$ \theta_{\text{Camera}} = \theta_{\text{C}} = \text{atan2}(2.0 \cdot (\text{q.x} \cdot \text{q.y} + \text{q.w} \cdot \text{q.z}), \quad \text{q.w}^2 + \text{q.x}^2 - \text{q.y}^2 - \text{q.z}^2) \qquad (4) $$

Hvor q er orienteringsverdiene til IMU-en.



Figur 41 Viser hvordan koordinatsystemet til kameraet orienteres i forhold til verdenskoordinatene når sjødronen roteres *(privat)*

Transformasjonsmatrisen i *Formel 5* representerer kameraet på sjødronen i forhold til verdenskoordinatsystemet.

$$ {}^{W}\mathrm{T}_{C} = \begin{bmatrix} \cos(\theta_C) & -\sin(\theta_C) & {}^{W}x_C \\ \sin(\theta_C) & \cos(\theta_C) & {}^{W}y_C \\ 0 & 0 & 1 \end{bmatrix}, \quad \text{hvor} \quad \begin{matrix} {}^{W}x_C = x\ posisjon\ fra\ IMU \\ {}^{W}y_C = y\ posisjon\ fra\ IMU \end{matrix} \qquad (5) $$

For å finne ut hvor objektene befinner seg i forhold til verdenskoordinatsystemet, måtte transformasjonsmatrisen ${}^{World}\mathrm{T}_{Object} = {}^{W}\mathrm{T}_{O}$ regnes ut. Dette gjøres ved å multiplisere de to matrisene funnet tidligere, se *Formel 6*. Her representerer symbolet $\oplus$ en matrisemultiplikasjon av transformasjonsmatriser fra robotikkemnet (ELE306) *[27]*.

$$ \begin{aligned} {}^{W}T_O &= {}^{W}T_C \oplus {}^{C}T_O \\ &= \begin{bmatrix} \cos(\theta_C) & -\sin(\theta_C) & {}^{W}x_C \\ \sin(\theta_C) & \cos(\theta_C) & {}^{W}y_C \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & {}^{C}x_O \\ 0 & 1 & {}^{C}y_O \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta_C) & -\sin(\theta_C) & {}^{W}x_O \\ \sin(\theta_C) & \cos(\theta_C) & {}^{W}y_O \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \qquad (6) $$

$$ \text{Hvor} \quad \begin{matrix} {}^{W}x_O = {}^{C}x_O \cos(\theta_C) - {}^{C}y_O \sin(\theta_C) + {}^{W}x_C \\ {}^{W}y_O = {}^{C}x_O \sin(\theta_C) + {}^{C}y_O \cos(\theta_C) - {}^{W}y_C \end{matrix} $$

Fordi bøyene er runde, trenger ikke orienteringen deres å bli tatt hensyn til i denne oppgaven. Dermed er det bare posisjonen til bøyene, ${}^{W}x_O$ og ${}^{W}y_O$, som er nødvendig å vite.

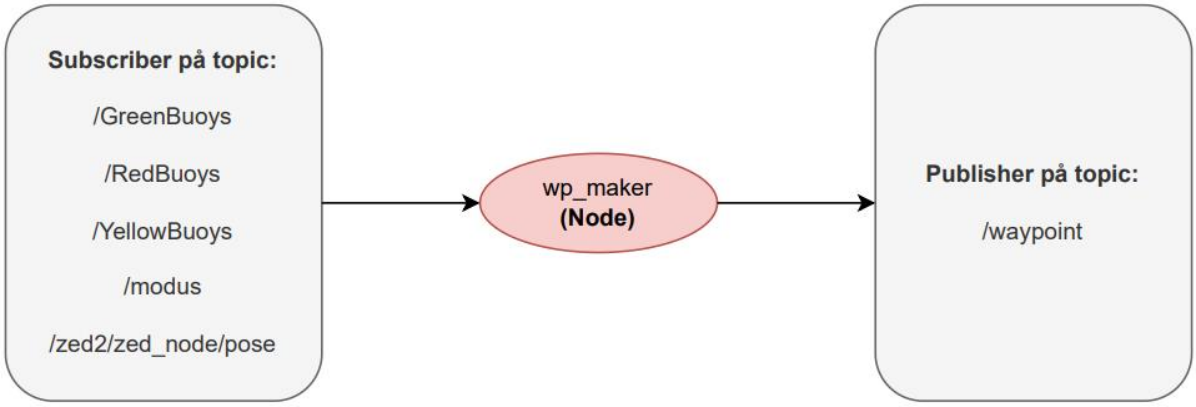
For at dette skal gi så gode resultater som mulig, er det viktig at all dataen som hentes fra sensorene er fra samme tidspunkt, da en liten forskyvning i tid mellom positur og bildene kan gi store utfall på resultatet. Dette ble tydeliggjort når algoritmen først ble testet, og dette ikke var blitt tatt hensyn til.

Av denne grunn ble det lagt inn en bufferfunksjon i algoritmen. Den lagrer et sett med data fra stereokameraet, og sender fargebildet fra dette settet til YOLO, for objekt detektering. Når resultatet fra YOLO er klart, blir det òg datasettet brukt til å lokalisere objektene. Dette gir færre iterasjoner av algoritmen, men øker nøyaktigheten når sjødronen er i bevegelse.

5.3 wp_maker.py (Se Vedlegg 16 – wp_maker.py)

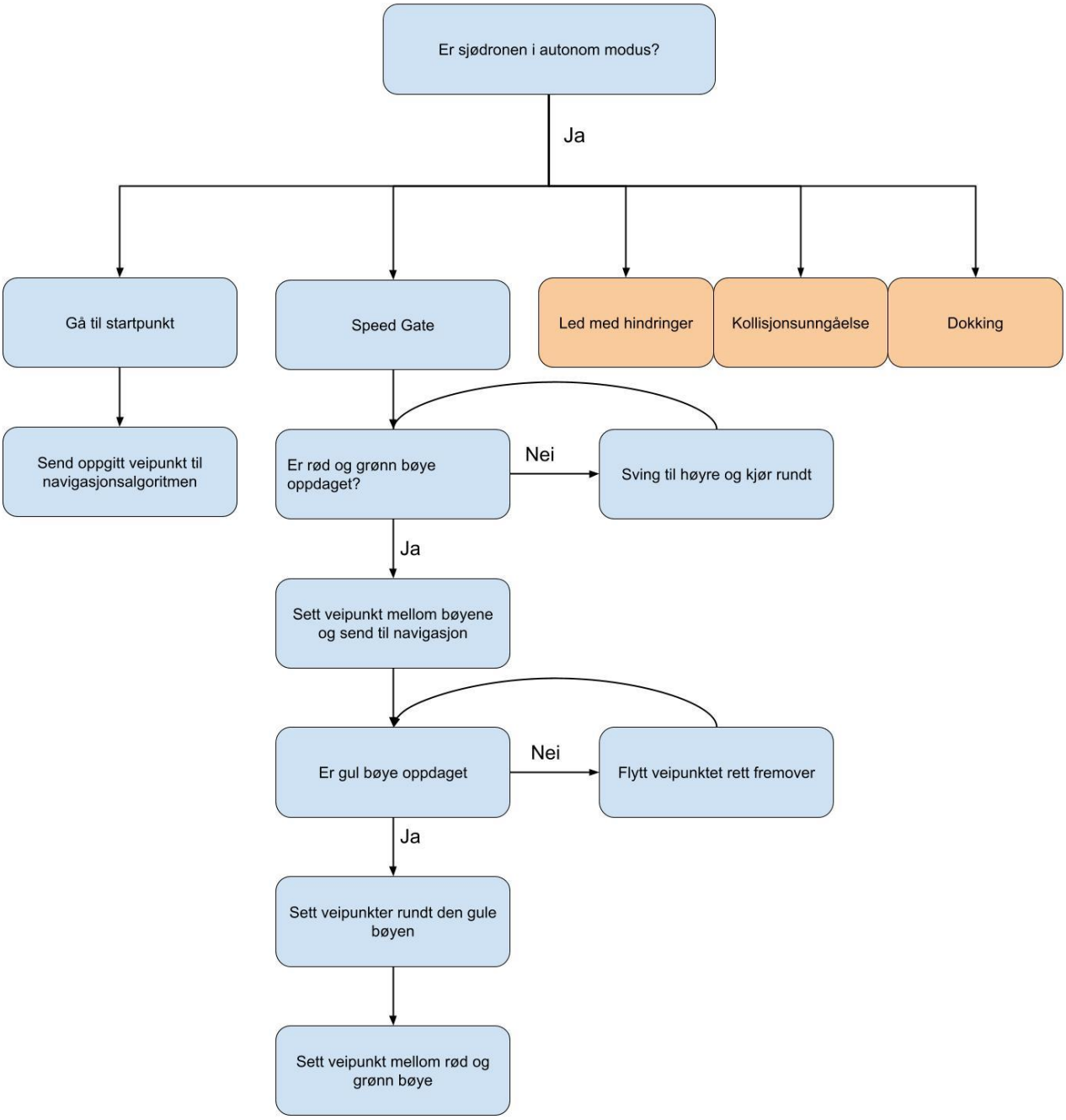
Denne algoritmen brukes for å planlegge banen sjødronen skal kjøre, ved å plassere veipunkter i verdenskoordinatsystemet. Den bruker plasseringen til objektene fra lokaliseringsalgoritmen og posisjonen til sjødronen, til å beregne plasseringen av veipunkter i forhold til hvilket oppdrag som skal utføres.

Topics som noden `wp_maker` (waypoint maker) subscriber og publisher på er vist på *Figur 42.*



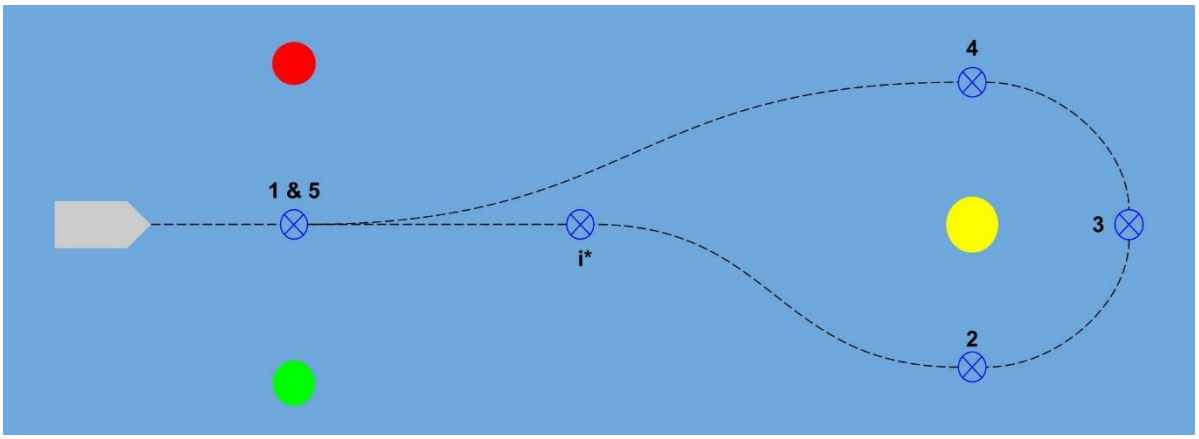
Figur 42 Oversikt over hvilke topics wp_maker subscriber og publisher på *(privat)*

Ved å hente resultatene fra lokaliseringsalgoritmen, får algoritmen inn plasseringen, til de forskjellige objektene, som sjødronen skal navigere i forhold til. Et enkelt flytdiagram over hvordan algoritmen fungerer er vist på *Figur 43*.



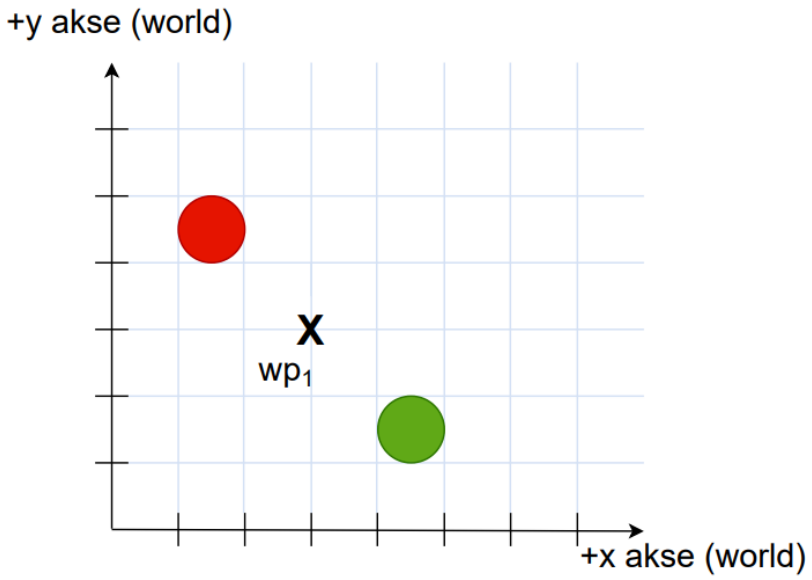
Figur 43 Flytdiagram for wp_maker.py *(privat)*

For Speed Gate oppdraget vil objektene som er relevante, være én bøye av hver farge og *Figur 44* illustrer hvordan gruppen ønsker å plassere veipunktene for oppdraget.



Figur 44 Oversikt over hvor gruppen tenker å sette veipunkt under Speed Gate oppdraget *(privat)*

Det første sjødronen må vite, er hvor den grønne og røde bøyen befinner seg, slik at den kan sette det første veipunktet mellom dem, som er grafisk fremstilt i *Figur 45*. Dette gjøres ved hjelp av *Formel 7* og *8*.



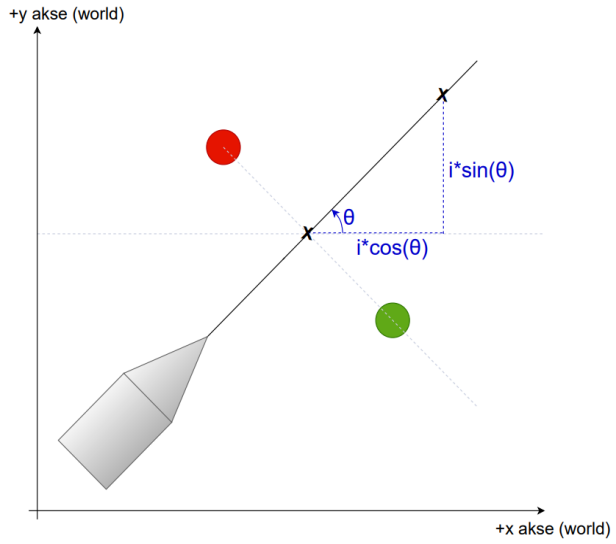
Figur 45 Hvordan første veipunkt skal plasseres *(privat)*

$$x_{wp_1} = x_{red} + \frac{x_{green} - x_{red}}{2} \quad (7)$$

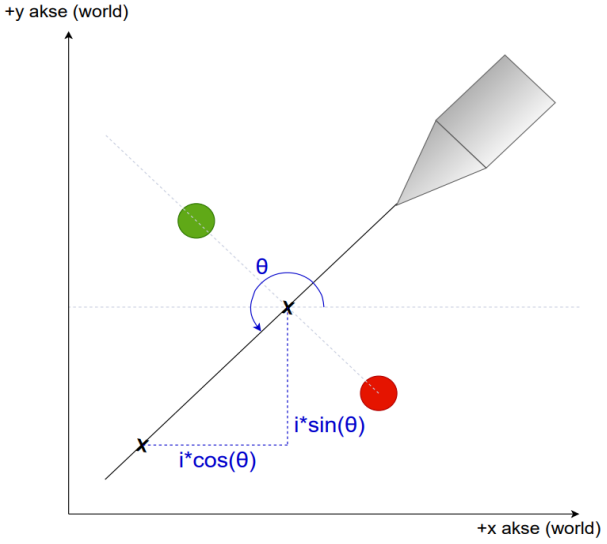
$$y_{wp_1} = y_{red} + \frac{y_{green} - y_{red}}{2} \quad (8)$$

Grunnet lange distanser vil ikke sjødronen klare å lokalisere den gule bøyen når den er ved det første veipunktet. Dette medfører at sjødronen ikke vil klare å plassere veipunktene rundt den gule bøyen, før den har kommet nærmere. Dette er tatt i betraktning, slik at et midlertidig veipunkt (wp_{i*}) settes én meter fremover, helt til sjødronen klarer å lokalisere den gule bøyen.

