Høgskulen på Vestlandet

Bacheloroppgave

ELE350

Predefinert informasjon

Startdato:	08-05-2023 09:00 CEST	**Termin:**	2023 VÅR
Sluttdato:	22-05-2023 14:00 CEST	**Vurderingsform:**	Norsk 6-trinns skala (A-F)
Eksamensform:	Bacheloroppgave		
Flowkode:	203 ELE350 1 O 2023 VÅR		
Intern sensor:	Raquel Motzfeldt Tirach		

Deltaker

Navn:	William Antonesen
Kandidatnr.:	209
HVL-id:	590413@hvl.no

Informasjon fra deltaker

Egenerklæring *:	Ja
Inneholder besvarelsen konfidensielt materiale?:	Nei
Jeg bekrefter at jeg har registrert oppgavetittelen på norsk og engelsk i StudentWeb og vet at denne vil stå på vitnemålet mitt *:	Ja

Gruppe

Gruppenavn:	BO23EF-02 SELV-KJØRENDE MOBIL ROBOT FOR FORSKNING INNEN JORDBRUK
Gruppenummer:	21
Andre medlemmer i gruppen:	Stian Svarstad Isene

Jeg godkjenner avtalen om publisering av bacheloroppgaven min *

Ja

Er bacheloroppgaven skrevet som del av et større forskningsprosjekt ved HVL? *

Ja, FutuRaPS og RoCutO

Høgskulen på Vestlandet

BACHELOROPPGAVE:

BO23EF-02 SELV-KJØRENDE MOBIL ROBOT FOR FORSKNING INNEN JORDBRUK

William Antonesen
Stian Svarstad Isene

22. mai. 2023

Dokumentkontroll

Rapportens tittel: BO23EF-02 SELV-KJØRENDE MOBIL ROBOT FOR FORSKNING INNEN JORDBRUK	*Dato/Versjon* 22. mai. 2023/1.0
	Rapportnummer: BO23EF-02
Forfatter(e): William Antonesen Stian Svarstad Isene	*Studieretning:* Automatiseringsteknikk med robotikk
	Antall sider m/vedlegg 77
Høgskolens veileder: Raquel Motzfeldt Tirach	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: Høgskulen på Vestlandet (HVL Robotics)	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Martin Fodstad Stølen*	

Revisjon	**Dato**	**Status**	**Utført av**
0.1	28.04.2023	Første utkast. Satt opp struktur	William Antonesen
0.2	18.05.2023	Utkast av oppgave	William/Stian
0.3	20.05.2023	Utkast med korreksjon	William/Stian
1.0	22.05.2023	Ferdig Rapport	William/Stian

Forord

Denne rapporten dokumenterer arbeidet gjort i vår avsluttende bacheloroppgave innen Automasjon med robotikk ved Høgskulen på Vestlandet, Campus Førde. Oppgaven omhandler navigering av mobil robotbase som vår kunde HVL Robotics benytter innen flere forskingsprosjekt rettet mot landbruk. Den mobile robotbasen var vi noe kjent med før prosjektet, ettersom vi begge hadde sommerjobb ved HVL Robotics sommeren 2022. Der brukte vi den for datainnsamling ved hjelp av programvare utviklet på et tidligere bachelorprosjekt [1].

Vi takker vår veileder Raquel Motzfeldt Tirach, Senioringeniør ved HVL Robotics, som har vært tilgjengelig for spørsmål og veiledning på kort varsel under hele oppgaven. Det å ha en så dedikert veileder har vert en meget positiv opplevelse for oss.

Vi takker også Martin Fodstad Stølen, Førsteamanuensis ved HVL og vår kunde representant under prosjektet, for en spennende oppgave med mye frihet til å ta arbeidet i retningen vi ønsket. Det har også vært hyggelig å ha en kunde representant som har engasjert seg i prosjektet og har vært lett tilgjengelig ved spørsmål.

Videre vil vi takke Lukas Schild, ph.d. stipendiat ved HVL Sogndal, for rettledning innen webutvikling, noe vi var lite kjent med før prosjektet startet.

Takk også til Harald Blaaflat Mundal for at vi fikk plass på den historiske Blaaflat gård i Lærdal for vår testing i felt.

Merk at vi har valgt å referere til GNSS som GPS, ettersom det er blitt dagligtale. Mer utfyllende beskrivelse i kapittel 3.2.

Sammendrag

Dette bachelorprosjektet har hatt som mål å utvikle løsning for å navigere en mobil robotbase innenfor et brukerdefinert område utendørs. Dette ved å bruke GPS-data fra sensorer om bord på den mobile basen. Løsningen består av programvare som gir bruker mulighet til å definere et område. Det planlegges så en rute som dekker gitt arbeidsområde og robotbasen navigerer autonomt gjennom ruten.

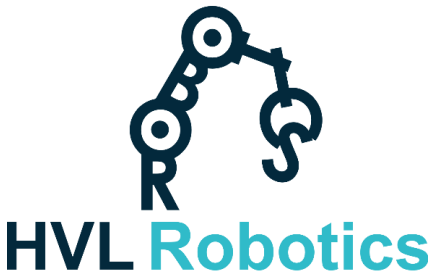
Innhold

Figur tabell

1 Innledning

1.1 Oppdragsgiver

HVL Robotics er et forskningsteam som har base ved HVL i Førde. De forsker på samarbeid mellom roboter og mennesker, og roboter rettet mot landbruk. Forskningsprosjektet denne oppgaven undergår er «Teknoløft Sogn og Fjordane» som har som mål å finne fleksible automatiserte løsninger for små og mellomstore bedrifter i gamle Sogn og Fjordane [2]. HVL Robotics med prosjektene RoCutO (*Robotic Grass Cutter For Orchards)* og FutuRaPS (*Future Raspberry Production System*) ser på mulighetene til å automatisere ulike arbeidsoppgaver hos frukt og bær produsenter ved hjelp av mobile roboter.



Figur 1: HVL Robotics logo

1.2 Problemstilling

FutuRaPS er et forskningsprosjekt med mål om å automatisere deler av bringebærproduksjonen i Vestland fylke. Prosjektet tar i bruk robotikk og stordata (*Big Data*) for å utvikle innovative løsninger tilpasset det utfordrende terrenget og klimaforhold som preger store deler av bringebærproduksjonen [3].

For å utføre noen av operasjonene som er nødvendige for en automatisert løsning, gikk HVL Robotics til innkjøp av en Husky robotbase, omtalt mer i kapittel 3.1.1. Det er en mobil robot drevet med fire hjul som skal kjøre autonomt inne på et område. Den er utstyrt med sensorer for å gi roboten informasjon om posisjon, omgivelser og hindringer, noe den kan bruke for navigering. Robotbasen har et omfattende sett med programvaregrensesnitt og utviklingsverktøy som gjør det mulig for brukeren å programmere roboten etter spesifikke oppgaver og behov.

Ved tidligere bachelorprosjekt er det utviklet algoritmer for å kjøre robotbasen langs rekker med busker ved hjelp av et dybdekamera. Dette er gjort for å samle inn data og informasjon om bærene [1]. Kunden ønsker å ta dette prosjektet videre ved å utvikle et program for å flytte roboten mellom rekkene.

Det ble utført en oppgave høsten 2022 av masterstudenter ved innovasjonslinjen til HVL, som tok for seg kartlegging av fruktdyrkerenes behov og ønsker. Der kom det fram et ønske om å automatisere oppgaver i et krevende miljø, som vil frigjøre bonden fra tidkrevende arbeid [4].

Som del av veien videre, ønsker oppdragsgiver et navigasjonsverktøy som kan brukes til å navigere autonomt på et større område ved å bruke GPS-signaler. Prosjektet RoCutO omhandler gressklipping innen jordbruk. Det er i den forbindelse et ønske om å lage en baneplanlegger som gjør det mulig å dekke et større område, og et brukergrensesnitt for lettere håndtering av roboten.

2 Kravspesifikasjon

Kravspesifikasjoner kan separeres til mål og ønsker, ettersom det er noen deler av oppgaven som er mer åpen. Under mål er det som må utføres og som skal prioriteres deretter. Under ønsker er funksjoner som kan velges mer fritt å implementere.

2.1 Mål

- Punkt-til-punkt navigering utendørs ved hjelp av GPS
    - Unngå hindringer, også kalt *Object avoidance*
    - Skifte navigeringsalgoritme basert på GPS-signalstyrke
- Demonstrasjon av system i felt
    - Test av enkel baneplanlegger

2.2 Ønsker

- Utvikle et brukergrensesnitt for å vise data, og hente data fra bruker
    - Vise kart med robotens posisjon
    - Brukertesting av løsning
- Baneplanlegger for å bruke med RoCutO prosjektet
    - Gå fra arbeidsområde til arbeidsområde
    - Dekke et helt område for gressklipping
- Simulert test miljø for Husky med GPS-signal

2.3 Justering av kravspesifikasjon

Et gjennombrudd etter prosjektstart ga HVL Robotics tilgang til programvare for punkt-til-punkt navigering utendørs med hjelp av GPS. Under navigering unngikk også roboten hindringer, men navigeringsalgoritmen endret seg ikke basert på GPS-signalstyrke. Det ble også observert under testing utført av HVL Robotics at målet forflyttet seg ved tap av GPS-signal.

Det ble derfor gjort følgende endringer i kravspesifikasjoner etter oppgaven var påbegynt:

- Finne løsning for bytte av algoritme basert på GPS-signal.
- Teste algoritme for å unngå hindringer i felt.
- Samle inn data for analysering av robotens oppførsel ved tap av GPS-signal.

3 Systemets oppbygging og verktøy

Kunden har allerede et system som skal brukes i denne oppgaven. Det er også flere verktøy for utvikling som kunden bruker i dag. Disse verktøyene skal benyttes i denne oppgaven.

3.1 Mobil robotbase

Mobil robotbase er en robot som kan bevege seg. Robotstøvsugere er et eksempel på en mobil robot. Biler med selvkjøring er også en mobile roboter, som noen av Tesla sine biler.

3.1.1 Husky UGV

Den mobile basen brukt er Husky UGV (*Unmanned Ground Vehicle*) som er utviklet av Clearpath Robotics [5]. My Bot Shop har modifisert og levert Husky brukt i prosjektet, med å installere GPS-mottaker, LIDAR og IMU, informasjon om disse kommer senere i oppgaven. I tillegg er det programvare som kommer fra både My Bot Shop og Clearpath for å gi den muligheten til å utføre navigasjon og autonom bevegelse både utendørs og innendørs.

Figur 2: Husky avbildet utenfor HVL Førde

3.1.2 Posisjonering med intern sensorikk

Om bord i den mobile plattformen er det flere sensorer som brukes til å estimere robotens bevegelse og posisjon, disse sensorene er uavhengige av ytre signaler. Ett odometer brukes til å måle rotasjonene til hjulene, mens *Inertial Measurement Unit* (IMU) bestående av akselerometer og gyroskop gir informasjon om akselerasjon og orientering. Ved å bruke akselerasjonsdataene estimeres robotens endring i posisjon over tid [6]. Gyroskopet gir informasjon om robotens orientering, som kan brukes sammen med odometer-dataen og IMU-dataen for å estimere bevegelse og posisjon. De sammensatte dataen blir også kalt odometri.

Slik orientering kalles *dead reckoning*, også kjent som bestikknavigering på norsk [7]. Bestikknavigering er en robust strategi når ytre signaler som GPS, landemerker eller visuell odometri, data fra kameraer, ikke er tilgjengelig. Små unøyaktigheter i sensor dataene kan føre til store feil i estimert posisjon over tid. I tillegg vil endringer i miljøet, som for eksempel ved ujevnheter i terrenget eller endringer i friksjonsforholdene, også påvirke estimatene. Derfor brukes ofte en kombinasjon av teknikker, som kalles sensorfusjon, for å kombinere data fra flere sensorer og gi mer pålitelige estimater av posisjon og bevegelse.

3.1.3 GPS-mottaker

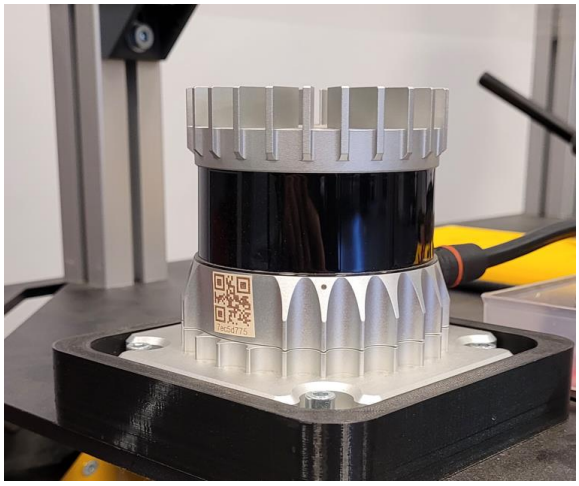
Roboten er utstyrt med GPS-mottakeren Emlid Reach RS2, som har mulighet for RTK-korreksjon. Mer informasjon om virkemåte i kapittel 3.2.

Figur 3: Emlid Reach RS2 montert på Husky

3.1.4 LIDAR

LIDAR (*Light Detection and Ranging*) er en type sensor som brukes til å måle avstand og skape et tredimensjonalt bilde av omgivelsene ved hjelp av laser [8]. LIDAR sender ut laserstråler som blir reflektert tilbake fra objekter i omgivelsene, og sensoren registrerer tiden det tar for lyset å gå frem og tilbake.

LIDAR brukes i en rekke applikasjoner, inkludert kartlegging, overvåking av miljø og selvkjørende kjøretøy. En fordel med LIDAR er dens evne til å registrere objekter på avstand. I denne oppgaven er det brukt LIDAR, av typen Ouster OS1, med rekkevidde på opptil 200 meter [9]. Dette gjør LIDAR egnet for avstandsmåling og kartlegging av terreng. I robotikk kan LIDAR brukes for å oppdage og navigere rundt hindringer i miljøet.



Figur 4: Ouster OS1 montert på Husky

3.1.5 Programvare

Husky ble levert med en del programvare, den har innebygd datamaskin som kjører Linux Ubuntu 20.02 som operativsystem og den har ROS-Noetic installert, informasjon om ROS kommer i 3.4.

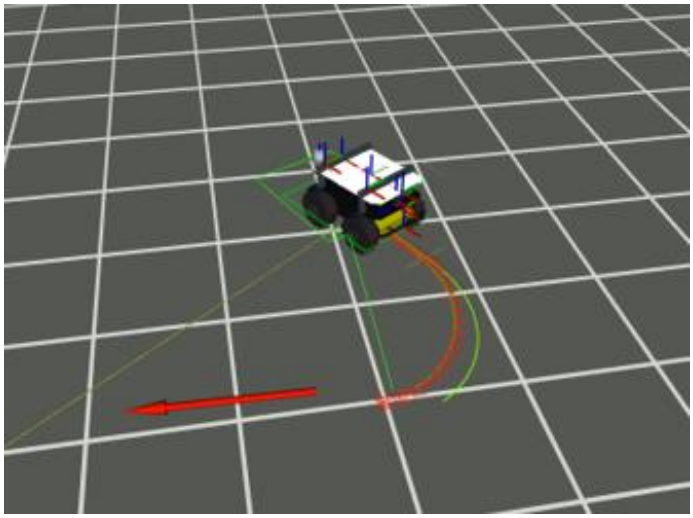
3.1.5.1 Navigasjonstakken

Den mobile basen er utstyrt med flere programvaremoduler og verktøy som brukes til å navigere roboten autonomt, kalt navigasjonstakken [10]. Den inkluderer flere funksjoner, lokalisering, kartlegging og baneplanlegging, som er nødvendige for å muliggjøre autonom navigasjon.

Navigasjonstakken består av flere nøkkelmoduler som kommuniserer med hverandre og med robotens sensorer og aktuatorer.

3.1.5.2 Baneplanlegger TEB

Robotbasen har TEB (*Timed Elastic Band*) baneplanlegger som er utviklet ved Det Tekniske Universitet i Dortmund [11]. Den genererer en bane som er gjennomførbar, og unngår hindringer. Den jobber i samarbeid med navigasjonstakken. Figur 5 viser TEB der den finner bane til angitt posisjon og orientering, representert av den røde pilen.



Figur 5: TEB baneplanlegger

TEB optimaliserer kontinuerlig banen ved å tilpasse seg til endringer i omgivelsene i sanntid, slik at roboten kan unngå hindringer og bevege seg jevnt. Dette gjøres ved å løse et flervariabelt optimeringsproblem. Vektleggingen av variablene er tilgjengelig for bruker [12].

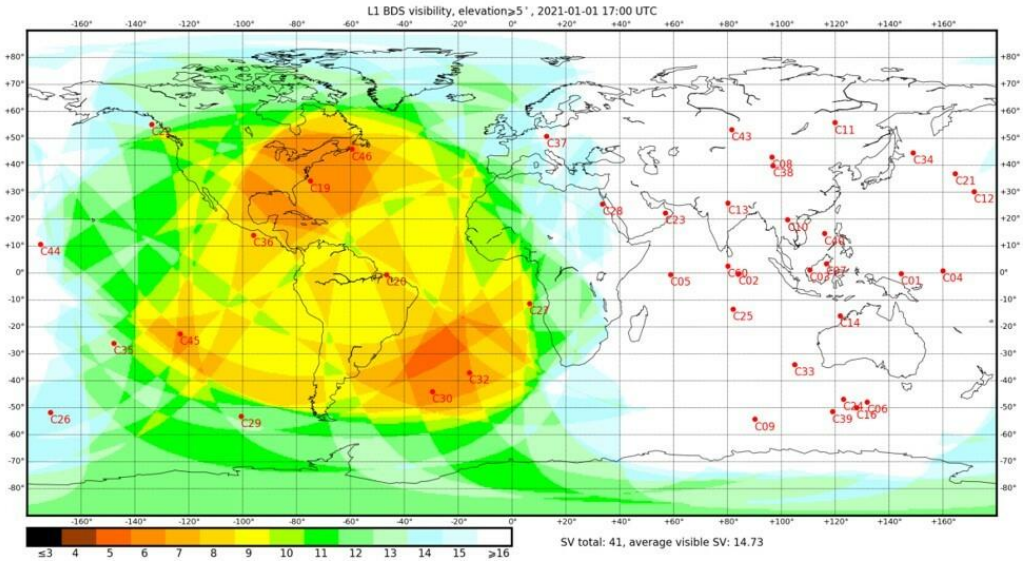
3.1.5.3 Kalmanfilter

Kalmanfilter er en matematisk metode som brukes til å estimere ukjent tilstand, her posisjon, av et system basert på målinger og estimater. Det kan brukes til å filtrere ut støy, minimere usikkerheten til et system og gi estimering av tilstanden.

Filteret består av to hovedtrinn: predikasjon og oppdatering. Predikasjonsfasen beregner et estimat av systemets tilstand basert på tidligere estimeringer og beskrivelse av systemets dynamikk. I denne oppgaven vil det være odometrien og kinematikken til roboten. I oppdateringsfasen blir estimatet sammenlignet med målinger fra sensorer, og basert på denne sammenligningen vil Kalmanfilteret justere sitt estimat av systemets tilstand og beregne oppdatert usikkerhet [13].

3.2 GNSS mottaker med RTK korreksjon

GNSS (*Global Navigation Satellite System*) er et samlebegrep for satellittbaserte lokaliseringssystemer. Det er i dag fire systemer som inngår i GNSS. GPS (*Global Positioning System*) som er utviklet i USA er nok den mest kjente. De andre tre er Russiske GLONASS (*Global'naya Navigatsionnaya Sputnikovaya Sist'ema*), europeiske Galileo og Kinesiske BeiDou. Virkemåten på de fire systemene er tilnærmet lik, men det er noe variasjon i satellitt banene som gjør at systemene dekker deler av kloden i ulik grad. Figur 6 viser dekning til BeiDou som har lavere dekningsgrad over Nord- og Sør-Amerika fordi det er bygget med et fokus på dekning av Kina [14].



Figur 6: BeiDou dekning, Kilde: [14]

I dette prosjektet brukes Emlid Reach RS2 for å lokalisere roboten. Dette er en GNSS-mottaker for alle de fire tidligere nevnte systemene. Dette gjør at selv om få satellitter fra et gitt system er tilgjengelig kan det kompenseres for ved å motta signaler fra satellitter fra flere systemer samtidig.

Siden GNSS-systemene har tilnærmet lik oppbygging er det hensiktsmessig å bare se nærmere på ett.

3.2.1 GPS

GPS-systemet består av 24 satellitter som går i bane rundt jorden. Ved å motta signaler fra minimum 3, er det teoretisk mulig å kalkulere avstanden til hver satellitt og dermed triangulere posisjonen til GPS-mottakeren. I praksis kreves mer enn 3 satellitter for å få et godt estimat av posisjon [15].

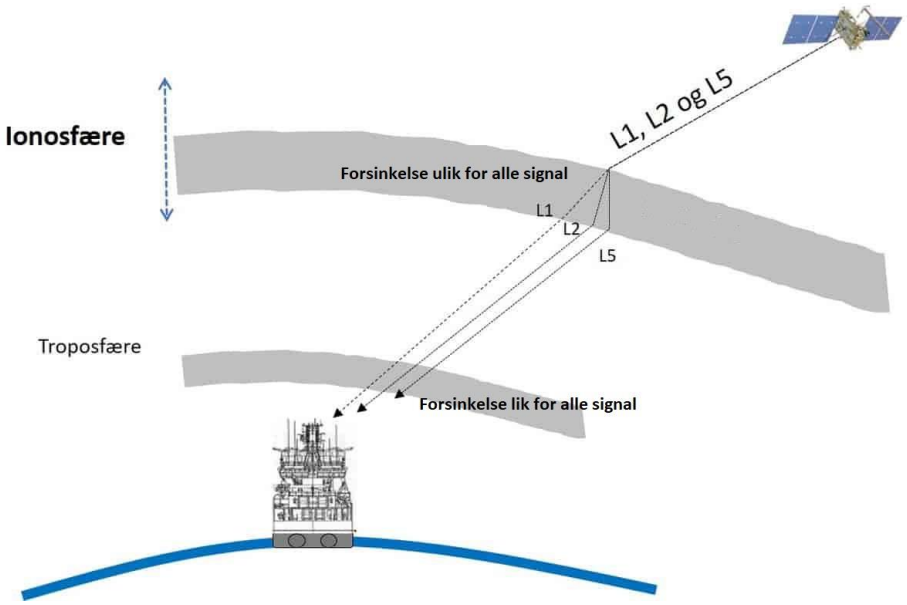
3.2.1.1 Unøyaktighet og forstyrrelser

Det er tre hovedårsaker som skaper størst forstyrrelser på signalet som når GPS-mottaker og forårsaker unøyaktighet.

Den første er at satellittens banedata kan avvike fra faktisk bane, eller satellittens klokke kan bygge seg opp et avvik. Til sammen kan disse feilene utgjøre unøyaktighet på 0,5-1 meter [15].

Hastigheten på signal fra satellitten endres når det passerer gjennom jordens atmosfære. For å kompensere for dette sender satellittene flere signaler, L1 (1575,42 MHz), L2 (1227,60 MHz) og L5 (1176,45 MHz) [15]. Disse signalene vil bli påvirket i forskjellig grad når de passerer gjennom ionosfæren. Mottaker regner ut hvor stort avviket er basert på når de forskjellige signalene blir mottatt. Dette reduserer avviket fra 5-50 meter ned til rundt 10 centimeter [15]. Når signalet passerer

troposfæren, vil det også påvirke signalets hastighet. Denne feilen er ikke frekvensavhengig og gir et avvik på 2-10 meter [15], se Figur 7. Dette blir kompensert for ved hjelp av utrekninger gjort i mottakeren og resulterer i avvik på noen titalls centimeter [15].



Figur 7: Ionosfære og troposfæres påvirkning av signal, Kilde: Omarbeide fra [15]

Signalene kan bli reflektert fra objekter i nærheten og dermed danne flerveisinterferens. For GPS-mottaker som måler på flere frekvenser er disse refleksjonene det som skaper de største feilene [15].

Nøyaktigheten til GPS i snitt målt ved bakkestasjoner igjennom 2020 er 1,82 meter [16]. Dette er ikke nøyaktig nok til å navigere en robotbase i jordbruksfelt. Det er derfor nødvendig med ytterligere korreksjon.

3.2.2 RTK

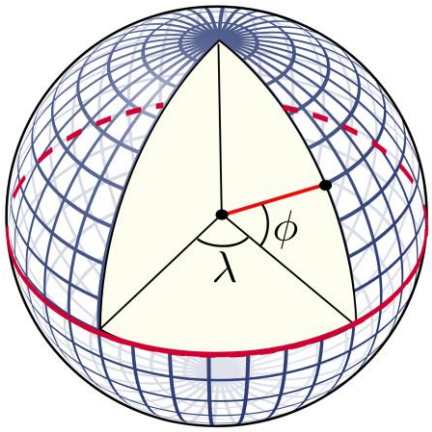
Real Time Kinematic forkortet til RTK blir brukt for å øke nøyaktigheten til GNSS-mottakere, også kalt rover når RTK blir brukt. Dette ved å bruke basestasjoner innenfor en radius på maks 10 km fra mottakeren. Basestasjonens posisjon er statisk og kjent, og har GNSS-mottaker som måler på tilnærmet det samme signalet som rover. Både basestasjon og rover vil begge motta satellittdata og kalkulere posisjon. Basestasjonen kalkulerer også avviket til sin kjente posisjon og sender det til rover over radio kommunikasjon, eller internett. Rover vil ta denne data fra basestasjon med i beregningen, og dette resulterer i bare noen centimeter avvik fra faktisk posisjon [17].

Det Norske Kartverket leverer et system kalt CPOS (Centimeterposisjon) som har basestasjoner over hele landet. Disse sender data fra basestasjon til rover over internett med bruk av NTRIP protokollen [18]. Det er dette systemet som brukes for RTK-korreksjon til GNSS-mottaker i dette prosjektet. Det blir dermed CPOS stasjonene som blir base og den mobile basen som tilsvarer rover.

3.2.3 Globale Koordinatsystemer

Globale koordinatsystemer er standardiserte måter å representere geografiske posisjoner.

Ett er det geografiske koordinatsystemet som bruker lengde og breddegrader for å beskrive posisjoner på jordoverflaten. Disse gradene kommer fra vinkelen fra jordens kjerne og ut til punktet som skal beskrives, der lengdegradene starter i Greenwich, England, og breddegradene starter ved ekvator, se Figur 8. Siden jorden ikke er sfærisk, vil disse vinklene representere forskjellige distanser på overflaten basert på hvor de er. Grader rundt Nordpolen representerer ikke de samme lengdene som grader ved ekvator. Dette gjør det vanskeligere å utføre kalkulasjoner basert på disse [19].



