NTNU
Det skapende universitet

Autonom retur og dokking av AVR robot

Lars Marius Strande

Master i industriell kybernetikk
Innlevert: juni 2017
Hovedveileder: Tor Engebret Onshus, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Sammendrag

Batterimonitorering: Det ble satt opp en spenningsdeler mellom batteripakken og mikrokontrolleren sin analog til digitalkonverter slik at batteriets spenning ble nedjustert til den interne spenningsreferansen som benyttes av konverteren. Tester ble utført på batteripakken og karakteristikk ble utarbeidet. Fra disse testene ble det satt en grenseverdi for når AVR-roboten har behov for lading. Det ble satt opp nødvendige funksjoner i kildekoden til roboten og kommunikasjonsprotokollen ble oppdatert til å håndtere endringer i batterinivået via kommunikasjonsmodulen inn til serveren.
Serverapplikasjonen ble oppdatert til å vise robotens batterispenning i brukergrensesnittet i sanntid og nødvendige overgangsbetingelser for å initiere returfunksjonalitet til ladestasjonen når kritisk nivå oppstår.

Returfunksjonalitet: Forbedringer ble utført på den eksisterende returfunksjonen for å få en mer robust løsning som takler forstyrrelser og nye hindringer under retur til ladestasjonen. Den nye løsningen er en oppdatert versjon av løsningen utarbeidet i prosjektet til Strande 2016 [1].

Dokkingfunksjonalitet: Ble løst med en algoritme som sammenligner to sett av punkter registrert av robotens IR-sensorer. Det første settet lagres når roboten forlater ladestasjonen og det andre lagres når roboten kommer tilbake til stasjonen etter retur. Basert på disse to genereres en transformasjonsmatrise som gjør det mulig for roboten å gjenskape en tilnærmet lik posisjon og orientering som hva den hadde når den forlot ladestasjonen. Teoretisk kontroll av transformasjonsmatrisen viser at den utfører korrekte kalkuleringer. Etter fysiske tester viste det seg at løsningen fungerte tilfredstillende men feiler når posisjonsestimatet blir for stort.

Økt nøyaktighet: Tester av sensorer ble utført og resultatene stemmer godt overens med tidligere tester som er utført på AVR-roboten. Det ble utviklet en funksjon slik at roboten kan korrigere posisjonsestimatet ved å bekrefte sin faktiske posisjon. I et forsøk på å gjøre nettopp dette samt å forbedre det generelle posisjonsesti-

matet ble bruk av translasjonsalgoritmen forsøkt implementert i den tradisjonelle kartleggingen også. Slik vil det være mulig å benytte allerede eksisterende funksjonalitet. Testene viste at løsningen reduserer feilen i posisjonsestimatet betraktelig men ulempen er at roboten må returnere til ladestasjonen for å oppdatere posisjonen.

Løsningene som har blitt utviklet i dette prosjektet har blitt grundig testet og dokumenterer god funksjonalitet. Roboten navigerer fortsatt langt fra perfekt men ved å selv kunne oppdage lavt batterinivå, finne tilbake og koble seg til laderen og samtidig oppdatere posisjonesestimatet til en mer korrekt verdi har systemets autonome egenskaper blitt forbedret betraktelig.

Abstract

Battery monitoring: It was crafted a voltage divider between the battery pack and the analog to digital converter(ADC) on the microcontroller in such a manner that it scaled down the voltage to match the internal voltage referance of the ADC. The system and the power supply was tested which the characteristics of the applied battery. From these datas the parameters for when the AVR-robot needed to start the return procedure in order to get back to the charger. The functions needed to solve this was implemented in the source code of the robot and the communication protocol was updated in order to handle the exchange of the battery values between the server and the robot. The server application was uptadetd in order to visualy present the voltage value in the user interface as well as the functionality of identifying when the battery voltage deceeds the predetermined parameter and then initiate the return to the station befor it shuts down.

Return function: Improvements where done to the existing return functionality to make the robot able to handle changes which can occure in the planned path back to the station.

Docking sequence: As a first attempt there was worked out a solution which initiated the server with geometrical values describing the station. This solution seemed to have an issue when calculating the robots translation errors. On the other hand it seemed to solve for the rotation error quite well. Despite this the need of a better and more robust solution was considered obvious. A solution which worked independent of the starting point and also corrected for translation errors. The result was an algorithm which calculated the errors by compairing two sets of measurements done by the IR sensors. Based on these two sets of points the algorithm creates a transformation matrix that enables the robot to recreate a close estimate to the original position, this with regards to both rotation and translation. Theoretical control of the transformation matrix shows that it computes correct. Physical tests where executed and the solution that was produced was sufficient given reasonable error to the position estimate.

Increased precision: The integrated sensors on the robot was tested and the results correspond well with documentation done in previous theses on the robot. In these theses, espesicaly [2], there has been done extensive work by filtering and optimizing the use of these sensors. Based on the work done in this theses it was decided that a different approach would be attempted. To trie to correct the error in the position estimate by using the scan matching algortihm introdused in the previous part. Thus eliminate the rotation and translation error when the robot is mapping. The tests produsedgood results by redusing the error of the position estimate the only draw back is that the robot needs to navigate back to the station in order to confirm its position.

The solutions which have been developed in this theses have been well tested and the results express good preformance. THe robots navigation and mapping is still far from perfect but by introducing abilitis like monitoring and detecting low battery, navigating back and connecting to the charger while updating and correcting the position estimate the conclusion is that the systems autonomical properties have been improved.

Forord

Ved Institutt for Teknisk Kybernetikk (ITK) på Norges Teknisk- Naturfaglige Universitet (NTNU) skrives masteroppgaven i siste semester. Oppgaven tilsvarer 760 - 960 timer og strekker seg over 20 uker. Arbeidet med denne oppgaven har jeg gjort individuelt men jeg har fortsatt arbeidet tett med to andre oppgaver som har blitt skrevet om det samme systemt. På denne måten har samtidig fått erfare hvordan det er å samarbeidet på et komplekst system.

Jeg skrev min fordypningsoppgave om det samme systemet i 2016 og fikk gjennom dette arbeidet satt meg godt inn i hvordan det hele henger sammen. Erfaringene og resultatene jeg satt igjen med fra prosjektet har utgjort et godt grunnlag jeg kunne bygge videre på i masteroppgaven. Det kreves ekstra arbeid å plukke opp et prosjekt som noen andre har utviklet, derfor har jeg forsøkt å forfatte denne rapporten slik at den kan skal være til nytte for videre arbeid med systemets autonome egenskaper.

Takk til Stepfano Bertelli og gutta på verkstedet i D040 for lån av utstyr og komponenter til det jeg trengte å lage. Takk til Kristian Lien, Jørund Amsen og Geir Eikeland for et godt samarbeidet og et bra arbeidsmiljø på kontoret. Og til slutt, takk til Tor Onshus for at jeg fikk arbeidet med dette prosjektet og for god veiledning gjennom prosjekt- og masteroppgave.

Lars Marius Strande
Trondheim, 04.06.2017

Akronym

BLE Bluetooth Low Energy
LIB Litium-Ion Battery
ADC Analog to digital Converter
RMSD Root Mean Square Deviation
SVD Singular Value Decomposition
LSS Least Square Solution
DR Dead-Reckoning
IR Infra red

Figurer

Innhold

1 Introduksjon

1.1 Motivasjon for prosjektet

Autonome roboter tillegges stadig nye roller i samfunnet og krav til funksjonalitet, autonomitet og sikkerhet øker. Autonome roboter har et stort potensiale innen mange felter deriblant kartlegging og søk i katastroferammede områder. For å kunne være til nytte i de ulike situasjonene må robotene kunne operere på egenhånd og håndtere utfordringer som skulle dukke opp i sanntid. Det er når man betrakter dette over et lengre tidsperspektiv at energiproblematikken melder seg, dersom en robot ikke har jevnlig tilgang til energi vil den etter en stund slutte å fungere. Det finnes ulike måter å løse dette på, et eksempel er de ulike roverene som har blitt sendt til Mars [5] som opprettholder sin funksjonalitet ved bruk av solenergi eller SlugBot [6] som etterligner et naturlig vesen og lever av naturen. Et annet viktig moment for en opprettholde autonomitet er god navigasjon. For å navigere en robot på en god måte er vi avhengig av ha et godt estimat på robotens posisjon.

LEGO prosjektet ble startet i 2004 og har gradvis utviklet seg siden det. Fra å starte opp med en robot basert på en 8 bit AVR mikrokontroller som alene kjørte rundt, eksisterer det nå i alt fire roboter som sammen genererer et digitalt kart av omgivelsene i en serverapplikasjon.

Det har underveis blitt tillagt mer funksjonalitet til systemet etterhvert som grunnleggende kartlegging kom på plass. I Haugedal [3] ble det for første gang tatt opp og implementert en form for deteksjon av ladestasjon. Dette ble utbedret og videreutviklet i Kristiansen [7] som også utviklet en løsning for retur. Som følge av store endringer i systemet i ettertid har denne funksjonaliteten falt bort. I Strande sin prosjektoppgave i 2016 [1] ble returfunksjonalitetet igjen utarbeidet og implementert.

1.2 Problemdefinering

Autonome mobile roboter har en begrenset batterikapasitet ombord som avgjør operasjonstiden. Enhver energikilde vil gi et gitt antall enheter med energi før påfylling kreves, i dette tilfellet av strøm fra et ladesystem. En slik gjenopplading krever at roboten avslutter sin aktivitet og kobles til en lader med assistanse fra en operatør. Dette behovet for interaksjon med operatør hindrer autonomitet over lengre tid for den aktuelle roboten.

Systemet er ment å være autonomt og må derfor også kunne håndtere lading av robotene. Denne masteroppgaven bygger videre på fordypningsprosjektet til Strande høsten 2016 [1] "Fjernstyring av autonome roboter" hvor grunnleggende funksjoner ble utviklet. Oppgaven tok for seg to mulig løsninger for å returnere roboten til området rundt ladestasjonen og det ble bestemt at dette best løses ved at serverapplikasjonen kalkulerte den raskeste veien basert på kartet som er generert. I samme rapport [1] ble det avdekket at omfattende feil i posisjonsestimatet setter begrensninger for funksjonaliteten til en slik returfunksjon. Disse oppdagelsene bekreftes i andre rapporter [8], [9], [10]. Derfor vil det også være et mål å redusere denne feilen så mye som mulig. Basert på erfaringer og arbeid fra fordypningsoppgaven [1] og andre oppgaver skal følgende deloppgaver løses:

- Forbedre returfunksjonalitet
- Implementere batterimonitorering
- Utvikle dokkingfunksjonalitet ved ladestasjonen
- Redusere feil på robotens posisjonsestimat

Ved å legge til funksjonalitet i serverapplikasjonen vil løsningen fungere på samtlige roboter. Endringer i hardware blir kun utført på AVR-roboten, derfor vil denne roboten være i hovedfokus i denne rapporten.

1.3 Rapportens oppbygning

Rapporten er skrevet i LaTeX med bruk av hyperreferanser til figurer og kapitler. Rapporten består av i alt sju kapitler og tre hoveddeler. Hvert hoveddel har sitt eget delkapittel med teoridel, beskrivelse av løsning, tester og resultater. Hvert kapittel har en egen diskusjonsdel hvor det blir reflektert over resultatene fra testene som er

utført. Denne disposisjonen er valgt i håp om å gjøre rapporten lettlest og oversiktlig for leseren.

- **Bakgrunn** - I et håp om at leseren lettere skal henge med i rapporten er det en fordel å ha en grundig forståelse for hvordan systemet fungerer. Her vil viktige momenter bli pressentert og gjennomgått i en viss detalj.
- **Batterimonitorering** - Beskrivelse av hvordan batteripakken er satt opp, hvordan det fungerer og hvordan løsningen for batterimonitorering er implementert. Resultatene fra testene utført på batteriet samt av den fullstendige løsning blir presentert.
- **Tilkobling til ladestasjon** - Presenterer oppgavens løsning av dokking til ladestasjonen for lading av batteriet. Det teoretiske grunnlaget for å forstå løsningen blir gjort rede for sammen med oppsettet og implementering av den. I tillegg blir tester og resultatene presentert.
- **Økt nøyaktighet** - Oppgavens forsøk på å forbedre robotens posisjonsestimat ved å benytte funksjonalitet som allerede er implementert.
- **Konklusjon** - Kandidaten er av den oppfatning at enhver rapport burde avrundes med en kort konklusjon som knytter det hele sammen og setter et punktum for oppgaven.

2 Bakgrunn

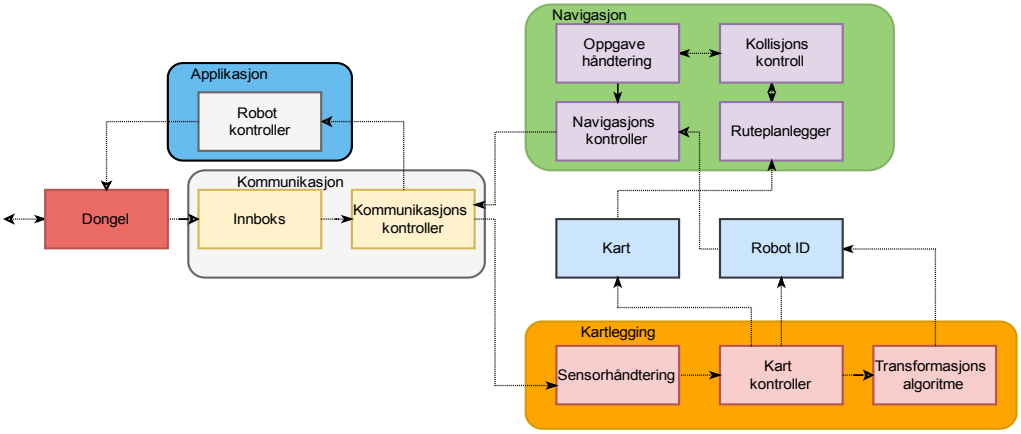
2.1 Prosjektet

Kartlegging av ukjente områder har vært et prosjekttema på NTNU i en årrekke. Prosjektet ble startet opp i 2004 ved avdelingen for Teknisk kybernetikk ved NTNU og planen var å sette opp et system for kartlegging av et ukjent område basert på enkle og rimelige deler.

Våren 2016 ble det bestemt at en ny server skulle settes opp da det ble vurdert til å være mer effektivt enn å jobbe videre med den gamle [8]. Dette resulterte i at mye av arbeidet som var utført tidligere: simulering, navigasjon og kartlegging og er skrevet i Matlab, måtte gjøres på nytt for å passe med den nye serveren som ble skrevet i Java.

Kristian Lien, Jørund Amsen, Henrik Melbø, Kristian Bjørnsen og meg selv har i løpet av våren 2017 arbeidet på prosjektet.

- **NXT og kommunikasjonsprotokoll** - Kristian Lien
- **Forbedringer av Arduino** - Jørund Amsen
- **Kartgenerering i serveren** - Henrik Melbø
- **Kartlegging av labyrint med kamera og Rasberry Pi** - Kristian Bjørnsen
- **Dokking og batterimonitorering AVR** - Lars Marius Strande



Figur 1: Informasjonsflyt for kartlegging og navigasjon i serveren.

2.2 Systemet

Systemet består i dag av fire roboter utstyrt med infrarøde sensorer for registrere avstander til hindringene den møter. Robotene kommuniserer med en serverapplikasjon på en PC og sender meldinger som beskriver omgivelsene rundt seg etterhvert som de navigerer seg gjennom det ukjent området. Serverapplikasjonen vil basert på denne dataen generere et kart. Basert på dette kartet vil serverapplikasjonen sende bevegelseskommandoer tilbake til robotene mot områder av interesse.

2.2.1 Serveren

Serverapplikasjonen er programert i Java og prosjektet er satt opp i Netbeans IDE 8.2. Fundamentet ble satt opp av Andersen, Rødseth og Thoen i 2016 [11] [8]. Java er et klasse- og objektorientert programmeringsspråk som er godt egnet for omfattende applikasjoner som denne. Serveren består av mange ulike klasser og objekter med ulike roller for helheten i systemet.

For å forstå den grunnleggende funksjonaliteten har illustrasjonen av informasjonsflyten blitt satt opp i figur 1. Denne viser hvordan kommunikasjon for navigasjonen basert på det genererte kartet fungerer. Her kan man se et lite utdrag fra modulene

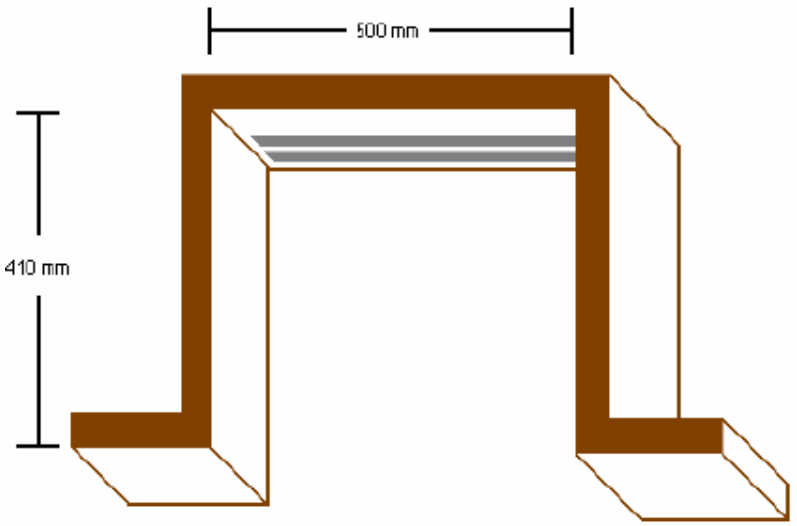
kommunikasjon, applikasjon, navigasjon og kartlegging. Serveren mottar data fra IR-sensorene, denne dataen inneholder avstand og vinkel relativt til roboten. Informasjonen håndteres i sensorhåndering-klassen hvor den blir sortert og satt inn i lister for hver robot og hvilke sensor som ser hva. Listene blir brukt i kartkontrolleren som plasserer nye hindringen på korrekt plass i kartet basert på robotens estimerte posisjon. Navigasjonssekvensen benytter seg av dette kartet for å planlegge neste mål for robotens videre ferd. Posisjonen vurderes opp mot potensielle hindringer i kollisjonskontrollen og dersom den blir godkjent blir oppgaven satt. Navigasjonskontrolleren gir beskjeden videre tilbake til kommunikasjonsmodulen og tilbake til roboten via dongelen. Roboten og datamaskinen kommuniserer ved hjelp av Bluetooth 4 og det er plassert en nRF51 Dongle hos begge parter.

2.2.2 AVR

Roboten som har blitt jobbet med i denne oppgaven er en AVR 8 bit. Den er ustyrt med tre hjul hvor de to framre har framdrift og bakhjulet er en passiv kule. Roboten driftes av en 11.1 Volts Li-ionbatteripakke som gir en effektiv teoretisk operasjonstid på ca 30 timer. Roboten kontrolleres av en ATMega 1248p mikrokontroller med 16KB RAM og er programmert i C ved bruk av Atmel Studio 7.0. Roboten styres av tasker satt opp ved bruk av freeRTOS [2]. Et slikt oppsett gir roboten muligheten til å multitaske og setter opp taskene etter prioritet.

