Høgskulen på Vestlandet

BACHELOROPPGAVE:

# BO22EB-29 Selvkjørende mobil robot for datainnsamling fra frukt- og bærproduksjon på Vestlandet

Chris Louis Johnsen
Aril Torheim
Isak Vamråk Førde

30. mai. 2022

# Dokumentkontroll

| *Rapportens tittel:* BO22EB-29 Selvkjørende mobil robot for datainnsamling fra frukt- og bærproduksjon på Vestlandet | *Dato/Versjon* 30. mai. 2022/2.0 |
|---|---|
| | *Rapportnummer:* B022EB-29 |
| *Forfatter(e):* Chris Louis Johnsen<br>Aril Torheim<br>Isak Vamråk Førde | *Studieretning:* Automatisering med robotikk |
| | *Antall sider m/vedlegg* 74 sider + zip fil |
| *Høgskolens veileder:* Martin Fodstad Stølen | *Gradering:* Åpen |
| *Eventuelle Merknader:* Vi tillater at oppgaven kan publiseres. | |

| *Oppdragsgiver*: Høgskulen på Vestlandet | *Oppdragsgivers referanse:* |
|---|---|
| *Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon):* | |

| **Revisjon** | **Dato** | **Status** | **Utført av** |
|---|---|---|---|
| 0.1 | 22.04 | Satt opp forslag til overskrifter | Aril.T |
| 1.0 | 28.05 | Første utkast | Aril. T |
| 1.1 | 29.05 | Korreksjonslesing | Isak.F |
| 2.0 | 30.05 | Ferdigstilt rapport | Aril.T,CLJ,Isak.F. |
| | | | |
| | | | |

# Forord

Denne rapporten er skrevet i forbindelse med vår avsluttende bacheloroppgave i Automatisering med Robotikk ved Høgskulen på Vestlandet våren 2022. Oppgaven har bestått av både teoretisk og praktisk arbeid omhandlende det å utvikle programvare til en robot som skal kunne navigere rundt i et bærproduksjonsmiljø.

Vi ønsker å takke veileder Martin Fodstad Stølen for en spennende oppgave, og for inspirerende oppfølging underveis. Vi ønsker også å takke Raquel Motzfeldt Tirach ved robotlabben i Førde for god hjelp med Husky robotplattformen underveis i prosjektet.

Takk til robotlabben ved Høgskulen på Vestlandet i Førde for lån av Husky robotplattformen. Takk til Teknoløftet og Njøs Frukt- og Bærsenter for at vi fikk utføre testing på RoboVest testfasilitet i Leikanger.

# Sammendrag

Dette bachelorprosjektet er gjennomført med mål om å utvikle et robotsystem basert på en kommersiell robotplattform som klarer å navigere langs bærrekker i et bærproduksjonsmiljø. Målet er at roboten i fremtiden skal kunne samle inn data for å overvåke sykdommer og estimere forventet produksjon. Tilhørende sikkerhetssystemer har også blitt utviklet i form av algoritmer som skal oppdage hindringer ved hjelp av kamera og en støtfanger med trykkbrytere som skal stanse roboten dersom den kolliderer med hindringer.

Robotplattformen som har blitt benyttet i oppgaven er en Husky UGV fra Clearpath Robotics. Det har blitt montert et realsense D435i kamera på roboten for å kunne gi roboten data fra omgivelsene. I tillegg har GPS-RTK systemet som er montert på roboten blitt forsøkt benyttet. Programvaren har blitt utviklet i Robot Operating System og med programmeringsspråket Python.

En simuleringsverden utformet som et bærproduksjonsområde har også blitt utviklet. Simulasjonsverdenen kan kjøres i programvaren Gazebo i Robot Operating System og har blitt brukt til å teste ut robotsystemet underveis.

Systemet klarer å utføre navigasjon i et begrenset område, for eksempel innenfor en bærtunnel. På grunn av at en programvare som fulgte med robotplattformen ikke virket ved slutten av prosjektperioden har systemet noen begrensninger knyttet til navigasjon over et større område. Robotsystemet har blitt testet ved RoboVest testfasilitet i Leikanger, og det klarer å utføre oppgavene det ble satt til der. Sikkerhetssystemet som ble utviklet har også vist seg å være robust.

# 1 Innhold

# 2 Innledning

## 2.1 Oppdragsgiver

Oppdragsgiver for denne oppgaven er Høgskulen på Vestlandet (HVL). Høgskulen tilbyr utdanninger innen ingeniør- og økonomifag, helse- og sosialfag, lærerutdanning og samfunnsfag. Høgskulen har 16 000 studenter fordelt på campuser i Bergen, Førde, Sogndal, Stord og Haugesund [1]. HVL deltar for tiden i forskningsprosjektet Teknoløft Sogn og Fjordane. I dette prosjektet skal testanlegget RoboVest åpnes. Ved RoboVest skal HVL utforske hvordan bringebærdyrking på Vestlandet kan se ut i fremtiden, med innslag av blant annet sensorer og robotikk [2].

Veileder for oppgaven er Martin Fodstad Stølen. Martin er førsteamanuensis ved instituttet for datateknologi, elektroteknologi og realfag ved HVL og har bred erfaring innenfor forskning, utvikling og innovasjon. Han er også grunnlegger av firmaet Fieldwork Robotics. Dette selskap utvikler roboter som høster frukt og grønnsaker for dyrkere [3].

## 2.2 Problemstilling

Hentet fra oppgavetekst:

*«De siste årene har for alvor avdekket sårbarheten i frukt- og bærproduksjonen, med avhengighet av utenlandsk arbeidskraft og risiko både med tanke på tilgang og smittevern. Samtidig dukker det nesten årlig opp saker i media der bærplukkere føler seg utnyttet. Dette er svært uheldig både for dyrkingen sitt renommé og for norsk frukt og bær generelt. Nye løsninger for å gjøre produsentene mer uavhengige av utenlandsk arbeidskraft og sikre en forutsigbar plukkekostnad pr. kg er avgjørende for framtiden i norsk frukt og bærproduksjon. Et første skritt i denne retningen er å ha autonome mobile roboter som kan navigere i typiske dyrkningsmiljø for frukt og bær, og kunne hjelpe dyrkeren ha oversikt over tilstanden på plantemasse og avling.* [4]*»*

Denne oppgaven er avgrenset til å utvikle en autonom robotløsning som kan navigere rundt i et dyrkningsmiljø. Oppgaven skal senere kunne videreutvikles til å samle inn data fra miljøet ved å montere sensorikk for dette formålet på robotplattformen. Denne dataen skal kunne gi dyrkeren informasjon om forventet produksjon, og oppdage forekomst av sykdommer som råte. Målet er at dette senere kan videreutvikles til en løsning som overvåker og høster bringebær helt autonomt.

## 2.3 Hovedidé for løsningsforslag

Prosjektet har tilgang til en kommersiell robot av typen Clearpath Robotics Husky, lokalisert i Førde. Denne roboten er integrert med Robot Operating System (ROS) og har GPS-RTK og LIDAR sensorer installert. Roboten har også software som muliggjør veipunkts navigasjon mellom GPS koordinater. Prosjektet skal videreutvikle denne robotløsningen til å kunne navigere mellom bringebærrekker i et produksjonsmiljø, muligens inne i en plasttunnel. Datainnsamlingen som roboten senere skal utføre skal gjennomføres med 3D-kameraer, og det er derfor ønskelig at navigeringen mellom bringebærrekkene kan utføres ved hjelp av data fra 3D-kamera også.

Oppdragsgiver ønsker også at det skal utvikles en fysisk sensor som kan fungere som en nødstopp for roboten dersom andre sikkerhetssystemer svikter. Denne sensoren kan for eksempel være utformet som en støtfanger på fronten av roboten, og skal stoppe roboten dersom den kjører i hindringer.

Software skal utvikles i ROS, og det vil være hensiktsmessig å først testet systemet i Gazebo simulatoren. Det vil ved slutten av prosjektperioden bli mulighet for å teste systemet på hardware ved å frakte Huskyen til RoboVest testfasilitet ved Njøs frukt- og bærsenter i Leikanger [4].

Målet med oppgaven kan kort oppsummeres i følgende punkter:

- Gjøre robotplattformen i stand til å navigere rundt i produksjonsmiljøet på en måte som er hensiktsmessig for å utføre datainnsamlingsoppgavene som den er tiltenkt.
- Implementere hensiktsmessige sikkerhetsbarrierer som beskytter materiell og personell i produksjonsmiljøet mot farer som robotplattformen kan forårsake.
- Koble opp softwaren mot den fysiske robotplattformen for å teste om simuleringsresultat sammenfaller med en reell test.

# 3 Kravspesifikasjon

Oppdragsgiver hadde i utgangspunktet ingen kravspesifikasjon for systemet utover beskrivelse av hva roboten skulle brukes til. Som et hjelpemiddel mot design og testprosessen satt prosjektgruppen opp noen krav til systemet. Disse kravene ble godkjent av oppdragsgiver. Kravene er delt opp i underkategoriene sikkerhet, navigering og simulering.

**Sikkerhet:**

- Robot skal kunne oppdage hindringer som eksempelvis mennesker ved hjelp av 3D-kamera, og stoppe med minst 30 cm klaring fra objektet.
- I tillegg skal robot ha en frontfanger som ekstra sikkerhetsbarriere som stopper dersom robot kjører i en hindring. Sikkerhetsbarrieren skal designes slik at robot ikke gjør skade på mennesker, dyr eller gjenstander dersom de skulle stå i veien.

**Navigering:**

- Roboten skal kunne naviger fra start av en bærrekke til andre enden av en bærrekke ved hjelp av 3D-kamera. Roboten skal kjøre sentrert mellom rekkene. Et avvik på ±10cm kan bli godtatt.
- Roboten skal kunne navigere fra parkeringsstasjon eller ende av bærrekke til start av neste bærrekke ved hjelp av egnet navigasjonsstrategi og sensorsystem.
- Roboten skal utføre nevnte navigeringsoppgave med en fart på minst 1 km/t.

**Simulering:**

- Simulering av verden skal inneholde 3 bærrekker og 1 parkeringsstasjon.
- Simuleringsverden skal inneholde hindringer for test av navigeringssystem.

# 4 Analyse av problem og mulige løsninger

Hele problemstillingen med den avgrensningen som er satt kan i korte trekk beskrives som et robot-navigeringsproblem med tilhørende sikkerhetsbarrierer. En stor del av oppgaven omhandler også simulering, men denne simuleringen blir kun utført for å nå det overordnende målet med å gjøre roboten i stand til å navigere autonomt rundt i et bærproduksjonsmiljø.

## 4.1 Robot navigering og navigeringsstrategier

Robot navigering omhandler problemet med å få en robot til å bevege seg til et mål. Tradisjonell navigering blant mennesker har blitt utført ved å følge et kart eller skilter. Enkelte robotoppgaver kan derimot bli utført uten noen form for kart. Dette blir kalt for reaktiv navigasjon. Roboten navigerer da med å for eksempel kjøre mot et lys, følge veggene igjennom en labyrint eller kjøre i tilfeldige retninger slik som en robotstøvsuger gjør [5].

Roboter kan også bruke et kart for å navigere. Dette gjør robotnavigeringen straks mer komplisert. Roboten trenger da et kart av miljøet den skal bevege seg i, og den må også kunne kontinuerlig finne ut hvor på kartet den befinner seg [5].

Navigeringen som denne roboten skal utføre kan deles inn i to faser:

- Navigering frem til bærrekke
- Sentrering av robot mellom bærrekker

For å navigere frem til bærrekken vil det være behov for å lokalisere hvor i verden roboten befinner seg i forhold til dette målet. Husky roboten har installert GPS-RTK system om bord som kan brukes til dette. Dette er et posisjoneringssystem med centimeterpresisjon, noe som er mer enn nøyaktig nok til å posisjonere roboten ved enden av en bærrekke [6]. Hvilke navigasjonsstrategier som blir brukt for å navigere ved hjelp av GPS-RTK sensoren beskrives nærmere i underkapittel.

Et GPS-signal vil ha behov for noe sikt mot himmelen for å kunne beregne sin posisjon. Dette vil ikke roboten alltid ha når den skal sentrere seg mellom bærrekker da disse ofte er plassert inne i plasttunneler. Det vil da være behov for en reaktiv navigasjonsstrategi som kan navigere roboten inne i teltet frem til den får GPS-signalet tilbake. Bringebærbuskene står på rette rektangulære rekker uten hindringer. Denne symmetrien og utformingen fører til en relativt lav kompleksitet i selve navigeringen, og en reaktiv navigasjonsstrategi vil derfor være egnet. Innenfor reaktiv navigering er braitenberg kjøretøy en underkategori. Dette er en type kjøretøy som navigerer mot et lys, og styrer fart og sving basert på en sensor som oppfatter dette lyset [5]. Ved å modifisere dette konseptet til problemet i denne oppgaven ville navigeringsproblemet forhåpentligvis bli løst. Modifiseringen blir å implementere en strategi som kjører roboten fremover med en konstant hastighet, og regulerer svingen på roboten basert på input fra en sensor. Når sensoren detekterer at roboten har nådd enden av bærrekken har den nådd målet og skal stanse.

Utviklingen av simuleringsmiljø, algoritmer og sikkerhetssystem blir videre beskrevet i underkapitlene, men først litt om Husky roboten og ROS.

## 4.2 Husky UGV

Husky UGV (unmanned ground vehicle) er en mobil robot plattform som er laget for å kunne brukes i ujevnt terreng. Roboten kan utstyres med en rekke ulike typer sensorer og manipulatorer. Huskyen leveres med enkodere med høy oppløsning og en nøyaktig kontroller noe som gjør at den kan kjøres i lave hastigheter, det gjør også at den kan navigere på «dead reckoning» med relativt høy nøyaktighet [7]. Roboten som er tilgjengelig for dette prosjektet er levert med følgende utstyr fra fabrikk:

- Lidar: Ouster OS1 3D
- GPS-mottaker: Emlid Reach RS2
- Inertial Measurement Unit: CH Robotics UM7
- Computer: Mini-ITX
- Ekstra computer for tung databehandling: Nvidia Jetson Xavier
- Wi-Fi router
- Software for "autonomous waypoint navigation using GPS-RTK"

Flere sensorer kan enkelt monteres på roboten og tilkobles robotens computer via USB, ethernet eller WIFI.


***Figur 1: Husky UGV robotplattformen i Førde***

Roboten har en maksfart på 1.0 m/s, og styres med differential drive prinsippet. Dette betyr at roboten svinger ved å kjøre hjulene på den ene siden bakover og hjulene på den andre siden fremover, roboten vil da kunne vinge rundt seg selv der den står. Bevegelsen til roboten i et XY-plan kan beskrives med følgende ligninger:

$$\dot{x} = v\cos\theta$$

$$\dot{y} = v\sin\theta$$

$$\dot{\theta} = \frac{v_{høyre} - v_{venstre}}{hjulavstand}$$

Roboten styres ved å publisere lineære og angulære hastighet til et av roboten sin /cmd_vel topic. Lineære x hastigheter får roboten til å kjøre frem eller bak, angulære z hastighet får roboten til å rotere rundt sin egen akse. Ved å kombinere disse to hastighetene kan roboten styres i ønsket retning.

Roboten kan styres manuelt med medfølgende trådløs joystick kontroller. Robotens software kan aksesseres med ekstern computer ved hjelp av Secure Shell (SSH) over robotens Wi-Fi router.

## 4.3 Robot Operating System

*Figur 2: ROS logo*

ROS er et open-source operativsystem, som arbeider på et annet operativsystem, fortrinnsvis Linux. ROS brukes av forskere og utviklere til å bygge og gjenbruke kode for robotapplikasjoner. ROS utfører en rekke tjenester som maskinvareabstraksjon, kommunikasjon mellom prosesser og pakke håndtering. ROS inneholder også verktøy og biblioteker for å bygge, skrive og kjøre kode på tvers av flere datamaskiner. ROS har støtte for en rekke programmeringsspråk blant annet Python, C++ og Lisp [8].

De viktigste komponentene for å forstå hvordan ROS fungerer er følgende:

**Pakker**

Software i ROS blir delt opp i pakker (packages). En pakke kan inneholde blant annet noder, biblioteker eller tredjeparts software. Målet med å dele opp software i ulike pakker er å samle software som hører sammen i en pakke som er lett å bruke og dele [9].

**Noder**

En node er et program som gjør beregninger. Eksempler på noder er programmer som leser data fra lasere og gjør beregninger på disse dataene, eller programmer som kontrollerer hjulene på en robot. Det er hensiktsmessig å dele beregninger som blir utført i et robotsystem opp i mange noder da dette øker feil toleransen og senker kompleksiteten på koden [10]. Dette gjør også at det blir enklere å feilsøke i systemet ved at man lettere kan lokalisere i hvilken node feilkode befinner seg.

**Topics**

Topics er busser som noder kommuniserer via. En node kan enten abonnere eller publisere i et topic, og på den måten sende eller motta data fra en annen node. Topics gir en kontinuerlig stream av data mellom noder. Det kontrolleres ikke hvem som sender og mottar data, og om dataen kommer frem [11]. For å forklare topics kan de forenklet sammenlignes med UDP-kommunikasjon. Dataen som nodene sender via topics er definert via messages. En message kan inneholde mange ulike data og datatyper i en og samme message. Messages er definert via msg filer.

**Services**

Services er en annen tjeneste som noder kommuniserer via. Til forskjell fra topics som streamer kontinuerlig, tar en service imot en forespørsel, og sender en tilbakemelding når forespørselen er utført. Dette kan for eksempel brukes dersom det er ønskelig å kjøre roboten rett frem til den detekterer et objekt. Da kan dette sendes som et service kall, og service noden gir tilbakemelding når roboten har stoppet foran objektet. Services er definert i srv filer [12].

**Action servers**

Dersom det er ønskelig med tilbakemelding når en service kjører, må det i stedet brukes en action server. En action server tar imot et mål, gir feedback underveis, og sender et resultat når målet er nådd. En action kan også kanselleres underveis [13]. En action server kan for eksempel brukes dersom det blir sendt en koordinat som roboten skal navigere til, og det er ønskelig at roboten sender data om hvor den er lokalisert imens den navigerer til dette målet.

## 4.4 Modellering av robot og verden i Gazebo

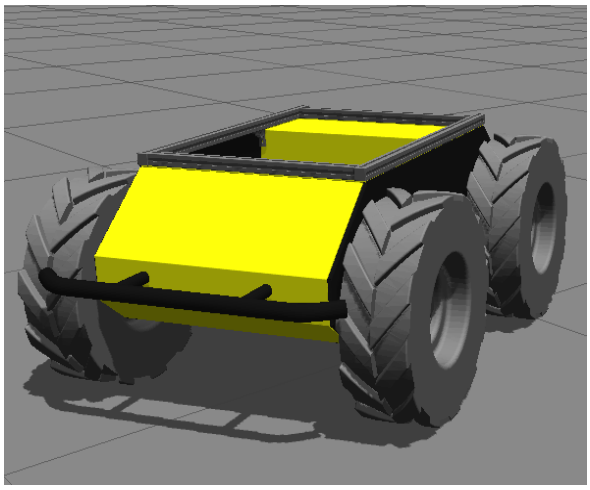
Siden Husky robotplattformen befant seg i Førde, testlokasjonen var i Leikanger og studentene holdt til i Bergen var det behov for å utvikle en digital tvilling slik at software kunne utvikles og testes underveis i prosjektet.

### 4.4.1 Digital tvilling

En digital tvilling er en virtuell modell som er basert på en fysisk modell, eksempelvis en robot som er utstyrt med forskjellige sensorer. Sensorene på en robot leser data som avstand, hastighet og posisjon. Ved å lage en digital tvilling som simulerer roboten og verdenen den skal bevege seg i vil dette kunne gi nyttig informasjon om hvordan roboten vil oppføre seg i den ekte verdenen.

Ifølge Andre Linz ved HS Osnabrück er det svært tidkrevende og kostbart å gjøre tester i felt, da man er avhengig av både personell, reising og gode værforhold. De lager derfor digitale tvillinger for alle sine roboter der de simulerer systemet før de implementerer algoritmene på roboten og går i felt [14].

Det finnes allerede pakker fra Clearpath Robotics som simulerer Husky roboten. Det var derfor kun behov for å bygge en verden som simulerer bærproduksjonsområdet i Leikanger. Denne verdenen ble bygget i ROS simulatoren Gazebo, og sensordataen ble lest med ROS programmet Rviz.



*Figur 3: Digital tvilling*


*Figur 4: Husky robotplattform*

### 4.4.2 Gazebo

Gazebo er en open-source simulator som er installert i ROS. Simulatoren kan blant annet simulere avanserte roboter, fysikk og sensordata. Gazebo er intuitiv i bruk og er mye brukt i utdanning og i industrien.

*Figur 5: Gazebo logo*

En robot kan simuleres i Gazebo ved hjelp av URDF filer. URDF filen definerer hvordan robot ser ut, hvilke bevegelige deler og ledd den har, og hvilke sensorer som er montert på den. Basert på dette kan Gazebo simulere hvordan roboten beveger seg rundt i verdenen, og hvilke data sensorene oppfatter. For å lese sensordata må andre programmer brukes, for eksempel Rviz.

Verdenen som Gazebo skal simulere defineres i en world fil. I denne world filen blir fysiske parametere som lys, gravitasjon og friksjon satt. Her blir det også definert hvilke modeller som skal plasseres ut i verdenen. Dette gjøres ved bruk av «x, y, z, r, p, y» koordinater for plassering og rotasjon, i tillegg til en link til filen som beskriver modellen som skal plasseres ut. Gazebo kan lese 3D-modeller bygget opp av mesh filer i formatene STL, DAE og OBJ. Dette er formater som kan beskrives med XML filer.

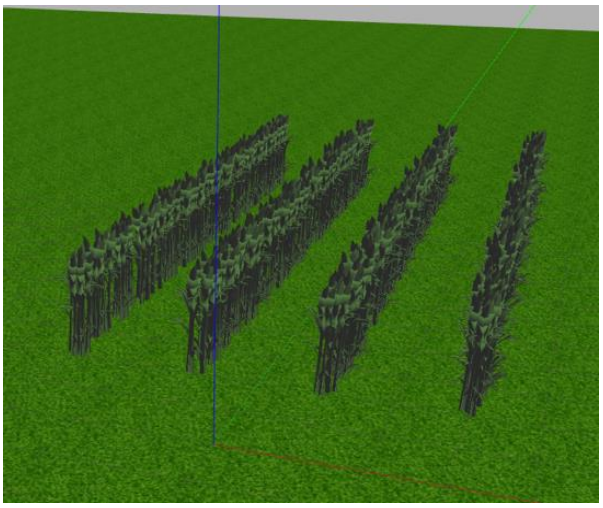
### 4.4.3 Design prosess

I prosjektet ble Gazebo brukt til å bygge en simulert verden i henhold til kravspesifikasjonene. Figur 6 under ble brukt som utgangspunkt for å lage en verden som lignet mest mulig på et bringebærproduksjonsområde. Bildet er tatt fra produksjonsområdet på NJØS i Leikanger.

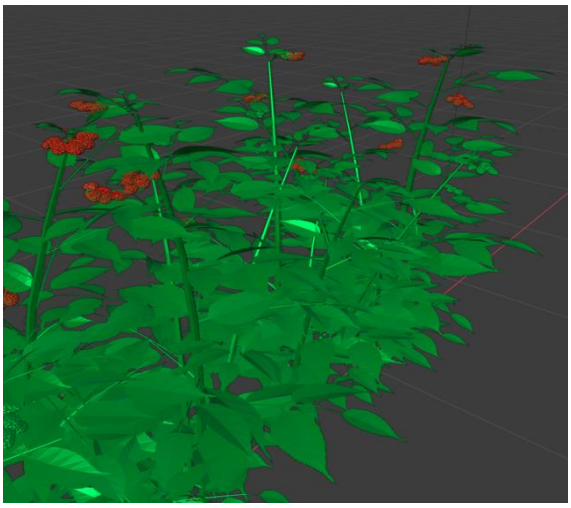
*Figur 6: Bringebærrekker, Njøs Leikanger*

Verdenen ble bygget opp av bærrekker plassert inne i drivhustunneler for at simulasjonen skulle gjenspeile området der testene for den fysiske roboten skulle utføres. Avstander, dimensjonering og utforming av bringebærbuskmodeller ble prøvd gjort mest mulig lik de virkelige omgivelsene slik at simulasjonen ville gi mest mulig realistiske svar på hvordan algoritmene ville oppføre seg på den fysiske roboten.

I tidlige modeller ble det forsøkt å bruke modeller med lav polygon for å spare datakraft, men for å lage en mest mulig realistisk simulering ble det bestemt at mer realistiske modeller måtte brukes. Figuren under viser den første og den endelige modellen som ble brukt.