θ er vinkelen mellom wp_1 og wp_{i*}, og utregningen av den vil være avhengig av plasseringen til den grønne og røde bøyen i verdenskoordinatsystemet. Når sjødronen er i startposisjon, skal den grønne bøyen være på høyre side og den røde bøyen på venstre side, i forhold til sjødronen. Dersom den grønne bøyen er til høyre for den røde, i verdenskoordinatsystemet, som i *Figur 46*, brukes *Formel 9* til å beregne vinkelen θ. Om den grønne bøyen derimot er til venstre for den røde, som i *Figur 47*, brukes *Formel 10*.



Figur 46 Illustrerer når den grønne bøyen er til høyre for den røde bøyen i verdenskoordinatsystemet *(privat)*



Figur 47 Illustrerer når den grønne bøyen er til venstre for den røde bøyen i verdenskoordinatsystemet *(privat)*

$$\theta = \frac{\pi}{2} + tan^{-1}\left(\frac{y_{green} - y_{red}}{x_{green} - x_{red}}\right) \quad (9)$$

$$\theta = \frac{3\pi}{2} + tan^{-1}\left(\frac{y_{green} - y_{red}}{x_{green} - x_{red}}\right) \quad (10)$$

For å finne x- og y-koordinatene til wp_{i*} ble *Formel 11* og *12* brukt. Her øker verdien $"i"$ med én når sjødronen når veipunktet.

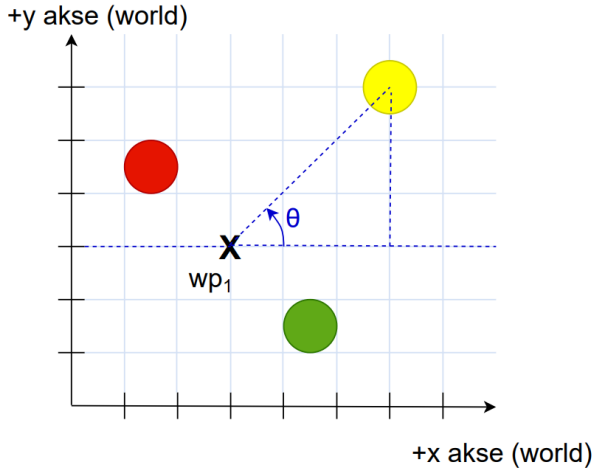
$$\mathrm{x}_{\mathrm{wp}_{i^*}} = \mathrm{x}_{\mathrm{wp}_1} + \mathrm{i} \cdot \cos(\theta) \quad (11)$$

$$y_{wp_{i^*}} = \mathrm{y}_{\mathrm{wp}_1} + \mathrm{i} \cdot \sin(\theta) \quad (12)$$

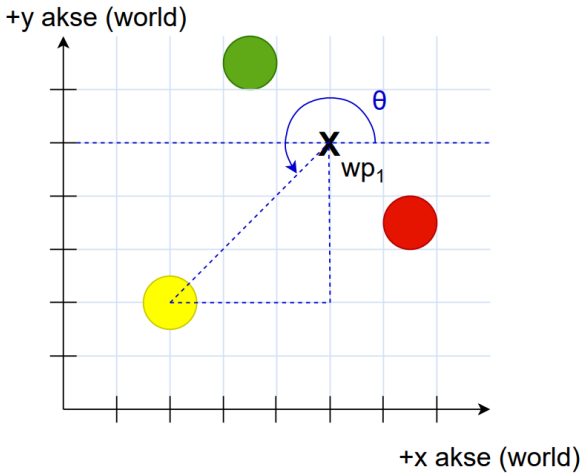
Når sjødronen klarer å lokalisere den gule bøyen, finner algoritmen en ny θ, som er vinkelen mellom det første veipunktet og den gule bøyen. θ regnes ut på to ulike måter, med *Formel 13* eller *14*, avhengig av de to mulige scenarioene vist på *Figur 48* og *Figur 49*:

$$\theta = tan^{-1}\left(\frac{y_{wp_1} - y_{yellow}}{x_{wp_1} - x_{yellow}}\right) \quad (13)$$

$$\theta = \pi + tan^{-1}\left(\frac{y_{wp_1} - y_{yellow}}{x_{wp_1} - x_{yellow}}\right) \quad (14)$$

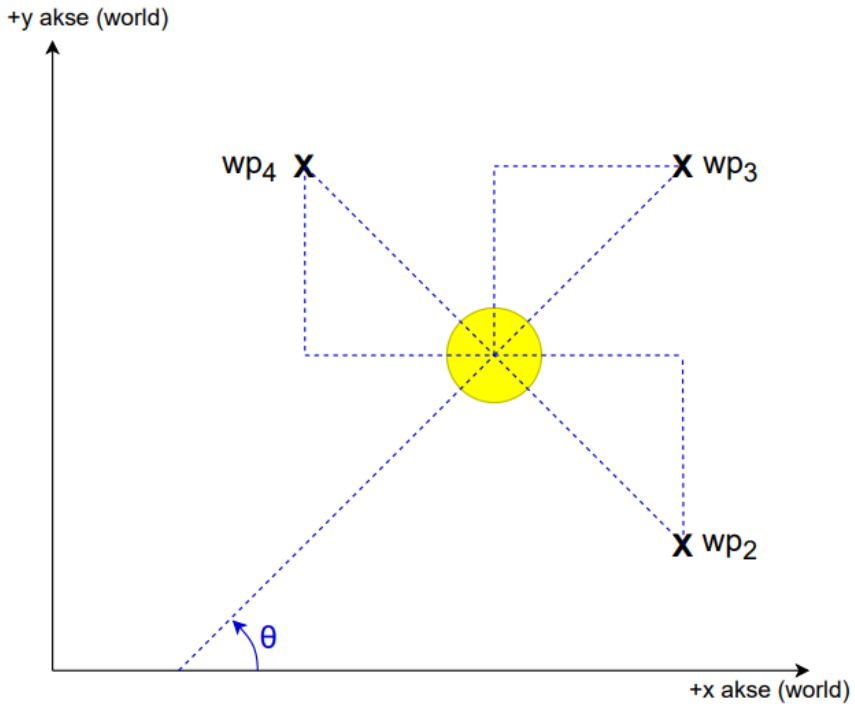


Figur 48 viser når x_{wp_1} er mindre enn x_{yellow} *(privat)*



Figur 49 viser når viser når x_{wp_1} er større enn x_{yellow} *(privat)*

Veipunktene blir deretter plassert rundt den gule bøyen som vist på *Figur 50* ved hjelp av *Formel 15* til *20*. Veipunktene blir plassert med en konstant avstand "C" fra den gule bøyen.



Figur 50 Veipunktene rundt den gule bøyen *(privat)*

$$x_{wp_2} = x_{yellow} + C \cdot \cos\left(\frac{\pi}{2} - \theta\right) \quad (15) \qquad y_{wp_2} = y_{yellow} - C \cdot \sin\left(\frac{\pi}{2} - \theta\right) \quad (16)$$

$$x_{wp_3} = x_{yellow} + C \cdot \sin(\theta) \quad (17) \qquad y_{wp_3} = y_{yellow} + C \cdot \cos(\theta) \quad (18)$$

$$x_{wp_4} = x_{yellow} - C \cdot \cos\left(\frac{\pi}{2} - \theta\right) \quad (19) \qquad y_{wp_4} = y_{yellow} + C \cdot \sin\left(\frac{\pi}{2} - \theta\right) \quad (20)$$

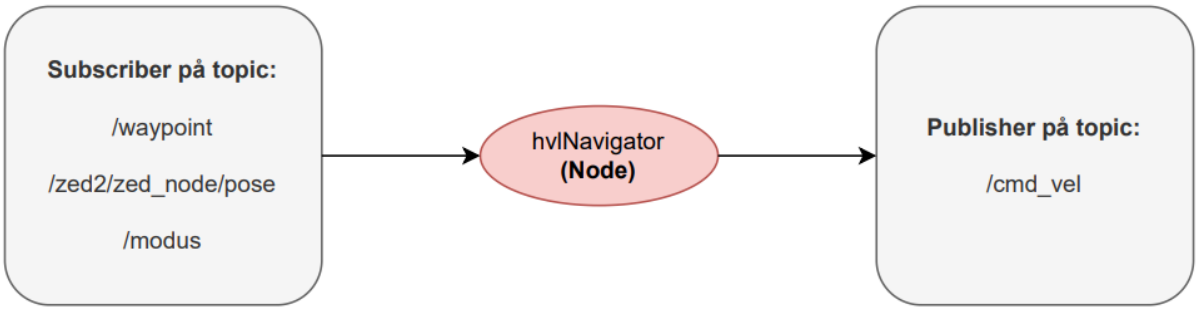
Når alle veipunktene rundt den gule bøyen er plassert setter algoritmen det siste veipunktet lik det første veipunktet, som var mellom den røde og grønne bøyen som *Formel 21* og *22* viser.

$$x_{wp_5} = x_{wp_1} \quad (21) \qquad y_{wp_5} = y_{wp_1} \quad (22)$$

5.4 navigation.py (Se Vedlegg 17 – navigation.py)

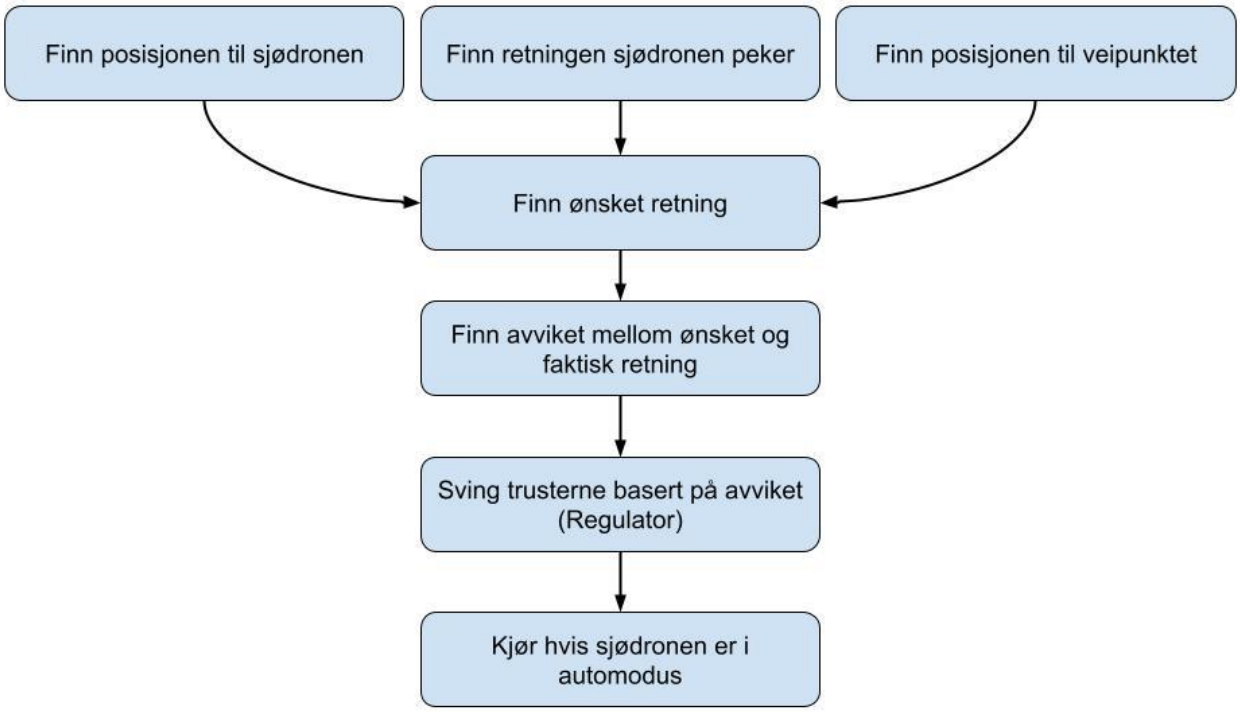
Denne algoritmen består av en regulator som får sjødronen til å kjøre til et veipunkt, gitt av algoritmen wp_maker, eller lagt inn manuelt. Ved hjelp av IMU-data fra stereokameraet, finner den ønsket retning som sjødronen skal kjøre mot for å nå veipunktet, og styrer sjødronen i den retningen, ved hjelp av en regulator.

Topics som noden `hvlNavigator` subscriber og publisher på er vist på *Figur 51*.



Figur 51 Oversikt over hvilke topics hvlNavigator subscriber og publisher på *(privat)*

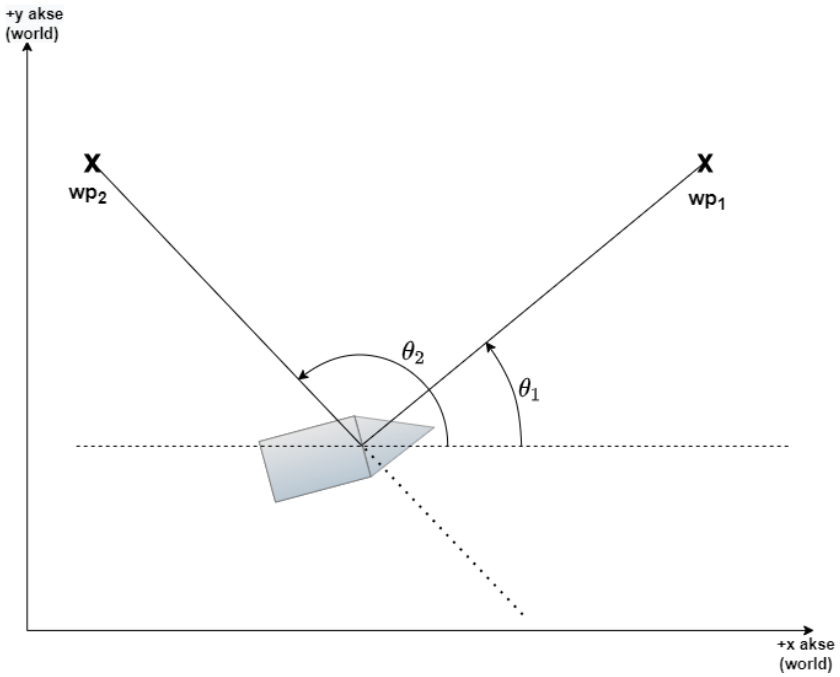
Et enkelt flytdiagram over hvordan algoritmen fungerer er vist på *Figur 52*.



Figur 52 Flytdiagram for navigation.py *(privat)*

På grunn av de geometriske begrensningene til tangens, vil det kunne oppstå to senarioer. Disse er begge tatt høyde for, ved å sjekke om sjødronen er til venstre eller høyre for veipunktet, sett i verdenskoordinatsystemet.

I *Figur 53* er wp_1 til høyre for sjødronen, hvor *Formel 23* brukes for å finne ønsket vinkel θ_1. wp_2 er til venstre for sjødronen og *Formel 24* brukes for å finne ønsket vinkel θ_2.