Figur 8: Viser breddegrader (φ) og lengdegrader (λ) på en klode, Kilde: [20]

Universal Transverse Mercator koordinat system (UTM) er en måte å beskrive global posisjon. Dette systemet baserer seg på projeksjon av jorden til en todimensjonal flate, der x-aksen går nord-sør, og y aksen går parallelt med ekvator. Ved å utføre den projeksjonen langs fastsatt breddegrad, en sentral meridian, vil systemet beskrive med centimeter nøyaktighet posisjoner langs den breddegraden, men med lavere presisjon ved posisjoner lengre vekke [21]. Dette er løst ved å dele jorden opp i 60 soner med 6 breddegrader mellom seg og en sentral meridian i midten av hver sone. UTM beskriver posisjon ved først å fastsette hvilken sone det er beskrevet, og så antall meter i nordgående retning fra ekvator, og øst fra grensen til nabosonen. Dette gjør at kalkulasjoner med disse punktene kan gjøres mye lettere [21].

3.2.3.1 Konvertering til UTM koordinater

Konverteringen til UTM koordinater foregår over et sett med serier som konvergerer seg ned til korrekt centimeter eller mindre i soner som går tre til fire grader fra den sentrale meridianen, til en meter nøyaktighet ved 10 grader[21].

$$x = k_o N \left[A + \frac{(1 - T + C)A^3}{6} + \frac{(5 - 18T + T^2 + 72C - 58e'^2)A^5}{120} \right]$$

$$y = k_o \{ M - M_0 \ + N \tan(\phi) \left[\frac{A^2}{2} + \frac{(5 - Y + \ 9C + 4C^2)A^4}{24} + \frac{(61 - 58T + T^2 + 600C - 330e'^2)A^6}{720} \right] \}$$

$$k = k_0 \left[1 + \frac{(1 + C)A^2}{2} + \frac{(5 - 4T + 42C + 13C^2 - 28e'^2)A^4}{24} + \frac{(61 - 148T + 157T^2)A^6}{720} \right]$$

Her er k_0 skaleringen til den sentrale meridianen, det blir brukt 0.9996 for UTM projeksjonen og k er skaleringen til punktet. ϕ er breddegraden, λ lengdegraden og a er lengden fra jordens senter til ekvator, her blir det brukt «World Geodetic System 1984» (WGS84) som referanseellipsoid for jorden [21].

$$e'^2 = \frac{e^2}{1 - e^2}$$

$$N = \frac{a}{(1 - e^2 \sin^2(\phi))^{\frac{1}{2}}}$$

$$T = \tan^2(\phi)$$

$$C = e'^2 \cos^2(\phi)$$

$$A = (\lambda - \lambda_0)\cos(\phi)\,, med\ \lambda\ og\ \lambda_0\ gitt\ i\ radianer$$

$$M = a\left[\left(1 - \frac{e^2}{4} - \frac{3e^4}{64} - \frac{5e^6}{256} - \cdots\right)\phi - \left(\frac{3e^2}{8} + \frac{3e^4}{32} + \frac{45e^6}{1024} + \cdots\right)\sin(2\phi)\ \ldots\right]$$

M er den sanne distansen langs den sentrale meridianen fra ekvator til breddegraden [21]

3.3 Navigasjon

Navigasjon handler om å bestemme og kontrollere bevegelse i et gitt miljø. Dette kan være utfordrende i mange områder, fra luftfart til bilkjøring. I robotikk representerer navigasjon en kritisk komponent for autonomi, og gjør det mulig for roboter å bevege seg i miljøet rundt dem uten menneskelig inngrep.

3.3.1 Robot navigasjon

Navigasjonssystemer i robotikk har til hensikt å løse to hovedproblemer: bestemme posisjon og planlegge bevegelsen til ønsket destinasjon.

For å kunne navigere autonomt, bør roboter ha tilstrekkelige kontrollsystemer. Disse systemene integrerer sensorinformasjon og bruker algoritmer for å planlegge robotens bevegelse og kontrollere den i sanntid.

3.3.2 Baneplanlegger

Baneplanlegger referer til en algoritme som beregner bane fra startposisjon til målposisjon. Banen bør ta hensyn til begrensninger og hindringer som skal unngås. I robotikk finnes det to hovedtyper av baneplanlegging: lokal og global.

Lokal baneplanlegger beregner en bane for roboten basert på data fra sensorer, og eventuelt et kart over omgivelsene. Lokal baneplanlegging brukes typisk når roboten er i bevegelse og kontinuerlig må justere sin bane for å unngå hindringer og andre farer i sanntid.

Global baneplanlegger beregner en bane fra startposisjon til målposisjon i forveien, vanligvis basert på et kart eller modell av omgivelsene. Global baneplanlegging brukes ofte i situasjoner der roboten må

navigere i store områder eller langs spesifikke baner, for eksempel i automatiserte lager eller produksjonsanlegg.

Både lokal og global baneplanlegging spiller viktige roller i robotnavigasjon, og brukes i kombinasjon for å sørge for at roboten beveger seg effektivt og sikkert gjennom sitt miljø.

3.4 ROS – Robot Operating System

ROS er et rammeverk for kommunikasjon brukt i robotikk. ROS gjør det mulig for flere programmer å kjører samtidig og kan hente data fra sensorer og hverandre [22].

Ett av hovedmålene bak utvikling av ROS var å kunne gjenbruke kode til andre prosjekt for å gjøre robotikk mer tilgjengelig [22].

3.4.1 Noder

ROS-node er et program, som utfører oppgaver for et helhetlig system. Noder kan være skrevet på forskjellige programmeringsspråk, de mest vanlige er C++ og Python. Ved å bruke nodesystemet til ROS kan prosjekt bli svært modulære, én node gjør én jobb, og det gir muligheten til å bruke om igjen samme node til andre prosjekt. Noder kan kommunisere med sensorer og aktuatorer.

3.4.2 Topic

Nodene kommuniserer med meldinger over *ROS-topic*. Meldingene har forhåndsdefinert struktur. Den strukturen kan variere mellom å være svært enkel, som tekst streng eller ett tall, til mer komplekse strukturer som inneholder kartdata, bilder eller kjøreplan.

Måten ROS håndterer sending av *topic* på er igjennom et sett med *publisher og subscriber*. *Publisher* sender melding på en *topic* og *subscriber* lytter etter meldinger. Noder kan ha flere *publisher* og *subscriber* samtidig.

3.4.3 Service og action

ROS-service og *ROS-action* er måter å samkjøre handlinger over flere noder. De tillater et system å distribuere arbeid. Begge oppretter servere som tar imot meldinger og gjør et arbeid basert på melding mottatt.

Service tar imot en melding, og sender tilbake respons basert på sin handling. Det kan være å utføre kalkulasjoner på innholdet i meldingen, eller gi tilgang til spesifikke data. Kommunikasjonen til en *service* er blokkerende, det vil si klienten venter på svar fra serveren før den fortsetter sin operasjon. En service er godt egnet til korte oppdrag. *Service*-melding er definert med to deler: forespørsel og respons.

Action tar imot en melding og begynner å utføre handlinger. *Action*-server er egnet til oppgaver som tar lengre tid å utføre, som for eksempel flytting av robotarm, eller navigering til et punkt. Når den utfører oppdraget, gir den kontinuerlig tilbakemelding. Den sender et resultat når jobben er ferdig. Det er også mulig å avbryte ett oppdrag underveis.

Action-melding er definert med tre deler. Ett mål som blir sendt til *action*-serveren fra brukeren. Tilbakemelding som blir sendt om brukeren ønsker informasjon under kjøring. Resultat som blir sendt når handlingen er ferdig.

3.4.4 Pakker

Pakker i ROS er den måten programvare blir distribuert og organisert. Pakker inneholder relaterte filer, som kildekode, konfigurasjonsfiler, datasett og dokumentasjon. En ROS pakke er organisert på bestemte måter, og inneholder alltid en fil med nødvendige metadata. Den filen inneholder blant annet pakkenavn, versjon av ROS, lisenser, avhengigheter.

Ved å bruke pakkestrukturen er programvare lett å dele. Det er også lettere å implementere andre sine løsninger i eget prosjekt.

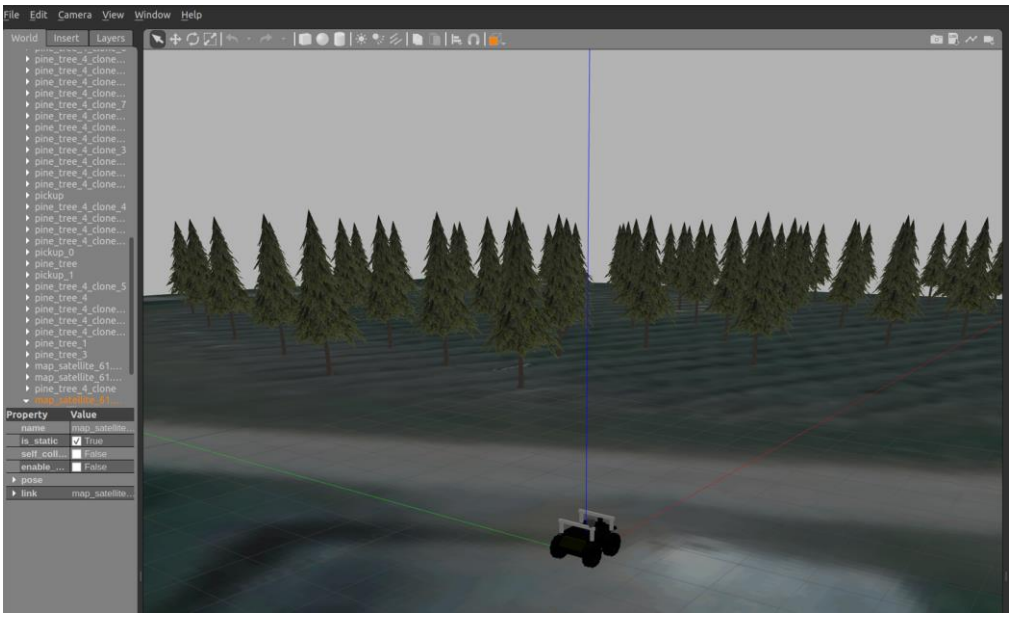
3.4.5 Launch filer

Launch-filer i ROS er filer som inneholder instruksjoner for å starte ROS noder med forskjellige parametere og konfigurasjoner. Det er også mulig å henvise til andre *launch* filer, med egne instruksjoner. Dette gir en praktisk måte å starte og administrere flere ROS noder på, spesielt i et større system med noder som er avhengige av hverandre.

3.4.6 Gazebo

Gazebo er en 3D -simulator med åpen kildekode som ofte brukes sammen med ROS for å simulere og teste roboter. Den har innebygget fysikkmotor for simuleringer [23].

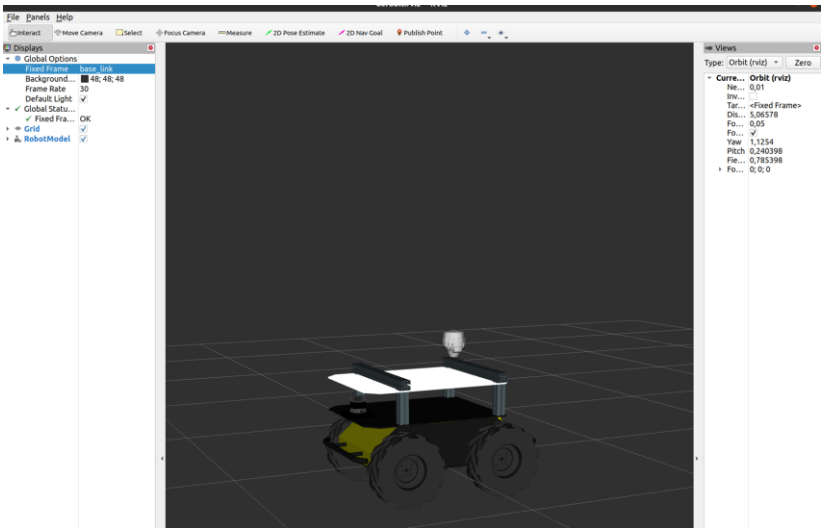
Gazebo kan vise 3D-modeller av roboter og omgivelser, simulere sensorer og kjøre ulike scenarier for testing [23]. Det er verktøy for integrasjon med ROS som gjør det mulig å visualisere og analysere data fra simuleringen. Eksempel på dette vist i Figur 9.



Figur 9: Simulering i Gazebo

3.4.7 RViz

RViz er et visualiseringsverktøy som kan brukes sammen med ROS. RViz står for *ROS-Vizualisation* og blir brukt til å visualisere sensor- og robotdata. RViz kan brukes til å vise for eksempel kart, sensordata, bevegelsesplaner, robotmodeller og video. Ved å bruke RViz kan brukere enklere verifisere at sensorikken fungerer som forventet. Figur 10 viser Husky modell i RViz.



Figur 10: Husky i RViz

3.4.8 ROSbag

ROSbag er et nyttig verktøy når det kommer til datainnsamling. Dette verktøyet lar brukeren spille inn alle meldingene på en, eller flere, *ROS-topic* for så spille det av igjen ved senere anledninger [24]. Dette gjør det mulig å analysere data fra tester i etterkant.

4 Problemanalyse og løsningsforslag

4.1 Brukergrensesnitt

Et av ønskene fra oppdragsgiver var å utvikle et brukergrensesnitt for å vise data og hente data fra bruker. Den viktigste dataen å vise til brukere er plassering av roboten på et kart, og mulighet til å se banen roboten har tatt på kartet. Data som skal hentes fra bruker er punkter roboten skal navigere gjennom eller punkter som definerer et arbeidsområde for roboten.

4.1.1 Brukergrensesnitt med Python

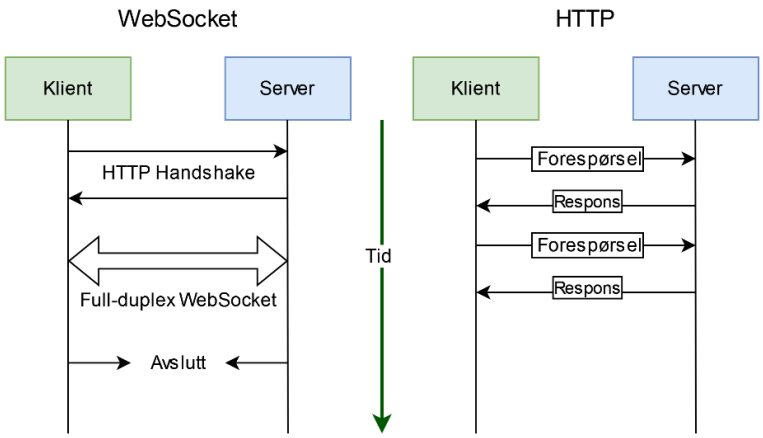
Den første løsningen som ble utforsket var et brukergrensesnitt skrevet i programmeringsspråket Python. Det ble vurdert siden det er et språk oppdragsgiver bruker for utvikling innen robotikk, og dermed er kjent. Kombinasjonen av Python pakken kalt Tkinter [25] som er brukt for å lage brukergrensesnitt med Python. Og pakken kalt TkinterMapView [26] som bygger på Tkinter og gjør det mulig å implementere kart i brukergrensesnitt.

Etter møte med oppdragsgiver kom det fram ønske om å kunne kjøre brukergrensesnittet ikke bare på PC, men også på iPad. Dette gjorde at flere muligheter måtte utforskes. Der kom det fram at det kan være fordelaktig å lage en nettapplikasjon. Dette gjør at brukergrensesnitt kan vises på alle enheter med nettleser.

For å kunne lage en nettapplikasjon må først løsning for å sende data mellom applikasjonen og ROS finnes. Dette hadde ikke vært et problem ved bruk av Python siden brukergrensesnittet kunne blitt bygget med tilhørende ROS-node som sender og mottar data.

4.1.2 WebSocket

En måte å overføre data mellom ulike applikasjoner over et nettverk på er WebSocket kommunikasjonsprotokollen. WebSocket-protokollen bygger på TCP (*Transmission Control Protocol*) og IP (*Internet Protocol*) og gjør det enkelt å sette opp langvarig full-duplex kommunikasjon mellom applikasjonene. Full-duplex betyr at det er mulig for begge applikasjonene å sende og motta data mellom hverandre samtidig. HTTP-protokollen fungerer ved at klienten sender forespørsel som blir besvart fra serveren som sender data som er forespurt i retur. For å få oppdaterte verdier må klient sende ny forespørsel, se Figur 11. Hver forespørsel til serveren blir behandlet som sin egen transaksjon. Dette siden HTTP-protokollen er en tilstandsløs protokoll. For å opprette WebSocket-kommunikasjon blir det først utført *HTTP-Handshake*. Deretter opprettes full-duplex kommunikasjonslinje mellom server og klient. Her kan klient bli oppdatert så raskt det er endringer som skjer på data i serveren, og klient kan samtidig holde data oppdatert på serversiden. For å avslutte WebSocket-kommunikasjonen kan enten klient eller server avslutte forbindelse. Den vil også avslutte om det blir brudd mellom klient og server.



Figur 11: Illustrasjon av WebSocket- og HTTP-kommunikasjon over tid

4.1.3 Oppbygning av webapplikasjon

Webapplikasjoner er ofte utviklet ved bruk av HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*) og JavaScript. Disse tre teknologiene har ulike funksjoner og jobber sammen for å danne komplette nettsider.

4.1.3.1 HTML

Hypertext Markup Language brukes til å definere strukturen på nettsiden. Her blir det definert hvor på nettsiden tekst, bilder eller andre elementer skal plasseres og organiseres. HTML er ikke et programmeringsspråk i tradisjonell forstand, men et *markup*-språk som bruker HTML-*tag* for å gruppere elementer sammen og plassere dem på nettsiden. Figur 12 viser et eksempel på hvordan HTML-*tag* fungerer i praksis. Her defineres paragrafer mellom start-*tag* « » og slutt-*tag* « ». Det forteller at alt mellom disse tilhører samme paragraf.

Figur 12: Eksempel på HTML (t.v) og hva som vises på nettside (t.h)

4.1.3.2 CSS

Cascading Style Sheets forkortet til CSS brukes for å bestemme hvordan elementene på nettsiden skal vises. Det kan være enkle ting som å endre farger, skrifttype og skriftstørrelser. Eller mer avanserte ting som at nettsiden flytter på elementer om den vises på små skjermer, som på smart telefoner, i forhold til om det er på stor PC-skjerm. Figur 13 viser et eksempel på hvordan endringer av utseende på HTML eksempelet i Figur 12 kan gjøres. Regler skrives i CSS for hvordan HTML-*tag* skal se ut. I HTML eksempelet er det to paragrafer. Begge disse får ett innrykk på 20 piksler. Det er fordi CSS definerer regler for alle paragrafer med *tag* «p» som sier at disse skal ha ett innrykk. De skal også ha farge definert med kode, som tilsvarer mørk grå farge.

Figur 13: Eksempel på hvordan CSS kan endre utsende på nettsiden i Figur 12

Ved å kombinere HTML og CSS er det mulig å lage statiske nettsider som kan inneholde tekst, bilder og annen informasjon. Typiske statiske nettsider kan være aviser eller blogger med artikler og innlegg, men slike statiske sider vil ikke ha interaktive eller dynamiske funksjoner. Verken HTML eller CSS er programmeringsspråk, de kan ikke behandle variabler eller funksjoner. Programmeringsspråket JavaScript kan bli brukt for webapplikasjoner og har god integrasjon med HTML og CSS.

4.1.3.3 JavaScript

For å gjøre webapplikasjonen interaktiv og dynamisk blir JavaScript ofte brukt. Mens HTML brukes til å strukturere innholdet og CSS brukes til å definere utformingen til siden, gjør JavaScript det mulig å legge til funksjonaliteter som animasjoner, brukerinteraksjon og oppdatering av innhold uten å måtte laste siden på nytt. JavaScript kan kjøres direkte i nettleser og trenger derfor ikke ekstra programvare.

Figur 14 viser et eksempel på bruk av JavaScript til å gjøre interaksjon med nettside. I HTML defineres en knapp og når den blir trykket på vil den kalle JavaScript-funksjonen. Denne funksjonen oppdaterer alle HTML-element med «h2»-*tag* som har id «antallTrykk» til å vise hvor mange ganger knappen har blitt trykket på av bruker. Denne enkle nettsiden er nå dynamisk siden den bruker funksjonen (knappTrykket) og variabelen (antall). Dette vil ikke være mulig å gjøre ved å bare bruke HTML og CSS, noe som viser hvorfor et programmeringsspråk som JavaScript kan være nyttig i en webapplikasjon.

Figur 14: Eksempel på hvordan JavaScript kan gjøre endring på nettside basert på brukerinput

4.1.4 JavaScript bibliotek

Et kode-bibliotek er samlinger med kode som løser et eller flere problemer. Det er flere biblioteker skrevet i JavaScript språket som har åpen kildekode og kan fritt brukes i egne applikasjoner. Noen problemer i denne oppgaven kan løses med disse bibliotekene. Bruk av ferdige løsninger fører til at det spares tid på å skrive koden selv, og mest sannsynlig blir applikasjonen mer stabil siden biblioteker

som regel er skrevet og vedlikeholdt av erfarne utviklere og er brukt i andre applikasjoner. Dette gjør at feil ofte er funnet, og at det finnes løsninger på problemer andre utviklere har støtt på.

4.1.5 React

React er et bibliotek med åpen kildekode utviklet av Meta [27], tidligere Facebook. Det er utviklet for å bygge brukergrensesnitt og introduserer også noen nye konsepter som kan velges å ta i bruk.

Komponenter er et av konseptene som introduseres og er sentralt i React. Mindre elementer blir programmert hver for seg i applikasjonen. Deretter blir disse elementene plassert på siden. Hver komponent er i hovedsak en JavaScript-funksjon som tar inn data som kalles *props* og returnerer det som skal vises på skjermen. Bruken av komponenter kan gjøre koden mer organisert og gjenbrukbar. Komponenter kan også bruke andre komponenter, som gjør det mulig å bygge komplekse brukergrensesnitt selv om den bare består av enkle komponenter.

States og *props* er hovedkonsept for datahåndtering i React komponenter. *States* er variabler i komponentene som kan forandre seg. *Props* er forkortelse for *properties* og er data fra foreldreklasse som blir satt inn i underliggende klasse.

Det er også mulig å benytte syntaksutvidelse for JavaScript som heter JSX. Denne gjør det mulig å kombinere HTML elementer i JavaScript kode. Dette for å gjøre koden mer lesbar og enklere å skrive. Når et utrykk i JSX møter et HTML-*tag* tolkes kode som HTML, og når den møter krøllparentes tolkes det som JavaScript utrykk.

React har også innebygget støtte for *event handling*. Dette for å gjøre handlinger når hendelser blir detektert. En hendelse kan være et museklikk, et element blir dratt på skjermen, et tastetrykk eller lignende.

Det finnes også biblioteker med ferdigutviklete React-komponenter. Et av disse er React bootstrap [28] som har elementer som knapper, advarsler med mer. Kombinert med tilhørende stiler og utforminger fra stilbiblioteket Bootwatch [29] trengs det lite CSS for et prosjekt.

4.1.5.1 Create React App

Meta har utviklet et verktøy for enkelt å kunne starte et nytt prosjekt kalt Create React App [30]. Ved å skrive inn en enkelt kommando i terminalen genereres mappestruktur, grunnleggende kode for webapplikasjonen og flere biblioteker importeres.

For å kunne utføre kommandoene og kjøre koden trengs Nodejs som er et JavaScript-*runtime enviroment* [31]. Dette betyr at Nodejs skaper et miljø hvor JavaScript kan kjøres uavhengig fra nettleser. NPM (*Node Package Manager*) er et pakkesystem som også er del av Nodejs. NPM gjør det mulig å installere pakker raskt og enkelt i terminal. Pakker brukt i applikasjonen legges til filer som inneholder lister over alle pakker brukt i applikasjonen, og hvilken versjon av pakkene som brukes. Dette gjør det lettere for andre å åpne prosjektet og installere alle utenforstående pakker ved å bruke en enkel kommando.

4.1.6 Interaktivt kart

For å kunne vise et kart er det fordelaktig å bruke et JavaScript-bibliotek. Det er flere biblioteker som allerede er utviklet for å kunne generere interaktive kart på nettsider.

4.1.6.1 Google Maps JavaScript API

Google har laget en løsning for implementasjon av interaktive kart kalt Maps JavaScript API [32]. Tjenesten er utviklet av et stort firma og har mye tilgjengelig dokumentasjon og kursmateriell. Google har også kart som blir oppdatert jevnlig og kan vise satellittbilder av området.

Det brukes API (*Application Programming Interface*) teknologi for å hente kart og funksjoner fra Google servere til applikasjonen. API fungerer ved å sende HTTP-forespørsel fra applikasjonen og server returnerer det som blir spurt etter. For eksempel et kart over et spesifikt område.

Bakdelen med å bruke denne løsningen er at det er en betalt tjeneste. Er det under et gitt antall forespørsler til server per måned er det gratis, men det må fremdeles registreres betalingsmetode.

4.1.6.2 Mapbox

Mapbox leverer også interaktive kart [33]. I likhet med Google er dette et firma som har laget et produkt der det er mye dokumentasjon, kursmateriell og guide for feilsøking. Mapbox har ikke tilgang til sin egen karttjeneste slik som Google med Google Maps, men bruker karttjeneste med åpen kildekode kalt Open Street Map [34]. Kartdata består av noen åpne datasett fra nasjonale karttjenester, samt data samlet av tusenvis av individuelle bidragsytere.

Mapbox leverer også et produkt og tjenester som det kreves betaling for. I dag gir Mapbox flere gratis forespørsler enn Google, men det er fortsatt en betalt tjeneste.

4.1.6.3 Leaflet

Leaflet er et interaktivt-kartprosjekt med åpen kildekode, startet av ukrainske Volodymyr Agafonkin i 2010 [35]. Ifølge data fra Leaflet sin GitHub er det i dag nesten 800 personer som har bidratt til prosjektet og over 175 000 prosjekter som bruker Leaflet [36].

Siden kildekoden er åpen, er det også gratis å bruke. Det er et JavaScript-bibliotek som gjør at bare kartdata må hentes ved bruk at API og ikke funksjoner. Leaflet er også mindre avansert i bruk enn for eksempel Mapbox, men mangler også funksjonaliteter sammenlignet med Maps JavaScript API og Mapbox. Siden Leaflet har åpen kildekode er det derimot mulig å legge til egne funksjoner. Siden det er mange brukere er det også utviklet løsninger utenfor Leaflet-biblioteket som kan brukes.