Roboten navigeres ved bruk av odometri og dead-reckoning (DR). Odometri er den mest brukte metoden for navigasjon av mobile roboter. Metoden gir god nøyaktighet en kort periode til lave kostnader og med høy oppdateringsrate. I AVR-roboten bestemmes avstanden den har kjørt basert på enkoder tikk og rotasjonen blir håndtert av fusjonert data mellom gyro og kompasset.

Roboten er utstyret med fire IR-sensorer som roterer på et periskop montert på toppen av roboten. Som vist i figur 18. Sensorene er plassert med 90° intervall, resultatet av denne fordelingen er at en hel rotasjon effektivt blir skannet etter en rotasjon på 90°. På denne måten elimineres kluss med ledninger og tårnet kan roteres uhindret. Roboten rapporterer sin posisjon samt avstander til hindringer rundt seg til PCen og server applikasjonen som bruker denne informasjonen til å generere et kart av omgivelsene. I dette kartet blir også roboten plassert. Etterhvert som den beveger seg gjennom det ukjente området stopper den når den har utført de mottatte meldingene. Den registrerer da omgivelsene og venter på en ny kommando. Denne prosessen repeteres helt til roboten og serveren sammen har kartlegger hele området den har adgang til.



Figur 2: Originalskisse av ladestasjonen fra prosjektoppgave [3].

2.2.3 Stasjon og ladeløsning

Ladestasjonen ble satt opp slik den er nå i prosjektoppgave av Haugedal i 2008 [3] og er gjort som en forbedring av den første ladestasjonen som ble bygd året før av Magnusson. Hovedfunksjonen til ladestasjonen er å være et bindeledd mellom laderen og batteripakken som sitter på roboten. I figur 2 vises den opprinnelige skissen av ladestasjonen fra 2008 av Haugedal. Skissen viser dimensjonene og hvordan det er plassert to ladeskinner på bakveggen av stasjonen. Tanken bak dette var at disse skinnene ville akseptere et større område for roboten å returnere til og at det ikke vil kunne komme et objekt som hindrer roboten fra å koble seg til.

Figur 3: Ladeoppsett på roboten.

Motparten til skinnene er en enkel fysisk implementering på baksiden av roboten og består av fire fjærbelastede konktakter med lav fjærkonstant for lang vandring. Oppsettet er symmetrisk og vil også øke feilen som kan aksepteres for dokking. Oppsettet er illustrert i figur 3

2.3 Kjente mangler

For å identifisere mangler i systemet blir det lagt ved en liste over utfodringer og svakheter i systemet. Manglene er basert egne erfaringer og møter med andre masterstudenter.

- Unøyaktighet i posisjonsestimat
- Ustabilitet i eksisterende returfunksjon
- Feilmelding fra serverapplikasjon etter kollisjon ved retur
- Mangler i kollisjonsdeteksjon mellom robotene

3 Batterimonitorering

En viktig del av å forbedre systemets autonome egenskaper vil være å detektere når roboten har behov for lading. Ved å gjøre dette vil systemet kunne operere over lengre tid uten behov for interaksjon fra en operatør. Kapittelet vil presentere aktuell teori, den planlagte løsningen samt tester og resultater.

3.1 Teori

3.1.1 Analog til digitalkonverter

I systemer hvor digitale og analoge verdier brukes om hverandre må man på enkel og nøyaktig måte kunne kommunisere mellom de to signaltypene. Poenget er at konvertereren tar et analogt signal, i form av en spenning eller strøm og gir det en tilhørende digital verdi. På denne måten kan for eksempel en datamaskin prosessere informasjon generert av et analgot måleinstrument.

Sammenhengen mellom den analoge og digitale verdien avhenger av oppløsning og aktuelle spenninger. Sammenhengen er som følger:

$$\frac{ADC \quad Resolution}{System \quad current} = \frac{Digital \quad value}{Analog \quad current} \tag{1}$$

3.1.2 Batteri og energiteknologi

Batterier selges i dag i med en rekke ulike spesifikke egenskaper som “Long Life”, “High Capacity”, “High Energy”, “Deep Cycle”, “Quick Charge” og lignende kryptiske betegnelser. Det finnes svært få industrielle eller lovfestede standarder som definerer disse betegnelsene. Dersom man ser bort ifra batteriets design kan virkningsgraden være avhengig av hvordan batteriet brukes samt eksterne påvirkninger. Batteriprodusentene unngår i stor grad disse vage betegnelsene og spesifikasjonene forklares ved å sette opp begrensinger for hvilke eksterne parametre batteriet leverer de angitte verdiene.

Under følger nøkkelparametre for å forstå hvordan batteriet fungerer og hvordan man kan si noe sikkert om effekten man kan forvente fra batteriet.

Krav til batterikapasitet

Elektrisk ladning måles i antall elektroner som er lagret i batteriet. Vi vet at ladning er gitt i Coulomb og at et elektron består av $1.602e^{-19}$ [C]. Én Amper som strømmer gjennom en ledning i et sekund vil bruke en [C] ladning, dette tilsvarer $6.24 * 10^{18}$ elektroner.

For å kunne beregne ladning og elektrisk strøm er det av interesse å se på forholdet mellom ampere og coulomb over tid. Hvor elektrisk strøm, I er gitt av ladningen Q per tidsenent t, som gir oss likningen:

$$I = \frac{dQ}{dt} \tag{2}$$

Mengden elektrisk ladning som beveger seg gjennom en leder i løpet av tiden t_0 og t kan forklares ved å integrere opp på begge sider:

$$Q(t) = Q(t_0) + \int_{t_0}^{t} I(t)dt \tag{3}$$

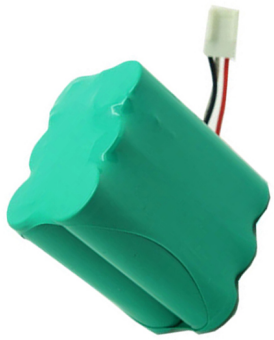
Uttrykket ved konsant strøm er enkelt gitt ved:

$$Q = It \tag{4}$$

Videre følger at man i løpet av 60 sekunder tilsvarer 60 [C] og at man da i løpet av 1 time vil ha sett 3600 [C] passere. For å forenkle arbeidet med mengden ladning som er lagret i batteriet ble dermed SI-enheten ampertimer opprettet og gir at 1 Ah = 3600 [C]. Men siden batteriet endrer spenning når det blir brukt er ikke dette en optimal måte å beregne mengden lagret energi i batteriet. For å si noe om energien ønsker man å finne wattimer. Ved å multiplisere gjennomsnittet eller nominell spenning over batteriet med batteriets kapasitet [Ah] får man et estimat på hvor mange wattimer batteriet innehar.

$$E = CV_{avg} \tag{5}$$

Hvor E tilsvarer energien i wattimer, C tilsvarer batteriets kapasitet i ampertimer og V_{avg} tilsvarer gjennomsnittlig spenning i volt under last.



Figur 4: Batteripakken på AVR-roboten.

3.1.3 Cutoff-spenning

I mange år var nikkel-kadmiumbatteri den mest brukte batteriteknologien før det gradvis ble utkonkurrert av den Li-Ion teknologien på 1990-tallet. Li-Ionbatterier (LIB) ble med det den mest brukte batteriteknologien i bærbar elektronikk. Litium er det letteste kjente metallet, har det høyeste elektrokjemiske potensialet samt størst energitetthet per vektenhet.

Elektronikk med Li-Ion batteri skrus av idet batteriet når 3.0 $\frac{volt}{celle}$, ved dette tidspunktet har batteriet omlag 5 % batterikapasitet igjen. Dette gjøres for at batteriet skal tåle selvutladning dersom batteriet ikke blir gjenoppladet umiddelbart. Et Li-Ionbatteri skal kunne håndtere å være "utladet" i flere måneder før det når et punkt hvor det ikke kan gjenopplades [12]. Dette skjer dersom batteriet havner under $2.5\frac{volt}{celle}$.

3.2 Batteripakken

Kjente parametre er lagt ved i tabell 1. For å kunne si noe om batterikapasiteten ble det utført en test hvor strømforbruket ble logget. Robotens handlinger deles inn

Tabell 1: Teoretiske batteriparametre

Type	Litium Ion
StartSpenning	11.1 [V]
Kapasitet	5200 [mAh
Ladetid	2 - 3 timer
Antatt Cutoff	9 [V]

i tre kategorier: standby, kartlegging med bevegelse og stillstående kartlegging.

Standby	0.06 [A]
Kartlegging og bevegelse	0.16 [A]
Stillestående kartlegging	0.12 [A]

Basert på målingene utført kan man si noe om antatt operasjonstid roboten har med det kjente batteriet. Ved å utnytte likningene for antall wattimer i et batteri 5. Dette tilsvarer for en batteripakke med tre batterier 1.776 wattimer.

$$\begin{aligned} E &= CV_{avg} \\ &= 0.16[A] \times 3.7[V] \\ &= 0.592[Wh] \end{aligned} \tag{6}$$

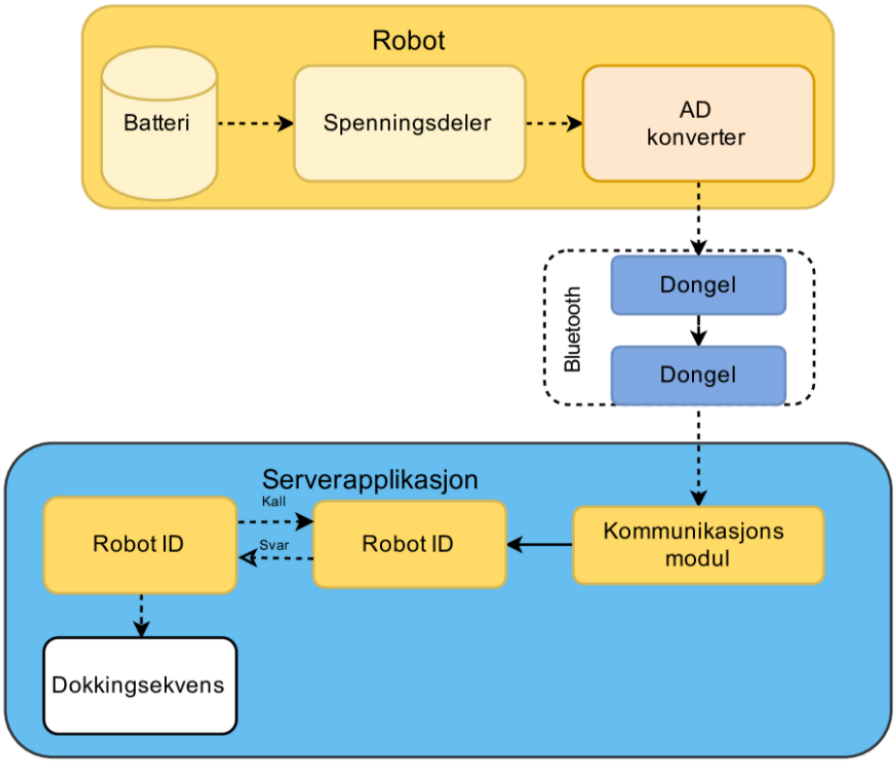
Batteriet kapasitet over tid dersom man antar maksimalt forbruk i roboten blir gitt ved 7. Legg merke til at strømmen som trekkes tilsvarer forbruket + tapet over spenningsdeleren. (160 + 10) mA.

$$\begin{aligned} C &= It \\ t &= \frac{C}{I} \\ t &= \frac{5200[mAh]}{170[mA]} \times 0.75 \\ t &= 22.9[h] \end{aligned} \tag{7}$$

3.3 Plan for løsning

For å effektiv kunne monitorere hvordan energinivået endrer seg ettersom roboten kartlegger området ble følgende plan satt opp. Spenningsnivået må hentes ut før den reguleres ned til 5 [V], dette fordi oppsettet er konstruert slik at det alltid vil levere en jevn spenning til mikrokontrolleren og dette nivået vil da alltid holdes lik 5 [V].

Som vist i flytskjema i figur 5 kan man se hvordan kommunikasjonen går fra batteriet og helt inn til robot.java i serverapplikasjonen. Hvor målingen fra batteriet vil bli lagret. Dette er mulig ved å benytte den integrerte ADCen mikrokontrolleren er utstyrt med. Basert på disse verdien kan man sammenligne den aktuelle spenningen opp mot en forhåndsbestemt vardi for når roboten må returnere. På dette tidspunktet vil retur kommandoen sende på samme måte som den i Strande i 2016 ble aktivert av en knapp i brukergrensesnittet.



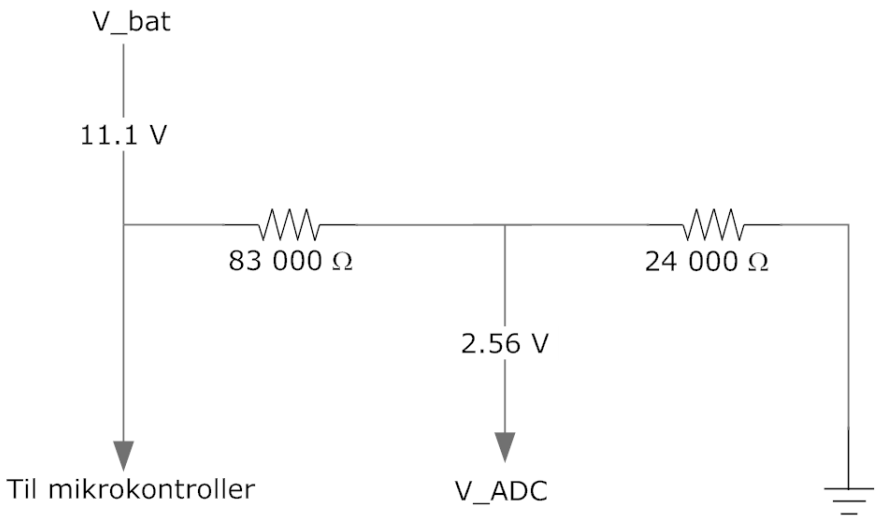
Figur 5: Flytskjema av batterimonitorering.

Spenningsdeler

En spenningsdelerer har som oppgave å reduserer inngangspenningen ned til et lavere nivå. Et enkelt eksempel er to motsander koblet i serie med en inngangspenning i den ene enden, en utgangspenning som forlater kretsen mellom motstandene og jording helt til slutt. Se figur 6. Valg av motstander er tatt basert på ønsket utgangspenning. Batteripakken til AVRen vil ha en nominell spenning på 11,1 Volt, til tross for dette kan et fulladet batteri levere hele 12,6 Volt. Dette skyldes overflatebehandling og tilsetningsstoffer av elektrolytter i batteriet som gjør det mulig å øke den nominelle spenningen. Dette til tross benyttes 11.1 volt videre i oppgaven.

$$V_{ADC} = \frac{R_2}{R_1 + R_2} V_{bat} \tag{8}$$

Størrelsen på motatandene ble avgjort ved å bruke ligning 8. Dersom man velger resistanser med høy motstand vil man redusere mengden strøm som trekkes fra batteriet men dette vil også påvirke nøyaktigheten på spenningsdeler. I dette tilfellet vil det være lite gunstig å trekke store mengder strøm fra batteriet, samtidig som kravet til nøyaktigheten ikke er avgjørende for funksjonen. Det ble først satt sammen en spen-



Figur 6: Koplingsskjema for spenningsdeleren.

ningsregulator av mindre motstander enn planlagt, da det ikke var tilgang til ønsket størrelse. Ved testing av denne spenningsdeleren kunne man tydelig se hvordan batteriet hurtig mistet strøm som følge av høy lekkasje gjennom denne. Sammenhengen mellom teori og praksis kom tydelig fram. Strømtapet gjennom spenningsdeleren ble bestamt av ligning 9.

$$i = \frac{V_{inn}}{R_1 + R_2} \tag{9}$$

Etterhvert som motstander av korrekt størrelse ble anskaffet ble en bedre spenningsdeler satt sammen. Det benyttes nå motsanter på 83 og 24 kOhm. Tapet over spenningsdeleren er markant redusert og er nå på 10 mA.

3.4 Implementering

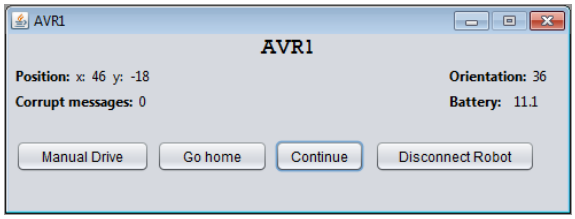
På roboten er det opprettet en ny meldingsprotokoll for kommunikasjon av batteriets spenningsnivå. Protokollen har også blitt satt opp på serversiden og blir kommunisert inn til det respektive robotobjektet. Kommunikasjonen er som vist i figur 5. Et utklipp fra robotens brukergrensesnitt er inkludert i figur 7 hvor spenningsnivået til roboten vises i volt. Oppdateringen skjer i sanntid og man kan til enhver tid gå inn i serveren og undersøke spenningen på roboten.

Det vil fortsatt være mulig å avgjøre om varslingen for lavt batterinivå skal tas manuelt som i [1], uavhengig av batteriets spenningsverdi eller automatisk som en

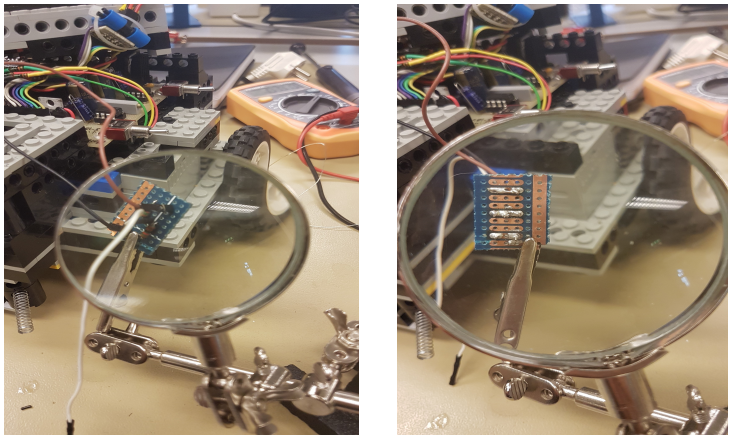
direkte respons til batteriets aktuelle verdi. Å beholde knappen som aktiverer retur har også en viktig rolle for videre testing av funksjonalitet.

Når batteriets nivå faller til et forhåndsdefinert kritisk nivå, vil returkommandoen aktiveres og roboten vil returnere til ladestasjonen som er beskrevet i Strande 2016 [1]. Etter dette er roboten nødt til å koble seg til ladestasjonen, se kapittel 4 for å utføre selve ladingen. Ved å introdusere en ny grenseparameter for batterinivået er det mulig å kontrollere at roboten mottar ønsket lading før den gjenopptar kartleggingen med fult batteri.

