NTNU
Det skapande universitetet

Sanntidsprogrammering på samarbeidande mobil-robotar

Erlend Ese

Master i kybernetikk og robotikk
Innlevert: juni 2016
Hovedveiledar: Tor Engebret Onshus, ITK

Noregs teknisk-naturvitskaplege universitet
Institutt for teknisk kybernetikk

Samandrag

Kommunikasjonsmodul

Ny kommunikasjonsmodul er utvikla og implementert, den nye løysninga er svært stabil, og vi kan sende og ta i mot meldingar frå fleire robotar samtidig, i sanntid.

Løysinga byggjer på BLE, eller Bluetooth Smart protokollen, som er den nyaste. Den er optimalisert for hyppige små meldingar. Tidlegare gjekk det ikkje an å feilsøke på kommunikasjonen, og den slutta å fungere sporadisk. Mulighet for programmering på modulen i seg sjølv gir oss utviklingsmuligheter, som til dømes maske-nettverk framfor stjerne, eller implementering av wi-fi, noko som kan føre til "uendelig" rekkevidde for n antal mobil robotar. I tillegg har vi ein godt utvikla protokoll i botn, som industrien satsar på. Måten kommunikasjonen er implementert saman med roboten på gjer at vi har muligheit å få roboten til å returnere før signalet vert då dårleg at vi mister kontakten heilt.

Testinga verifiserer at ytinga og redundansen til kommunikasjonsmodul er betre enn tidlegare, og det var riktig å oppgradere.

Oppgraderingane til AVR roboten

Det er gjort fleire fysiske forbetringar på AVR roboten, der dei viktigaste er nytt kretskort, bytte av mikrokontroller, forbetring av sensor-tårn, bytte av hjul og motorspenning.

Det nye kretskortet tar ut fleire av IO-ane frå mikrokontrolleren, og har i tillegg lagt dei ut slik at ein kjem til med målepinnar. Det er montert ein rekke kondensatorar for å beskytte komponent mot støy frå servo og IR sensorar. Banene er lagt på ein slik måte at det ikkje interfererer med viktige signal på kretskortet. For å auke kapasiteten på roboten er det bytta mikrokontroller frå ATmega32 til ATmega1284p. Fysisk er mikrokontrolleren heilt lik, men den har mykje meir program- og data minne, samt fleire alternativ på IO-ane. Sensor-tårnet har blitt forbetra ved å bytte ut dei gamle sensorane til ein nyare modell som har eit breiare måleområde, dei er skrudd fast med maskinskruer, og tilkoplinga er bytta til ein plugg med rett tverrsnitt. Bytte av hjul og motorspenning gjer at roboten kjører nesten tre gonger raskare enn den gjorde tidlegare. Dei generelle forbetringane på konstruksjonen til roboten gjer at vi kan demontere toppen og enkelt komme til for måling, noko som er viktig for ein prototype som skal vidare utviklast.

Ny programvare

Det er utvikla nytt skreddarsydd sanntids-program til robotane. No kjører, skanner, kontrollerer-og-estimerer posisjonen og kommuniserer dei samtidig.

Den gamle programvaren er gjennomgått fullstendig, og det som har fungert tilfredsstillande er tatt med vidare. Deler som ikkje fungerte, eller som kunne vore betre, har blitt gjort på nytt med ei anna tilnærming. Det er fokusert på å gjere koden modulbasert og meir forståelig. FreeRTOS gjer at vi kan kjøre funksjonar som tråder på roboten, og slik multi-taske fleire oppgåver. Eigne trådar for oppgåver som ikkje er avhengig av kvarandre er oppretta, og kombinasjonen av dette gjer at roboten kan utføre arbeidet meir effektivt enn tidlegare.

Det er gjort målingar og utrekningar for å dokumentere teoretisk kapasitet for reknekraft på robotane, og dei viser at vi har med visse antagelser fortsatt har mykje kapasitet til overs. Testinga antyder ingen sanntidsproblem.

Ny posisjonsregulator og estimator

Posisjoneringa til roboten er forbetra ved å implementere ein ny regulator med referansemodell og utvide eksisterande estimator med fleire sensorar og kalman-filter.

Tidlegare var det eit problem med ein aukande bias på posisjonsestimatoren. Under testing vart det avdekt at det var hovudsakleg estimatet for headinga til roboten som var årsaken til dette. For å forbetre estimatet for headinga vart det implementert nye sensorar til å estimere og korrigere headinga dette ved bruk av eit kalman filter. Bruk av elektromagnetisk kompass innandørs er ei utfordring då det er mange kilder for forstyrrelsar, og det vart gjort ein del tilpassingar for spesialtilfelle rundt dette. Desse såg ut til å fungere bra.

Testing viser og at til tross for raskare rørsle er posisjoneringa blitt betre.

Arduino roboten

Programmet som var skreven til AVR roboten er implementert og tilpassa til å kjøre på den nye Arduino roboten.

Hovud-forskjellen i koden ligg i ei fil. Roboten oppfører seg akkurat som AVR roboten, med unntak av motorstyringa som har fått ein ekstra regulator. Motorane til Arduino roboten er svært kraftige, og måtte girast ned. Etter ned-giringa vart senter for rotasjons-punktet på motorane forskyvd. Server programmet tek utgangspunkt i at sensor-tårnet er plassert i senter på roboten, då dette ikkje stemmer vart plotta litt feil ved rotasjon. Ein enkoder var ødelagt i sluttfasen av prosjektet, så det var ikkje mulig

å få teste anna enn estimatet til headinga på denne roboten. Ein av avstands-sensorane slutta å virke i sluttfasen. Roboten tek i mot meldingar og sender målingar i sanntid som forventa. Estimatet på headinga, og målingane frå roboten viste gode resultat under testinga.

Kan derfor slå fast at implementasjonen av koden på Arduin roboten fungerer, og dersom dei øydelagde komponenta vert bytta, kan dei linjene i programmet som tilhøyrer motorstyringa bli kommentert inn igjen, og Arduino roboten vil fungere like bra som AVR roboten.

Verifisering

Ytelsen til systemet som heilheit er verifisert ved å kartlegge ein labyrint på ca 4.4m^2 i eit laboratorie der vi fekk samanlikna estimert posisjon med ekte posisjon ved hjelp av eit tracking system med mange kamera.

Testinga viste gode resultat og robotane kan samarbeide om å generere eit kart i sanntid på serveren. Robotane får feil i estimata etter kvert, over tid vil kalmanfilteret og kompasset på roboten føre til at headinga blir korrigert, gitt at roboten ikkje alltid er i områder med magnetiske anomali. Tidlegare er det dessverre er det ikkje utført nokon testar vi kan samanlikne med som er større enn ca 1m^2, men vi har dokumentert resultata våre godt nok til at framtidig arbeid kan gå tilbake og samanlikne med det vi har gjort.

Hausten 2015 prøvde eg å køyre ein del testar med den gamle versjonen av systemet. Det tok meir tid å prøve å få serveren og roboten til å kommunisere enn det gjorde å gjennomføre sjølve testane.

Under testinga med det nye programmet no var det lett å slå fast at det nye er mykje meir stabilt og påliteleg enn det gamle programmet som kjørte i matlab. Saman med den oppgraderte kommunikasjonsmodulen har vi no eit system som virka kvar gong, og dersom det oppstår situasjoner går det an å finne årsaka fort og få gjort noko med det.

Abstract

Communication module

A new communication module was developed and implemented. The solution is redundant and allow us to send and receive messages from multiple robots, in real-time.

The solution is built on the state-of-the-art Bluetooth Smart protocol. It is optimized for small periodic messages. The previous communication module did not allow for debugging and would randomly stop working. Programmable communication module yields new possibilities. With the industrial standard protocol in the bottom, the groundwork for further improvements has been laid. Mesh network topology, or implementation of wi-fi yields theoretical infinite range for n mobile robots. The way we have implemented the communication module with the robot guarantee that we can call back the robot in case of bad reception.

AVR Robot upgrades

Multiple enhancements to the AVR robot has been applied, in more detail, new printed circuit board, upgraded micro-controller, improved sensor tower, larger wheels and higher motor voltage.