Det har også blitt produsert et bibliotek kalt React Leaflet som gjør det enkelt å implementere Leaflet i React applikasjoner [37]. Dette biblioteket erstatter ikke Leaflet, men genererer et React objekt som kan gjøres endringer på ved å bruke det originale Leaflet biblioteket.

Leaflet kan bruke flere karttjenester for å hente kartdata. Den mest vanlige er Open Street Map som også Mapbox bruker, men andre tjenester kan også brukes.

4.1.6.4 OpenLayers

OpenLayers er også et interaktivt-kartprosjekt med åpen kildekode [38]. Som Leaflet er det også et JavaScript-bibliotek som bruker tredjepart karttjeneste for å implementere kartdata. OpenLayers har mer innebygget funksjonalitet enn Leaflet, men har ikke et bibliotek med like mange brukere for støtte mot React.

4.1.7 Robot Web Tools og ROS

Robot Web Tools har flere nyttige JavaScript-bibliotek som kan brukes i dette prosjektet. Robot Web Tools bruker åpen kildekode og jobber for å utvikle ulike løsninger for å kunne styre og monitorere roboter over nettverk ved å bruke JavaScript og ROS.

4.1.7.1 WebSocket og ROS

ROSbridge suite [39] kan installeres på roboter som kjører ROS. Den vil da opprette WebSocket-server som kan starte kommunikasjon med andre applikasjoner på nettet ved bruk av JavaScript. Det gjør det mulig å utvikle en webapplikasjon som kan hente data fra og sende til ROS.

4.1.7.2 Roslibjs

ROSLibjs er et JavaScript bibliotek som gjør det mulig å opprette ROSLib instanser i webapplikasjoner. Disse instansene kombinert med tidligere nevnte ROS-bridge-Websocket gjør det mulig for JavaScript-applikasjoner å få tilgang til data i ROS. Den kan også publisere data på *ROS-topic*, bruke *ROS-service* og kommunisere med *ROS-action*.

4.1.8 Konklusjon - Brukergrensesnitt

Selv om ingen i prosjektet har bakgrunn fra HTML, CSS og JavaScript programmering ser det ut til at fordelene med å bruke disse strukturene og programmeringsspråket veier opp for tidligere erfaring med Python. Det ser ut til å være flere ressurser for utvikling av webapplikasjon tilgjengelig for disse, og de er også designet for webutvikling.

Verktøy utviklet fra Robot Web Tools-prosjektet ser ut til å gjøre implementering enklere enn ved bruk av andre verktøy. Disse vil trolig også redusere tidsbruk ved utvikling.

React-biblioteket vil gjøre læringskurven brattere enn om bare HTML, CSS og JavaScript brukes. Det ser likevel ut som fordelene med redusert behov for CSS og mulighetene til å dele koden opp i klare blokker vil gjøre applikasjonskoden mer oversiktlig.

Som interaktivt kart ser Leaflet ut som det rette valget for denne applikasjonen. Dette fordi den er gratis å bruke, og den har et bibliotek som gjør det enklere å implementere med React. OpenLayers ble også vurdert, men det ser ut til å være mer avansert å implementere og har et mindre antall brukere som kan føre til redusert tilgang til støtte og ressurser.

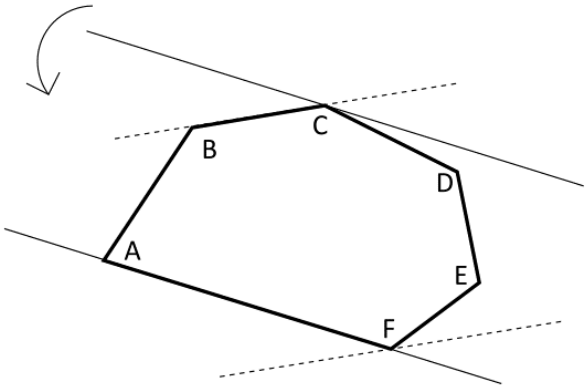
4.2 Baneplanlegging

For å kunne oppnå ønsket navigering, som definert i prosjektet, er det nødvendig å bruke en algoritme som planlegger bane for å dekke et helt område. For å velge hvilken algoritme som skal brukes er det ønskelig å sammenligne under følgende kriterier. Det er ønskelig å ha effektiv bane, som ikke krever for mye datakraft å kalkulere. Vesentlige faktor som påvirker effektiviteten til banen er hvor mye roboten må svinge, ettersom den vil bremse opp før svingen, og akselerere etterpå. Dette tar tid og bruker mer energi. Det er også ønskelig å velge en løsning som kan brukes ved andre aspekter av prosjektet, ikke bare gressklipping.

4.2.1 Rotating Caliper Path Planning

Rotating Caliper Path Planning er baneplanleggingsalgoritmen som baserer seg på den matematiske metoden *Rotating Calipers*, som også kan kalles Roterende Skyvelære på norsk.

Metoden involverer bruk av to parallelle linjer som omslutter hele polygonet. Først plasseres linjene på hvert sitt punkt, deretter roteres linjene rundt polygonet samtidig som de holdes parallelle. Ved å observere når en av linjene passerer et hjørne, finner vi såkalte antipodale par. Disse er to parallelle linjer der den ene følger siden av polygonen og det andre danner tangent fra ett av hjørnene. Se Figur 15. Når de parallelle linjene har fullført en runde rundt polygonen vil alle antipodale par blitt identifisert. Denne teknikken var først diskutert i Michael Shamos sin Ph.d. avhandling, der han også utviklet en algoritme for å finne alle antipodale par i polygoner[40, s. 78–80].



Figur 15: Illustrering, generering av antipodale par

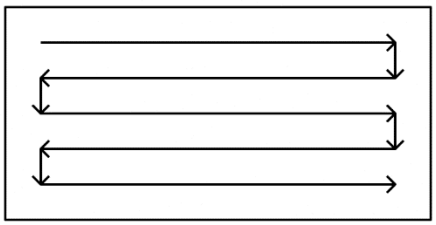
Det er ønskelig å finne de parene som har kortest distanse imellom seg, da dette vil gi den banen med færrest svinger og derfor være den mest effektive [41]. Deretter genereres banen som går fram og tilbake parallelt med linja i polygonet.

Denne teknikken vil ikke kunne brukes på områder som er konkave, da den banen som går fram og tilbake ikke vil fungere i slike områder.

4.2.2 Boustrophedon Cellular Decomposition

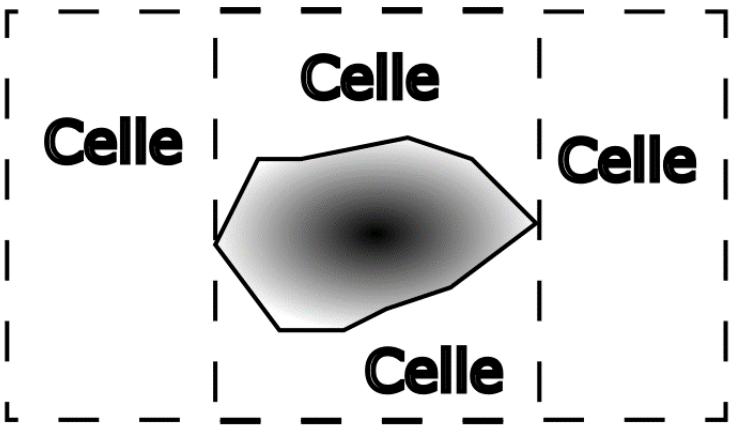
Boustrophedon Cellular Decomposition baserer seg på å splitte opp området i celler, sånn at hele arealet kan bli beskrevet som et sett med polygoner. Deretter blir det kalkulert fram-og-tilbake bevegelser inne i hver av de cellene, så når hver celle har blitt dekket, så har også hele området blitt dekket [42]. Denne baneplanleggeren tar ikke hensyn til kostnadene av bevegelsene, og risikerer å ta flere, unødvendige linjer for å forsikre seg dekning av området. Dette skjer spesielt i overgangen mellom cellene.

Navnet Boustrophedon var først brukt i 1699 og betyr «Som oksen går» og beskriver måten å pløye et felt på, der oksen går i rett linje over åkerer for så å snu seg og gå en ny rett linje over åkeren slik at den blir dekket helt [42]. Se Figur 16 for eksempel.



Figur 16: Illustrasjon av Boustrophedon bane

Cellene blir konstruert ved å sveipe ei linje over området som blir brukt, hvor det dannes nye celler for hvert hinder som blir oppdaget. Det blir også dannet nye celler når hinderet ikke lenger er i veien. Se for Figur 17 illustrasjon av denne konstruksjonen.



Figur 17 Konstruksjon av Boustrophedon celler

Når alle cellene er blitt konstruert gjenstår det å lage fram-og-tilbake bevegelsen for å definere banen inne i cellene. Denne algoritmen fungerer godt når det er store områder som ikke er tilgjengelige og må planlegges rundt, men den tar ikke hensyn til effektiviteten til banen.

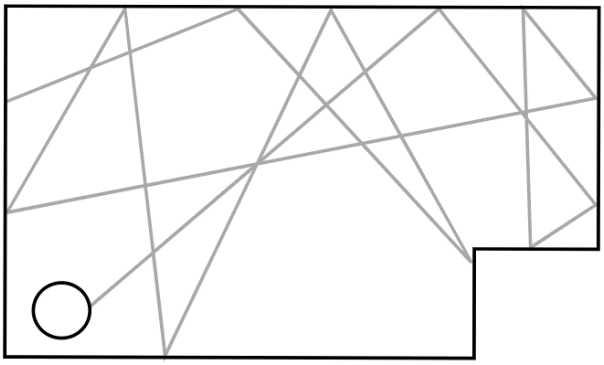
4.2.3 Baneplanleggere for robot støvsugere

Robot støvsugere er vanlige roboter, de er gjerne differensielt styrt som skal dekke et helt område basert på forskjellige algoritmer. For å oppnå flere av de lettere algoritmene trenger roboten kun å ha støtfanger sensor, som oppdager når roboten treffer hindringer. De trenger heller ikke en forhåndsdefinert plan, og de krever lite datakraft, såkalt reaktiv navigasjon.

4.2.3.1 Tilfeldig Bane

Tilfeldig Bane Algoritme er et samlebegrep for teknikker av baneplanleggere som bruker et element av tilfeldighet. Den enkleste variasjonen av slike algoritmer er å kjøre helt til roboten treffer et objekt, for så å snu seg på stedet i tilfeldige tidsperioder før roboten kjører videre.

Den store fordelen med slike algoritme er at den krever særdeles lite kalkuleringer og planlegging fra roboten sin side, og er derfor billig å gjennomføre. Ulempen er at det er ingen garanti for at hele området blir dekket. Figur 18 Illustrerer en slik tilfeldig bane.

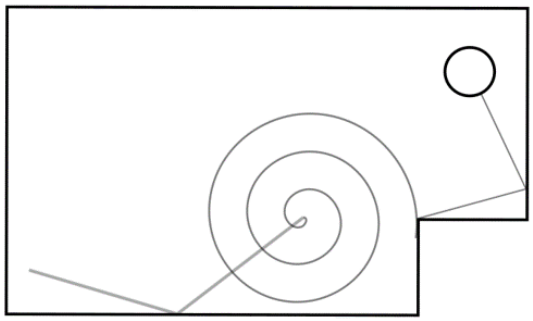


Figur 18 Illustrasjon av tilfeldig bane

4.2.3.2 Spiral Algoritme

Dette er variasjoner av tilfeldige baner, men med store nok forskjeller til at den er verdt å nevne. Roboten kjører tilfeldig rundt til den oppdager at den har tilfredsstillende med ledig område rundt seg, det kan være med data fra en LIDAR eller andre sensorer, for så å begynne å kjøre i sirkler. En differensialstyrt robot vil kunne utføre den bevegelsen lett, ved å gradvis øke hastigheten kun på det ene hjulet. Se Figur 19 for illustrasjon.

Tidligere arbeid viser at spiral algoritme var den tregeste til å dekke et område [43].

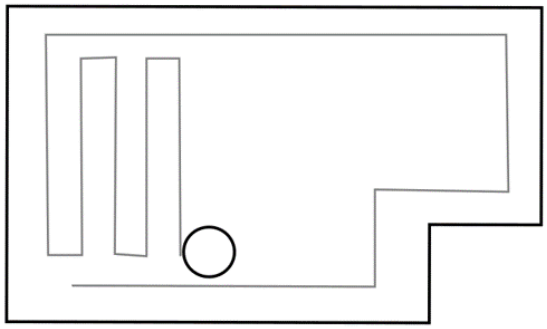


Figur 19 Illustrasjon av spiral bane

4.2.3.3 Slange og vegg følger algoritme

Slange bevegelser er når roboten kjører rett fram, for så å snu på stedet og kjøre i andre retningen. Små unøyaktigheter i enten sensor data eller skjevhet mellom hjulene vil potensielt over tid føre til store feilmarginer i banen som blir tatt.

For å gjøre algoritmen mer robust er det anbefalt å kjøre algoritmer som følger kanten til området for å få kunnskap over området som skal dekkes, det gjør også vanskelige områder, som hjørner, lettere å dekke [43]. Deretter brukes slange banen for å dekke det indre området. Figur 20 illustrerer en slik bane.



Figur 20 Illustrasjon av slange bane med vegg følging

4.2.4 Konklusjon

Å bruke de algoritmene som er ment for robot støvsugere vil ikke kunne møte alle utfordringene som prosjektet krever. Det å kun bruke reaktive teknikker vil heller ikke gjøre det mulig å planlegge stopp langs banen der det er ønskelig å gjøre andre arbeid samtidig. Det er heller ingen garanti at disse teknikkene dekker hele området, som er et absolutt krav til prosjektet.

Både *Rotating Caliper Path Planner* og *Boustrophedon Cellular Decomposition* gjør det mulig å planlegge baner som kan dekke hele områder i forveien, men det er kun *Rotating Caliper Path Planner* som tar hensyn til effektiviteten til disse banene.

Området som skal navigeres i vil være enten åpne åkere, eller mellom rader av busker eller trær. Der det er åkere vil det ikke være hindringer, og i de andre tilfellene vil det ikke være store områder som er utilgjengelige. Det gjør at fordelen med *Boustrophedon Cellular Decomposition* faller vekk.

Med tanke på de kriteriene som er blitt gitt faller valget av baneplanleggingsalgoritme på *Rotating Caliper Path Planner*.

4.3 Prosjektkritiske ROS-pakker

Rammeverket som blir brukt for å kontrollere og programmere den mobile basen som skal utføre alle oppgavene som er blitt definert, er ROS. Det er derfor verdt å undersøke hvilke pakker som allerede er ferdig utviklet.

4.3.1 Navigation

Navigasjonspakken inneholder verktøy for å lage, og utføre bevegelse for å forflytte den mobile basen til et gitt punkt, med gitt orientasjon. Den har noen hovedfunksjoner: kartlegging, lokalisering, planlegging og utføring av bevegelser.

Kartleggingsmodulen gjør det mulig å opprette modell av det fysiske miljøet rundt roboten, ved å bruke sensorer som samler inn data om omgivelsene. Det kan være LIDAR eller kamera som kan fange opp dybdeinformasjon. Dette kan senere brukes til å planlegge bevegelser rundt i rommet.

Lokaliseringsmodulen brukes til å estimere roboten sin posisjon og orientering i forhold til ett etablert kart. Dette gjøres ved å bruke odometer sensorer som oppdager bevegelse over tid, og kan derfor brukes til å lage et estimat av posisjon i forhold til ett startpunkt, og sensorer som tar inn ekstern informasjon fra området rundt, som LIDAR eller kamera. Ved å sammenligne disse sensordataene med kjente landemerker, kan roboten estimere mer nøyaktig posisjon i forhold til et kart.

Planleggingsmodulen kan brukes til å genere sikker og optimalisert bevegelsesbane for roboten fra startposisjon til ett gitt mål. Den tar hensyn til hindringer i miljøet, om det er et kart tilgjengelig, og begrensningene knyttet til robotens bevegelser.

Navigasjonspakken har også modul for å styre roboten, den brukes til å kontrollere robotens faktiske bevegelser langs den planlagte banen.

Disse forskjellige aspektene av navigasjonspakken kjøres ikke nødvendigvis alene, det er teknikker som bruker flere av de samtidig. Det er mulig å lage kart samtidig som det blir brukt for lokalisering. Dette kalles SLAM (*Simultaneous Localization and Mapping*) hvor roboten lager og oppdaterer et kart i sanntid.

4.3.1.1 Move Base

Navigasjonspakken har en ROS-action-server som heter «move base» som tar imot et mål som roboten skal navigere seg i. Det målet inneholder posisjon og orientasjon i et koordinatsystem definert i forhold til roboten.

4.3.2 Robot Localization

Robot Localization blir brukt for å estimere posisjon og orientasjon av roboten i et gitt miljø ved å implementere et Kalmanfilter. Dette filteret gjør det mulig å kombinere sensordata fra flere forskjellige sensorer for å få bedre estimere robotens tilstand. Det er også en node tilgjengelig for transformasjon av GPS-posisjoner til koordinater i forhold til roboten, som gjør det mulig å navigere basert på GPS-informasjon.

Robot Localization pakken gjør det mulig for utviklere å implementere disse metodene for å oppnå presisjonsnavigasjon ved hjelp av nøyaktige posisjonsestimeringer [44].

4.3.3 TEB Local Planner

TEB (*Timed Elastic Band*) *local planner* er en navigasjonsplanlegger-pakke som er designet for å genere smidige bevegelsesplaner for mobile roboter i dynamiske omgivelser.

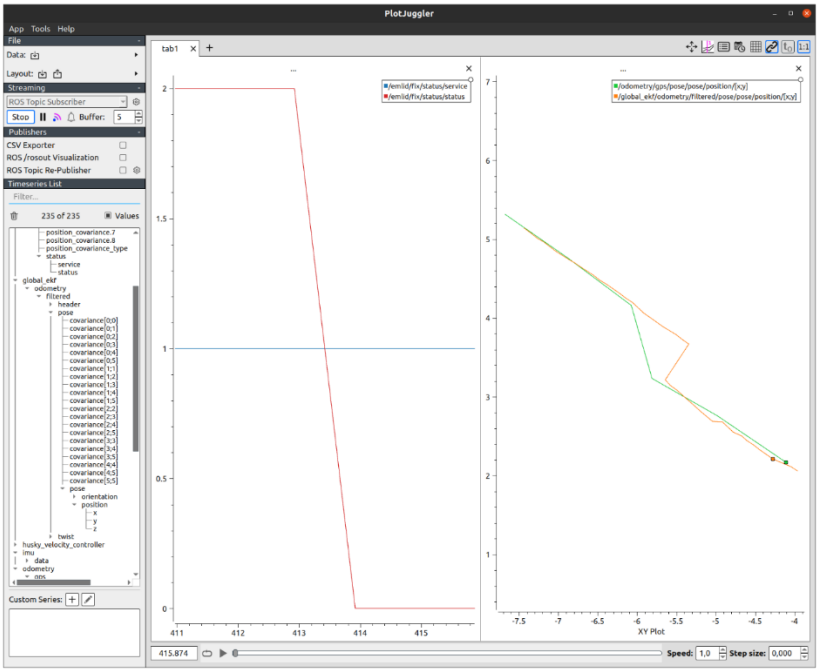
TEB planleggeren bruker konseptet med elastiske band, som omtalt i kapittel 3.1, for å representere mulige bevegelsesbaner. Disse blir brukt i samarbeid med bevegelsesplanleggeren «/move_base» til Navigation pakken til ROS når den planlegger, og utfører banen for å oppnå dynamisk, sikker, og effektiv utførelse av bevegelse.

Ved å utnytte elastiske bånd kan TEB-planleggeren tilpasse bevegelsesbanen basert på endringer i omgivelsene og eventuelle hindringer som oppstår.

4.3.4 Visualiseringsverktøy

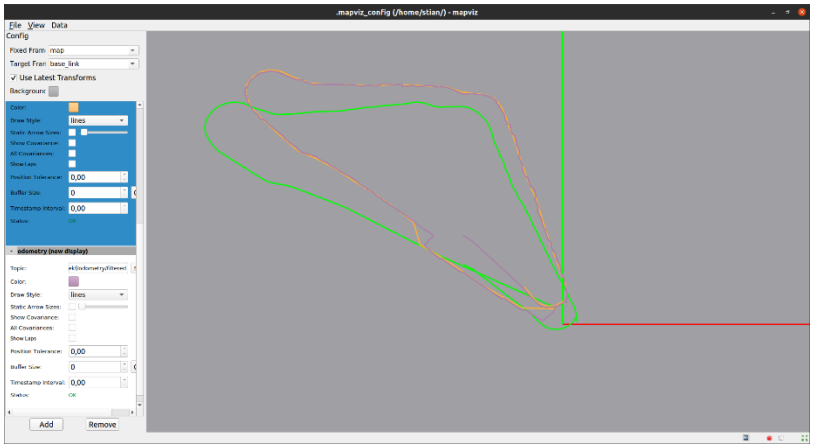
ROS-pakkene PlotJuggler og MapViz tilbyr verktøy for visualisering av data. Der RViz visualiserer for det meste i sanntid, er disse verktøy for å visualisere data over tid.

PlotJuggler gir grafisk visning av data, og er spesielt nyttig for tidsrelaterte data som logger og sensordata [45]. Noen av bruksområdene kan være å visualisere akselereringen til en mobil base eller observere nøyaktig når et signal blir sendt. PlotJuggler er fullstendig kompatibelt med ROS, noe som gjør det mulig å analysere og visualisere data som blir sendt på *ROS-topic* [45]. Figur 21 viser programmet i praksis.



Figur 21: Eksempel på bruk av PlotJuggler

MapViz er utviklet for å vise to-dimensjonal data, som kart informasjon eller bane data [46]. Med sine mange verktøy er det mulig å vise flere forskjellige datatyper fra *ROS-topic*, som odometri data og GPS-punkt. Se Figur 22 for eksempel av baner plottet med odometri data.



Figur 22: Eksempel på bane plottet i MapViz

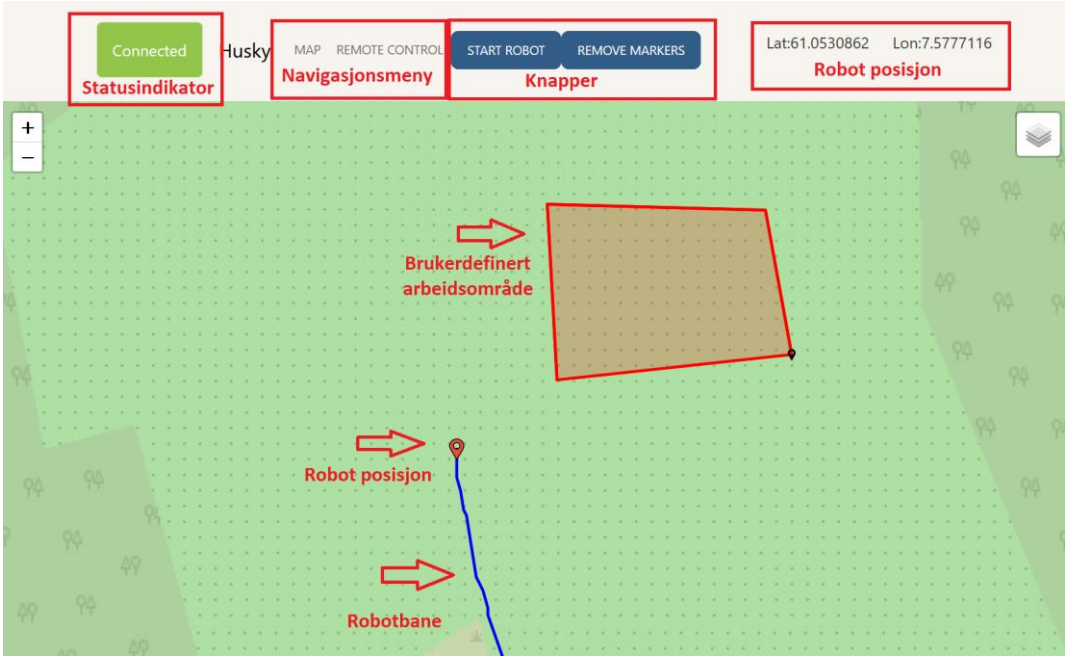
5 Realisering av valgt løsning

Den endelige løsningen består av et brukergrensesnitt som er en webapplikasjon. Denne henter område på et kart der den mobil basen skal operere fra brukeren. Denne dataen blir lagret til fil ved hjelp av en *ROS-service* som webapplikasjonen kommuniserer med. Denne dataen blir så hentet av en *ROS-action* som leser GPS-punkter fra fil. Når brukeren sender ett mål til action serveren, blir det definert hvilken handling som skal bli tatt. De to hovedoppgavene til *ROS-action* serveren er å flytte seg til GPS-punkt som er definert av brukeren, eller å planlegge og utføre dekkende baneplanlegger.

5.1 Brukergrensesnitt

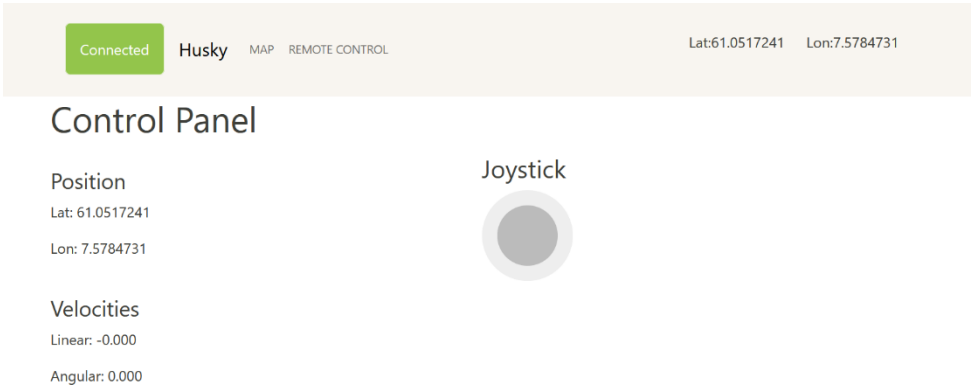
Brukergrensesnittet er laget for at brukeren skal definere et arbeidsområde til roboten. Dette ved å vise et interaktivt kart til brukeren. Brukeren kan merke et område i kart og deretter starte roboten.

Figur 23 viser utforming av brukergrensesnitt. Statusindikator er grønn om applikasjonen har kontakt med ROS-WebSocket, og rød om den ikke har kontakt. Navigasjonsmenyen viser oversikt over andre sider i applikasjonen. En knapp starter roboten og en annen sletter markører plassert av bruker på kartet. Oppe i høyre hjørne vises koordinater roboten befinner seg på, og rød markør viser posisjonen til roboten i kartet. Det vises også i kartet hvor roboten har vært med den blå linjen. Brukere kan også definere arbeidsområdet til roboten som vil vises som en rød polygon.