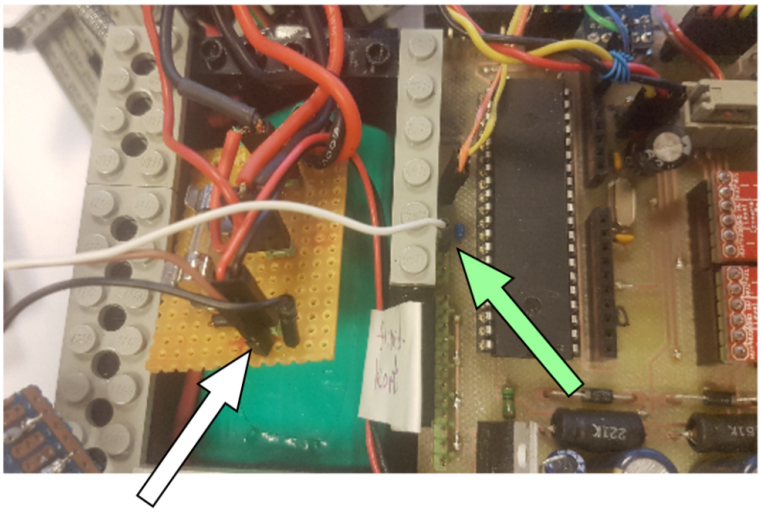
På robotsiden må den aktuelle ADC-pinnen omprogrammeres og settes til å motta spenningen fra batteriet. Til nå har tre av ADC-pinnene blitt brukt som utgangsignal for LED-lampene som har blitt brukt for debugging, det var naturlig å bytte ut en av disse. Ledningsoppsettet har blitt satt opp så det enkelt kan endres tilbake dersom man trenger alle tre lampene til debugging.



Figur 7: Utklipp fra serverapplikasjonen for den aktuelle roboten.



Figur 8: Spenningsdeleren som ble laget.



Figur 9: Tilkobling av spenndingsdeler og monitoreringssignal

Bygging av spenningsdeleren ble utført på verkstedet på D040 ved NTNU. Det var viktig å få den så liten som mulig slik at den passet inn i roboten. Valg av motstander er nevnt og begrunnet i 3.3. Dokumentasjon fra verkstedet vises i figur 8. Den fysiske tilkoblingen vises i figur 9 hvor spenningen hentes ut fra batteriet som vist med den hvite pilen. Her sitter en pin som kobles til spenningsdeleren med en hunnkobling som enkelt kan trekkes ut dersom det skulle være behov for å demontere monitoreringen. Den samme løsningen benyttes på ADC-inngangen som vises med grønn pil.

3.5 Tester

For å bestemme den eksakte grenseverdien for når roboten må starte på returferden, må batteriet testes grundig. Den teoretiske cutoff-spenningen er satt fra produsenten til å skje ved 9 volt, som vist i tabell 1. For å finne den faktiske spenningen på det aktuelle batteriet ble voltmeteret koblet til og roboten ble startet opp. Rett før den skrudde seg av ble spenningen over batteriet målt til 9.61 volt, atlså 14 % høyere enn den teoretiske cutoff-spenningen. Dette er nå en nøkkelverdi for resten av arbeidet og er punktet man ønsker å unngå for å forbedre systemets autonomitet. Det må bestemmes en grenseverdi for spenningsnivået som med god nok margin før vi når dette punktet. For å si noe om den faktiske operasjonstid ble voltmeteret nok en gang satt over batteriet men denne gangen var batteriet koblet fra roboten og var i stedet koblet til en motstand som trakk 0.17 amper. Dette var en forenkling av testen slik at det ikke var behov for å kjøre roboten under hele forsøket. Testen gav batterikarakteristikk som presenteres i figur 11 og viser en reell operasjonstid på i

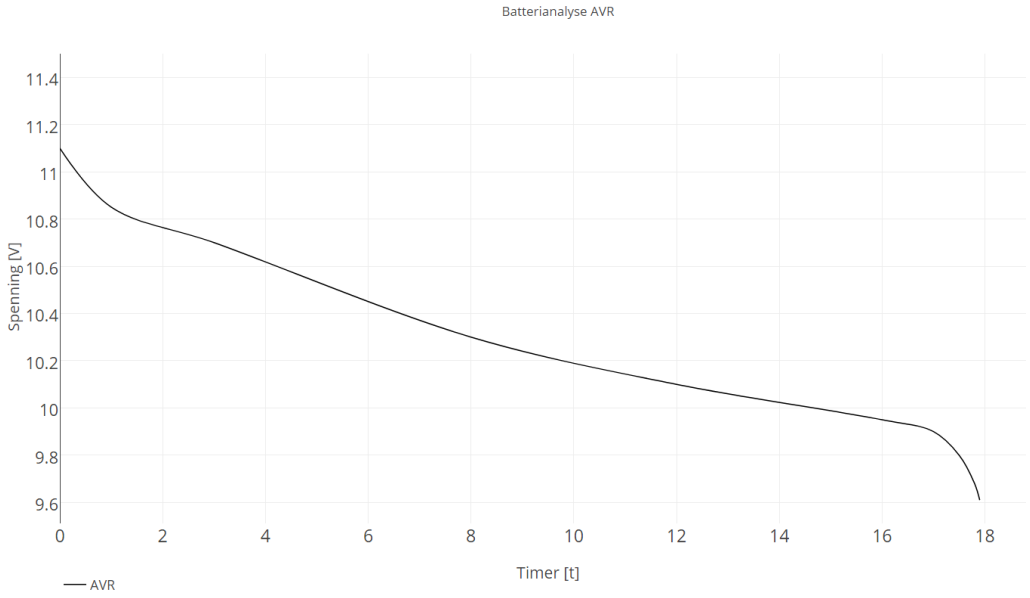
Figur 10: Test av batteriets cutoff-spenning .

overkant 17 timer. Dette tilsvarer 77 % av den teoretiske verdien. Karakteristikken viser hvordan batteriet tappes mest i starten og at det gjevnes ut etter hvert. Etter ca 17 timer synker batterispenningen relativt fort ned mot cutoff-spenningen. På dette tidspunktet er det kun interessant å teste at batterinivået blir kommunisert og at deteksjonen og varslingen blir trigget ved verdier under grenseverdien. For å teste dette på en effektiv måte ble batteriet tappet helt for strøm, for så å bli ladet litt opp slik at testtiden ble redusert. Fra analyse av batteriKarakteristikken ble det valgt en grenseverdien på 9.9 volt. Dette burde gi roboten god margin til å komme seg tilbake til ladestasjonen før den stopper. Denne verdien ble lagt inn i serverapplikasjonen og systemet ble satt igang. Funksjonstesten viste at systemet fungerte som ønsket. Dokumentasjon av testen finnes i vedlagt DVD, se vedlegg A.

3.6 Diskusjon

Spenningsdeleren ble satt inn for å nedjustere batterispenningen til referansespenningen til mikrokontrolleren. Det ble testet ut en rekke ulike sammensetninger av motstander i spenningsdeleren for å oppnå et nøyaktig resultat uten å trekke for mye strøm fra batteriet. Dette er den avgjørende balansen for valg av motstand og det ble vurdert dit hen at nøyaktigheten til monitoreringen kommer i andre rekke. Dette da hovedfunksjonen til batteriavlesningen er å varsle systemet når nivået faller under den kritiske verdien. Unøyaktigheter kan derfor tas høyde for når den kritiske grensen velges slik at systemet håndterer den største mulig feilen. Avgjørelsen falt derfor på å bruke store motstander.

Det eksisterende pin-oppsettet på mikrokontrolleren benyttet seg av alle ADC-



Figur 11: Illustrasjon av batterikarakterestikk

inngangene. Denne våren var vi to som arbeidet med AVR-roboten og det ble etter samtale oss imellom bestemt at det var den gule led-lampen som spilte minst rolle. Derfor ble pin 6 først benyttet. Det viste seg at denne inngangen gav unormale verdier. Årsaken til dette ble ikke avklart til tross for feilsøking på mikrokontrolleren og kode. På bakgrunn av dette ble pin 7 satt opp som inngang for batteriet og resulterte i korrekte verdier. Da det viste seg at pin 6 fint kunne brukes som utgang for led-lamper ble den røde dioden koblet hit for ikke å miste funksjonalitet.

I kildekoden til roboten måtte den resulterende digitale verdien håndteres på en god måte. Siden det gikk på bekostning av nøyaktigheten i spenningsregulatoren vil verdien kunne variere noe. I et forsøk på å redusere dette blir det samle opp flere verdier hvor gjennomsnittet av dem blir sendt videre til serverapplikasjonen. Protokollen for kommunikasjon med serverapplikasjonen ble utviklet for å håndtere den aktuelle verdien på samme måte som meldingene om robotens posisjon og punkter blir kommunisert. På denne måten vil det bli en grei sak å håndtere meldingen i innboksen i serverapplikasjonen.

4 Tilkobling til ladestasjon

Implementering av funksjonalitet som tillater roboten å koble seg til ladestasjonen uten innblanding av noe ekstern forstyrrelse er avgjørende for et autonomt system.

I dette kapittelet vil teorien den aktuelle løsningen baserer seg på bli pressentert. For å redusere uhensiktsmessig lange matematiske bevis vil teorien legges frem så enkelt som mulig, med referanse til aktuelle matematiske beviset. Det teoretiske grunnlaget som legges til grunne i dette kapittelet vil også brukes i kapittelet for økt nøyaktighet

4.1 Teori

4.1.1 Singulærverdi dekomposisjon (SVD)

Er en form for numerisk betraktning mye utbredt i teoretiske studier men benyttes også i praktiske anvendelser. En singulærverdi dekomposisjon til en matrise A er en av de viktiste faktoriseringene av matrisen og inneholder viktig informasjon om den og de eksisterende fundamentale underrommene som for eksempel nullrommet.

SVD kan brukes for å uttrykke en eksplisitt representasjon av rang og nullrommet for en matrise A Formell definisjon hentet fra Cornell. La m og n være vilkårlige verdier. Gitt $A \in \mathbb{C}^{m \times m}$ er da en SVD av A gitt ved

$$A = VSW^T \tag{10}$$

hvor

$V \in \mathbb{C}^{m \times m}$ er en unitær matrise,
$W \in \mathbb{C}^{n \times n}$ er en unitær matrise,
$S \in \mathbb{R}^{m \times n}$ er en diagonal matrise.

De diagonale verdien σ_i av matrisen S er kjent som de singulære verdiene til A [13]. Anvendelsen i algoritmen som er aktuell for denne rapporten er for å beregne optimal rotasjon mellom to sett av punkter.

4.1.2 Minste kvadraters metode (LS)

Kan beskrives som en måte å estimere og finne sammenhengene mellom en eller flere variabler samtidig som den skal minimere variansen. Variansen er kjent som kvadratet av aviket mellom det observerte og det estimerte. Dette er bakgrunnen for navnet, minste kvadraters metode.

4.1.3 LS av bevegelse ved bruk av SVD

Gitt to sett av punkter som er satt opp i sorter rekkefølge $P = \{p_1, p_2, ...p_n\}$ og $Q = \{q_1, q_2, ..., q_n\}$ kan man finne den optimale kombinasjonen basert på minste kvadraters metode. Den aktuelle algoritmen som benyttes til dokkingsekvensen minimerer uttrykket:

$$(R, \mathbf{t}) = \underset{R \in SO(d), t \in R^d}{argmin} \sum_{i=1}^{n} w_i \|(R\mathbf{p}_i + \mathbf{t}) - \mathbf{q}_i\|^2 \tag{11}$$

Hvor R representerer rotasjonsmatrisen og $\mathbf{t}$ representerer translasjonen. Videre utledning kan leses i [14].

4.2 Retur

Til tross for tilfredstillende resultater fra den eksisterende returfunksjonaliteten som ble utviklet og dokumentert i tidligere prosjektrapport av Strande [1] ble det bestemet at denne skulle forbedres og videreutvikles. Årsaken til dette var at den ikke kunne håndtere endringer i kartet på en tilstrekkelig måte, og den gav feilmeldinger ved små forstyrrelser i kartet.

4.2.1 Restrukturering i serverapplikasjon

For å redusere unødvendig kompleksiteten og samtidig øke funksjonaliteten av løsningen presentert i prosjektrapport [1] ble implementeringen revurdert og endret. Dersom man tar for seg RobotTaskManager.java kan man med økt stabilitet[1] gi roboten en bedre funksjonalitet enn den tidligere løsningen. Psaudokode av det nye oppsettet er inkludert i under.

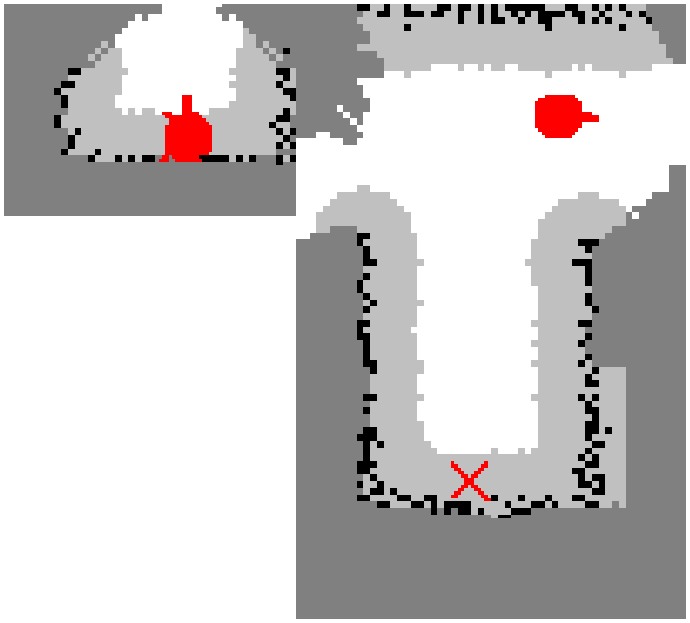
[1] Stabilitet - hvor ofte applikasjonen termineres med feilmelding

Listing 4.1: Restrukturering av returfunksjon.

```
%\begin{lstlisting}[language=Java, caption = Restrukturering av
    returfunksjon., label = returny, basicstyle = \small]
Utdrag fra pathPlanningFunction
@Override public void run():
// Runs the A* algorthim for return...
if(robot is heading home but not there yet){
// Position robot
    find robot position
    put robot position in map
// Position base
    find base position
    put base position in map
// Find path home
    utilize PathPlanningFunction from current pose to base
    if (list of way home empty){
        set robot @base
        robot is not assigned
        break while
    }
    current target set to base location
    get next position from path home list
    set robot destination to next position
    robot is assigned
    // Run algorithm again if stuck
    if(robot is stuck){
        remove stuck flag
        break while
    }
}
```

Med den nye implementasjonen kan serverapplikasjonene nå håndtere kollisjoner under returen til ladestasjonen. Dette ble ikke løst på en tilstrekkelig måte tidligere. Dette blir gjort ved å legge til kodelinjer i den eksisterende tasken CollisionManager.

Sentralt i funksjonen er roboten, med kritisk informasjon som brukes av RobotController, CollisionManager og RobotTaskManager. RobotController holder orden på hvorvidt roboten er tilegnet en oppgave eller ikke, dette settes i den boolske variablen "assigned" i robot. Videre kommuniseres informasjon om hvorvidt roboten har fått tildelt en oppgave men ikke klarer å utføre den, og på en eller annen måte har



Figur 12: Dokumentasjon av feil ved retur.

satt seg fast. Dersom dette skjer vil stuck flagget bryte whileløkken og starte tasken på nytt. Når dette skjer vil en ny rute bli kalkulert av returalgoritmen.

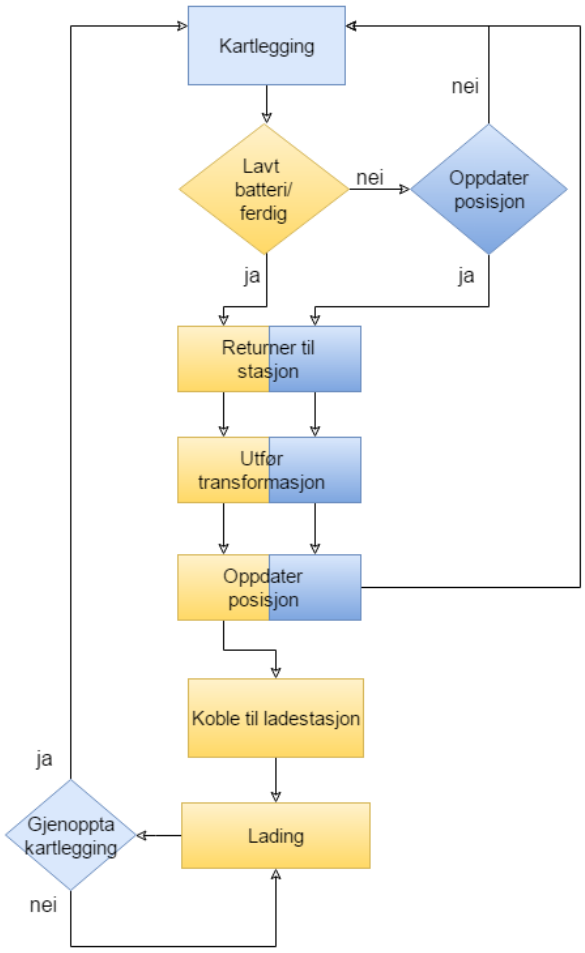
Roboten vil nå også returnere koble seg til ladestasjonen når den er ferdig med å kartlegge labyrinten.

4.3 Dokkingsekvens

På grunn av overflow i kommunikasjonen ved bruk av millimeter har roboten kun presisjon ned på [cm] nivå. Dette gir en reduksjon i den sårt trengte nøyaktigheten ved dokkingsekvensen. Posisjonsestimeringen i roboten jobber i [mm] men det blir oppskalert til [cm] før det kommuniseres over til serveren [2].

For å sikre en tilfredstillende tilkobling til ladestasjonen må følgende utfordringer håndteres.

- Sikre retur tilbake til ladestasjonen.
- Identifisere avvik i robotens orientasjon.
- Identifisere avvik i robotens posisjon.
- Oppnå kontakt mellom ladeskinnen og roboten.



Figur 13: Oversikt over den fulle sekvensen ved returnering.

– Tillate roboten å fortsette kartlegging.

Kravet som ble satt til robotens retur til ladestasjonen fra prosjekt [1] var at den skulle ende opp i nærheten av ladestasjonen. Returpunktet var i [1] definert som punktet hvor roboten startet kartleggingen. Når roboten skal returnere vil kollisjonsdetekteringen kunne nekte roboten å utføre den planlagte ruten tilbake, se dokumentasjon i figur 12. I denne figuren ser man startposisjonen inne i ladestasjonen. I tillegg kan man se det genererte kartet i det roboten forsøker å returnere og man kan tydelig se hvordan returpunktet befinner seg inne i det begrensede[2] området. Dette er opprinnelsen for første punkt i problemdefineringen 4.3.

I figur 13 vises helheten av sekvensen som returnerer roboten. Figuren består av en

[2]Når et punkt blir oppdaget og merket i det genererte kartet blir et bestemt område rundt punktet også merket som utigjengelig.

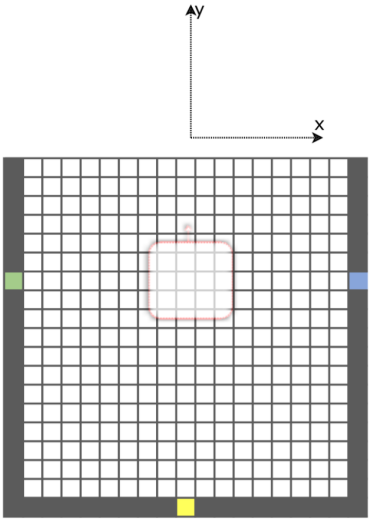
prematur introduksjon til posisjonsoppdatering som blir introdusert senere i oppgaven. Illustrasjonen viser også hvordan roboten returnerer til ladestasjonen når kartleggingen er fullført. Systemet har også muligheten til å gjenoppta kartleggingen når ladingen er ferdig.