The new printed circuit board breaks out more IO from the micro-controller, and has probing areas for measurement tools. The circuit has been protected by several capacitors to protect components from the noisy sensors and servo. The signal traces has been placed to avoid signal interference from voltage traces. To increase capacity the micro-controller was changed from ATmega32 to ATmega1284, which is physically identical, but with notably more program and data memory, and more peripherals for the IO. The sensor tower has been improved by replacing the old IR sensors with a new model with a wider measurement area. They have been mounted by using proper machine bolts, and incorrect connectors has been replaced. Larger wheels and higher motor voltage made the robot drive almost three times faster than before. Other upgrades made it possible to detach the top to access measurement points, which is useful for prototyping.

New software

New software tailored for real-time operation has been created. The robots now drive, scan, control and estimate their pose, and communicate simultaneously.

The previous software has been reviewed thoroughly, satisfactory functionality has been kept, while improvements has been done with a new perspective on the rest. FreeRTOS allow us to run multiple functions as threads on the robot, and thus allow us to multi-task. Separate threads was created for independent tasks, allowing the robot to perform more effective than before.

Measurements and calculations to document theoretical capacity on the micro-controllers show that with certain assumptions we still have a lot of capacity to go. Live testing suggest no real-time problem on the robots.

New pose controller and estimator

The positioning of the robot was improved by implementing a new control algorithm with a reference model, and expanding the existing pose estimator, by using more sensors and a kalman-filter.

Previously there was a problem with an increasing bias on the pose estimator. Testing uncovered that the problem seemed to be most drastic for the heading estimates. To improve the heading estimated, new sensors were implemented to estimate and correct the heading with a kalman-filter. Using an electronic compass indoors is a challenge due to magnetic anomalies, so a special algorithm was developed to avoid these.

Arduino Robot

The new real-time software was implemented on a new Arduino. The main difference from the software running on the AVR robot is located in one file. The robots have identical behaviour, except for a new regulator implemented in the Arduino robot. The motors in the Arduino robot were very powerful, to cope with this we geared them down. After this the centre of rotation were shifted, causing a disturbance in the plot during rotation. Towards the end-phase a sensor for the wheel encoder broke, and an IR sensor stopped working. It was not possible to test the pose estimator thoroughly. Despite broken sensors, the robot receives and executes messages, and send measurements in real-time as expected.

Verification

The systems performance as a whole was tested by creating a labyrinth of approximately 4.4m^2 in a laboratory where we could compare estimated pose with real measurements tracked from a multiple-camera system.

Even though the project report is in Norwegian, all the software and documentation are written in English.

Føreord

Alle studentar ved Institutt for Teknisk Kybernetikk (ITK) på Noregs Teknisk-Naturvitskaplege Universitet (NTNU) skal i 10. semesteret gjennomføre ei masteroppgåve, der varigheita er 20 veker, tilsvarande 760-960 timar per student. Oppgåvene kan utførast individuelt eller i grupper.

Denne oppgåva er gjennomført individuelt, men arbeidet skal samansveisast med to andre grupper som arbeider parallelt med andre delar av systemet.

Prosjektet starta i 2004 og har vore vidareutvikla i fordjupingsprosjekt og masteroppgåver frå forskjellige studentar. Den overordna hensikten med prosjektet er å navigere og kartlegge eit ukjent område ved hjelp av ein robot, og i dei siste åra, fleire robotar som samarbeider.

For min del starta prosjektet hausten 2015 der eg greidde ut muligheiter for sanntidssystem til robotane, medan eg satt meg inn arbeidet som hadde blitt gjort tidligare. Masteroppgåva byggjer på dette fordjupingsprosjektet og tek arbeidet med robotane vidare i form av forbetringar og vidare utvikling.

Takk til Joakim Tønnessen fra Nordic Semiconductor for hjelp med kommunikasjonsmodulen.

Takk til dei som har arbeidd med andre delar av prosjektet, Eirik Thon, Thor Eivind Svergja Andersen og Mats Gjerset Rødseth. God kommunikasjon og samarbeid har vore nøkkelfaktoren til suksess.

Spesielt takk til Tor Onshus for ei stimulerande prosjektoppgåve og god rettleiing.

Erlend Ese
Trondheim, juni 2016

Innhald

Figurar

Tabellar

Kodesnuttar

Akronym

AS6 Atmel Studio 6.2.

BLE bluetooth low energy.

EKF extended kalman filter.

FIFO first-in first-out.

I^{2}C inter-integrated circuit.

IMU inertial measurement unit.

IR infrared.

ISR interrupt service routine.

NFC near field communication.

PWM pulse width modulation.

RTOS real-time operating system.

SPI serial peripheral interface.

TWI two wire interface.

USART Universal synchronous/asynchronous receiver/transmitter.

WCET worst-case execution time.

Ordliste

AVR Robot I tidlegare prosjekt beskriven som IR Robot. Denne vert styrt av ein AVR 8bit mikrokontroller.

Arduino Robot Denne vert styrt av ein AVR 8bit mikrokontroller på ein Arduino Mega. Ny i 2016.

NXT Robot Styrt av ein NXT Brick, som er ein Lego Mindstorms kontroller.

EV3 Robot Styrt av ein EV3 Brick, som er ein Lego Mindstorms kontroller. Ikkje vidare utvikla våren 2016.

Dongel nRF51 development kit, 2.4ghz radiosendar i form av ein USB dongel.

Kartlegging Automatisert kjøring av roboten, den kjører og kartlegger eit område ved hjelp av instruksar frå serveren.

Pose Engelsk for positur, samlebegrep for posisjonen og retninga til eit objekt.

Scheduler Oppgåveplanleggar til sanntidsystem, "bestemmer" kortid tasker skal få kjøre.

Sensor-tårn Avstands-sensorane er montert på ein servo, eller servo-liknande konstruksjon.

Sentral Kommunikasjonsdongelen på serveren som kommuniserer med robotane.

Server Maskina som styrer robotane.

Task I FreeRTOS er kvar tråd for utføring namngitt oppgåve, eller "task".

Del I

Introduksjon og bakgrunnsmateriale

Kapittel 1

Introduksjon

1.1 Motivasjon og bakgrunn

Robot-samarbeid er ei fundamental utfordring innan robotikk, der ein utforskar moglegheitene til å fordele arbeidsoppgåver på n robotar. Bruksområde kan vera alt frå liv reddande person-søk til optimal plenklypping.

I 2004 starta prosjektet på NTNU der ein robot styrt av ein RXC LEGO mindstorms kontroller vart bygd. Den vart seinare oppgradert med ein AVR mikrokontroller. AVR roboten har vorte utvikla i fleire år av forskjellige studentar for å forbetre eller utforske nye moglegheiter. Systemet vart utvida med ein til robot, styrt av ein NXT frå LEGO Mindstorms. Robotane vert styrt sentralt frå ein datamaskin med programvare med algoritme for samtidig lokalisering og kartlegging.

1.1.1 Tidlegare arbeid

AVR Roboten vart først først bygd av Skjelten (2004). Ein LEGO Mindstorms NXT robot vart inkludert i systemet av Bakken (2008). Prosessen er oppsummert frå 2004 til 2012 i prosjektrapportane til Tusvik (2009) og Homestad (2013).

Prosjektet vart så vidareutvikla skuleåret 2013-2014 av Halvorsen. Halvorsen (2013) utvida programmet til å kunne styre to robotar paralellt ved å kjøre to instansar matlab som kommuniserte ved hjelp av TCP/IP. Halvorsen (2014) utvikla og implementerte ein algoritme for samarbeid mellom to robotar ved hjelp av ein navigasjonsdel og ein baneplanleggingsdel.

Hausten 2015 vart ein tredje robot, LEGO Mindstorms EV3 av Steuper (2015), ved bruk av ein offisiell matlab support pakke. På grunn av problem med IR sensorane vart ikkje implementeringa fullført.

Ese (2015) bygde om NXT roboten og vurderte fleire real-time operating system (RTOS). FreeRTOS vart implementert på AVR roboten, med eit minimum av endringar i den eksisterande programvaren.