Figur 23: Viser funksjoner i webapplikasjonen

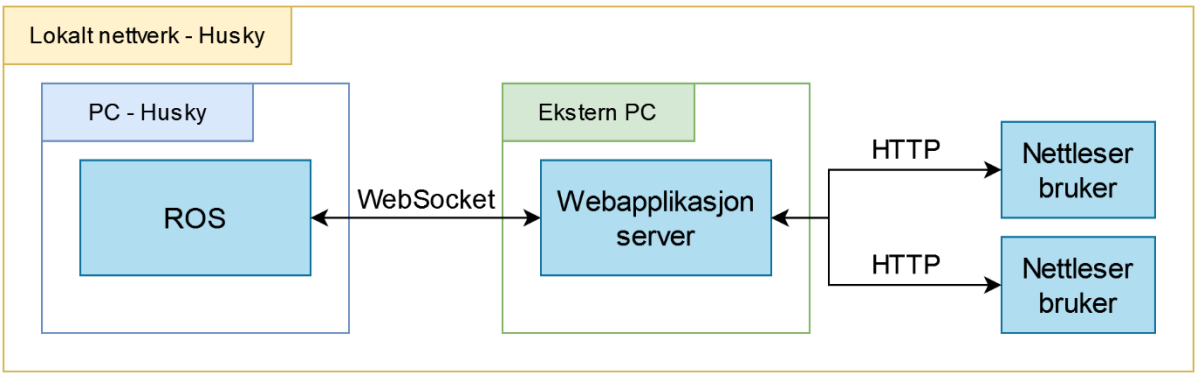
Under siden merket med «Remote Control» er en joystick som kan kjøre roboten direkte fra nettleser, vist i Figur 24. Den viser også hastigheter roboten har. Den linjere hastigheten er hastighet roboten har, framover om positiv, eller bakover om den er negativ. Angulærhastighet er hastigheten roboten roterer med. Denne er positiv når roboten roterer mot venstre og negativ ved rotasjon mot høyre.



Figur 24: Kontroll panel for å styre robot fra nettleser

5.1.1 Nettverksoppbygging

Serveren kjører på ekstern PC som er koblet til nettverker til Husky robotbasen. Serveren kommuniserer over WebSocket til ROS instansen på robotbasen. Brukeren som har enhet tilkoblet det lokale nettverket til Husky kan åpne nettleseren til gitt IP adresse og port for å få tilgang til brukergrensesnittet. Se illustrasjon i Figur 25.

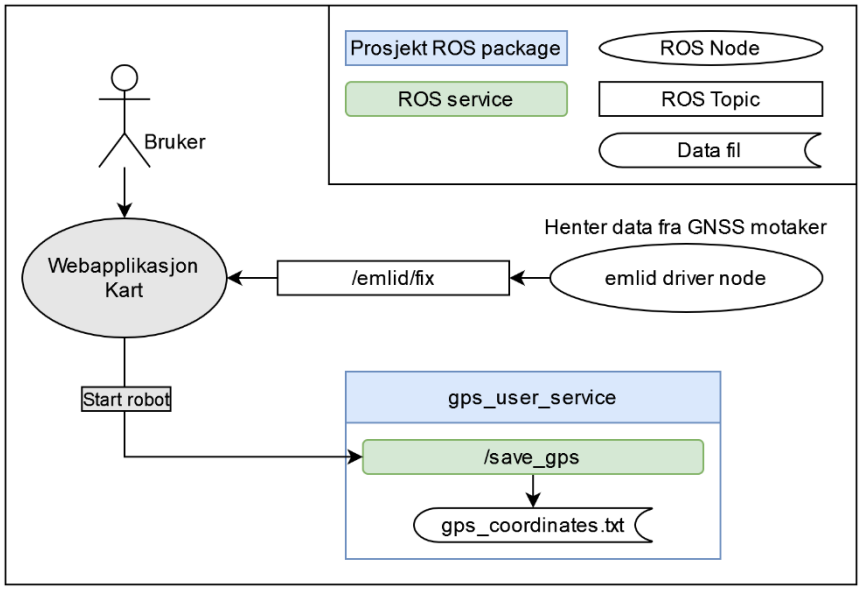


Figur 25: Nettverksoppbygging mellom Husky og bruker av webapplikasjon

Når server blir kjørt kan port bli definert og IP adresse vil være maskinens IP på lokalt nettverket. Det er denne IP og port kombinasjonen brukere kobler seg til igjennom nettleser. For nærmere instruksjoner se applikasjonens GitHub side [47].

5.1.2 Kommunikasjon mellom brukergrensesnitt og ROS

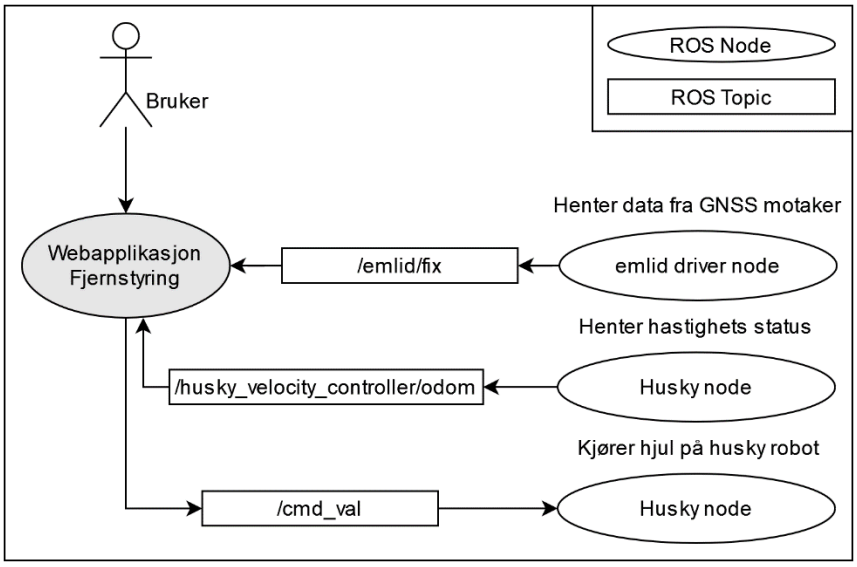
Kart applikasjonen trenger GPS-data fra roboten for å vise hvor den er på verdenskartet. Denne blir hentet fra et *ROS-topic* som noden til GPS-mottakeren publiserer posisjonsdata til, se Figur 26. Det siste datapunktet som er blitt mottatt bestemmer plassering for markør som representer roboten i kartet. Dette datapunktet legges også til i en liste med punkt for å tegne banen roboten har tatt på kartet. Når knappen for å starte roboten trykkes vil applikasjonen sende listen med posisjonskoordinater til tilhørende *ROS-service*. Disse vil bli lagret til fil i samme pakke som *ROS-service* er lokalisert.



Figur 26: Kommunikasjon mellom kart applikasjon og ROS

ROS-pakken «gps_user_service» er laget for dette prosjektet og pakken finnes på GitHub [48]. Funksjonen den har er å lagre data slik at den er tilgjengelig for andre ROS-pakker. I pakken er kode for å teste service og i GitHub ligger det også pakke med kode for å hente ut data som er lagret.

Siden for fjernstyring i applikasjonen kommuniserer mer aktivt gjennom ROS. Figur 27 viser hvilke *ROS-noder* og *ROS-topic* som brukes for å sende og motta data mellom applikasjonen og ROS. Den henter ut posisjonsdata fra GPS-mottaker og hastighets data fra robotbasen. Applikasjonen sender også data til robotbasen for å kjøre hjul når joystick blir operert.

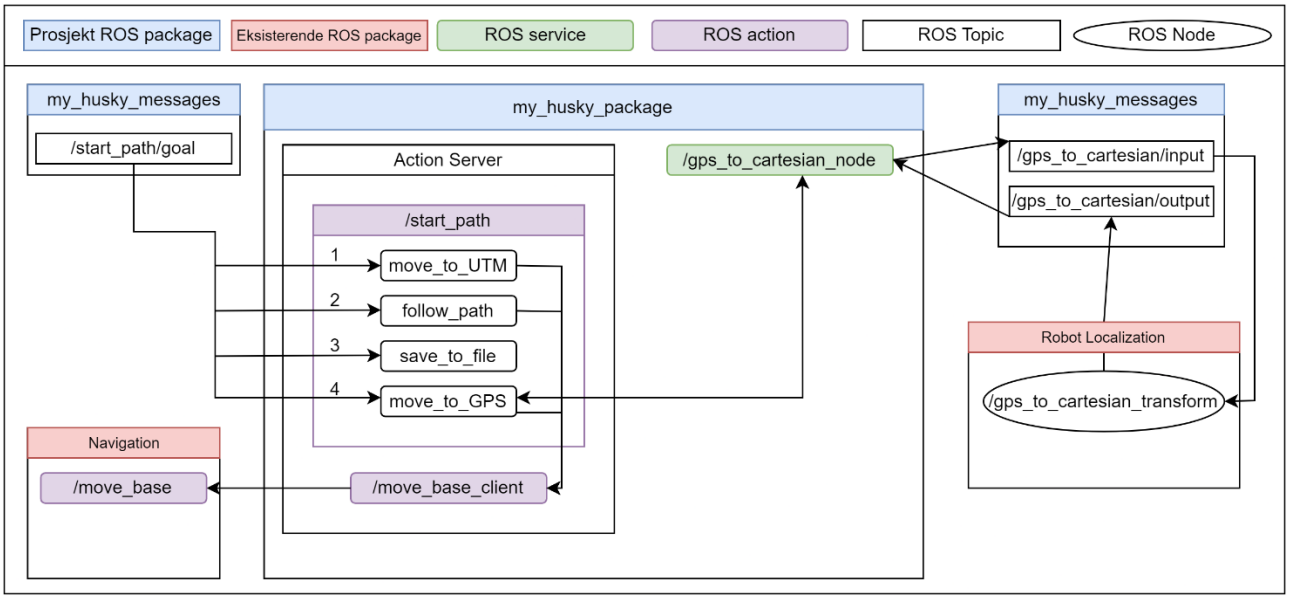


Figur 27: Kommunikasjon mellom fjernstyringsapplikasjon og ROS

5.2 Prosjektutviklet ROS-pakker

For å håndtere kommunikasjon mellom flere noder i ROS har det blitt utviklet en *ROS-action* for kontroll over systemet. Hovedmålene til *ROS-action* serveren er å navigere til brukerdefinerte GPS-punkt, og utføre den planlagte banen som dekker område angitt av brukeren. Det er også blitt laget en *ROS-action* for å håndtere kommunikasjon til *ROS-action* «move_base» som er tilbudt igjennom navigasjonspakken til ROS. Denne *ROS-action-klienten* sender posisjon, orientering og hvilket koordinatsystem det skal navigeres etter. Pakken dette programmet ligger på er tilgjengelig igjennom GitHub, under mappen «my_husky_package» [49].

Figur 28 viser den forenklede oppbyggingen av action serveren og dens oppgaver.



Figur 28: Oppbygging av action server

5.2.1 My husky messages

Det er blitt laget en pakke der alle meldingene som er spesielt laget for denne applikasjon er definerte, denne pakken er «my_husky_messages». Denne pakken er tilgjengelig på GitHub [49].

Meldingen som er på «/start_path/goal» *ROS-topic* er egendefinert *action*-melding, denne meldingen er definert med ett tall som mål, og tekststreng i både tilbakemeldingen og resultatmeldingen.

Input meldingen fra «/gps_to_cartesian_node» inneholder GPS-koordinatene breddegrad og lengdegrad, definert som «Latitude» og «Longitude». Output meldingen inneholder disse omgjort til kartesiske i formen «x_axis» og «y_axis».

5.2.2 Move to UTM

Denne handlingen flytter huskyen til brukerdefinerte punkt med UTM-koordinater. Den starter med å lese GPS-punktene fra filen som er generert fra brukergrensesnittet, «gps_coordinates.txt», konverterer disse til UTM-koordinater gjennom et Python bibliotek [50], og bruker *ROS-action* «move_base» for å forflytte huskyen til disse punktene. Mer informasjon om UTM-koordinater er i kapittel 3.2.3.

5.2.3 Follow path

Denne handlingen leser banen som er blitt planlagt fra filen, «sweeping_path.txt», og sender disse punktene til *ROS-action* «move_base». For å generere banen er det nødvendig å bruke programmet «PathPlanner.py», der settes manuelt inn bredden på banen, punktene som definerer området og bunnlinje. Dette programmet vil lage filen «sweeping_path.txt».

5.2.4 Save to file

Denne handlingen leser GPS-punktene fra fil, konverterer til UTM-koordinater, og lagrer det til fil. Disse er brukt for å lage banen. For mer informasjon angående konvertering til UTM-koordinater, se kapittel 3.2.3.1.

5.2.5 Move to GPS

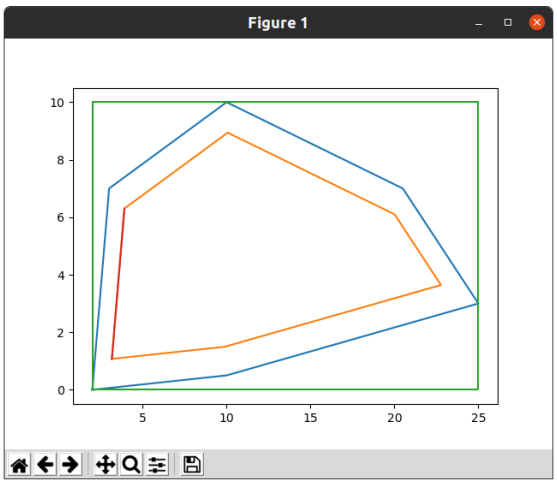
Denne handlingen flytter Huskyen til punkt gitt ved GPS-koordinater. Dette oppnås ved å bruke *ROS-service* «/gps_to_cartesian_node» som sender GPS punkt til Robot Localization sin *navsat transform ROS-node*, «/gps_to_cartesian_transform». Denne *ROS-noden* er opprettet for å konvertere GPS-punktene til kartesiste-punkter i henhold til roboten sitt eget koordinatsystem. De resulterende kartesiske-koordinater blir sendt til *ROS-action* «move_base».

5.3 Baneplanlegging

Baneplanleggingsalgoritmen har blitt utviklet med *Rotating Caliper Path Planning* som utgangspunkt, den er lagret som «PathPlanner.py» og laget som klasse i Python. Den tar som input hjørnene til ett polygon og linjen til ett antipodalt par, som er beskrevet i kapittel 4.2.1 og gir ut liste over alle punktene i banen. Baneplanleggeren tar også hensyn til bredden av banen. Dette programmet er tilgjengelig gjennom GitHub [49].

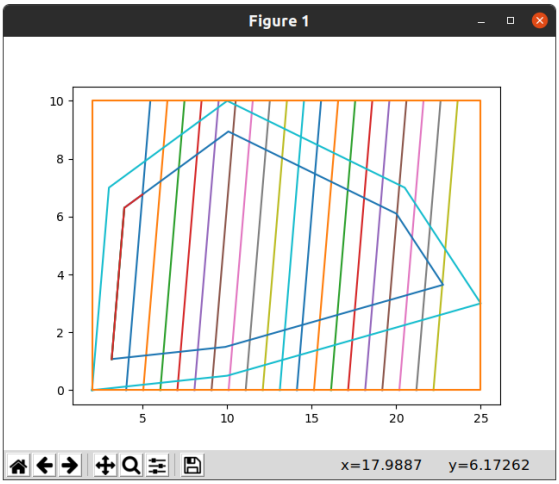
Algoritmen fungerer med å finne parallelle linjer som strekker seg fra grunnlinjen, som er den angitte inngangslinjen, ut til det motstående hjørnet med distanse mellom seg lik bredden til banen roboten skal ta. Deretter finner den krysningspunktene, som er endepunktene for frem-og-tilbake bevegelsen.

Først blir det dannet et rektangel rundt polygonen som blir brukt til å skape de parallelle linjene, deretter blir polygonen krympet med halvparten av bredda til banen, se Figur 29. Dette er for å hindre roboten i å kutte for bredt.



Figur 29: Blå er originale området, oransje er det krympa området, grønn er rektangelet rundt

Deretter blir det dannet parallelle linjer som går fra ytterpunktene, som definert av det grønne rektangelet i Figur 29, der krysningspunktene til det indre polygonen er endepunktene i frem-og-tilbake bevegelsen.

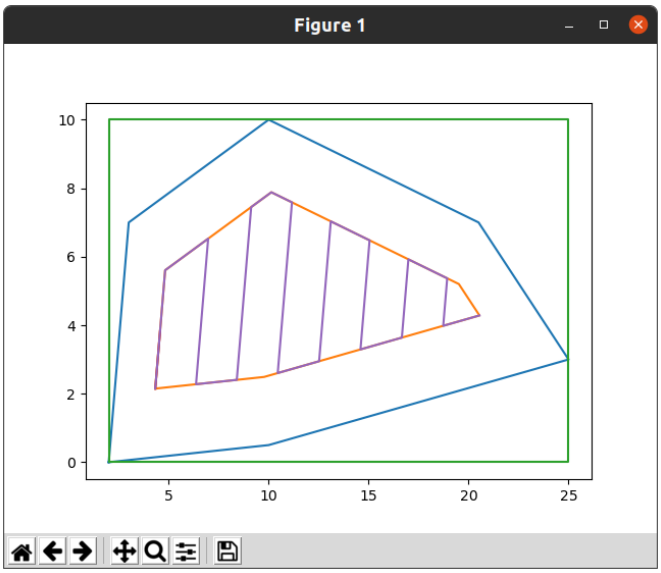


Figur 30: Parallelle linjer fra grunnlinje til motstående hjørne

Krysningspunktene til det indre polygonen blir lagret i liste, og bevegelsen langs det indre polygonen sin kant blir funnet. Det gjøres ved å ta krysningspunktet til neste linje og sjekke om det er kanter på den linjen, om en kant blir funnet så legges den til i listen over koordinater som banen skal innom. Deretter fortsetter algoritmen med neste linje til alle linjene er tatt.

Figur 31 viser den ferdig planlagte banen, denne er planlagt med bredere bane. Dette er gjort for å bedre illustrere hvordan kanten på det indre polygonen blir opprettholdt.

Denne banen blir så lest til fil som *ROS-action* serveren kan bruke til å sende til robotbase, «sweeping_path.txt».

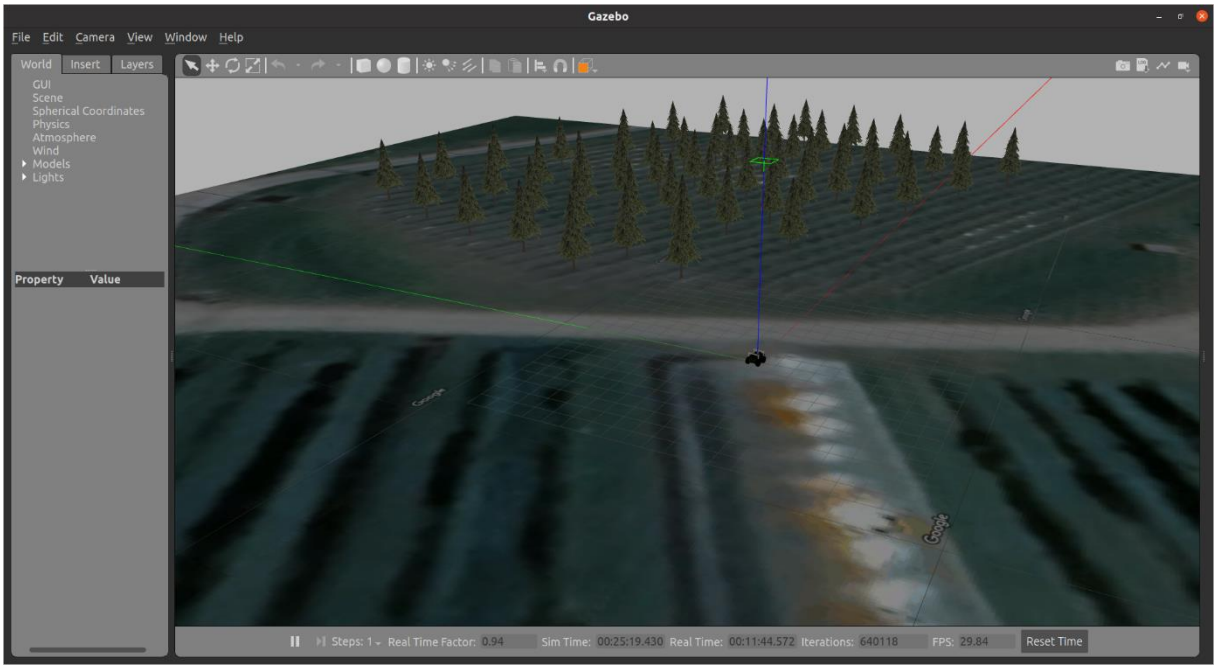


Figur 31: Planlagt bane

5.4 Digital tvilling i Gazebo

Det er også blitt utviklet en digital tvilling av systemet i simuleringsverktøyet Gazebo, der er det laget en verden som skal representere en gård. Det er hentet ut et satellittbilde av Njøs frukt og bær senter i Leikanger, som er et forskningssenter for frukt og bær [51], og lagt til noen enkle hinder som skal representere ett område med noen trær i. Det er blitt valgt å legge til et bilde som underlag slik at brukeren får bedre oversikt over distansene på kartet. Utviklingen av simuleringen har tatt inspirasjon fra forrige bacheloroppgave på Husky roboten, som også lagde en simulering i Gazebo [1].

Simulasjonen har alle sensorer som den fysiske Husky roboten har, det blir også simulert GPS-data for test av navigasjon. Kode for simuleringen ligger i prosjektets GitHub [49].



Figur 32: Simulasjon av Husky i Gazebo

6 Testing

6.1 Brukertesting

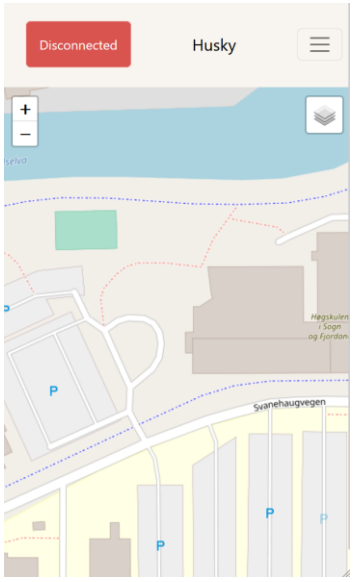
For å se om brukergrensesnittet er brukervennlig og intuitivt utføres brukertesting. Det ble bestemt å utføre brukertesten med *Rigorus*[1]og *Empirical*[2] lab-metoden [52, Kap. 14]. Dette for å ha et kontrollert miljø for utførelse av testen, og sammenligning av data kan gjøres uten for mye støy skal påvirke resultatene [52, Kap. 12].

Under testen ble det samlet objektiv data ved å observere bruker, og fra bruker direkte. Bruker ble oppfordret til å tenke høyt da det er anbefalt under testing av brukergrensesnitt [52, Del 12.5.2]. Det ble også samlet subjektiv data fra brukere ved slutten av testen i from av ideer om forbedringer eller andre meninger de hadde om applikasjonen.

Det ble laget et scenario som var likt for alle som var med på brukertestingen. De tildelte oppgavene var blant annet å markere to områder på kartet. Først på mobil enhet deretter på PC. Dette for å få datapunkter for å sammenligne nøyaktighet på ulike enheter. For scenario og rapport, se Appendiks D .

6.1.1 Observasjoner og tilbakemelding

Det viser seg at knapper for å sende data til robot, og for å slette markører buker har plassert, kan være problematisk for noen å finne på mobiltelefoner. Meny for disse er lukket når applikasjonen åpnes på telefoner, se Figur 33.

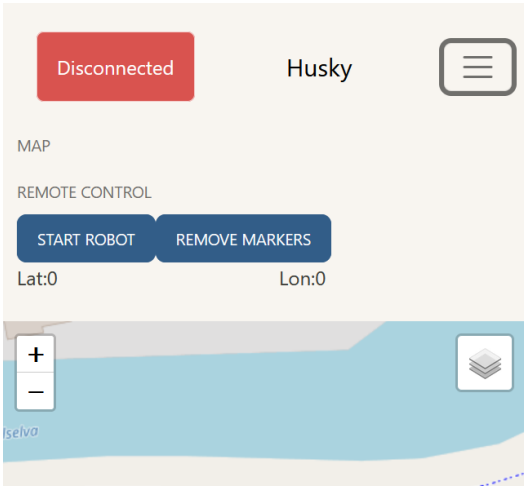


Figur 33: Hvordan applikasjonen ser ut på mobiltelefon

Flere deltagere prøvde å flytte markører på skjermen ved å trykke på markørene eller holde på dem for å prøve å få opp flere valg. Det kom fram ved observasjon og som tilbakemelding når deltakere tenkte høyt.

[1] Rigid, nøyaktig og effektiv metode for å minimalisere støy på data og maksimere effektivitet av testing.
[2] Data fra test basert på faktiske observasjoner, data innsamling og brukertesting.

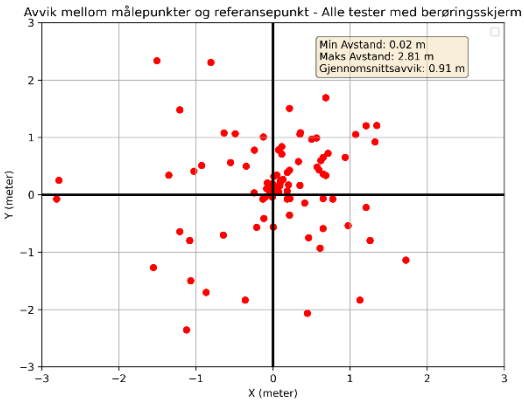
Når kandidater skulle finne lenken til fjernkontrollen med bruk av mobiltelefon og iPad var det problematisk for flere. Noen kom med tilbakemelding på at det ser mer ut som tekst enn noe som kunne trykkes på, se Figur 34. Det gikk også 42,2 sekunder i snitt før deltagerne trykket på lenken. Data vist på side 66, Tabell 1.