Figur 53 Ønsket kjøreretning for to veipunkter *(privat)*

$$\theta_1 = tan^{-1}\left(\frac{x_{wp_1} - x_{sjødrone}}{y_{wp_1} - y_{sjødrone}}\right) \quad (23)$$

$$\theta_2 = \pi + tan^{-1}\left(\frac{x_{wp_2} - x_{sjødrone}}{y_{wp_2} - y_{sjødrone}}\right) \quad (24)$$

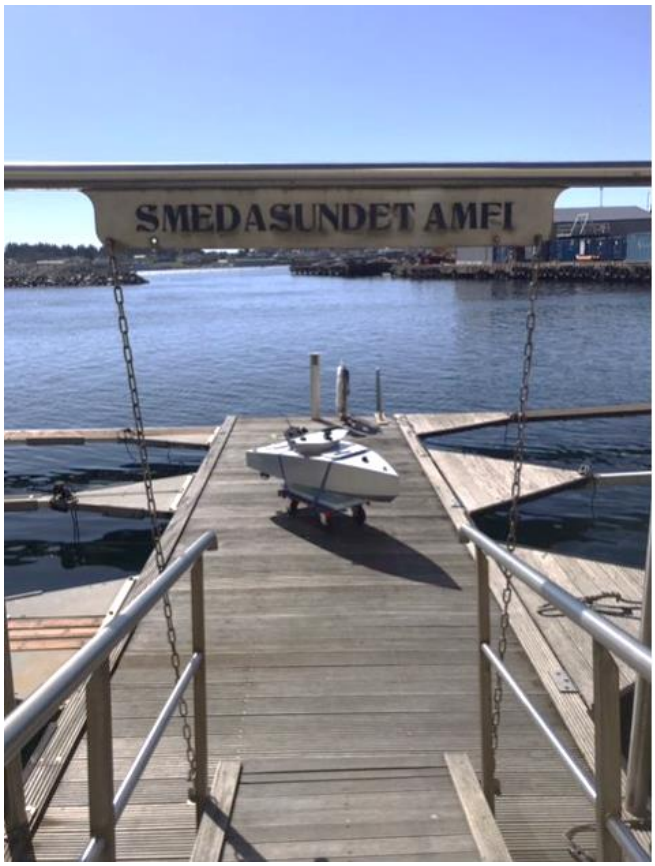
Avviket som blir brukt til regulatoren, blir regnet ut ved å trekke vinkelen sjødronen har (θ_{camera}) fra ønsket vinkel ($\theta_{ønsket}$), som i *Formel 25*. Vinkelen til sjødronen θ_{camera} blir regnet ut på samme måte som i lokaliseringsalgoritmen (*Formel 4*), hvor orienteringen til stereokameraet, på kvaternion form, blir konvertert til en vinkel, som representerer rotasjonen til dronen om z-aksen.

$$Avvik = \theta_{ønsket} - \theta_{camera} \quad (25)$$

6 Testing

I starten av oppgaven tok gruppen kontakt med sjøfartsdirektoratet. Dette var med tanke på om det var nødvendig med tillatelse, for å sjøsette og teste sjødronen i Haugesund. Kommunikasjonen foregikk på e-post og gruppen fikk som svar at når det gjelder regelverk, er det avhengig av sjødronens størrelse og operasjon. Sjødroner som er under 15 meter og som ikke benyttes i kommersiell bruk, har få eller ingen sertifikat eller tilsynskrav, så lenge farkosten ikke er til sjenanse for farled eller trafikk. Sjødronen for dette prosjektet er rundt 2 meter lang og 1 meter bred, som vil si at gruppen ikke trengte noen tillatelse for å sjøsette eller teste sjødronen. I tillegg spurte gruppen om det var mulig å få teste sjødronen hos sjøfartsdirektoratet, som har et dedikert testområde. Dette var mulig, men med tanke på størrelsen på sjødronen var ikke et dedikert testområde nødvendig.

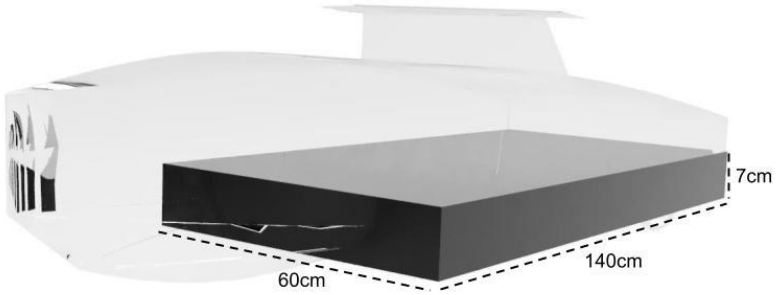
Gruppen tok derfor kontakt med styret for borettslaget Smedasundet Amfi, da dette var en plass der sjødronen enkelt kunne trilles ned fra campus. Dette var de positive til og Smedasundet Amfi var området der sjøsettingen og testingen på vann foregikk i løpet av bacheloroppgaven, se *Figur 54*.



Figur 54 Sjødronen på gruppens testområde utenfor Smedasundet Amfi *(privat)*

6.1 Sjøsetting av sjødronen

Gruppemedlemmet som var tilskuer på Autodrone konkurransen i 2021, fikk vite av USN-studentene, med identisk sjødrone, at de måtte legge ballast oppi sjødronen. Dette var for å unngå at sjødronen skulle tippe eller ligge dårlig på vannet. Det ble estimert hvor stort volum sjødronen burde fortrenge, som vist i *Figur 55*, for å få vannlinjen i passende høyde. Volumet ble regnet ut med *Formel 26* og brukt i *Formel 27* til å regne ut vekten sjødronen burde ha, med ballast. Dette var for å få en formening om hvor mye ballast som måtte anskaffes.



Figur 55 Forenklet skisse av sjødronens volum som fortrenger vann *(privat)*

$$\text{Volum} = \text{lengde} \cdot \text{bredde} \cdot \text{høyde} = 140\ \text{cm} \cdot 60\ \text{cm} \cdot 7\ \text{cm} = 58800\ \text{cm}^3 \qquad (26)$$

$$\text{Masse} = \frac{\text{Fortrengt Volum}}{\rho_{\text{sjøvann}}} = \frac{58800\ \text{cm}^3}{1{,}03\ {}^{\text{g}}/_{\text{cm}^3}} \approx 57087{,}4\ \text{g} \approx 57\ \text{kg} \qquad (27)$$

Sjødronen ble veid til cirka 30 kilogram uten ballast, som medførte at en ballast på 20-30 kilogram i teorien måtte anskaffes. I praksis ble det funnet ut at en ballast på 15-20 kilogram fungerte fint.

For å få tak i ballasten til sjødronen og forankring til bøyene, ble faglærer Runald Walter Meyer på maskinlinjen kontaktet. Han kom med en stor kasse med metallpinner, se *Figur 56*, og foreslo å legge så mange som mulig i brusflasker. På *Figur 57* er de fylte flaskene avbildet, hvor flasken til venstre er boret hull i, og er ment som forankring til bøyene. Flasken til høyre er ikke boret hull i, og er ment som ballast til sjødronen. Totalt 5 flasker ble fylt, der hver flaske veide rundt 5 kilogram. Tre av disse flaskene er ment for forankring til bøyene og de resterende to ble brukt som ballast til sjødronen. I tillegg ble en gammel elektromotor på cirka 10 kilogram brukt som resterende ballast.

Figur 56 Metallpinne *(privat)*

Figur 57 Forankring til bøyene og balast til sjødronen *(privat)*

Dette medførte at sjødronen totalt veide cirka 50 kilogram, med ballast. Når sjødronen ble sjøsatt, lå den fint i vannet, som kan sees i *Figur 58*. Det ble også testet å ta ut en av flaskene slik at sjødronen veide cirka 45 kilogram, men da ble sjødronen litt ustabil.

Figur 58 Sjøsettingen av sjødronen *(privat)*

6.2 Navigering

Når sjødronen var blitt sjøsatt, ble den autonome navigeringen på sjødronen testet. Dette var på et punkt hvor wp_maker algoritmen ikke var påbegynt enda, men det var ønskelig å se om sjødronen klarte å komme seg til et satt punkt på egenhånd.

Sjødronen ble programmert til å bruke GPS-en og kompasset for å orientere seg, og det ble brukt GPS-koordinater for å sette et veipunkt. Etter noen mislykkede forsøk, der parameterne til regulatoren ble justert, ble konklusjonen å ta opp sjødronen og feilsøke på land.

Det ble konkludert med at kompasset som ble brukt ikke fungerte som det skulle. Mer om dette kan leses om i kapittel 7.4 Utfordring med kompasset på Navio2-kortet.

Navigasjonsalgoritmen har blitt testet flere ganger på land, ved å legge inn et veipunkt og se om trusterne endrer stilling slik at sjødronen svinger mot veipunktet. Disse testene ble utført med IMU-data istedenfor GPS og kompass, da det ikke er GPS signaler inne på campus, og etter at feilen med kompasset ble oppdaget. På grunn av værforholdene har ikke dette blitt testet på vann enda.

6.3 Speed Gate på land

Når sjødronen skulle testes for Speed Gate oppdraget, ble dette først gjort på land. Banen ble satt opp slik *Figur 59* viser, der den røde ballongen ble byttet ut med en person som satt i stolen til venstre, da den røde ballongen hadde sprukket.

Sjødronen ble stroppet fast på en tralle, svingt i retningen trusterne pekte og trillet fremover. På denne måten ble det sjekket om veipunktene ble plassert riktig i forhold til bøyene.

Speed Gate på land ble testet flere ganger, der resultatene varierte. Veipunktene ble plassert riktig i forhold til den "røde" og grønne bøyen, og sjødronen ble dyttet fremover i banen. Det midlertidige veipunktet (wp_{i^*}) ble fulgt helt til sjødronen kunne se den gule bøyen. Når veipunktene rundt den gule bøyen skulle settes ble disse som oftest plassert feil. Dette medfører at det må gjennomføres flere forsøk med justeringer i algoritmene før konkurransen.

Figur 59 Speed Gate testing på land *(privat)*

6.4 Speed Gate på vann

Testing av Speed Gate på vann ble ikke gjennomført før innlevering av bachelorrapporten, grunnet dårlig tid og værforhold. Før en eventuell testing av dette, er det en rekke forberedelser som må gjøres. Dette innebærer å teste at forankringen til bøyene har riktig vekt og anskaffe nok tau for å feste bøyene til forankringen. I tillegg må en ny rød ballong anskaffes, siden den har sprukket. Det må også avklares hvordan bøyene skal settes ut, ettersom gruppen ikke har tilgang til båt og mest sannsynlig må bade for å få plassert og hentet bøyen.

7 Diskusjon

Under bacheloroppgaven har det vært mye nytt å sette seg inn i og gruppen har møtt mange utfordringer. Noen av disse utfordringen var forutsett og andre ikke. Disse utfordringene medførte at gruppen måtte gjøre endringer i den opprinnelige fremdriftsplanen som ble laget i starten av oppgaven.

Gruppen har jobbet jevnt og kontinuerlig med bacheloroppgaven under hele semesteret. Allikevel har de i perioder både ligget foran og bak fremdriftsplanen, da noen oppgaver har tatt lengre tid enn forventet og andre kortere. Den endelige fremdriftsplanen er vedlagt som **Vedlegg 18 – Fremdriftsplan**.

7.1 Utfordring med Python og ROS

Noen av utfordringene som gruppen måtte håndtere i starten av bacheloroppgaven, var å lære seg å skrive Python kode, hvordan ROS skulle brukes og hvordan ROS fungerer generelt. Dette hadde gruppen lite kjennskap til fra før, hvor PhD-kandidatene Gizem Ates og Laurenz Elstner, var til god hjelp.

7.2 Utfordring med minnekortet i NVIDIA Jetson

En av de uforutsette utfordringene gruppen møtte på, var når lagringskapasiteten på minnekortet til NVIDIA Jetson, allerede i starten av oppgaven, nesten var fylt opp. Dette ble oppdaget under installasjonen av YOLO, da det kom varsel om at det ikke var nok plass for å utføre installasjonen. Siden dette var på et tidlig tidspunkt i oppgaven og egne algoritmer ikke var påbegynt enda, ble det bestemt å starte på nytt med et nytt minnekort. Minnekortet i NVIDIA Jetson var originalt på 64GB, som ble oppgradert til 128GB. Å bytte minnekortet innebar at å installere følgende på nytt:

- Det Linux baserte operativsystemet Ubuntu
- ROS
- Alle pakkene tilhørende de forskjellige komponentene (ZED2, RPLidar)

Måten disse og YOLO ble installert, var ved å følge fremgangsmåtene i linkene i **Vedlegg 19 – Installasjons prosedyre**. I tillegg måtte alle algoritmene fra USN som kunne komme til nytte, overføres fra det gamle minnekortet.

7.3 Utfordring med å få YOLO til å fungere

Det ble møtt en del utfordringer i forbindelse med å få YOLO til å fungere. Dette medførte at objekt detekteringsfasen tok vesentlig lengre tid enn forventet på grunn av begrenset kunnskap om ROS og hva YOLO trengte for å installeres. Dette løste seg etter hvert når gruppen ble flinkere til å lese feilmeldingene fra ROS og fant riktig installasjonsprosedyre for YOLO.

7.4 Utfordring med kompasset på Navio2-kortet

Denne utfordringen ble oppdaget i forbindelse med testing av navigeringsalgoritmen, der sjødronen skulle kjøre til en gitt GPS-posisjon. Sjødronen kjørte, men ikke riktig i forhold til GPS-posisjonen, så den ble tatt opp for feilsøking på land. Under feilsøkingen, viste det seg at det var en feil med kompasset på Navio2-kortet, som gjorde at det viste feil og brukte veldig lang tid til å stabilisere seg, selv om sjødronen stod helt stille. Veilederne for prosjektet ble informert om utfordringen og gruppen fikk låne et annet Navio2-kort for å undersøke nærmere. Denne uforutsette utfordringen kom på et sent tidspunkt, og på grunn av prioritering av rapportskriving, kom ikke gruppen fram til en løsning. Det ble konkludert med at dette ikke var et kritisk problem, da IMU-en fra stereokameraet kan brukes istedenfor.

7.5 Sjødronens begrensninger

Tittelen på bacheloroppgaven er «Autonom Sjødrone», selv om sjødronen ikke kan generaliseres som et helautonomt eller selvgående fartøy. Sjødronen vil ikke kunne håndtere alle mulige settinger, da den kun er programmert til å utføre oppdragene knyttet til Autodrone konkurransen.

8 Konklusjon

Under bacheloremnet på høstsemesteret 2021 skulle det dannes grupper og leveres en søknad, som kort beskrev hva bacheloroppgaven gikk ut på.

Gruppen for denne oppgaven skrev her at oppgaven var å:

- Sette seg inn i hvordan sjødronen er satt sammen og fungerer.
- Beskrive det som allerede er gjort hittil, siden sjødronen er laget av USN.
- Få alt utstyret på sjødronen til å kommunisere og jobbe sammen.
- Lage algoritmer som gjør at sjødronen skal kunne utføre Speed Gate oppdraget i Autodrone konkurransen. *Hvis tiden tillater det, vil gruppen gå videre med flere oppdrag til konkurransen.*

Det har vært en veldig stor oppgave, som kanskje har vært større enn forventet i starten. Tilgang til tidligere bacheloroppgaver, med tilsvarende oppgave, har vært til god hjelp. Dette gjelder oppgaver fra HVL *[29]*, NTNU (*[30], [31], [32]*) og USN *[33]*. Under oppgaven har det vært mye nytt å sette seg inn i, lære og forstå, som både har vært gøy og utfordrende. Oppgaven inneholdt mye praktisk og teoretisk arbeid, som til tider har vært krevende å sjonglere.

Gruppen sitter igjen med en god del kunnskaper innenfor ROS og hvordan man lager Python-algoritmer i ROS. Det har også vært mange nye komponenter og algoritmer å bli kjent med, der gruppen måtte finne ut hvordan disse kunne brukes. Etter mye prøving og feiling er resultatet en autonom sjødrone som kan utføre Speed Gate oppdraget i Autodrone konkurransen.

Gruppen er veldig fornøyd med resultatet og mener denne oppgaven danner et godt grunnlag for videreutvikling av sjødronen, for kommende Autodrone konkurranser. I tillegg håper gruppen at oppgaven tas opp igjen av kommende bachelorgrupper.

8.1 Resterende arbeid

Selv om gruppen har jobbet jevnt og kontinuerlig gjennom hele semesteret, ble ikke ambisjonen om å få til flere konkurranseoppdrag oppfylt. Som nevnt tidligere er det totalt fire oppdrag i Autodrone konkurransen, hvor denne bachelorrapporten kun tar for seg det ene oppdraget, Speed Gate, grunnet tid og mulighet.