1.2 Mål

LEGO robotane utfører oppgåver i sekvensiell rekkefølgje. AVR roboten vart testa med FreeRTOS hausten 2015 av Ese (2015), men programvaren vart ikkje omskreven til å ta nytte av dette. Det vart også avdekt problem med kommunikasjonen, samt posisjonsestimatoren til robotane.

Halvorsen (2014) foreslo: "*If the robots could broadcast their position continuously, then a more flexible and autonomous navigation algorithm could be implemented.*"

Følgjande målsettingar vart satt opp:

- Lage ny sanntidstilpassa løysing for trådlaus kommunikasjon
- Utvikle ny sanntids-programvare til robotane som gjer at dei multi-tasker
- Forbetre posisjonsestimat lokalt på robotane

Dette gjer at robotane skanner omgjevnadane, estimerer posisjonen sin og kommuniserer med serveren i sanntid, kontinuerlig.

Samtidig vert ein fjerde robot vert bygd av Rødseth & Andersen (2016).

For å nå hovudmåla vart det satt følgjande delmål:

1. Få nødvendig oversikt over alternativ for trådlaus kommunikasjon
2. Lag kretskort til robotane for implementering av nye sensorar og kommunikasjon
3. Skriv nytt RTOS program på AVR roboten med FreeRTOS i bunn
4. Utvikle ny posisjon-regulator og -estimator med fleire sensorar
5. Implementer det nye programmet på ein ny Arduino robot
6. Konkluder resultata ved testing
7. Dokumenter arbeidet som er gjort og foreslå vidare arbeid

Føremålet med denne rapporten er å gi lesaren innsikt i korleis måla vart nådd, resultata, og ein beskriving til kva og korleis vidare arbeid kan utførast.

1.3 Avgrensing

Arbeidet i denne oppgåva avgrensar til programmeringa på AVR og Arduino robotane, og kommunikasjons-modulen. Programvaren til serveren med navigasjon, kartlegging og simulator vert ikkje endra eller forbetra i denne oppgåva. LEGO Mindstorms robotane vert ikkje forbetra i denne oppgåva, men det vert utvikla ein prototyp av eit utvidelseskort til robotane med den nye løysinga.

1.4 Samarbeid med Andersen, Rødseth og Thon

Denne oppgåva har gått samtidig med to andre oppgåver.

Thon (2016) utvikla ei algoritme for kartlegging og navigasjon, og ein ny simulator til systemet.

Rødseth & Andersen (2016) utvikla ny server applikasjon som tar seg av grensesnittet mellom robotar, kartlegging-algoritmen og brukar, bygde ein ny Arduino robot og var med å programmere kommunikasjons-modulen på server sida. Mot sluttfasen skal oppgåvene samansveisast for å danne eit komplett system for samarbeidande robotar brukt til navigasjon og kartlegging.

1.5 Om rapporten

Rapporten er skreven i LaTeX og PDF dokumentet har høgoppløyste bilete og vektorfigurar. Alle referanser og kryssreferanser fungerer som hyperlinker.

Rapporten har 17 kapittel fordelt på 5 hovuddelar. I tillegg er det ein vedlagt DVD. Referanselista ligg heilt på slutten av dokumentet.

Del ein: Vi ser på utgangspunktet til prosjektet, mål og avgrensing i kapittel 1. Aktuell teori og teknologi som er brukt i kapittel 2 og utstyret som er brukt i kapittel 3.

Del to: Vi ser påarbeidet som er blitt utført, og dokumenteringa rundt dette.

- Kapittel 4 gjeld utvikling og bruk av kommunikasjons-modul
- Kapittel 5 til 9 gjeld sensorane som er brukt, det nye sanntids-programmet som er implementert, og alt rundt dette
- Kapittel 10 til 12 er spesifikt om robotane

Del tre: Vi ser gjennom testinga som vart utført av systemet i sin heilheit, og resultata etter desse. I kapittel 13 ser vi på resultata til AVR roboten. I kapittel 14 ser vi på resultata til Arduino roboten, i kapittel 15 ser vi på resultata til den nye kommunikasjons-modulen, og i kapittel **??** står det kort om robotsamarbeid.

Del fire: I kapittel 16 drøfter eg løysingane som vart valt, eventuelle svakheiter ved desse og kjem med forslag til vidare arbeid i kapittel 17.

Del fem: Vedlegg. Sjå side 5 for beskriving av vedlegga.

1.5.1 Vedleggsbeskriving

A - Beskriving av filvedlegg

Beskriving til det som ligg på DVDen

B - Hurtigguide til robotane

Korleis bruke robotane

C - Kretskort

Bilete av kretskorta som vart utvikla. Designfilene ligg på DVDen

D - Pinouts

Oversikt over kva IO på robotane er brukt til

E - Importering av Keil uVision5 prosjekt

Ein "How-to" eg skreiv for å importere prosjekt som ikkje er skreven i keil uVision

F - Forslag til motorstyring

Eit forslag til korleis styre motorane med PWM og eit to-polt relè

G - Straummålingar

Straummålingar som vart gjort av AVR og Arduino robotane i forskjellige tilstander

H - Tidsbruk

Oversikt over tidsbruk for delmåla i oppgåva

I - Abot

Det vart begynt på å bygge ein til robot, men den vart ikkje ferdig. Opplysningane om denne hamna derfor i vedlegg

J - Utrekningar for responstidsanalyse

Utrekningane som vart gjort for å rekne ut eit teoretisk kapasitetsestimat.

Kapittel 2

Teori

2.1 Beskriving av systemet

Robotane kommuniserer over Bluetooth til ein server. Dei tek i mot meldingar med instruksar som bestemmer kva arbeidsoppgåver som skal utførast. Under kartlegging får robotane melding om kvar dei skal køyre, og melder tilbake sin eigen posisjon, og resultatet frå avstandssensorar som er montert på ein servo som roterer.

Serveren genererer plott og gir vidare instruksar til kor robotane skal kjøre, i form av ein relativ vinkel og avstand ($\theta_{sp}, Dist_{sp}$).

Robotane er avbilda på side 54 og 59.

2.2 Tilstand ved oppstart

AVR roboten kjører all koden med FreeRTOS i bunn. Ein task køyrer mesteparten av instruksane som skal utførast, medan ein anna task styrer vinkelen på sensor-tårnet. Ein interrupt rutine styrer kjøringa. Programmet er ineffektivt då det ikkje er utvikla for å bli kjørt ved hjelp av eit RTOS. I tillegg viser videoar frå testing gjort i 2013 og 2014 at det er eit sanntidsproblem med sensortårnet på roboten. Den er i stand til å kjøre og kartlegge ved hjelp av ein server, men har problem med å estimere posisjonen etter rotasjon. Kommunikasjonen feiler regelmessig, og går via ein Bluetooth 1.0 modul utan moglegheiter for programmerin eller debugging.

Serveren som styrer robotane er basert på ein matlab toolbox som har blitt mykje endra over åra. Konsekvensen er innfløkt kode, mykje feilmeldingar som ikkje gir meining, og eit program som har massevis av delvis fungerande funksjonalitet. Karta som vert laga i programmet liknar ikkje spesielt på dataen som blir mata inn, tilsynelatande fordi programmet prøver å gjere alle målingar om til rette veggar. For at robotane skal kunne samarbeide må det kjørast ein instans av matlab per robot, noko som ikkje skalerer for n samarbeidande robotar.

2.3 AVR 8bit

For å programmere roboten brukte vi C i Atmel Studio 6.2 (AS6). Det er skreve modul-basert programvare, med lite bruk av globale variablar, då dette fort vert uoversiktlig. Funksjoner skal gjere ein ting og ikkje manipulere variablar som er utfor skopet.

Tabell 2.1: Interrupt Sense Control, 11-1 frå ATmega1284p datablad

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request.
0	1	Any edge of INTn generates asynchronously an interrupt request.
1	0	The falling edge of INTn generates asynchronously an interrupt request.
1	1	The rising edge of INTn generates asynchronously an interrupt request.