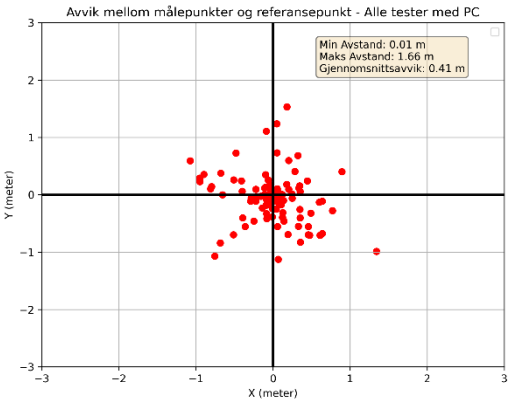
Figur 34: Viser meny åpen i applikasjonen på mobiltelefon

6.1.2 Avvik i markert område

Det ble samlet inn GPS-punkter som deltagerne plottet i brukergrensesnitt. I Figur og Figur viser avviket til referansepunkt som ligger i hjørnene på arbeidsområdene som skulle defineres. Plottene viser at tester utfør på berøringsskjerm har i snitt større avvik enn tester gjort på PC.

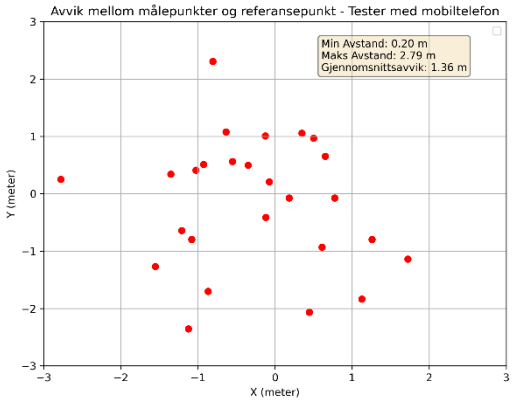


Figur 35: Avvik plottet fra tester med berøringsskjerm

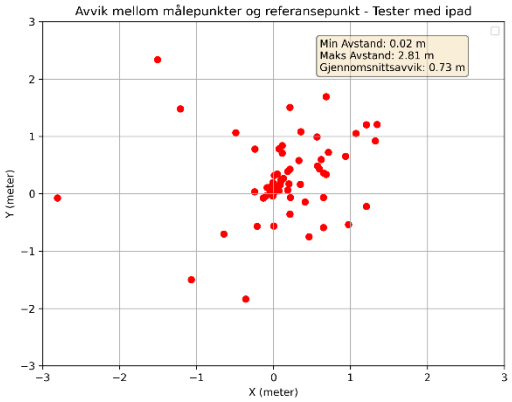


Figur 36: Avvik plottet fra tester med PC

Denne dataen har også blitt plottet for bare tester gjort med mobiltelefon, se Figur , og iPad, se Figur . En ting å nevne med datasett fra mobiltelefon er at det bare er plottet 26 punkter fra 2 kandidater da resten av innsamlet data fra mobiltelefoner ble annullert, se D.8 Resultat av målinger, Appendiks D for mer informasjon. Sammenligning av disse plottene og plott fra test på PC tilsier at nøyaktighet øker med større skjerm.



Figur 37: Avvik plottet fra tester med mobiltelefon



Figur 38: Avvik plottet fra tester med iPad

Full rapport ligger i Appendiks D .

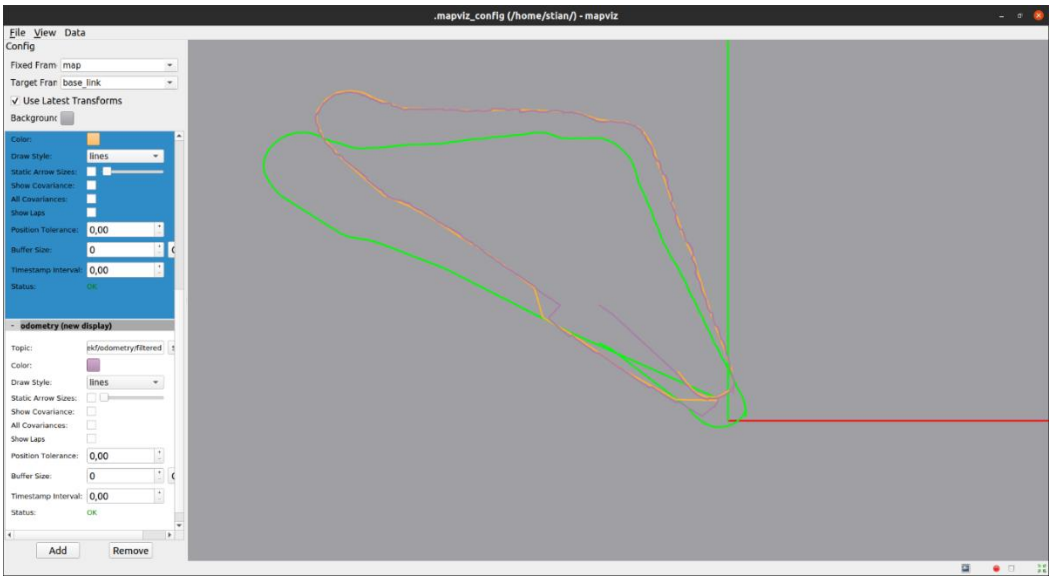
6.2 Felttesting

Som en del av testing av løsningen ble det utført felttester, den ene var ved Campus Førde og den andre var ved Blaaflat gård i Lærdal.

6.2.1 Campus Førde

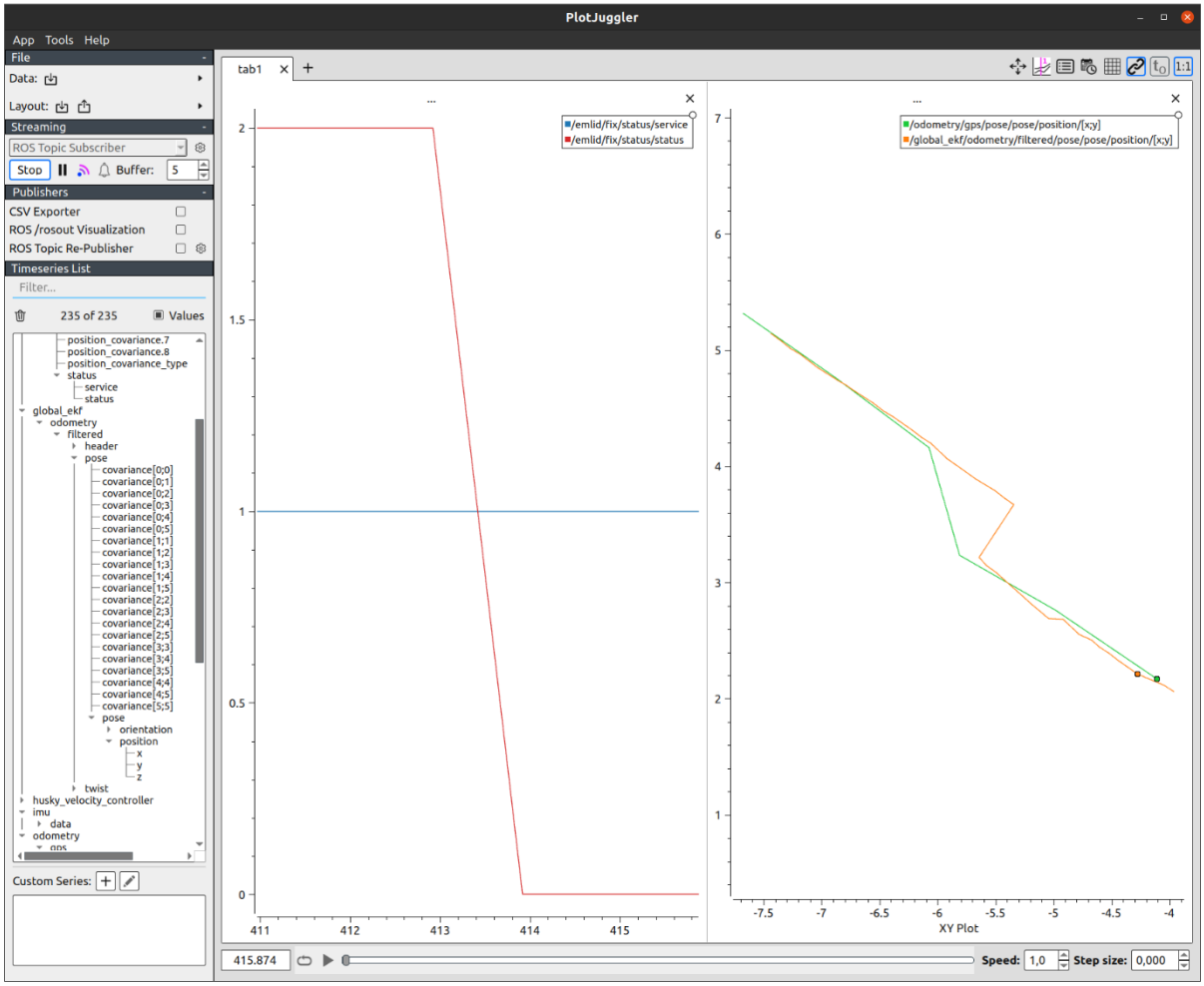
Testen ved Campus Førde ble gjort for å bli bedre kjent med Husky og hvordan systemet fungerer i praksis. Veileder hadde informert om en ustabil kobling med GPS-mottakeren, og ønsket å få mer data på hvordan dette gjenspeilet seg ved fysiske tester. Det ble utført flere manøvrer som ble loggført ved hjelp av *ROSbag,* som muliggjorde analysering av data ved senere anledning.

Figur 39 viser odometri drift over tid. Den mobile basen hadde allerede kjørt litt før denne testen ble gjort, og driften i odometri hadde allerede begynt. Testen ble også utført på gress, med forhold som ikke er optimale med tanke på friksjonsunderlaget.



Figur 39: Posisjons data fra Husky. Grønn: intern odometri, oransje: GPS-posisjon, lilla: Kalmanfiltrert estimat. Visualisert i MapViz

Figur 40 viser hva som skjer når GPS-mottaker mister RTK korreksjon. Signalet fra GPS-mottaker inneholder et tall, som representerer status på korreksjon, ved 2 har den RTK korreksjon og ved 0 har den ingen korreksjon. Grafen til høyre representerer den estimerte banen til roboten, der den oransje streken er det Kalmanfiltrerte estimatet mens det grønne er GPS-posisjonen. Ved tap av RTK-korreksjon blir det en distinkt forandring i estimert posisjon.

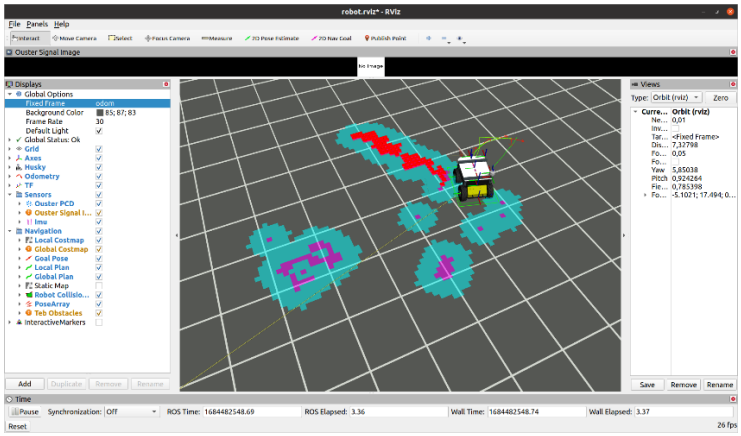


Figur 40: Tap av RTK-korreksjon. Visualisert i PlotJuggler

Mer utfyllende rapport fra testingen se Appendiks E .

6.2.2 Blaaflat gård i Lærdal

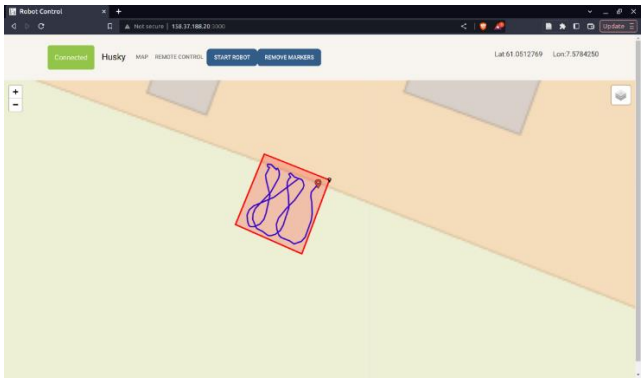
Testene i Lærdal rettet seg mer mot en helhetlig testing av systemet som hadde blitt utviklet. Der ble det avdekket problematisk oppførsel av LIDAR sensor. Den oppdaget gress rundt Husky som objekter og prøvde å unngå det. LIDAR viste seg også å kun fungere ved oppstart av tilhørende driver, noe som gjorde det mulig å utføre flere tester selv om det var gress rundt Husky roboten.



Figur 41: Gress som hindring. Visualisert i RViz

GPS-konverteringen som ble utviklet viste seg å være mangelfull, da navigeringen til disse punktene ikke var overensstemmende med de punktene som ble gjengitt. Det ble avduket at *My Bot Shop* sin supplerte GPS-navigasjon brukte UTM-koordinater, så det ble utviklet en mulighet for å navigere etter dette koordinatsystemet, noe som førte til tilfredsstillende resultater.

Under testingen av baneplanleggeren kom det fram viktigheten av orientering på posisjonen som ble sendt til *ROS-action* «move_base» for navigering. Denne inneholdt en tilfeldig verdi da dette ikke var definert på forhånd, og det førte til en bane som ikke var tilfredsstillende, se Figur 42.



Figur 42: Bane fullført innen angitt område

For mer utfyllende rapport fra testingen se Appendiks F

7 Diskusjon

Når det komme til vår fremdriftsplan, valgte vi å gå for en *Agile* tilnærming [53]. Der vi ikke hadde forhåndsdefinerte tidsfrister på hva som skulle bli løst, men dedikerte arbeidsperioder til å jobbe med et problem før vi analyserte fremgangen. Det var naturlig for oppgaven å bruke denne tilnærmingen, da deloppgavene var uavhengige av hverandre.

I starten gikk en del tid med på å sette seg inn i *JavaScript* som var helt nytt for oss. Det å utvikle applikasjonen i et språk som var ukjent førte til en tregere start en ventet og planlagte en sprint tid på noen uker, men det endte med å ta noen måneder. Vi valgte også å bruke *React* noe som vi håpet skulle forenkle deler av utviklingen, men når vi ser tilbake på dette burde vi nok unngått *React* for at læringskurven ikke skulle bli så bratt.

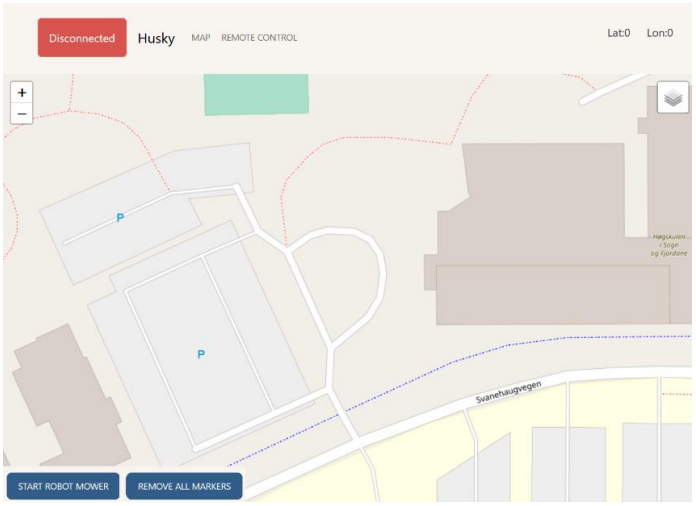
Under forprosjektet, og ved starten av selve prosjektet, brukte vi mye tid på å lese forskningsartikler før vi bestemte oss for hvilke løsninger vi skulle gå for når det gjelder baneplanleggeren. Artiklene vi leste om de forskjellige algoritmene kokte det ned til det som ble kalt for en «enkel frem og tilbake bevegelse», noe som førte til en tankegang at dette var fort gjort å implementere. Dette viste seg å være alt annet enn trivielt, og tidsbruken ble deretter. Algoritmen for å kunne utregne denne bevegelsen måtte lages fra bunnen av, uten eksterne kilder for hjelp som vi kunne finne. Det er godt mulig den måten denne bevegelsen ble utregnet på er unødvendig kompleks. Det gikk over en måned på dette problemet alene. For at baneplanleggeren skal fungere, setter det noen krav til polygonene og bunnlinja til banen. Se forbedringsforslag 8.1.2.

Vi fikk ikke lagt til alle funksjoner som vi ønsket, men vi leverer et produkt med grunnleggende funksjonalitet og potensialet til å bygge videre på. Det at vi har fått positive tilbakemeldinger fra HVL Robotics for arbeidet gjort under oppgaven, forsterker denne tankegangen. Vi har tro på at de verktøyene vi har utviklet vil være til hjelp når det kommer til videreutvikling av FutuRaPS og RoCutO sine systemer.

7.1 Brukergrensesnitt og brukertest

Det var flere ting under brukertesting, og testing i felt, som ser ut til å være fordelaktig å endre på. Brukertest viste at plassering av knapper for operasjoner i kart og av robot bør bli flyttet en annen plass enn i navigasjonsmenyen. For å tydeliggjøre sammenheng mellom knappene og kartet er det å ligge dem synlig over kartet en mulig løsning, se Figur 43.

Under felttesting savnet vi en knapp for å slette banen roboten har tatt tidligere. Banedata fra roboten og arbeidsområdet den skal jobbe i burde også vært lagret på server siden, ikke i nettleser. Om et arbeidsområde blir sendt til roboten og nettsiden lastes inn på nytt forsvinner arbeidsområdet fra kartet. Dette gjelder også robotens tidligere bane.



Figur 43: En mulig løsning på uklarhet rundt knapper i kart applikasjon

Det bør implementeres en startside med informasjon som introduserer bruker til virkemåten av applikasjonen. Når brukere forstod hva nettsiden forventet av dem hadde de generelt gode tilbakemeldinger. Her bør også det informeres om at robotens arbeidsområde må være innenfor gjerder og kanter roboten ikke kan navigere gjennom eller forbi. Når flere brukere skulle definere arbeidsområdet var de mer opptatt av at hele arbeidsområdet skulle være innenfor markert felt enn hvor hjørnepunktene ble plassert. Det vil føre til at roboten vil prøve å dekke områder den ikke kan bevege seg i. Det skal sies at bruker som hadde erfaring fra gårdsdrift var veldig oppmerksom på at området måtte defineres nøye av denne grunn.

Etter en markør er plassert bør hver enkelt markør kunne flyttes eller slettes. Det hadde fjernet irritasjonsmoment og spart tid når det blir trykket feil og alle punkter må fjernes. Flere brukere prøvde å trykke på tidligere plasserte markører for å få opp en meny.

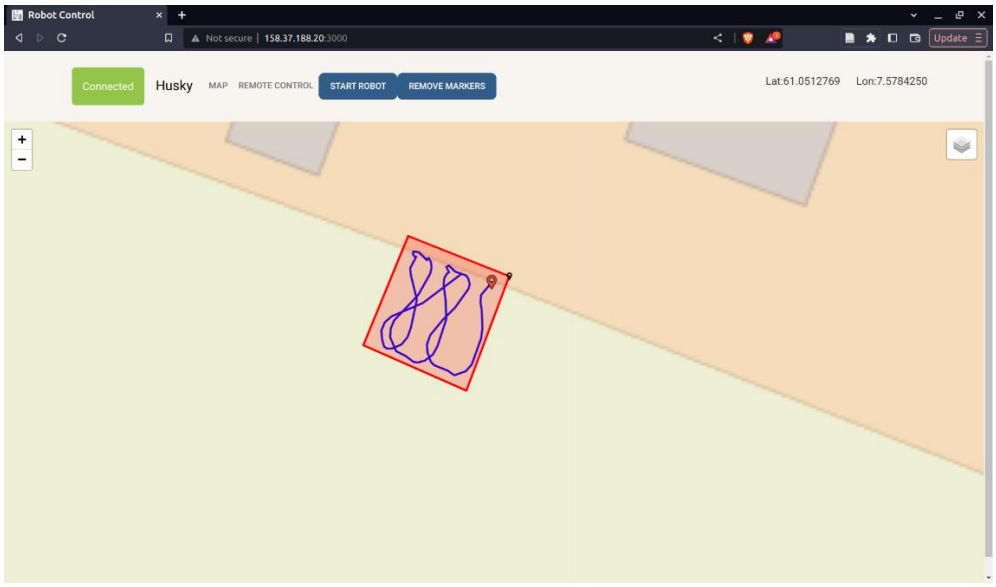
7.2 Baneplanlegging og kontrollering igjennom ROS

Etter felttesting kom det fram tydelig svakheter med implementasjonen i ROS og baneplanleggeren. Det kom også frem problemer med selve roboten og dens sensorer. En mulig løsning på LIDAR problemet, se 0, ville vært å øke terskelen som skal til for at et objekt er en hindring, ellers er det mulig å ignorere data fra de nederste strålene til LIDAR sensoren. Den siste løsningen byr på sine egne problemer, som ved for sen, eller for dårlig, detektering av hindringer, men det er noe som er verdt å tenke på ved videreutvikling.

Transformasjonen av GPS-punktene var feilaktig, da de ikke gav tilbake korrekte koordinater i Husky robotens koordinatsystem. Dette viste seg å ikke være et problem da det var ett alternativ lett tilgjengelig, ved å bruke UTM koordinater istedenfor. Dette fordi det kun er en matematisk kalkulering fra GPS-koordinater til UTM-koordinater, mer forklart i kapittel 3.2.3.1 – Konvertering til UTM koordinater. Den originale metoden for bruk av GPS-signaler innebar en egen ROS-node for transformasjon, som var avhengig av flere aspekt av systemet og var derfor mer sårbar for unøyaktigheter.

Roboten krysset over grensene til området som var definert. Det vil være fordelaktig å implementere en begrensning i arbeidsområde som roboten må holde seg innenfor.

Baneplanleggeren fungerte som forventet, men den må også finne orientering i sammenheng med posisjon for å få ønsket navigering. Slik systemet er i dag er det en tilfeldig orientering, noe som fører til baner uten rette linjer, men med innsvingninger til punktene.

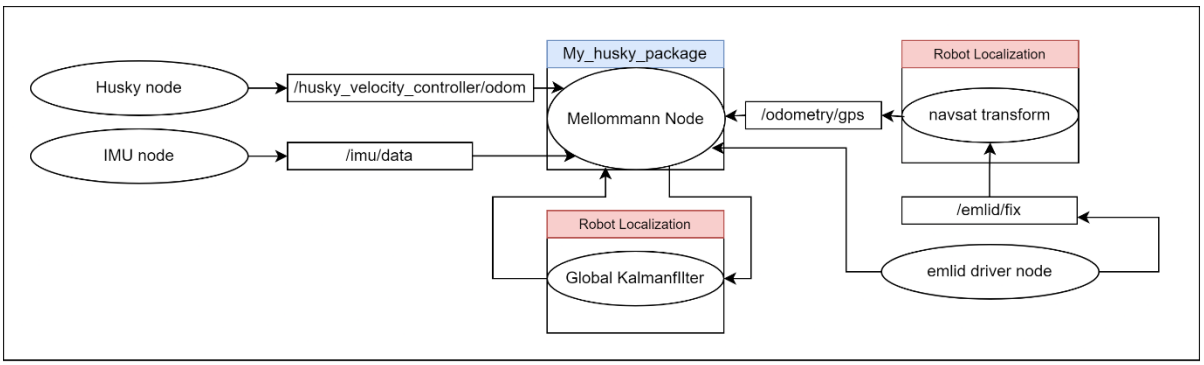


Figur 44: Bane fullført innen angitt område

7.3 Håndtering av tapt GPS-signal

Noe som kom fram fra testingen i felt var tap av presisjon når vi mistet RTK-korreksjonen i GPS-signalet. Med RTK-korreksjon er avviket på noen centimeter, men uten er avviket på 1.82 meter i snitt, se kapittel 3.2.1.1 og 3.2.2 for mer informasjon. Dette avviket ble svært tydelig i felt, og vi kunne lett se det i våre analyser av dataen i ettertid. Løsningen som ble bestemt var å lage en mellommann-node mellom sensor dataen og Kalmanfilteret. Tanken bak denne noden var å lagre den siste filtrerte posisjoneringen, og ved tap av RTK signal ta forskjellen i odometri og legge til posisjonen. Dette er en løsning som vil fungere godt over korte tidsperiode, men med tanke på drift i odometrien vil det ikke være en løsning for drift i lengre perioder.

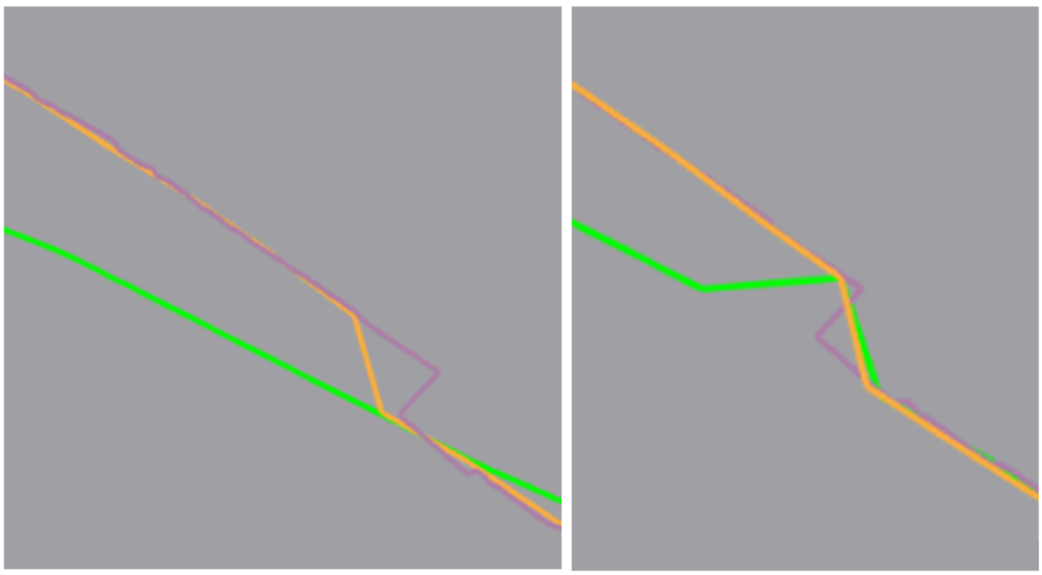
Figur 45 viser koblingskjemaet til denne noden i forhold til ROS systemet vi har. Den har en *ROS-subscriber* for alle sensor dataene som går til Kalmanfilteret, og om det er RTK korreksjon signal fra *emlid driver* noden blir all informasjonen videresendt til kalmanfilteret gjennom en *ROS-publisher*. Om det derimot ikke er RTK-korreksjon blir differansen i odometri kalkulert og lagt til det forrige filtrerte posisjonsestimatet.