Det ble bestemt at dokkingen skulle løses ved å arbeide i serverapplikasjonen. Grunnlaget for denne avgjørelsen er at funksjonaliteten vil fungere på alle robotene. Funksjonaliteten blir på denne måten kun implementert en gang. Videre er dette en god måte å kunne benytte seg av eksisterende orienteringsfunksjonalitet og kartet som er utviklet tidligere i prosjektet.

4.3.1 Kartet

For å få en bedre forståelse for hvordan løsningen virker vil det være en fordel å vite mer om hvordan sensordata fra roboten behandles og sette opp på serveren. Kartleggingskontrolleren oppretter et GridMap-objekt som er definert som et HashMap[3] som initieres med posisjoner og ruter. Rutestørrelsen defineres når serveren starter opp og vil vise seg å ha innvirkning på nøyaktigheten man kan oppnå. Rutene blir satt opp med rader og kolonner som fungerer som koordinater i kartet og beskriver posisjonen. Når målingene fra sensortårnet kommer inn vil kartleggingskontrolleren først finne ruten i GridMap-objektet hvor roboten befinner seg. Robotens plasseres ved å se på den estimerte posisjonen den har basert på DR. Videre plasseres hindringen som er oppdaget av sensoren i korrekt vinkel og avstand relativt til robotens plassering i kartet. På denne måten blir hindringen tildelt et globalt koordinat som

[3] Abstraksjonsfunksjon i java. Kombinerer nøkler til verdier.



Figur 14: Eksempel på kart. Gul (0, 0), blå (18, 26) og grønn (-18, 26).

stemmer overens med de andre robotene. I figur 14 er kartet generert fra ladestasjonen illustrert. Rutestørrelsen er i eksempelet satt til 2 cm. Roboten vil plasseres i koordinat (0, 26). Den gule ruten representerer koordinat (0, 0) og vil bli sett av sensor 3 når tårnet står i 0° og av sensor 2 når tårnet står i 90° og gir en avstandsverdi mellom 22 - 24. Den blå ruten (18, 26) vil bli oppdaget av sensor 3 og 4 ved henholdsvis 0 og 90° og gir en avstandsverdi mellom 10 - 12. Til slutt den grønne ruten (-18, 26) som kan oppdages av sensor 2 og 4 ved 0 og 90 ° med tilnærmet samme sensordata som den blå ruten.

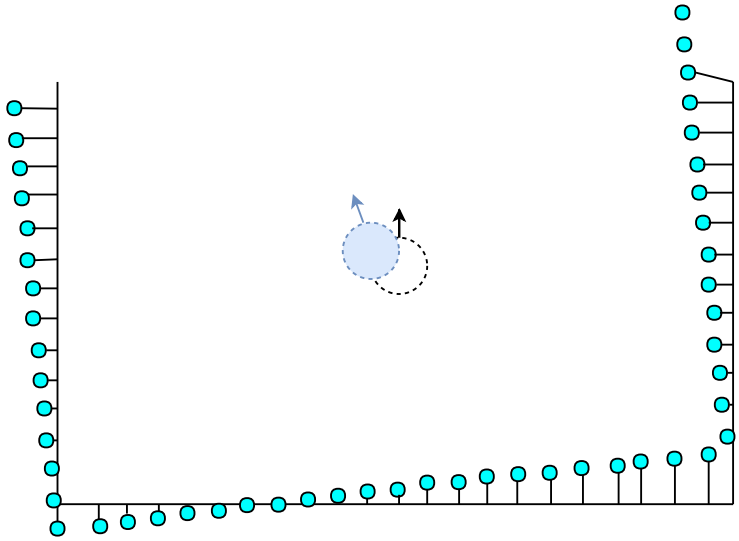
Det kommer tydelig fram fra eksempelet at nøyaktigheten til kartet fort blir dårlig dersom rutestørrelsen blir satt for stor. Med den aktuelle rutestørrelsen vil målinger fra et objekt plassert innenfor et område på 4 cm^2 bli skrevet til samme punkt noe som vil være et i overkant grovt estimat for navigeringen som kreves ved dokking. Det er derfor rimelig å anta at initieringsverdien av rutene i kartet burde reduseres til 1 cm slik at man øker nøyaktigheten i kartet.

4.3.2 Sortering

Den aktuelle løsningen er avhengig av at det utføres to separate kartlegginger med robotens IR-sensorer. Disse vil videre omtales som to sett av punkter som legges i hver sin liste. Settene består av målinger som representerer konturene til robotens nærmeste omgivelser. Som følge av at serveren håndterer målingene relativt til sin egen posisjon kan settene lagres uavhengig av sensortårnets orientering. Settene vil videre omtales som S_{ref} og S_{ny}. S_{ref} blir registrert når roboten forlater ladestasjonen og S_{ny} blir registrert når roboten returnerer. Sorteringen blir en vesentlig del av dokkingalgoritmen og baserer seg på to ulike metoder.

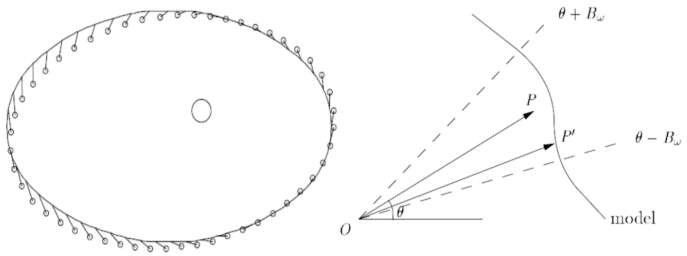
Nærmeste punkt

Denne går, som navnet tilsier ut på å finne det punktet i det ene settet som har kortest avstand til et punkt i det andre settet. Som illustrert i figur 15 kan man se den heltrukkede linjen som representerer punktene oppdaget i S_{ref} utført av den hvite roboten som ser rett frem og de blå sirklene som er oppdaget i S_{ny} av roboten i returposisjon illustrert med blå stiplet sirkel. Denne løsningen fungerer godt dersom rotasjonen er lav og hvis konturene som kartlegges er rette [15]. Dette gir metoden en svakhet når det kommer til rotasjon. Som sett i modellen har vektorene som knytter de nye punktene sammen med referansepunktene ingen konsekvent retning og de har en tendens til å utligne hverandre når man vurderer vinkelforskjellen mellom settene. Metoden fungerer derimot godt når det kommer til translasjon.



Figur 15: Illustrasjon av nærmeste punkt

For å forsikre oss om at metoden finner et unikt punkt som ikke avviker fra sannsynlige antakelser blir det satt en øvre grense for hvilke avstand som aksepteres mellom punktene. Denne grensen vil videre bli omtalt som δ-verdien.



Figur 16: Illustrasjon av absolutt avstand. [4].

4.3.3 Absolutt avstand

Det har i tillegg blitt vurdert en metode som setter opp de to settene i korrekt rekkefølge basert på avstanden fra robotens faktiske posisjon. Dersom man tar to punkter fra hvert sett, P og det korresponderende punktet $P^{'} = R_{\omega}P + T$ og vi ser bort fra translasjonen og antar at den relative avstanden til P og $P^{'}$ er lik. $|P^{'}| \sim |P|$. Dersom man tar for seg polarkoordinatene til de to punktene kan de forbindes på følgende måte: $\hat{\theta} = \theta + \omega$ dette tilsier at de to punktene har samme polavstand men at de har ulik polarvinkel, denne ulikheten representeres av ω som vist til høyre i figur 16. Metoden burde også kunne håndtere små translasjoner og presentere et ok estimat om ω.

På samme måte som forrige metode krever denne en grense for hva som kan aksepteres. Grensen blir her satt ved å innføre en øvre og nedre grense for vinkelen rundt punktet som matches. $\theta - B_{\omega}$ og $\theta + B_{\omega}$. Denne måten å pare punktene på er forsøkt implementert for å bedre håndteringen av rotasjon men er ikke ferdigstilt enda.

4.3.4 Transformasjon

Hovedmålet med transformasjonsalgoritmen som er implementert er å kunne håndtere to sett med punkter som er sortert på en optimal måte og dermed uttrykke forflytningen ved bruk av en rotasjons- og translasjonsmatrise. Arbeidet med denne algoritmen var omfattende og tok mye tid. Det finnes mye teori rundt temaet om autonome roboter og usikkerheter opp mot robotens plassering og orientering. Det var utfordrende å finne korrekt teori for det aktuelle problemet som algortimen skulle baseres på. Som nevnt i teoridelen 4.1 baserer den aktuelle algoritmen seg på singulærverdi dekomposisjon og kvadratisk gjennomsnitt.
Algoritmen starter med de to settene av punkter som hver utgjør en $N \times 2$ matrise

på formen:

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} \tag{12}$$

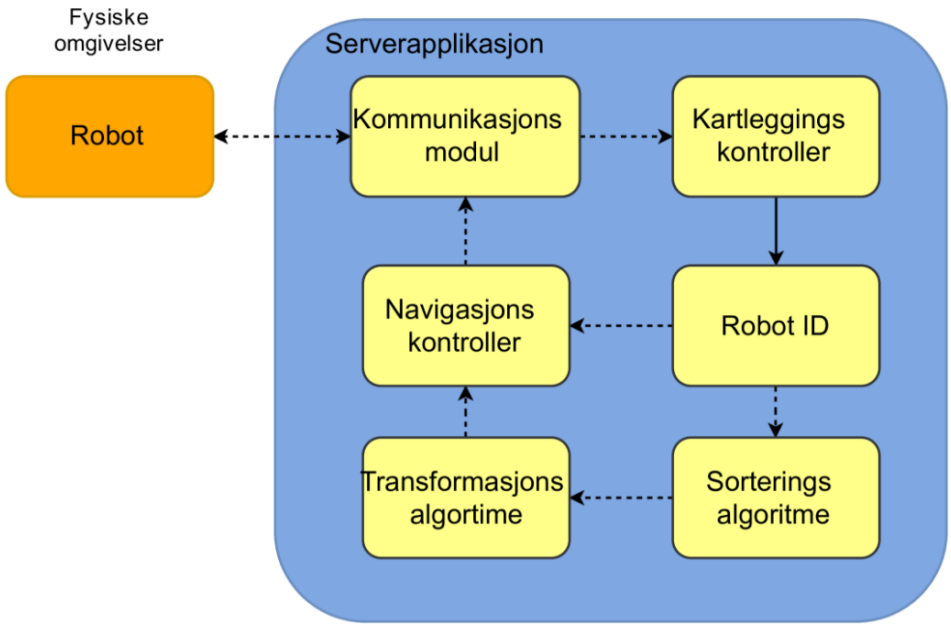
Algoritmen bestemmer midtpunktet av settet som videre blir satt opp i en kovariansmatrise ved å benytte minste kvadraters metode. Translasjonen bestemmes basert på settenes respektive midpunkt. Hvor posisjonsforskjellen mellom de to bestemmer avviket i robotens posisjon.
Til slutt blir den optimale rotasjonen bestemt basert på at man først finner en singulærverdidekomposisjon som vist i likning 10 og setter den opp på følgende måte:

$$\begin{aligned} R_{opt} &= WSV^T \\ &= W \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix} V^T \end{aligned} \tag{13}$$

4.3.5 Implementering

Avgjørelsen om å implementere funksjonaliteten inn på serversiden av systemet ble tatt basert på den eksisterende returfunksjonaliteten. På denne måten vil dokkingsekvensen eksistere et nivå dypere og som en direkte følge av den eksisterende tasken. Implementeringen er utført i flere steg illustrert i flytdiagram 17. Stiplede linjer representerer informasjonsflyt og heltrukkede linjer representerer tilatelse til å redigere data. Først må målinger av robotens fysiske omgivelser hentes inn når roboten starter kartleggingen og forlater ladestasjonen. Informasjonen blir sendt til serverapplikasjonen hvor den behandles som en hvilken som helst oppdateringsmelding som beskrevet i [8]. Neste steg skiller seg fra den eksisterende funksjonaliteten da kartleggingskontrolleren blir bedt om å registrere et sett med målinger for bruk i tilkoblingssekvensen. Denne prosessen utføres to ganger, én når roboten forlater ladestasjonen og en andre gang når roboten kommer tilbake. Begge settene blir lagret hos det aktuelle robotobjektet slik at det kan hentes og brukes ved behov.

De to settene må sorteres og pares opp for videre bruk noe som skjer i sorteringsarlgortimen som sender settene videre til transformasjonsalgortimen. Algoritmen behandler informasjonen slik det beskrives nærmere senere. Navigasjonskontrolleren bruker resultatet fra transformasjonen til å korrigere robotens posisjon og orientering ved å sende informasjonen til kommunikasjonsmodulen og videre tilbake til roboten.



Figur 17: Flytskjema for implementert dokkingløsning.

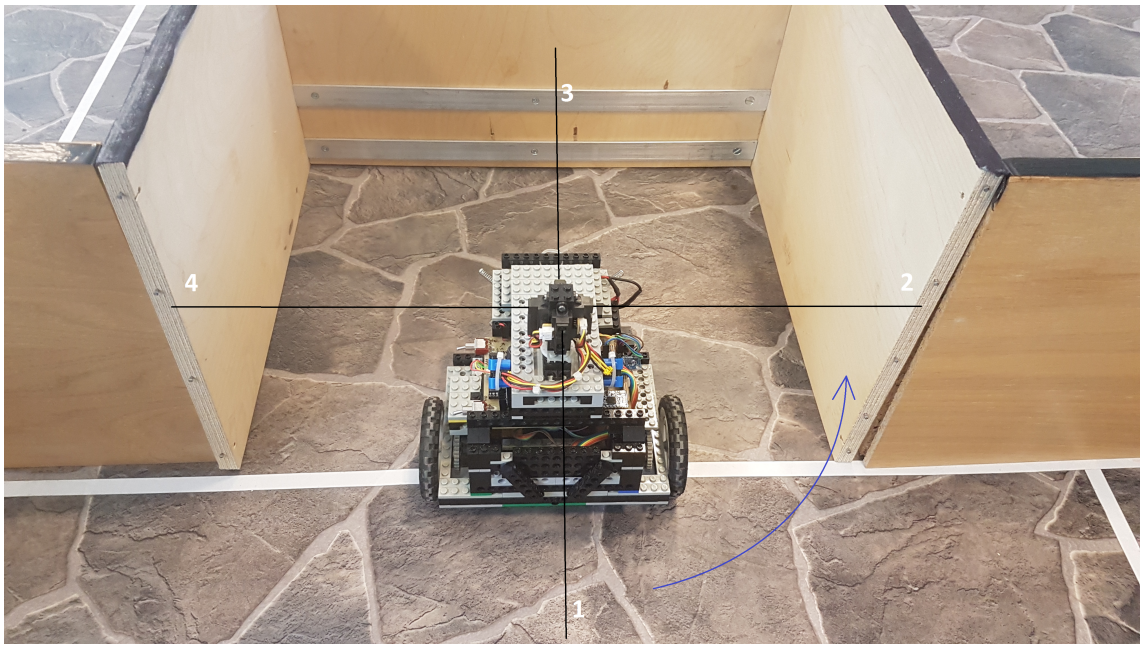
Nå er robotens posisjon tilnærmet lik posisjonen ved oppstart av kartleggingen. Avstanden tilbake til bakvekken av ladestasjonen må identifiseres. Denne avstanden sende fra roboten skrives inn i robotobjektet og rett til navigasjonskontrolleren da det ikke er behov for noe form for beregning av denne verdien.

En grundigere gjennomgang

Det ble lagt fokus på å hente inn data fra et punkt det er mulig for roboten å returnere til gitt unøyaktigheter og svakheter i det eksisterende systemet beskrevet i 2.3. En kjent begrensing det ble tatt høyde for i starten var sensorenes øvre begrensing som beskrives i [8] til 40 cm. Denne begrensingen viste seg kun å være gjelende for genereringen av kartet i serveren men setter ikke begrensing for den aktuelle målingen som algoritmen benytter seg av. Den faktiske grensen for settene som genereres for bruk i denne algoritmen er fra 10 til 80 cm. Disse avstandene er uansett viktige å ha i bakhodet da roboten fortsatt krever gode landemerker den kan identifisere og finne igjen når roboten returnerer. Gode verdier i S_{ref} stiller altså krav til et tydelig fysisk oppsett av ladestasjonen. Innhenting av S_{ref} ble prøvd i ulike posisjoner, dersom man utførte denne 35 cm foran startposisjon som antas å være i (0, 0) fikk man et godt utgangspunkt. Dette gir en posisjon $P_{ref} = (0, 35)$ med orientering $\theta_{ref} = 90°$. Registreringen av disse verdiene var naturlig å utføre gjennom den eksis-

Tabell 2: Robotens posisjon.

Sekvens	Posisjon	Orientering
Start	$(0,0)$	$90°$
P_{ref}	$(0,35)$	$90°$
P_{ny}	-	-
$\hat{P}$	$\sim P_{ref}$	$\sim \theta_{ref}$



Figur 18: Bilde av P_{ref} og sensorer.

terende kartleggingskontrolleren serverapplikasjonen. Registreringen starter å lagre verdiene fra alle de fire sensorene mens tårnet roteres $90°$. Dermed vil roboten registrere omgivelsene i alle retninger. Når roboten har mottatt varsel om lavt batteri, blitt bedt om å returnere eller er ferdig med kartleggingen vil den finne veien tilbake som beskrevet i [1] og i kapittel 4.2. Når den har fullført stien hjem og godkjenner returposisjonen P_{ny}, med en nøyaktighetsmargin på 1 cm vil roboten rotere til det den tror er $90°$ og mener å befinne seg ortogonalt på ladestasjonens bakvegg. Med en gang denne orienteringen er oppnådd vil registreringen av S_{ny} starte. På grunn av feil i robotens posisjonsestimat vil ikke dette være den korrekte posisjonen å fortsette dokkingsekvensen fra. Den vil trolig være posisjonert med feil orientering og plassering. $P_{ref} \neq P_{ny}$. Det vil da være opp til den videre løsningen å identifisere denne feilen og plassere roboten på riktig posisjon for videre dokking.

Detektering av feil skjer ved å sette opp en transformasjonsmatrise mellom de to settene med målinger. Med det menes å finne translasjonen og den optimale rota-

sjonen som vil legge det nye settet over det eksisterende. Transformasjonsmatrisen vil bestå av en rotasjonsmatrise og en translaskonsvektor på formen:

$$R(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$\vec{t} = \begin{bmatrix} t_x \\ t_y \\ 1 \end{bmatrix} \tag{15}$$

Som sammen utgjør en homogen transformasjonsmatrise:

$$T = \begin{bmatrix} cos\theta & -sin\theta & t_x \\ sin\theta & cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

som utfører en rotasjon etterfulgt av en translasjon.

Målet med matrise 16 er å manipulere den faktiske posisjonen i P_{ny} til en ny posisjon $\hat{P} \sim P_{ref}$. For å finne den ønskede posisjonen $\hat{P}$ utføres følgende kalkulasjon:

$$\begin{gathered} \hat{P} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} \\ \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & t_x \\ sin\theta & cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \end{gathered} \tag{17}$$

Sorteringen av de to settene blir utført av en funksjon lagt til i kartleggingskontrolleren. Funksjonen tar inn tre variabler: S_{ref}, S_{ny} og robotens faktiske posisjon. Sorteringen skjer som beskrevet i 4.3.2 og resulterer i en liste som kun inneholder de punktene som har funnet en motpart fra det andre settet som innfrir kravene som er satt. Den nye listen kalles matchedList og brukes videre i transformasjonsalgoritmen. Psaudokode er inludert i 4.2.