*Figur 7: Første bringebærbuskmodell*

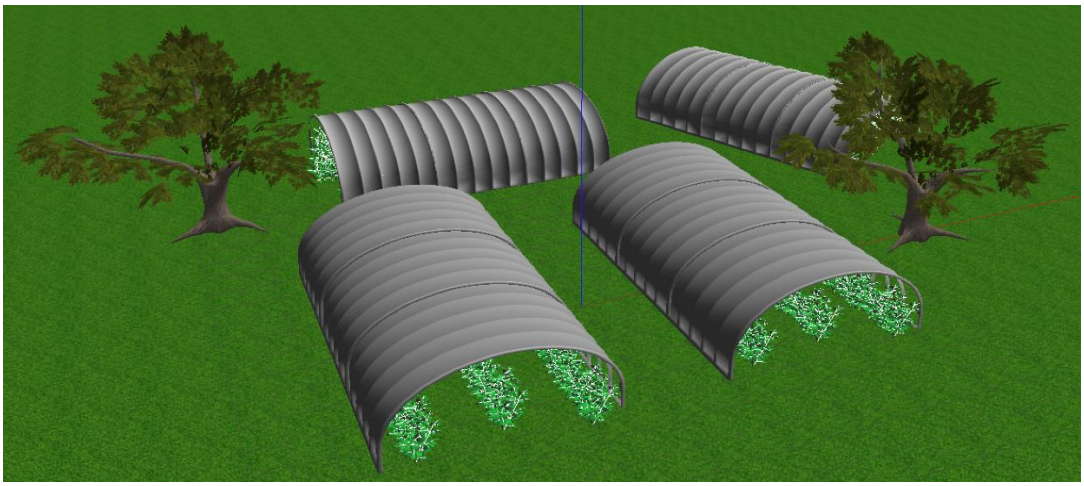


*Figur 8: Endelige bringebærbuskmodell*

Et problem ved å ikke bruke modeller med lav polygon var at det krevde mye PC kraft å kjøre simuleringene. Modellene måtte derfor skaleres noe ned da simuleringen kjørte med rundt 5 – 10 FPS uten å kjøre noe som helst av algoritme til roboten.

For å skalere ned modellene ble 3D-modelleringsverktøyet Blender og mesh-fil manipuleringsverktøyet Meshlab brukt. Blender ble først brukt til å destruere polygons slik at kvaliteten på 3D-modellene ble noe dårligere, der modellen fortsatt beholdt sin form. Deretter ble Meshlab brukt for å fjerne usynlige hjørner og polygons som var > 0.5cm. Med de optimaliserte 3D-modellene kjørte simulasjonen jevnt med 50+ FPS.

Den ferdige simuleringsverdenen er vist i figuren under.



*Figur 9: Ferdig utviklet simuleringsverden*

## 4.5 Sentrering av robot i bærrekker

### 4.5.1 RGB-D kamera og OpenCV

Algoritmen som skal sentrere roboten mellom to bærrekker skal gjøre dette ved hjelp av data fra et RGB-D kamera. RGB-D kameraer er mye brukt i detektering av frukt i jordbruksapplikasjoner [15], det er derfor sannsynlig at et slikt kamera vil bli brukt når datainnsamlingsalgoritmene for denne roboten skal utvikles. Det er da en fordel å kunne bruke hardware som uansett må monteres på roboten ved en senere anledning til rekkesentrering.

Et RGB-D kamera fanger et 2D-bilde bestående av piksler med ulik intensitet i rød, gul blå spekteret slik som et vanlig kamera. I tillegg fanges dybden på pikslene i bildet slik at bildeprosesserings-algoritmene vet hvor langt borte fra kamera RGB pikslene befinner seg. Bildet kan derfor settes sammen til et bilde i 3D-rommet. Det brukes hovedsakelig tre teknikker for å detektere dybden på pikslene i et RGB-D kamera [16]:

- Structured light: Et infrarødt lys mønster prosjekteres ved hjelp av en lyskilde over et objekt. Et kamera vil detektere mønsteret på objektet, og mønsteret vil endre seg basert på geometrien til objektet. Dette brukes til å beregne avstanden til pikslene på objektet.
- Time of flight: En infrarød stråle blir sendt mot den aktuelle pikselen som avstanden skal kalkuleres til. Dersom strålen treffer et objekt, vil den reflekteres tilbake til kameraet. Siden lysets hastighet er kjent, kan distansen til pikselen beregnes ved å måle tiden det tar for strålen å bli reflektert.
- Active infrared stereo: Bruker triangulering for å beregne distansen til hver enkel piksel. Objektet blir sett fra to forskjellige kamera linser. Siden distansen mellom linsene er kjent, kan distansen til pikselen beregnes basert på hvordan de to kameralinsene oppfatter objektet. Forskjellen fra et vanlig stereokamera er at dette har en aktiv lyskilde som sender ut et infrarødt lys som prosjekteres på objektet.

Alternativet til et RGB-D kamera kunne en LIDAR blitt brukt. Dette er en laser som sender ut infrarødt lys, og bruker time of flight prinsippet for å måle avstanden til objekter i retningene innenfor måleradiusen. Bakdelen med en LIDAR i forhold til et RGB-D kamera er at dersom laserstrålene treffer i punkter på busken som ikke har blader eller greiner vil ikke busken bli detektert. RGB-D kamera vil sannsynligvis derfor gi mer robuste målinger enn en LIDAR.

Kameraet som har blitt montert på Huskyen for å teste algoritmene er et Intel realsense 435i. Dette kameraet bruker active infrared stereo teknologien for å beregne dybden på pikslene, og kan beregne dybde bildene med en hastighet på opptil 90 FPS. RGB linsen har en hastighet på 30 FPS [17]. Dette skal være raskt nok til å gi stabil data til en kontroller som kan kontrollere bevegelsene til roboten.

For å analysere data fra RGB-D kamera ble blant annet OpenCV benyttet. OpenCV er et open-source software bibliotek som brukes til å utvikle computer vision applikasjoner. Biblioteket har støtte mot Python, C++, Java og MATLAB [18]. For å kunne brukes sammen med ROS brukes cv_bridge fra vision_opencv pakken. Denne pakken inneholder metoder som konverterer bilder mellom datatypen ROS sender bilder over topics på (sensor_msgs.msg), over til datatypen som OpenCV bruker for å analysere bildene.

Videre er de ulike algoritmene som er utviklet som forslag til endelige løsninger beskrevet. Utviklingen var en iterativ prosess der konsepter ble testet ut, svakheter ble funnet og algoritmen videreutviklet.

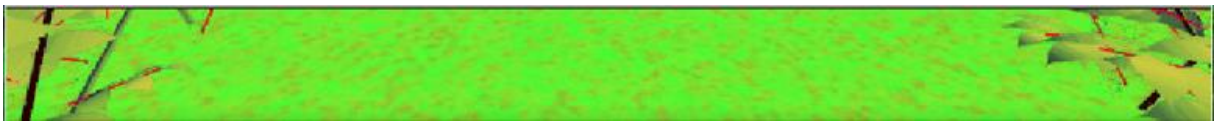
### 4.5.2 RGB-filter algoritme

Den første algoritmen som ble utviklet benytter et fargefilter for å detektere bringebærbuskene som roboten skal sentrere seg mellom. Algoritmen ble utviklet ved å bruke tutorialen «Line Follow with OpenCV» fra «The Construct» som template [19]. Dette gjorde det mulig å starte utvikling av sentreringsalgoritmer uten kunnskaper om OpenCV fra før.

*Figur 10: Stream fra realsense RGB kameralinse*

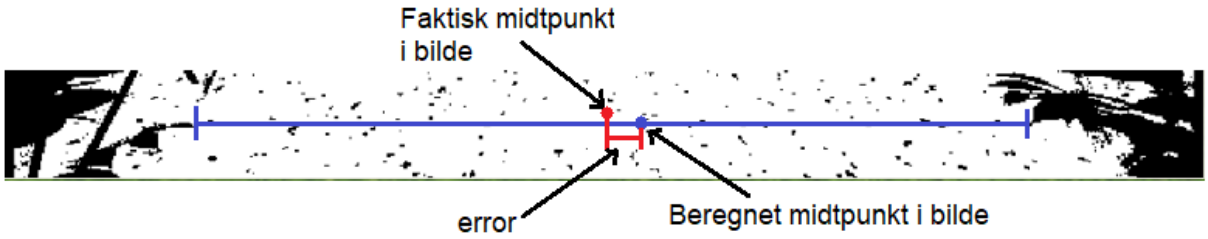
Algoritmen henter inn stream fra RGB-kameralinsen og klipper ut en stripe fra midten av bildet. Denne stripen inneholder dataen vi trenger for å detektere midtpunktet mellom bringebærrekkene. Ved å klippe ned bilde bruker algoritmen mindre datakraft under videre bildebehandling.

Bildet konverteres deretter til HSV farge spekteret. Dette fargespekter ligger nærmere hvordan det menneskelige øyet oppfatter farger enn RGB [20]. HSV er ofte brukt i computer vision da det gjør det enklere å skille farger fra hverandre.



*Figur 11: RGB-algoritme bilde i HSV farge spekter*

Bildet blir til slutt behandlet med en maske som setter farger innenfor et bestemt HSV spekter til svart, og alle andre farger til hvit for å skille ut bringebærbuskene i bildet.




*Figur 12: RGB-algoritme bilde masket med forklaring av error kalkulasjon*

Algoritmen regner ut hvor den første svarte pikselen på høyre og venstre siden ut fra midtpunktet av bildet er. Midtpunktet mellom disse punktene blir så beregnet, og en error mellom midtpunktet mellom de første svarte pikslene og det faktiske midtpunktet i bildet forteller om roboten er sentrert. Ved error signal = 0 er roboten sentrert.

Error signalet blir sendt inn i en enkel P-regulator som skalerer error signalet slik at det gir et passende pådrag til roboten. Dette pådraget blir publisert som en angulær hastighet rundt z-aksen til robotens /cmd_vel topic. Dette får roboten til å svinge for å prøve å sentrere roboten. Mens algoritmen kjører,

blir det også publisert en konstant hastighet i lineær x-retning. Dette betyr at robotkontrolleren kun regulerer på sving.

Svakheter med denne algoritmen er at det vil være vanskelig å skille farger på ekte bringebærbusker fra andre farger som finnes i bringebærproduksjonen. Algoritmen vil også bli påvirket av lysforholdene i produksjonen. Følg link for å se demo for kjøring av algoritmen i simulering.

https://www.youtube.com/watch?v=4meaBIZm8po

### 4.5.3 Edge detection algoritme

Edge detection algoritmen bygger videre på RGB algoritmen, men med mer avanserte OpenCV funksjoner. En bringebærbusk vil ha mange kanter, da alle blader og greiner vil danne en kant. Algoritmen vil detekterer kanter i bildestream fra realsense RGB-kameralinsen og dermed detektere hvor bringebærbuskene befinner seg i bildet. Kantene detekteres ved hjelp av OpenCV funksjonen Canny Edge Detection.

Først blures bildet for å fjerne støy, slik at kun de mest tydelige kantene blir detektert. Dette gjøres ved å bruke et gaussian blur filter. Dette filteret setter hver enkelt piksel til snittet av fargene i de omliggende pikslene. Snittingen baseres på en Gaus kurve der piksel verdier som ligger langt vekk i fra verdien til pikselen som skal snittes har mindre å si for resultatet enn de pikslene som ligger nærme i verdier [21].

*Figur 13: Edge detection blurred*

Kantene blir deretter detektert ved hjelp av OpenCV canny funksjon. Canny detekterer kanter ved å se på endringen i intensiteten i nabopiksler både i x og y retning. Gradientene i bildet blir så beregnet og i de punktene der magnituden til gradienten er høy er det sannsynlig at det finnes en kant. Canny ser deretter på nabo gradienter og fjerner kanter som ikke oppfyller parameterne som er satt for funksjonen. Dette gjøres for å fjerne falske kanter [22].

*Figur 14: Edge detection canny*

OpenCV dilate funksjon blir brukt for å gjøre kantene tydeligere, og bitwiseNot funksjon blir brukt for å gjøre svarte farger hvit, og hvite farger svart. Ved å gjøre dette kan samme kontroller som er brukt i RGB-filter algoritmen brukes for å styre roboten til å sentreres mellom bringebærrekkene.

*Figur 15: Edge detection dilated og bitwiseNot operasjon*

Svakheter med denne algoritmen er at det sannsynligvis vil være mange flere kanter som blir detektert i et bringebærproduksjonsmiljø enn i et simulert miljø, og det vil sannsynligvis kreve mye tuning av parametere for å få algoritmen til å fungere i felt. Algoritmen vil også sannsynligvis bli påvirket av lysforhold i felt, noe som kan gjøre den lite robust. Algoritmen fungerer derimot bra i simulasjon. Se video for demonstrasjon.

https://www.youtube.com/watch?v=t1uxctGT4Ck

### 4.5.4 Depth detection algoritme

I motsetning til de to foregående algoritmene bruker depth detection algoritmen realsense kameraets dybde bilde. Dette er et svart-hvitt bilde som kun består av en fargekanal. Intensiteten på pikslene i bildet går fra 0-255, der intensitet 0 er helt svart og 255 er helt hvit. Piksler som er nærme kamera vil være svarte, og piksler som befinner seg langt borte vil være hvite. Bildet er dermed en representasjon for hvor langt borte fra kameralinsen objektene som blir filmet befinner seg.

*Figur 16: Depth detection algoritme dybde bilde*

Ved å bruke en maske på dette bildet der piksler innenfor et gitt spekter filtreres ut vil buskene på siden kunne detekteres. Ved å bruke samme kontroller som i de to foregående algoritmene vil roboten kunne styres til å være sentrert mellom bringebærbuskene.

*Figur 17: Depth detection masket bilde*

Det ville også vært mulig å bruke andre typer filter for å detektere bringebærbuskene i bildet. Dette ble ikke brukt tid på da den neste algoritmen bruker noe av det samme konseptet. Video demonstrasjon av algoritmen kan sees her:

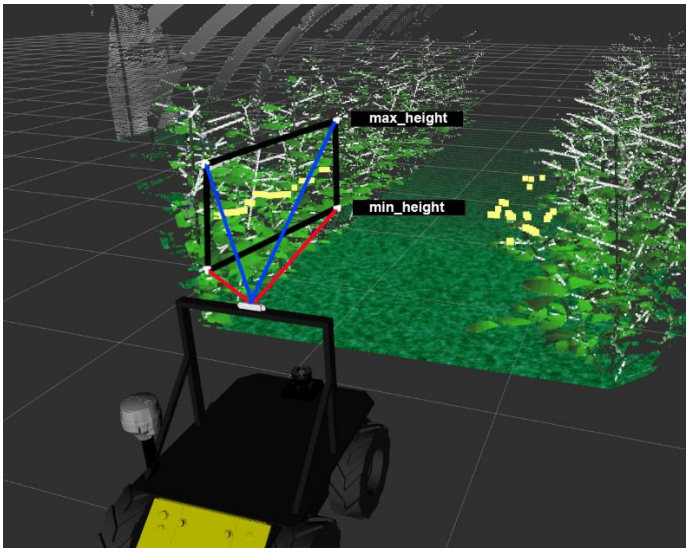
https://www.youtube.com/watch?v=sA5X0FnT-R4

### 4.5.5 Point cloud depth detection algoritme

Point cloud er et datasett med punkter som blir representert i rommet. Hvert punkt har en RGB-farge og en tilhørende XYZ-koordinat i rommet. Point cloud kan for eksempel brukes til å lage 3D-datamodeller av fysiske objekter [23]. Point cloud vil derfor også være aktuelt å bruke når roboten senere skal samle inn data fra bringebærproduksjonen.

Point cloud depth detection algoritmen behandler point cloud dataen fra realsense kameraet med ROS pakken pointcloud_to_laserscan. Denne pakken inneholder en node som bruker en point cloud som input, og gir output i form av datatypen LaserScan. Noden kjøres med parametere som setter blant annet hvilket område av point clouden som skal betraktes for å beregne LaserScan verdiene [24]. Eksempelvis når noden beregner lengden på laserstrålen ved 30° vil noden betrakte alle punktene som

er innenfor høyden minimum og maksimum høyde i vinkel 30° fra kamera. Det punktet som er nærmest i dette rommet vil bli lengden på laserstrålen i LaserScan topic ved 30°.




*Figur 18: Pointcloud to laserscan algoritme*

Dette vil lage en mer robust algoritme enn ved bruk av en LIDAR, da det er mye større sjanse for at det befinner seg en del av busken som kan detekteres innenfor dette området, enn at det befinner seg noe den kan detektere akkurat på det punktet som en enkelt laserstråle treffer.

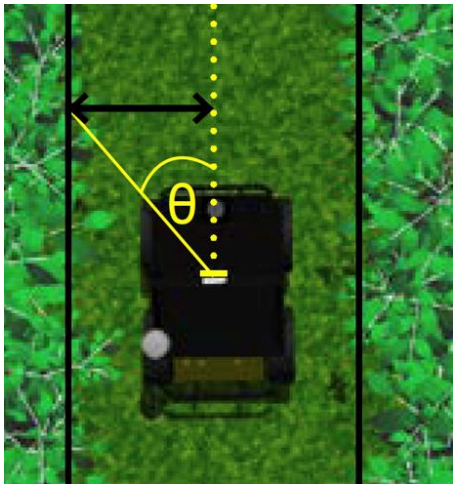
LaserScan verdiene fra høyre og venstre side blir sammenlignet for å beregne om roboten er sentrert mellom bringebærbuskene. Dette blir deretter brukt som error i samme kontrolleren som i de foregående algoritmene.

### 4.5.6 Forbedret kontroller for rekkefølging

Etter at RoboVest testfasilitet i Leikanger var satt opp ble gruppen tilsendt bilder av den endelige utformingen. Det viste seg at testfasiliteten dette første året ville bestå kun av to bærrekker, og avstanden mellom disse var mye større en gruppen var forespeilet tidligere i prosjektet. Det ble derfor behov for en kontroller som kunne styre roboten til å følge ved siden av en enkel bærrekke med en ønsket avstand.

*Figur 19: Illustrasjon av rekkefølging*

Dette ble forsøkt oppnådd ved å se på gjennomsnittet av laserstrålene på en av sidene og beregne error ved å sammenligne denne verdien med en ønsket verdi. Dette viste seg å være noe ustabilt da error noden måler snittet av laserstrålene over et visst område. Dette fører til at laserstrålene som måler langt fremme på rekken har mer å si for avviket enn en som måler lenger bak på rekken. En algoritme som vekter laserstrålene riktig ble derfor nødvendig å utvikle.




*Figur 20: Vekting av laserstråler*

Ved å vekte hver enkelt stråle med $\sin\theta$, der θ er vinkelen fra senterlinjen foran roboten til strålen, vil hver enkelt laserstråle representere lengden fra punktet der laserstrålen treffer bærrekken til senterlinjen foran roboten. Dette vil vekte laserstrålene riktig, og vil også gi en riktig «er-verdi» til error kalkuleringen. «Er» verdi vil da representere gjennomsnittlig lengde fra bærrekke til senterpunkt foran roboten.

Et problem med denne kontrolleren var at når roboten nærmet seg enden av en bærrekke startet den å svinge, og kjørte i rekken. Dette skyldes at laserstrålene som blir brukt til å beregne error ser et stykke foran roboten. Når laserstrålene ser forbi enden av bærrekken tror roboten at rekken svinger. Roboten prøver derfor å svinge etter. Dette ble løst med å ignorere stråler lenger enn 3 meter i error_calculator noden. I tillegg ble det laget en node som gir beskjed om at robot har nådd enden av en bærrekke slik at roboten kan stoppe. Denne noden blir beskrevet senere i rapporten.

Denne forbedrede kontrolleren gir roboten mer fleksibilitet da roboten også kan navigere langs enkle rekker.

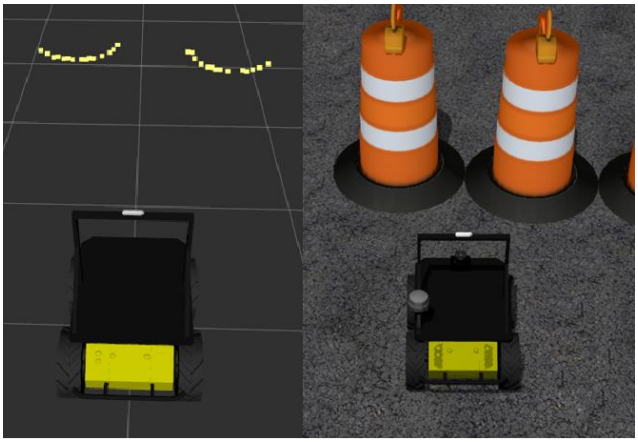
## 4.6 Navigering frem til bærrekke

En algoritme som skal navigere frem til starten av en bærrekke vil kreve noe mer sensordata en navigering langs bærrekker. Det er ingen faste geometriske punkter som roboten kan følge for å nå målet. Dette gjør at roboten trenger data fra sensorikk som kan fortelle den hvor i terrenget den befinner seg. Roboten vil også være avhengig av sensorikk som kan oppdage hindringer som den støter på.

### 4.6.1 Sensorikk

Informasjon om hvor roboten befinner seg kan hentes fra robotens Global Positioning – Real-time kinematic system (GPS-RTK). Et standard satellittbasert Global Positioning System (GPS) gir posisjonsinformasjon med meters presisjon. Dette vil ikke være nøyaktig nok for å kunne vite at roboten er plassert i riktig posisjon i forhold til en bærrekke. GPS-RTK gir derimot posisjonsdata med centimeterspresisjon. Dette oppnås ved å måle posisjonen sin ved hjelp av satellittdata, og kontrollere for feil basert på data fra basestasjoner. Basestasjonen kan enten være en egen enhet med kjent posisjon som plasseres ut, eller man kan kjøpe tjenesten fra internettleverandører [25]. Signalet blir da hentet fra de samme basestasjonene som mobiltelefoner får internett ifra. Dette er en løsning som vil gi god nok presisjon på robotens posisjonsmålinger.

Roboten vil også trenge sensordata som detekterer eventuelle hindringer den må unngå. En vanlig datatype å bruke til dette formålet er LaserScan. Eksempler på sensorer som kan levere LaserScan data er LIDAR og RGB-D kameraer. LaserScan data presenteres i et array med lengden de ulike laserstrålene har målt, sammen med informasjon om vinkel på de ulike laserstrålene og maksimal målelengde [26]. Ved å bruke disse dataene vil roboten kunne detektere hvilke retninger den ikke må kjøre for å kollidere med hindringer.



*Figur 21: Visualisering av LaserScan data*

### 4.6.2 Navigasjonsstrategi

Et viktig hjelpemiddel for en kart-basert navigasjonsstrategi er cost maps. Cost maps deler opp kartet i et kvadratisk rutenett med tilhørende verdier. Verdiene forteller om kvadratene i rutenettet som roboten planlegger å passere er okkupert av en hindring, eller om den er ledig. Roboten kan dermed planlegge beste rute mot målet.

Pakken move_base er en velkjent ROS pakke som bruker cost maps til å navigere seg mot et mål. Den bruker en global- og lokal-planlegger til å utføre dette. Move_base kan bruke den globale planleggeren til å beregne kortest rute igjennom rutenettverket fra punktet som roboten befinner seg i, frem til

målet, basert på koordinater i fra GPS-RTK systemet. Dersom roboten møter hindringer på veien som blir detektert av LaserScan sensoren, vil den bruke den lokale planleggeren til å beregne kortest rute rundt hindringen og tilbake på ruten til den globale planleggeren. Move_base kan bruke en rekke forskjellige algoritmer til å kalkulere kortest rute, men Dijikstras algoritme er kanskje den mest vanlige. Dijikstras algoritme er en velkjent metode til å kalkulere kortest vei fra A til B gjennom et nettverk [27].

En metode å generere kart over området som roboten beveger seg i er Simultaneous Localization and Mapping (SLAM). SLAM lager et kart, oppdaterer det og lokaliserer roboten i kartet samtidig. Når roboten i denne oppgaven skal flytte seg fra enden av en bærrekke til en ny bærrekke vil det sannsynligvis ikke være store hindringer mellom de to punktene. Et tomt kart, som oppdateres med de hindringene som roboten ser i øyeblikket vil derfor sannsynligvis være nok for denne oppgaven. Navigasjonsstrategien basert på cost maps og Dijikstras algoritme uten SLAM vil derfor være robust nok til å løse dette problemet.