Figur 45: Mellommann node for håndtering av tap av RTK signal

Vi har ikke fått tid til å teste det på den fysiske Husky roboten, men med å legge til denne logikken på en tidligere *ROSbag* fikk vi et resultat som var lovende.

Som vist på Figur 46 blir den grønne linjen, som representerer odometrien til Husky roboten, forflyttet til å følge den Kalmanfiltrerte linjen når Husky mister RTK-korreksjon. Det er ikke testet med et Kalmanfilter som tar hensyn til disse verdiene, derfor følger den fortsatt GPS-signalet uten RTK korreksjon og blir unøyaktig. Ved å implementere dette Kalmanfilteret er tanken at Huskyen kan navigere seg med kun odometrien som referansepunkt til den får tilbake RTK-korreksjon.



Figur 46: Uten korreksjon på venstre side, med korreksjon på høyre side

8 Konklusjon

Oppdragsgiveren hadde et ønske om å få utviklet ett system som kunne navigere seg autonomt i landbruksområder, med et brukergrensesnitt for definering av dette området. Brukergrensesnittet som har blitt utviklet inneholder funksjonaliteter som oppfyller disse ønskene. Brukeren kan avgrense ønsket arbeidsområdet gjennom et kart, og lagre informasjonen for videre bruk.

Ved å bruke de systemene som er på roboten med våre løsninger har det blitt utviklet en måte å navigere til punkt beskrevet med GPS-data. Denne navigasjonen tar ikke hensyn til orientering, kun posisjonering. Ved å implementere en funksjon for orientering, vil det være hensiktsmessig for utførelse av diverse arbeid i et landbruksområde, som for eksempel spraying, klipping og datainnsamling [4].

Det er blitt utviklet en algoritme for planlegging av en bane som dekker et helt område. Algoritmen trenger kun informasjon om hjørnene av området, og bredden av roboten. Dette er for å utføre gressklipping og andre oppgaver over et større, åpent område, som er en del av den tiltenkte bruken av systemet.

Alt av kode utviklet har blitt gjort med en tanke på videreutvikling hos HVL Robotics, og er dokumentert deretter i GitHub og kommentarer i kode. Dette er også noe oppdragsgiver har uttrykt ønske for under møter og veiledningstimer.

For liste over GitHub med kode, se vedlegg Appendiks C .

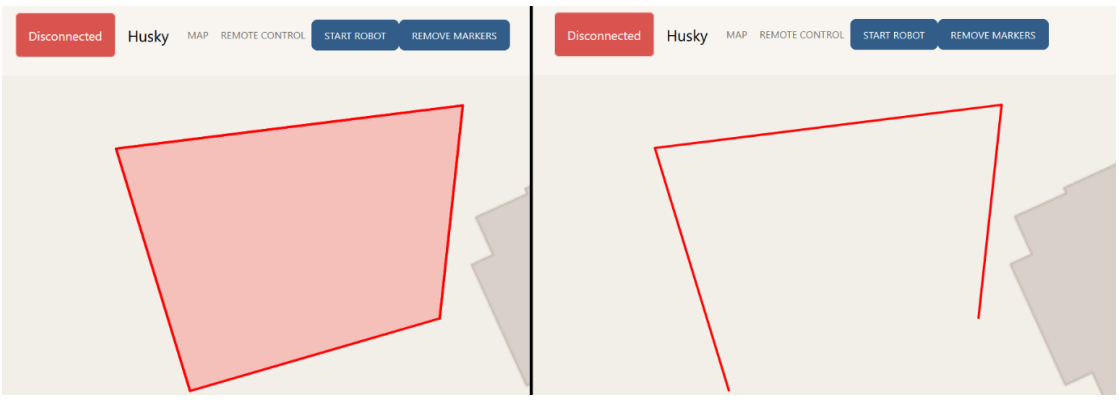
8.1 Kjente problemer og videreutvikling

8.1.1 Videreutvikling av brukergrensesnitt

Det er flere mulige veier videre med brukergrensesnittet. Noen er det gjort forarbeid for, siden ideen var å implementere dem som en del av dette prosjektet, men tiden strakk ikke til.

8.1.1.1 Flere navigeringsmuligheter i kart

På kartet blir arbeidsområdet merket med en polygon, men mellom punktene blir det også merket en enkel strek. Ideen med dette var å implementere muligheten for bruker til å definere et arbeidsområde, eller lage en bane roboten skal kjøre. Dette er implementert i ROS, men ikke i webapplikasjonen. Figur 47 viser hvordan dette kan se ut. Støtte for dette er lagt til i React-komponenten «MapComp.jsx».



Figur 47: Bruker definert arbeidsområde (t.v) og bruker definert bane (t.h)

Det hadde også vært fordelaktig å kunne lagre arbeidsområder som kan kjøres igjen på senere tidspunkt. Dette for å forhindre at bruker må legge til et nytt arbeidsområde hver gang roboten skal brukes.

8.1.1.2 Server på roboten

Et naturlig steg videre vil være å legge server inn på selve robotbasen for å slippe å kjøre den på en ekstern PC. Det vil også være fordelaktig om server starter opp sammen med roboten.

8.1.1.3 Fjernkontroll med kamera

For tiden er det ikke montert kamera på Husky roboten. Om det senere blir montert kamera er det mulig å vise video i fjernkontroll vinduet. Dette gjør at bruker ikke trenger å se roboten for å manøvrere den.

8.1.1.4 Server på internett

Server er bare tilgjengelig på det lokale nettverket til roboten. Derfor er det begrenset hvor bruker kan starte og stoppe roboten fra. En vei videre vil være å utvikle en løsning for å koble applikasjonen til internett. Det krever da mer sikkerhet, som for eksempel innlogging, slik at ikke alle på det åpne internett kan styre roboten.

8.1.1.5 Vise objekt rundt robot i kart

Roboten er utstyrt med en LIDAR som kan lage et kart over hindring rund seg i sanntid. Dette vil være mulig å hente ut av ROS og plasseres i kart ved hjelp av funksjoner i Leaflet.

8.1.2 Videreutvikling av baneplanleggeren

8.1.2.1 Utvidelse av den eksisterende algoritmen

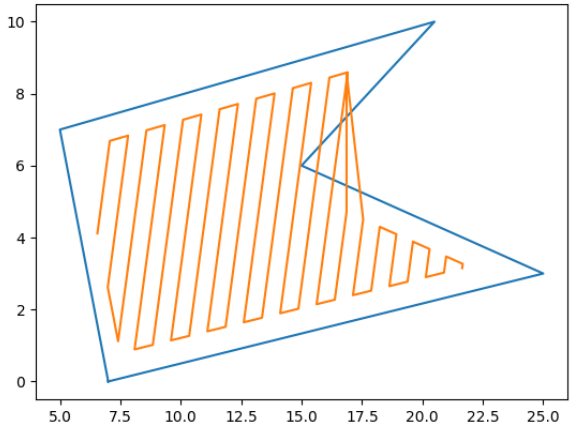
Algoritmen for å generere parallelle linjer som går fra en grunnlinje som strekker seg langs y-aksen er blitt utviklet, men den er ikke blitt modifisert for å virke på en grunnlinje som strekker seg langs x-aksen.

8.1.2.2 Automatisk identifisering av antipodale par

Shamos sin algoritme [40, s. 78–80] for å finne alle antipodale par er en av grunnpilarene bak *Rotating Caliper Path Planner* og den ble ikke implementert.

8.1.2.3 Konvekse og konkave polygoner.

Baneplanleggeren er ikke i stand til å håndtere konkave polygoner, så det å finne en løsning for å dele området inn i mindre konvekse polygoner vil være en del av en fremtidig løsning. Figur 48 viser problemet.



Figur 48: Baneplanlegging over et konkavt polygon

8.1.2.4 Orientering som del av banen.

Under testing viste det seg at den lokale baneplanleggeren som blir brukt krever også en orientering. Det vil derfor være nødvendig å utvikle en måte å bestemme orientering som kreves for å oppnå ønsket navigering.

Referanser

[1] I. V. Førde, C. L. Johnsen, og A. Torheim, «BO22EB-29 Selvkjørende mobil robot for datainnsamling fra frukt- og bærproduksjon på Vestlandet», Bachelor thesis, Fakultet for Ingeniør- og naturvitskap, Høgskulen på Vestlandet, Bergen, Norway, 2022. Åpnet: 17. mai 2023. [Online]. Tilgjengelig på: https://hvlopen.brage.unit.no/hvlopen-xmlui/handle/11250/3028463

[2] «Teknoløft Sogn og Fjordane». https://app.cristin.no/projects/show.jsf?id=573142 (åpnet 17. april 2023).

[3] «FutuRaPS: Future Raspberry Production System for Western Norway - Prosjektbanken», *Prosjektbanken - Forskningsrådet*. https://prosjektbanken.forskningsradet.no/project/FORISS/336603 (åpnet 15. mai 2023).

[4] Gruppe 10, «Sluttrapport Arbeidskrav 2». HVL.

[5] «Husky UGV - Outdoor Field Research Robot by Clearpath», *Clearpath Robotics*. https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/ (åpnet 18. mai 2023).

[6] N. Kjerstad, «treghetsnavigasjon», *Store norske leksikon*. 25. januar 2023. Åpnet: 10. mai 2023. [Online]. Tilgjengelig på: https://snl.no/treghetsnavigasjon

[7] N. Kjerstad, «dead reckoning», *Store norske leksikon*. 9. juli 2021. Åpnet: 10. mai 2023. [Online]. Tilgjengelig på: https://snl.no/dead_reckoning

[8] J. A. Holtet, «lidar», *Store norske leksikon*. 1. april 2022. Åpnet: 2. mai 2023. [Online]. Tilgjengelig på: https://snl.no/lidar

[9] «High-resolution OS1 lidar sensor: robotics, trucking, mapping», *Ouster*. https://ouster.com/products/scanning-lidar/os1-sensor (åpnet 2. mai 2023).

[10] «navigation - ROS Wiki». http://wiki.ros.org/navigation (åpnet 10. mai 2023).

[11] C. Rösmann, F. Hoffmann, og T. Bertram, «Timed-Elastic-Bands for time-optimal point-to-point nonlinear model predictive control», i *2015 European Control Conference (ECC)*, Linz, Østerrike, jul. 2015, s. 3352–3357. doi: 10.1109/ECC.2015.7331052.

[12] «teb_local_planner - ROS Wiki». http://wiki.ros.org/teb_local_planner (åpnet 10. mai 2023).

[13] P.Corke, «Localization: Localization with a Map», i *Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised, Extended And Updated Edition*, Springer, 2017, s. 160–164.

[14] N. Kjerstad og B. Forssell, «BeiDou», *Store norske leksikon*. 7. november 2021. Åpnet: 19. april 2023. [Online]. Tilgjengelig på: https://snl.no/BeiDou

[15] B. Forssell og N. Kjerstad, «GPS», *Store norske leksikon*. 26. februar 2023. Åpnet: 21. mars 2023. [Online]. Tilgjengelig på: https://snl.no/GPS

[16] «GLOBAL POSITIONING SYSTEM STANDARD POSITIONING SERVICE PERFORMANCE ANALYSIS REPORT», FAA William J. Hughes Technical Center, Atlantic City International Airport, NJ 08405, 112, jan. 2021. [Online]. Tilgjengelig på: https://www.nstb.tc.faa.gov/reports/2020_Q4_SPS_PAN_v2.0.pdf

[17] L. Mæhlum, «RTK», *Store norske leksikon*. 16. februar 2023. Åpnet: 21. mars 2023. [Online]. Tilgjengelig på: https://snl.no/RTK

[18] «Få veiledning om CPOS», *Kartverket.no*, 29. september 2022. https://kartverket.no/til-lands/posisjon/hva-er-cpos (åpnet 25. april 2023).

[19] M.D.Kenedy, «Chapter 1: Some Concepts That Underpin GIS», i *Introducing Geographic Information Systems with ArcGIS: A Workbook Approach to Learning GIS*, Newark, UNITED STATES: John Wiley & Sons, Incorporated, 2013, s. 3–88. Åpnet: 16. mai 2023. [Online]. Tilgjengelig på: http://ebookcentral.proquest.com/lib/hogskbergen-ebooks/detail.action?docID=7103552

[20] P. Mercator, *English: Figure showing definition of latitude (φ) and longitude (λ) on a sphere. The graticule at intervals of 10 degrees is taken from Sphere-wireframe.png at Commons.* 2010. Åpnet: 21. mai 2023. [Online]. Tilgjengelig på: https://commons.wikimedia.org/wiki/File:Latitude_and_longitude_graticule_on_a_sphere.svg

[21] J.P.Snyder, «Transverse Mercator Projection», i *Map Projections: A Working Manual*, First Thus edition.Books Express Publishing, 2012, s. 48–65.
[22] «ROS/Introduction - ROS Wiki». http://wiki.ros.org/ROS/Introduction (åpnet 2. mai 2023).
[23] «Features -- Gazebo». https://gazebosim.org/features (åpnet 10. mai 2023).
[24] «rosbag - ROS Wiki». http://wiki.ros.org/rosbag (åpnet 19. mai 2023).
[25] «tkinter — Python interface to Tcl/Tk», *Python documentation*. https://docs.python.org/3/library/tkinter.html (åpnet 29. april 2023).
[26] T. Schimansky, «TkinterMapView - simple Tkinter map component». 28. april 2023. Åpnet: 29. april 2023. [Online]. Tilgjengelig på: https://github.com/TomSchimansky/TkinterMapView
[27] «React». https://react.dev/ (åpnet 13. mai 2023).
[28] «React-Bootstrap». https://react-bootstrap.github.io/ (åpnet 13. mai 2023).
[29] «Bootswatch: Free themes for Bootstrap». https://bootswatch.com/ (åpnet 13. mai 2023).
[30] «Create React App». https://create-react-app.dev/ (åpnet 14. mai 2023).
[31] «Node.js», *Node.js*. https://nodejs.org/en (åpnet 14. mai 2023).
[32] «Google Maps Platform Documentation | Maps JavaScript API», *Google Developers*. https://developers.google.com/maps/documentation/javascript (åpnet 14. mai 2023).
[33] «Maps, geocoding, and navigation APIs & SDKs | Mapbox». https://www.mapbox.com/ (åpnet 15. mai 2023).
[34] «OpenStreetMap», *OpenStreetMap*. https://www.openstreetmap.org/about (åpnet 15. mai 2023).
[35] «Leaflet — an open-source JavaScript library for interactive maps». https://leafletjs.com/ (åpnet 15. mai 2023).
[36] «Leaflet/Leaflet». Leaflet, 15. mai 2023. Åpnet: 15. mai 2023. [Online]. Tilgjengelig på: https://github.com/Leaflet/Leaflet
[37] «React Leaflet | React Leaflet». https://react-leaflet.js.org/ (åpnet 15. mai 2023).
[38] «OpenLayers - Welcome». https://openlayers.org/ (åpnet 15. mai 2023).
[39] «rosbridge_suite». Robot Web Tools, 29. april 2023. Åpnet: 30. april 2023. [Online]. Tilgjengelig på: https://github.com/RobotWebTools/rosbridge_suite
[40] M. I. Shamos, «computational geometry», Ph.D Avhandling, Yale Univ., New Haven, CT, 1978. [Online]. Tilgjengelig på: http://euro.ecom.cmu.edu/people/faculty/mshamos/1978ShamosThesis.pdf
[41] J. I. Vasquez Gomez, M. M. Melchor, og J. C. Herrera Lozada, «Optimal Coverage Path Planning Based on the Rotating Calipers Algorithm», i *2017 International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, Cuernavaga, Mexico, nov. 2017, s. 140–144. doi: 10.1109/ICMEAE.2017.11.
[42] H. Choset og P. Pignon, «Coverage Path Planning: The Boustrophedon Cellular Decomposition», jan. 1998, doi: 10.1007/978-1-4471-1273-0_32.
[43] J. Sörme og T. Edwards, «A Comparison of Path Planning Algorithms for Robotic Vacuum Cleaners», Bachelor thesis, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, 2018. Åpnet: 11. mai 2023. [Online]. Tilgjengelig på: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-229771
[44] «robot_localization wiki — robot_localization 2.7.4 documentation». http://docs.ros.org/en/noetic/api/robot_localization/html/index.html (åpnet 11. mai 2023).
[45] «PlotJuggler», *PlotJuggler*. https://plotjuggler.io (åpnet 19. mai 2023).
[46] «Home», *Mapviz*. https://swri-robotics.github.io/mapviz/ (åpnet 19. mai 2023).
[47] W. Antonesen, «UI-Husky». 2. mars 2023. Åpnet: 16. mai 2023. [Online]. Tilgjengelig på: https://github.com/Axo-xD/ui-husky
[48] W. Antonesen, «Service for Husky». 8. mai 2023. Åpnet: 16. mai 2023. [Online]. Tilgjengelig på: https://github.com/Axo-xD/husky-gps-service
[49] S. S. Isene, «husky_gps_navigation». 19. mai 2023. Åpnet: 20. mai 2023. [Online]. Tilgjengelig på: https://github.com/Stian-Isene/husky_gps_navigation

[50] T. Bieniek, «utm: Bidirectional UTM-WGS84 converter for python». Åpnet: 16. mai 2023. [OS Independent]. Tilgjengelig på: https://github.com/Turbo87/utm
[51] «Om Njøs frukt- og bærsenter», *Njøs frukt- og bærsenter*. https://www.njos.no/om-oss/om-njos-frukt-og-baersenter (åpnet 21. mai 2023).
[52] H. R. Hartson og P. S. Pyla, *The UX Book: process and guidelines for ensuring a quality user experience*. Amsterdam ; Boston: Elsevier, 2012.
[53] «What is Agile Software Development?», *Agile Alliance |*, 29. juni 2015. https://www.agilealliance.org/agile101/ (åpnet 20. mai 2023).
[54] «ClickUp™ | One app to replace them all». https://clickup.com/ (åpnet 20. mai 2023).
[55] «The Visual Collaboration Platform for Every Team | Miro», *https://miro.com/*. https://miro.com/ (åpnet 20. mai 2023).

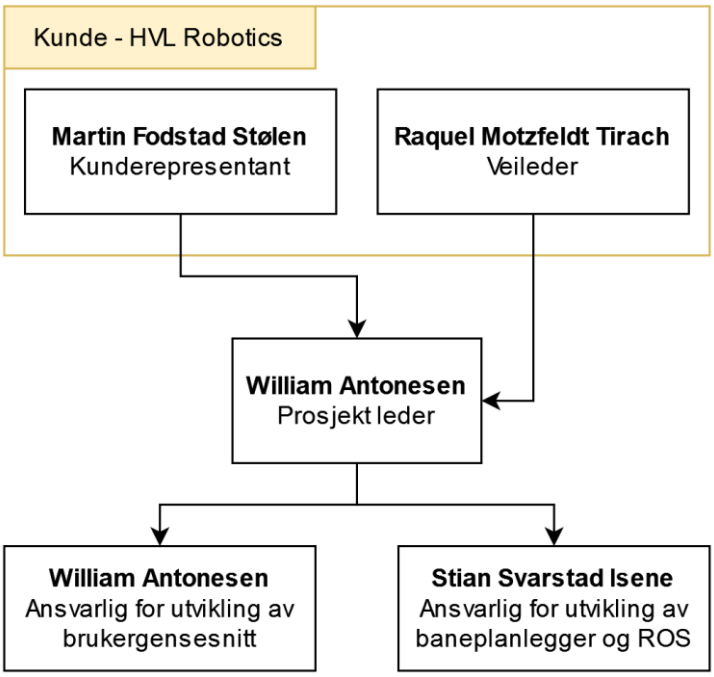
Appendiks A Forkortelser og ordforklaringer

API	Application Programming Interface
Big Data	Store datasett
CSS	Cascading Style Sheets
FutuRaPS	Future Raspberry Production For Western Norway
Gazebo	3D simulator for robotikk
GNSS	Global Navigation Satelite System
GPS	Global Positioning System
HTML	HyperText Markup Language
http	HyperText Transfer Protocol
Husky UGV	Husky Unmanned Ground Vehicle
HVL	Høgskulen på Vestlandet
IMU	Inertial Measurement Unit
IP	Internet Protocol
JavaScript	Programmeringspråk
LIDAR	Light Detection And Ranging
NPM	Node Package Manager
Python	Programmeringspråk
RoCutO	Robotic Grass Cutter for Orchards
ROS	Robotics Operating System
ROSbag	Logging av ROS-Topic
RTK	Real Time Kinematic
RViz	Robotic Vizualization
SLAM	Simultaneous Localization and Mapping
TCP	Transmission Control Protocol
TEB	Timed Elastic Band
Websocket	Websocket er en internettprotokoll som muliggjør toveis kommunikasjon mellom klient- og serverapplikasjoner.

Appendiks B Prosjektledelse og styring

B.1 Prosjektorganisasjon

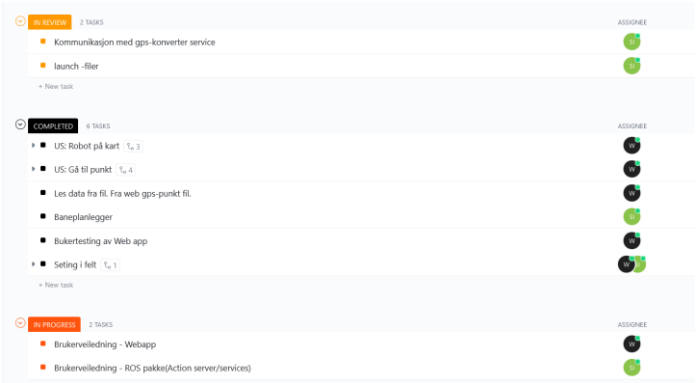
Dette prosjektet hadde en noe spesiell oppynging da veileder er en del av forskingsgruppen HVL Robotics som er kunde i prosjektet, se Figur 49. Det gjorde også at veileder hadde et unikt innblikk i hva som var ønskelig å få ut av prosjektet.



Figur 49: Prosjekt struktur

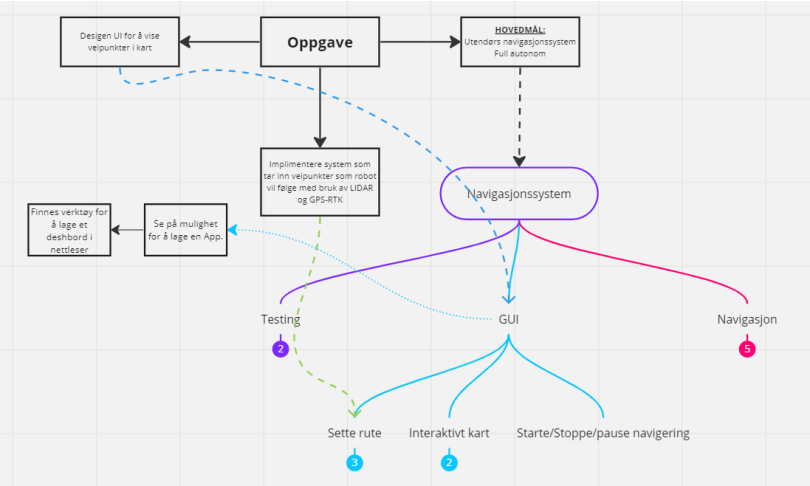
Prosjektleder

Prosjektleders ansvar var å kalle inn til møter internt og veiledningsmøter. Han hadde også ansvar for å passe på det var framdrift i prosjektet. Dette ble gjort ved å sette opp planer på plattformene *ClickUp* [54] og *Miro* [55]. *ClickUp* ble brukt for å sette opp mål og delegere oppgaver, se eksempel i Figur 50.



Figur 50: Sprints vist i ClickUp

Miro ble brukt for å visualisere problemer og løsninger. Og for å sette opp framdriftsplaner. Figur 51 viser eksempel etter møte hvor problemstilling i oppgaven ble visualisert og diskutert.



Figur 51: Eksempel på bruk av Miro for visualisering av oppgaven

Ansvarlig for deloppgaver

Oppgaven ble naturlig delt i to hoved løsninger: brukergrensesnitt og baneplanlegger og ROS. Det ble derfor satt en ansvarlig person for hver av disse deloppgavene.

Dette ble gjort for å separere ansvar mellom gruppemedlem og for at begge skulle få eierskap til en del av oppgave.

Veiledningsmøter

Det ble holdt regelmessige veiledningsmøter med intern veileder, Raquel Motzfeldt Tirach og kunderepresentant Martin Fodstad Stølen. Disse møtene ble det ført møtereferat for og de ligger som vedlegg i filen «Motereferat.zip»

B.2 Prosjektform

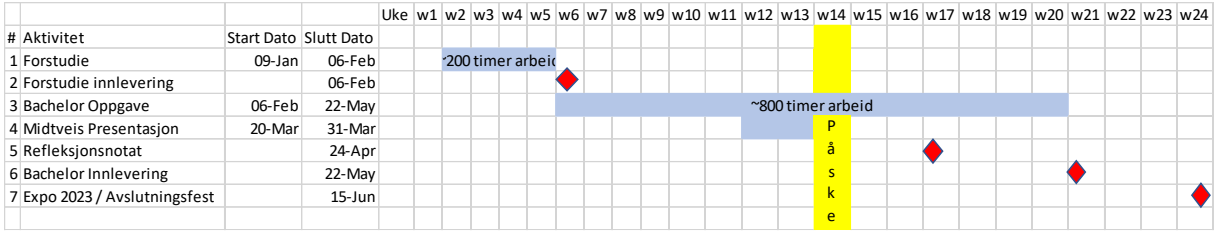
Da dette er en oppgave der utviklingen skjer parallelt og ikke sekvensielt, med stor usikkerhet i tidsbruken til de forskjellige løsningene, ble det bestemt å bruke en *Agile* prosjekthåndtering. Denne håndteringen baserer seg på å jobbe i mindre arbeidsperioder med forhåndsbestemte problematikk som skal løses, der arbeidet blir reflektert over i ettertid. Disse periodene blir kalt en *Sprint*, og det muliggjør fleksibilitet når det kommer til uforutsette hindringer. Denne tilnærmingen innebærer også å levere funksjonelle deler av produktet, selv under utvikling. Dette gjør det mulig å få tilbakemelding direkte fra kunden og brukere tidlig i prosessen [53].

En slik prosjekthåndtering er spesielt egnet for prosjekt der kravene og tidsbruken er usikre, med et ønske om å kunne holde seg fleksibelt. Et typisk eksempel på dette er programvareutvikling og produktutvikling.