I midten av bacheloroppgaven fikk gruppen tilsendt én bøye av hver farge, som benyttes i konkurransen. Dette begrenset mulighetene til kun å kunne teste «Speed Gate» og «Dokking» oppdragene, da de andre oppdragene benytter flere bøyer enn gruppen har tilgang til i Haugesund. Under Autodrone konkurransen er det satt av én testedag, der gruppen vil få muligheten til å teste de andre oppdragene.

Bacheloremnet har hatt flere obligatoriske arbeidskrav som forstudierapport, midtveispresentasjon, refleksjonsnotat, EXPO-plakat og muntlig bachelorfremføring. Dette har medført at en del tid har gått med til å fullføre disse.

Gruppen har planer om å jobbe videre med sjødronen etter innlevering av bachelorrapporten, siden denne har innleveringsfrist 30.mai. Autodrone konkurransen holdes i Horten den 12. til 14. juni, som gir gruppen to uker på å gjøre sjødronen klar til Autodrone etter bachelorrapporten er levert.

8.2 Mulig videreutvikling

Gruppen har dannet seg flere tanker underveis, om andre måter enkelte oppgaver kunne blitt løst. I dette delkapittelet fortelles det kort om hvilke videreutviklinger som kan være interessante å undersøke.

8.2.1 Utvide YOLO sitt datasett

Gruppens inntrykk er at jo større datasett man har for å trene YOLO algoritmen, jo mer nøyaktig blir den. En mulig videreutvikling vil være å utvide datasettet, spesielt med bøyebilder, da det er viktig at sjødronen ser forskjell på bøyene.

8.2.2 KOLT (Known Object Localization and Tracking)

Etter lokaliseringsalgoritmen allerede var laget, kom gruppen over algoritmen KOLT *[34]* som ble tolket som en algoritme som tar seg av lokaliseringen av objekter. Gruppen har ikke satt seg inn i hvordan algoritmen fungerer, og det fantes ikke mye informasjon om den på nettet. Algoritmen skal være ROS-vennlig, fungere med ZED-stereokameraet og YOLOv2. På dette stadiet hadde gruppen allerede trent ferdig YOLOv3, som var enda en av grunnene til at denne algoritmen ikke ble undersøkt nærmere.

8.2.3 EKF (Extended Kalman Filter) *[30]*

Extended Kalman Filter er et verktøy som brukes for å forutsi verdier for dynamiske systemer, som for eksempel hos roboter. Den lineariserer først de ikke-lineære funksjonene og deretter følger samme konsept som et vanlig Kalman Filter. Et Kalman Filter blir også kalt for lineært kvadratisk estimering, som er en matematisk ligning som brukes for å beregne midler for innsamlet data innen en tidsfrist for å estimere tilstanden til en prosess. Kalman Filteret gjør estimering med høy nøyaktighet og brukes til å ta spådommer og beslutninger.

8.2.4 SLAM (Simultaneous Localization And Mapping)

SLAM er en ofte brukt metode for å hjelpe roboter med å kartlegge områder, samtidig som den finner sin egen posisjon. Dette er noe gruppen vurderte å ta i bruk, men la fra seg fordi IMU-en i stereokameraet viste riktig posisjon av sjødronen.

9 Referanser

[1] «MARKOM2020,» [Internett]. Available: https://markom2020.no/.

[2] «Definition of waypoint,» Merriam Webster Dictionary, [Internett]. Available: https://www.merriam-webster.com/dictionary/waypoint.

[3] «AutoDrone Rules and Task Descriptions,» [Internett]. Available: https://drive.google.com/file/d/1znsjKdCcPNmfPysDiKEFOY0PmX390RNk/view.

[4] «Raspberry Pi bilde,» RS Components, [Internett]. Available: https://no.rs-online.com/web/p/raspberry-pi/1373331.

[5] «Navio2 HAT bilde,» innovelec, [Internett]. Available: https://innovelec.co.uk/products/emlid-navio2-autopilot-hat-kit/.

[6] «What is IMU? Inertial Measurement Unit Working & Applications,» ARROW, [Internett]. Available: https://www.arrow.com/en/research-and-events/articles/imu-principles-and-applications.

[7] «ZED2 bilde,» StereoLabs, [Internett]. Available: https://www.stereolabs.com/zed-2/.

[8] «Jetson FAQ: What is Jetson?,» NVIDIA Developer, [Internett]. Available: https://developer.nvidia.com/embedded/faq.

[9] «NVIDIA Jetson bilde,» ELFA DISTRELEC, [Internett]. Available: https://www.elfadistrelec.no/no/nvidia-jetson-xavier-nx-utviklingssett-seeed-studio-102110427/p/30185707?trackQuery=NVIDIA+JETSON&pos=8&origPos=8&origPageSize=50&track=true.

[10] «LiDAR bilde,» SLAMTEC, [Internett]. Available: https://www.slamtec.com/en/Lidar/S1.

[11] «Servomotor bilde,» Shantou JiXian Electronic Technology Co., [Internett]. Available: http://www.jx-servo.com/en/Product/SERVO/12Vbs/462.html.

[12] «Truster bilde,» BlueRobotics, [Internett]. Available: https://bluerobotics.com/store/thrusters/t100-t200-thrusters/t200-thruster-r2-rp/.

[13] «Radiokontroller bilde,» EleFun, [Internett]. Available: https://www.elefun.no/p/prod.aspx?v=45591.

[14] «Navio2 ROS-manual,» emlid, [Internett]. Available: https://docs.emlid.com/navio2/ros/.

[15] «ROS (ELE306),» HVL, [Internett]. Available: https://robotics-lab-documentation.readthedocs.io/en/latest/texts/ros0.html.

[16] «ROS,» ROS.org, [Internett]. Available: https://www.ros.org/.

[17] «ZED2 ROS-manual "Getting Started with ROS and ZED",» StereoLabs, [Internett]. Available: https://www.stereolabs.com/docs/ros/.

[18] «LiDAR ROS-manual,» Slamtech github, [Internett]. Available: https://github.com/slamtec/rplidar_ros.

[19] «Creating a ROS msg and srv,» ROS.org, [Internett]. Available: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv.

[20] H. Bandyopadhyay, «"YOLO: Real-Time Object Detection Explained",» v7labs, [Internett]. Available: https://www.v7labs.com/blog/yolo-object-detection.

[21] Manishgupta, «"YOLO - You Only Look Once",» Towards Data Science, [Internett]. Available: https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4.

[22] M. Bjelonic, «YOLO ROS: Real-Time Object Detection for ROS,» Github, [Internett]. Available: https://github.com/leggedrobotics/darknet_ros.

[23] «roboflow,» [Internett]. Available: https://roboflow.com/.

[24] TheAIGuy, «YOLOv3 in the CLOUD: Install and Train Custom Object Detector (FREE GPU),» YouTube, [Internett]. Available: https://www.youtube.com/watch?v=10joRJt39Ns.

[25] «Open Images Dataset V6 + Extentions,» [Internett]. Available: https://storage.googleapis.com/openimages/web/index.html.

[26] TheAIGuy, «Creating a YOLOv3 Custom Dataset | Quick and Easy | 9,000,000+ Images,» YouTube, [Internett]. Available: https://www.youtube.com/watch?v=_4A9inxGqRM.

[27] P. Corke, Robotics, Vision and Control, Second Edition.

[28] «Extracting Yaw from a Quaternion,» StackOverflow, [Internett]. Available: https://stackoverflow.com/questions/5782658/extracting-yaw-from-a-quaternion#:~:text=Having%20given%20a%20Quaternion%20q%2C%20you%20can%20calculate%20roll%2C%20pitch%20and%20yaw%20like%20this%3A.

[29] J. Stavland og S. D. Raa, «Algoritmer og sensorer som en drone trenger for å delta i AutoDrone 2020,» Høgskulen på Vestlandet - Nautikk, Haugesund, 2021.

[30] M. Stava, M. G. Gaasholt og K. H. San, «Developing waypoint navigation and buoy detection using YOLO for an autonomous surface vessel,» NTNU, Ålesund, 2021.

[31] I. H. Øksne, S. Odden, M. Aanning og M. Holt, «Graph-based slam and navigation in a simulated environment and buoy classification using YOLO for an Autonomous Surface Vessel,» NTNU, Ålesund, 2020.

[32] S. V. Grini, «Object Detection in Maritime Environments,» NTNU, Trondheim, 2019.

[33] T. T. Olsen og T. Falleth, «Autonome fartøy og navigering,» USN, Horten, 2021.

[34] N. Rosenberg, «KOLT_ROS,» GitHub, [Internett]. Available: https://github.com/diggerdata/kolt_ros.

10 Vedlegg

10.1 Vedlegg 1 – Modus.msg

```
bool auto

int16 mode

bool gps
```

10.2 Vedlegg 2 – ObjectPosition.msg

```
string klasse

float32[] X

float32[] Y

int16[] probability
```

10.3 Vedlegg 3 –Waypoint.msg

```
float64 radius

float64 x

float64 y
```

10.4 Vedlegg 4 – Auto_Mode.msg

```
bool mode

float64 channel
```

10.5 Vedlegg 5 – Motor.msg

```
float64 x

int32 dir
```

10.6 Vedlegg 6 –Sea_Twist.msg

```
Twist right_thruster

Twist left_thruster

bool auto_mode
```

10.7 Vedlegg 7 – Servo.msg

```
float64 z
```

10.8 Vedlegg 8 – Twist.msg

```
Motor linear

Servo angular
```

10.9 Vedlegg 9 – Autodrone_Startbeskrivelse_HVL

Either connect via wifi "AutoDrone_HVL"(password [redacted]) or Ethernet (DHCP or set 192.168.2.80)

Propulsion & Steering Start up (Navio2/Raspberry pi) 192.168.2.49:

Connect by ssh to pi@192.168.2.49 passord: [redacted] (use the ssh client MobaXterm)

To start the propulsion and steering, you need to start the roscore and mavros node.

To do this type: "*cd catkin_ws/bin*"

in the command shell. This is where the ROS start script is located.

Then type: "*./ros_start.sh*"

this will start the roscore and the mavros node.

NOTE: If ros_start.sh show an Error or cannot connect, retry "*./ros_start.sh*".

When roscore and mavros node are running, you can start the propulsion and steering launcher.

Type: "*roslaunch controller cmd_vel_controller.launch*"

NOTE: If changes are made remember to change the code in Servo.py, Thruster.py & controller.py accordingly. If not, the launcher will not work.

Note2: Turn on the RC controller. The Long toggle switch (top left) will switch between Auto-mode and RC control mode. For RC control: the right joystick Left/Right will turn the thrusters left/right, joystic Forward/Back will run the thrusters forward or backwards with the speed set by the left joystick.

NB: Be careful not to run the thrusters at high speed in air – they can be destroyed)

Navigation System (NVIDIA Jetson Xavier) 192.168.2.79:

Ssh autodrone@192.168.2.79 passord: [redacted] (use MobaXterm)

NOTE: The master must start Roscore first, in this case the Navio2/Raspberry pi is the master. The Propulsion and steering must run for some of the Navigation system programs.

To start Propulsion and Steering see README in Navio2/Raspberry pi Catkin_ws/src.

To start the Navigation system, you will need to be in the "*home/autodrone*" path.

Then type: "*./Start.sh*"

this will start the Zed2 camera, Web_server, the waypoint list, color tracker, the waypoint maker, Lidar. The GPS functions must be updated by HVL to use the Navio2 GPS.

To move to different scripts in the Start.sh script, in MobaXterm press down "Ctrl+B" and then "N" for next folder or an number (0-7) for a specific script folder.

NOTE: If any of the programs get an error or auto shutdown, retry the program script, NOT retry "./Start.sh".

If everything runs then start the Navigation.py script, located in the Main package.

In a new shell type: "*cd catkin_ws/src/main/src*"

then type: "*python navigation.py*"

if everything went well there should be some error saying that "contr_aut is not defined" or "mode is not defined".

If that is the case, then you need to define a mode in the "Tracking/BoatMode" topic where the number behind mode tells which mode the program is set to. If contr_aut is not defined then the propulsion and steering system is not running.

mode 0 = get to waypoint

mode 1 = Speed gate

mode 2 = collision avoidance

mode 3 = obstacle gate

mode 4 = docking

In a new shell type: "*rostopic pub /Tracking/BoatMode main/BoatMode "mode: 0"* "

Next you need to set the controller mode in the propulsion and steering system to automatic.

10.10 Vedlegg 10 – controller_radio.py

```
#! /usr/bin/env python
#-------------------------------------------------------------------#
#       Radiocontroller connection and scaling to percentage        #
#       for controlling servos and thrusters                        #
#       Produced by: Tobias Tufte, Torben Falleth Olsen             #
#-------------------------------------------------------------------#

import rospy
from controller.msg import Sea_Twist, Auto_Mode
import time

channel = [0, 0, 0, 0, 0]

# Channel 0 = Right stick down/up
# Channel 1 = Right stick left/right
# Channel 2 = Left stick down/up
# Channel 3 = Left stick left/right
# Channel 4 = SF-Switch down/up

#Channel input value from controller is aprox between ~982 -
> ~2006 and the middle at ~1495.

pub = rospy.Publisher('cmd_vel', Sea_Twist, queue_size=1)              #
 Publishing to Servo and thursters
pub_auto = rospy.Publisher('control_auto', Auto_Mode, queue_size=1)    #
 Publishing boolean for what mode the controller is in, auto or manual

def controller_in():                                                   #
 Radio input and scaling
    for i in range(0, 5):
      string = '/sys/kernel/rcio/rcin/ch' + str(i)
      ch_data_raw = open(string)
      channel[i] = ((float(ch_data_raw.read()) - 1000.0)*0.1)          #
 Converts input range to aprox 0-100
#      print ("Channel: " , i, channel[i])
      if channel[i] > 98:                                              #
 Fixes inaccurate max value
        channel[i] = 100
      if channel[i] < 2:                                               #
 Fixes inaccurate min value
        channel[i] = 0
      if (channel[i] < 51 and channel[i] > 49):                        #
 Fixes inacurate mid value
        channel[i] = 50
      if (channel[4] < 25):                                            #
 Converts SF-Switch to bool
        channel[4] = True
```

```
      else:
        channel[4] = False
#      print ("Channel: " , i, channel[i])
#      print "---------------------- "

def publishing_data():                                                  #
Publishing the data from the channels to the motors/thrusters

  data_out = Sea_Twist()
  auto = Auto_Mode()
  if (channel[4]):                                                      #
If channel 4 is false then publish the data
#    print(" Controller on")
    auto.mode = False
    pub_auto.publish(auto)

    # Links controller to Sea_Twist.msg for controlling the servos and
thrusters manually.
    data_out.right_thruster.linear.x = channel[2]
    data_out.left_thruster.linear.x = channel[2]
    data_out.right_thruster.angular.z = channel[1]
    data_out.left_thruster.angular.z = channel[1]

    # Defines direction of thrusters
    if channel[0] <= 45:                                                # Makes
 the thrusters go in reverse
      data_out.left_thruster.linear.dir = 2
      data_out.right_thruster.linear.dir = 2
    elif channel[0] >= 55:                                              # Makes
 the thrusters go forward
      data_out.left_thruster.linear.dir = 1
      data_out.right_thruster.linear.dir = 1
    else:
      data_out.left_thruster.linear.dir = 0
      data_out.right_thruster.linear.dir = 0

    data_out.auto_mode = channel[4]
    pub.publish(data_out)

  if not (channel[4]):                                                  # If co
ntroller is off, set auto mode true.
#    print("Controller off")
    auto.mode = True
    pub_auto.publish(auto)


def run():
    rospy.init_node('controller_node', anonymous = True)
```

```
    while not rospy.is_shutdown():
        controller_in()
        publishing_data()

if __name__ == '__main__':
    try:
        run()
    except rospy.ROSInterruptException:
        pass
```

