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.

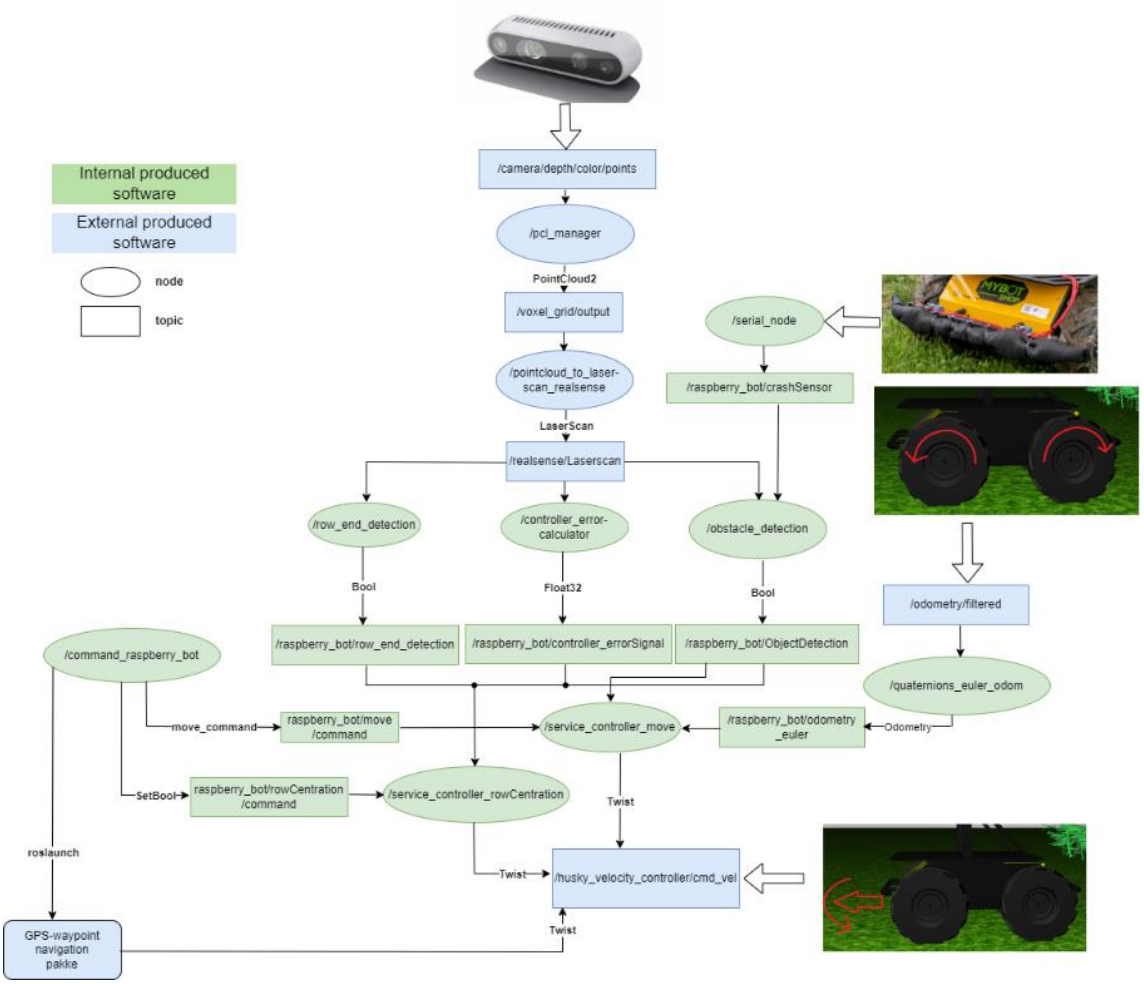
Det er ikke alle topicene i simuleringen som heter det samme som de tilsvarende topicene på den fysiske roboten. Eksempelvis for RGB-D kamera topics så er det de underliggende programmene som enten streamer fra en simulasjon eller fra et fysisk kamera som bestemmer navnene på disse topicene. Det blir 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.

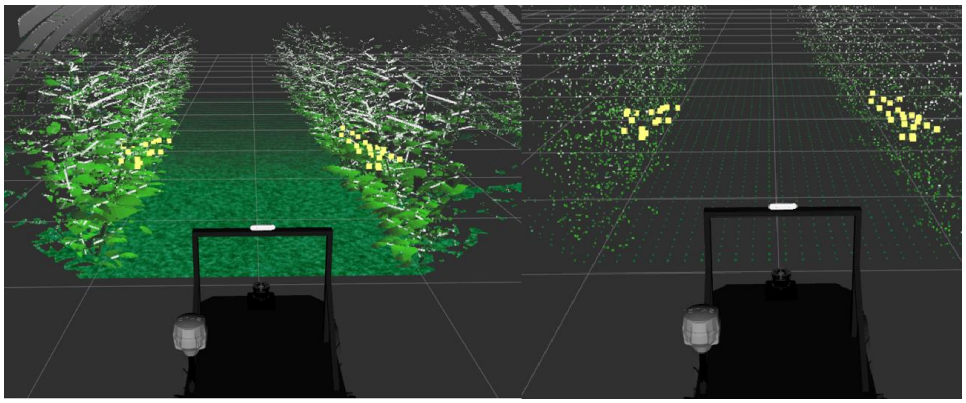
Point cloud to LaserScan realsense

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

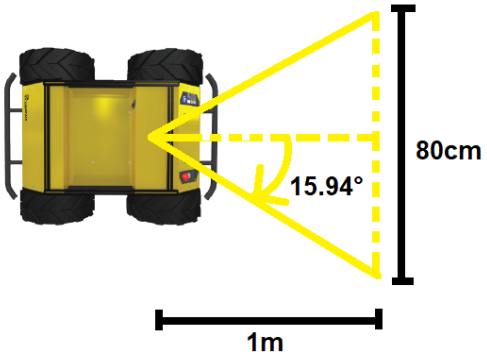
Row end detection

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 π og 2π 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.

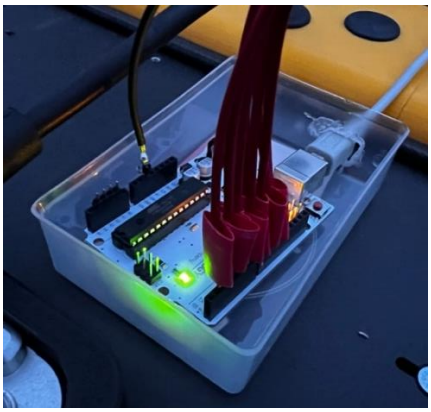
5.2.3 GPS-waypoint navigation

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.