### 4.6.3 GPS-waypoint navigation pakke

Husky roboten leveres med en ferdig utviklet navigasjonspakke som baserer seg blant annet på navigasjonsstrategiene som er beskrevet over.

Pakken bruker LaserScan data fra 3D LIDAR sensoren som er montert på roboten til å detektere hindringer, og data fra en rekke sensorer for å detektere posisjon. Posisjonen beregnes ved at data fra robotens GPS-RTK-, enkodere-, og Inertial Measurement Unit-sensorer (IMU) blir sendt gjennom et kalmanfilter. Kalmanfilteret er en statistisk metode som kan blant annet brukes til å estimere posisjon basert på data fra flere sensorer samtidig.

Basert på alle disse sensorene beregner GPS-waypoint navigation pakken ved hjelp av move-base noden lineære og angulære hastigheter som må sendes til hjul kontrolleren for at roboten skal bevege seg mot målet.

En bakdel med å bruke denne pakken er at alle GPS-punktene må manuelt registreres ved å kjøre roboten til starten av en bærrekke og lagre punktet. Dette er derimot en jobb som kun må utføres en gang, og etter dette vil roboten kunne kjøre samme rute om igjen uten hjelp fra en operatør.

## 4.7 Sikkerhet ved selvkjørende mobil Robot

### 4.7.1 Sikkerhetsanalyse

Utvikling av et selvkjørende robotsystem krever en del sikkerhetsmessige tiltak for at resultatet skal følge dagens forskrifter og være sikkert nok til å kjøre på egenhånd. Det er tatt utgangspunkt i ANSI/RIA R15 fra 2020, som er en Amerikansk standard for industrielle mobile roboter. Denne standarden lister ulike krav og metoder for å ivareta sikkerheten til personell som oppholder seg i nærheten av en industriell robot. Tidligere standarder for robotsikkerhet tok utgangspunkt i stasjonære roboter som var adskilt fra personell med fysiske sikkerhetsbarrierer [28]. Teknologien har utviklet seg og etter hvert som mobile roboter begynte å komme for fullt, har det blitt utviklet nye standarder som ivaretar sikkerheten uten å begrense utviklingen i for stor grad. Ettersom denne oppgaven omhandler utvikling av et selvkjørende robotsystem, vil sikkerhetstiltak være svært relevant.

Ifølge ANSI/RIA 5.1.7.1 kan god navigasjon være en sikkerhetsfunksjon. Dette kan være i form av å unnvike hindringer eller en kollisjon, men «trajectory re-planning» skal ikke introdusere nye farer. Siden denne roboten kjører mellom bringebærbusker som ikke har mye plass rundt seg, kan en god løsning være å bruke robotens LIDAR til å se etter mulige hindringer. LIDAR vil da kunne stoppe roboten til disse hindringene forsvinner.

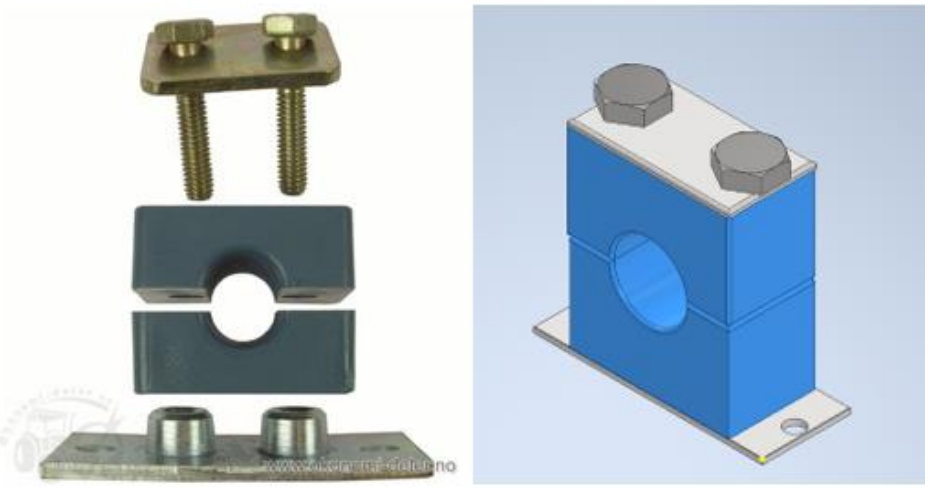
### 4.7.2 Første design av frontfanger R1

ANSI/RIA 5.1.7.2 sier at om en kollisjon ikke kan hindres så skal et system være på plass for å minske skadene ved kollisjon.

For at den selvkjørende robotbasen skulle tilfredsstille dette kravet ble det bestemt å designe et system som skulle fungere som siste sikkerhetsbarriere. Om roboten treffer en person eller gjenstand på grunn av at LIDAR ikke klarte å detektere objektet i tide så skal det være en barriere som stoppe roboten ved kollisjonen. En mulig løsning ble å design en frontfanger. I starten av prosjektet var ideen å designe en støtfanger med endebrytere. Støtfangeren skulle i tillegg til å stoppe roboten ved en kollisjon også dempe støtet fra selve kollisjonen.

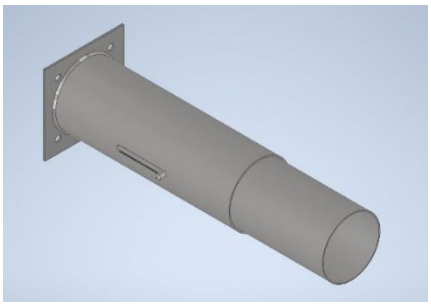
Designet ble gjort i Autodesk Inventor, et 3D-modellerings verktøy som ville gjøre det mulig å optimalisere en løsning før en eventuell realisering. I prosessen var det viktig å ha i bakhodet at utviklingen av denne frontfangeren skulle designes på en slik måte at det skulle være mulig for prosjektgruppen å fysisk bygge den og teste den på robotbasen. Siden det er en Husky robotbase som er utgangspunktet for hele prosjektet, måtte denne legges til grunn ved design av frontfangeren. Både foran og bak på roboten var det allerede festet en liten stålfanger originalt. Denne ble tatt i betraktning i designet og skulle brukes som anker for å feste frontfangeren. Siden roboten stod i Førde ble en 3D-fil lastet ned fra Clearpath Robotics slik at nødvendige mål kunne hentes ut for å skalere frontfangeren riktig. Et problem var at fil – formatet som ble tilsendt fra Clearpath Robotics var en IGS-fil som ikke var gunstig for import inn i Inventor. Det var godt nok til å få ut de nødvendige målene, men siden importen bare ble en skall-modell av roboten var det ikke mulig å bruke den til noe mer. Hadde det for eksempel vært en STEP fil kunne det ferdige designet av fangeren blitt montert på modellen for å se hvordan det faktisk passet og hvordan det ville se ut. I teorien kunne man ha fikset skall-modellen av roboten, men da måtte man modellert hver del av huskyen på nytt, noe som hadde tatt ekstremt mye tid i forhold til nytten.

Tanken bak første design var å utvikle en frontfanger som kunne dempe et sammenstøt samtidig som den stopper roboten. For å kunne feste fangeren i robotens eksisterende stålfanger ble det tatt utgangspunkt i industri–rørklammer. Målene på eksisterende stålfanger var 26mm i diameter tatt ut av 3D-modellen. Med rørklammer på 25mm diameter ville det være mulig å stramme den godt til. Det var flere varianter å velge i, og det ble valgt klammer med større sveiseplate med festehull slik at det skulle være mulig å feste de med skruer til resten av designet.



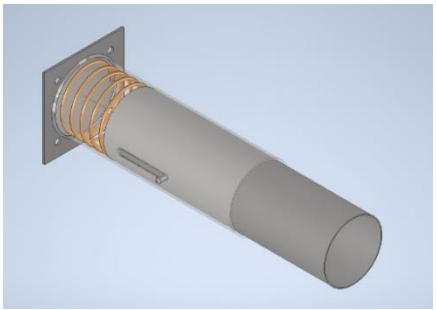
*Figur 22: Valgt rørklammer hos grossist og gjenskapt ved å 3D-modellere i Inventor.*

Videre ble det tegnet en feste-plate like lang som bredden av huskyen som et anker for resten av komponentene. Tanken var å designe en egen demper som kunne bygges av bachelorgruppen, dette skulle gjøres ved å sette en mindre sylinder inn i en større med en fjær som presset det lille sylindere utover. For at det innerste sylindret skulle kunne holde seg på plass måtte det ha en mottaker som begrenset hvor langt ut det kunne gå.



*Figur 23: To sylindere inni hverandre med mottaker i det innerste*

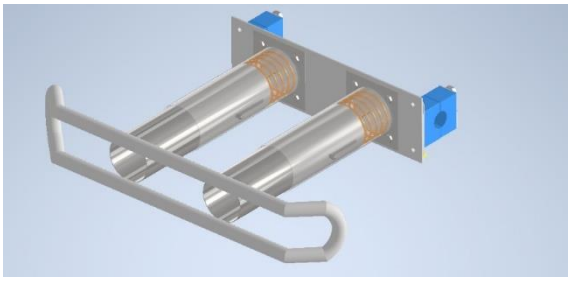
For å visualisere hvordan dette ville passe og se ut ble en fjær modellert inn, dette ville også gjøre det mulig å vite hvor stor fjær som var nødvendig.



*Figur 24: Ytre sylinder gjennomsiktig for å visualisere fjæren*

Mesteparten av designet ble fullført, eneste som manglet var mekanismen for å feste en endebryter som skulle bli utløst når frontfangeren ble trykket inn. Det ble synlig at frontfangeren ble litt mer komplisert enn først tenkt og unødvendig stor. Det ble derfor i samråd med veileder bestemt å prøve å lage et mindre og litt enklere design. Grunnlaget for begrunnelsen var at det kanskje ville være

vanskelig og ta litt mye tid å fysisk bygge fangeren etter dette designet. Størrelsen ble også så stor at den ville ta mye plass og i stor grad dominere huskyen.

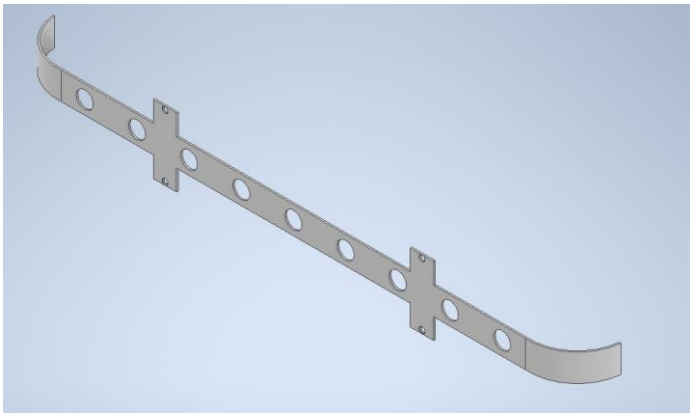


*Figur 25: Resultatet av frontfangeren før designet ble forkastet*

### 4.7.3 Andre design av frontfanger R2

For å gjøre frontfangeren mindre og lettere å bygge ble det valgt å gå vekk fra funksjonen som skulle dempe et eventuelt krasj. Som sikkerhetsmekanisme ville det være godt nokk å lage en frontfanger som sendte signal tilbake til ROS om at det var skjedd ett sammenstøt, og som da stoppet roboten.

Selv om et nytt design vil tilføre litt ekstra arbeid i designprosessen, hadde prosjektgruppen troen på at dette ville være med på å gi bedre et resultater seinere i prosjektet. Det nye designet tok utgangspunkt i basen fra den første frontfangeren. Rørklammerne ville fortsatt være en god løsning og ble derfor brukt videre i dette designet. Tanken var å lage en metallplate i ca. samme bredde som huskyen med hull til mange trykkbrytere, på denne måten ville huskyen stoppe uansett hvor i fronten den skulle treffe.

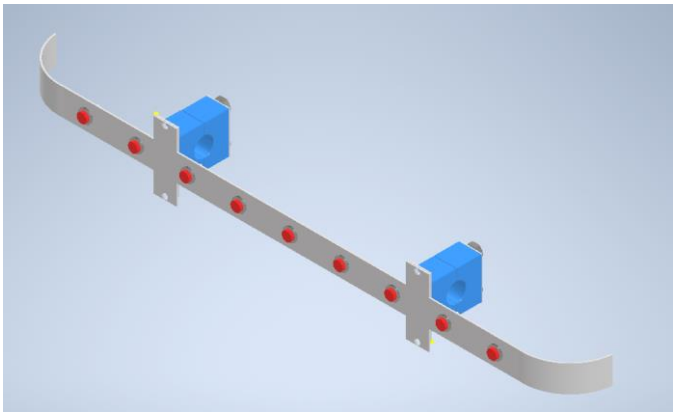


*Figur 26: Stålplate med hull til brytere og til å feste rørklammer*

Det var ikke lett å finne passende brytere, og tanken var i utgangspunktet å bruke noen mindre rektangulære microbrytere. Problemet med disse var at de hadde veldig liten trykk flate, i tillegg ville det være utfordrende å lage mange rektangulære hull i stålplaten. Etter å ha søkt litt rundt ble det funnet noen utstikkende pushbrytere som kunne egne seg godt til formålet, disse virket robust, hadde større trykk flate og var runde. Runde brytere ville gjøre det mye enklere å lage hull i metallplaten ved bygging.

For at rørklammerne skulle kunne festes ble det tegnet noen «ører» på hver side med hull i, på denne måten ville ikke resten av fangeren bli så bred. De bryterne som ble besluttet å bruke var det ikke mulig

å finne en 3D-modell av. For å gi en visualisering hvordan det ville se ut ble det derfor brukt noen andre litt mindre brytere som vi fant modell av.



*Figur 27: Endelig mekanisk design av frontfanger*

### 4.7.4 Kommunikasjon mellom frontfanger og ROS

For å kunne sende digitale signaler fra frontfangeren til husky computeren måtte det kobles opp en mellomenhet som mottok siglene for så å sende en melding til ROS om at en av bryterne var utløst. For å gjøre dette ble det foreslått å bruke en Arduino. Veileder mente at dette kunne være god ide da det var mulig å laste ned et eget bibliotek for å kommunisere med ROS via seriell kommunikasjon fra Arduinoen.

## 4.8 Konklusjon

Simuleringsverdenen er et viktig verktøy for testing, men ikke en del av den ferdige løsningen. Simuleringen ble brukt videre i testingen som er beskrevet senere i rapporten. Simuleringen sparte mye tid under testene som blir utført på den fysiske roboten da det var mindre usikkerhet knyttet til hvordan de ulike algoritmene ville fungere under tester. Simuleringen var også en nødvendighet da den fysiske roboten kun var tilgjengelig i korte perioder mot slutten av prosjektperioden.

Point cloud depth detection algoritmen med forbedret kontroller viste seg å være den beste algoritmen for sentrering og følging av bærrekker. Denne ble derfor implementert i det ferdige produktet. Algoritmene måtte fortsatt implementeres på en måte som gjorde at den blir lett å kalle opp når roboten er i drift. Hvordan dette har blitt utført vil bli tatt for seg i neste kapittel som omhandler den ferdige løsningen.

Frontfangeren ble utviklet for å tilfredsstille krav om sikkerhet om en ukontrollert hendelse skulle oppstå, det er vanskelig å se for seg hvordan en fysisk elektro-mekanisk sammenstilling vil passe på det eksisterende systemet. For å kunne lage et produkt som tok vare på sikkerheten, men som også var mulig å bygge måtte det lages en plan og et design å bygge etter. Autodesk Inventor har vært et godt hjelpemiddel. Det har gjort det mulig å utelukke det som ikke vil fungere og fokusere mer på det som kan fungere.

# 5 Realisering av valgt løsning

Den endelige programvaren er satt sammen av en rekke noder som gjør ulike beregninger. Data blir overført mellom nodene ved å abonnere og publisere på ulike topics. Noen av nodene inneholder også servicer som kan kalles opp. Da kjøres denne servicen til oppgavene er utført, og noden gir så tilbakemelding om dette.

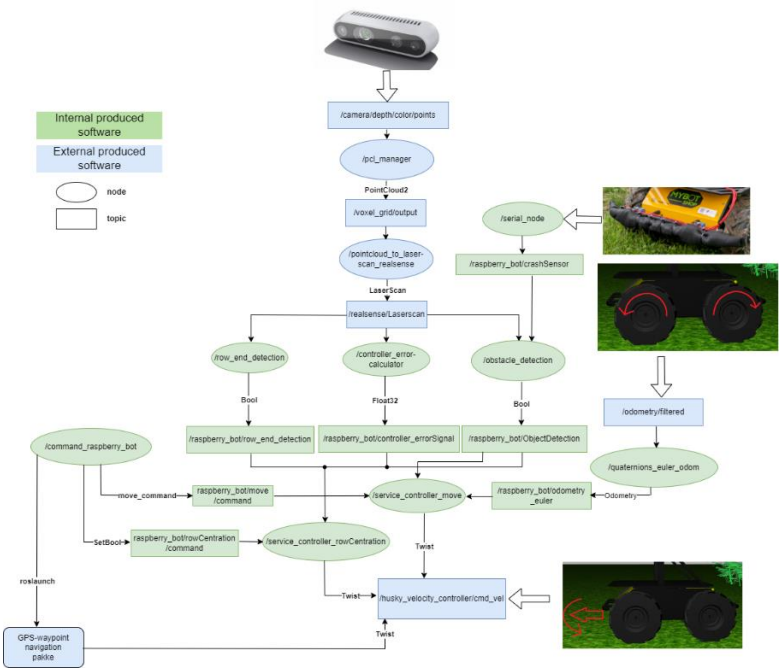
Det er ikke alle topicene i simuleringen som heter det samme som de tilsvarende topicene på den fysiske roboten. Eksempelvis for RGB-D kamera topics så er det de underliggende programmene som enten streamer fra en simulasjon eller fra et fysisk kamera som bestemmer navnene på disse topicene. Det blir derfor levert en programvarepakke for simulasjon, og en for den fysiske roboten som vedlegg til denne oppgaven. I tillegg ble det som tidligere nevnt nødvendig å lage en algoritme for rekkefølging i tillegg til rekkesentrering. I beskrivelsen av programvaren i dette kapittelet brukes programvaren for den fysiske roboten med rekkesentrering som utgangspunkt.

Gjennom designprosessen i forrige kapittel ble det vurdert ulike løsninger for å ivareta de mest sentrale krav om sikkerhet. Dette kapittelet skal ta for seg en realisering av et produkt som fremkommer av analysen og som til slutt ble det beste designet av en sikkerhetsmekanisme.

## 5.1 Programstruktur

Nodene i softwaren er delt opp slik at en enkelt node skal kun gjøre beregninger knyttet til en enkelt del av programmet. Dette er for at softwaren skal bli mer oversiktlig, og gjøre feilsøking enklere. Dette vil også gjøre softwaren mer robust da den ikke nødvendigvis trenger å krasje dersom en enkelt node skulle krasje.

Strukturen i softwaren er vist i figuren under. Figuren viser ikke alle nodene som kjører på huskyen, men den viser nodene som er relevante for denne oppgaven. Datatypen som sendes mellom nodene er merket på linjene som går inn på topicene.




*Figur 28: Software struktur*

Alle nodene i figuren over med unntak av command_rasberry_bot startes ved å kjøre en launch fil i terminalvinduet. Her kjøres enten en launch fil for rekkesentrering, eller en launch fil for rekkefølging. Deretter kan command_rasberry_bot programmet kjøres. Command_raspberry_bot noden er en service klient som er bygget opp som en liste med service kall. I denne noden kan det legges inn ulike service kall som kjøres i tur og orden for å gjennomføre navigasjonen som roboten skal utføre.

## 5.2 Noder

Rekkesentrerings- og rekkefølgings-algoritmene er tidligere utledet i kapittel 4. For å implementere algoritmene som en service kreves det en del noder som beregner data som servicene er avhengig av. Det er også implementert en service som kan kjøre og rotere roboten basert på odometry. Alle nodene i figuren over vil bli beskrevet i dette kapittelet.

### 5.2.1 Rowcentration service

Servicen kjører rekkesentreringsalgoritmen når den får en service kall i form av en boolsk true. Algoritmen kjører robot midt mellom to rekker til ende av rekke er detektert. Dersom noden får signal om at det er detektert hindringer fra obstacle_detection node stopper roboten til hindringen ikke lenger er detektert. Når roboten har nådd enden av rekken sender rowcentration service melding tilbake til service klient som kallet servicen om at service er fullført

**Controller error calculator**

Error calculator noden kjører algoritmen som kalkulerer avviket mellom venstre og høyre bærrekke basert på input fra LaserScan topic. Algoritmen sammenligner gjennomsnittet av de 5 laserstrålene som er lengst til venstre, med de 5 laserstrålene som er lengst til høyre. LaserScan topic data kommer ifra point clouden som blir generert fra RGB-D kameraet. Algoritmen betrakter ikke laserstråler som er lenger enn 3 meter for å unngå forstyrrelser. Antall laserstråler som skal betraktes kan justeres inne i nodens Python skript.

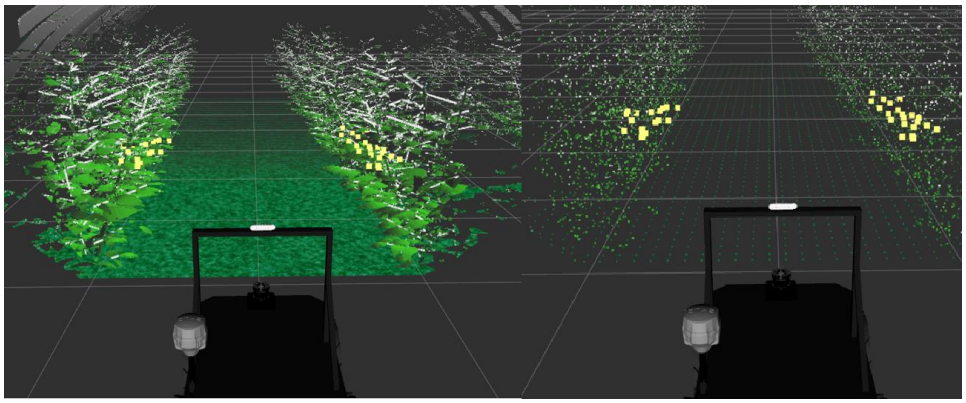
**Point cloud to LaserScan realsense**

Denne algoritmen er forklart i kapittel 4.5.5. Algoritmen konverterer pointcloud dataen fra realsense RGB-D kameraet om til LaserScan datatype. Algoritmen er stilt inn til å betrakte området fra ±34° med ca. en laserstråle for hver grad, i høyden ± 30cm. Algoritmen stilles inn til å betrakte laserstråler lenger enn 3 meter som infinity. Disse strålene settes til max + 1, altså 4 meter.

**PCL manager**

Denne noden skalerer ned antall punkter i point clouden fra RGB-D kameraet ved hjelp av et voxel filter. For at roboten skal vite hvor alle punktene i en point cloud befinner seg i forhold til sin egen koordinatramme må alle punktene kjøres gjennom en transformasjon fra kameraet sin koordinatramme til sin egen. Siden det er svært mange punkter i en point cloud blir denne prosessen veldig datakrevende. Uten å skalere ned point clouden klarte robotens datamaskin kun å prosessere to bilder i sekundet. Dette gjorde at kontrolleren ble ustabil.

Kontrolleren er derimot ikke avhengig av mange punkter, og ved å skalere ned point clouden til å kun betrakte punkter som er innenfor en avstand på 8 meter, og øke punktene i point clouden til å være 5 cm store klarte datamaskinen å prosessere 15 bilder i sekundet. Dette gir en mye mer stabil kontroller.



*Figur 29: Point cloud visualisert i Rviz med og uten voxel filter*

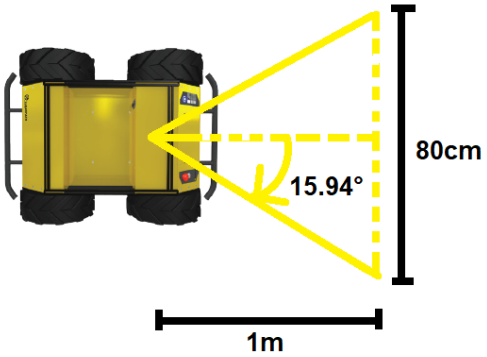
**Row end detection**

Denne algoritme detekterer når roboten har nådd enden av en bærrekke. Når roboten nærmer seg enden, vil flere og flere laserstråler gå mot infinity. Algoritmen er innstilt til å varsle om at enden av rekken er nådd dersom 10 stråler på sidene går mot infinity. Ved signal fra denne noden vil rowcentration service vite at den har fullført oppgaven sin.