Manipulering av register vert gjort ved bitskifting. Frå tidlegare prosjektarbeid på roboten er det nytta fleire måtar å gjere dette på. Det er å føretrekke den metoden som gir mest klarheit for flest. Til dømes tydar kodelinjene under akkurat det same.

```
EICRA |= (1«ISC01)|(1«ISC00)|(1«ISC11)|(1«ISC10)|(0«ISC21)|(1«ISC20);
EICRA |= 0b11111;
EICRA |= 0x1F;
EICRA |= 31;
```

Det kan diskuterast kven av desse linjene som er "best", eg vil argumentera for den første. Dette er ved å bruke makroane som Atmel har laga, slik at ein enkelt kan kontrollere kva operasjonar som vert utført i samsvar med databladet Ein som ikkje er kjend med registera til mikrokontrolleren vil lettast kunne sjekke linja opp mot tabellen til registeret i databladet.

Merk at operasjonen `(0«ICS21)` "or"-bitskifter ein null. Dette er ein "ikkje-operasjon" og den vert fjerna av kompilatoren. Denne er tatt med for å visuelt lettare lese og skrive endringar i samsvar med tabellen i databladet, sjå tabell 2.1 for eit døme på ein tabell vi finn i databladet.

Det anbefalast å gjere seg kjent med bitskifting og C programmering ved vidare arbeid på roboten.

2.4 AVR Peripherals: USART, TWI, SPI, PWM, ISR, ADC og JTAG

For å komunisere mellom AVR og nRF51 dongelen er det nytta Universal synchronous/asynchronous receiver/transmitter (USART). For å styre servoane og motoren på Arduino roboten er det nytta pulse width modulation (PWM). For å behandle enkoderane er det nytta interrupt service routine (ISR). For å lese av målingane til avstandssensorane er det brukt den integrerte analog komparatoren (ADC). For å kommunisere med inertial measurement unit (IMU) er det nytta serial peripheral interface (SPI), og for å kommunisere med kompass er det nytta inter-integrated circuit (I^2C), som Atmel kaller two wire interface (TWI). For å programmere roboten

er det brukt JTAG på AVR roboten, medan USB bootloaderen er brukt på Arduino roboten. Detaljane til alle desse funksjonene vert ikkje utbrodert i oppgåva då fleire av dei vert gjennomgått i emnet TTK4155 - *Industrielle og innbygde datasystemers konstruksjon*, og det fins mykje teori og eksempel i bøker og på internett rundt dette.

Kjennskap til funksjonane er ikkje nødvendig med mindre ein skal gjere arbeid som er direkte relatert til desse.

2.5 Sanntid og FreeRTOS

FreeRTOS er brukt på AVR og Arduino robotane. FreeRTOS er godt dokumentert i brukarrettleiinga til Barry (2009) og nettsida til FreeRTOS har eksempel på det meste. Muteks, teljande og binære semaforar, køar og prioritert task scheduling er brukt. For å finne teoretisk kapasitet til systemet er det nytta utrekningar som er pensum i emna TTK4145 *Sanntidsprogrammering* og TTK4147 *Sanntidssystem*. Teorien vert ikkje vidareformidla i anna grad enn dei spesifikke utrekningane som er gjort og metoden til desse.

Det er ein fordel å ha hatt eit av desse emna ved vidare arbeid på roboten, men ikkje ein forutsetning.

2.6 Bluetooth Smart og nRF51

Bluetooth 4, bluetooth low energy (BLE) eller Bluetooth Smart er brukt i kommunikasjonen på ein nRF51 chip, som har ein 32 bit ARM m0 prosessor. Dongelen vert programmert med C, men programmet er skreven med event handling tilnærming på grunn av bluetooth stacken til Nordic Semiconductor, og erfaring med C++ vil være ein fordel. Manipulering av register skjer på same måte som på AVR roboten, men det er noko meir abstraktert.

Dei avgjerande poenga som gjorde av BLE vart brukt framfor andre løysingar er skreven i rapporten, men BLE som protokoll er ikkje dekt, for meir detaljar anbefalast det å gå gjennom spesifikasjonen til Bluetooth-spec (2010). Programmering av nRF dongelen vert gjort i Keil uVision 5. For informasjon om softdevicane til Nordic Semiconductor, derav bluetooth stacken, viser eg tutorialane dei har lagt ut.

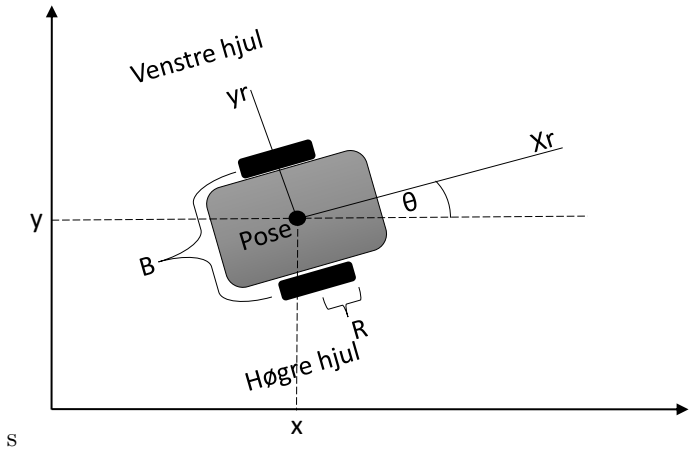
2.7 Posisjonering

For å styre roboten er det brukt PID-regulatorar og referansemodellar. For å estimere posisjonen til roboten er teorien frå Brown & Hwang (2012) brukt, som er pensum i TTK4115 *Lineær systemteori*. For videre arbeid på posisjonering er TTK4215 *Systemidentifikasjon og adaptiv regulering* og TTK4190 *Fartøystyring* nyttige emne.

2.8 Elektronikk

I prosjektet har det vore brukt ein del elektronikk, med både aktive og passive komponent. Teorien og detaljane rundt dette er ikkje utgreidd, men det er ei forutsetning med kjennskap til grunnleggande og vidarekommen elektronikk ved arbeid på sensorar og kretskort.

2.9 Matematisk modell



s

Figur 2.1: Robotmodell

"Pose" er engelsk for positur, og er brukt for å forklare posisjonen og retninga til roboten, i form av x, y og θ. Differensial-drevne mobil-robotar er ikkje holonomiske som beskreven av Spong et al. (2006). Dei er mykje brukt i forskjellige hobbyprosjekt og akademiske oppgåver, og det fins mykje tilgjengelig informasjon frå begge hold på internett. Det er populært å modellere dei analytisk som einhjulssykkel, eller "the unicycle model" som beskreven av Carona et al. (2008).

Vi kan då modellere roboten slik, der v er farta, og ω er vinkelfarta til einhjulssykkelmodellen.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.1}$$

På den faktiske roboten styrer vi farta på motorane. Vi kan modellere roboten slik

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2}(v_h + v_v)\cos(\theta) \\ \frac{R}{2}(v_h + v_v)\sin(\theta) \\ \frac{R}{L}(v_h - v_v) \end{bmatrix} \tag{2.2}$$

Der v_i er farta til hjula. Set vi (2.1) = (2.2) får vi

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \frac{R}{2}(v_h + v_v) \\ \frac{R}{L}(v_h - v_v) \end{bmatrix} \implies \underline{\underline{\begin{bmatrix} v_h \\ v_v \end{bmatrix} = \begin{bmatrix} \frac{2v+\omega L}{2R} \\ \frac{2v-\omega L}{2R} \end{bmatrix}}} \tag{2.3}$$

Denne likninga er nyttig fordi den gir oss ein overgang frå den analytiske modellen (2.1) med input v og ω til hastigheit på høgre og venstre hjul v_h og v_v for robotmodellen vår.