Listing 4.2: Sorteringsalgoritmen.

```
public SimpleMatrix scanMatching(S_ref, S_new, currentPosition){
// Matching set of points to distance
//Create variables
    upperLimit;              //delta-value
    angleLimit;
for(point:S_new){
    for(point:S_ref){
        find distance between points;
        if(distance is shorten then the last best and the limit){
            update shortestDist;
            update bestMatch;
        }
        // The new point have been compaird with all the ref
           points
        if(a match has been found){
        save both points to the matchedList;
        }
    }
    // When all possible points are paird.
   Translation = TransformationAlg (matched points);

   for(point:S_new){
   find distance to point;
   for(point:S_ref){

       if(distance is between limits and angle between limts){
           update smallestAngle;
            update bestPoint
        }
        // The new point have been compaird with all the ref
           points
        if(a match has been found){
        set both points to the matchedList;
        }
    }
     // When all possible points are paird.
    TransformationAlg dokking = new TransAlg(matched points);

    R = dokking get rotation matrix;
    t = dokking get translation vector;
```

```
    T = Combine Rotation and Translation;
    return T;
}
```

Implementeringen av transformasjonsalgoritmen ble blant annet løst gjennom å importere to nye pakker i Java.

- SimpleMatrix
- Simple SVD

Disse gjør det mulig å sette opp matriser og man får en rekke vektøy for matriseregning som ikke finnes originalt i java. I tillegg kan man ved hjelp av den andre pakken sette opp en singulærverdidekomposisasjon av en matrise med en enkelt kommando. Kalkuleringen som utføres i algortimen er beskrevet nærmere i psaudokode i listing 4.3.

Listing 4.3: Transformasjonsalgoritme.

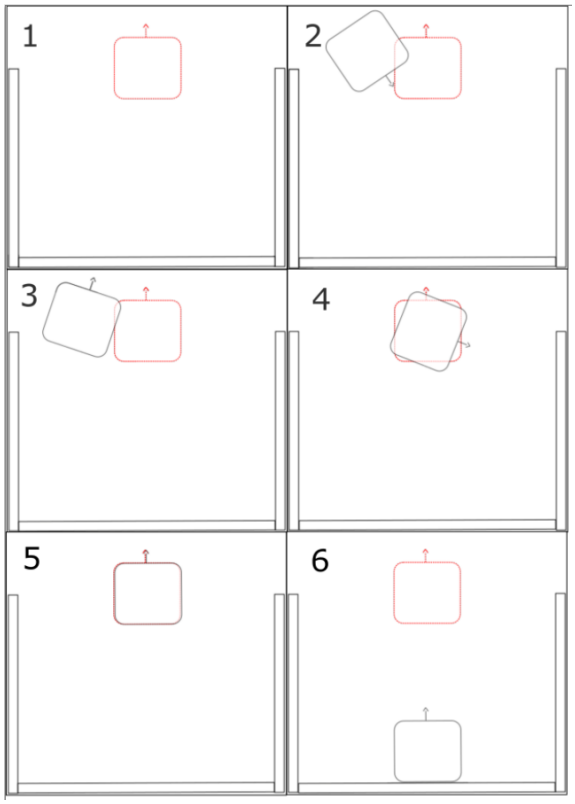
```
// Find SVD
A   = create covariance matrix of the two sets;
svd = create SVD from matrix A;
V   = take the transpose of svd;
W   = extract diagonal values from svd;

// Correct for flipped axis and make the rotation clockwise
R   = W multiplied by V;
d   = absolute value of R's determinant;

// Calculate optimal rotation
S   = identity matrix 2*2;
insert d into S at (1,1);
rotation matrix = W * S * V;

// Calculate translation matrix
create HashMap of centroids of the two sets;
translation matrix = R * ( (-) centroids of newSet + centroids
                     of refSet transposed);
```

Fra den genererte transformasjonsmatrisen finner man ut hvilke rotasjon roboten må utføre for å komme parallelt med det opprinnelige posisjonen. Fra rotasjonsmatrisen



Figur 19: Illustrasjon av hvordan roboten manøvreres ved tilkobling.

ble det valgt å trekke ut sinusleddet fra kolonne 1 rad 2. På denne måten får man informasjon om retningen på rotasjonen da denne verdien vil inneholde informasjon om fortegnet til rotasjonen. Rotasjonen blir satt til en variabel i den aktuelle roboten og hentes fram i navigasjonsdelen av dokkingen.

Manøvreringen av roboten til den ønskede posisjonen $\hat{P}$ utføres i navigasjonskontrolleren hvor all annen navigering skjer. Det ble bestemt at dette best løses ved å sette opp en switch case da enkelte manøvreringer kan skjer flere ganger. Valg av switch statement ble satt til en variabel som hver enkelt robot blir tildelt når kontakten opprettes mellom den og serveren. Variabelen initieres til -1, som sørger for at ingen forstyrrelser i det opprinnelige systemet. Variabelen kan betraktes som en semafor. Da kan man enkelt sende navigeringen inn i ønsket sekvens ved å endre denne parameteren.

Hovedsegmentene som må utføres er

- Rotere roboten til θ_{ny} (3)
- Vente til S_{ny} er innhentet og lagret

- Manøvrere roboten til $\hat{P}$ (4)
- Rotere roboten til $\hat{\theta}$ (5)
- Manøvrere roboten inn til ladeskinnen i ladestasjonen (6)
- Sette roboten i pausemodus

Listen kobles opp mot illustrasjonen i figur 19 for visualisering. I illustrasjonen kan man i vindu 1 se robotens posisjon når den starter kartleggingen. Posisjonen som er merket i rød stiplet linje tas med videre slik at man kan se feilen i returposisjonen. Resten av vinduene beskrives gjennom listen over.

Det er en utfording at roboten ikke kan utføre kommandoer mindre enn 5 cm og 5°. Løsningen på dette er å innføre konstanter for å skalere verdiene som sendes. Tre av disse konstantene er α for rotasjon, σ for avstanden tilbake til ladeskinne og δ for øvre grense av avstand mellom punktene når de sorteres. Valgene av disse viser seg å ha stor innvirkning og blir nøyere beskrevet i testene 4.4.

Noen momenter fra koden

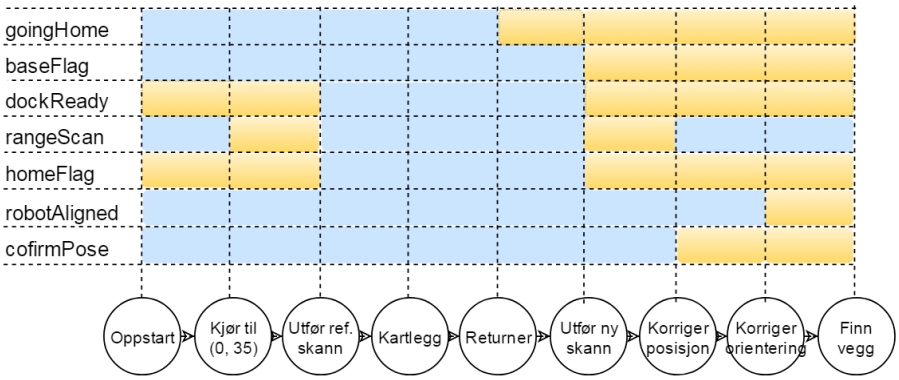
For å gjøre transisjonen over til nestemann som skal jobbe med dette tema vil det være viktig at tankegangen bak implementasjonen beskrives tilstrekkelig. Derfor inkluderes en beskrivelse av oppsettet til koden for dokking. Dette delkapittelet vil ikke være nødvendig for å forstå løsningen men til fordel dersom man skal sette seg inn i og utbedre implementeringen.

Ved oppstart blir roboten bedt om å kjøre 35 cm fram. Denne verdien kan endres i NavigationController.java. Denne er nå satt til punktet (0, 35, 0) da dette er et punkt i forhold til ladestasjonen som gir mye informasjon om områdets geometriske sammenstilling og er grundigere dokumentert i starten av dette delkapittelet. Valg av returposisjon blir satt i Robot.java hvor stasjonen defineres ved en bestemt posisjon kalt, "basePosition". Dette punktet kan endres fritt og vil ikke påvirke funksjonaliteten ytterligere enn at det kan redusere nøyaktigheten dersom rekkevidden av ir-sensorene blir dårligere. Oppsettet nå sender roboten tilbake til den samme posisjonen som S_{ref} blir registert, (0, 35). Det stilles ingen krav til at punktene er sammenfallende siden transformasjonsmatrisen baserer seg på punkter med globale variabler og har derfor ingen innvirkning på korreksjonen av posisjonsfeilen.

Selve registreringen av de to settene blir gjort i MappingController.java. Roboten initieres med en del boolske variabler satt og utgjør sammen den logiske grunnlaget for når settene skal innhentes, samt andre viktige momenter ved dokkingen. Alle variablene finnes i Robot.java. Logikken består av variablene oppramset i figur 20. Illustrasjonen viser en suksessfull dokkingsekvens og de faktiske boolske verdiene. Under kan man se de aktuelle hendelsene og de vertikale stiplede linjene beskriver tidspunktet hvor overgangsbetingelsene vurderes. Da må de aktuelle variablene være positive/negative i henhold til figuren for at den aktuelle hendelsen skal startes. Variablene spiller i tillegg en viktig rolle for hverandre slik at de skal vite på hvilke tidspunkt hvilke variabel skal bli justert fra høy til lav og vice versa.

Bruk av switch case har såvidt blitt nevnt tidligere. Denne ble satt opp for å redusere overdreven bruk av if-setninger og som et forsøk på å gjøre det hele mer smidig. Switch casen er satt opp til å håndtere sju ulike tilfeller.

I tilfelle tre behandles robotens orientering, dette har vist seg å være en utfordrende affære da roboten sliter med å justere for små vinkler. Som nevnt ble konstanten α innført for at roboten skal overkompansere litt. I tillegg ble det lagt til at roboten først roterer 90° mot klokka for så å rotere tilbake men nå med den korrigerte verdien lagt til. Dette resulterer i kommandoen: -(robotens orientasjon + korreksjon - 90°.



Figur 20: Logisk kontroll ved viktige hendelser. Gul og blå representerer henholdsvis høy og lav boolsk verdi.

Tabell 3: Beskrivelse av tilfeller i switch case

Nr	Beskrivelse
1	Sender roboten til den oppdaterte posisjonen
2	Korrigere for feil i orienteringen
3	Oppdatere robotens posisjon i serveren
4	Starte søk etter vegg med ladeskinne og rygge inntill
5	Pause roboten
6	Starte roboten på nytt
7	Dersom roboten ikke oppdager noe vegg, send den 15 cm bakover

På denne måten endter man opp med ønsket retning.

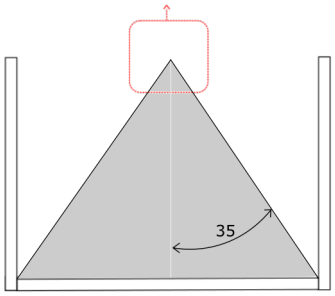
Neste tilfelle som burde beskrives grundigere er nummer fire. Her blir switch casen brutt for å utnytte en funksjon i MappingController.java. Her blir verdiene fra sensor to og tre analysert for å finne avstanden inn til veggen med skinnene. De to sensorene burde ha visuell kontakt med veggen ved henholdsvis 0 og 90 grader. I utgangspunktet kreves kun visuell kantakt fra en av disse men for å effektivisere sekvensen og slippe å potensielt måtte vente mens en hel rotasjon ble utført kan den nå oppdage veggen med begge sensorer. Måleverdien fra den respektive sensoren registreres til robot.java og blir den aktuelle backUpDistancen. Den registrerte avstanden blir behandlet i funksjonen findWall i MappingController.java. Her blir den også nedskalert for å passe med virkeligheten, dette skjer ved å multiplisere med konstanten δ. Nå kan switch casen startes opp igjen men nå initieres den rett inn igjen i tilfelle fire. Den vet nå at den har en distanse den skal kjøre og utfører dette direkte.

På dette tidspunktet skal roboten ha oppnådd kontakt mellom ladeskinnene og fjær-

kontakten bak på roboten. For å sette roboten i pause benyttes en eksisterende funksjon fra application.java som kan fryse roboten. Handshaken brytes ikke mellom serveren og roboten men alle funksjoner stanses og roboten bruker på dette tidspunktet ikke strøm på noe. For at de andre robotene skal kunne fortsette kartleggingen uforstyrret må switch casen brytes og switch statementet settes til den orginale verdien lik -2, som ikke har noen annen funksjon enn at det bryter med overgangsbetingelsen for while-løkka som omfavner switch casen.

4.3.6 Krav

For å kontrollere om løsningen kan utføre dokkingensekvensen ble det satt minimumskrav som må oppfylles. Kravene omfatter eliminering av rotasjonsfeil og et minstekrav for antall par som blir matchet i transformasjonsalgoritmen. Det kreves at 50% av punktene i S_{ny} må finne et motstykke i referanselisten innenfor grensen δ. Gitt en returposisjon som faller innenfor grensen for hva algoritmen kan håndtere må feilen i orientering reduseres ned til minimum 35°. Kravene kommer som en direkte følge av stasjonoppsettet og ladesystemet som beskrevet i 2.2.3.



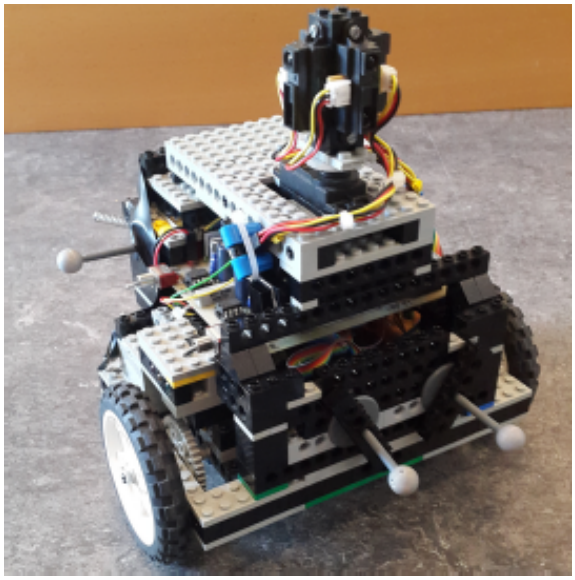
Figur 21: Illustrasjon av krav til orientering ved retur.

4.4 Tester

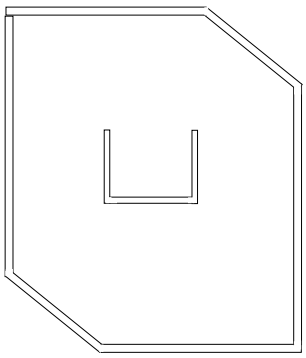
Løsningen som har blitt presentert i dette kapittelet setter et høyt krav til nøyaktighet. For å oppnå dette har det blitt utført omfattende testing gjennom hele utviklingsprosessen. Testene og dens resultater førte til omfattende endringer og på et punkt ble løsningen totalt endret. Den originale dokkingfunksjonen ble skrotet og nevnes ikke nærmere i rapporten. Mot slutten av arbeidet oppstod det en feil i kommunikasjonen mellom serveren og AVR-roboten da den en dag koblet seg ut under kjøring. Problemet har blitt omfattende testet og debugget. Problemet ser ut til å oppstå ved vilkårlige tidspunkter under en kjøring noe som gjør debugging vanskelig. Problemet ser ut til å opptre uavhengig av hvilke versjon som er flashet på robot, 2 × dongel og serverapplikasjon. Det ble også testet med software som ble vedlagt fra [2], [8] og [11] uten tegn til forbedringer. Problemet kan skyldes at mikrokontrolleren mottar ett signal den ikke skal motta. Dette er mulig å undersøke nærmere ved å benytte seg av MCU funksjonaliteten i mikrokontrolleren hvor årsaken skulle være mulig å finne. Denne funksjonen er implementert som beskrevet i atmega1284p datablad side 56. Tidvis fungerte kommunikasjonen bra og det var mulig å kjøre lengre tester men det ble nødvendig å støtte seg til simulatoren også.

Bruk av OptiTrack Motion Capture System

For å sikkert kunne produsere resultater av løsningen som ble implementert ble bevegelseslabben på B333 ved NTNU benyttet. Systemet består av en rekke kamera plassert i taket rundt i rommet. Objektet som skal registreres utstyres med



Figur 22: Roboten under testing.



Figur 23: Labyrinten brukt ved test av returfunksjon.

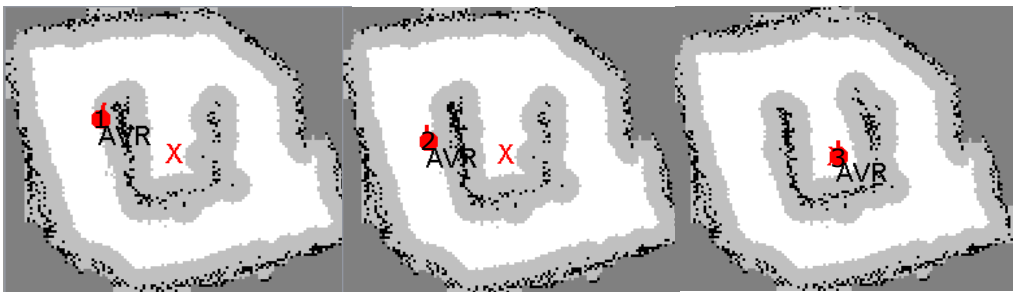
refleksjonskuler og som blir fanget opp av de ulike kameraene i et todimensjonalt bilde. Refleksjonskulenes posisjon blir så kalkulert og plassert i to dimensjoner og de overlappende posisjonsdataene blir så beregnet til tre dimensjoner ved bruk av triangulering. Til tross for at man nå har kunnskap om robotens tredimensjonale posisjon er det kun behov for x og y aksen i dette tilfelle.

Illustrasjon 22 viser oppsettet av refleksjonskulene på AVR-roboten. For å få eksakt plassering er det viktig at kulene er plassert utover slik at sentrum blir i det faktiske midtpunktet av roboten.

4.4.1 Returfunksjon 2.0

Det ble satt opp en mindre labyrint som vises i figur 23. Testen ble utført ved å la roboten kjøre inn i labyrinten for så å bli bedt om å returnere på normal måte. Robotens feilestimat gjør at returen blir vanskelig da det dukker opp en hindring i robotens planlagte rute tilbake. Slik blir robotens rute umulig å utføre da begrensninger blir satt i kartet.

Fra figur 24 illustreres roboten i posisjon 1 da den støter på et begrenset område. I 2 har den kalkulert en ny kjøreliste og rygger tilbake for å komme forbi det problematiske punktet og til slutt når den har returnert i 3.



Figur 24: Returtest.