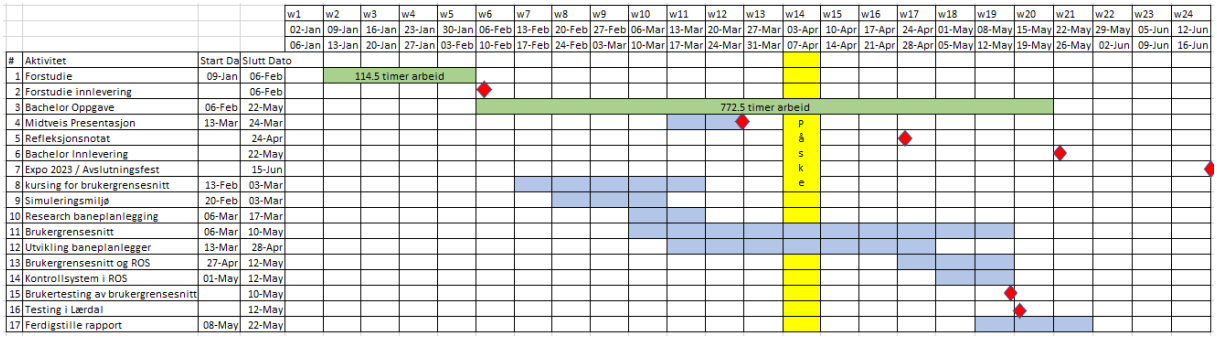
Under forprosjektet ble problemstillingene vi kunne møte kartlagt, og kravene fra oppdragsgiver ble kategorisert inn i «mål» og «ønsker». Dette ble gjort for å få en bedre oversikt over hva som inngikk i en potensiell løsning, og for å få et grovt estimat på tidsbruken dette kom til å ta. Det ble også planlagt å ikke fylle ut et Gantt diagram på forhånd, men heller fylle ut etter hvert som vær sprint ble fullført.

B.3 Fremdriftsplan

Merk, Gantt-diagrammet ved ferdigstilt prosjekt inneholder ikke forstudieperioden da dette ikke forandret seg fra det originale diagrammet og det gjør det betydelig mer leselig. En kopi ligger som vedlegg i zip filen «GanttDiagram.zip».



Figur 52: Det originale Gantt diagrammet fra forprosjektet



Figur 53: Gantt diagram ved ferdigstilt prosjekt

Timeliste følger med som vedlegg i filen «Timelister.zip». Der ligger det ett excel-ark som er blitt brukt for loggføring av timer.

B.4 Utgifter

Utgiftene som er knyttet til prosjektet. Satsene er hentet fra HVL sine retningslinjer for studenter i verv. Det var kun reisen til Lærdal som hadde personlige utgifter for gruppen.

Reise til Lærdal:			
	Pr stk	Antall	Sum NOK
Kilometergodtgjørelse	3.5	290	1015
Passasjertillegg	1	290	290
Hotellrom i Lærdal	1495	2	2990
Døgndiet ved overnatting	400	2	800
Trekk for frokost	-20%		640
Ferje	123	2	246
Autopass rabatt	-10%		221.4
Total			5156.4

Appendiks C Brukerdokumentasjon

Her ligger linker til alle GitHub repo som er blitt brukt i dette prosjektet, og brukerdokumentasjonen er der.

Samlingspakken husky_gps_navigation: https://github.com/Stian-Isene/husky_gps_navigation

Under denne pakken finnes:

- My_husky_messages: https://github.com/Stian-Isene/husky_gps_navigation/tree/main/my_husky_messages
- My_husky_package: https://github.com/Stian-Isene/husky_gps_navigation/tree/main/my_husky_package
- Husky-gps-service: https://github.com/Stian-Isene/husky_gps_navigation/tree/main/husky-gps-service
- Sweeping_coverage_path_planner: https://github.com/Stian-Isene/husky_gps_navigation/tree/main/sweeping_coverage_path_planner

I tillegg finnes pakken for brukergrensesnittet: https://github.com/Axo-xD/ui-husky

Appendiks D Brukertesting av applikasjon

Dato utført: 10.05.2023

D.1 Intern testing

Før start av brukertesten tok vi et skritt tilbake og prøvede å se på applikasjonen fra en brukers perspektiv. Dette for å forhåpentligvis finne de mest åpenbare problemene for en bruker før vi tar inn personer for å teste applikasjonen. Dette førte til at vi fant og korrigerte følgende:

- På knappen som trykkes for å sende data fra applikasjonen til ROS stod det «Send Data». Dette kan være uklart for en bruker. Det som skal skje fra brukerens perspektiv når data sendes er at roboten skal starte arbeid. Vi endret tekst på knapp til «Start Robot Mower».

- Knappen for å fjerne markører bruker har plassert på kart hadde teksten «Reset Data». Dette var også noe som kunne være forvirrende for bruker. Teksten er nå «Remove all markers».

- Når startknappen for roboten blir trykket, og ROS ikke er tilkoblet kommer det opp en tilbakemelding om dette i konsoll. Dette er ikke noe våre brukere kommer til å benytte seg av. Derfor kommer det nå opp en popup som forteller bruker «Robot not connected. Please make sure the robot is powered on and connected to the network». Det ble også lagt til flere popup feilmelding for når robot er koblet til ROS, men ROS-service ikke kjører. Eller om ROS-service returnerer en feil.

- Knappene var på hver sin linje når de ble åpnet på en mobiltelefon. Dette førte til at de ikke lenger så ut som en knapp. De ble flyttet slik at de er plassert vedsiden av hverandre på mobiltelefoner.

- Navn på fanen til nettsiden var «React App» som ikke var veldig beskrivende for hva som faktisk var i fanen. Navn ble byttet til «Robot Control» og et mer passende ikon ble bruk.

D.2 Planlegging av test

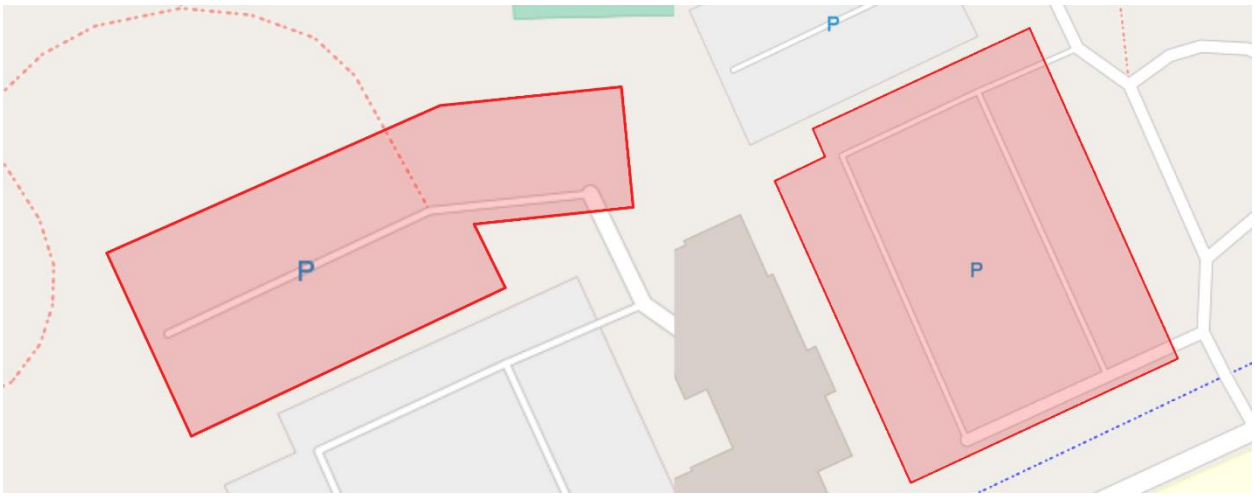
Etter å ha sett på mulige løsninger for brukertest kom vi fram til at vi ønsket god kvalitet på data og tilbakemeldinger. Vi fant en bok som gikk i dybden på temaet om brukertesting og hva som er viktig å fokusere på å få data med høy kvalitet .

Det ble definert mål med testen, se disse i D.3 og laget et test-senario med oppgaver som brukere fikk utlevert, se D.4 . Dette senarioet med oppgaver var utviklet for å returnere den data som vi var ut etter ut ifra målene.

D.3 Mål med testen

Det er i hovedsak to mål med testen. Det første er å finne ut hvordan brukeropplevelsen er og om det er noe i designet av nettsiden som er uklart eller kan forbedres. Det andre er å samle inn data fra samme punk fra alle brukere. Vi ønsker å bruke denne dataen til å se hvor stort avvik brukere har fra faktisk punkt i kart og om denne variasjonen endres når de bruker en mindre skjerm på en telefon eller en større skjerm på en PC.

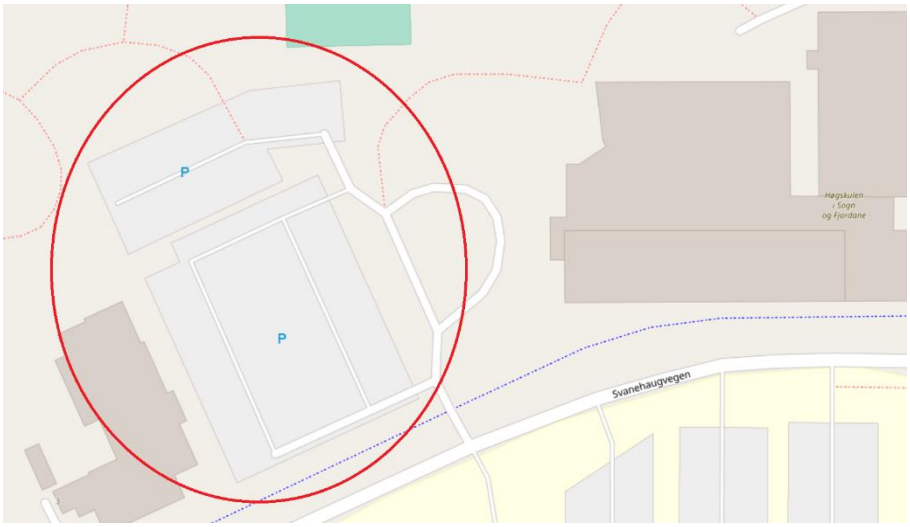
Området vi bruker for testene er parkeringsplasser vest for Campus Førde. For å markere området trengs minimum 7 punkt for den nordlige parkeringsplassen og 6 punkt for den sørlige. Se figur under. Disse punktene er blir brukt for å se på variasjon fra brukere.



Figur 54: Nordlig parkeringsplass (t.v) og sørlig parkeringsplass (t.h) markert i applikasjon

D.4 Test senario

Du en bonde og har mottatt din nye robot som skal hjelpe deg på gården. Du ønsker å teste den ved å klippe din innmark som ligger vest for HVL Campus Førde (Parkeringsplassene).



Brukerveiledning:

Oppgave 1 (Mobil)

1. Scann QR-koden.
2. Marker arbeidsområdet (Den minste parkeringsplassen) på kart.
3. Start roboten.

Oppgave 2

Roboten ble ferdig med første mark. Gjør det samme med den neste (Den store parkeringsplassen)

Oppgave 3

Roboten var for langt fra målet. Du får en melding om at du må kjøre den manuelt. Hvordan går du fram?

Oppgave 4 (PC)

1. Marker første arbeidsområdet (Den minste parkeringsplassen) igjen på kart.
2. Start roboten.
3. Marker andre arbeidsområdet (Den store parkeringsplassen) igjen på kart.
4. Start roboten.

D.5 Utførelse av test

Tester ble utført individuelt med kandidaten i samme rom. Om mulig utførte de testen med sin personlige mobiltelefon, men noen kandidater fikk problemer med å laste inn siden og fikk da låne en telefon. Halvparten brukte også en iPad istedenfor en telefon.

Testen be utført ved å gi kandidaten en rask introduksjon og levere ut senario vi ville spille ut. Vi ga beskjed om at vi ønsket at kandidaten skulle tenke høyt og at vi ikke kom til å hjelpe om de spurte om noe spesifikt. Vi ga ikke mer informasjon under testen, men ga hint om kandidaten sto fast over lenger tid. Dette ble også notert om hint ble gitt.

For å teste hvor naturlig det var å navigere med navigasjons linker øverst på siden spurte vi om kandidater kan prøve å finne en måte å kjøre roboten manuelt.

Når testene var utført diskuterte vi opplevelsen og spurte om det var konkrete tilbakemeldinger til utforming av nettsiden.

D.6 Test kandidater

Testing be utført på Campus Førde og kandidater ble som ble hentet inn hadde tilknytning til HVL. Dette var ikke en ideell måte å få varierte deltagere på så vi endte med at alle var menn i aldersgruppen 18-35. Alle kandidatene hadde også minst erfaring fra et programmeringsfag. Og to personer hadde arbeidserfaring fra programmering og robotikk. Det var bare en person som var med på testen som var bonde.

Fordelen med å utføre test på Campus Førde var at testen gikk raskt og vi dermed fikk gjort testen på en dag. Det hadde vært fordelaktig å utføre tester en annen plass hvor vi hadde fått kandidater med mer variert bakgrunn.

D.7 Tilbakemeldinger og observasjoner under testing

Det var en del av kandidatene som hadde problemer med å finne knappen for å fjerne alle markerte punkt i kartet. Dette førte til litt frustrasjon og noen måtte få et hint om at det var en meny de kunne se igjennom etter noen minutter uten progresjon.

Det var også flere som ønsket mulighet til å flytte eller slette enkelt punkter. Noen prøvde også å holde på markør for å få opp en meny eller få mulighet til å slette punkt når de brukte touch-skjerm. Noen zoomet også inn med dobbeltklikk. Dette førte til at de zoomet inn, men også plasserte en markør.

Samme problemet med å finne knapp for å fjerne markører hadde noen problemer med å finne knappen for å starte roboten. Flere prøvde å trykke på alt og dermed sendte flere kommandoer for å kjøre roboten til nye koordinater.

Når kandidatene skulle starte på neste område var den none som ikke slettet markører fra tidligere og forventet at de skulle forsvinne av seg selv. Når de ikke bel slettet av seg selv slette de oftest markører og startet på nytt.

Det var mange som fant det lite intuitivt å trykke på lenken til «Remote Control». Det var flere som trodde det bare var tekst og ikke lenker. Her var det noen som trengte hint for å komme i mål med oppgaven.

Når oppgaven gikk over på PC hadde kandidaten kjennskap til applikasjonen. Knapper og lenker var også synlig uten å måtte åpne menyen. Flere kommenterte at de likte utformingen bedre på PC. Og samtlige kom seg greit gjennom oppgaven på nytt på PC.

Tilbakemeldingene som ofte gikk igjen etter oppgaven:

- Knappene skulle vært plassert på selve kartet og være synlig uten å måtte gå inn i en undermeny på mobile enheter. Eller plassere seg ulike plasser basert på skjermstørrelse.
- Det ikke var intuitivt hvordan man skulle starte å markere et område.
- «Remote Control» var ikke tydelig at var en link slik som utformingen er i dag.
- Ønsker om å ha en landingside med informasjon hvor man kan velge å åpne kartet eller fjernkontroll.
- Mulighet til å flytte markører etter de har blitt plassert for å gjøre justeringer.

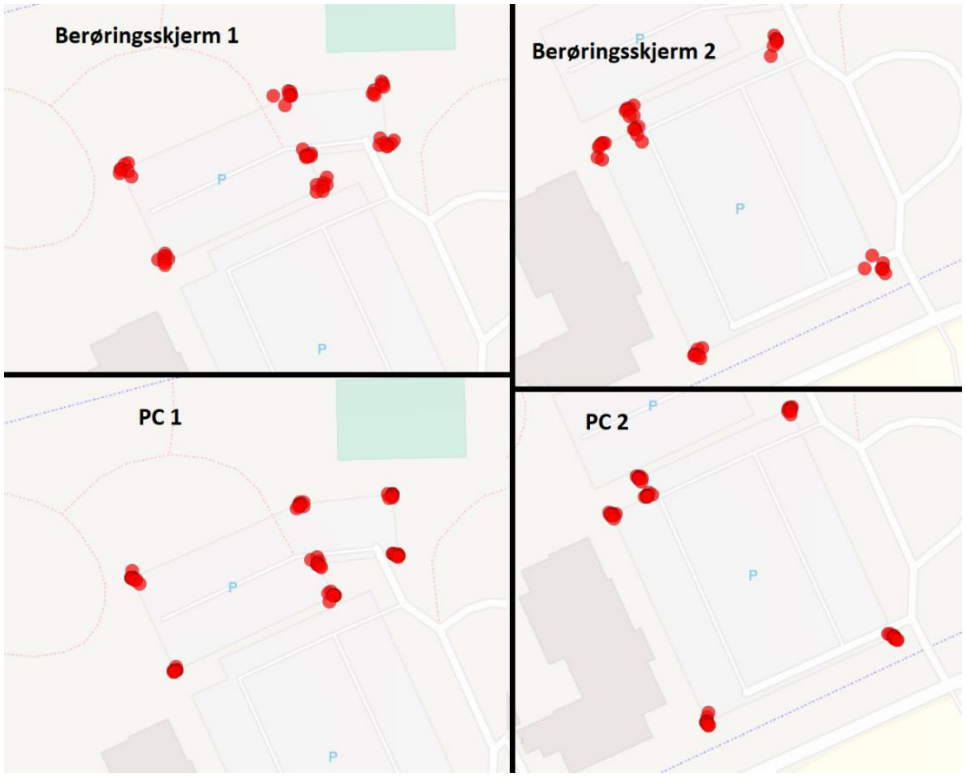
Mange var positiv til løsning når de hadde forstått prinsippet. Og da spesielt når det kom til utformingen av siden når den var på en PC. Det var da mye mer oversiktlig en på en mindre skjerm.

D.8 Resultat av målinger

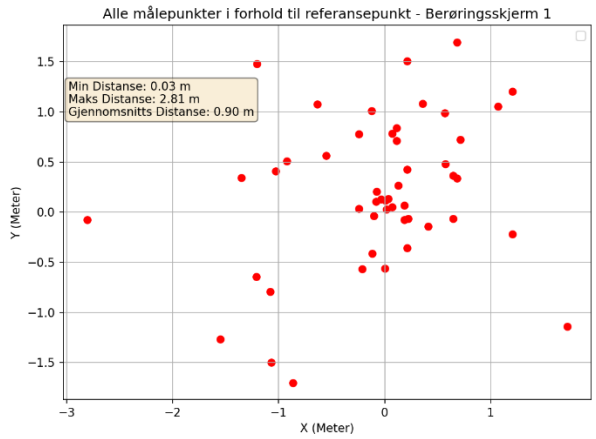
Det var totalt 10 personer som utførte testen. Alle brukte først mobiltelefon eller iPad og deretter PC til å markere to forskjellige arbeidsområder i kart. Dette førte til 40 sett med data punkter å sammenligne mot referanseverdi.

Det var 3 sett med data som ble annullert mobiltelefon. Dette fordi brukere ikke markerte alle punkter og hellet et større område rundt arbeidsområdet. Det var også en test som ble annullert fra PC av samme grunn. Det var ingen annullert data fra ipad tester.

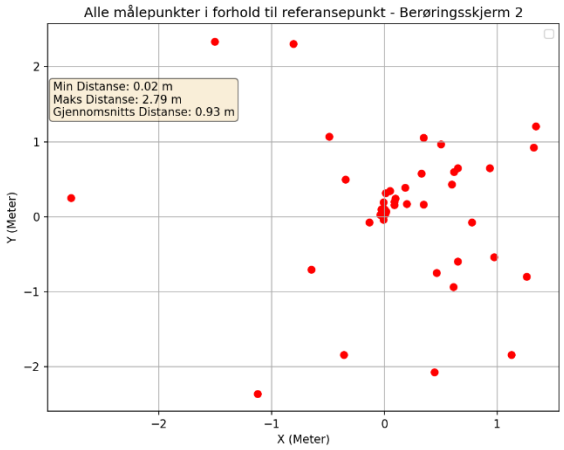
Se målinger i figur under.



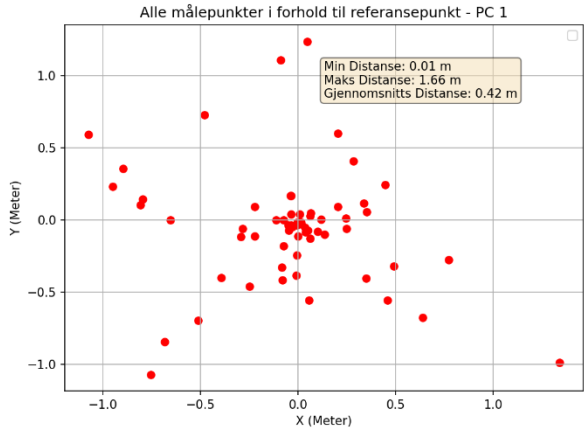
Figur 55: Viser alle datapunkter tatt med i dataanalyse delt inn i fire kategorier



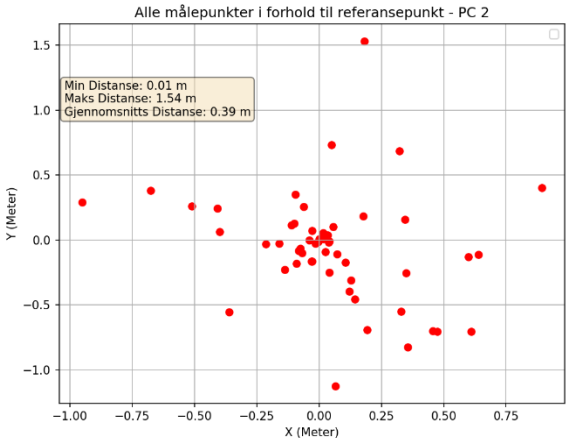
Figur 56: Avvik fra referansepunkt av test berøringsskjerm 1



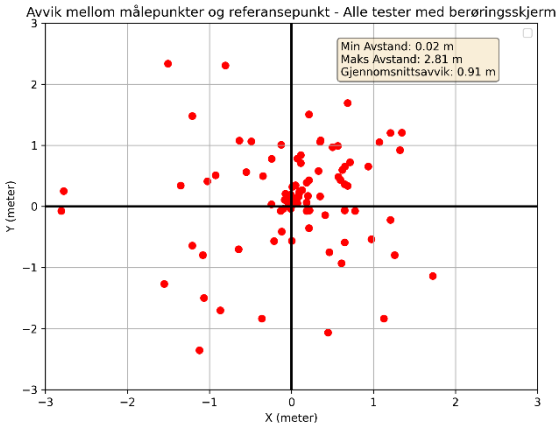
Figur 57: Avvik fra referansepunkt av test berøringsskjerm 2



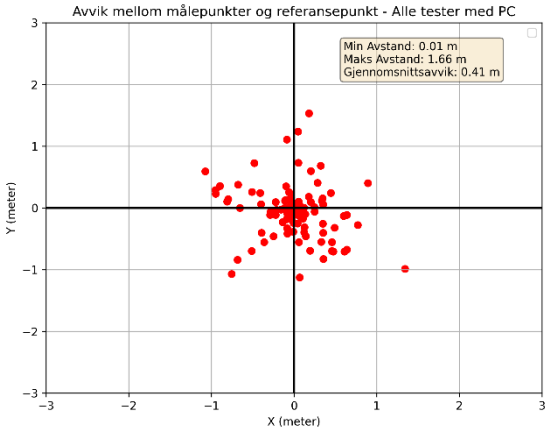
Figur 58: Avvik fra referansepunkt av test PC 1



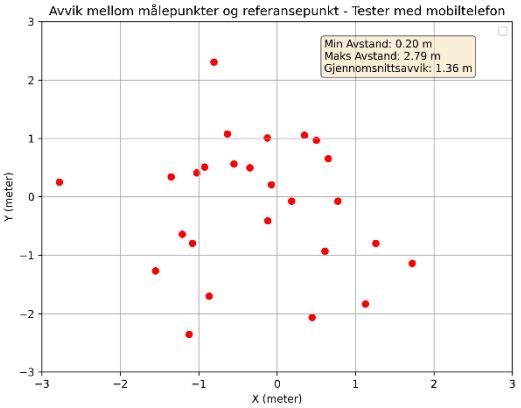
Figur 59: Avvik fra referansepunkt av test PC 2



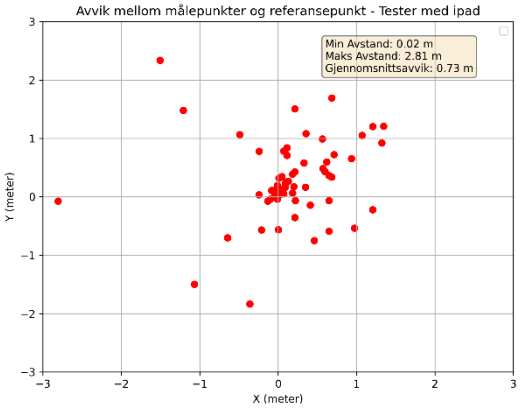
Figur 60: Avvik fra referansepunkt av samlet data fra berøringsskjerm



Figur 61: Avvik fra referansepunkt av samlet data fra PC



Figur 62: Avviksdata fra referansepunkt, tester med mobiltelefon



Figur 63: Avviksdata fra referansepunkt fra tester med ipad

Tabell 1: Gjennomsnitts tid på å funnføre oppgaver

Oppgave	Gjennomsnittstid [s]
1: Marker første området (berøringsskjerm)	162,5
2: Marker andre området (berøringsskjerm)	44,7
3: Finn mulighet til å kjøre manuelt (Trykke lenke)	42,2
4: Marker begge området (PC)	49,3

Se vedlagte zip-fil (Brukertest og analyse) for rå data, notater fra hver enkelt test og kode brukt i analyse. Script som heter «sammenlign_punkter_og_skriv_cvs» er det som har blitt brukt for plotting og kalkulering av avvik.

Appendiks E Rapport fra testing ved Campus Førde

Sted: Campus Førde

Dato: 13 og 20. April

Til stede: Stian Svarstad Isene og William Antonesen.

Ved flere anledninger ville vi teste ut hvordan GPS-RTK sensoren fungerte ute i feltet og hvordan vår webapplikasjon fungerte i praksis. Vi visste at GPS-RTK sin tilkobling ikke alltid fungerte, og vi ønsket å se hvordan det påvirket systemet. Vi skulle også ta noen rosbags, for så i ettertid analysere dataen vi hentet inn.

E.1 13. April

Vi begynte med å se på hvilke pakker som skulle kjøre for å få til den innebygde navigeringen til Husky, det viste seg å være flere *launch*-filer som måtte kjøres før vi kunne starte navigeringen. Deretter tok vi med oss Husky ut til rundkjøringen ved inngangen til garasjen ved Campus Førde.