10.11 Vedlegg 11 – servo.py

```
#!/usr/bin/env python
#-------------------------------------------------------------------#
#       Control movement of servo                                   #
#       Produced by: Tobias Tufte and Torben Falleth Olsen          #
#-------------------------------------------------------------------#
import sys
import time
import rospy
import navio.pwm
import datetime
from controller.msg import Sea_Twist

PWM_OUTPUT_right = 5
 # Identifying witch PWM pin that's in use on the navio2 and enabling i
t
pwm_right = navio.pwm.PWM(PWM_OUTPUT_right)
pwm_right.initialize()
pwm_right.set_period(50)
pwm_right.enable()

PWM_OUTPUT_left = 1
 # Identifying witch PWM pin that's in use on the navio2 and enabling i
t
pwm_left = navio.pwm.PWM(PWM_OUTPUT_left)
pwm_left.initialize()
pwm_left.set_period(50)
pwm_left.enable()

var_right = 1.5
 # 0.5-2.5ms = 0-180 degrees, 1.5ms = 90 degrees
var_left = 1.5


def convert(msg):
    global var_right
    global var_left

    var_right = -
2.0*(msg.right_thruster.angular.z/100)+2.5              # converts from
percent to 0.5-2.5ms pwm
    var_left = -
2.0*(msg.left_thruster.angular.z/100)+2.5               # converts from
 percent to 0.5-2.5ms pwm

def run():
    rospy.init_node('Both_servos', anonymous = True)
```

```
    sub = rospy.Subscriber('/cmd_vel', Sea_Twist, convert)
 # Subscribes on the /cmd_vel topic


if __name__ == '__main__':

    run()
    while not rospy.is_shutdown():
        rate = rospy.Rate(500)
  # Publishing the PWM signal in the correct frequency
        pwm_right.set_duty_cycle(var_right)
  # Publishing the PWM signal to the right servo

        pwm_left.set_duty_cycle(var_left)
  # Publishing the PWM signal to the left servo
        rate.sleep()
```

10.12 Vedlegg 12 – thruster.py

```
#!/usr/bin/env python
#-------------------------------------------------------------------#
#       Control of thruster                                         #
#       Produced by: Tobias Tufte and Torben Falleth Olsen          #
#-------------------------------------------------------------------#
import sys
import time
import rospy
import navio.pwm
import datetime
from controller.msg import Sea_Twist

PWM_OUTPUT_left = 3
    # Identifying witch PWM pin that's in use on the navio2 and enablin
g it
pwm_left = navio.pwm.PWM(PWM_OUTPUT_left)
pwm_left.initialize()
pwm_left.set_period(50)
pwm_left.enable()

var = 1.5
    # 1.1-1.9ms = -
100 too 100%, 1.5ms = Stop and Initialiation of thrusters.
var1 = 1.5

PWM_OUTPUT_right = 4
    # Identifying witch PWM pin that's in use on the navio2 and enablin
g it
pwm_right = navio.pwm.PWM(PWM_OUTPUT_right)
pwm_right.initialize()
pwm_right.set_period(50)
pwm_right.enable()

def convert(msg):
  global var
  if msg.right_thruster.linear.dir == 0:
    # STOP
    var = 1.5
    # From Prosent to PWM

  if msg.right_thruster.linear.dir == 1:
    # Forward
    var = 0.4*(msg.right_thruster.linear.x/100)+1.5
    # From Prosent to PWM

  if msg.right_thruster.linear.dir == 2:
    # Reverse
```

```
    var = 0.4*(msg.right_thruster.linear.x/-
100)+1.5                        # From Prosent to PWM

  global var1
  if msg.left_thruster.linear.dir == 0:
    # STOP
    var1 = 1.5
    # From Prosent to PWM

  if msg.left_thruster.linear.dir == 1:
    # Forward
    var1 = 0.4*(msg.left_thruster.linear.x/100)+1.5
    # From Prosent to PWM


  if msg.left_thruster.linear.dir == 2:
    # Reverse
    var1 = 0.4*(msg.left_thruster.linear.x/-
100)+1.5                        # From Prosent to PWM

def run():
  rospy.init_node('thruster', anonymous = True)
  sub = rospy.Subscriber('/cmd_vel', Sea_Twist, convert)
    # Subcribing on topic /cmd_vel


if __name__ == '__main__':
    run()
    while not rospy.is_shutdown():
        rate = rospy.Rate(200)
    # Publishing frequency
        pwm_right.set_duty_cycle(var)
    # Publishing to the right motor
        pwm_left.set_duty_cycle(var1)
    # Publishing to the left motor
        rate.sleep()
```

10.13 Vedlegg 13 – com-LED.py

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

import serial, time
import rospy
from controller.msg import Auto_Mode, Sea_Twist

run_mode = False
s_a_r = 50.0
s_a_l = 50.0
t_l_r = 0
t_l_l = 0
t_x_r = 0.0
t_x_l = 0.0

ser = serial.Serial()
ser.port = "/dev/ttyUSB0"  #"/dev/ttyLED"

ser.baudrate = 9600
try:
    ser.open()
except Exception, e:
    print "error open serial port: " + str(e)
    exit()

def main():
    global s_a_r
    global s_a_l
    global t_l_r
    global t_l_l
    global t_x_r
    global t_x_l
    global run_mode
    #print run_mode

    if ser.isOpen():

        try:
            ser.flushInput() #flush input buffer, discarding all its co
ntents
            ser.flushOutput()#flush output buffer, aborting current out
put
                    #and discard all that is in buffer
            #print "serie port:"(ser.isOpen())
            if run_mode == True:
                ser.write("B")
                #print("Blue LED")
```

```
            elif (run_mode == False and (((not(t_l_l==0) and not(t_l_r==0)) or (not(s_a_r==50.0)and not(s_a_l==50.0))) or (not(t_x_r==0.0) or not(t_x_l==0.0)))):
                ser.write("Y")
                #print ("Yellow Led")
            else:
                ser.write("R")
                #print ("Red")

            #ser.close()
        except Exception, e1:
            print "error communicating...: " + str(e1)

    else:
        print "cannot open serial port "

def BoatMode(msg):
    global run_mode
    run_mode = msg.mode

def cmd_vel(data):
    global s_a_r
    global s_a_l
    global t_l_r
    global t_l_l
    global t_x_r
    global t_x_l
    s_a_r = data.right_thruster.angular.z
    s_a_l = data.left_thruster.angular.z
    t_l_l = data.left_thruster.linear.dir
    t_l_r = data.right_thruster.linear.dir
    t_x_l = data.left_thruster.linear.x
    t_x_r = data.right_thruster.linear.x

#def sub_s

if __name__ == '__main__':
    rospy.init_node('led_light_node', anonymous = True)
    sub = rospy.Subscriber('/control_auto', Auto_Mode, BoatMode)
    sub_twist = rospy.Subscriber('/cmd_vel', Sea_Twist, cmd_vel)
    while not rospy.is_shutdown():
        main()
    ser.close()
```

10.14 Vedlegg 14 – mode_selector.py

```
#!/usr/bin/env python
#-*-coding: utf_8-*-
#
#   This program control what mission the seadrone is set to, by passin
g through input
#   from the operator
#
#   Made: spring semester 2022
#   By: Magnus Eide & Linn Jeanette Myhren
#

import rospy
import math as m
from controller.msg import Auto_Mode
from std_msgs.msg import Int16, Bool
from hvl_drone.msg import Modus
import os

pub = rospy.Publisher('modus', Modus, queue_size = 10)

class modus():
    auto = False
    mode = 0

    def __init__(self, auto, mode):
        self.auto = auto
        self.mode = mode

    def __init__(self):
        pass

def callbackAuto(Auto):
    if Auto.mode:
        mo.auto = True
    else:
        mo.auto = False

    data_out = Modus()
    data_out.auto = mo.auto
    data_out.mode = mo.mode
    data_out.gps = mo.gps
    print("----")
    print("Automode: {}\nModus:    {}".format(data_out.auto, data_out.m
ode))
    pub.publish(data_out)

def callbackMode(Data):
```

```
    mo.mode = Data.data

def callbackGPS(Data):
    mo.gps = Data.data

def main():
    rospy.init_node('mode_selector')

    rospy.Subscriber("/control_auto", Auto_Mode, callbackAuto)
    rospy.Subscriber("/automode", Int16, callbackMode)
    rospy.Subscriber("/UseGPS", Bool, callbackGPS)

    rospy.spin()

if __name__ == '__main__':
    mo = Modus()
    main()
```

10.15 Vedlegg 15 – localizer.py

```
#!/usr/bin/env python
#-*-coding: utf_8-*-
#
#   This program takes data from the ZED camera and sends an image to t
he YOLO algorithm.
#   Afterwards it uses the result to localise the objects in world coor
dinates
#
#   Made: spring semester 2022
#   By: Magnus Eide & Linn Jeanette Myhren
#

import rospy
import math as m
import time
from sensor_msgs.msg import Image
from geometry_msgs.msg import PoseStamped
from darknet_ros_msgs.msg import BoundingBoxes
from hvl_drone.msg import ObjectPosition
import message_filters
import cv2
from cv_bridge import CvBridge
import os

wanted = ["RedBuoy", "GreenBuoy", "YellowBuoy", "Person"] #Objects we w
ant to localize
Dist = 3                # meters between two registered buoys of the sa
me color
midPoint = False
Delay = 0.6

YellowPub = rospy.Publisher("/YellowBuoys", ObjectPosition, queue_size=
1)
GreenPub = rospy.Publisher("/GreenBuoys", ObjectPosition, queue_size=1)
RedPub = rospy.Publisher("/RedBuoys", ObjectPosition, queue_size=1)
imagePub = rospy.Publisher("/ColorImage", Image, queue_size=1)

class Verifier():
    def __init__(self):
        self.complete = True
        self.first = True

class Timer():
    def __init__(self, currTime, nextTime):
        self.Current = currTime
        self.Next = nextTime
```

```
class Data():
    def __init__(self):
        self.depth = Image()
        self.pose = PoseStamped()
        self.image = Image()
        self.box = BoundingBoxes()

class Object():
    def __init__(self, Class, prob, x, y):
        self.Class = Class
        self.Prob = prob
        self.X = x
        self.Y = y

    def __init__(self):
        pass

class Distance():
    def __init__(self, depth, x, y):
        self.depth = depth
        self.x = x
        self.y = y

    def __init__(self):
        pass

def callbackZed(depth, image, pose):
    T.Current = time.time()
    if T.Current >= T.Next:
        T.Next = T.Current + Delay
        C.complete = True
    if C.complete:
        imagePub.publish(image)
        D.image = image
        D.depth = depth
        D.pose = pose
        C.complete = False

def callbackBox(data):
    if (D.pose.header.stamp.nsecs == D.image.header.stamp.nsecs) and (D
.pose.header.stamp.nsecs == D.depth.header.stamp.nsecs):
        if C.complete == False:
            D.Box = data
            T.Next = time.time() + Delay
            callback(D.Box, D.depth, D.pose)
            T.Next = time.time() + Delay
            C.complete = True
    else:
        C.complete = True
```

```
        T.Next = time.time() + Delay

def callback(Box, Depth, pose):
    if True:
        camY = pose.pose.position.y
        camX = pose.pose.position.x
        q = [pose.pose.orientation.x, pose.pose.orientation.y, pose.pos
e.orientation.z, pose.pose.orientation.w]   # x y z w orientation (quat
ernion)
        # Convert quaternions to radians rotated around Z-axis
        heading = m.atan2(2.0*(q[0]*q[1] + q[3]*q[2]), q[3]*q[3] + q[0]
*q[0] - q[1]*q[1] - q[2]*q[2] )

        if heading < 0:
            heading += 2.0*m.pi

        for box in Box.bounding_boxes:
            if box.Class in wanted:
                os.system("clear")
                Xc = (box.xmax-box.xmin)/2 + box.xmin
                Yc = (box.ymax+box.ymin)/2 + box.ymin
                bridge = CvBridge()
                cv_image = bridge.imgmsg_to_cv2(Depth, "passthrough")

                MyDepth = Distance()
                                                    # Use the mid
dle of the box to calculate and position
                MyDepth.depth = cv_image[Yc/2, Xc]
                #MyDepth.depth = cv_image[Yc/2, Xc]
                MyDepth.x = Xc
                MyDepth.y = Yc

                if midPoint == False:                   # Use the nea
rest point within the box
                    for x in range(box.xmin, box.xmax+1):
                        for y in range(box.ymin, box.ymax+1):
                            if MyDepth.depth > cv_image[y/2, x]:
                                MyDepth.depth = cv_image[y/2, x]
                                MyDepth.x = x
                                MyDepth.y = y

                # MyDepth.depth += 0.3                      # measure t
o the centre of the buoye

                if (MyDepth.depth > 0):
                    deg = -MyDepth.x*120.0/cv_image.shape[1] + 60.0
                    X = MyDepth.depth*m.cos(m.radians(deg))           #
 distance
```

```
                    Y = MyDepth.depth*m.sin(m.radians(deg)) - 0.06    #
 horizontal position compensating for displacement of camera lens

                    # print "{}, {}".format(X, Y) # position from boat
                    # Transform position according to the world coordin
atesystem
                    # Equations is given by the transformationmatrix
                    X = (m.cos(heading)*X-m.sin(heading)*Y+camX)
                    Y = (m.sin(heading)*X+m.cos(heading)*Y+camY)
                    # print "{}, {}".format(X, Y) # position in world c
oordinates
                    # print heading
                    Ob = Object()
                    Ob.Class = box.Class
                    Ob.Prob = box.probability*100.0
                    Ob.X = X
                    Ob.Y = Y

                    if box.Class == "YellowBuoy":
                        addToList(yellowList, Ob)
                    elif box.Class == "RedBuoy":
                        addToList(redList, Ob)
                    elif box.Class == "GreenBuoy":
                        addToList(greenList, Ob)

        clean(redList)
        clean(greenList)
        clean(yellowList)
        printList(yellowList)
        printList(greenList)
        printList(redList)

    YellowPub.publish(yellowList)
    GreenPub.publish(greenList)
    RedPub.publish(redList)

def addToList(theList, object):
    if len(theList.X) == 0:
        append(object, theList)
    else:
        numbrero = 0
        minDist = 1000
        for i in range(0, len(theList.Y)):
            distance = findDistance(object.X, object.Y, theList.X[i], t
heList.Y[i])
            if distance < minDist:
                minDist = distance
                numbrero = i
        if distance > Dist:
```

```
            append(object, theList)
        else:
            filter(object, theList, numbrero)


def append(Obj, List):
    List.X.append(Obj.X)
    List.Y.append(Obj.Y)
    List.probability.append(Obj.Prob * 0.1)


def findDistance(x0, y0, x1, y1):
    distance = m.sqrt((y1-y0)**2 + (x1-x0)**2)
    return distance


def filter(Obj, List, i):
    List.X[i] = (List.X[i] + Obj.X) / 2
    List.Y[i] = (List.Y[i] + Obj.Y) / 2
    List.probability[i] = (List.probability[i] + Obj.Prob*0.005)
    if List.probability[i] >= 100: List.probability[i] = 100


def clean(list):
    for b1 in range(0, len(list.X)):
        bx1 = list.X[b1]
        by1 = list.Y[b1]
        for b2 in range(b1 + 1, len(list.X)):
            bx2 = list.X[b2]
            by2 = list.Y[b2]
            D = findDistance(bx1, by1, bx2, by2)
            if D < 2:
                list.X[b1] = (bx1 + bx2) / 2
                list.Y[b1] = (by1 + by2) / 2
                list.probability[b1] = max(list.probability[b1], list.p
robability[b2])
                del list.X[b2]
                del list.Y[b2]
                del list.probability[b2]
                return


def printList(List):
    print "{}:".format(List.klasse)
    for i in range(0,len(List.X)):
        print("({:.3f} {:.3f})     {:.0f}%".format(List.X[i], List.Y[i]
, List.probability[i]))
```