Likningane som er brukt for å estimere posisjonen ved hjelp av odometri vart først implementert av Syvertsen (2006). Han henta dei frå dokumentet "Where am I" av Borestein et al. (1996) kap.1.3, dei kan utledast ved å numerisk integrere (2.1):

$$\begin{bmatrix} \hat{x}_k \\ \hat{y}_k \\ \hat{\theta}_k \end{bmatrix} = \begin{bmatrix} \hat{x}_{k-1} \\ \hat{y}_{k-1} \\ \hat{\theta}_{k-1} \end{bmatrix} + \begin{bmatrix} v_{k-1}T_s\cos(\theta_{k-1} + \frac{1}{2}T_s\omega_{k-1}) \\ v_{k-1}T_s\sin(\theta_{k-1} + \frac{1}{2}T_s\omega_{k-1}) \\ T_s\omega_{k-1} \end{bmatrix} \tag{2.4}$$

der

$$v_{k-1}T_s = \frac{1}{2} \cdot \Big(\frac{(\Delta\tau_h + \Delta\tau_v) \cdot 2\pi R}{\tau_{tot}} \Big) \tag{2.5}$$

$$T_s\omega_{k-1} = \frac{1}{L} \cdot \Big(\frac{(\Delta\tau_h - \Delta\tau_v) \cdot 2\pi R}{\tau_{tot}} \Big) \tag{2.6}$$

R er radiusen på hjulet, og $\Delta\tau_i$ er endringa av tikk sidan sist iterasjon τ_{tot} er tal tikk på ein full rotasjon på hjulet.

$\frac{2\pi R}{\tau_{tot}}$ og L er konstantar, ideelt kun avhengig av storleiken til- og avstanden mellom hjula til roboten. Dei er definert som `WHEEL_FACTOR_MM` og `WHEELBASE_MM` i `defines.h` i programmet til robotane.

Kapittel 3

Utstyrsliste

Oppsummering av programvare, komponent og verktøy som er brukt.

3.1 Programvare

- Windows 7
- Atmel Studio 6.2 - Programmering av AVR og Arduino robot
- Keil uVision5 - Programmering av nRF51
- J-Link RTT Viewer - SEGGER sanntids-debugging til nRF51
- J-Flash - nRF51 flashing av bluetooth stack
- nRF51 windows 7 driver, fra Nordic Semconductor
- Termite - Seriell kommunikasjon til nRF51 dongel
- Matlab 2015a - Prototyping, modellering, plotting
- Sourcetree - Versjonskontroll av kjeldekode
- Eagle - kretskort design
- Powerpoint - Teikning av vektorfigurar til rapport
- MS project og Excel - Prosjektstyring

3.2 Komponent

- LSM6DS3 - 6DOF IMU, Gyro og akkselerometer
- GP2Y0A21YK med ledning - Sharp IR sensor for avstandsmåling
- HMC5883L - Elektronisk kompass
- nRF51 Dongle - Bluetooth 4 development kit
- ATmega1284p - Mikrokontroller for ein av robotane
- Logic level converters - Endring av spenningsnivå mellom 3 og 5 volt
- Div LEDs, resitorar, kondensatorar m.m.

3.3 Andre verktøy

- Oscillioskop - Feilsøking og tidtaking, spesielt nyttig til kommunikasjon
- Multimeter - Feilsøking av koblinger, baner og ledninger
- STK 500 utviklarkort og strømforsyning for atmel mikrokontrollere
- DC Spenningsforsyning - tilkobla robot for å ikkje tappe batteri
- Protomat - kretskortfres på verkstaden
- Optitrack Motion Capture Systems - Testing i slangelabben

Del II

Metodar

Kapittel 4

Kommunikasjonsmodul

4.1 Vurdering av mulige løysingar

Kommunikasjonen frå PC gjekk tidlegare via ein Free2Move Bluetooth-til-RS232 modul som sto på AVR roboten. NXT Roboten har innebygd Bluetooth. Free2Move modulen var frå 2004, og brukte Bluetooth 1.0 protokollen. Ei utfordring med denne modulen er at linken mellom robot og server feiler regelmessig, og det går ikkje an å spore feilen og løyse den på anna måte enn å starte alt på nytt, og prøve igjen. Den var heller ikkje tilpassa for sanntidsoppgåver.

På grunnlag av dette vart det bestemt at ein ny kommunikasjons-modul måtte utviklast. Vi kom fram til tre aktuelle løysingar som vist i tabell 4.1.

Tabell 4.1: Samanlikning av protokoller

Protokoll	Fordel	Ulempe	Hardware
WiFi	Tidlegare erfaring og mykje velprøvde alternativ lett tilgjengelig, "uendelig" rekkevidde	Høgast straumforbruk, avhengig av å koble til eit eige nettverk med eigen router	nRF24L01, ESP8266
ZigBee	Lavt straumforbruk, mesh topologi	Ingen tidlegare erfaring, meir utilgjengelig hardware og eksempel enn f.eks WiFi og BLE	xBee, ZigBit
Bluetooth 4 / BLE / Bluetooth Smart	Lågast straumforbuk, raskast oppkoblingstid	Små pakker (maks 23B)	nRF51 USB Dongle

På grunn av at NXT roboten kun kan kommunisere via I^2C eller RS485, måtte vi ta omsyn til dette.

nRF51 USB Dongle har 6 GPIO pinnar tilgjengelig, der pinnane kan setjast til å bli brukt for kva periferi vi ønsker, vi kan velge I^2C, SPI eller USART etter behov. Sidan den har USB kan den pluggast rett i EV3 roboten. Dette gjer den perfekt til prototyping.

Vi hadde lett tilgjengelig nRF51 USB Donglar frå Omega verksted.

4.1.1 BLE nøkkelpunkt

Bluetooth Special Interest Group (SIG) består av mange kjende aktørar innanfor trådlaus kommunikasjon. Dei har publisert ein "roadmap" over kva som er satsingsområde fra 2016, som i hovudsak består av

- Kjappare overføring og kapasitet
- Lengre rekkevidde
- Maskenett-topologi

Bluetooth SIG oppretta og ei eiga Mesh Working Group:

"*Some applications want good coverage that can extend an entire building, and in some applications you don't want a central hub. You want one message to be bound along with no hub in the middle.*" -Errett Kroeter, talsperson frå Bluetooth SIG. (Bluetooth-blogg 2016).

dettes er veldig interessant for robotane, og gjer bruken av BLE svært aktuell.

BLE er velprøvd og godt dokumentert. Den bruker tre kanalar den kan advertise på, og 36 tilkoplingskanalar i 2.4GHz bandet. Den tek i bruk adaptiv frekvenshopping, og vel vekk kanalar med mykje interferens, som gjer BLE til ein påliteleg protokoll.

Alle pakkar som vert sendt men ikkje verifisert, vert sendt på nytt. Når pakker vert sendt på nytt, tek det lengre tid å få alle pakkane gjennom, og det vil avgrense den totale gjennomstrauminga av meldingar. Ein pakke som vert sendt på nytt vert sendt på ein ny kanal, og vil etter kvert bli sendt på ein som ikkje har interferens, (Bluetooth-spec 2010).

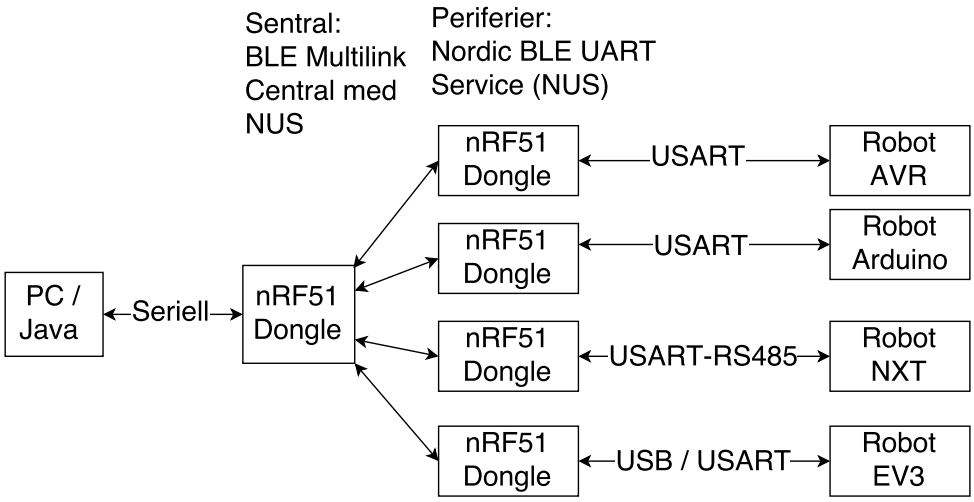
På bakgrunn av alt dette vart nRF51 USB Dongle med BLE valgt. I BLE vert "slaven" kalla "Peripheral", periferi, medan "masteren" vert kalla "Central", sentral. Ein Sentral kan koble til fleire slaver, men ein slave kan kun koble til ein sentral.