**Obstacle detection**

Obstacle detection algoritmen betrakter laserstrålene som RGB-D kamera ser innenfor et område foran roboten. Dersom 5 eller flere av laserstrålene målere kortere enn 70cm skal roboten stanse. Dette signalet går ut til både rowcentering service og move service. Dersom hindringer blir fjernet vil roboten fortsette navigering. Antall stråler som skal være limit for stans, og stanslengde kan justeres i nodens Python skript.

Laserstrålene som blir betraktet i algoritmen er innenfor området ±15.94°. Dette betyr at området som blir overvåket er 80cm bredt 1m foran kameraet på roboten. Roboten er 67cm bred. Dette området ble valgt for at roboten kun skal detektere hindringer som den ikke vil klare å passere. Grunnen til at 5 eller flere stråler må være under grensen på 70cm for at roboten skal stanse er at algoritmen skal være robust mot støy.




*Figur 30: Illustrasjon av obstacle detection algoritme*

For en demonstrasjon av sensordataene som blir brukt til å detektere hindringer, følg denne linken:

https://www.youtube.com/watch?v=p2X5qW-Pwe4

**Serial node**

Støtfangeren med trykkbrytere som er montert på roboten er koblet mot en Arduino micro controller. Arduinoen er koblet til roboten via USB seriell kommunikasjon. Dersom en av bryterne på fangeren trykkes inn publiseres en boolsk true variabel i topicet /rasberry_bot/CrashSensor. Obstacle detection node abonnerer på denne topicen, og stopper roboten på lik linje med dersom en hindring blir detektert i RGB-D kameraet. Forskjellen er at dersom dette signalet blir utløst vil ikke roboten navigere videre dersom trykkbryteren ikke lenger er trykket inn. Programmet må stanses og launch filen som starter alle nodene må kjøres på nytt.

### 5.2.2 Move service

Move service er en service som kan kalles for å flytte roboten i en rett linje, eller rotere den rundt sin egen akse. Roboten bruker data fra /odometry/filtered topic for å beregne når disse bevegelsene er utført. Denne topicen publiserer robotens estimerte posisjon basert på data fra robotens enkodere og IMU vektet i et kalmanfilter.

Servicen kan kalles igjennom service kallet /raspberry_bot/move/command, og med den egenutviklede service kall datatypen raspberry_bot/move_command. Denne datatypen består av argumentene «Bevegelse» og «Input».

Bevegelsene som kan kalles er «linear» med input i meter, og «rotasjon» med input i radianer.

Den lineære bevegelsen utføres ved å kjøre roboten rett frem med linear hastighet 0.3m/s. Når kallet starter blir x,y koordinaten fra odometry topicet lager. Formel for distanse mellom to punkter i x,y planet blir brukt for å beregne tilbakelagt distanse for roboten. Når roboten har kjørt ønsket distanse stoppes roboten, og servicen sender melding til service klient om at oppgaven er utført. Roboten stopper opp dersom det blir detektert hindringer i obstacle_detection node, og fortsetter når hindringen er fjernet.

Ved rotasjons kall blir robotens orientering i forhold til Z-aksen i odometry topic lagret. Roboten roteres med en angulær hastighet på 0.3 rad/s i ønsket retning til ønsket rotasjon er oppnådd. Feil på grunn av nullstilling ved rotasjon forbi $\pi$ og $2\pi$ området blir håndtert.

**Quaternions to euler**

Robotens angulære orientering i odometry topic blir representert i quaternions. Dette er en representasjon som er effektiv for dataoperasjoner, men representasjonen er vanskelig å tyde for mennesker. Representasjonen blir derfor regnet om til euler vinkler i denne noden. Siden roboten kun kan rotere rundt Z-aksen (yaw), er det kun denne vinkelen som trengs for å beregne robotens orientering. Vinkelen blir publisert i /raspberry_bot/odometry_euler topic, og blir brukt av move service noden.

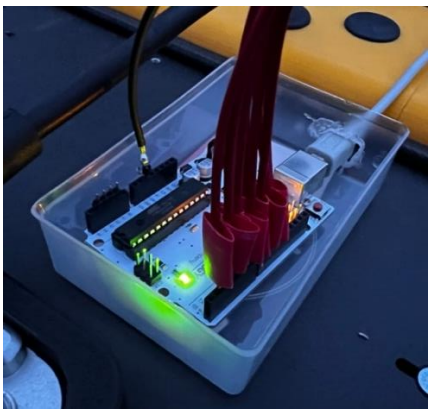
### 5.2.3 GPS-waypoint navigation

Funksjonen til GPS-waypoint navigation pakken er beskrevet i kapittel 4.6.3. Denne noden kan kjøres parallellt med de egenutviklede nodene. Noden kjøres ved at command_raspberry_bot programmet kaller en launch fil i GPS-waypoint navigation pakken. Denne launch filen inneholder en link til en fil som inneholder GPS-koordinaten som roboten skal navigere til. Når roboten har navigert til dette punktet stenger noden ned, og command_raspberry_bot kan kalle neste service kall i sin liste.

## 5.3 Frontfanger med trykksensor

Det endelige designet som ble valgt som ekstra sikkerhetsbarriere for robotsystemet var en frontfanger med trykkbrytere. Frontfangeren ble produsert hos arbeidsgiver til en av gruppemedlemmene. Selve byggeprosessen er nærmere beskrevet i Appendiks G.

Frontfangeren består av trykkbrytere som er plassert langs hele frontfangeren, og dermed overvåker hele sonen som robotplattformen kan kollidere i under kjøring. Ved en kollisjon vil en eller flere av bryterne bli presset inn, og skader kan forhindres.



*Figur 31: Arduino mikrokontroller montert på Husky*

Trykkbryterne er koblet mot en Arduino mikrokontroller som hele tiden overvåker tilstanden på trykkbryterne. Arduinoen er viderekoblet mot datamaskinen på robotplattformen via USB-seriell kommunikasjon. Her blir det publisert en boolsk variabel til et ROS topic. Denne variabelen forteller om en av trykkbryterne har blitt trykket inn. Dersom dette skjer vil obstacle_detection noden som abonnerer på dette topicet sende signal om dette videre, og robotplattformen vil stanse. Mer detaljer om hvordan dette er gjort kan leses i Appendiks G.2


*Figur 32: Husky robotbase utstyrt med frontfanger*

Designet passet bra på robotplattformen, og ser ut som et naturlig ekstrautstyr til huskyen.

# 6 Testing

Etter testuken i Førde viste det seg at GPS-waypoint navigation pakken ikke virket på den fysiske roboten. Gruppen hadde kontakt med utviklerne av programvaren i Tyskland, og Raquel ved robotlabben i Førde jobbet med å få pakken til å fungere på den fysiske roboten. Problemene ble dessverre ikke løst før testing av systemet i Leikanger, og det ble derfor nødvendig å gjøre noen endringer i navigasjonsstrategien.

GPS-waypoint navigation pakken ble fortsatt testet i simulering, men på den fysiske testen ved RoboVest testfasiliteten i Leikanger ble det kun brukt odometry. Dette vil si at roboten bruker data fra bevegelsessensorer for å estimere endringer i sin posisjon over tid. Siden testen skulle foregå inne i et enkelt telt med to bærrekker lot dette seg gjøre, men i et større anlegg ville man vert avhengig av GPS-RTK systemet.

## 6.1 Simulering av navigasjon i Gazebo

Testen ble delt opp i 6 ulike deler:

- Testing av rekkesentreringsalgoritme
- Testing av rekkefølgingsalgoritme
- Testing av move service
- Testing av GPS-waypoint navigation
- Testing av sikkerhet
- Testing av et sammensatt system

### 6.1.1 Testing av rekkesentreringsalgoritme

Roboten blir stilt opp sentrert på enden mellom to bærrekker. Rekkesentreringsalgoritme blir så kalt opp. Roboten skal navigere sentrert mellom rekkene, og stoppe når den kommer til enden.

Testen kan bevitnes her: https://www.youtube.com/watch?v=fNnHE3A_3QI

Testen viser at roboten har noen svingninger i starten, før den plasserer seg fint midt imellom rekkene. Når roboten har svinget seg inn til midtpunktet mellom rekkene holder den fint denne posisjonen. Roboten stopper av seg selv når det kommer til enden.

### 6.1.2 Testing av rekkefølgingsalgoritme

Roboten blir stilt opp på enden av en rekke med rekken på venstre side. Roboten blir stilt med ca. 30° vinkel, for å se at roboten svinger seg inn mot ønsket verdi. Roboten skal holde 1 meter avstand mellom senterlinja foran roboten og rekken. Når roboten når enden av rekken skal roboten manuelt snues, før rekkefølging på høgre siden skal kalles.

Testen kan bevitnes her: https://www.youtube.com/watch?v=5hkxwF6GJlo

Testen viser at roboten svinger seg fint inn mot ønsket verdi, med noen svingninger. Roboten stopper når den nærmer seg enden av rekken, og algoritmen fungerer både for høyre og venstre side.

### 6.1.3 Testing av move service

Move service testes ved å sende service kall som skal teste de ulike bevegelsene. Roboten skal kjøre et kvadrat, der sidene er 2 meter lange. Når roboten kommer tilbake til utgangspunktet skal den snu 180° i omvendt retning av de svingene den har kjørt i kvadratet for å teste at algoritmen fungerer i begge retninger.

Testen kan bevitnes her: https://www.youtube.com/watch?v=rjhvvP5ln5w

Testen viser at roboten klarer å utføre bevegelsene som blir kalt opp. Roboten kjører i rette linjer, men spinner noe til sidene når den skal rotere. Dette skyldes sannsynligvis at det er vanskelig å simulere friksjonen mellom roboten og underlaget når en differential drive robot skal snu på rundt sin egen akse. Tester på den fysiske roboten har vist at dette problemet ikke oppstår der.

### 6.1.4 Testing av GPS-waypoint navigation

GPS-waypoint navigation skal testes ved at to GPS-koordinater skal lagres. Roboten skal navigere fra origo i den simulerte verdenen, frem til første punkt, videre frem til andre punkt.

Testen kan bevitnes her: https://www.youtube.com/watch?v=k3PLHD1endk

Testen viser at navigeringen er altfor ustabil til å kunne brukes, og roboten når heller ikke sine mål. En annen metode må derfor benyttes for å flytte roboten fra rekke til rekke i simuleringen.

### 6.1.5 Testing av sikkerhet

Sikkerhetssystemet skal testet ved at en person blir satt ut foran roboten i simuleringen. Move service blir deretter brukt til å kalle at roboten skal kjøre i en rett linje. Roboten skal stoppe når den nærmer seg personen. Når personen fjernes, skal roboten fortsette å kjøre i en rett linje.

Testen kan bevitnes her: https://www.youtube.com/watch?v=8wd7rt0t7-s

Roboten stopper med god margin fra personen. Når personen blir flyttet sidelengs står også roboten i ro. Når personen kommer utenfor frontpartiet på roboten fortsetter roboten sin navigering uten å krasje i personen.

### 6.1.6 Testing av sammensatt system

Siden GPS-waypoint navigation pakken er for ustabil blir det vanskelig å navigere roboten fra telt til telt. Testen vil derfor bestå av å prøve å få roboten til å navigere gjennom de to rekkene inne i et enkelt telt. Dette vil bli ganske nært den navigeringsoppgaven som må utføres i Leikanger også.

Testen kan bevitnes her: https://www.youtube.com/watch?v=ksZ8WSXnB3A

Testen viser at roboten klarer å navigere gjennom et telt, men rotasjonen som roboten utfører er noe ustabile.

### 6.1.7 Oppsummering

Følgende krav har blitt vist oppfylt i denne testen:

**Sikkerhet:**

- Robot klarer å oppdage hindringer og stoppe med en klaring på minst 30 cm fra objektet.

**Navigering:**

- Roboten klarer å navigere fra start av en bærrekke til andre enden av en bærrekke ved hjelp av 3D-kamera. Roboten kjører sentrert mellom rekkene, men avviket er vanskelig å anslå i en simulering. Systemet virker stabilt
- Roboten utfører nevnt navigeringsoppgave med en fart på 0.3 m/s, altså 1.08 km/t

**Simulering:**

- Simulering av verden inneholder minst 3 bærrekker, men ingen parkeringsstasjon. Det er heller ikke noen poeng med en parkeringsstasjon som utgangspunkt når GPS-waypoint navigation pakken ikke fungerer.
- Simuleringsverden inneholder mulighet for å sette inn hindringer for test av navigering- og sikkerhetssystemer.

Følgende krav har ikke blitt oppfylt i testen:

- Roboten klarer ikke å navigere fra parkeringsstasjon eller ende av bærrekke til start av neste bærrekke. Dette skyldes at GPS-waypoint navigation pakken ikke fungerer.

## 6.2 Test av hardware og software på Husky robot

Sted: RoboVest testfasilitet, Leikanger
Til stede: Aril Torheim, Isak Vamråk Førde, Chris Louis Johnsen

Siden GPS-waypoint navigation pakken ikke fungerte måtte det brukes odometry for å prøve å navigere roboten fra rekke til rekke. Målet var å lage et program der roboten fikk navigert langs alle sidene av rekkene i teltet. Testen ble delt opp i 2 ulike deler:

- Test av navigering på RoboVest testfasilitet
- Test av sikkerhetssystemer

### 6.2.1 Test av navigering på RoboVest testfasilitet

Testen ble utført ved å lage et program der roboten skulle følge en bærrekke til enden, kjøre rundt enden, og følge andre siden av rekke opp igjen. Deretter skulle roboten traversere over til neste bærrekke og utføre samme her runde her. Når roboten var ferdig med dette skulle den kjøre til punktet som den startet på og starte runden på nytt. Testen benytter seg av move service og rowfollowing service.

Testen kan bevitnes her: https://www.youtube.com/watch?v=Y344rYtpLJs

Roboten klarte å gjennomføre navigasjonsoppgaven som den ble satt til. Når roboten starter å følge en ny bærrekke svinger den raskt inn til ønsket verdi som var 1 meter avstand mellom senterlinjen foran roboten til bærrekken. Når roboten nådde ønsket verdi holdt den seg stabilt på denne avstanden til den nådde enden av bærrekken. Roboten klarer også å navigere til neste bærrekke ved bruk av odometry data.

### 6.2.2 Test av sikkerhetssystemer

Sikkerhetssystemet består av objekt deteksjon ved bruk av kamera og støtfangeren med trykkbrytere. Systemet ble testet ved at en person stilte seg i en rett linje foran roboten. Et service kall for en lineær bevegelse ble deretter sendt til roboten. Roboten skal da stoppe før den kolliderer med personen. Deretter blir kameraet snudd 90° slik at dette systemet i praksis ikke fungerer lenger. Roboten vil da fortsette å kjøre mot personen. Roboten skal stoppe når en av trykkbryterne på støtfangeren blir trykket inn.

Testen kan bevitnes her: https://www.youtube.com/watch?v=QFJbMiH3Tlk

Testen viste at roboten stoppet med ca. en halv meter klaring til personen, noe som stemmer overens med forhåndsdefinert stopp verdi. Når kameraet ble snudd 90° kjørte roboten i personen, men stoppet fort. Personen vil merke denne kollisjonen, men støtet påfører personen så liten kraft at det ikke er fare for personskade.

Testen ble også utført med kollisjon med objekt i form av en plast kasse. Denne testen gav samme resultat.

### 6.2.3 Oppsummering

Følgende krav har blitt oppfylt i denne testen:

**Sikkerhet:**

- Roboten klarer å stoppe for hindringer og stoppe med en klaring på 30 cm fra objektet.
- Robotens støtfanger fungerer som en ekstra sikkerhetsbarriere som stopper roboten dersom den skulle kjøre i en hindring. Sikkerhetsbarrierens design er slik at roboten ikke påfører skader dersom et menneske skulle stå i veien.

**Navigering:**

- Roboten klarer å navigere fra starten av en bærrekke til andre enden av en bærrekke ved å bruke kameradata. Roboten klarer å gjøre dette med et avvik på under 10 cm.
- Roboten klarer å navigere fra et startpunkt, og kjøre langs alle bærrekkene i et enkelt dyrkningstelt, og klarer å gjøre dette med en hastighet på 1.75 km/t i denne testen.

Roboten klarer ikke å navigere mellom ulike dyrkningstelt da GPS-waypoint navigation pakken ikke fungerer.

# 7 Diskusjon

Prosjektet har for det meste bestått av programmering, simulering og testing av ulike løsninger. Det har satt krav til forholdsvis gode datakunnskaper når det kommer til ROS, Linux, Python og nettverkskommunikasjon. Den største utfordringen i prosjektet var at mye av denne kunnskapen var kunnskap som gruppen ikke hadde på forhånd og som måtte tilegnes i starten av prosjektperioden. Det var en del frustrasjon i gruppen i denne tiden da det føltes ut som fremdriften var dårlig, men etter hvert som kunnskapen kom løsnet dette og flere delmål ble raskt nådd.

At oppgaven besto av både utvikling av software og hardware gjorde oppgaven variert, men kanskje også mer krevende da det blir flere aspekter som må koordineres og til slutt sys sammen. Etter at utviklingen var fullført og de ulike delene av prosjektet ble satt sammen i et samlet system var det veldig gøy å se at systemene som var blitt simulert i månedene før faktisk virket på den fysiske roboten. En god simuleringsmodell sparte uten tvil mange timer i felt da mesteparten av utviklingen kunne gjøres på forhånd. Som nevnt tidligere tar de aller fleste ting når det kommer til programmering og testing lenger tid i felt, derfor var det avgjørende å få gjort så mye som mulig på forhånd slik at arbeidet som måtte gjøres i felt var ting som ikke kunne simuleres på forhånd.

Fremdriftsplanen som var satt opp på forhånd var nok noe optimistisk, spesielt med tanke på mangelen av kunnskap rundt ROS og Linux. Det er selvsagt vanskelig å estimere på forhånd hvor lang tid det vil ta å utføre en oppgave som må tilegnes kunnskap om først. Sett i ettertid burde det nok vert lagt inn en egen arbeidspakke med kursing i ROS, Linux og Python tidlig i prosjektperioden. Dette ville nok spart gruppen for en del frustrasjon, og kanskje lagt et bedre grunnlag for å kunne følge fremdriftsplanen videre. Heldigvis var det satt av noen åpne uker på slutten av prosjektperioden til eksamen og utforutsatte hendinger. Dette gjorde det mulig å komme i havn med prosjektet, selv om det ble hektisk på slutten. Punktene i fremdriftsplanen har fortsatt blitt fulgt, selv om det ikke har vert på de riktige datoene. Fremdriftsplanen og flytskjemaet for fremgangsmetode som ble satt opp i forstudien har

fungert som et godt verktøy til å fordele oppgaver mellom gruppemedlemmene, og generelt for å holde oversikt over kommende mål i prosjektet.

Siden GPS-waypoint navigation systemet ikke ble operativt før prosjektperioden var ferdig er systemet enda ikke i stand til å navigere over et større område med flere dyrkningstelt. Dette er noe som delvis har vert ute av vår kontroll da dette er et eksternt utviklet system. Det hadde selvfølgelig vert mulig å prøve å utvikle sitt eget system, men dette hadde nok vert veldig tidkrevende. Et av hovedpoengene med ROS er gjenbruk av tidligere utviklet kode. Det kan derfor ikke sees på som en lettvint løsning å integrere inn et ferdigutviklet system. Utfordringen med ROS er ikke nødvendigvis å skrive kode, men å få all koden til å fungere sammen i et samlet system. Gruppen har lagt fram navigasjonsstrategier og sensorikk som kreves for å løse navigasjonsproblemet med å navigere roboten fra rekker til rekke. Det er kun å få programvaren til å fungere som mangler. Det er også vist i brukerdokumentasjonen hvordan dette systemet kan brukes i samspill med den øvrige programvaren når GPS-waypoint navigation systemet en dag fungerer.

Programvaren som blir levert med oppgaven er forholdsvis omfattende, men ved hjelp av programstruktur oversikten i kapittel 5.1 bør det være ganske overkommelig for eventuelle fremtidige studenter å få en oversikt over hvordan data flyter gjennom systemet. Det har også vert fokus på å skrive beskrivende kommentarer underveis i alle Python-skriptene slik at det skal være lett for studenter som kommer etter å skjønne hva hvert enkelt skript utfører. Det vil allikevel kreve en god del kunnskaper om ROS for å skjønne i detalj hvordan skriptene fungerer. På grunn av at det måtte utvikles rekkefølgingsalgoritmer i tillegg til den originale rekkesentreringsalgoritmen mot slutten av prosjektet inneholder programvaren en del ulike utgaver av ganske like programmer. For eksempel finnes det 3 utgaver av programvaren som detekterer enden av en rekke. En for rekkefølging venstre side, en for høyre side, og en for rekkesentrering. Riktig program blir kjørt når man kjører launch filen ved oppstart av programvaren, men systemet ville vært mer fleksibelt dersom det kun var et program som tok seg av alle de tre senarioene, og at venstre, høyre eller sentreringsalgoritme ble valgt med et service kall. Det ble ikke tid i prosjektperioden til å forbedre og teste dette.

Som en ekstra modul til prosjektet ble det lagt vekt på å utvikle et system for å ivareta krav om sikkerhet. Ved utvikling av ny teknologi er det viktig at sikkerhetsaspektet følger med, dette er viktig for at et fremtidig operativt system skal få tillatelse til å bli tatt i bruk i områder der mennesker kan oppholde seg. Ved å designe, bygge og teste et system for å ivareta personsikkerhet kan man si at utgangspunktet for en selvkjørende robotbasen er sikrere enn hva den ville vært uten.

# 8 Konklusjon

Systemet som har blitt utviklet i dette prosjektet klarer å kjøre selvstendig rundt i et bærproduksjonsmiljø med visse begrensninger. Systemet klarer å navigere langs bærrekker innenfor kravene som ble satt på forhånd. Navigasjon fra bærrekke til bærrekke var planlagt utført ved bruk av GPS-waypoint navigation systemet som følger med husky robotplattformen. Dette systemet fungerer ikke ved enden av prosjektperioden og setter derfor noen begrensninger til denne delen av navigasjonen. Roboten klarer å navigere innenfor et enkelt telt med bærrekker, men den vil få problemer dersom den skal navigere seg videre til et nytt telt. Navigasjonsproblemet har blitt løst ved å bruke odometry til å navigere fra bærrekke til bærrekke innad i et telt.

Sikkerhetssystemene fungerer som tiltenkt. Hindringer blir detektert ved bruk av realsense D435i kamera, og roboten klarer å stoppe for hindringer. Dersom denne sikkerhetsbarrieren skulle feile stopper støtfangeren med trykkbrytere roboten når den kolliderer med en hindring uten at det påfører mye kraft på hindringen.

Systemet i sin helhet er robust, og klarer å produsere samme resultat ved gjentatte tester. Eneste svakheten som har blitt oppdaget er dersom personer står langs en bærrekke, og beveger seg ut fra bærrekken når roboten nærmer seg. Roboten vil da prøve å svinge rundt personen, og dersom personen beveger seg vekk fra bærrekken samtidig vil roboten svinge enda mer. Roboten vil da miste bærrekken av syne og komme ut av kurs. Den vil da sannsynligvis ikke finne tilbake til bærrekken.

Simuleringsverdenen som leveres med programvaren har også vist seg å være et godt verktøy, og vil forhåpentligvis bli nyttig for eventuelle fremtidige studenter som skal arbeide med roboten.

For videre arbeid anbefales det å få GPS-waypoint navigation systemet til å fungere. Dette vil gjøre systemet mer robust, og vil utvide området som roboten kan navigere selvstendig på betraktelig. Det anbefales også å implementere valg av venstre, høyre og sentreringsalgoritme i en service kall slik at man kan utføre forskjellige typer navigering langs rekker uten å måtte bytte hvilken programvare som kjører.

Et naturlig steg videre vil også være å prøve å samle inn data fra bærproduksjonsområde som roboten navigerer i, og gjøre analyser på denne dataen. Bruk av ulike SLAM algoritmer med kameraet som allerede er montert på roboten kan være et alternativ, men et eller flere kamera som er vendt mot bærrekken vil sannsynligvis kunne gi bedre data om bærene.

I denne oppgaven har det kun vert brukt enkle P-regulatorer til å kontrollere roboten. Dette har fungert bra, men det kunne vert interessant å prøve ut andre typer regulatorer også. I move service har det kun blitt brukt IF setninger for å styre pådrag under lineære og angulære bevegelser med roboten. Dette har fungert godt, men nøyaktigheten på spesielt de angulære bevegelsene kunne blitt forbedret ved å for eksempel bruke en PI-regulator.

# Referanser

[1] T. I. Hanse, «Høgskulen på Vestlandet i Store Norske Leksikon,» [Internett]. Available: https://snl.no/H%C3%B8gskulen_p%C3%A5_Vestlandet. [Funnet 2 Mai 2022].