```
def main():
    rospy.init_node('localizer')

    global C, D, T
    C = Verifier()
    C.complete = True
    D = Data()
    T = Timer(float(time.time()), float(time.time() + Delay))

    global redList, greenList, yellowList
    redList = ObjectPosition()
    redList.klasse = "RedBuoy"
    greenList = ObjectPosition()
    greenList.klasse = "GreenBuoy"
    yellowList = ObjectPosition()
    yellowList.klasse = "YellowBuoy"

    subDepth = message_filters.Subscriber('/zed2/zed_node/depth/depth_r
egistered', Image)
    subPosition = message_filters.Subscriber('/zed2/zed_node/pose', Pos
eStamped)
    subImage = message_filters.Subscriber('/zed2/zed_node/rgb_raw/image
_raw_color', Image)

    rospy.Subscriber('/darknet_ros/bounding_boxes', BoundingBoxes, call
backBox)
    ts = message_filters.TimeSynchronizer([subDepth, subImage, subPosit
ion], 10)
    ts.registerCallback(callbackZed)
    rospy.spin()

if __name__ == '__main__':
    main()
```

10.16 Vedlegg 16 – wp_maker.py

```
#!/usr/bin/env python
#-*-coding: utf_8-*-
#
#   This program creates the path the seadrone will follow to complete
the missions in Autodrone
#
#
#   Made: spring semester 2022
#   By: Magnus Eide & Linn Jeanette Myhren
#

import rospy
import math as m
import time
from sensor_msgs.msg import NavSatFix
from geometry_msgs.msg import PoseStamped
from hvl_drone.msg import Modus, ObjectPosition, Waypoint
from std_srvs.srv import Empty

pub_wp = rospy.Publisher("/waypoint", Waypoint, queue_size=1)

class Buoy():
    def __init__(self, x, y):
        self.X = x
        self.Y = y

    def __init__(self):
        pass

class Timer():
    def __init__(self, curr, next):
        self.Current = curr
        self.Next = next

class Track():
    def __init__(self):
        self.Finished = False
        self.Distance = 1000
        self.index = 0
        self.currWP = Waypoint()

class SpeedGate():
    def __init__(self, rx, ry, gx, gy, yx, yy, th, list, co):
        self.Rx = rx
        self.Ry = ry
        self.Gx = gx
        self.Gy = gy
```

```
        self.Yx = yx
        self.Yy = yy
        self.theta = th
        self.first = False
        self.wps = list
        self.counter = co

    def __init__(self):
        self.first = False

def callbackModus(Modus):
    T.Current = time.time()
    if (Modus.auto == True):  #Seadrone in auto
        F.Distance = m.sqrt((pos.Y - F.currWP.y)**2 + (pos.X-
F.currWP.x)**2)
        if (F.Distance < F.currWP.radius):
            F.index += 1

#       Go to startpoint
#**********************************************************************
*
        if Modus.mode==0:
            if T.Current >= T.Next:
                print("Autonom, Modus: 0")



#       Speed Gate
#**********************************************************************
*
        elif Modus.mode==1: #Speed Gate

            if T.Current >= T.Next:
                print("Autonom, Modus: 1")

            if (F.index >= len(SG.wps)):
                F.index -= 1

            F.currWP = Waypoint()
            F.currWP = SG.wps[F.index]

            try:
                redandgreen = (len(redList.X) > 0) and (len(greenList.X
) > 0)
            except:
                redandgreen = False

            if (redandgreen == True): #if the seadrone has seen one gre
en and one red buoy
```

```
                SG.Rx = redList.X[0]
                SG.Gx = greenList.X[0]
                SG.Ry = redList.Y[0]
                SG.Gy = greenList.Y[0]

                # Find the middle between the buoys
                wpX = SG.Rx+((SG.Gx-SG.Rx)/2)
                wpY = SG.Ry+((SG.Gy-SG.Ry)/2)

                # Make waypoint
                wp0.y = wpY
                wp0.x = wpX
                wp0.radius = 1
                SG.wps[0] = wp0
                if SG.first == False:
                    wp1 = wp0
                    SG.wps.append(wp1)
                    SG.first = True

                # Move waypoint if yellow buoy is not detected
            if (True):
                if (len(yellowList.X) == 0) and SG.first:
                    wp1 = Waypoint()
                    wp1.radius = 1
                    if F.Distance <= F.currWP.radius:
                        F.index = 1
                        F.currWP = SG.wps[F.index]

                    # When within radius
                    if F.Distance <= wp1.radius:
                        if SG.Rx < SG.Gx:
                            SG.theta = 3 * m.pi / 2 + m.atan((SG.Gy-
SG.Ry)/(SG.Gx-SG.Rx))

                        else:
                            SG.theta = m.pi / 2 + m.atan((SG.Gy-
SG.Ry)/(SG.Gx-SG.Rx))


                        wp1.x = wp0.x + SG.counter * m.cos(SG.theta)
                        wp1.y = wp0.y + SG.counter * m.sin(SG.theta)

                        SG.wps[1] = wp1
                        SG.counter += 1
                        F.index = 1
                        F.currWP = SG.wps[F.index]
```

```
                    # Set waypoints around the yellow buoy
                elif (len(yellowList.X) > 0) and SG.first and not F.Fin
ished:
                    SG.Yx = yellowList.X[0]
                    SG.Yy = yellowList.Y[0]

                    # Finn ny theta fra start til gul bøye
                    if wp0.x < SG.Yx:
                        SG.theta = m.atan((wp0.y - SG.Yy) / (wp0.x - SG
.Yx))
                    else:
                        SG.theta = m.pi + m.atan((wp0.y - SG.Yy) / (wp0
.x - SG.Yx))

                    # sett høyre waypoint vinkelrett på theta
                    wp2 = Waypoint()
                    wp2.x = SG.Yx + rad*m.cos(m.pi/2-SG.theta)
                    wp2.y = SG.Yy - rad*m.sin(m.pi/2-SG.theta)
                    wp2.radius = rad
                    SG.wps.append(wp2)

                    # sett midterste waypoint x meter bak bøyen
                    wp3 = Waypoint()
                    wp3.x = SG.Yx + rad * m.sin(SG.theta)
                    wp3.y = SG.Yy + rad * m.cos(SG.theta)
                    wp3.radius = rad

                    SG.wps[3] = wp3

                    # sett venstre waypoint vinkelrett på theta
                    wp4 = Waypoint()
                    wp4.x = SG.Yx - rad * m.cos(m.pi/2 - SG.theta)
                    wp4.y = SG.Yy + rad * m.sin(m.pi/2 - SG.theta)
                    wp4.radius = rad

                    SG.wps[4] = wp4

                    # sett siste waypoint (samme som første)
                    wp5 = wp0
                    SG.wps[5] = wp5

                    # Just to make it stop the thrusters

                    F.Finished = True

            # Troubleshoot print
            if T.Current >= T.Next:
                print redandgreen
                print SG.first
```

```
                print F.Finished
                print "Counter: {}".format(SG.counter)
                print "Distance: {}".format(F.Distance)
                print F.currWP
                # print "Theta: {}".format(SG.theta)


#       Obstacle channel
#*********************************************************************
*
        elif Modus.mode==2: #Obstacle channel
            if T.Current >= T.Next:
                print("Autonom, Modus: 2")


#       Collision avoidance
#*********************************************************************
*
        elif Modus.mode==3: #Collision avoidance
            if T.Current >= T.Next: print("Autonom, Modus: 3")


#       Docking
#*********************************************************************
*
        elif Modus.mode==4: #Docking
            if T.Current >= T.Next: print("Autonom, Modus: 4")



    else:
        if T.Current >= T.Next: print("Remote controlled")


    if T.Current >= T.Next:
            T.Next += 1

    publish(F.currWP)

def publish(waypoint):
    try:
        pub_wp.publish(waypoint)
    except:
        print "Publishing waypoint failed!"

def callbackYellow(List):
    yellowList.X = List.X
    yellowList.Y = List.Y
```

```
def callbackRed(List):
    redList.X = List.X
    redList.Y = List.Y

def callbackGreen(List):
    greenList.X = List.X
    greenList.Y = List.Y

def callbackZed(pose):
    pos.X = pose.pose.position.x
    pos.Y = pose.pose.position.y

def main():
    rospy.init_node('wp_maker')
    global redList, greenList, yellowList, pos, init, T, F
    redList = Buoy()
    greenList = Buoy()
    yellowList = Buoy()

    pos = Buoy()
    F = Track()
    F.currWP = Waypoint()
    F.currWP.x = 0
    F.currWP.y = 0
    F.currWP.radius = 0
    T = Timer(time.time(), time.time()+1)

    # For SpeedGate
    global SG, wp0, wp1, rad
    SG = SpeedGate()
    wp0 = Waypoint()
    wp1 = Waypoint()
    SG.wps = [F.currWP]
    SG.counter = 1
    rad = 1

    rospy.Subscriber("/YellowBuoys", ObjectPosition, callbackYellow)
    rospy.Subscriber("/RedBuoys", ObjectPosition, callbackRed)
    rospy.Subscriber("/GreenBuoys", ObjectPosition, callbackGreen)
    rospy.Subscriber("/modus", Modus, callbackModus)
    rospy.Subscriber("/zed2/zed_node/pose", PoseStamped, callbackZed)

    rospy.spin()

if __name__ == '__main__':
    main()
```

10.17 Vedlegg 17 – navigation.py

```
#!/usr/bin/env python
#-*-coding: utf_8-*-
#
#   This program is navigating the drone to a waypoint
#   defined by the topic: /waypoint/current
#   And requests a new waypoint when reached
#
#
#   Made: spring semester 2022
#   By: Magnus Eide & Linn Jeanette Myhren
#

import rospy
import math as m
import time
from std_msgs.msg import Float64
from std_srvs.srv import Empty
from sensor_msgs.msg import NavSatFix, LaserScan
from geometry_msgs.msg import PoseStamped
from hvl_drone.msg import Modus, ObjectPosition, Waypoint
from controller.msg import Sea_Twist

SeaPub = rospy.Publisher('cmd_vel', Sea_Twist, queue_size = 10)

# Parameters
collisionDistance = 4 # meter
maxSpeed = 10
Pparam = 1.8
Iparam = 0.02

class Regulator():
    def __init__(self, wplat, wplon, wprad, polat, polon, head, dist, i
ndex):
        self.wpX = wplat      # waypoint latitude
        self.wpY = wplon      # waypoint longitude
        self.wprad = wprad      # waypoint radius
        self.wpIndex = index
        self.poX = polat      # current position latitude
        self.poY = polon      # current position longitude
        self.head  = head       # heading of vessel (degrees)
        self.dist = dist        # distance to next waypoint

    def __init__(self):
        pass

class Driving():
    def __init__(self, rate, angle, mdir, speed):
```

```
        self.Mdir = mdir        # Direction of trusters 0=none 1=forwar
d 2=backward
        self.rate = rate        # Steering direction 0=none 1=right 2=l
eft
        self.angle = angle      # Angle of thrusters [0-100] -> +-
90 deg
        self.speed = speed      # Speed of thrusters [0-100]

class Collision():
    def __init__(self, Ldistance, Langle, Rdistance, Rangle):
        self.leftDist = Ldistance
        self.leftAng = Langle
        self.rightDist = Rdistance
        self.rightAng = Rangle

    def __init__(self):
        pass

class Positioning():
    def __init__(self, mode):
        self.Mode = mode

class Timer():
    def __init__(self, currTime, nextTime):
        self.Current = currTime
        self.Next = nextTime

# Callback functions #################

def waypointCallback(wp):
    new = True
    if reg.wpX == wp.x and reg.wpY == wp.y:
        new = False
    else:
        reg.wpX = wp.x
        reg.wpY = wp.y
        reg.wprad = wp.radius

# set position and heading according to the ZED pose
def zedPoseCallback(pose):
    if gps.Mode == False:
        reg.poX= pose.pose.position.x
        reg.poY = pose.pose.position.y
        q = [pose.pose.orientation.x, pose.pose.orientation.y, pose.pos
e.orientation.z, pose.pose.orientation.w]  # x y z w orientation (quate
rnion)
        yaw = m.atan2(2.0*(q[0]*q[1] + q[3]*q[2]), q[3]*q[3] + q[0]*q[0
] - q[1]*q[1] - q[2]*q[2])                  # Rotation about Z axis
```

```
        heading = yaw*180.0/m.pi
        if heading < 0:
            heading += 360.0
        reg.head = heading

        # Find distance to current waypoint
        if gps.Mode:
            r = 6378.137  # Radius of the earth in km
            d_X = m.pi / 180.0 * (reg.wpX - reg.poX)
            d_Y = m.pi / 180.0 * (reg.wpY - reg.poY)
            a = m.sin(d_X / 2) ** 2 + m.cos(reg.poX * m.pi / 180.0) * m
.cos(reg.wpX * m.pi / 180.0) * m.sin(
                d_Y / 2) ** 2
            c = 2 * m.atan2(m.sqrt(a), m.sqrt(1 - a))
            reg.dist = r * c * 1000
        else:
            reg.dist = m.sqrt((reg.wpX - reg.poX) ** 2 + (reg.wpY - reg
.poY) ** 2)

def GPSCallback(signal):
    if gps.Mode == True:
        reg.poX = signal.longitude
        reg.poY = signal.latitude

def compassCallback(heading):
    if gps.Mode == True:
        reg.head = 360 - heading.data

# EXPERIMENTAL FUNCTION
# Avoid collision when going forward
def lidarCallback(scan):
    # Safe values for the lidar is found to be in ranges:
    # 16-482 port and 820-1283 starboard
    # Respectivly 4.5-135 and 230-359.4 degrees
    angleSpace = 360.0 / len(scan.ranges)   #~0.28
    rang = scan.ranges
    minLeft = 1000
    leftIndex = 0
    minRight = 1000
    rightIndex = 0

    # Find shortest range on port side
    for i in range(16,483):
        if rang[i] != float('inf'):       # excludes 'inf' values
            if rang[i] < minLeft:
                minLeft = rang[i]
                leftIndex = i

    # Find shortest range on starboard side
```

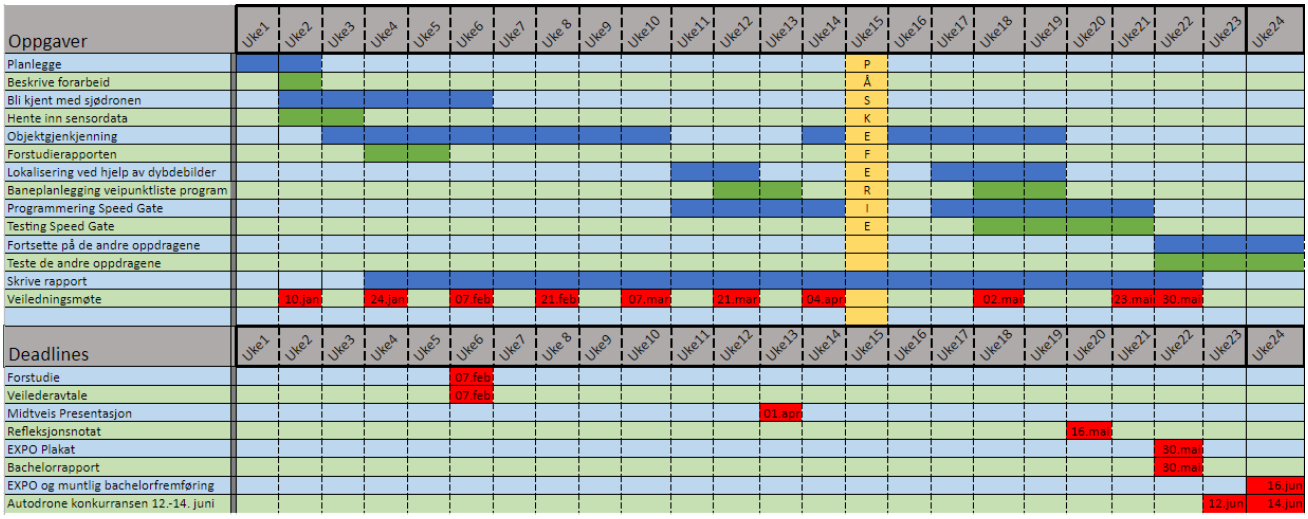
```
    for i in range(820, 1284):
        if rang[i] != float('inf'):       # excludes 'inf' values
            if rang[i] < minRight:
                minRight = rang[i]
                rightIndex = i

        obst.leftAng = leftIndex*angleSpace - 2             # compensat
ing 2 degrees
        obst.leftDist = minLeft
        obst.rightAng = 360.0 - rightIndex*angleSpace + 2   # compensat
ing 2 degrees
        obst.rightDist = minRight