4.2 Framgangsmåte for utviklinga

For å få satt opp kommunikasjonen som ønska, gjekk eg gjennom tutorialane til Nordic Semiconductor, og prøvde fleire forskjellige eksempel frå SDKen dei gir ut.

Vi kom i kontakt med ein Nordic Semiconductor ansatt, Joakim Tønnesen, under karrieredagane. Vi fekk besøke han hjå Nordic Semiconductor, og fekk ei innføring i Keil og hjelp til å setje opp eit utgangspunkt med sentral og periferi som fungerte.

Utan hjelp ville vi nok brukt mykje lengre tid på å få dette til. Når vi hadde noko som fungerte gjekk det greit å få tilpassa det til vårt behov.



Figur 4.1: Blokkskjema for trådlaus kommunikasjon

På nRF51 er det mulig å bruke andre metoder for radiokommunikasjon enn BLE. "Gazell" og "Enhanced ShockBurst" er to av alternativa. Med eksempela dei har gjort tilgjengelig viste at det oppsto problem når fleire einingar skulle kommunisere saman, og protokollen til desse verka mindre gjennomført enn BLE.

Nordic har også gitt ut ein IoT SDK som skal fungere på nRF51 donglane, så det er mulig å bruke internett i staden for Bluetooth.

4.3 Endeleg løysing

Den endelege løysinga basert på eksempela Nordic Semiconductor har publisert. Utviklarforumet til Nordic, "Devzone" vore ei veldig nyttig informasjonskjelde. For å kommunisere med nRF51 dongelen vert det brukt USART på begge robotane. Fordelen med dette framfor SPI og I^2C er at begge mikrokontrollarane kan initialisera kontakt. Eit blokkskjema for kommunikasjonen er vist i figur 4.1. I programvaren må du velge kva Bluetooth stack du vil bruke, og du kan velge mellom forskjellige softdevices frå Nordic. S130 stacken er brukt til både sentral og periferi.

4.3.1 Central (Server)

Programmeringa av sentral dongelen vart gjort i samarbeid med Rødseth & Andersen (2016). Vi bruker ein kombinasjon av to døme som er gitt ut frå Nordic; "Multi-link Central" og "Nordic Uart Service Central". Marco Russi hadde alt kombinert desse og laga eit eksempel, så vi brukte dette (web: github Russi 2016). Eksempelet han hadde laga var ikkje satt opp for å kompilere i Keil. Eg laga ein guide til korleis

importere koden inn i Keil uVision for bruk på windows, denne ligg i vedlegg E på side 105. Vi gjorde ein del endringar på koden til Marco, der mitt bidrag var utover testing var:

- Endra meldingsoppsettet til å passe det vi bruker, tidlegare var det ein rekke AT+ kommandoar som vart terminert med punktum
- La inn alternativ for å kjøre dongelen i debug modus
- Lysdiodene viser kva status dongelen har, raud = tilkobla og i robot modus, blå = kommando modus
- Når ein melding er mottatt frå periferi, skriver sentralen ut avsendar og indeks foran meldingen som vart mottatt
- La inn at indekseringa skulle resette før skanning
- Debug-definisjon med ein rekke meldingar til brukar dersom dongelen skal brukast manuellt direkte i terminal

Tabell 4.2: Kommandoliste frå server

Kommando	Beskriving
start	Starter skann etter periferieiningar
list	Listar opp periferieiningar og skriv ut indeks:Adresse:Namn
stop	Stopper skann etter periferieiningar og skriv kor mange som vart oppdaga
conn i	Koplar til periferi eining og går i datamodus
switch i	Byttar mellom periferieining som er tilkopla, frå kommando modus og går i datamodus
drop i	Koplar frå periferi eining
*	Går til kommando modus frå datamodus
reset	Programvareomstart

For oversikt over kommandoane som vert brukt på sentralen, sjå tabell 4.2.

Vi sender meldingar frå java-applikasjonen på serveren via ein virtuell seriell port over USB. Vi kan velge kva periferi vi vil snakke til ved å velge indeks i ei liste. Sentralen er avgrensa til å kommunisere med 3 robotar samtidig per dags dato. Når DEBUG er definert vil ikkje dongelen fungere med java-applikasjonen.

Etter kvart som Nordic Semiconductor utvikler bluetooth stacken sin videre, vil det bli mulig med fleire.

4.3.2 Peripheral (Robot)

Programmet som ligg på periferi-dongelen har eg satt opp, og er basert på BLE NUS eksempelet tilgjengelig i SDK pakken frå Nordic. Eg har brukt den siste versjonen fra SDK11 og lagt inn first-in first-out (FIFO) kø på meldingane som går frå USART til Bluetooth. Implementeringa av FIFO gjorde at dongelen ikkje krasjer sjølv om den fekk fleire meldingar enn den kunne sende ut. I tillegg har eg lagt inn SEGGER real time terminal.

Ein ny mulighet med dongelen er at det går an å koble til denne med Nordic Semiconductor sin app "nRF Toolbox" og koble til ved å bruke "USART" servicen. Du vil då få kommunikasjonen til og frå roboten akkurat som serveren, og kan derfor styre roboten med mobiltelefon. Det er ikkje mulig å koble til fleire robotar samtidig med mobil-appen.

Det går an å koble dongelen til EV3 roboten i USB porten, og kjøre "BLE APP UART" eksempelet slik at du kan kommunisere via ein virtuell com port.

4.4 Meldingsprotokoll

For å kommunisere sender vi meldingar som string av characters. Vi valgte å gjera det slik fordi at det vert lett å tolka inn-kommande og utgåande meldingar.

Vi har i hovudsak delt meldingstypane opp i tre deler. Det vert kun sendt heiltal. Sjå tabell 4.3 og 4.4.

Tabell 4.3: Meldingar fra server til robot

Fra server	Kommentar
{S,CON}\n	"Status: Confirm" For å verifisere at handshaken er mottatt. Roboten starter kjøring og skanning og behandler videre meldingar
{U,%i,%i}\n	"Update, vinkel,avstand" Gir nye settpunkt til roboten i grader og cm
{S,PAU}\n	"Status, pause" Stopper tasken som styrer tårnet og målingar, slik at roboten slutter å sende updates. Nyttig dersom ein robot t.d. er ferdig å skanne, eller er komt for langt vekk og må komme tilbake gjennom eit område som alt er kjend. Merk at posisjonen til roboten ikkje vert oppdatert til serveren medan roboten står i pause
{S,UNP}\n	"Status, unpause" Fortsetter som før etter å ha fått ein pause melding
{}\n	Vert ikkje brukt, men roboten svarer med eit utropstegn om den får to krøllparantes. Vart brukt til debugging, som ein "ping" for å sjekke at roboten responderer

Tabell 4.4: Meldingar fra robot til server

Fra robot	Kommentar
{H,%i,...,%i}\n	"Handshake" Roboten sender for å vise at den er klar etter den er tilkobla. Inneheld informasjon om robot. Roboten initialiserer før denne blir sendt, så den skal ikkje koblast til før denne er mottatt av server
{U,%i,...,%i}\n	Update, x,y,heading,servo,s1,s2,s3,s4" Oppdatering fra robot til server om posisjon og målingar
{S,IDL}\n	Status, idle" Roboten står i ro etter å ha nådd eit settpunkt, vert og sendt med eit interval medan den står i ro. Vert også sendt ut dersom roboten vert sett i pause"
\t ! \n	Vert ikkje brukt, men roboten svarer med eit utropstegn om den får to krøllparantes. Vart brukt til debugging, som ein "ping" for å sjekke at roboten responderer