[2] I. A. Husabø, «Robotar på veg inn i frukt- og bærnæringa, på Vestlandsforskning,» [Internett]. Available: https://vestforsk.no/nn/2022/robotar-pa-veg-inn-i-frukt-og-baernaeringa. [Funnet 2 Mai 2022].

[3] Fieldwork Robotics, «Fieldwork Robotics,» [Internett]. Available: https://fieldworkrobotics.com/. [Funnet 2 Mai 2022].

[4] Høgskulen på Vestlandet, *BO22EB-29 HVL Bær Datainsamling Robot, Oppgaver Bergen,* Høgskulen på Vestlandet, 2021.

[5] P. Corke, Robotics, Vision and Control, Brisbane: Springer, 2017, p. 125.

[6] TerrisGPS, «EXPLAINING RTK IN GNSS / GPS, fra TerrisGPS,» [Internett]. Available: http://www.terrisgps.com/what-is-gps-gnss-rtk/.

[7] Clearpath Robotics Inc., «Husky Unmanned Ground Vehical, clearpathrobotics.com,» [Internett]. Available: https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/. [Funnet 19 Mai 2022].

[8] ROS.org, «What is ROS? fra ROS.org wiki,» [Internett]. Available: http://wiki.ros.org/ROS/Introduction. [Funnet 3 Mai 2022].

[9] ROS.org, «Packages, på ROS wiki,» [Internett]. Available: http://wiki.ros.org/Packages. [Funnet 3 Mai 2022].

[10] ROS.org, «Nodes, på ROS wiki,» [Internett]. Available: http://wiki.ros.org/Nodes. [Funnet 3 Mai 2022].

[11] ROS.org, «Topics, på ROS Wiki,» [Internett]. Available: http://wiki.ros.org/Topics. [Funnet 3 Mai 2022].

[12] ROS.org, «Services, på ROS Wiki,» [Internett]. Available: http://wiki.ros.org/Services. [Funnet 3 Mai 2022].

[13] ROS.org, «actionlib, på ROS Wiki,» [Internett]. Available: http://wiki.ros.org/actionlib. [Funnet 3 Mai 2022].

[14] A. Linz, «05 - Highlights of ROS in Agricultural Domain, Andreas Linz, HS Osnabrück, fra Youtube,» 21 Januar 2020. [Internett]. Available: https://www.youtube.com/watch?v=S-TfjMSXbM4&t=504s. [Funnet 3 Mai 2022].

[1 5] F. Longsheng, G. Fangfang, W. Jingzhu, L. Rui, M. Karkee og Z. Qin, «Application of consumer RGB-D cameras for fruit detection and localization in field: A critical review,» [Internett]. Available: https://www.sciencedirect.com/science/article/pii/S0168169920319530.

[1 6] Intel, «Beginners guide to Depth,» [Internett]. Available: https://www.intelrealsense.com/beginners-guide-to-depth/.

[1 7] Intel, «Intel RealSense Depth Camera 435i,» [Internett]. Available: https://www.intelrealsense.com/depth-camera-d435i/.

[1 8] OpenCV, «OpenCV About,» [Internett]. Available: https://opencv.org/about/.

[1 9] The Construct, «[ROS Tutorials] ROS Perception Unit2# Follow Line with OpenCV (Python),» 21 September 2017. [Internett]. Available: https://www.youtube.com/watch?v=ukGa74saFfM. [Funnet 29 Mars 2022].

[2 0] Programming Design Systems, «Color models and color spaces, Programming Design Systems,» [Internett]. Available: https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html.

[2 1] LearnOpenCV.com, «Applying Gaussian Blurring to an Image in OpenCV,» [Internett]. Available: https://learnopencv.com/image-filtering-using-convolution-in-opencv/#gauss-blur-opencv.

[2 2] LearnOpenCV.com, «Canny Edge Detection,» [Internett]. Available: https://learnopencv.com/edge-detection-using-opencv/.

[2 3] Wikipedia, «Point Cloud, Wikipedia,» [Internett]. Available: https://en.wikipedia.org/wiki/Point_cloud. [Funnet 29 April 2022].

[2 4] ROS, «pointcloud_to_laserscan, ROS wiki,» [Internett]. Available: http://wiki.ros.org/pointcloud_to_laserscan.

[2 5] O. Utiugova, «Emlid, Introduction to RTK GNSS,» 22 July 2021. [Internett]. Available: https://emlid.com/introduction-to-rtk-gps/. [Funnet 26 Mai 2022].

[2 6] ros.org, «ros.org, sensor_msgs/LaserScan message,» [Internett]. Available: http://docs.ros.org/en/noetic/api/sensor_msgs/html/msg/LaserScan.html. [Funnet 26 Mai 2022].

[2 7] ROS.org, «ROS Wiki, move_base,» [Internett]. Available: http://wiki.ros.org/move_base. [Funnet 26 Mai 2022].

[2 8] L. Moretz, «automate.org,» 15 02 2021. [Internett]. Available: https://www.automate.org/industry-insights/mobile-robot-standard-r15-08-1-2020-what-you-need-to-know. [Funnet 03 05 2022].

[29] IBM, [Internett]. Available: https://www.ibm.com/topics/what-is-a-digital-twin.

[30] P. &. P. J. &. D. W. &. G. A. Staczek, «ReasearchGate,» 26 August 2021. [Internett]. Available: https://www.researchgate.net/publication/356528334_A_Digital_Twin_Approach_for_the_Improvement_of_an_Autonomous_Mobile_Robots_AMR's_Operating_Environment-A_Case_Study.

[31] Robotic Industries Association, For Industrial Mobile Robots - Safety Requirements, American National Standards Institute, Inc, 2020.

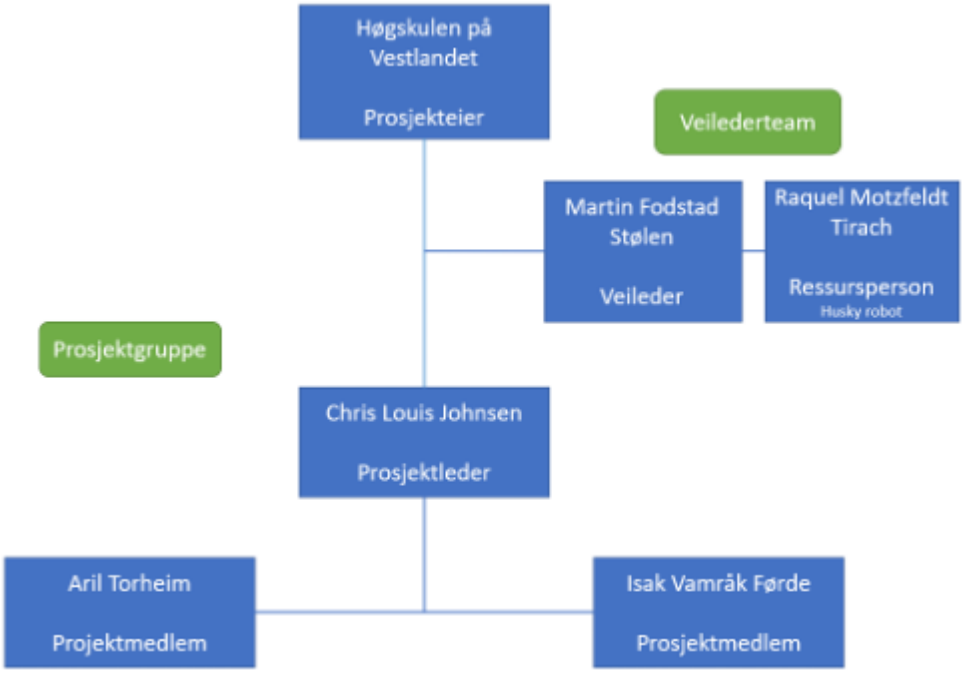
# Appendiks A Forkortelser og ordforklaringer

| | |
|---|---|
| 2D | 2-Dimensional |
| 3D | 3-Dimensional |
| FPS | Frames Per Second |
| GND | Ground |
| GPS | Global Positioning System |
| GPS-RTK | Global Positioning – Real-time kinematic |
| HSV | Hue Saturation Value |
| IMU | Inertial Measurement Unit |
| LIDAR | Light Detection and Ranging |
| ROS | Robot Operating System |
| RGB-D | Red, Green, Blue – Depth |
| RViz | ROS Visualization |
| SLAM | Simultaneous Localization and Mapping |
| SSH | Secure Shell |
| UGV | Unmanned Ground Vehicle |
| USB | Universal Serial Bus |

# Appendiks B Prosjektledelse og styring

## B.1 Prosjektorganisasjon

Gruppen har vert organisert som en linje- og staborganisasjon med Chris Louis Johnsen som prosjektleder. Prosjekteier har vert Høgskolen på Vestlandet. Martin Fodstad Stølen har vært veileder og faglig ressurs gjennom prosjektperioden.




*Figur 33: Organisasjonskart*

Chris Louis har vert ansvarlig for delegering av arbeid, og følge opp at gruppemedlemmene arbeider mot å nå milepæler som er satt i tidsplanen. Gruppemedlemmene har selv vært ansvarlige for at tidsfristene som er satt for sine arbeidspakker blir nådd.
Oppgavene ble tildelt i starten av prosjektet når fremdriftsplanen ble utarbeidet. Dette kan sees i fremdriftsplanen ved appendiks B.3.

## B.2 Prosjektledelse

Prosjektplattform som ble brukt var Microsoft Office. Alt av materiale som ble produsert til prosjektet ble lastet opp til OneDrive. Det ble laget en mappestruktur i OneDrive for de ulike delene av prosjektet. Denne mappestrukturen har ekspandert gjennom hele prosjektperioden ettersom nytt materiale har blitt laget.

Kommunikasjon mellom gruppen og ressurspersoner har hovedsakelig foregått gjennom Microsoft Teams og Outlook, samt at møter med veileder har foregått på Zoom. Det er også blitt tatt i bruk Messenger og Discord for intern kommunikasjon mellom gruppemedlemmene.

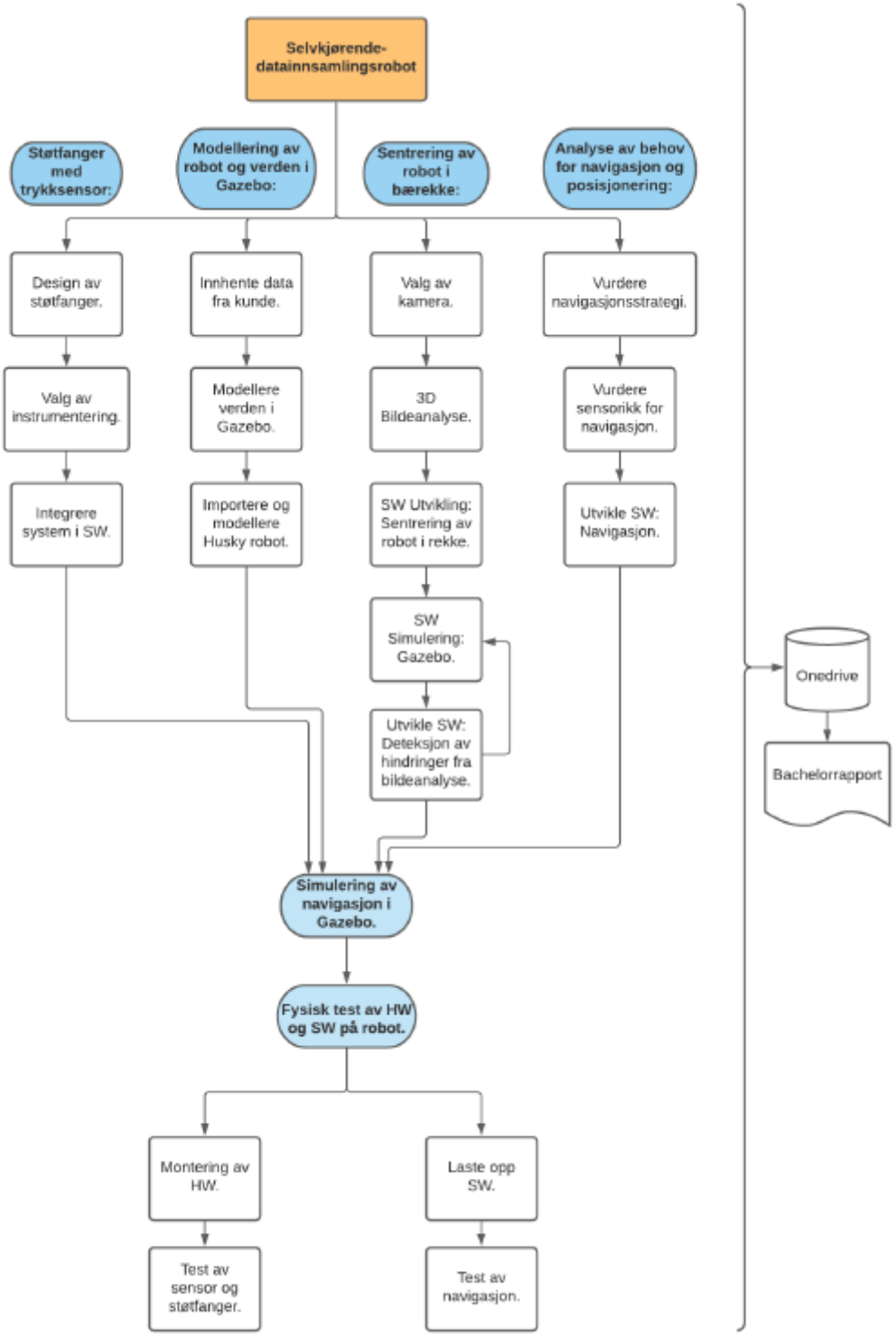
For å holde kontroll på fremgang har gruppemedlemmene timeført og loggført prosjektarbeid i et felles Excel ark.

Prosjektformen har fungert bra. Mappestrukturen har vert oversiktlig, kommunikasjonen internt og eksternt har fungert bra og gruppemedlemmene har hele tiden visst hva de skal jobbe med basert på fremdriftsplanen.

## B.3 Fremdriftsplan

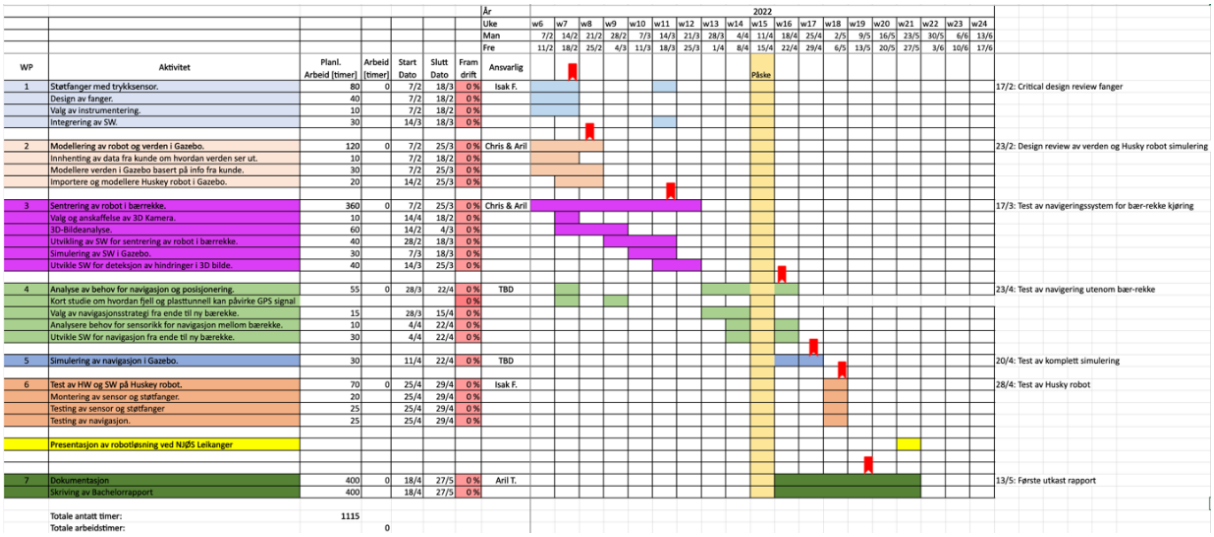
**Flytskjema**

Det ble utviklet et flytskjema i forstudien med naturlige delmål og parallelle linjer frem mot det endelige målet i prosjektet. Basert på dette flytskjemaet ble fremdriftsplanen satt opp.




*Figur 34: Milepæl flytskjema*

**Opprinnelig fremdriftsplan:**



| WP | Aktivitet | Planl. Arbeid [timer] | Arbeid [timer] | Start Dato | Slutt Dato | Fram drift | Ansvarlig | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | År | 2022 |
| | | | | | | | Uke | w6 w7 w8 w9 w10 w11 w12 w13 w14 w15 w16 w17 w18 w19 w20 w21 w22 w23 w24 |
| | | | | | | | Man | 7/2 14/2 21/2 28/2 7/3 14/3 21/3 28/3 4/4 11/4 18/4 25/4 2/5 9/5 16/5 23/5 30/5 6/6 13/6 |
| | | | | | | | Fre | 11/2 18/2 25/2 4/3 11/3 18/3 25/3 1/4 8/4 15/4 22/4 29/4 6/5 13/5 20/5 27/5 3/6 10/6 17/6 |
| 1 | Støtfanger med trykksensor. | 80 | 0 | 7/2 | 18/3 | 0 % | Isak F. | 17/2: Critical design review fanger |
| | Design av fanger. | 40 | | 7/2 | 18/2 | 0 % | | |
| | Valg av instrumentering. | 10 | | 7/2 | 18/2 | 0 % | | |
| | Integrering av SW. | 30 | | 14/3 | 18/3 | 0 % | | |
| 2 | Modellering av robot og verden i Gazebo. | 120 | 0 | 7/2 | 25/3 | 0 % | Chris & Aril | 23/2: Design review av verden og Husky robot simulering |
| | Innhenting av data fra kunde om hvordan verden ser ut. | 10 | | 7/2 | 18/2 | 0 % | | |
| | Modellere verden i Gazebo basert på info fra kunde. | 30 | | 7/2 | 25/3 | 0 % | | |
| | Importere og modellere Huskey robot i Gazebo. | 20 | | 14/2 | 25/3 | 0 % | | |
| 3 | Sentrering av robot i bærrekke. | 360 | 0 | 7/2 | 25/3 | 0 % | Chris & Aril | 17/3: Test av navigeringssystem for bær-rekke kjøring |
| | Valg og anskaffelse av 3D Kamera. | 10 | | 14/4 | 18/2 | 0 % | | |
| | 3D-Bildeanalyse. | 60 | | 14/2 | 4/3 | 0 % | | |
| | Utvikling av SW for sentrering av robot i bærrekke. | 40 | | 28/2 | 18/3 | 0 % | | |
| | Simulering av SW i Gazebo. | 30 | | 7/3 | 18/3 | 0 % | | |
| | Utvikle SW for deteksjon av hindringer i 3D bilde. | 40 | | 14/3 | 25/3 | 0 % | | |
| 4 | Analyse av behov for navigasjon og posisjonering. | 55 | 0 | 28/3 | 22/4 | 0 % | TBD | 23/4: Test av navigering utenom bær-rekke |
| | Kort studie om hvordan fjell og plasttunnell kan påvirke GPS signal | | | | | 0 % | | |
| | Valg av navigasjonsstrategi fra ende til ny bærekke. | 15 | | 28/3 | 15/4 | 0 % | | |
| | Analysere behov for sensorikk for navigasjon mellom bærekke. | 10 | | 4/4 | 22/4 | 0 % | | |
| | Utvikle SW for navigasjon fra ende til ny bærekke. | 30 | | 4/4 | 22/4 | 0 % | | |
| 5 | Simulering av navigasjon i Gazebo. | 30 | | 11/4 | 22/4 | 0 % | TBD | 20/4: Test av komplett simulering |
| 6 | Test av HW og SW på Huskey robot. | 70 | 0 | 25/4 | 29/4 | 0 % | Isak F. | 28/4: Test av Husky robot |
| | Montering av sensor og støtfanger. | 20 | | 25/4 | 29/4 | 0 % | | |
| | Testing av sensor og støtfanger | 25 | | 25/4 | 29/4 | 0 % | | |
| | Testing av navigasjon. | 25 | | 25/4 | 29/4 | 0 % | | |
| | Presentasjon av robotløsning ved NJØS Leikanger | | | | | | | |
| 7 | Dokumentasjon | 400 | 0 | 18/4 | 27/5 | 0 % | Aril T. | 13/5: Første utkast rapport |
| | Skriving av Bachelorrapport | 400 | | 18/4 | 27/5 | 0 % | | |
| | Totale antatt timer: | 1115 | | | | | | |
| | Totale arbeidstimer: | | 0 | | | | | |

*Figur 35: Første fremdriftsplan*

Den originale tidsplanen inneholdt grove timeestimat for de ulike gjøremålene. Gruppemedlemmene fikk også ansvar for egne områder i tidsplanen.

**Hva som fungerte bra:**

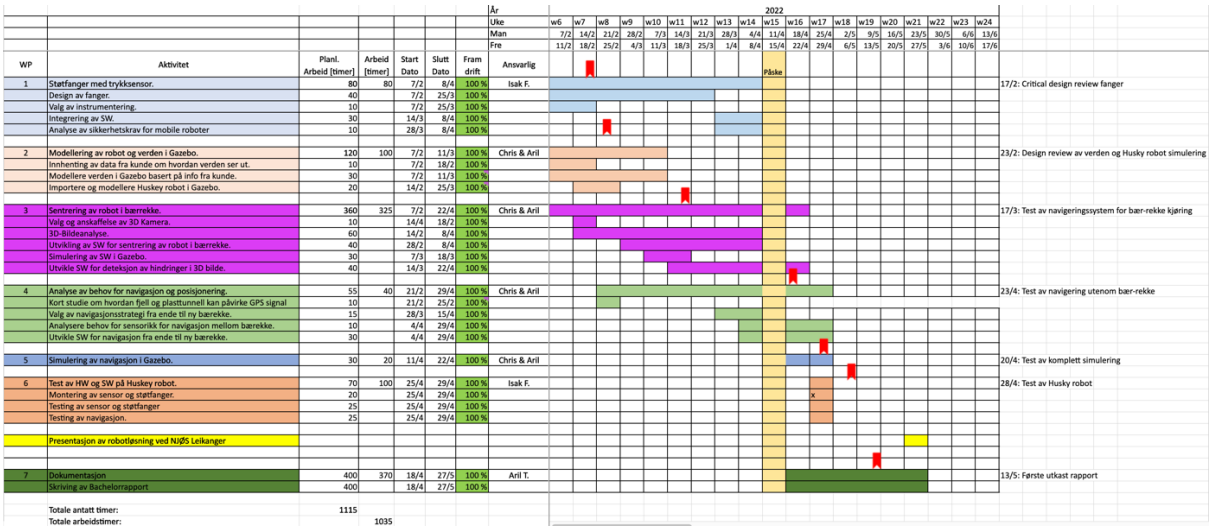
- Tidlig delegering av oppgaver gjorde at prosjektmedlemmene kom raskt i gang med sine oppgaver.
- God oversikt over start og sluttdatoer på jobbpakkene gjorde at man kan lett kunne planlegge frem i tid.

**Hva som fungerte dårlig:**

- De grove tidsestimatene stemte ikke helt overens med hva tid som faktisk gikk med til de ulike oppgavene.

Før midtveispresentasjonen ble den originale fremdriftsplanen revidert. Enkelte av gjøremålene ble dratt lenger frem i tid. Dette lot seg gjøre da det originalt var satt en del ledig tid mot slutten av prosjektperioden.