    # print "Left: {} m at {} deg".format(obst.leftDist, obst.leftAng)
    # print "Right: {} m at {} deg".format(obst.rightDist, obst.rightAn
g)
    # print"____"

def modeCallback(mode):
    gps.Mode = mode.gps
    #
    # Regulator
    #

    try:
        wantedTheta = m.atan((reg.wpY-reg.poY)/(reg.wpX-
reg.poX))    # Radians
        wantedTheta = m.degrees(wantedTheta)
        if reg.wpX < reg.poX:
            wantedTheta += 180.0

        if wantedTheta < 0:
            wantedTheta += 360.0

    except:
        wantedTheta = reg.head    # You can't devide by 0

    if gps.Mode:
        currentTheta = reg.head - 90
        if currentTheta < 0:
            currentTheta += 360
    else:
        currentTheta = reg.head

    sse = wantedTheta - currentTheta
    if sse > 180:
        sse = sse - 360
    if sse < -180:
        sse = sse + 360
```

```
    # turn left if SSE > 0 and right if SSE < 0
    # regulate with parameters
    steer.angle = 50 - (sse * Pparam) - (sse * (steer.angle-
50) * Iparam)
    if steer.rate > 2:
        steer.rate = 2
    if steer.rate < -2:
        steer.rate = -2

    if steer.angle > 100:
        steer.angle = 100
    if steer.angle < 0:
        steer.angle = 0

    # print once a sec
    T.Current = time.time()
    if T.Current >= T.Next:
        T.Next += 1

        print "____"
        print "Time: {}".format(T.Current)
        print "I am at: {}, {}".format(reg.poX, reg.poY)
        print "Going to: {}, {}".format(reg.wpX, reg.wpY)
        print "Heading:        {}".format(currentTheta)
        print "Wanted heading: {}".format(wantedTheta)
        print "SSE:            {}".format(sse)
        print "Distance: {} m".format(reg.dist)
        print "Svinger {}".format(steer.angle)


    # If there is a waypoint in the list, drive at set max speed
    if reg.wpX == 0.0 and reg.wpY == 0.0:
        steer.Mdir = 0
    else:
        steer.speed = maxSpeed
        steer.Mdir = 1

    # EXPERIMENTAL
    # if there is an obsticle in front or in the turning direction
    # brake down
    if ((obst.leftAng < 30) and (obst.leftDist < 1)) or ((obst.rightAng
 < 30) and (obst.rightDist < 1)):
        if (obst.leftDist < collisionDistance) or (obst.rightDist < col
lisionDistance):
            steer.speed = (1-min(obst.leftDist, obst.rightDist))*50
            steer.Mdir = 2
```

```
    # reduces the speed when testing on land
    if steer.speed > maxSpeed: steer.speed = maxSpeed

    if mode.auto == True:
        twist.left_thruster.linear.x = steer.speed
        twist.right_thruster.linear.x = steer.speed
        twist.left_thruster.linear.dir = steer.Mdir
        twist.right_thruster.linear.dir = steer.Mdir
        twist.left_thruster.angular.z = steer.angle
        twist.right_thruster.angular.z = steer.angle
        SeaPub.publish(twist)


    else:
        # print "RC mode"
        pass

def main():
    rospy.init_node('hvlNavigator')

    global reg, twist, steer, obst, gps, T
    reg = Regulator()                       # Containing values for the r
egulator
    reg.wpIndex = 0
    reg.wpX = 0.0
    reg.wpY = 0.0
    reg.wprad = 0.0
    twist = Sea_Twist()
    steer = Driving(0.0, 50.0, 0, 10.0)    # Initial values straight for
ward 0 speed
    obst = Collision()
    gps = Positioning(False)                # will it navigate using GPS
reciever or IMU
    T = Timer(float(time.time()), float(time.time() + 1))    # Timer for
 printing

    rospy.Subscriber("/waypoint", Waypoint, waypointCallback)
    rospy.Subscriber("/zed2/zed_node/pose", PoseStamped, zedPoseCallbac
k)
    rospy.Subscriber("/mavros/global_position/global", NavSatFix, GPSCa
llback)
    rospy.Subscriber("/mavros/global_position/compass_hdg", Float64, co
mpassCallback)
    rospy.Subscriber("/scan", LaserScan, lidarCallback)

    rospy.Subscriber("/modus", Modus, modeCallback) # Basicly doing eve
rything

    print("Ready to handle waypoints")
```

```
    rospy.spin()