4.5 Kjente problem og løysingar

4.5.1 Sentral

- Dongelen kan vise at du er tilkopla til ein robot, men ved inspeksjon ser du på roboten at den fortsatt advertiser. Løysinga er å prøve å koble til igjen, eller kjøre ein reset, ny skann, og tilkobling

- Seriellporten kan slutte å fungere. Liknande problem fins til dømes på Arduino som også bruker seriell kommunikasjon via USB. Løysinga er å dra dongelen ut frå USB porten og sette den i på nytt, evt i ein anna USB port

- Vi brukte dongelen på ein MacBook. Sentralen sleit med å motta meldingar. Grunnen var sannsynlegvis enten dårleg drivar, eller at USB utgangen ikkje klarer å levere nok straum til dongelen

4.5.2 Periferi

- Dersom dongelen vert utilsikta fråkobla, vil roboten fortsette å sende meldingar, heilt til dongelen timer ut. Dette gjer at ein buffer vert fylt opp med meldingar, og viss ikkje roboten og dongelen vert restarta vil alle desse meldingane bli sendt til sentralen når den kobler til igjen. Løysinga er å koble frå ved å bruke "drop i" kommandoen eller slå av alt og på igjen samtidig

- Dersom dongelen får meldingar på TX/RX pinnane før den er "Klar" kan den få ein error som fører til restarting eller krasj. Dette vart løyst ved koble ein pinne frå dongel til AVR, denne skal være låg når dongelen er klar, og vert

sjekka på AVR før den sender noko på UART linja. Ei løysing rundt dette er å deaktivere debug mode, då vil dongelen berre restarte framfor å bli låst i ein error loop. men dersom det er ein vedvarande feil vil dongelen bli låst i ein restart-loop

4.6 Debugging

Det er tre måter å feilsøke kommunikasjonen på, og dei kan kombinerast.

- Definer "DEBUG" flagget: programmet stopper i ei løkke i app_error.c ved feil, brukaren kan legge inn eigne hendelsar
- Kjør "debug mode" for å steppe gjennom koden
- Segger J-link: Kan kjøre statusmeldingar og prints gjennom real time terminal

For å kjøre "debug mode" må koden som du skal debugge kompilerast før du starter. DEBUG flagget må være definert, og du bør kjøre uten optimalisering. Når feil oppstår på nRF dongelen, vert funksjonen `app_error_handler` i `app_error.c` kalt, og variablane `error_code`, `m_line_num`, og `m_p_file_name` er verdt å merke seg.

Om du setter ein breakpoint på while(loop); på linje 84 i `app_error.c` kan du sjekke verdien på variablane nevnt over, og du kan sjå kvar programmet ditt feiler. Eit døme, sjå tabell 4.5.

Tabell 4.5: Døme på feilkoder

Variabel	Verdi	Tyder
error_code	0x00003004	Feilkode (hex) 3004
m_p_file_name	0x000203EC "../../../_main.c"	Vart trigga i main.c
m_line_num	0x000001A8	Linje nummer i filen over i hex, altså linje 424

Om du går til `main.c` på linje 424 vil du då sjå ein variabel med navn "`NRF_ERROR_***`" og du kan høgreklikke på denne og gå til definition, du havner i "`nrf_error.c`" og sjekke kva verdien til `error_code` tyder, i dette tilfellet er det "No Memory for operation"

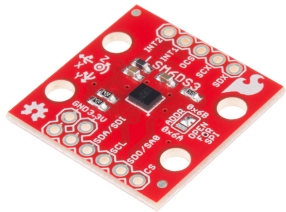
Segger J-Link Real Time Terminal er eit alternativ til USART for å printe feilkoder. Fordelen er at RTT ikkje påvirker oppførselen til sanntidssystemet. For å bruke denne må J-Link RTT Viewer brukast på datamaskinen. Viser til Segger tutorialen til Nordic.

Kapittel 5

Sensorar

5.1 IMU - LSM6DS3

IMU som er brukt er avbilda, sjå figur 5.1. Den har 6 frihetsgrader, akselerasjon i xyz, og vinkelhastigheit om xyz. Drift-spenninga er 3.3V, og det må være ein nivåkonverter mellom AVR og IMU.



Figur 5.1: LSM6DS3 IMU breakout (Foto: sparkfun)

For å kommunisere med denne har eg nytta SPI, då den har raskast oppdateringsfrekvens, maks 10Mhz. Det er og mogleg å interface sensoren med I^2C. Det var ein jumper som måtte fjernast for at IMUen kunne brukast med SPI. Dette vart gjort på alle tre IMUane som var kjøpt inn, så dersom ein av dei som ikkje er i bruk skal brukast med I^2C, så må jumperen loddast på nytt.

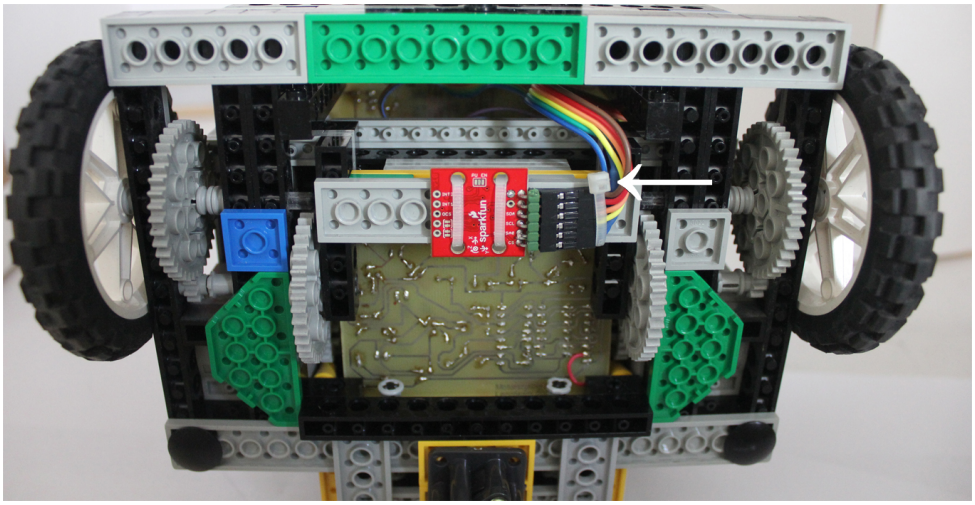
For å bruke IMUen skreiv eg om drivaren som er laga for Arduino biblioteka. Denne er i utgangspunktet skriven i ein variant av C++, og drivaren som er omskriven er sannsynlegvis ikkje optimal.

Det går an å stille på måleområdet og bandbredda til IMUen.

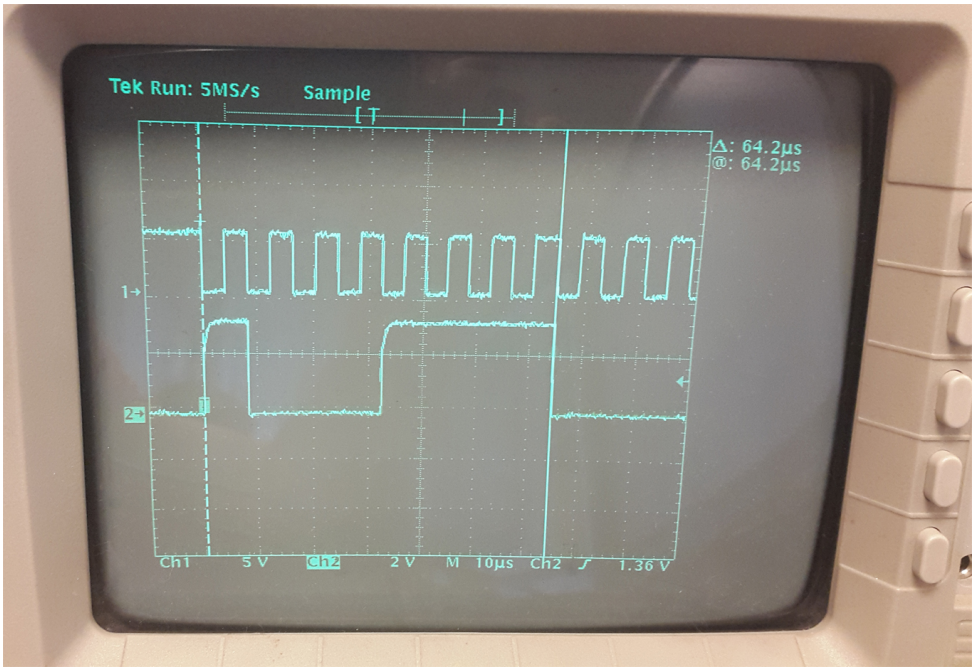
IMUen treng ein liten motstand i serie på MISO linja når den er tilkobla ATmega1284p på AVR roboten, sjå figur 5.2. Dette var ikkje naudsynt på Arduino roboten. Årsaka til dette er ikkje kjend, men det er antatt at det har noko med den indre motstanden i mikrokontrolleren å gjere.