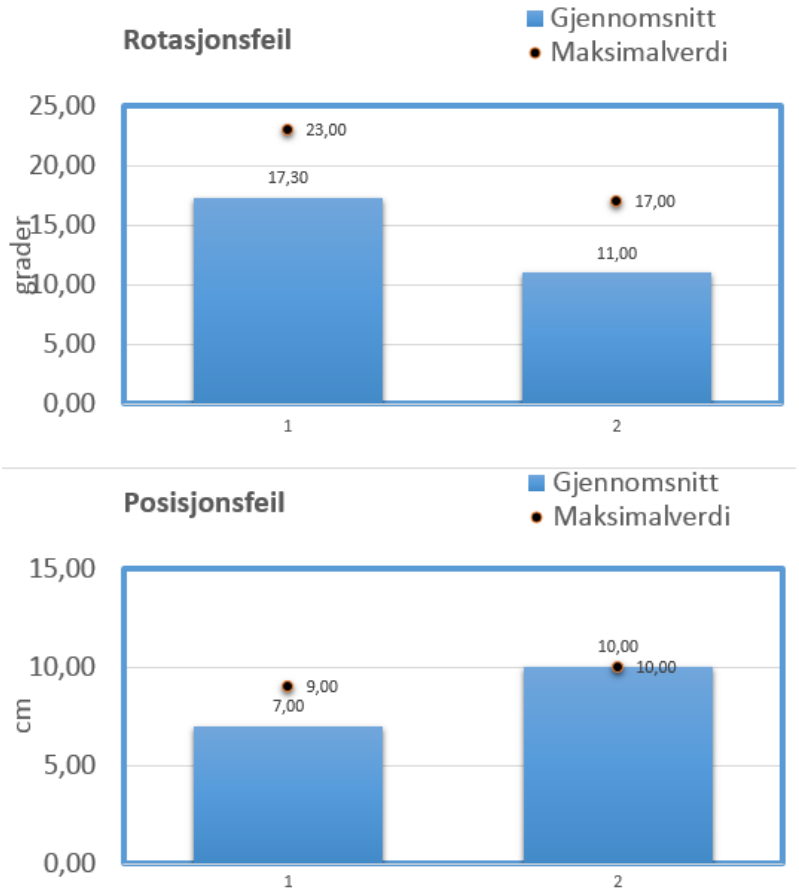
4.4.2 Test med ladestasjon

Test av δ-verdi

Testene av dokkingfunksjone ble utført ved å sende roboten fram til punktet definert som $P_{ref} = (0, 35)$. Målet med dette var å se om roboten kunne håndtere feil i posisjonen ved retur. For å effektivt stressteste systemet ble det derfor påført bestemte endringer på ladestasjonens posisjon etter at S_{ref} ble registrert. Endringene som ble testet ut var rotasjon og sideveis forflytning. Det vil bli presentert dokumentasjon fra tre av testene som ble utført.

Første test ble utført med et kart initiert med ruter tilsvarende 2 cm som forklart i 4.3.1 og en δ-verdi = 20 cm. På dette tidspunktet var det ikke innført andre konstanter for justering av dokkingen. Figur 25 illustrerer resultatene av 10 forsøk med de gitte parametrene. Første søyle viser gjennomsnittet av den resterende feilen i rotasjonen til roboten etter justeringen er utført. Den høyeste feilen som oppstod iløpet av testserien vises med en svart prikk.

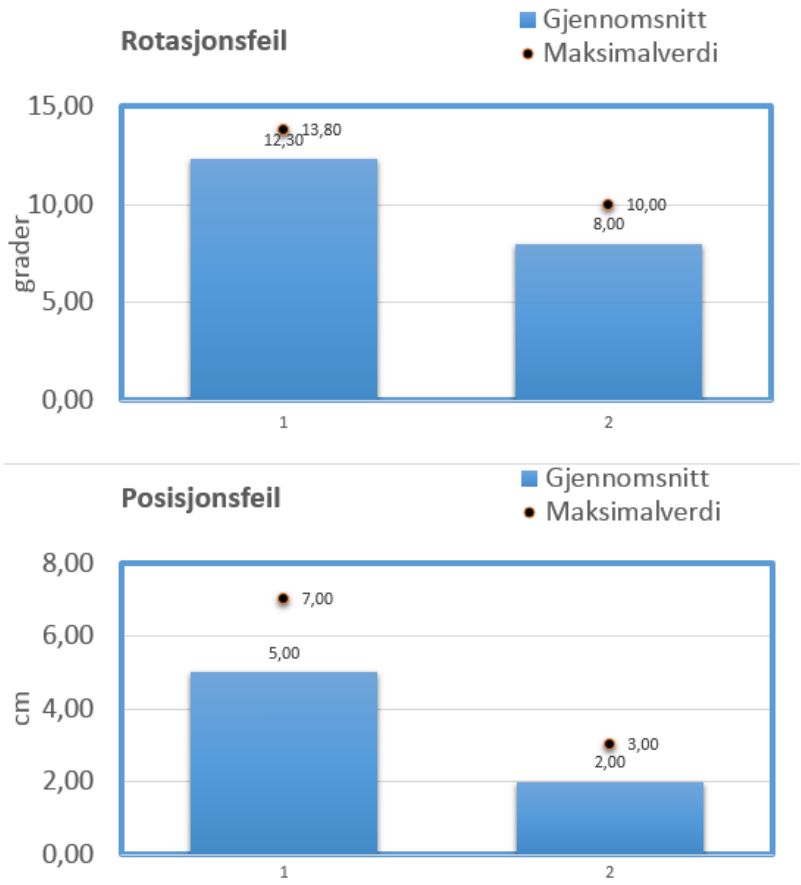
Under de første testene presterte roboten dårlig. Sorteringsalgoritmen fant 480 par av 530 mulige, noe som burde gitt et godt resultat. Maksverdiene er også innenfor kravene som er satt. Under testen var det tydelig at algoritmen identifiserte en feil i



Figur 25: Dokkingtest 1. Søyle en viser rotasjonsfeil med $\delta = 20$. Søyle to viser rotasjonsfeil med $\delta = 5$.

robotens posisjon. Og den justerer for denne feilen i korrekt retning noe som tilsier at algoritmen fungerer slik den skal men at nivået på outputen ligger for lavt. En videre konklusjon som kan trekkes fra dette er at nøyaktigheten av rotasjonskorreksjonen kan økes ved å redusere δ-verdien basert på at det ble produsert bedre resultater da denne ble redusert til 5 cm. Dette ble først prøvd ut som en ekstrem verdi og burde resultert i langt dårligere resultater da avstanden mellom posisjonene fort kan overskride 5 cm.

Translasjonstesten ble utført ved å flytte ladestasjonen 5 cm i x og 5 cm i y retning, totalt 7 cm. Resultatene med de samme parametrene som første rotasjonstest viste store feil og understreker at den aktuelle løsningen ikke er god nok. Med δ-verdien som virket lovende basert på resultatende med ved rotasjonstesten gav elendige resultater da translasjonsalgoritem fant for få matcher, kun 110 av 530. Fra del to av figur 25 ser man at søyle to ikke justerer noe for translasjonsfeilen noe som gir mening da grensen til δ-verdien er lavere en forflytningen.



Figur 26: Dokkingtest 2. Søyle en viser resterende gjennomsnittet av rotasjonsfeil med rutestørrelse 2 cm. Søyle to viser resterende rotasjonsfeil med rutestørrelse 1 cm

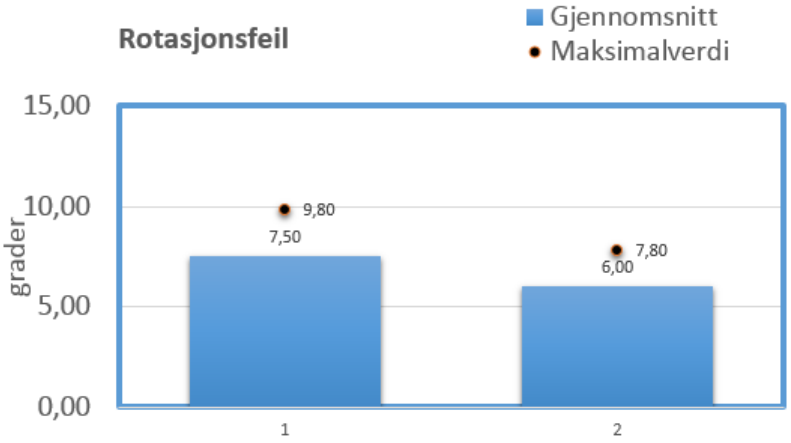
Basert på erfaringene gjort fra første test vil grensen i translasjonsalgoritmen økes til 15 cm og nye forsøk ble gjort. De to testsettene som legges frem i figur 26 skilles fra hverandre ved at serie 1 har et kart initiert med rutestørrelse 2 cm og serie 2 initieres med rutestørrelse 1 cm. Resultatet viser at man kan øke δ-verdien uten å måtte gå på bekostning av nøyaktigheten.

Innføring av konstanter

Testene viste noen systematiske feil i løsningen. Som at roboten ikke er i stand til å justere for små feil i orientering og at avstanden den skal rygge blir overestimert. Den førstnevnte skyldes robotens grove grense for hvilke bevegelser den kan utfør og den siste skyldes kalibreringen og settpunkt av IR-sensorerne ligger for høyt. Det ble derfor bestemt at to konstanter skulle innføres, α for vinkelen og σ for nedskalering av avstanden tilbake til ladeskinnen. Posisjonsfeilen var god i test 2 og vil derfor ikke bli testet videre og figur 27 viser derfor kun rotasjonsfeilen.

Tabell 4: Parametre fra test 3.

Serie 1	Serie 2
$\delta = 15$	$\delta = 15$
$\alpha = 1.5$	$\alpha = 3$
$\sigma = 0,75$	$\sigma = 0,75$
rute = 1 cm	rute = 1 cm



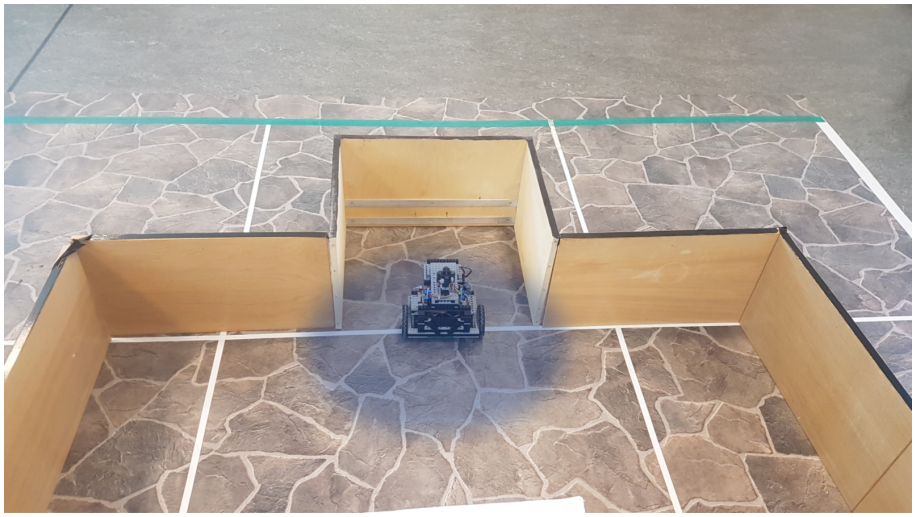
Figur 27: Dokkingtest 3. Viser gjennomsnittet av rotasjonsfeil basert på parametre gitt i tabell 4.

Verdiene algoritmen beregnet var for små og roboten klarte ikke i håndtere verdiene. Derfor ble de to konstantene satt til:

Basert på data samlet fra de totalt 30 forsøkene som ble utført endte parametrene opp slik som de ble satt i serie 2 4. Den resterende feilen i rotasjon har et gjennomsnitt på 6°, en verdi som ansees å være et meget godt resultat gitt utgangspunktet. Når det gjelder feil i posisjonen gitt de samme parametrene blir gjennomsnittet av feilen lik 2 cm som vist i 26 del 2.

4.5 Diskusjon

Som nevnt tidligere var det en viktig avgjørelse hvorvidt dokkingen skulle settes opp lokalt på roboten eller i serveren. Det kan argumenteres for at så mye som mulig av en løsning burde implementeres på roboten slik at den vil være mer robust dersom roboten skulle miste kontakten med serveren. Roboten vil på denne måten bli mer selvstendig. En slik løsning ville vært mulig dersom man opprettet en task som kunne filtrerte ut de ønskede verdiene fra ir-sensorene på en lignende måte som det nå har blitt implementert på serveren. Settet ville da kun vært tilgjengelig for den



Figur 28: Illustrasjon av maksimalt avvik som håndteres av løsningen.

aktuelle roboten. Dette til tross har denne oppgaven fokusert på å gjøre systemet som helhet mer autonomt og robotens selvstendighet har ikke blitt vektet i stor grad. Når problemet løses i serveren er det mulig for flere roboter å benytte seg av de ulike settene som genereres og andre trenger ikke å utføre sine egne. Roboten har kun kjennskap til sin egen posisjon relativt til startposisjonen og har ingen informasjon om det genererte kartet. Slik det er satt opp nå blir settet tildelt globale koordinater. Utvidet bruk av funksjonen blir også mulig når det løses på serveren.

I arbeidet med oppsettet i serveren ble det vurdert om det skulle opprettes en subklasse av den eksisterende robotklassen. Denne ville initiere de ulike robotene når dokkingsekvensen ble startet opp. Dette ble ikke gjort, noe som har ført til at alle roboter nå blir satt om med variablene som kreves fra oppstart. Leseren blir gjort oppmerksom på at resultatet av dette er at robotklassen nå er meget omfattende og inneholder mye funksjonalitet som ikke nødvendigvis blir brukt.

Oppsettet av settene som samles inn har blitt løst slik at den maksimale rekkevidden til robotens ir-sensorer nå blir utnyttet. Verdier opp mot 80 cm unna roboten vil nå bli registrert og kan brukes til å beskrive robotens faktiske posisjon.

Valg av konstanter og parametre ble, som tidligere dokumentert, gjort på grunnlag av testene som ble utført. Parameteren med størst innvirkning på den praktiske funksjonen av løsningen er δ-verdien. Siden den bestemmer maksimal avstand mellom punktene i de to settene som matches bestemmer denne variabelen i praksis også maksimalavviket mellom faktisk og estimert posisjon når roboten returnerer. For visualisering inkluderes illustrasjon av maksimalområdet som kan håndteres. Se

figur 28.

Valg av σ ble testet ved å kjøre en rekke tester hvor roboten skulle rygge inn til ladeskinnen. Ved flere anledninger viste det seg at avstanden den aktuelle sensoren registerte for høye verdier. Resultatet av dette ble at roboten fortsatte å rygge etter at den nådde ladeskinnen noe som forstyrrer posisjonsestimatet og ødelegger kartet som er generert. Dette skjer fordi roboten ikke har noen måte å oppdage spinn mellom hjulene og underlaget. Enkodertikkene vil fortsette å telle nedover mens hjulene roterer. At IR-sensoren gir høye verdier skyldes at kalibreringen ligger litt høyt for slik bruk men er godt kalibrert for kartgenerering. Derfor blir ikke sensorene i seg selv justert men problemet løses ved å bruke denne konstanten.

Valget av α-verdien ble tatt for minimere feilen til rotasjonen. Verdien blir multiplisert med orienteringskorreksjonen etter at denne hentes ut fra rotasjonsmatrisen. Utfordingen her var at roboten ikke klarte å rotere når kommandoen som blir gitt beståre av for små endrigner. Derfor ble den oppskalert slik at roboten håndterer den bedre.

5 Økt nøyaktighet AVR

Arbeidet med dette kapittelet er tett knyttet opp mot masteroppgave skrevet parallelt av Jørund Amsen som har restrukturert og endret mye i kildekoden til en tilnærmet identisk robot. Dette for å forbedre navigasjonen på Arduino-roboten. Både Arduino og AVR baserer navigasjonen på et godt estimat for robotens posisjon til enhver tid. Med posisjon sikter man til et x- og y-koordinat samt en vinkel for retningen. (x, y, θ). I litteratursøket ble det stadfestet at det per dags dato ikke finnes noen enkelstående løsning på utfordringer knyttet til nøyaktig navigasjon. Løsningene som eksisterer kan sorteres inn i to hovedkategorier, *relativ* og *absolutt* posisjonsmåling. Siden det ikke finnes noen perfekt løsning er det konsensus om at navigasjonen løses best ved å kombinere en løsning fra den ene kategorien med en fra den andre. De mest brukte løsningen kan deles opp på følgende måte:

1. Odometri
2. Intern navigasjon
3. Kompass
4. Landemerke markert med laser
5. Navigere på landemerker
6. Modell matching
7. GPS

1-2 går under kategorioen *Relativ posisjon (DR)* og 3 - 7 er *Absolutt posisjon.* Leseren blir gjort oppmerksom på at GPS er uaktuell for innendørs navigasjon.

Robotene baserer navigasjonen på odometri og intern navigasjon. Odometri baserer seg på å bruke integrasjon av robotens bevegelse over tid, dette vil til slutt vil

føre til en kumulativ feil. Spesielt er metoden svak for feil på robotens orientering som igjen vil føre til store feil i posisjonsestimatet, videre omtalt som DR-feil [16]. Til tross for store svakheter er odometrimetoden en viktig del av navigasjonen i mobile roboter. Men bruk av algoritmer for eksakt navigasjon på roboten vil ikke alene være tilstrekkelig. Den kumulative feilen gjør at roboten vil drifte bort fra den faktiske posisjonen og man er derfor avhengig av å kombinere med en løsning for absolutt posisjon. Noen eksempler på alternativer man kan benytte er kalibrering og mekaniske utbedringer [16], ekstrahere relativ posisjon ved bruk av flere roboter [17] eller bruk av landskapskgjenkjennende algoritmer som etterligner et nevronettverk [18]. De to siste er eksempler på *absolutt posisjon*.

Den interne navigasjonen benytter seg av gyroskop og akselerometer for å måle endringsraten i orienteringen og akselerasjonen. Målingene integreres et nødvendig antall ganger, $\int \dot{\theta} dt$ og $\int\int \ddot{\alpha} dt$, for å uttrykke den eksakte verdien. Fordelen ved å benytte disse er at de kan operere på egenhånd og trenger ingen ekstern påvirkning, svakheten er at de vil over tid drifte vekk fra den korrekte verdien som følge av at integrasjonen medfører en konstant liten feil og sensorene vil over tid bli upålitelige.

5.1 Feilsøking

For å effektivt kartlegge hvordan roboten faktisk fungerer ble det bestemt at en ny kommunikasjonsprotokoll skulle utarbeides. På denne måten kan man få en tydelig oversikt over hvilke dataverdier som eksisterer på roboten. En slik funksjon må kunne lagre og logge dataen på en oversiktlig måte slik at man enkelt kan gå inn i loggfilen å se de utvalgte dataene. Til nå har dette blitt utført ved å bruke Termite 3.1. Programmet har blitt brukt ved å koble det opp mot serverdongelen slik at kommandoene kan sendes manuelt. Meldingene fra roboten kommer opp i konsollen i Termite. Bruken av programmet krever at meldingene sendes fra main-tasken i robotens kildekode. Dette fungerer OK men det vil være en fordel å ha denne muligheten også i serveren slik at debugging og logging kan utføres når systemet opererer normalt.

Den oppdaterte kommunikasjonsprotokollen er hovedsaklig implementert til i kildekoden til roboten. Det ekssiterer allerede en omfattende kommunikasjonsprotokoll. Begrensningene på kommunikasjon mellom roboten og serveren er satt av bluetooth-dongelene. Hver melding som oversendes må være under 23 byte [19], dersom meldinger overskrider denne grensen vil den bli delt opp i flere meldinger. Serveren identifiserer begynnelsen og slutten av meldingene ved å benytte seg av krøllparanteser. Protokollen for roboten ble utarbediet som en del av denne oppgaven og

Henrik Melbø utviklet protokollen i serverapplikasjonen.