if __name__ == '__main__':
    main()
```

10.18 Vedlegg 18 – Fremdriftsplan



Oppgaver	Uke1	Uke2	Uke3	Uke4	Uke5	Uke6	Uke7	Uke 8	Uke9	Uke10	Uke11	Uke12	Uke13	Uke14	Uke15	Uke16	Uke17	Uke18	Uke19	Uke20	Uke21	Uke22	Uke23	Uke24
Planlegge															P									
Beskrive forarbeid															Å									
Bli kjent med sjødronen															S									
Hente inn sensordata															K									
Objektgjenkjenning															E									
Forstudierapporten															F									
Lokalisering ved hjelp av dybdebilder															E									
Baneplanlegging veipunktliste program															R									
Programmering Speed Gate															I									
Testing Speed Gate															E									
Fortsette på de andre oppdragene																								
Teste de andre oppdragene																								
Skrive rapport																								
Veiledningsmøte		10.jan		24.jan		07.feb		21.feb		07.mar		21.mar		04.apr				02.mai			23.mai	30.mai		

Deadlines	Uke1	Uke2	Uke3	Uke4	Uke5	Uke6	Uke7	Uke 8	Uke9	Uke10	Uke11	Uke12	Uke13	Uke14	Uke15	Uke16	Uke17	Uke18	Uke19	Uke20	Uke21	Uke22	Uke23	Uke24
Forstudie						07.feb																		
Veilederavtale						07.feb																		
Midtveis Presentasjon													01.apr											
Refleksjonsnotat																				16.mai				
EXPO Plakat																						30.mai		
Bachelorrapport																						30.mai		
EXPO og muntlig bachelorfremføring																								16.jun
Autodrone konkurransen 12.-14. juni																							12.jun	14.jun

10.19 Vedlegg 19 – Installasjons prosedyre

1. INSTALLERE UBUNTU PÅ MINNEKORTET TIL NVIDIA JETSON:

https://developer.nvidia.com/embedded/learn/get-started-jetson-xavier-nx-devkit#intro

2. INSTALLERE ROS MELODIC:

http://wiki.ros.org/melodic/Installation/Ubuntu

3. LAGE CATKIN_WS:

https://wiki.ros.org/catkin#Installing_catkin

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

4. INSTALLERE YOLO:

https://github.com/leggedrobotics/darknet_ros

5. OM MAN FÅR CV_BRIDGE FEILKODE ETTER YOLO ER INSTALLERT:

https://github.com/ros-perception/vision_opencv/issues/345

6. INSTALLERE ZED SDK:

https://www.stereolabs.com/developers/release/

- Velg "ZED SDK for Jetpack 4.6"
- Gå inn i "Downloads" folder i terminal
- $ chmod +x ZED_SDK_Tegra_JP46_v3.6.5.run
- $ ./ZED_SDK_Tegra_JP46_v3.6.5.run

7. INSTALLERE ZED ROS:

https://github.com/stereolabs/zed-ros-wrapper

8. INSTALLERE RPLIDAR ROS:

https://github.com/Slamtec/rplidar_ros

9. INSTALLERE ZED ROS EXAMPLES:

https://github.com/stereolabs/zed-ros-examples

10.20 Vedlegg 20 – Timeliste Linn

Dato	Beskrivelse	Antall timer
Uke 2, mandag 10.01.2022	Hatt veiledningsmøte, laget framdriftsplan, kontaktet sjøfartsdirektoratet og beskrevet utstyr på båten.	6
Uke 2, tirsdag 11.01.2022	Koblet oss opp på båten og blitt litt kjent med eksisterende kode.	5
Uke 2, onsdag 12.01.2022	Blitt kjent med kameraet og lidaren. Blitt mer kjent med eksisterende kode og hatt litt introkurs i python.	7
Uke 2, torsdag 13.01.2022	Blitt mer kjent med eksisterende kode og oppdatert GPS.	7
Uke 2, fredag 14.01.2022	Fått til "Hello World" med publisher og subscriber både på Jetson kortet og på MobaXterm.	3
Uke 3, mandag 17.01.2022	Hatt møte med Gizem og Laurenz, laget sensordata script og begynt å tenke på hvordan vi skal bruke kameraet.	7
Uke 3, onsdag 19.01.2022	Jobbet med openCV og få fram fargebildet og dybdebildet fra kameraet.	7
Uke 3, torsdag 20.01.2022	Jobbet med YOLO. Ikke fått til.	7
Uke 3, fredag 21.01.2022	Forbedret oss til veiledermøtet.	1
Uke 4, mandag 24.01.2022	Hatt veiledermøte, laget møtereferat, oppdatert timelisten, fremdriftsplanen og jobbet mer med forarbeidsrapporten.	7
Uke 4, onsdag 26.01.2022	Jobbet med forstudierapporten.	6
Uke 4, torsdag 27.01.2022	Byttet SD-kort og begynt å laste inn på nytt og skrevet mer på forstudierapporten.	10
Uke 4, fredag 28.01.2022	Skrevet mer på forstudierapporten og fortsatt å laste inn ting på nytt.	6

Uke 5, mandag 31.01.2022	Skrevet på forstudierapporten og fått lastet ned YOLO.	7
Uke 5, onsdag 02.02.2022	Hatt «veiledningsmøte» og skrevet på forstudierapporten.	7
Uke 5, torsdag 03.02.2022	Skrevet på forstudierapporten og lastet ned Ubuntu på nytt. Alt er oppe og går og har fått YOLO til å kjøre.	7
Uke 5, fredag 04.02.2022	Forbedret oss til veiledningsmøte, rettet mer på forstudierapporten og fått GPS-en til å funke.	6
Uke 6, mandag 07.02.2022	Hatt veiledningsmøte, ferdigstilt budsjettet og levert forstudierapporten. Begynt å sett på SLAM og mer om ROS.	6
Uke 6, onsdag 09.02.2022	Jobbet med SLAM på stereokameraet og lest oss opp på SLAM.	6
Uke 6, torsdag 10.02.2022	Rotet med Yolo og må starte på nytt.	6
Uke 6, fredag 11.02.2022	Jobbet mer med YOLO.	4
Uke 7, mandag 14.02.2022	Jobbet mer med YOLO.	6
Uke 7, onsdag 16.02.2022	Jobbet mer med YOLO.	7
Uke 7, torsdag 17.02.2022	Jobbet mer med YOLO og begynt å skrive på bachelorrapporten.	7
Uke 7, fredag 18.02.2022	Jobbet mer med YOLO og skrevet mer på bachelorrapporten.	4
Uke 8, mandag 21.02.2022	Jobbet mer med YOLO.	7
Uke 8, onsdag 23.02.2022	Jobbet mer med YOLO.	7
Uke 8, torsdag 24.02.2022	Jobbet med Jetson, laget Start.sh med YOLO og startet å fått lidaren oppe å går igjen.	5
Uke 8, fredag 25.02.2022	Fått lidaren til.	4
Uke 9, mandag 28.02.2022	Begynt såvidt på lokalisering.	6

Uke 9, onsdag 02.03.2022	Fortsatt på lokaliseringen. Kommet greit i gang.	7
Uke 9, torsdag 03.03.2022	Fortsatt på lokalisering. Står litt fast.	6
Uke 9, fredag 04.03.2022	Fått til lokalisering!	5
Uke 10, mandag 07.03.2022	Hatt veiledningsmøte og møte med Gizem og Laurenz. Byttet navn på pakken og testet avstandsmålinger.	7
Uke 10, onsdag 09.03.2022	Jobbet med modusdetekteringsprogram.	7
Uke 10, torsdag 10.03.2022	Fått ferdig modusdetekteringsprogrammet og begynt på waypointlist programmet.	7
Uke 11, mandag 14.03.2022	Skrevet på bachelorrapporten og begynt på midtveispresentasjonen. Prøver å lage waypointliste-program, men har ikke fått til enda.	7
Uke 11, onsdag 16.03.2022	Forbedret oss til veiledningsmøtet, jobbet mer med midtveispresentasjonen, skrevet mer på rapporten og jobbet mer med waypointliste-programmet + litt på navigeringsprogrammet.	7
Uke 11, torsdag 17.03.2022	Hatt møte med Gizem og Laurenz. Sett mer på waypointliste-programmet og sett på pathfinder_ros.	6
Uke 11, fredag 18.03.2022	Sett mer på waypointliste-programmet.	5
Uke 12, fredag 25.03.2022	Oppdatering på hva som er gjort iløpet av uken siden Linn har hatt corona.	1
Uke 12, lørdag 26.03.2022	Oppdatert budsjettet siden vi nå har fått ballongene som skal brukes fra USN.	2

	Oppdatert framdriftsplan med tanke på avlyst veiledningsmøte 18.april. Jobbet mer med midtveispresentasjonen.	
Uke 13, mandag 28.03.2022	Jobbet med midtveispresentasjonen. Skrevet på bachelorrapporten og testet lokaliseringen på sjødronen.	7
Uke 13, tirsdag 29.03.2022	Jobbet med midtveispresentasjonen.	2
Uke 13, onsdag 30.03.2022	Skrevet på bachelorrapporten, tatt kontakt med Smedasundet igjen og sett mer på lokalisering og pose.	7
Uke 13, torsdag 31.03.2022	Jobbet mer med IMU-en, skrevet på bachelorrapporten og øvd til midtveispresentasjonen.	6
Uke 13, fredag 01.04.2022	Forberedt oss til midtveispresentasjon. Laget «anker» til bøyene. Forberedt oss til veiledningsmøte. Hatt midtveispresentasjon.	5
Uke 14, mandag 04.04.2022	Hatt veiledningsmøte. Tatt bilder av bøyene. Lagt datasett for bøyene på roboflow.com. Prøvd å finne ut hvordan vi kan få dette inn på ROS/Jetson.	7
Uke 14, onsdag 06.04.2022	Hatt første testdag hvor vi har testet sjødronen på vannet. Tatt bilder av bøyene sammen til YOLO treningen.	6
Uke 14, torsdag 07.04.2022	Lagt datasett for bøyene på roboflow.com.	8

	Begynt å sett på YOLO treningen. Gått over scriptet til localization.	
Uke 14, fredag 08.04.2022	Testet sjødronen.	3
Uke 15, lørdag 16.04.2022	Jobbet med den muntlige bachelorpresentasjonen.	2
Uke 15, søndag 17.04.2022	Prøvd å starte treningen av YOLO.	7
Uke 16, mandag 18.04.2022	Fått til å starte treningen av YOLO! Skrevet på rapporten og muntlige presentasjonen.	7
Uke 16, tirsdag 19.04.2022	Lagret bilder og satt firkant rundt objektene på roboflow.	3
Uke 16, fredag 22.04.2022	Laget balasten og anker til bøyene. Testet sjødronen, fungerte ikke som vi håpet. Har problemer med kompasset.	7
Uke 16, søndag 24.04.2022	Laget EXPO-plakat utkast.	2
Uke 17, mandag 25.04.2022	Fått YOLO til å ikke se blått og til å detektere bøyer!	6
Uke 17, onsdag 27.04.2022	Sett på mode_selector og localization scriptene og ryddet opp i kode. Skrevet på bachelorrapporten.	7
Uke 17, torsdag 28.04.2022	Skrevet refleksjonsnotatet, sett på localizer scriptet som vi har problemer med og startet å markere bildene som skal brukes til den andre yolotreningen.	6
Uke 17, fredag 29.04.2022	Markert bilder som skal brukes til den andre yolotreningen. Levert inn refleksjonsnotatet.	9
Uke 17, lørdag 30.04.2022	Trent opp YOLO. Forberedet oss til veiledningsmøtet på mandag. Skrevet på bachelorrapporten.	7

Uke 17, søndag 01.05.2022	Skrevet på bachelorrapporten.	3
Uke 18, mandag 02.05.2022	Lagt inn YOLOv2. Hatt veiledningsmøte. Skrevet på bachelorrapporten.	7
Uke 18, tirsdag 03.05.2022	Prøvd å løse localizer.py problemet.	7
Uke 18, onsdag 04.05.2022	Jobbet med bachelorrapporten, muntlig bachelorfremføring og localizer.py scriptet. Går framover. Rettet opp på navigation.py.	8
Uke 18, torsdag 05.05.2022	Jobbet med bachelorrapporten. Localizer.py scriptet fungerer. Testet og rettet på wp_maker.py.	6
Uke 18, fredag 06.05.2022	Rettet på wp_maker.py. Skrevet på bachelorrapporten. Tatt flere bilder av bøyene. Undersøkt ferdig datasett på YOLO.	6
Uke 18, lørdag 07.05.2022	Markert bøyebildene på roboflow.	5
Uke 18, søndag 08.05.2022	Lastet ned ferdigtrente datasett for Bird, Boat, House og Person. Begynt å trene YOLO for tredje gang.	10
Uke 19, mandag 09.05.2022	Testet YOLO algoritmen og skrevet på bachelorrapporten.	6
Uke 19, tirsdag 10.05.2022	Trent ferdig YOLO.	4
Uke 19, onsdag 11.05.2022	Lagt inn ferdigtrente YOLO algoritmen, testet og feilsøkt på localization.py, wp_maker.py og navigation.py.	6
Uke 19, torsdag 12.05.2022	Jobbet mer med bachelorrapporten og laget	4

	tegninger og satt inn formler på localizer.py.	
Uke 19, fredag 13.05.2022	Jobbet mer med bachelorrapporten og fullført localizer.py kapittelet.	5
Uke 19, lørdag 14.05.2022	Skrevet mer på bachelorrapporten.	5
Uke 20, mandag 16.05.2022	Skrevet på bachelorrapporten.	8
Uke 20, torsdag 19.05.2022	Skrevet på bachelorrapporten. Rettet på fremdriftsplanen.	8
Uke 20, fredag 20.05.2022	Skrevet på bachelorrapporten.	5
Uke 20, søndag 22.05.2022	Skrevet på bachelorrapporten. Testet Speed Gate på vann.	7
Uke 21, mandag 23.05.2022	Hatt veiledningsmøte og skrevet på bachelorrapporten.	7
Uke 21, tirsdag 24.05.2022	Skrevet på bachelorrapporten.	7
Uke 21, onsdag 25.05.2022	Skrevet på bachelorrapporten.	7
Uke 21, torsdag 26.05.2022	Skrevet på bachelorrapporten.	8
Uke 21, fredag 27.05.2022	Skrevet på bachelorrapporten.	8
Uke 21, lørdag 28.05.2022	Fullført bachelorrapporten.	6
Uke 21, søndag 29.05.2022	Levert bacheloroppgaven.	2

10.21 Vedlegg 21 – Timeliste Magnus

Dato	Start	Slutt	Pause	Gjort	Timer/dag
10.01.2022	10:00:00	20:00:00	01:00:00	Møte med veiledere og skrevet tekniske beskrivelser	09:00:00
11.01.2022	10:00:00	16:00:00	02:00:00	Satt oss inn i oppbygningen av båten og forsøkt å lese data fra sensorer	04:00:00
12.01.2022	08:00:00	16:00:00	01:00:00	Forsøkt å lese data fra sensorer	07:00:00
13.01.2022	08:00:00	16:00:00	01:00:00	Forsøkt å lese data fra sensorer opdatert drivere, uten bedre resultat.	07:00:00
14.01.2022	08:00:00	11:30:00	00:00:00	Laget HelloWorld program	03:30:00
15.01.2022				Helg	00:00:00
16.01.2022				Helg	00:00:00
17.01.2022	08:30:00	15:30:00	00:30:00	Implementert sensordata i python, og sett på hvordan styre aktuatorene.	06:30:00
18.01.2022				Innovasjon og systemtenking	00:00:00
19.01.2022	08:00:00	15:30:00	00:30:00	Fått bilder fra ROS message inn i OpenCV. Fått nøkkel til 2019	07:00:00
20.01.2022	09:00:00	16:30:00	00:30:00	Prøvd oss på YOLO. Gått tom for lagringskapasitet	07:00:00
21.01.2022	11:00:00	12:00:00	00:00:00	Pyntet på forprosjekt oppgaven og gjort ferdig presentasjon for veiledermøtet	01:00:00
22.01.2022				Helg	00:00:00
23.01.2022				Helg	00:00:00
24.01.2022	08:30:00	16:30:00	00:30:00	Mandagsmøte, fokusert på forstudierapporten	07:30:00
25.01.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
26.01.2022	08:00:00	15:00:00	00:30:00	Fokusert på forstudierapporten	06:30:00
27.01.2022	08:00:00	18:30:00	01:00:00	Fikset lagringsproblemet, og begynt å installere pakker på nytt	09:30:00
28.01.2022	10:00:00	16:00:00	00:30:00	Installert pakker til kameraet og lidaren, men har problemer med å kjøre de	05:30:00
29.01.2022				Helg	00:00:00
30.01.2022				Helg	00:00:00
31.01.2022	08:00:00	16:00:00	01:00:00	Fortsatt med å sette opp ROS og installere alt som trengs for å kjøre YOLO	07:00:00
		Timer i Januar	88:00:00		
01.02.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
02.02.2022	08:30:00	17:00:00	00:30:00	Jobbet med forstudierapporten etter møte med veilederne	08:00:00
03.02.2022	08:00:00	16:00:00	00:30:00	Installert YOLO, ROS, pakker til kameraet og lidaren og gjort ferdig forstudierapporten.	07:30:00
04.02.2022	09:30:00	16:30:00	00:30:00	Funnet ut at GPSen funker og begynt å lage oppstarts-fil for å starte alle ROS nodene med en kommando	06:30:00
05.02.2022				Helg	00:00:00
06.02.2022				Helg	00:00:00

07.02.2022	09:00:00	15:30:00	00:30:00	Mandagsmøte, levert forstudierapporten. Laget ferdig oppstart programmet som starter master, og alle nodene.(Denne må fylles på etterhvert) Begynt å se på ROS mer i dybden.	06:00:00
08.02.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
09.02.2022	08:00:00	15:30:00	00:30:00	Sett på muligheter for mapping, og funnet ut at rtabmap er installert via ZED kameraet. Derfor går vi nok for denne.	07:00:00
10.02.2022	09:00:00	18:00:00	00:30:00	Sett på implementasjonen av YOLO i Python, med ZED kameraet. Har også reinstallert alt på SD kortet. Igjen...	08:30:00
11.02.2022	12:00:00	16:30:00	00:00:00	Sett på implementasjonen av YOLO i Python.	04:30:00
12.02.2022				Helg	00:00:00
13.02.2022				Helg	00:00:00
14.02.2022	08:30:00	17:00:00	01:00:00	Sett mer på implementasjonen av YOLO.	07:30:00
15.02.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
16.02.2022	08:00:00	17:00:00	00:30:00	Sett mer på implementasjonen av YOLO og hvordan lage veipunkter.	08:30:00
17.02.2022	08:30:00	16:00:00	00:30:00	Funnet ut av feilkode ifm kjøring av YOLO, og fått ny feilkode	07:00:00
18.02.2022	10:30:00	14:00:00	00:30:00	Mer problemløsing ifm. YOLY	03:00:00
19.02.2022				Helg	00:00:00
20.02.2022				Helg	00:00:00
21.02.2022	10:00:00	12:00:00	00:00:00	Veiledermøte og sett etter ting vi gjør feil, som gjør at darknet_ros ikke virker for oss.	02:00:00
22.02.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
23.02.2022	09:00:00	12:00:00	00:00:00	Jobbet med rapporten	03:00:00
24.02.2022	09:00:00	14:00:00	00:00:00	Jobbet med rapporten	05:00:00
25.02.2022	10:00:00	13:00:00	00:00:00	Jobbet med rapporten og sett på løsninger for HMI	03:00:00
26.02.2022				Helg	00:00:00
27.02.2022				Helg	00:00:00
28.02.2022	11:00:00	17:00:00	01:00:00	Jobbet med rapporten	05:00:00
		Timer i Februar	92:00:00		
01.03.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
02.03.2022	08:30:00	18:00:00	01:00:00	Fått til å måle dybde i en piksel som er en god start for å lokalisere objekter i rommet	08:30:00
03.03.2022	08:30:00	16:30:00	00:30:00	Jobbet med å lokalisere flere objelter samtidig, uten stort hell	07:30:00
04.03.2022	11:00:00	16:00:00	00:00:00	Fått til mye på lokalisering	05:00:00
05.03.2022				Helg	00:00:00
06.03.2022				Helg	00:00:00
07.03.2022	08:30:00	16:30:00	00:30:00	hatt mandagsmøte og møte med Gizem og Laurenz, testet YOLO og lokalisering på avstand.	07:30:00
08.03.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
09.03.2022	08:15:00	15:30:00	00:30:00	Begynt på waypoint programmet	06:45:00
10.03.2022	08:30:00	17:00:00	01:30:00	laget program for å velge modus og sett mer på waypointer	07:00:00

11.03.2022	00:00:00	00:00:00	00:00:00	*Fjelltur i finværet*	00:00:00
12.03.2022				Helg	00:00:00
13.03.2022				Helg	00:00:00
14.03.2022	08:30:00	16:30:00	00:30:00	Jobbet mer med waypointer	07:30:00
15.03.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
16.03.2022	08:00:00	16:00:00	00:30:00	Mer waypointer	07:30:00
17.03.2022	09:00:00	16:00:00	00:30:00	Fått input fra Gizem og Laurenz, og prøvd å forstå oss på Marius Tannum sin waypointlist løsning. Har også sett på pathfinder_ros	06:30:00
18.03.2022	10:30:00	17:30:00	02:00:00	Jobbet mer med waypointer og lokalisering	05:00:00
19.03.2022				Helg	00:00:00
20.03.2022				Helg	00:00:00
21.03.2022	08:00:00	18:00:00	01:00:00	Veiledermøte og begynt på navigasjonsprogram. Ballongene som skal brukes som bøyer er også ankommet.	09:00:00
22.03.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
23.03.2022	08:30:00	16:00:00	00:30:00	Laget regulator for å styre mot et veipunkt	07:00:00
24.03.2022	08:30:00	14:30:00	00:30:00	Laget enkel algoritme for å unngå kollisjon når båten kjører fremover. Funnet mulig egnet sted for testing.	05:30:00
25.03.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
26.03.2022				Helg	00:00:00
27.03.2022				Helg	00:00:00
28.03.2022	10:00:00	16:00:00	00:30:00	Jobbet med lokalisering av dronen, da GPS ikke er nøyaktig nok	05:30:00
29.03.2022	08:00:00	16:00:00	03:30:00	Innovasjonsdag, men har jobbet med midtveispresentasjonen, og sett på lokalisering av dronen	04:30:00
30.03.2022	08:30:00	18:00:00	01:00:00	Jobbet med litt av alt egentlig	08:30:00
31.03.2022	08:30:00	16:00:00	01:00:00	Jobbet med navigasjonsalgoritmen og midtveispresentasjonen	06:30:00
		Timer i Mars	115:15:00		
01.04.2022	07:30:00	14:00:00	00:30:00	Midtveispresentasjon og lokalisering av objekter	06:00:00
02.04.2022				Helg	00:00:00
03.04.2022				Helg	00:00:00
04.04.2022	08:30:00	17:00:00	00:30:00	Veiledermøte, blåst opp ballongene og jobet med lokalisering av objekter	08:00:00
05.04.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
06.04.2022	07:30:00	15:00:00	00:30:00	Gjort dronen klar for sjøsetting, kjørt den og prøvd å justere parametrene så den kjører pent mot veipunktet	07:00:00
07.04.2022	08:00:00	17:00:00	00:30:00	Fikset på lokaliseringsalgoritmen, justert navigasjonsalgoritmen og begynt så vidt på speedgate oppgaven	08:30:00
08.04.2022	07:00:00	17:00:00	01:30:00	Prøvd justeringene på navigasjonen, og gjort om litt på den da det ikke funka.	08:30:00
09.04.2022				Helg	00:00:00
10.04.2022				Helg	00:00:00

11.04.2022					00:00:00
12.04.2022					00:00:00
13.04.2022					00:00:00
14.04.2022					00:00:00
15.04.2022					00:00:00
16.04.2022				Helg	00:00:00
17.04.2022				Helg	00:00:00
18.04.2022					00:00:00
19.04.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
20.04.2022					00:00:00
21.04.2022					00:00:00
22.04.2022	08:00:00	17:30:00	01:30:00	Testet sjødronen og prøvd å finne ut av kompassproblemene	08:00:00
23.04.2022				Helg	00:00:00
24.04.2022				Helg	00:00:00
25.04.2022	08:00:00	16:00:00	00:30:00	Programmert utkast til speedgate	07:30:00
26.04.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
27.04.2022	08:00:00	16:00:00	01:00:00	Prøvd å få lokaliserings algoritmen til å funke, og diskutert EXPO plakaten	07:00:00
28.04.2022	11:00:00	17:00:00	01:00:00	Prøvd å få lokaliserings algoritmen til å funke	05:00:00
29.04.2022	11:00:00	12:00:00	00:00:00	jobbet med EXPO plakaten	01:00:00
30.04.2022					00:00:00
		Timer i April	66:30:00		
01.05.2022					00:00:00
02.05.2022	09:00:00	16:00:00	00:30:00	Hatt veiledermøte og prøvd å få lokaliserings algoritmen til å funke med varierende hell	06:30:00
03.05.2022	08:30:00	17:30:00	00:30:00	Jobbet med rapporten og prøvd å gjøre om på lokaliseringsalgoritmen. Har også forsøkt å se om et annet kompass har samme problemer som vårt.	08:30:00
04.05.2022	08:00:00	17:00:00	00:30:00	Fikset litt på navigasjonsalgoritmen og fått lokaliseringsalgoritmen til å funke mer optimalt, men fortsatt litt igjen.	08:30:00
05.05.2022	08:00:00	15:00:00	00:30:00	Jobbet med å få baneplanleggings algoritmen til å funke som den skal, mot waypointlist	06:30:00
06.05.2022	08:30:00	16:00:00	00:30:00	Fjernet waypointlist til fordel for egen løsning i waypoint_maker, og fått til å sette de første veipunktene.	07:00:00
07.05.2022				Helg	00:00:00
08.05.2022				Helg	00:00:00
09.05.2022					00:00:00
10.05.2022	00:00:00	00:00:00	00:00:00	Innovasjon og systemtenking	00:00:00
11.05.2022					00:00:00
12.05.2022					00:00:00
13.05.2022					00:00:00
14.05.2022				Helg	00:00:00

15.05.2022	12:00:00	18:00:00	01:00:00	Gått gjennom rapporten for å ferdigstille for utkast	05:00:00
16.05.2022	09:30:00	17:30:00	00:30:00	Gjort ferdig første utkast av rapporten	07:30:00
17.05.2022					00:00:00
18.05.2022					00:00:00
19.05.2022				Eksamen	00:00:00
20.05.2022	14:00:00	18:00:00	00:00:00	Jobbet med og testet wp_maker	04:00:00
21.05.2022				Helg	00:00:00
22.05.2022	12:00:00	18:30:00	00:30:00	Testet speedgate på land, og gjort justeringer på lokaliseringen	06:00:00
23.05.2022	08:00:00	17:00:00	02:00:00	Gått gjennom rapporten med veiledere, og jobbet mer på den	07:00:00
24.05.2022	08:00:00	16:30:00	00:30:00	Jobbet med rapporten	08:00:00
25.05.2022	09:00:00	17:30:00	00:30:00	Jobbet med rapporten	08:00:00
26.05.2022	08:30:00	20:00:00	01:00:00	Jobbet med rapporten	10:30:00
27.05.2022	10:30:00	20:00:00	01:00:00	Jobbet med rapporten	08:30:00
28.05.2022	10:30:00	18:00:00	01:00:00	Jobbet med rapporten	06:30:00
29.05.2022	12:00:00	14:00:00	00:00:00	Lese gjennom rapporten	02:00:00
30.05.2022					00:00:00
31.05.2022					00:00:00
		Timer i Mai	110:00:00		
		Timer totalt	471:45:00		