Vi kjørte litt rundt, testet den innebygde navigasjonsalgoritmen. Webapplikasjonen fungerte svært bra, med både manuell kontroll av husky og visning på kart. Vi tok også noen *ROSbags* som vi skulle se mer på.

GPS-RTK signalet var svært upålitelig, den mistet kontakten flere ganger under testing. Det viste seg også å gi svært dårlig resultat i navigering, med en robot som tydelig bommet på punktet den skulle til. Vi testet nøyaktighet ved å markere området som den mobile basen skulle gå til, noe som den bommet grovt på. Vi målte ikke hvor mye den bommet.

E.2 20. April:

For å få mer data fra husky bestemte vi oss for å kjøre en innsamling til av data, der vi mer systematisk katalogiserte data tilknyttet en aksjon som den mobile basen skulle gjøre. Det ble laget en plan av hva som skulle bli gjort. Vi ville ha data som forklarte oss hvorfor navigasjonen ikke fungerte som ønsket.

- Stå i ro uten å ha på GPS
- Stå i ro med GPS
- Kjøre i rett linje uten GPS
- Kjøre i rett linje med GPS
- Kjøre «follow_waypoints» programmet

Der det også var skrevet på forhånd korresponderende kommandoer i ROS for å ta vare på all dataen i *ROSbags* med tilhørende navn.

På dagen bestemte vi oss for å teste ved siden av volleyballbanen ved Campus Førde der det er åpent, for å ha muligheten til gode GPS-RTK-signaler.

Når Husky kjørte i rett linje, ble det kjørt frem og tilbake, deretter ble det kjørt i en sirkel når vi kom tilbake til startpunktet. Det var ikke utført noen målinger for å forsikre oss om at vi kom til nøyaktig samme plass, det var heller ikke formålet med testen. Vi ville ha odometri data for å kunne se

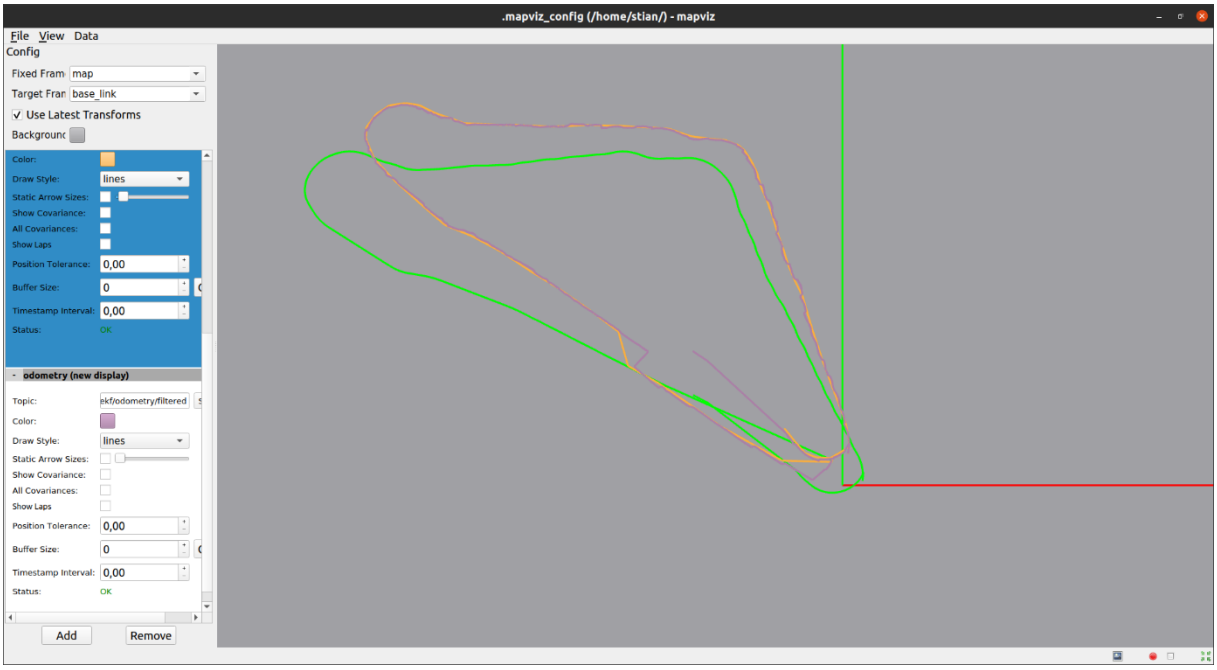
hvordan systemet fungerte sammen og for å se om det var noe som kunne forklare tidligere unøyaktighet som dårlig definerte transformasjoner, men vi fant ingen tegn til det.

Det ble fulgt ett «follow_waypoint»-program, som består av to deler. I den første delen kjører Husky til punkter som blir lagret i programmet, og i den andre delen så kjører Husky av seg selv til de samme punktene. Det var svært lærerikt å se hvordan den lokaliserer seg basert på dataen den får. Under testing mistet husky RTK signal.

E.3 Resultat av test:

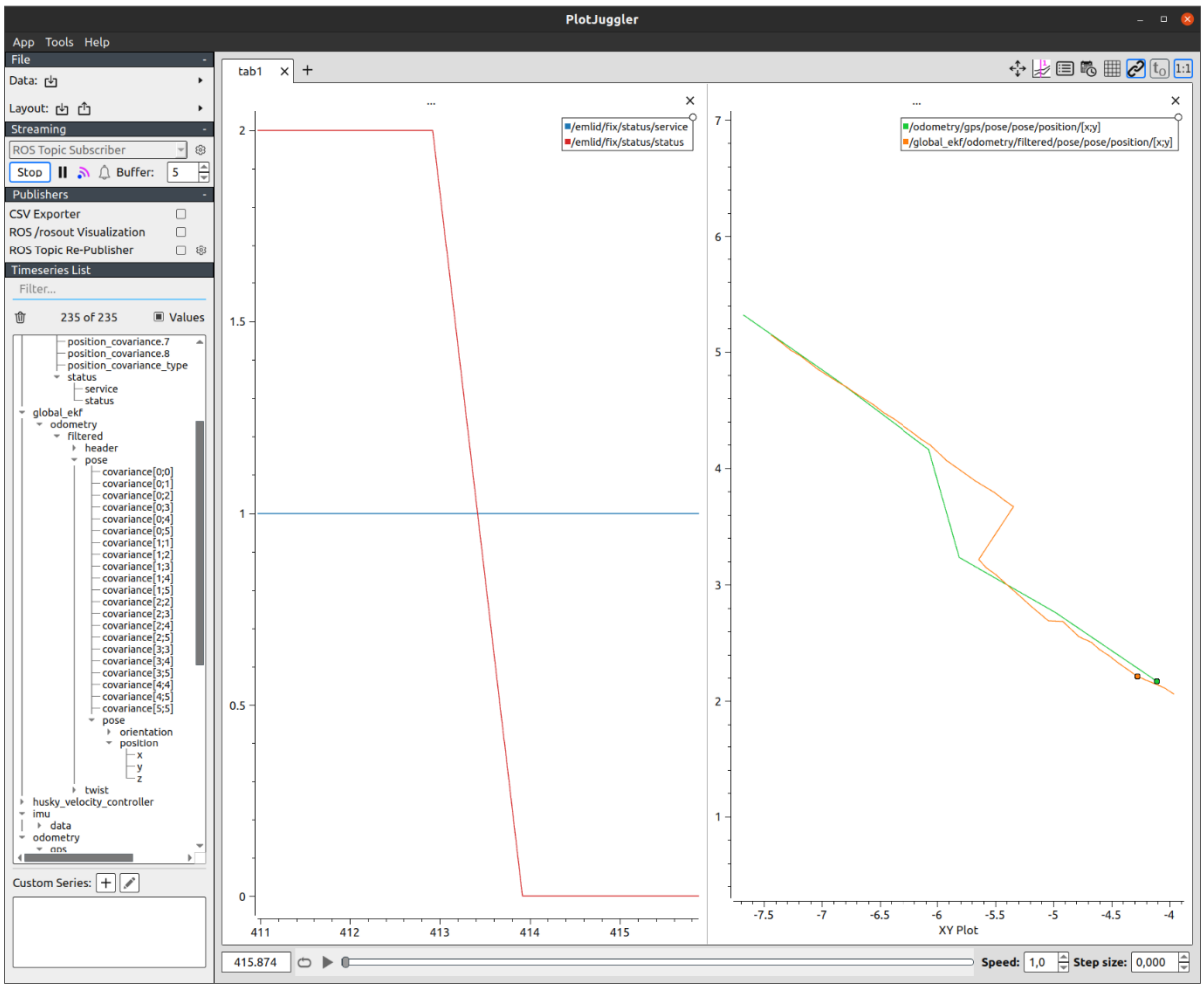
Ettersom vi tok flere *ROSbags*, var det mulig å kjøre dataen gjennom noen visualiseringsverktøy. Det vi endte opp med å bruke var PlotJuggler for visualisering av sensor data, spesielt statusen til RTK delen av GPS-signalet og MapViz for å visualisere odometri dataen vi fikk. Da odometri fra GPS og internt i Husky, samt resultatet fra kalman filteret.

Vi fikk se hvor mye odometri drift faktisk har å si for vårt system, og vi fikk sett hvor mye det har å utgjøre på estimert posisjon når vi ikke lenger har RTK korreksjon.



Figur 64: Odometri data for testing i felt visualisert i MapViz.

Her er den grønne den interne odometrien, som vi ser er den ganske lik i sin bevegelse, men har et stort avvik. Den oransje er GPS-dataen, og den lilla er Kalmanfilteret sitt resultat. Her ser vi også tidspunktet roboten mister RTK-signal ved at linja tar et distinkt hopp.



Figur 65: Tap av RTK-korreksjon. Visualisert i PlotJuggler

En annen visualisering av samme hendelse, der går status signal, som kommer fra GPS-mottakeren, fra to til null, noe som representerer et tap av RTK-korreksjon. Til høgre i Figur 65 er det tydelig også hva dette har å si for den estimerte posisjonen til den mobile basen.

Appendiks F Rapport fra testing i Lærdal

Sted: Blaaflat gård i Lærdal

Dato 12. Mai 2023

Til Stede: Stian Isene, William Antonesen og Raquel Motzfeldt Tirach

Ønsket med denne testen var å få prøvd hele systemet til roboten, helst i en setting så tru til det faktiske bruket vårt system vil ha. Ved å ta denne testen hos Harald Blaaflat Mundal ved Blaaflat gård i Lærdal ønsket vi å se hvordan systemet vårt håndterte seg langs rekker av trær i en gård.

F.1 Forarbeid:

I forkant bestemte vi oss for å planlegge hva som skulle testes og hvordan. Vi vil teste våre implementasjoner, men også enkelte begrensninger til Husky roboten og dens sensorer.

Hva som skal testes:

- Våre løsninger:
    - Angi et område i brukergrensesnittet, for så å gi det til Husky.
    - Gå til angitt GPS-punkt.
    - Baneplanleggingsalgoritme for å dekke et område.
    - Eget kalman filter løsning
        - Kjøre samme aksjon med og uten vår løsning
- Huskyen og dens sensorer:
    - GPS-RTK mottaker.
        - Oppførsel nærme bygg
        - Under trær
    - «Object-avoidance» algoritmen som ligger som en del av TEB local planner

Hvordan skal vi teste:

For å teste våre løsninger ønsker vi et flatt område med åpen himmel for å ha de beste forutsetningene for roboten. Testing inneholder også en plan for å teste under mindre gunstige forhold, der vi kjører Husky roboten igjennom områder som utfordrer GPS-mottakeren samtidig som vi loggfører den dataen.

Det blir å kjøre igjennom en sekvens av våre programmer i dette området, for å teste alle funksjonalitetene det har.

F.2 Testingen:

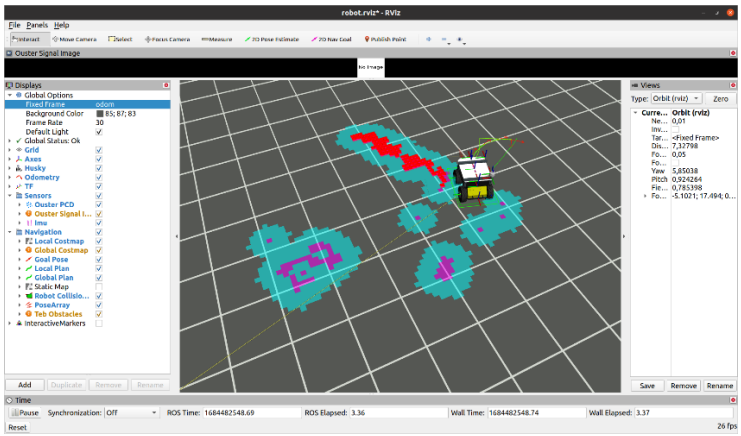
Vi bestemte oss for å reise inn dagen før for å få så mye som mulig ut av dagen, da det er en 3 timer lang kjøretur hver vei. Vi ankom Blaaflat gård klokken 08.00, og etter en introduksjon av området forsøkte vi å starte Husky roboten. Det var ingen kommunikasjon mellom Husky PC og mikrokontrolleren som driver systemet. Vår første tanke var at det var en ledning som hadde kommet løs under transport, så vi skrudde av toppdekselet for å verifisere status av alle koblinger. Vi fant ingen løse ledninger, men da vi hadde skrudd på toppdekselet igjen og testet startsekvensen, koblet den seg

til med en gang. Det er ikke verifisert, men det kan være en ledning som var løs eller så kan det være en komponent som trenger ladning. Batteriet var koblet ut under transport. Dette er noe som må testes ved senere anledninger.

Vi fikk bruke plattingen til Harald som stasjon, og koblet oss opp der. Alle program ble kjørt og testingen kunne starte. Den første testen vi kjørte var av brukergrensesnittet. Der ble det markert en rekke punkter i kartet, og så sendt en henvendelse til ROS-service for lagring av disse punktene i en fil ved å trykke på knappen i brukergrensesnittet. Dette fungerte uten problemer.

Deretter startet testingen for å kjøre Husky roboten, det ble valgt nye punkt og sendt henvendelse til ROS-action serveren for navigering til angitte punkter. Ingenting skjedde, Huskyen stod helt i ro og ville ikke bevege seg. Vi verifiserte at alt var i orden med å kjøre Husky manuelt rundt med joystick-kontrolleren før vi testet å publisere på «/move_base/goal» ROS-topic manuelt, og Husky roboten kjørte da fremover. Det ble utført en sjekk at vårt program publiserte rett til «/move_base/goal» ved å teste programmet som ble supplert av My Bot Shop, «waypoint_follower», og der ble det oppdaget ved å lese på meldingene dette programmet sendte, at de bruker UTM koordinater. Da bestemte vi oss for å utvide vår ROS-action til å også ha et underprogram som bruker disse koordinatene, mens vi utviklet dette ankom Raquel gården.

Vi testet videre med å kjøre til angitte punkt, nå en test med GPS-koordinater og en med UTM-koordinater. Då ingenting virket ble det brukt RViz for å se om det var noe gale med sensorene, og det viste seg at LIDAR sensoren plukket opp gresset rundt Husky og behandlet det som objekter den ikke kunne kjøre over, og da trudde den var innelåst. Det viste seg også at LIDAR sensoren kun fungerte på det initiale lokale kartet, området rett rundt Husky når LIDAR driveren ble initialisert, men ingenting utenom. Det var da mulig å starte LIDAR sensor, for så å kjøre Husky til ett nytt punkt der den ikke oppdaget gresset rundt seg. Dette er grunnen til at det funket å sende navigerinsgmål direkte til «/move_base/goal» etter at vi kjørte Husky med joystick-kontrolleren under feilsøkingen tidligere, den var ikke lenger i det punktet den oppdaget gresset. Problemer med LIDAR sensoren var noe som gikk igjen hele dagen, noen ganger fungerte den fint lenge, og andre ganger ikke i det hele tatt. Vi fant aldri ut av hva det var som førte til denne oppførselen. Viser seg også at «object avoidance» fungerer, da den stopper av seg selv når den oppdager en hindring.



Figur 66: Gress som hindring, visualisert i RViz

Figur 67: Husky blant gress under testing av LIDAR

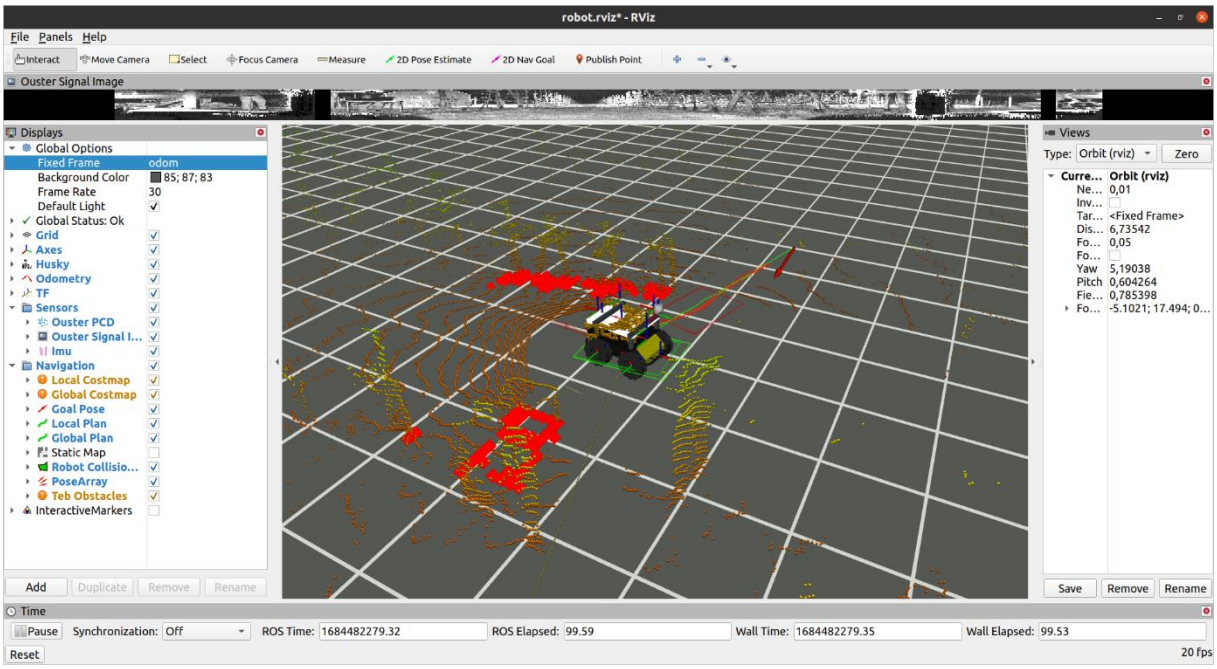
Med dette problemet løst, ble navigering til punkt endelig testet. Ved å navigere seg til UTM punkt var det stor grad av nøyaktighet, punktet som ble navigert til vises på kartet i brukergrensesnittet og det stemte overens med det punktet som ble valgt. Det ble ikke målt den faktiske presisjonen.

Navigering til GPS viste seg å ikke fungere skikkelig, punktene ble ikke transformert ordentlig. Det ble bestemt å skrote videreutvikling av denne navigeringen, da UTM navigering viste seg å fungere på en tilfredstillende måte. Det er også færre steg innvolvert i denne konverteringen, da det kan konverteres direkte fra GPS-koordinater istedenfor å transformeres i forhold til robotens koordinatsystem.

Det ble tatt ROSbag av alle testene vi kjørte.

Under testing av brukergrensesnittet ble det også konkludert med et ønske om å kunne nullstille dataen fra robotbanen, istedenfor å måtte laste siden på nytt. Det vil også være fordelaktig å lagre banedata og definert arbeidsområde på serverside og ikke bare på brukersiden, da dette skapte problemer om man lastet siden på nytt eller lukket den ved et uhell.

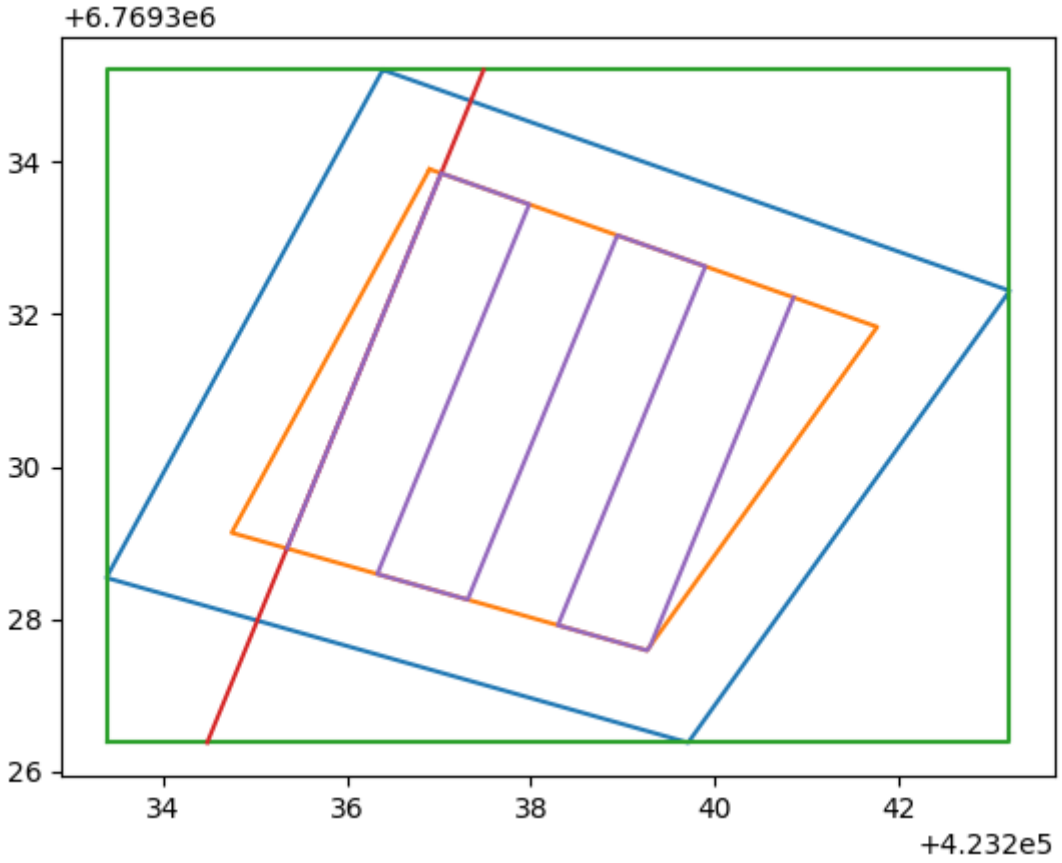
Videre ble det utført tester mellom trærne, da dette er et mer passende arbeidsområde og Raquel ønsket data fra dette for andre deler av prosjektet. Problemet med LIDAR var til stede også her, noen ganger fungerte den også utover kun den initiale posisjonen og igjennom hele testen. Dette medførte en begrenset test i forhold til hva vi hadde ønsket, men ga mye informasjon i forhold til videreutvikling av systemet.



Figur 68: LIDAR data ved frukttrær

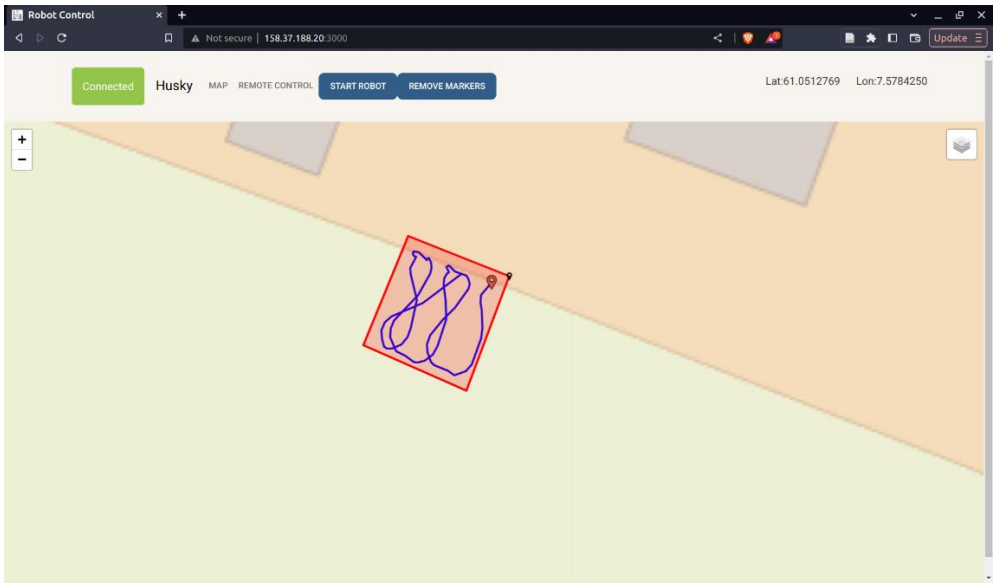
Figur 69: Husky blant frukttrær

Deretter ble baneplanleggeren testet. Den ble originalt utviklet for å bruke kartesiske koordinater som gitt i forhold til den mobile basen, så det å bruke den til å regne ut UTM viste seg å ikke være noe problem da disse også er representert med kartesiske koordinater. Vi testet på et grusområde, da dette ville minimere problemene med LIDAR sensoren. Området ble bestemt, og satt inn manuelt i baneplanleggeren, og banen ble kalkulert. Det er noen feil med denne banen, da bunnlinja ikke er korrekt, men de parallelle linjene er fungerende.



Figur 70: Planlagt bane i felt test

Selve kjøringen av baneplanlegger viste en svakhet ved vår løsning. Den banen vil kreve en orientering, da TEB planleggeren ikke bare baserer seg på posisjon, men også orientering. I vår løsning gir vi den en null verdi i orientering, og det fører til en bue på banen, da det blir tatt med i planleggingen av banen. Under testing ble roboten observert å krysse grensene vi hadde gitt den, det vil da være fordelaktig å legge inn en form for begrensning på området den har lov å bevege seg innenfor.



Figur 71: Bane fullført innen angitt område

Vi avsluttet dagen med å kjøre Husky roboten i det svært bratte terrenget som var en del av gården til Harald, der tok vi flere ROSbag for å identifisere hvordan den oppfører seg i slikt krevende terreng.

Figur 72: Husky i bratt terreng

Det ble ikke tid for testing av egen Kalmanfilter løsning.

F.3 Oppsummering

Testingen gav mye god informasjon, spesielt på svakheter og veien videre for dette prosjektet. I helhet fungerte systemet overaskende godt, med tydelige svakheter i enkelte implementeringer, og vi er godt fornøgde med dagen.

Takk til Harald for lån av gård og Raquel for god veiledning.