Debugging av de aktuelle sensorene ble utført ved å benytte seg av denne nye funksjonen og resultatene fremlegges her. Roboten er utstyrt med sensorene som framgår i tabell 5.

5.1.1 Akseleromenter

Ved testing av akseleremoteret kjøres roboten og verdiene fra IMUen hentes ut i x-retning. I masteroppgaven til Ese 2016 [2] beskriver han hvordan akselerometeret gir store utslag/forstyrrelser ved ujevnt underlag og som følger av vibrasjoner. Dette støttes opp av resultatene fra testene som ble utført i denne oppgaven. Når roboten starter translatorisk bevegelse ser man reaksjonen i akselereometeret. Verdien alternerer mellom positive og negative verdier. Det har ikke blitt gjort rede for hvorvidt dette skyldes vibrasjoner fra underlaget eller noe annet men det er uansett vanskelig å benytte seg av disse verdiene. Utsnitt fra debuggingsfunksjon finnes i figur 29. Resultatene viser en dårlig signal-to-noise rate ved lave akselerasjoner og at verdiene drifter over forskyves over tid. Svakheten ovenfor unøyaktigheter ved underlaget skyldes at en liten forstyrrelse bort fra et perfekt flatt underlag vil gi sensoren en akselerasjonskomponent g. Ved tester for å eliminere dette har er de beste resultatene som har blitt produsert gitt en forkyving på 1 til 8 cm/s, dette er feil på en størrelsesorden som er uakseptable for robot navigasjon [20].

Tabell 5: Sensor AVR

Sensor	Egenskap
Akselerometer	$\ddot{x}$ og $\ddot{y}$
Enkoder	$\dot{x}$ og $\dot{\theta}$
Gyro	$\dot{\theta}$
Kompass	θ

```
[0]:AVR:rWh:773      lWh:773
[0]:AVR:rWh:779      lWh:779
[0]:AVR:rWh:785      lWh:785
[0]:AVR:rWh:791      lWh:792
[0]:AVR:rWh:796      lWh:796
[0]:AVR:rWh:796      lWh:797
```

Figur 29: Enkoderverdier etter å ha kjørt en meter fram

5.1.2 Enkoder

For å teste ut verdiene til enkoderen bli roboten kjørt manuelt fram og tilbake mens enkodertikkene ble logget gjennom debuggingsfunksjonen. Etter å ha kjørt roboten en meter fram viste loggen verdiene som vises i figur 29.

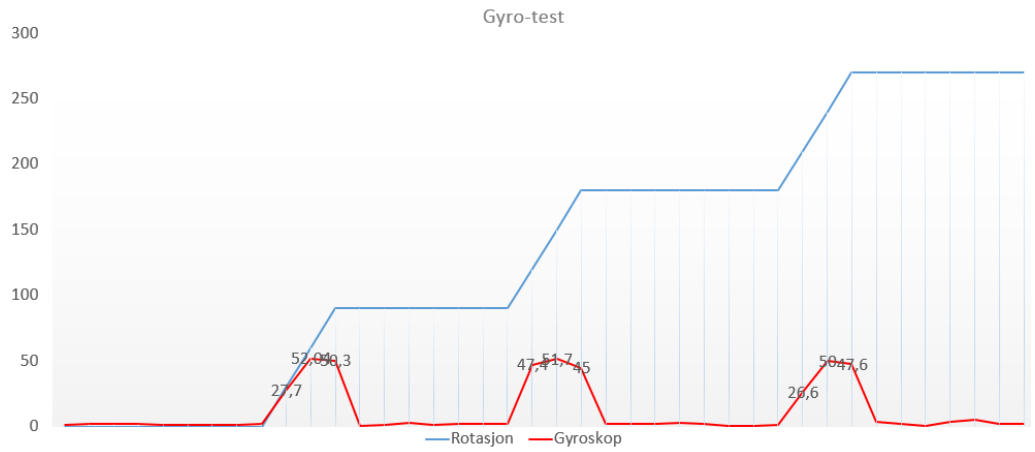
Avviket mellom høyre og venstre hjul blir et tikk. Når roboten blir kjørt en meter tilbake ser det ut til at avviket i enkoderene følger samme feilen. Og den venstre ser ut til å holde seg et tikk bak. Fra flere tester viste det seg at dette er en liten og systematisk feil og har derfor liten negativ innvirkning på posisjonsestimatet.

5.1.3 Gyroskop

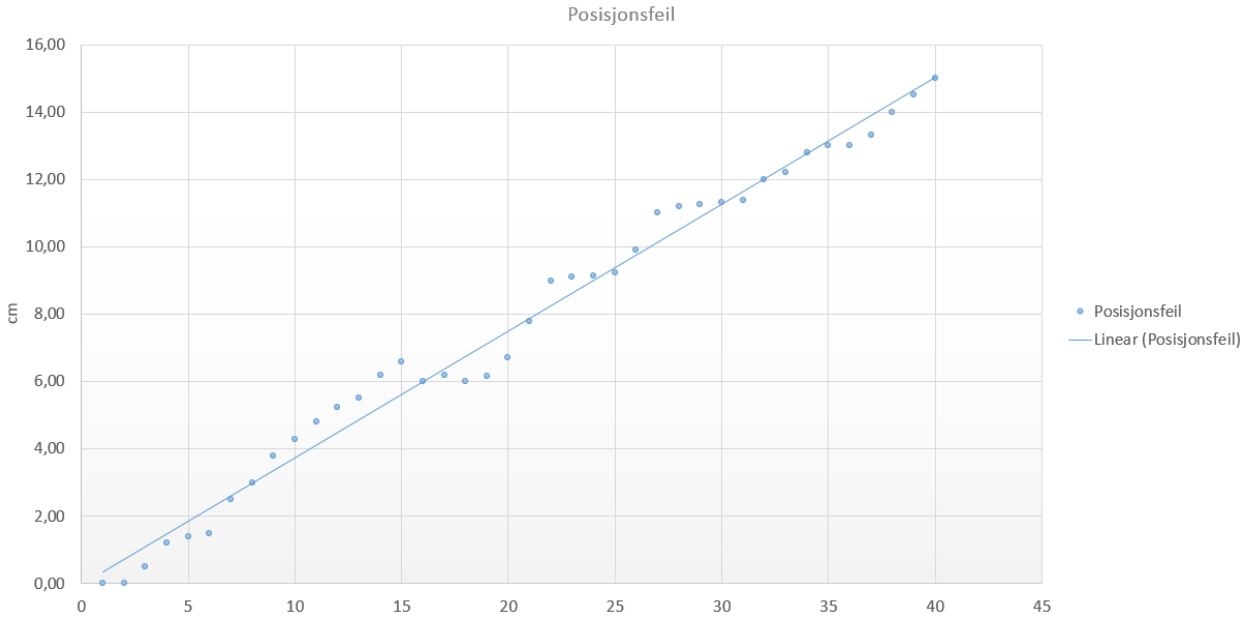
Gyroskopet gir verdier for vinkelhastighet, $\dot{\theta}$ ble også testet og det gav gode resultater ved kjøring i firkant. Til tross for bruk av et extended kalmanfilter eksisterer det fortsatt en bias på gyroskopet når roboten står stille. Plot fra testen av gyroskopet er lagt ved i illustrasjon 30. For å unngå dette vil gyroskopet kun ha innvirkning på posisjonsestimatet når roboten endrer retning og mister med dette en viktig funksjon ved at den ikke kan korrigere for orienteringsendringer når roboten kjører rett fram. Men siden enkoder testen gav gode resultater er det lite som tyder på at roboten vil endre retning mens det kjører rett fram.

5.1.4 Kompass

Kompasset blir grundig gjort rede for i masteroppgave [2] hvor kompassverdien blir vektet og justert. Her blir det lagt fram gode resultater av kompassets funksjon. Kompasset vil ikke bli håndtert nærmere i denne oppgaven.



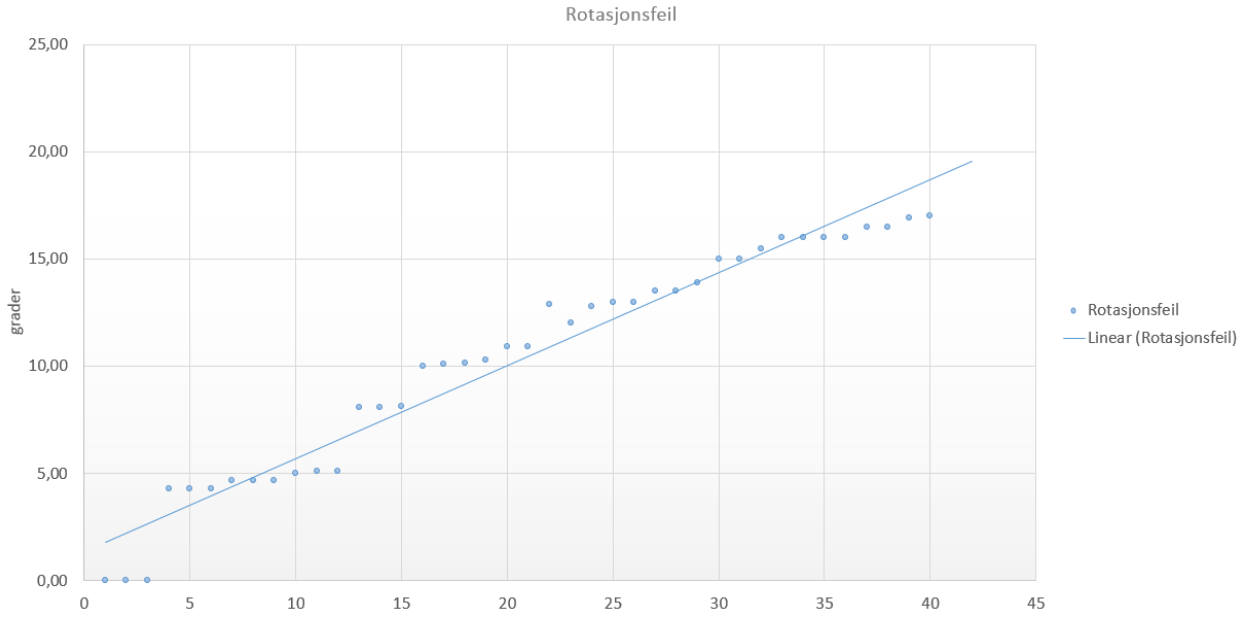
Figur 30: Gyroskopverdier etter firkanttest



Figur 31: Initiell posisjonsfeil med lineær trend

5.2 Initiell feil

For å kunne si noe om den faktiske feilen som opptrer ble den relative feilen mellom robotens faktiske posisjon og den estimerte posisjonen dokumentert ved kjøring i labyrint på B333 med bruk av optitrac motion tracking system. Som følge av problemene med AVR-roboten og at den mistet kontakten med serveren ble dette en utfordrende affære. Til tross for dette ble det produsert resultater som illustrerer feilen. Posisjonsfeilen vises i figur 31 i cm over 40 samplinger, strekningen som ble logget var tilnærmet 7 meter. Den kumulative feilen kommer tydelig fram fra resultatene. En lineær tilnærming tik feilen viser et resultat på $2.14\frac{cm}{m}$. Rotasjonsfeilen vises i figur 32 under samme kjøring. Testen resulterte i en feil for rotasjonen på $2.37\frac{grader}{m}$. Det gjøres oppmerksom på at denne feilen selvfølgelig er avhengig av hvor mange ganger roboten roterer, noe det ikke sies noe om i denne figuren.



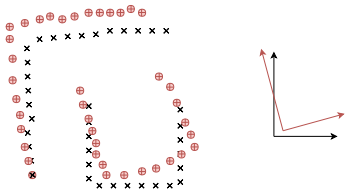
Figur 32: Initiell rotasjonsfeil med lineær trend

5.3 Plan for løsning

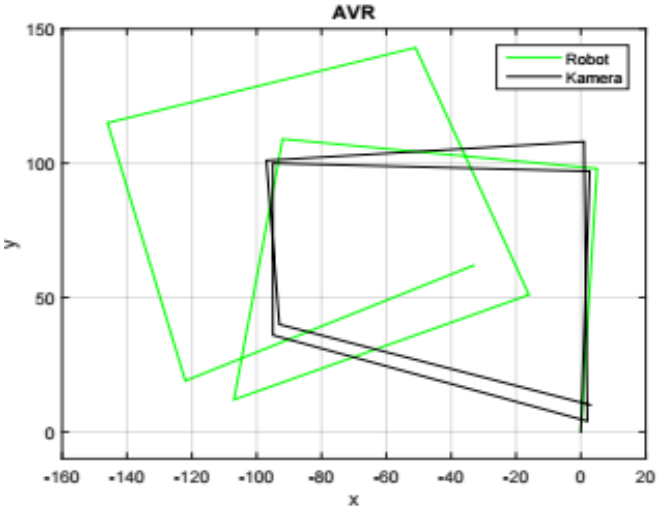
Som nevnt tidligere er det en kjent svakhet for autonome roboter som baserer posisjonsestimatet på odometri og DR at estimatet gradvis blir dårligere over tid. Denne feilen har blitt redusert i masteroppgave [2]. Arbeidet gikk ut på å redusere feilen til robotens sensorer. Basert på testene som er utført tidligere samt logging av sensordata og tester utført i denne rapporten gir sensorene som benyttes av roboten verdier som skal kunne fungere ok.

Basert på at arbeidet som har blitt lagt ned i denne oppgaven baserer seg på sammenligning av ir-data og behovet for introduksjon av absolutt posisjon i systemet var det naturlig velge denne løsningen. Målet med å innføre en ny funksjon er å videreutvikle posisjonskorreksjonen fra dokkingsekvensen og dra nytte av denne funksjonaliteten også under normal drift av systemet. På denne måten vil roboten kunne kontrollere sin posisjon opp mot det den tidligere har sett av omgivelsene. Og det vil være en vital del i å gjøre systemet mer autonomt.

Den ønskede virkningen vises i figur 33 hvor man ser en kort kjøring med rotasjons- og translasjonsforskyving. De svarte kryssene representerer sensordata fra turen ut mens røde sirkler representerer data fra returturen. Ulempen med denne løsningen er at roboten må returnere for å forbedre estimatet samt at målingene er knyttet sammen i et punkt og vil derfor redusere nøyaktigheten av resultatet fra transformasjonsalgoritmen. Transformasjonen som blir kalkulert er vist i de to koordinat-



Figur 33: Ønsket virkning.



Figur 34: Firkanttest utført på B333.

systemente hvor svart representerer originalaksen og det røde viser resultatet etter transformasjonen.

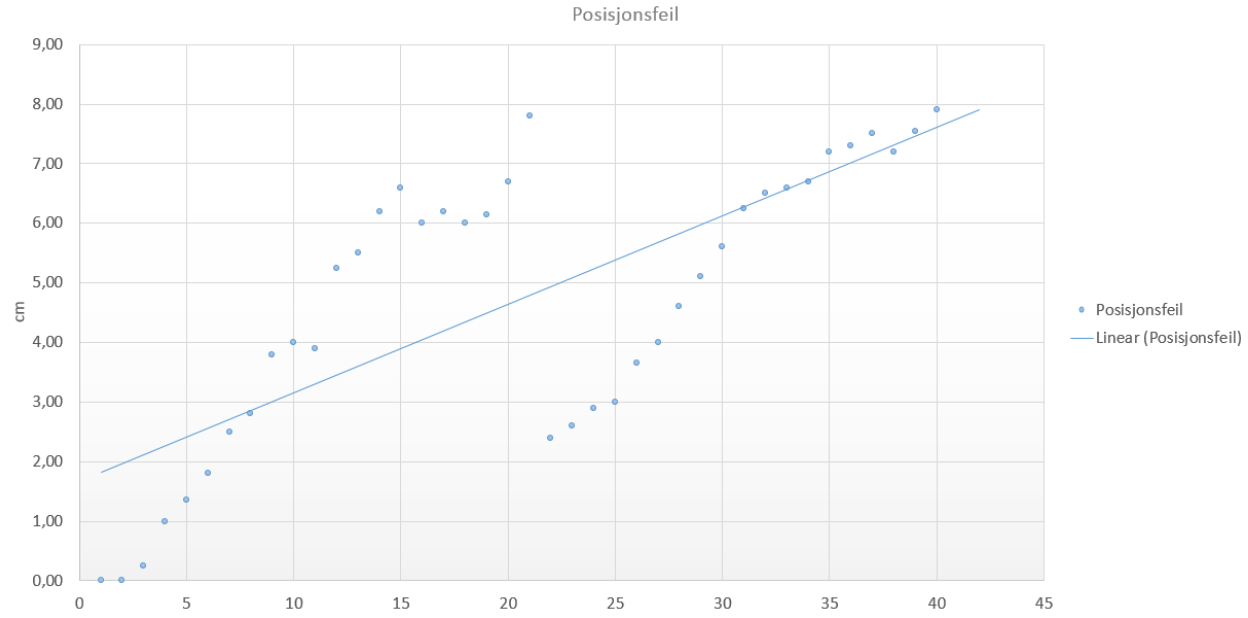
Gjennom å oppdatere robotens posisjon ved å bruke transformasjonsmatrisen vil roboten vil posisjonsestimatet være forbedret.

5.4 Implementering

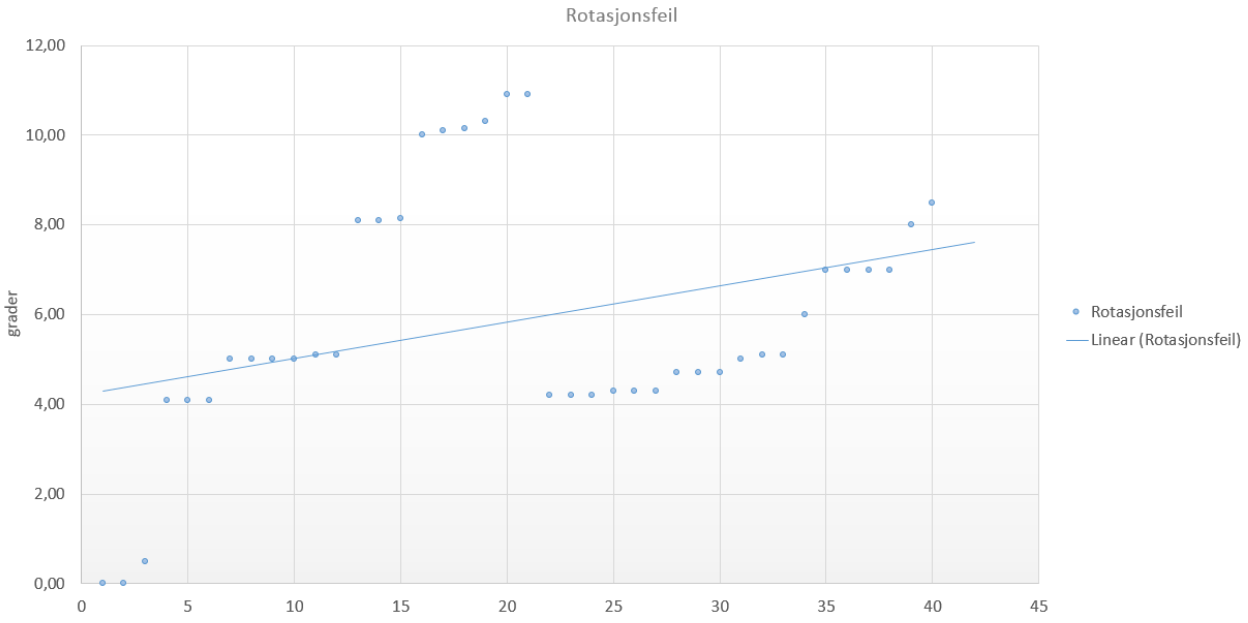
Implementering av dette ble forsøkt ved å øke innsamlingsintervallet for S_{ref} slik at verdiene fra IR-sensoren lagres mens roboten kjører ut. Og stoppes når roboten blir bedt om å returnere, for så å starte innsamlingen av S_{ny} som lagrer data mens roboten navigerer seg tilbake til stasjonen. Denne løsningen viste seg å bli for sårbar for feil i målingen siden siste del av S_{ref} nå vil bestå av målinger som er utsatt for estimeringsfeilene som skal elimineres. De beste resultatene ble oppnådd ved å kun bruke korte innsamlingsintervaller på samme måte som under dokkingen. Når

roboten har returnert til ladestasjonen og utført S_{ny} blir transformasjonsmatrisen generert og satt. Videre består løsningen av omfattende bruk av denne matrisen, den vil nå brukes til å kompansere for feilen generert opp til dette punktet. Transformasjonsmatrisen blir lagret i objektet til den aktuelle roboten slik at den kan benyttes ved behov.