**Oppdatert fremdriftsplan:**



| WP | Aktivitet | Planl. Arbeid [timer] | Arbeid [timer] | Start Dato | Slutt Dato | Fram drift | Ansvarlig | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | År | 2022 |
| | | | | | | | Uke | w6 w7 w8 w9 w10 w11 w12 w13 w14 w15 w16 w17 w18 w19 w20 w21 w22 w23 w24 |
| | | | | | | | Man | 7/2 14/2 21/2 28/2 7/3 14/3 21/3 28/3 4/4 11/4 18/4 25/4 2/5 9/5 16/5 23/5 30/5 6/6 13/6 |
| | | | | | | | Fre | 11/2 18/2 25/2 4/3 11/3 18/3 25/3 1/4 8/4 15/4 22/4 29/4 6/5 13/5 20/5 27/5 3/6 10/6 17/6 |
| 1 | Støtfanger med trykksensor. | 80 | 80 | 7/2 | 8/4 | 100 % | Isak F. | 17/2: Critical design review fanger |
| | Design av fanger. | 40 | | 7/2 | 25/3 | 100 % | | |
| | Valg av instrumentering. | 10 | | 7/2 | 25/3 | 100 % | | |
| | Integrering av SW. | 30 | | 14/3 | 8/4 | 100 % | | |
| | Analyse av sikkerhetskrav for mobile roboter | 10 | | 28/3 | 8/4 | 100 % | | |
| 2 | Modellering av robot og verden i Gazebo. | 120 | 100 | 7/2 | 11/3 | 100 % | Chris & Aril | 23/2: Design review av verden og Husky robot simulering |
| | Innhenting av data fra kunde om hvordan verden ser ut. | 10 | | 7/2 | 18/2 | 100 % | | |
| | Modellere verden i Gazebo basert på info fra kunde. | 30 | | 7/2 | 11/3 | 100 % | | |
| | Importere og modellere Huskey robot i Gazebo. | 20 | | 14/2 | 25/3 | 100 % | | |
| 3 | Sentrering av robot i bærrekke. | 360 | 325 | 7/2 | 22/4 | 100 % | Chris & Aril | 17/3: Test av navigeringssystem for bær-rekke kjøring |
| | Valg og anskaffelse av 3D Kamera. | 10 | | 14/4 | 18/2 | 100 % | | |
| | 3D-Bildeanalyse. | 60 | | 14/2 | 8/4 | 100 % | | |
| | Utvikling av SW for sentrering av robot i bærrekke. | 40 | | 28/2 | 8/4 | 100 % | | |
| | Simulering av SW i Gazebo. | 30 | | 7/3 | 18/3 | 100 % | | |
| | Utvikle SW for deteksjon av hindringer i 3D bilde. | 40 | | 14/3 | 22/4 | 100 % | | |
| 4 | Analyse av behov for navigasjon og posisjonering. | 55 | 40 | 21/2 | 29/4 | 100 % | Chris & Aril | 23/4: Test av navigering utenom bær-rekke |
| | Kort studie om hvordan fjell og plasttunnell kan påvirke GPS signal | 10 | | 21/2 | 25/2 | 100 % | | |
| | Valg av navigasjonsstrategi fra ende til ny bærekke. | 15 | | 28/3 | 15/4 | 100 % | | |
| | Analysere behov for sensorikk for navigasjon mellom bærekke. | 10 | | 4/4 | 29/4 | 100 % | | |
| | Utvikle SW for navigasjon fra ende til ny bærekke. | 30 | | 4/4 | 29/4 | 100 % | | |
| 5 | Simulering av navigasjon i Gazebo. | 30 | 20 | 11/4 | 22/4 | 100 % | Chris & Aril | 20/4: Test av komplett simulering |
| 6 | Test av HW og SW på Huskey robot. | 70 | 100 | 25/4 | 29/4 | 100 % | Isak F. | 28/4: Test av Husky robot |
| | Montering av sensor og støtfanger. | 20 | | 25/4 | 29/4 | 100 % | | |
| | Testing av sensor og støtfanger | 25 | | 25/4 | 29/4 | 100 % | | |
| | Testing av navigasjon. | 25 | | 25/4 | 29/4 | 100 % | | |
| | Presentasjon av robotløsning ved NJØS Leikanger | | | | | | | |
| 7 | Dokumentasjon | 400 | 370 | 18/4 | 27/5 | 100 % | Aril T. | 13/5: Første utkast rapport |
| | Skriving av Bachelorrapport | 400 | | 18/4 | 27/5 | 100 % | | |
| | Totale antatt timer: | 1115 | | | | | | |
| | Totale arbeidstimer: | | 1035 | | | | | |

***Figur 36: Oppdatert fremdriftsplan***

## B.4 Timelister

Vedlagt er timelistene som har blitt ført underveis i prosjektet. De timene som har gått med til gjøremål som er merket på fremdriftsplanen er merket med korresponderende farge.

| Navn: Aril Torheim | | |
|---|---|---|
| Dato: | Timer: | Aktivitet: |
| 17.jan | 1,0 | Planlegging av hva slags type filorganisering og tidsplanleggings-metode |
| 21.jan | 2,0 | Undervisning om tidsplan/nedbryting av prosjekt |
| 24.jan | 2,0 | Startet å dele jobbpakker opp i gjøremål og lage tidsplan |
| 25.jan | 4,0 | Ferdiggjøre tidsplan og startet på forstudie |
| 26.jan | 4,0 | Fortsatte på forstudie (om prosjektet, avgrensning, innledning) |
| 27.jan | 2,5 | Fortsatt på forstudie (fremgangsmetode) |
| 28.jan | 5,0 | Undervisning om kravspekk, diskutert forstudie med Martin, og fortsatt på forstudie |
| 01.feb | 5,0 | Kravspesifikasjon, organisasjonskart, søking etter sikkerhetsstander |
| 02.feb | 3,0 | Fullført første utkast forstudie. Kildehenvisninger, lagt inn budsjett og finskrevet div. |
| 04.feb | 2,0 | Oppdatert forstudie og laget veiledningsavtale |
| 07.feb | 4,0 | Laget siste revisjon av forstudie, levert rettleiingsavtale, forslag til design støtfanger |
| 08.feb | 3,0 | Research 3D kamera og Gazebo world modellering |
| 09.feb | 3,0 | Gazebo world modelling, lage world og launch filer |
| 10.feb | 6,0 | Gazebo world modelling, modellering av busker og få åpnet de i world |
| 14.feb | 4,0 | Modellering av busker i Sketchup |
| 15.feb | 2,0 | Low poly bush models og møte |
| 16.feb | 6,0 | Low poly bush models C# skript, XML fil og veiledningsmøte |
| 21.feb | 5,0 | Laget rekker med busker i world fil |
| 22.feb | 4,0 | Ground plane |
| 07.mar | 6,0 | Dualboot Ubuntu (Var feil versjon, måtte avinstallere og reinstallere) |
| 08.mar | 4,0 | Installasjon av ROS melodic og opprettet workspace |
| 09.mar | 6,0 | Ukesmøte og Clearpath robotics ROS Tutorial fra start til Udev Rules |
| 10.mar | 6,0 | Clearpath robotics ROS tutorial, navigasjon av Husky og Jackal ved bruk av laser og mapping |
| 11.mar | 6,0 | The construct kurs, Linux |
| 12.mar | 7,0 | The construct kurs, Python |
| 13.mar | 6,0 | The construct kurs, ROS in 5 days, Topics |
| 14.mar | 8,0 | The construct kurs, ROS in 5 days, Topics og services |
| 15.mar | 6,0 | The construct kurs, ROS in 5 days, Services og python classes |
| 16.mar | 8,0 | The construct kurs, ROS in 5 days, Actions |
| 17.mar | 6,0 | The construct kurs, ROS in 5 days, Actions, debugging and apendix |
| 18.mar | 6,0 | Teleop joy with playstation controller |
| 21.mar | 4,0 | Teleop joy with playstation controller and launch file for world - modell spawning |
| 22.mar | 4,0 | Navigasjon og slam i ROS |
| 23.mar | 0,5 | Veiledningsmøte |
| 24.mar | 4,0 | Testing av 3D kamera konfigurasjoner på Husky |
| 28.mar | 4,0 | Testing av slam algorytmer, prøve bruke 3D kamera i Slam |
| 29.mar | 4,0 | Bruk av RGB filter og point cloud i openCV |
| 30.mar | 8,0 | Utviklet RGB line sentration following algoritme med openCV |
| 31.mar | 3,0 | Fullføre PP presentasjon, og øve til midtveispresentasjon |
| 01.apr | 3,0 | Siste øving og fremføring av midtveis presentasjon |
| 04.mai | 8,0 | Utvikling av edge detection algoritme for sentrering i bærrekke med openCV |
| 05.apr | 3,0 | Utforsking av muligheter for detektering av plan med open3D |
| 06.apr | 5,0 | Utvikling av depth detection algoritme for sentrering i bærrekke med openCV |
| 07.apr | 3,0 | Utforsking av optical flow for rekke sentrering |
| 08.apr | 2,0 | Funnet og installert openCV pakker for optical flow beregning |
| 09.apr | 4,0 | Utviklet algoritme for rekke sentrering ved bruk av optical flow |
| 10.apr | 3,0 | Feilsokt algoritme for rekke sentrering ved bruk av optical flow |
| 11.apr | 2,0 | Testet algoritme for rekke sentrering ved bruk av optical flow |
| 12.apr | 3,0 | Reasearch om GPS-waypoint navigasjon for bruk i jordbruksapplikasjoner (The construct kursmodul) |
| 19.apr | 6,0 | Bruk av HiDrive pakke for aa utvfore gps waypoint navigation |
| 20.apr | 3,0 | Konfigurering og bruk av pointcloud to laserscan pakke |
| 21.apr | 5,0 | Utvikling av object stop algoritme ved bruk av pointcloud to laserscan pakke |
| 22.apr | 6,0 | Utvikling av rekke sentrering ved bruk av pointcloud_to_laserscan |
| 22.apr | 1,0 | Satt opp forslag til overskrifter rapport |
| 24.apr | 3,0 | Forbredt programvare til test i Førde |
| 25.apr | 11,0 | Test av software på Husky robot i Førde, opplasting av software og feilsøking |
| 26.apr | 10,0 | Test av software på Husky robot i Førde, fysiske tester av rekke sentrering og rekkefølging |
| 27.apr | 3,0 | Satt opp software struktur flytskjema |
| 28.apr | 6,0 | Rapport rekkesentrering: kamera, openCv etc. Renskrevet python skripter og tatt video/screenshot ved algoritme test |
| 29.apr | 7,0 | Rapport: Skrevet om de ulike rekkesentreringsalgoritmene |
| 30.apr | 2,0 | Oppdeling av software i flere programmer for a gore software mer oversiktlig |
| 01.mai | 3,0 | Programmering av controller noder til software |
| 02.mai | 7,0 | Skriving av kapittel 1, 2 og start på 3 i rapport |
| 02.mai | 4,0 | Programmering av row_end_detection og obstacle_detection noder |

| | | |
|---|---|---|
| 03.mai | 4,0 | Rapportskriving om ROS |
| 04.mai | 5,0 | Rapportskriving om robotnavigering |
| 05.mai | 3,0 | Rapportskriving og tur til Førde |
| 08.mai | 6,0 | Programmering av move service |
| 09.mai | 4,0 | Feilsøking waypoint_navigation pakke |
| 10.mai | 4,0 | Videreutviklet move service, og gjort row centration om til service |
| 11.mai | 13,0 | Forbedret rowCentration algoritme med trigonometri, testet depth_to_laserscan og laget downscale filter for pointcl |
| 12.mai | 4,00 | Feilsøking waypoint_navigation pakke |
| 13.mai | 2,00 | Rapportskriving om husky |
| 15.mai | 7,00 | Feilsøking waypoint_navigation pakke |
| 16.mai | 9,00 | Prøvd å sette sammen program med row centration og waypoint GPS pakke. |
| 20.mai | 3.0 | Forbredt programvare ti Leikganer, og skrevet om et skript fra rowcentration til row following |
| 21.mai | 3.0 | Startet oppkobling av husky i Leikanger, lagt over programvare og litt feilsøking på intelrealsense streams |
| 22.mai | 12,00 | Testing av ulike programvarer på Husky i Leikanger |
| 23.mai | 14,00 | Testing og dokumentering av ulike programvarer på Husky. Forbredelse til fremvisning ved teststasjon åpning |
| 24.mai | 6,00 | Demonstrasjon av selfkjørende robot ved testasjon for NRK, Sogn avis og under åpning av teststasjon |
| 25.mai | 3,00 | Rapportskriving om forbedret kontroller og navigasjon fra rekke til rekke |
| 26.mai | 9,00 | Rapportskriving. Fullført kapittel om waypoint navigation, startet på kapittel 5 og laget node kart |
| 27.mai | 11,00 | Rapportskriving. Rapport fra Leikanger realisering av valgt løsning, testing av simulasjon, brukermanual |
| 28.mai | 6,00 | Rapportskring. Brukermanual, Installasjonsmanual, testing av fysisk robot |
| 29.mai | 10,00 | Fullført rapport |
| | 419,00 | |

| Navn: Chris Louis Johnsen | | |
|---|---|---|
| Dato: | Timer: | Aktivitet: |
| 14.01.2022 | 1,00 | Opprette mappestruktur og div. filer, Onedrive. |
| 17.01.2022 | 2,00 | Planlegging av hva slags type filorganisering og tidsplanleggings-metode. |
| 24.01.2022 | 3,00 | Utdype work packages i Gantt, tidsplanlegging (til forstudie). |
| 25.01.2022 | 4,00 | Ferdiggjøre antatt tidsplan(Gantt), Lage Flowchart av løsninger. |
| 26.01.2022 | 4,00 | Ferdiggjøre Flowchart og skrive forstudie. |
| 27.01.2022 | 2,50 | Skrive forstudie, oppdatere flowchart, revisjonslogg, møtereferat. |
| 28.01.2022 | 5,00 | Møte med Martin, møtereferat, forstudie, undervisning. |
| 01.02.2022 | 5,00 | Kravspesifikasjoner, ISO sertifisering og lese opp på research papirer. |
| 02.02.2022 | 3,00 | Ferdiggjøre første utkast forstudie, figurliste, citations, vedlegg. |
| 04.02.2022 | 3,00 | Vitenskaplig metode, rettleingsavtale e-sign, oppdatere forstudie. |
| 07.02.2022 | 4,00 | Husky GitHub, Husky repository, ROS setup. |
| 08.02.2022 | 3,00 | Husky modell i Gazebo, testet ut spawn i map, feilsøke transform error i ROS. |
| 09.02.2022 | 5,00 | Feilsøking transform error, nodes, publishers og subscribers. (ROS). |
| 10.02.2022 | 2,00 | Feilsøke i .launch filer og kurs i ROS TF. |
| 14.02.2022 | 4,00 | OpenCV ROS. |
| 16.02.2022 | 6,00 | Møte med Martin, referat, GitHub, Valg av 3D Kamera - litteratursøk vedlagt i Teams. |
| 18.02.2022 | 2,00 | Litteratursøk om Real Time Kinematics GPS til bruk for Husky. |
| 21.02.2022 | 5,00 | GPS RTK, og EMLID RS2 package i ROS. |
| 22.02.2022 | 4,00 | GPS RTK og Intel RealSense OpenCV. |
| 09.03.2022 | 3,00 | Møte med Martin og OpenCV tutorial |
| 11.03.2022 | 3,00 | Distance detection tutorial - Realsense d435 |
| 14.03.2022 | 8,00 | Lage world med Polytunnel og nye busker |
| 15.03.2022 | 5,00 | Fortsettelse med World |
| 16.03.2022 | 4,00 | Ferdiggjøre World med Polytunnel og busker |
| 17.03.2022 | 4,00 | Texturizing av busker |
| 18.03.2022 | 6,00 | Lage world på nytt pga feil i kode. |
| 21.03.2022 | 4,50 | Ferdiggjøre world, fikse mesh fil og teksturering i meshlab |
| 22.03.2022 | 4,00 | OpenCV og Python tutorial i ROS - line following. |
| 23.03.2022 | 0.5 | Møte med Martin |
| 24.03.2022 | 4,00 | Pointcloud med Realsense kamera - ROS |
| 28.03.2022 | 4,00 | Pointcloud med Realsense kamera - ROS |
| 29.03.2022 | 4,00 | Pointcloud to laserscan (bør kanskje skrive en rapport på det, koffor laserscan ikkje var ideelt?) |
| 30.03.2022 | 6,00 | Sjekke bruk av RTAB SLAM med Pointcloud - ROS |
| 31.03.2022 | 3,00 | Fullføre PP presentasjon, og øve til midtveispresentasjon |
| 01.04.2022 | 3,00 | Midtveispresentasjon øving |
| 04.04.2022 | 3,00 | RTAB SLAM med Pointcloud |
| 05.04.2022 | 1,00 | Sjekke muligheter for kjøring av RTAB SLAM, nødvendig med dualboot grunnet lagringsplass |
| 06.04.2022 | 0,50 | Møte med Martin |
| 06.04.2022 | 3,50 | Dualboot Ubuntu, reinstallering av macOS image pga diskfeil |
| 07.04.2022 | 2,00 | Dualboot Ubuntu, videre undersøking av diskfeil |
| 12.04.2022 | 10,00 | Formatering av macOS, rette opp diskfeil, installere ubuntu og rette WIFI feil, ROS melodic |
| 13.04.2022 | 2,00 | Fullføre ROS installasjon, ROS packages, The Construct navigation tutorial med GPS |
| 20.04.2022 | 3,00 | Møte med Martin, skrive bachelorrapport angående world. |
| 24.04.2022 | 3,00 | Bachelorrapport skriving - World |

| | | |
|---|---|---|
| 25.04.2022 | 10,00 | Feilsøking og test av Husky, oppkobling, software og HW |
| 26.04.2022 | 11,00 | Test av Husky, oppkobling, software og HW |
| 27.04.2022 | 3 | Raspberry bot software structure |
| 28.04.2022 | 6 | Bachelorrapport skriving - World - Gazebo |
| 29.04.2022 | 7 | Bachelorrapport skriving - Digital Twin for AMR |
| 01.05.2022 | 2 | Skrive om igjen Gazebo - rapportskriving - endre referanser |
| 02.05.2022 | 7 | Bachelorskriving, design og iterasjoner |
| 03.05.2022 | 4 | Rapportskriving ferdiggjøre simulerings kapittelet |
| 04.05.2022 | 5 | Rapportskriving og proofreading/retting/forbedringer, redigere bilder til rapport |
| 07.05.2022 | 5 | Lage Youtube video og redigere bilder til rapport |
| 12.05.2022 | 5 | Test av world filer og lage ny world i henhold til testfasilitet |
| 13.05.2022 | 5 | Ferdiggjøre ny world i henhold til testfasilitet |
| 22.mai | 12,00 | Test av Husky i Leikanger, filming |
| 23.mai | 14,00 | Test av Husky i Leikanger, filming |
| 24.mai | 6,00 | Test av Husky i Leikanger, filming |
| 26.mai | 10,00 | EXPO Plakat og film |
| 27.mai | 8,00 | EXPO Plakat og film |
| 28.mai | 10,00 | Ferdiggjøre EXPO plakat, lage, klippe og redigere filmer til Youtube |
| 29.mai | 10,00 | Rapportskriving, appendiks B, redigere flere filmer |
| | 301,00 | |

| Navn: Isak Vamråk Førde | | |
|---|---|---|
| Dato: | Timer: | Aktivitet: |
| 24.01.2022 | 3,00 | Fremdriftsplan |
| 30.01.2022 | 3,00 | Budsjettutkast og |
| 07.feb | 3,00 | Dokumentert tanker om støtfanger, laget forslag til mulig ny løsning. |
| 11.feb | 2,00 | Undersøke ulike sensormodeller og skissering for design i cad |
| 12.feb | 4,00 | Laste inn huskymodell, gjort mål av husky og planlagt muligheter for å løse mekansike utfordringer |
| 16.feb | 1,00 | Møte med martin og litt research av mulige komponenter |
| 19.feb | 3,00 | Startet 3D moddelering rørklamme for feste av fanger, funnet passende rørklamme. |
| 19.feb | 8,00 | Fullført rørklamme for feste av fanger, begynt på festeplate og sylinder til demping |
| 20.feb | 8,00 | Tegnet fjæring til demper og begynt sammenstilling av fullstendig modell. |
| 22.feb | 6,00 | Tegnet kufanger i front av fanger og gjorde fullstendig sammenstilling |
| 08.mar | 6,00 | Internt møte, forberedelse til veiledermøte |
| 09.mar | 6,00 | Veiledermøte, idemyldring for nytt utkast av mindre fanger etter input fra Martin |
| 12.mar | 4,00 | Design av nytt utkast av fanger |
| 13.mar | 4,00 | Design av nytt utkast av fanger |
| 19.mar | 4,00 | Fullføre design av ny fanger |
| 20.mar | 4,00 | Fullføre design av ny fanger |
| 22.mar | 4,00 | Internt møte, forberedelse til veiledermøte |
| 23.mar | 4,00 | Veiledermøte, |
| 26.mar | 6,00 | Endringer og tilpasninger etter funn av nye brytere |
| 27.mar | 4,00 | Endringer og tilpasninger etter funn av nye brytere |
| 31.03.2022 | 3,00 | Fullføre PP presentasjon, og øve til midtveispresentasjon |
| 01.04.2022 | 3,00 | Midtveispresentasjon øving |
| 25.apr | 11,0 | Test av software på Husky robot i Førde, opplasting av software og feilsøking |
| 26.apr | 10,0 | Test av software på Husky robot i Førde, fysiske tester av rekke sentrering og rekkefølging |
| 30.apr | 5,00 | Lest i ANSI/R15 standard etter krav til rapportskriving |
| 03.mai | 5,00 | Brukt standard og startet rapportskriving om sikkerhet |
| 04.mai | 1,00 | Veiledermøte med martin |
| 05.mai | 2,00 | Leitet frem og bestillt deler til bygging av frontfanger |
| 06.mai | 1,00 | Justert litt design med nye brytere og lastet ned 3d filer av bestillte brytere |
| 07.mai | 2,00 | Funnet deler til sammenstilling av fanger |
| 08.mai | 6,00 | Begynnt på fanger, skjøre ut metalldel i metallplate med vinkelsliper |
| 13.mai | 4,00 | Rapportskriving design av fanger |
| 14.mai | 3,00 | Rapportskriving design av fanger |
| 15.mai | 2,00 | Kontaktblokk manglet, funnet riktig og bestillt (ikke standard så måtte leite/forespør direkte) |
| 16.mai | 3,00 | Refleksjonsnotat |
| 18.mai | 3,00 | Skjærte ut resten av fanger med vinkelsliper |

| | | |
|---|---|---|
| 18.mai | 3,00 | Skjærte ut resten av fanger med vinkelsliper |
| 19.mai | 4,00 | Pusset, filt og slipet fangeren for å bli kvitt skarpe kanter. Laget hull til feste og brytere |
| 20.mai | 5,00 | Montert og koblet alle bryterene i fangeren, montert kevlar beskyttelse rundt brytere med strips |
| 21.mai | 3,00 | Leikanger |
| 22.mai | 12,00 | Leikanger |
| 23.mai | 14,00 | Leikanger |
| 24.mai | 6,00 | Leikanger |
| 25.mai | 5,00 | Endret smått og kommentert i arduinoprogram og skrevet rapport |
| 26.mai | 5,00 | Finjustert alle 3D modeller så de er klar til å legges ved som vedlegg |
| 27.mai | 5,00 | Rapportskriving første utkast av frontfanger justere overskrifter |
| 28.mai | 10,00 | Rapportskriving Sikkerhetsystemer |
| 29.mai | 10,00 | Rapportskriving Sikkerhetsystemer |
| 30.mai | 14,00 | Fullføring av rapport, diskusjon og gjennomlesing og retting |
| | 244,00 | |

# Appendiks C Brukerdokumentasjon

## C.1 Brukerdokumentasjon

**Oppstart av simuleringsverden**

Simuleringsverdenen startes ved å skrive følgende kommando i et tomt terminalvindu:

```
aril@aril-Komplett-PC:~$ roslaunch raspberry_bot tunnel.launch
```

Gazebo vil da åpne verdenen. Verdenen vil sannsynligvis spawne uten modeller. Dette skyldes at stien til modellene i skriptet tunnel.world i raspberry_bot pakken må tilpasses datamaskinen den kjøres på. Det er stien til modellene berryrow.dae og GazeboPolyTunnell.dae som må modifiseres. Når riktig sti er funnet kan alle stiene i skriptet modifiseres samtidig ved å bruke funksjonene «Find in current buffer» og «Replace in current buffer» i Linux programmet Atom.

*Figur 37: Simuleringsverdenen*

Når verdenen er startet må selve roboten spawnes. En variabel som henviser til urdf.xaco filen som beskriver roboten må først kjøres i terminalvinduet. Denne filen ligger i mbs_husky_description pakken og heter mbs_husky_updates.urdf.xacro. Variabelen settes eksempelvis slik:

```
aril@aril-Komplett-PC:~$ export HUSKY_URDF_EXTRAS=/home/aril/catkin_ws/src/mbs/mbs_husky_description/
urdf/mbs_husky_updates.urdf.xacro
```

Variabel for å spawne realsense kameraet må også settes. Kjør følgende i samme terminalvindu:

```
aril@aril-Komplett-PC:~$ export HUSKY_REALSENSE_ENABLED=1
```

Når dette er gjort må følgende kommando kjøres i samme terminalvindu:

```
aril@aril-Komplett-PC:~$ roslaunch husky_gazebo spawn_husky.launch
```

Roboten skal da spawne i origo i verdenen.