Transformasjonsmatrisen er identisk til den som benyttes ved dokkingsekvensen. Forskjellen er hvor den utnyttes. I dokkingsekvensen var målet å sende roboten til den optimale posisjonen i forhold til ladestasjonen. Nå ønsker man å permanent oppdatere robotens posisjonsestimat til noe som er nærmere den reelle posisjonen. Det ble forsøkt å bruke den eksisterende funksjonen setPosition() fra Robot.java klassen, dette fungerte heller dårlig da det viser seg at denne funksjonen benyttes i flere deler av serveren og den oppdaterte posisjonen derfor ble overskrevet. Løsningen på dette var å oppdatere robotens posisjon i kartet. Dette skjer i MappingController-klassen. Denne transformasjonen må kun skje en gang.



Figur 35: Resulterende posisjonsfeil med lineær trend



Figur 36: Resulterende rotasjonsfeil med lineær trend.

5.5 Tester

Etter at løsningen ble implementert ble testen kjørt på nytt. Under denne testen ble roboten bedt om å utføre transformasjonen etter at sampling 20 ble registrert. Til leserens informasjon ble en serie på femten målinger ekskludert fra plottet da dette er posisjonen under retur og korreksjon av posisjonsestimatet. Fra figur 35 - 36 fremkommer resultatene. Posisjonsfeil er nå redusert til $1.14\frac{cm}{m}$. Og rotasjonsfeil

er redusert til $1.2\frac{grader}{m}$. Korreksjonen vises tydelig idet den relative feilen reduseres fra henhåldsvis 7.9 til 2.4 cm og 10.5 til 4.1 grader. Resultatet stemmer godt overens med hva vi kan forvente av transformasjonensalgoritmen.

5.6 Diskusjon

Siden labyrinten og roboten ikke eksisterer i en perfekt verden vil det uansett hvor godt man filtrerer og optimaliserer bruken av sensorene etter en gitt tid bli en feil på posisjonsestimatet. Når roboten navigerer baserer den alt på startposisjonen. Posisjonen defineres i serveren og knyttes opp mot punkt (0, 0, 90°). På dette tidspunktet vet vi at det er posisjonsestimatet er korrekt men så fort roboten begynner å kjøre vil posisjonsestimatet etterhvert bli dårligere siden punktet roboten videre beveger seg fra etterhvert vil bli feil. Fra testene av IMU og sensorene på roboten viste de ok verdier som burde gi roboten en bedre navigering enn den har nå. Det er rom for å optimalisere bruken av sensorene i robotens software men det vil uansett på et tidspunkt være behov for å eliminere feilen og bekrefte posisjonen mot omgivelsene på samme måte som systemer som navigerer utendørs benytter seg av GPS-posisjonen. Denne posisjonen kan dessverre ikke brukes innendørs så dette må løses på en alternativ måte.

En utfording ved bruk av metoden som her ble valgt er å sette en grense for når man må slutte å generer det opprinnelige kartet. Altså når blir posisjonsestimatet dårlig? Rettere sagt, når blir posisjonsestimatet for dårlig? Jeg har ikke klart å finne noen god løsning på å oppdage denne feilen i sanntid. Så for å kunne si noe om dette ble det utført tester på bevegelseslabben på B333. Plot vist i figur 34. Testen gikk ut på å kjøre roboten i en firkant med størrelse 1 × 1 meter samtidig som robotens estimerte posisjon ble sammenlignet med robotens faktiske posisjon. Resultatet fremgår av figur 34. Posisjonsestimatet er relativt godt fram til den har kjørt om lag 1.5 meter. Etter dette punktet blir posisjonsestimatet ekstremt dårlig og roboten presterer langt fra tilfredstillende og behovet for forbedringer er tydelige. Som følge av at løsningen som implementeres krever at systemet kjøres mens den kartlegger vil det ikke være noe hensikt i å reteste firkantkjøringen da løsningen ikke vil ha noen innvirkning på resultatet.

Metoden som presenteres i rapporten vil slik den fungerer nå korrigere posisjonsestimatet mot den opparbeidede feilen i det genererte kartet. I ettertid vil det være naturlig å tenke at en bedre implementasjon ville vært å korrigert feilen i kartet, som er en direkte følge av feilen i posisjonsestimatet mot den oppdagede feilen. En slik implementering vil føre med seg et behov for å forskyve og rotere hele eller deler

av det genererte kartet. Fordelen med dette vil være at alle robotene vil ha tilgang på et korrigert kart. Dersom man legger inn en algoritme for linjegjenkjennelse av kartet vil det være enklere å koble segmentene hentet inn fra hver robot sammen. Her skal det nok en gang være mulig å benytte seg av transformasjonsmatrisen som er utviklet i denne oppgaven.

Kvantitative målinger av robotens DR-feil er en utfordrende affære. Basert på manglende veldefinerte målingsprosedyrer blir det vanskelig å kalibrere systemet. Mot slutten av arbeidet med dette prosjektet ble det oppdaget at det faktisk har blitt utarbeidet en standard for slike tester som kalles *University of Micigan Benchmark*(UMBmark). Testen krever at roboten kjøres i en 4×4 meter lang firkant med fire 90° rotasjoner. Denne testen skal repeteres fem ganger den ene retningen og fem ganger den andre. På grunn av tidsmangel mot slutten av prosjektet ble ikke denne testen utført. Derfor anbefales det for videre tester utført på roboten at denne metoden benyttes, videre forklaring finnes i Borenstein og Feng [21].

6 Konklusjon

6.1 Konklujon

Fra testene utført i de respektive kapitlene ble det dokumentert gode resultater. Funksjonaliteten til batterimonitoreringen fungerer godt til tross for at det har blitt benyttet en enkel tilnærming til et komplisert problem. Roboten vil nå kunne kommunisere vital data om batterinivået både til en operatør som håndterer brukergrensesnittet eller for automatiske deteksjonen av kritisk nivå i serveren. Dokkingsekvensen som på dette tidspunktet initieres har vist seg å håndtere avvikene som kan forventes etter kartlegging av labyrinter i størrelsesordenen som på dette tidspunktet er aktuell for systemet. Løsningen er satt opp for å håndtere ulike startposisjoner og krever minimalt av initialisering i systemtet. Dette gjør løsningen robust og funksjonell. Forbedringen av posisjonsestimatet vil fortsatt kreve en del arbeid men løsningen som har blitt pressentert i denne oppgaven reduserer feilen noe i tillegg til å introdusere et mål for absolutt posisjonering. Roboten navigerer langt fra perfekt men ved å kunne oppdatere for feil i estimatet vil systemet nå kunne korrigere seg selv. Helheten av løsningene implementert i denne oppgaven har i stor grad forbedret systemets autonome egenskaper.

7 Videre arbeid

7.1 Øke oppløsning på kartet

Dersom man vil forbedre nøyaktigheten til dokkingsekvensen vil det absolutt være en fordel å øke oppløsningen til kartet som genereres. Det har til nå vært satt til 2 cm per rute, dette ble i denne oppgaven justert ned til laveste mulige verdi for den aktuelle løsningen som er 1 cm. For å få til dette må hele kartet i serverapplikasjonen restruktureres. Når dette gjøre vil det være en fordel å tenke på at kartet burde ha korreksjonsfunksjon som kan oppdatere feil i det gererte kartet i sanntid og kunne koble sammen segmentene generert fra de ulike robotene, se neste punkt.

7.2 Benytte iterativ closest point

Det er mulig å overlappe to forkjellige kart ved å bruke en metode som kalles iterative closest point. Metoden ligner litt på den som ble implementert i denne oppgaven men behandler større sett av data og vil i dette tilfelle kunne brukes på hele kartet. Metoden blir ofte brukt i situasjoner da området allerede er kjent og man kan lage en modell av strekningen roboten skal kjøre men den kan tilpasses til det aktuelle formålet.

7.3 Gjenkjenne landemerker i labyrinten for forbedring av posisjonsestimat

Det er mulig å implementere *Bayesian Landmark Lerning*-algoritme i systemet. Denne vil gi roboten muligheten til å selv lære seg å gjenkjenne landemerker rundt om i labyrinten samt å lære seg måter å gjenkjenne dem på. Når roboten blir gitt et datasett fra IR-sensorene som er knyttet opp mot punktet den befinner seg. Datasettet trekker ut sammenhenger mellom sensoren og kartet som er generert og og setter opp en kunstig nevronetverk som gjenkjenner landemerker i labyrinten. Dette

er mulig ved å minimere posisjonsestimatet.

7.4 Implementere Arduino-kode på AVR

Masteroppgaven til Jørund Amsen har restrukturerer navigasjonene på roboten. I et forsøk på å forbedre navigasjonen på AVR kan det være en idé å skrive inn denne koden også på AVR. Den nye løsningen har vist seg å være svært nøyaktig.

7.5 Fusjonering av robotene

Siden løsningen som er presentert i denne rapporten er implementert i serveren er det en hel rekke muligheter når det kommer til samhandling. Da spesielt i arbeidet med å forbedre posisjonsestimatet. Et av de store problemene når robotene kartlegger sammen er at de blir forskøvet og vridd i ulike retninger. Dersom man kan benytte seg av transformasjonsalgoritmen og justere inn en andre roboter så de blir korrekt posisjonert og orientert i forhold til de andre vil dette problemet være løst. Dette kan gjøre ved at ulike sett lagres i en flerdimensjonell referansematrise lagres i serveren og fungerer som "safe zones" for systemet. De forskjellige robotene kan da benytte seg av disse punktene og kan med jevne mellomrom korrigere for feil i posisjonsestimatet.

Disse sonene kan enten settes opp av en eller flere av robotene eller av kameraet som er planlagt implementert på en drone i masteroppgaven til Bjørnsen [22] fra 2017. Slik jeg forstår det er systemet ment å fortsatt kunne kartlegge som denne dronen ikke vil se. Men det vil da være mulig å ha flere punkter som de bakkenavigerende robotene kan korrigere seg etter. Dette krever at kartene settes sammen og at kartleggingen fra dronen blir gitt globale koordinater i systemet.

Bibliografi

[1] Lars Marius Strande. Fjernstyring av autonome roboter. Technical report, Institutt for teknisk kybernetikk - NTNU, 2016.

[2] Erlend Ese. Sanntidsprogrammering på samarbeidande mobil-robotar. Technical report, Institutt for teknisk kybernetikk - NTNU, 2016.

[3] A. Haugedal. Forbedring av de autonome egenskapene til en lego-robot. Technical report, Institutt for teknisk kybernetikk - NTNU, 2008.

[4] Feng Lu and Evangelos Milios. Robot pose estimation in unknown enviroments by matching 2d range scans. Technical report, Reliable Software Technologies, 1999.

[5] Chris Evans. Mars microrover power subsystem, 2013. http://mars.jpl.nasa.gov/MPF/roverpwr/power.html.

[6] Ian Kelly. *SlugBot: A Robotic Predator in the Natural World.* Compal Hall, Oita, Japan, 2002.

[7] Morten Andrè Kristiansen. Fjernstyring av lego-robot. Technical report, Institutt for teknisk kybernetikk - NTNU, 2009.

[8] Eirik Thon. Mapping and navigatopn for collaborating mobile robots. Technical report, Department of Engineering Cybernetics - NTNU, 2016.

[9] Øyvind Ulvin Halvorsen. Collaborating robots. Technical report, Department of engineering cybernetics - NTNU, 2014.

[10] Trond Kåre Homestad. Fjernstyring av legorobot. Technical report, Institutt for teknisk kybernetikk - NTNU, 2013.

[11] Thor Eivind Svergja Andersen and Mats Gjerset Rødseth. System for self-navigating autonomous robots. Technical report, Department of Engineering Cybernetics - NTNU, 2016.

[12] Masakui Yoshi and Ralph J. Brodd. *Lithium-Ion Batteries*. Springer, 2009.

[13] Cornell Computer Science. The Singular Value Decomposition. `http://www.cs.cornell.edu/courses/cs322/2008sp/stuff/TrefethenBau_Lec4_SVD.pdf`, 2016. [Online; accessed 5.mai, 2017].

[14] Olga Sokrine-Hornung and Michael Rabinovich. Least-squares rigid motion using svd. Technical report, Department of Computer Science, ETH Zurich, 2017.

[15] Paul J. Basel and Neil D. McKay. A method for registration fo 3-d shapes. *Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[16] J. Borenstein and H. R. Everett et al. Mobile robot positioning: Sensors and techniques. Technical report, Center for Intelligent Machines, McGill University, 1996.

[17] I. M. Rekleitis, G. Dudek, and E. Milios. Multi-robot exploration of an unknown environment, efficently reducing the odometry error. Technical report, The University of Michigan, Advanced Tech Lab, 2002.

[18] S. Thrun. Finding landmarks for monile robot navigation. *ICRA*, 1(2):958–963, 1998.

[19] Bluetooth SIG Inc. Specification of the bluetooth system. 2010.

[20] B. Barshan and H. F. Durrant-Whyte. An intrtial navigation system for a mobile robot. Technical report, Departement of Engineering Science, Univercity of Oxford, 1993.

[21] J. Borenstain and L. Feng. Measuremet and correction of systematic odometry errors in mobile robots. Technical report, Advanced Tech Lab, Univercity of Micigan, 1996.

[22] Kristian Bjørnsen. Mapping a maze with a camera using rasbarry pi robots. Technical report, Department of Engineering Cybernetics - NTNU, 2017.

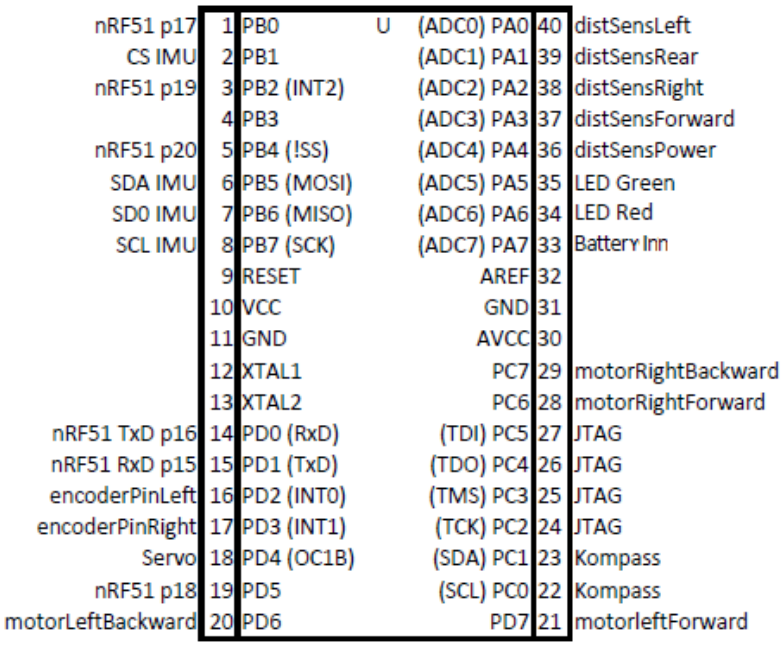
Vedlegg

A Innhold DVD

1. SSNAR
2. SSNAR, Strande 2016
3. SSNAR, Thoen 2016
4. AVR
5. AVR Old Protocoll
6. AVR Ese
7. Dokumentasjon batterimonitorering
8. Dokumentasjon dokkingfunksjonalitet
9. Dokumentasjon test av alle roboter
10. Figurer
11. Prosjektplan
12. Tidligere oppgaver
13. Literatur

B Pinout mikrokontroller

ATmega1284 pinout



C Sammenføyning av løsninger

Mot slutten av prosjektet gikk det med mye tid på å flette sammen løsningen vi hadde utviklet sammen til et fungerende system. Dette omfatter kildekoden til både serveren og roboten. Det ble tidlig bestemet at vi skulle benytte oss av github slik at alle kunne holde sin kode oppdatert til enhver tid. Dette gikk ikke helt etter planen og etter noen uker endte vi opp slik vi ikke skulle og skrev på hver vår versjon. Årsakene for at det gikk slik er vanskelig å si, men alle må ta sin del av skylden for at vi ikke fikk til bruken av github på en god måte. Sammarbeidet med de andre masterstudentene har ellers gått fint, det har vært lav terskel for når vi kan spørre hverandre om råd og hjelp til teoretiske og praktiske utfordringer vi har støtt på underveis. Resultatet av den misslykkede introduksjonen av github ble at den siste uken ble brukt til å reimplementere koden. For meg personlig ble dette en tidkrevende affære da jeg måtte gjøre opp igjen mye av batterimonitoreringen og kommunikasjonen da protokollen mellom serveren og roboten var forbedret av Lien. Leseren blir gjort oppmerksom på at testene som har blitt presentert i rapporten ble utført med den gamle kommunikasjonsprotokollen, men funksjonaliteten har ogsp blitt testet etter sammnflettingen og det fungerer fortsatt bra. Endringene som har blitt gjort for å få det hele til å passe sammen skal ikke spille inn på funksjonalitet eller resultatene. Endringene ligger også på et detaljnivå som uansett ikke har blitt pressentert i rapporten.

D Problemer med AVR

Problemene med roboten har blitt forsøkt forbedret. Feilsøkingen har vært vanskelig da feilen oppterer usystematisk og uavhengig av kildekoden som benyttes. Feilsøking og debugging har blitt utført gjennom NetBeans(server), Atmel Studio(robot) og J-Link Rtt Viewer V6.16 (dongel). Endringene som har blitt utført på batteripakken har blitt forsøkt koblet ut uten at det har blitt noe bedre. Slutningen jeg har kommet til er at det må ha oppstått en feil med hardware på roboten, det kan være en dårlig tinnbro eller at en komponent har dårlig kontakt. Det anbefalest at videre feilsøking vil inkludere kartlegging av elektriske kretser ved bruk av oscilloskop.

E User manual for docking sequence

The returning and docking sequence enables the robot to connect to the charger. This ability can be manually engaged by pressing the Home button in the Robot-specific information interface. The current voltage supplied by the battery can be seen in this interface as well. The application is programmed to force the robot to return if the voltage level dropps beneath 9.9 volts. When the robot has successfully docked to the charger the mapping can be restarted by pressing the Resume button.