*Figur 38: Husky robot spawnet i simuleringsverdenen*

Ved å koble til en joystick til PC'en skal det nå være mulig å kjøre roboten rundt i simulerings verdenen.

**Rowcentration service**

For å kjøre rekkesentreringsalgoritme må først alle programmene tilknyttet dette startes opp. Dette gjøres på samme måte i simulering og på den fysiske roboten. Kjør følgende kommando i et tomt terminalvindu:

```
aril@aril-Komplett-PC:~$ roslaunch raspberry_bot rowcentration.launch
```

Alle noder knyttet til navigasjon og sikkerhet ved rekkesentrering vil da starte opp.

For å starte rekkesentreringen må først roboten stilles opp med fronten mot senterlinjen mellom to bærrekker slik at kamera oppfatter rekkene. Deretter kjøres følgende service kall i et tomt terminalvindu:

```
aril@aril-Komplett-PC:~/catkin_ws$ rosservice call /raspberry_bot/rowCentering/c
ommand "data: true"
```

Dette terminalvinduet vil være "opptatt" til roboten når enden av rekken. Servicen vil returnere en tom melding når roboten har nådd enden av rekken, og terminalvinduet vil igjen bli ledig for bruk.

```
aril@aril-Komplett-PC:~/catkin_ws$ rosservice call /raspberry_bot/rowCentering/c
ommand "data: true"
success: True
message: ''
```

**Rowfollowing service**

For å kjøre rekkefølgingsalgoritme må brukeren først vite om roboten skal følge en rekke som ligger til høyre eller venstre side for sin senterlinje. Kjør en av følgende kommandoer i terminalvindu:

```
aril@aril-Komplett-PC:~$ roslaunch raspberry_bot rowFollowing_left.launch
aril@aril-Komplett-PC:~$ roslaunch raspberry_bot rowFollowing_right.launch
```

For å starte rekkefølging må først roboten stilles opp slik at bærrekken kan oppfattes av kameraet. Deretter kjøres følgende service kall i et tomt terminalvindu:

```
aril@aril-Komplett-PC:~/catkin_ws$ rosservice call /raspberry_bot/rowFollwer/com
mand "data: true"
```

Dette terminalvinduet vil være "opptatt" til roboten når enden av rekken. Servicen vil returnere en tom melding når roboten har nådd enden av rekken, og terminalvinduet vil igjen bli ledig for bruk.

```
aril@aril-Komplett-PC:~/catkin_ws$ rosservice call /raspberry_bot/rowFollwer/com
mand "data: true"
success: True
message: ''
```

**Move service**

Move service kan brukes til å enten kjøre roboten i rette linjer, eller snu den rundt sin egen akse. Move service er aktivert når enten rekkefølgings- eller rekkesentreringsservice er aktivt. Følg bruksanvisningen til en av disse servicene for å samtidig aktivere move service.

Det er noen forskjeller mellom service kallet for move service i simulasjon og på den fysiske roboten. Dette skyldes at den egenproduserte srv_msg move_command ikke virket på den fysiske roboten. Det måtte derfor brukes en msg som innehold noen av de samme datatypene. Msg typen som ble bruk var turlesim.srv Spawn.

Move service trenger to argumenter for å kjøre:

Bevegelse: enten «linear» eller «rotasjon».

Input: meter for linear bevegelse, og radianer for rotasjon bevegelse.

For den fysiske roboten heter argumentet Bevegelse i stedet «name», og input heter i stedet «x».

Eksempelvis et service kall som skal kjøre roboten 1 meter rett frem, kalles med følgende kommando i et tomt terminalvindu:

```
aril@aril-Komplett-PC:~$ rosservice call /raspberry_bot/move/command "Bevegelse: 'linear'
Input: 1.0"
```

Linear kall kjører roboten i en rett linje fremover til den har kjørt en distanse tilsvarende variabelen Input. Rotasjons kall roterer roboten rundt sin egen akse til den har rotert en vinkel tilsvarende Input i radianer.

Dette terminalvinduet vil være “opptatt” til roboten når enden av rekken. Servicen vil returnere variabelen «success: True» når roboten har nådd enden av rekken, og terminalvinduet vil igjen bli ledig for bruk.

```
aril@aril-Komplett-PC:~/catkin_ws$ rosservice call /raspberry_bot/move/command "Bevegelse: 'linear'
Input: 1.0"
success: True
```

**GPS-waypoint navigation**

For å kjøre GPS-waypoint navigation må først følgende kommandoer kjøres i ulike terminalvinduer:

```
aril@aril-Komplett-PC:~$ roslaunch mbs_husky_waypoint_navigation waypoint_husky_
movebase.launch
```

```
aril@aril-Komplett-PC:~$ roslaunch mbs_utm_handler localization.launch
```

```
aril@aril-Komplett-PC:~$ roslaunch mbs_husky_waypoint_navigation waypoint_naviga
tion.launch
```

Instruksjoner for hvordan pakken brukes vil dukke opp i terminalvinduet som waypoint_navigation.launch kommandoen blir utført i.

**Bruk av command service klient**

Command service klienten er bygget opp som en liste som kaller ulike servicer fra roboten. Service klienten er bygget opp med metoder som utfører selve service kallet, slik at det skal kreves så lite kode som mulig i selve listen. Listen defineres inne i metoden:

```
def command_program(self):
```

Kall av move service kan utføres med følgende metodekall:

```
bevegelse = "linear"
input = 2.5
self.move_command(bevegelse, input)
```

Kall av rekkesentrering kan utføres med følgende metodekall:

```
self.rowCentering()
```

Kall av rekkefølging kan utføres med følgende metodekall:

```
self.rowFollowing()
```

Dersom GPS-waypoint navigation skal kalles krever dette noe mer tekst. GPS-waypoint navigation er en egen pakke, og må derfor kjøres med roslaunch. For at roslaunch kallene skal virke må først følgende kommandoer være utført:

```
aril@aril-Komplett-PC:~$ roslaunch mbs_husky_waypoint_navigation waypoint_naviga
tion.launch
```

```
aril@aril-Komplett-PC:~$ roslaunch mbs_utm_handler localization.launch
```

GPS-koordinatene som roboten skal navigere til må være listet i en .txt fil. Denne filen blir linket til i en launch fil av typen send_goals.launch i pakke mbs_husky_waypoint_navigation. For å kalle opp pakken og få den til å navigere roboten til GPS-koordinatene benyttes følgende metodekall:

```
input = "/home/aril/catkin_ws/src/mbs/mbs_husky_waypoint_navigation/launch/include/send_goals.launch"
self.start_waypointnav(input)
```

Her må filstien i input variabelen byttes ut med riktig sti til send_goals.launch filen. Egne send_goals.launch filer kan også lages ved å duplisere den originale filen.

**Spesielle noder for den fysiske roboten**

Den fysiske roboten har noen noder som simuleringen ikke har. Disse nodene beskrives videre her. Med unntak av seriell noden vil det ikke være behov for å starte disse nodene manuelt da dette gjøres av launch filene, men det kan være greit å ha en oversikt over hva de ulike nodene gjør dersom det skulle oppstå problemer.

**Seriell node**

For å opprette kommunikasjon med Arduinoen som overvåker tilstandene på trykksensorene på støtfangeren må seriell noden startes. Først må rettighetene for USB seriell kommunikasjonen settes for at seriell kommunikasjon skal bli tillatt. Dette gjøres ved å kjøre følgende kommando:

```
administrator@administrator:~/catkin_ws$ sudo chmod +777 /dev/ttyACM0
```

Dersom navnet på tilkoblingen endrer seg, kan man prøve å finne navnet på Arduinoen ved å bruke følgende kommando:

```
aril@aril-Komplett-PC:~$ ls -la /dev/tty*
```

Seriell noden startes med følgende kommando:

```
administrator@administrator:~/catkin_ws$ rosrun rosserial_python serial_node.
py
```

Kommunikasjon skal nå være aktivert og noden skal gi beskjed videre til obstacle_detection node dersom en kollisjon blir detektert.

**rs_camea.launch**

Denne launch filen oppretter kontakt med realsense kameraet og setter alle parameterne knyttet til dette.

**tf_realsense.launch**

For at roboten skal kunne visualisere bildet som blir sett i kamera i forhold til sin egen koordinat ramme må kameraets koordinat ramme kobles mot en koordinat ramme som befinner seg på roboten. Dette blir utført med denne launch filen. Koordinat rammen camera_link blir festet til koordinat rammen top_plate_front_link på roboten.

## C.2 Installasjon

**Hardware**

Simuleringen av softwaren er utført på PC med følgende spesifikasjoner:

- Prosessor: AMD Ryzen 7 5800X
- Grafikkort: GeForce RTX 3070
- Minne: 32GB DDR4 RAM

Det ble installert Ubuntu 18.04 Bionic Beaver operativsystem i dual-boot konfigurasjon for å kunne kjøre ROS. ROS versjonen som ble benyttet var ROS Melodic Morenia

**raspberry_bot**

Software pakken raspberry_bot som følger med denne oppgaven må plasseres i /catkin_ws/src. For å installere pakken brukes kommandoen catkin_make i et terminalvindu åpnet i mappen /catkin_ws. Dersom dette ikke virker brukes følgende kommando:

```
aril@aril-Komplett-PC:~/catkin_ws$ catkin_make --only-pkg-with-deps raspberry_bot
```

**Husky**

Husky simuleringspakken fra Clearpath Robotics må også installeres i /catkin_ws/src. Denne pakken kan lastes ned fra GitHub med følgende kommando:

```
aril@aril-Komplett-PC:~/catkin_ws/src$ sudo git clone -b melodic-devel https://github.com/husky/husky
.git
```

Pakken installeres med catkin_make kommando i catkin_ws. Dersom dette ikke fungerer kan catkin_make –only-pkg-with-deps kommando, etterfulgt av navnet på hver enkelt pakke i husky mappen brukes.

**MBS**

Dette er den spesialtilpassede pakken som følger roboten fra My Bot Shop. Høyskolen på Vestlandet har kjøpt denne pakken. For å få tilsendt denne pakken må en av kontaktpersonene som arbeider med Husky roboten på høyskolen kontaktes. Pakken kommer i en mappe som heter HiDrive. Når gruppen fikk tilsendt mappen manglet det 3 filer. Disse filene må leggeinn i mappen før installasjonen blir kjørt. En av filene som manglet var GPStoMAP.srv. For å plassere denne filen riktig må det opprettes en ny mappe med navn «srv» inne HiDrive mappen. GPStoMAP.srv plasseres inne i denne mappen.

For å starte installasjon av pakken brukes følgende kommando inne i et terminalvindu i HiDrive mappen:

```
aril@aril-Komplett-PC:~/catkin_ws/src/HiDrive$ sudo ./husky_installation.bash
```

Installasjonsveiledning vil dukke opp i terminalvinduet.

Når installasjon er fullført, må catkin_make utføres. Dersom dette ikke virker må catkin_make –only-pkg-with-deps kommando, etterfulgt av navnet på hver enkelt pakke i mbs mappen brukes.

**Hjelpepakker**

Det trengs også en del tilleggspakker for å kjøre simulering. Det er viktig å laste ned riktig gren av pakken. Finn riktig gren eller branch inne på GitHub siden og bruk dette som et ekstra argument når filen skal lastes ned. Eksempelvis: sudo git clone -b melodic-devel, før linken som man finner inne på GitHub siden.

Følgende hjelpepakker må installeres:

| | |
|---|---|
| perception_pcl: | https://github.com/ros-perception/perception_pcl |
| fiducial_msgs: | https://github.com/BYUMarsRover/fiducial_msgs |
| costmap_converter: | https://github.com/rst-tu-dortmund/costmap_converter |
| find-object: | https://github.com/introlab/find-object |
| geometry2: | https://github.com/ros/geometry2 |
| lms1xx: | https://github.com/clearpathrobotics/LMS1xx |
| move_base_flex: | https://github.com/magazino/move_base_flex |
| navigation: | https://github.com/ros-planning/navigation |
| navigation_msgs | https://github.com/ros-planning/navigation_msgs |
| pointcloud_to_laserscan: | https://github.com/ros-perception/pointcloud_to_laserscan |
| realsense-ros: | https://github.com/IntelRealSense/realsense-ros |
| rgbd_launch: | https://github.com/ros-drivers/rgbd_launch |
| robot_localization | https://github.com/cra-ros-pkg/robot_localization |
| teleop_twist_joy: | https://github.com/ros-teleop/teleop_twist_joy |
| twist_mux: | https://github.com/ros-teleop/twist_mux |
| velodyne_simulator: | https://github.com/lmark1/velodyne_simulator |

# Appendiks D Kildekode og vedlegg i Zip-fil

Programvare og en del tegninger er levert med oppgaven som en zip-fil.

Zip-filen inneholder følgende mapper:

**raspberry_bot fysisk robot:**

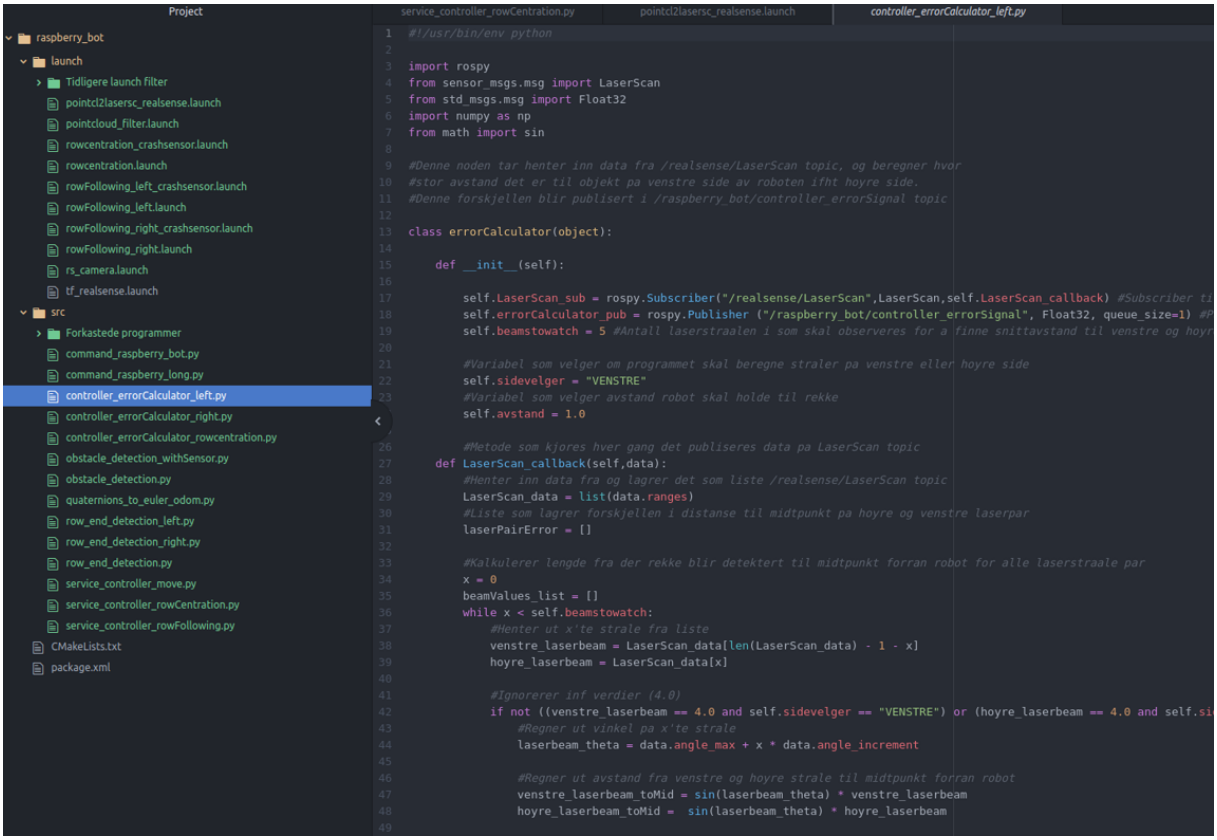
- Programvaren i form av en ROS pakke for den fysiske roboten

**raspberry_bot simulering:**

- Programvaren i form av en ROS pakke for simulering

**Vedlegg kollisjonsfanger**

- Arduino skript
- Frontfanger 3D-modeller
- Frontfanger koblingsskjema
- Mekanisk målsatte tegninger




*Figur 39: Eksempel kode*

# Appendiks E Rapporter fra testturer

## E.1 Rapport fra testing i Førde

Sted: Førde
Dato: 25-26 April
Til stede: Aril Torheim, Chris Louis Johnsen, Isak Vamråk Førde, Martin Fodstad Stølen og Raquel Motzfeldt.

Prosjektgruppen reiste til Førde for å teste ut diverse software på Husky roboten. Målet var å finne ut om software fungerte like bra i felt som i simulering. Det viste seg som forventet at algoritmer som baserte seg på bildebehandling ved hjelp av openCV ble veldig påvirket av støy i felt, og at point cloud depth detection var mer robust.

**Dag 1**

Dag 1 startet med et møte sammen med Martin der fremdriften ble gjennomgått, og det ble satt opp en plan for hvordan gruppen skal komme i mål med oppgaven. Martin gikk også gjennom hvordan testfasiliteten i Leikanger blir utformet. Denne blir bestående av kun to bærrekker, noe som fører til at rekkesentreringsalgoritmen kun kan brukes på et sted. Det vil derfor være aktuelt å utvikle en rekkefølgings algoritme som skal kjøre en fast avstand fra en rekke, istedenfor å sentrere seg mellom to rekker. En slik algoritme ble kjapt utviklet og testet i simulator slik at denne også kunne testet på roboten i Førde. Rekkefølging fungerte bra, men det ble problemer når roboten kom til enden av rekken siden kameraet så for langt frem. Roboten ville derfor prøve å svinge for tidlig for så å kjøre i rekken når den nærmer seg enden. Det måtte studeres nærmere senere for å finne en god løsning på dette problemet.

Dagen fortsatte med å gjøre seg kjent med husky roboten, koble seg til roboten fra PC via SSH, installere egenprodusert software og laste ned pakker som manglet. Raquel var til god hjelp under dette arbeidet. Selve oppkoblingen gikk ganske smertefritt, men det oppstod en del problemer når realsense kameraet skulle kobles opp. Første problemet var å finne riktige pakker som skulle installeres for å kunne streame bilde til roboten. Når dette var løst var det problemer med å sette opp transformasjonen fra kamera koordinat rammen, til robot koordinat rammen. Begge rammene var definert, men de måtte knyttes sammen. Etter en del søking på nettet ble løsningen funnet. Pakken som har kontroll på transformasjonene mellom de ulike koordinat rammene heter TF. Det måtte lages en launch fil for TF pakken som beskrev transformasjonen fra kamera koordinat rammen til den koordinat rammen på roboten som kameraet var montert på. Problemet ble dermed løst.

Videre gikk dagen med til å starte testing av algoritmene. Første testene ble utført med å sette opp to rekker med papp-plater og prøve å få roboten til å kjøre sentrert mellom disse. RGB algoritme fungerte dårlig da det var vanskelig å justere maskene slik at riktige farger ble filtrert ut. Edge detection fungerte noe bedre, men ble påvirket av kanter i bakgrunnen. Pointcloud depth

*Figur 40: Testing av algoritmer i Førde*

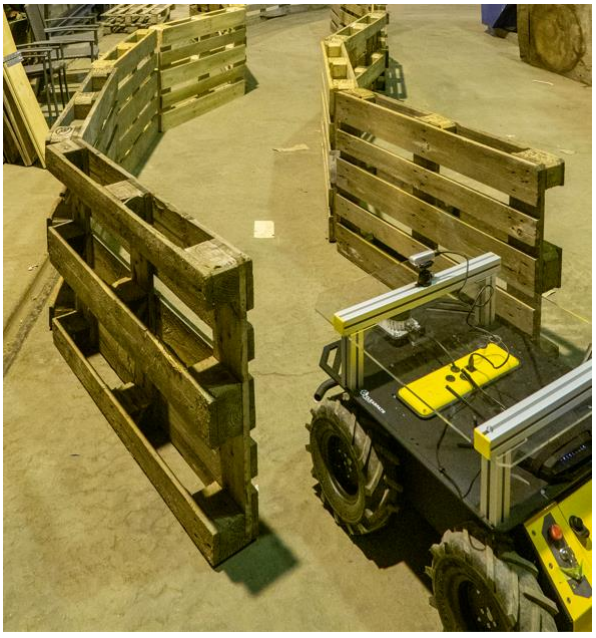
detection virket ganske stabilt og gav håp for videre testing neste dag.

**Dag 2**

Dag to ble rekkefølgings- og rekkesentreringsalgoritmer ved bruk av point cloud depth detection testet. Rekkefølgingsalgoritme ble testet ved å prøve å få roboten til å følge en vegg på utsiden av bygget. Roboten ble stilt opp parallelt og med ca. 1,5 meter avstand fra veggen. Algoritmen ble stilt inn til at robot skulle ha 1 meter avstand fra veggen før den ble startet. Roboten svinget fint inn mot veggen før den rettet seg opp og holdt en parallell linje med riktig avstand. Samme test ble utført med flere forskjellige avstander, og generelt sett ble resultatet dårligere når roboten befant seg lenger ifra veggen. Det virket som det var noe forstyrrelser når roboten kjørte forbi store glassruter, noe som kunne føre til at roboten kom ut av kurs. Samme testene ble utført ved å stille opp en rekke med paller. De ble stilt opp i en liten halvsirkel for å se om roboten klarte å holde riktig avstand gjennom svinger også. Dette fungerte bra så lenge ikke svingen gikk samme vei som den siden av roboten pallene står på, for eksempel dersom pallene står på venstre og roboten må kjøre en venstre sving. Dette er samme problemet som oppstår i simulering, nemlig at roboten tror den er på enden av rekken, og svinger dermed inn i rekken.

*Figur 41: Test av rekkefølgingsalgoritme*

Rekkesentreringsalgoritme ble testet ved å stille opp to rekker med paller, dette ble gjort for å teste om roboten klarte å kjøre sentrert mellom disse. Testene ble først utført i sakte tempo, og etter hvert i raskere tempo. P-ledet i regulatoren måtte stilles noe etter hvert som farten ble økt. På grunn av dette ble det vurdert å sette inn en faktor som skalerer P-ledd basert på lineær hastighet dersom robot skal kjøre i forskjellige hastigheter. Denne algoritmen virker å være veldig robust, noe som gir et godt grunnlag for videre utvikling.



*Figur 42: Test av rekkesentreringsalgoritme*

Video av testen kan sees ved å klikke på linken under:

https://www.youtube.com/watch?v=cyRIVTwfmDw

GPS-waypoint navigation ble også testet. Her var det noe problemer med pakken som følger med roboten som gjorde at dette ikke fungerte. Raquel skal ta kontakt med produsent, og forhåpentligvis er dette problemet løst før testing i Leikanger.

**Oppsummering**

Point cloud depth detection algoritmene virker å være robuste nok til å fungere i felt. Denne algoritmen blir derfor brukt i videre utvikling. Foreløpig detekterer ikke algoritmen hindringer, dette er derfor noe som må implementeres. Algoritmene detekterer heller ikke når den kommer til enden av en eventuell bærrekke, dette er også noe som må implementeres ved videre utvikling. Problemet med at rekkefølgingsalgoritmen kjører i rekken når den nærmer seg enden må også løsest. Alt i alt har testingen gitt godt utbytte i form av å vise hva som fungerer bra, hva som ikke fungerer bra og hva som må arbeides med videre frem mot siste test i Leikanger.

## E.2 Rapport fra testing i Leikanger

Sted: RoboVest testfasilitet, Leikanger
Dato: 21-24 Mai
Til stede: Aril Torheim, Chris Louis Johnsen, Isak Vamråk Førde og Martin Fodstad Stølen.

Prosjektgruppen reiste til RoboVest testfasilitet i Leikanger for å teste ut den endelige softwaren og hardwaren på Husky roboten. Målet var å få roboten til å navigere rundt i teststasjonen. Rekkefølgingsalgoritmen viste seg å fungere mye bedre enn forventet, selv med lite blader på bringebærbuskene. Gruppen klarte dermed å nå målet om å få roboten til å navigere rundt i teststasjonen. Det ble også utført demo for publikum under åpningen av RoboVest testfasiliteten, med rundt 50 tilskuere og media.

**Dag 1**

Gruppen skulle reise til Leikanger for å gjøre tester på Husky roboten, og klargjøre for fremvisning av selvkjørende robot under åpningen av RoboVest testfasilitet. Gruppen hadde siste eksamen fredag 20. mai, og det ble derfor bestemt å reise opp lørdagen for å få best mulig tid til testing før åpningen.

RoboVest testfasilitet inneholdt ingen bringebærrekker som roboten kunne sentrere seg mellom enda, da det var for stor avstand mellom bærrekkene som var satt opp. Gruppen hadde derfor gjort klar en rekkefølingsalgoritme som skulle brukes i stedet.

Gruppen ankom Leikanger på kvelden i 18 tiden lørdag 21.mai. Første kvelden ble brukt til å få husky roboten på plass, og koble den opp mot PC og realsense kamera slik at det ikke skulle gå med tid til dette neste dag. Det ble forsøkt å streame bilde fra realsense kamera til PC, men dybde bildene var uklare og point cloud stream var heller ikke tilgjengelig. Det ble derfor foretatt en del internett søk om kvelden på hotellet for å finne ut av hva dette problemet kunne være for å få mest mulig ut av neste dag.

*Figur 43: Oversiktsbilde RoboVest testfasilitet*

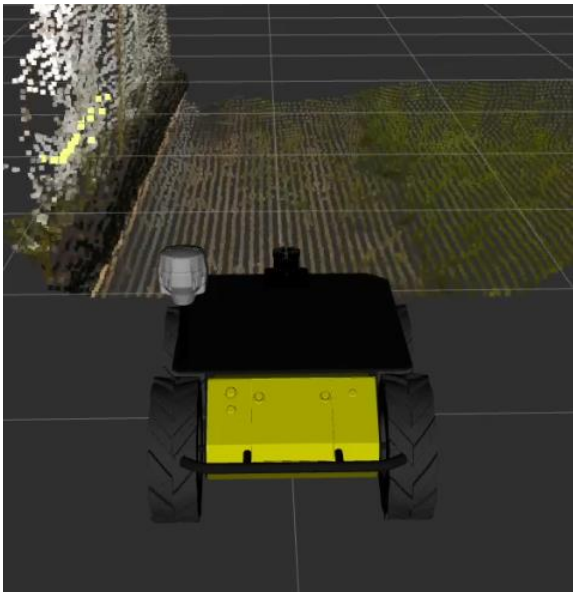
Det var ganske lite blader på bringebær-buskene i forhold til hva gruppen var forespeilet, og det ble knyttet spenning til hvordan dette ville påvirke rekkefølgingsalgoritmen.

**Dag 2**

Problemet med Realsense point cloud stream viste seg å være at ene parameteren i launch filen som blir brukt for å kjøre kamera stream var satt til «pointcloud_enabled = false». Point cloud stream viste seg i motsetning til dybde stream å være tydelig, og kamera problemene ble derfor sett på som løst.

Python skriptene til de ulike nodene ble oppdatert slik at de abonnerer på riktige topics. Alle skriptene som skulle kjøres ble også lagt inn i en felles launch fil, slik at alle skriptene kunne kjøres med en kommando.

Roboten ble stilt opp vedsiden av en bærrekke, samtidig som sensordata ble visualisert i Rviz. Point cloud stream visualiserte tydelig bærrekke, og LaserScan punktene legger seg tydelig langs bærrekken. Det overasket gruppen hvor godt dette fungerte, og det gav gruppen tro på at rekkefølgingsalgoritmen ville fungere i virkeligheten også. I figuren under vises visualiseringen i Rviz, der de gule prikkene langs rekken til venstre i bildet er laserstrålene som er visualisert fra /realsense/LaserScan topic.



*Figur 44: Visualisering av point cloud og LaserScan*

Rekkefølging med 1 meter avstand ble testet ut. Roboten ble stilt opp på skrå mot rekken, og rekkefølging service ble kalt opp. Roboten kjørte mot rekken, og svinget seg fint inn, slik at den hadde 1 meter avstand fra senterlinjen av roboten til bærrekken. Det virket som algoritmen fungerte like bra, om ikke bedre i virkeligheten enn i simuleringen. Arbeidet med å lage et program der roboten kjørte langs rekkene i hele teltet kunne derfor fortsette.

Resten av dagen ble brukt til å lage et program der roboten fulgte ene rekken ned til enden av teltet før den svinget over på andre siden og fulgte dette siden opp igjen. Det gikk med en del tid til å finne riktige vinkler og lengder som roboten måtte kjøre for å havne på riktige plasser, men når dette arbeidet var gjort klarte roboten å repetere den samme ruten om og om igjen. Ruten ble satt i en while loop slik at roboten kunne kjøre opp og ned langs rekkene av seg selv helt til programmet ble stoppet.

Støtfangeren med trykkbrytere ble også klargjort for montering. Det ble loddet på pinner på alle ledninger som skulle kobles mot Arduino kontrolleren, og ledningene ble koblet. Det ble også laget et

testprogram for å teste om frontfangeren fungerte som tiltenkt. Det fungerte ikke fra starten og resten av dagen gikk med til å finne ut hvor feilen lå.

**Dag 3**

Detektering av hindringer via realsense kamera ble testet. Dette fungerte meget godt, og roboten stoppet presist og med god margin for hindringer. Når hindringen ble fjernet fortsatte roboten navigering. Det ble også testet ut hvordan roboten reagerte på hindringer som befant seg litt til siden for senterlinjen til roboten. Dette fungerte også meget bra. Dersom hindringen befant seg på den siden som bærrekken var klarte også roboten å navigere rundt hindringen og fortsette navigeringen dersom ikke hindringen var for stor. Det oppstår problemer dersom en person står langs bærrekken, og flytter seg mot senterlinjen foran roboten imens roboten prøver å navigere rundt hindringen. Roboten detekterer dette som at bærrekken har en sving, og roboten svinger dermed vekk ifra bærrekken slik at den forsvinner ut ifra bildet til realsense kameraet. Dette fører til at roboten tror den er ved enden av rekken, og stopper opp sin navigering.

Et nytt program der roboten navigerer langs begge sider av rekkene i teltet ble også laget. Roboten navigerer da totalt langs 4 bærrekke sider. Det oppstod problemer på et punkt der det manglet to bringebærpotter, og dermed greiner. Dette ble løst ved å sette opp antall stråler som måtte gå mot infinity for at ende av bærrekke skulle detekteres.

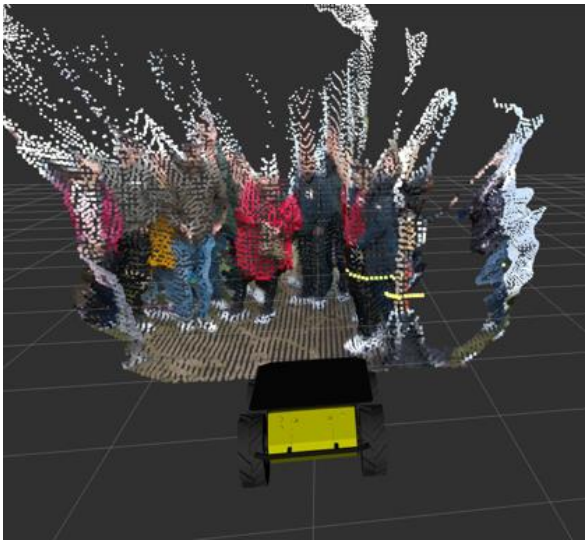
Videre feilsøking på frontfanger avslørte en koblingsfeil i forbindelse med Arduino sine interne pull-up motstander. Når dette var løst fungerte testprogrammet som tiltenkt. Vider ble et nytt program for å sende kommunikasjon over seriell bus til roboten implementert. For å gjøre dette måtte det også lastes ned et eget bibliotek. Videre ble frontfangeren montert på roboten for å kunne utføre test av programmet gjennom den serielle kommunikasjonen. Her ble det mange timer med feilsøking, da kommunikasjonen mellom de to enheten ikke fungerte som først antatt. Feilene ble til slutt utbedret, og test kunne utføres. Test ble utført der realsense kamera ble snudd til siden, og roboten ble kjørt mot en hindring. Testen var vellykket, og roboten klarte altså å stoppe når den krasjet i hindringer, uten at den påførte for mye kraft mot objektet som den krasjet med.

Det ble tatt mye foto og video underveis igjennom denne dagen som kan brukes når den endelige rapporten skal skrives.

**Dag 4**

Dagen startet med intervju av NRK, og en demo der roboten navigerte rundt alle rekkene i teltet. Det regnet mye denne morgenen, og det ble mye gjørme på dekkene til roboten når det kjørte ut av teltet for å snu mot neste bærrekke. Roboten klarte likevel å kjøre hele runden selv under gjørmete forhold.

Dagen fortsatte med seminar på fylkeshuset med innlegg fra blant annet dyrkere, og firmaer som jobber med utvikling av teknologi knyttet til frukt- og bærproduksjon.



*Figur 45: Spente tilskuere fanget opp i realsense kameraets point cloud*

Etter snorklipping på RoboVest testfasilitet ble det kjørt demo der roboten navigerte opp og ned halve teltet på innsiden av rekkene. Demonstrasjon der roboten kjørte runden gjennom hele teltet ble ikke utført, da det sto mange folk opp og ned langs rekkene i øvre del av teltet. Det var ca. 50 personer til stede. Journalist fra Høgskulen på Vestlandet og Sogn Avis var også til stede for å lage reportasjer fra åpningen. Demonstrasjonen gikk bra, og det ble mange interessante diskusjoner med personer som kom med spørsmål rundt roboten og teknologien som ble brukt.

*Figur 46: Bilde fra HVL sin reportasje.*

*https://www.hvl.no/aktuelt/denne-teknologien-skal-redde-frukt--og-barnaringa/*

**Oppsummering**

Testingen gikk bra, og programvaren fungerte mye bedre enn ventet. Testene ga også pekepinner på svakheter, som at roboten mister synet av rekken dersom den prøver å navigere rundt eventuelle store hindringer som kan befinne seg langs rekkene, og dermed kommer ut av kurs. Sikkerhetssystemer som detektering av hindringer og fangeren med trykkbrytere fungerte også tilfredsstillende og vil gjøre bruken av roboten rundt mennesker en del tryggere.

Gruppen syntes det var gøy å få teste ut softwaren på den fysiske roboten etter mange måneder med simulering. Det ble selvfølgelig ekstra gøy når testene fungerte så bra som de gjorde. Gruppen fikk tatt mye video og bilder av testene som kan brukes i den endelige rapporten. Testene ga et godt grunnlag for å fullføre den endelige rapporten.

# Appendiks F Analyse av GPS-RTK i plasttunnel

Til Husky roboten er det bestilt EMLID REACH RS2 som er en RTK GPS mottaker. Med roboten er det også bestilt software for autonomt veipunkts navigering ved bruk av GPS RTK. Hensikten til denne analysen er å dokumentere hvorfor og hvordan GPS RTK kan nyttes til navigasjon. GPS RTK er en metode som gir posisjonsdata i sanntid med en nøyaktighet ned mot ±1cm. Eksempelvis bruk av GPS kan være som navigasjon til mobile roboter og til fart og akselerasjonsmålinger.

Med bruk av GPS RTK øker man nøyaktigheten drastisk ved å måle vektorer til en *base* som befinner seg i et punkt med kjent posisjon. *Rover* er mottakeren som det skal beregnes posisjon og posisjonsendringen til. *Rover* blir da den mottakeren av GPS RTK systemet som skal bevege seg og skal festes på roboten. Sanntidsmålinger krever kommunikasjon mellom *base* og *rover,* det er ulike kommunikasjonsmetoder som direkte kommunikasjon med kabel, radio, mobiltelefon eller at *basen* laster opp sine måledata via internett.

RTK krever minst fem satellitter med god satellitt geometri. Å få en god satellitt geometri (Dilution of Precision, DOP) kan være utfordrende med tanke på høye fjell og dype daler i Leikanger.
Dersom GPS RTK *basen* har god oversikt over området som *roveren* skal kjøre i og at den har god forbindelse for data-overføring vil nøyaktigheten på posisjon fortsatt være god nokk, spesielt dersom Network-RTK er tatt i bruk. Fordelen med Network-RTK er at med flere *baser* kan man dekke et større område og igjen oppdatere nøyaktig posisjon.

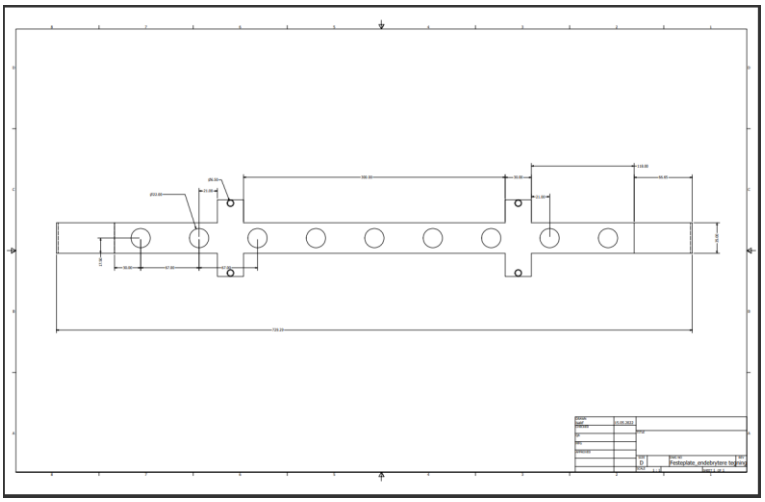
Les meir om DOP-feil (geometrisk feil) og multi-path error i kjeldene under:

Land vehicle positioning using GPS and dead reckoning: https://ur.booksc.eu/book/7143438/d5f590
A single GPS receiver as a Real-Time, Accurate Velocity and Acceleration Sensor: https://www.researchgate.net/publication/242079770_A_Single_GPS_Receiver_as_a_Real-Time_Accurate_Velocity_and_Acceleration_Sensor
Advantages & disadvantages of RTK: https://geodetics.com/ppk-vs-rtk/
DOP: https://snl.no/DOP_-_satellittgeometri

# Appendiks G Frontfanger for sikkerhet

## G.1 Bygging av frontfanger med trykksensor

Prosjektgruppen ble enig om å sette i gang med bygging av design R2 av frontfangeren slik at den kunne bli klar til fysisk testing på roboten i Leikanger. For å kunne bygge frontfangeren måtte det bestilles inn deler (innkjøpslisten kan ses i Appendiks G.3). Bryterne ble bestilt i god tid, men da de kom frem ble det oppdaget at det ikke fulgte med kontaktblokker, noe som er helt nødvendig for å kunne koble og bruke bryterne. Kontaktblokkene ble da bestilt med en gang og i vente på delene ble det tegnet en målsatt mekaniske 2D-tegning. Denne tegningen skulle brukes ved bygging slik at utskjæringen av metallplaten ble så nøyaktig som mulig. For PDF eller DWG fil av tegning se appendiks D.



*Figur 47: Mekanisk 2D-tegning*

For å bygge frontfangeren trengtes det verktøy og en plass å være. En av gruppemedlemmene fikk lov å bruke arbeidsplassens verksted på ettermiddagene. Dette gjorde det mye enklere å utføre det fysiske arbeidet. Det var vanskelig å få tak i en god metallplate som kunne brukes som utgangspunkt for støtfangeren, men fikk leitet frem en montasjeplate i 3mm stål på verkstedet som gikk an å bruke. Ved hjelp av et skyvelære ble alle de ytre målene fra tegningen risset ned i stålplaten, deretter tegnet nøyaktig rundt med en tusj for å enkelt kunne se hvor det skulle skjæres. Det best egnede verktøyet som var tilgjengelig var en batteridreven vinkelsliper med kappskive. Dette gikk helt fint, det tok bare litt tid også var det litt vanskelig å få til nøyaktige utskjæringer der hvor feste til rørklammerne skulle være.

***Figur 48: Metallplate ferdig oppmålt og klar for utskjæring***

Når platen var skjært ut ble den grovfilt rundt hele og finpusset med sandpapir slik at alle skarpe kanter etter utskjæringen var vekke. Deretter ble det målt opp nøyaktig hvor alle hullene skulle være utfra tegningen. De minste hullene på 6mm til feste av rørklammer ble boret, mens de store hullene på 22mm ble stanset ut med en hullstansemaskin som stod på verkstedet. Denne kunne stanse hull på 22.5mm, dette er standard størrelse til slike brytere og gjorde jobben mye enklere.

*Figur 49: Frontfanger ferdig pusset og med hull til brytere*

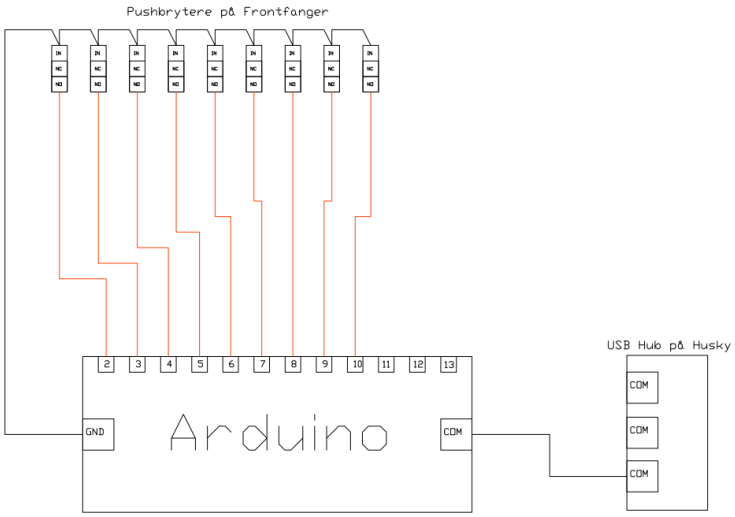
Så langt var det mest utfordrende arbeidet gjort, neste var å montere sammen alle delene. Kontaktblokkene ble klikket på bryterbasen og skrudd på den utskjæret metallplaten. Når rørklammerne skulle monteres på platen kunne man se med en gang at det ikke kom til å fungere som først tenkt. Designet var med noen mindre brytere for å visualisere utseende, og derfor ble det ikke oppdaget at bryterne stakk for langt bak i forhold til rørklammerne. Dette er jo et godt eksempel på hvorfor det er viktig å planlegge og lage et godt design før man bygger. En liten detalj som ikke ble tatt med i designet viste seg å være viktigere enn antatt.

*Figur 50: Bryter stikker for langt bak, og vil stikke enda lengre bak med kontaktblokk.*

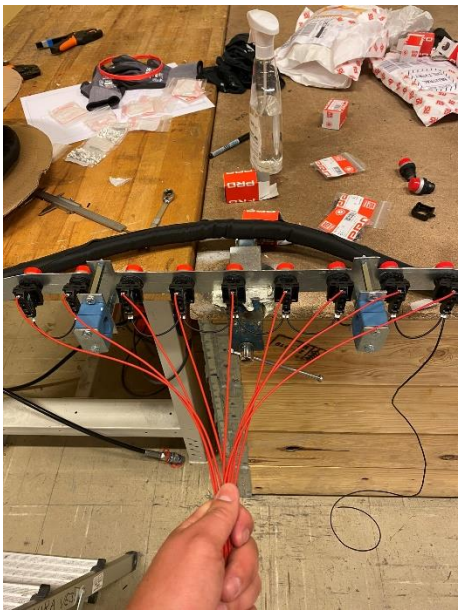
Byggingen av frontfangeren ble som nevnt tidligere gjort på et verksted, men tiden var litt begrenset da sammenstillingen ble gjort i dagene før bachelorgruppen skulle reise til Leikanger. Heldigvis var det en del skruer og bolter tilgjengelig på stedet som kunne brukes. Avstandsbolter på 50mm ble løsningen for å forskyve metallplaten med bryterne litt lengre frem, da ville rørklammerne passe med litt margin.

*Figur 51: Alle brytere med kontaktblokk og rørklammer med avstandsbolter*

Når hele frontfangeren var sammenstilt måtte alle bryterne kobles, de ble koblet med løse ledninger i enden slik at de var klar til å implementeres på huskyen i Leikanger. Bryterne skulle ha en felles COM fra Arduinoen, også skulle hver bryter kobles til en egen digital inngang slik at det var mulig å vite hvilke bryter som ble utløst. Koblingen ble gjort etter forhåndstegnet skjema. For DWG eller PDF fil av skjema se vedlegg.




*Figur 52: Koblingsskjema mellom Arduino, Frontfanger og Husky*



*Figur 53: Ferdig koblet frontfanger fanger med felles COM (Sort) og individuelt signal (rød)*

Frontfangeren ble dekket til for å se litt mer presentabel ut og for å gi litt ekstra beskyttelse mot metallkanter. En kevlar-duk som opprinnelig er for beskyttelse av kabel passet greit og ble stripset rundt metallplaten.

*Figur 54: Frontfanger tildekket og klar til montering og funksjonstest i Leikanger*

## G.2 Implementering av frontfanger på Husky

Mulighetene for montering og testing av frontfangeren på huskyen var begrenset da den ikke var tilgjengelig i Bergen. Tiden i Leikanger før fremvisning ble derfor viktig for funksjonstesting og justeringer. Siden Arduinoen hadde egne tilkoblingspunkt som var laget for noen tilhørende små tilkoblings pinner, måtte disse loddes på ledningene fra bryterne i frontfangeren. Siden Arduinoen skulle kommunisere med huskyen gjennom seriell kommunikasjon, ble det laget et testprogram Vedlegg (Frontfanger_interface_testprogram). Dette programmet skulle teste at signal ble mottatt fra bryterne og sende en string på seriell kommunikasjon som kunne leses ved bruk av seriell overvåkning i Arduinoprogrammet på PC-en. Til å begynne med ville ikke bryterne fungere skikkelig og det måtte en del feilsøking til for å se om det var dårlige eller feile koblinger. Problemet lå i bruken av interne pull-up motstander i Arduinoen. Pull-up motstand er en motstand som trengs for at Arduino skal fungere skikkelig med trykkbrytere. Det kan enten bruke fysiske motstander inn på hver digital inngang, eller det kan brukes interne pull-up motstander i Arduinoen. Ved bruk av eksterne motstander vil det sendes spenning fra Arduinoen gjennom bryteren og inn på inngangen hvor den ledes til GND. Ved bruk av interne pull-up motstander vil de digitale inngangene få tilført spenningen internt fra Arduino kortet via en motstand og ut fra inngangen. Derfor må inngangen ledes til GND gjennom bryteren for å få et signal. Det som er viktig å passe på er at signalet da vil være invertert, siden det ligger spenning på inngangen vil den ligge som True helt til en bryter blir trykket. Da blir bryteren false. Koden var riktig og alle de interne pull up motstandene var aktivert, men felles fra bryterne var koblet til 5.5V og ikke GND slik den skulle. Når dette var koblet om funket alt.

For å stoppe huskyen ved en utløst bryter måtte det publiseres en message ved bruk et eget bibliotek i Arduino som heter Roslib, dette måtte lastes ned og importeres i Arduino programmet. Det begynte å minke på tiden, så det ble bestemt å lage en felles boolsk variabel som ville sende en True om en av bryterne ble presset inn. På denne måten trengte vi bare å sende en felles message over til ROS. Dette ble gjort for å være sikker på at det var tid til å få det ferdig å teste funksjonen før fremvisningen dagen etter. Dette programmet kan ses i vedlegg (Frontfanger_publish_program_final). Optimalt burde alle bryterne sendt hver sin message inn til ROS slik at man kunne vite hvor på huskyen bryteren ble utløst. Dette hadde ikke vært noe problem å få til, men tiden strakk ikke til på slutten.

*Figur 55: Frontfanger ferdig montert på husky og koblet til Arduinoen*

## G.3 Innkjøpsliste

| Type: | Stk: | Link: |
|---|---|---|
| RS PRO Extended Red Push Button Head Spring Return, 22mm | 12 | https://no.rs-online.com/web/p/push-button-heads/1881129 |
| RS PRO Contact Block - SPNO | 12 | https://no.rs-online.com/web/p/contact-blocks-light-blocks/1881170 |
| Rør klammer med festeplate | 2 | https://www.okonomi-deler.no/rorklammer-sveise-og-skruplate-enkel-5 |
| Playknowlogy Uno Rev. 3 Arduino-kompatibelt utviklingskort | 1 | https://www.kjell.com/no/produkter/elektro-og-verktoy/arduino/utviklingskort/playknowlogy-uno-rev.-3-arduino-kompatibelt-utviklingskort-p88860 |
| Playknowlogy Motstandssortiment 600-pk. | 1 | https://www.kjell.com/no/produkter/elektro-og-verktoy/elektronikk/motstander/playknowlogy-motstandssortiment-600-pk.-p90646 |
| Dibotech Loddebolt 30 W | 1 | https://www.kjell.com/no/produkter/elektro-og-verktoy/verktoy/lodding/loddebolter/dibotech-loddebolt-30-w-p40172 |