Under initialisering vert det sendt ein verdi der IMUen skal svare med ein anna verdi. Denne er definert som "Who am I" i databladet til IMUen. Dersom IMUen ikkje svarar med rett kode i retur, fortsetter mikrokontrolleren å prøve. Det vil sei at om IMUen ikkje er tilkobla vil mikrokontrolleren fryse ved initialisering som nemnt i

kapittel 6. Sjå figur 5.3 for eit døme på debugging av SPI kommunikasjon.



Figur 5.2: IMU plassert i senter av hjul. Kvit pil peikar på MISO motstand



Figur 5.3: Oscilloskop for debugging av SPI, sender 0x8f + 0x00 på MOSI

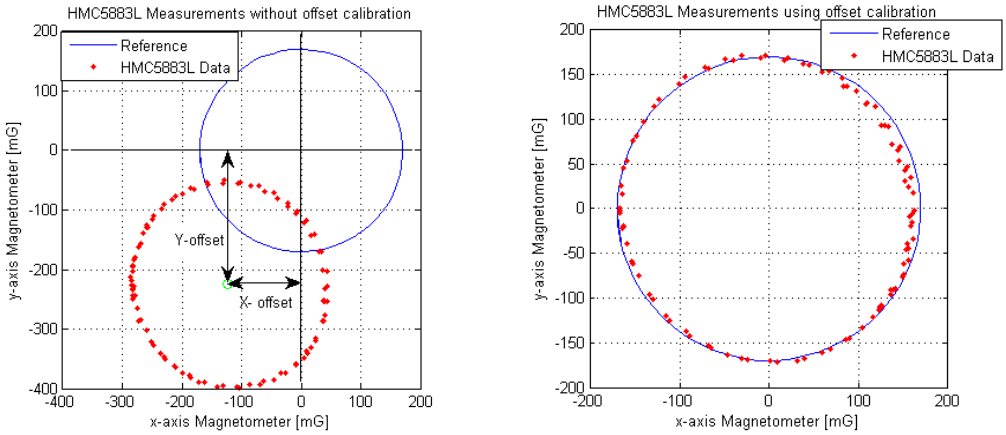
5.2 Elekromagnetisk kompass - HMC5883L

Kompasset som er brukt er avbilda i figur 5.5. Drift-spenninga til kompasset på dette kortet er både 3.3V og 5V. Det er tilkobla 5V. Kompasset måler magnetisk feltstyrke i xyz retning, og kan brukast til å finne magnetisk nord i absolutt posisjon ved hjelp av *Atan*2 funksjonen. Kompasset er ikkje tilt-kompansert, men det er ikkje nødvendig så lenge roboten kjører på eit vassrett underlag.

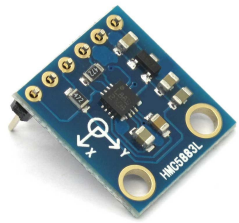
Kompasset kommuniserer med mikrokontrolleren ved hjelp av I^2C, maks frekvens for å kommunisere med kompasset er 160hz. Dersom det feiler eller ikkje er tilkopla vil roboten fryse under initialisering, og om det feiler under kjøring vil programmet fryse når det prøver å kommunisere på I^2C bussen, som nemnt i kapittel 6.

For at komponent på roboten ikkje skal gi innverknad på målingane må kompasset kalibrerast. Eit døme før og etter kalibrering er vist i figur 5.4. Som vi ser er det vanskelig å finne ei heading med *Atan*2 når sirkelen vi måler ikkje er om origo. For meir informasjon om korleis kompasset vert kalibrert sjå web: Hobbylogs (2016).

Det er skreve ein eigen task for kompasskalibrering. Detaljane om dette er beskriven i kapittel 8 og i vedlegg B.



Figur 5.4: Før og etter kalibrering (Plot: Hobbylogs)



Figur 5.5: HMC5883L kompass breakout (Foto: google)

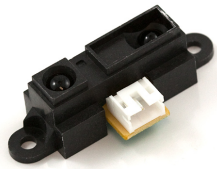
5.3 Avstandssensor - GP2Y0A21YK

For å måle avstand til veggane bruker vi infrared (IR) sensorar, sjå figur 5.6. Det er skreve mykje om desse i fleire av rapportane til tidlegare arbeid, mellom anna i kapittel 4.1.1 av Tusvik (2009)
Kort oppsummert:

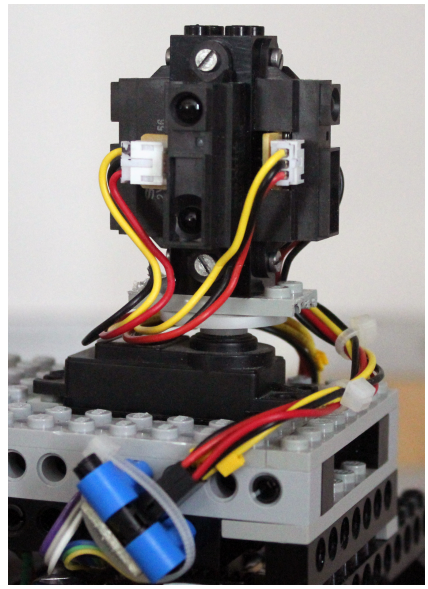
- Måleområdet er ca 5-80 cm, men best 10-40 cm. Innafor 5 cm får du samme analogverdi som ved over 80, dvs at roboten sergjennom vegger
- Gir ut ein del målestøy, og har ein varians på eit par cm
- Tar 300 målingar i sekundet
- Treng ca 30ms "settling time" etter dei er påskrudd

AVR Roboten hadde i utgangspunktet ein eldre versjon av desse sensorane. Eg kalibrerte desse på nytt, men oppdaga at måleområdet var vesentleg dårlegare enn den nye versjonen sensorane som var kjøpt inn. Minimums-avstanden sensorane kan måle mykje tettare på dei nye enn dei gamle. Problemet dei gamle sensorane var at roboten "ser" gjennom veggar dersom dei kjem for nær. Sensorane vart derfor bytta ut med den nye typen. Det vart gjort ei kalibrering på ei av dei nye sensorane, og resultatet til dette vart brukt på alle sensorane på begge robotane. Ved måling såg vi at det var ein liten bias på avstanden, men antok at dette ikkje var nok til å gjere noko utslag på kartlegginga.

Tidlegare var det problem med at koblingane til sensorane feila Tusvik (2009). Årsaken til dette var at tilkoblinga ikkje hadde rett tverrsnitt. Sensorane var i tillegg festa med gummistrikk og teip. Sensorane har no blitt festa med maskinskruer, og tilkoblingane er bytta ut til rett type, sjå figur 5.7.



Figur 5.6: GP2Y0A21YK IR sensor (Foto: sparkfun)



Figur 5.7: Ny sensor-tårn løysing med skikkelige tilkoplingar og maskinskruer

Tabellen til kalibreringa vart laga i matlab basert på målingar og linjer regresjons-analyse. Skriptet til dette er lagt ved på DVD.

5.4 Enkoderar

På AVR roboten er det brukt optokoplar som gir ut høgt signal når eit tannhjul blokkerer ein lys-stråle. På Arduino roboten er det brukt ei magnetisk skive som roterer, dokumentert av Rødseth & Andersen (2016). Enkoderen til AVR roboten vart laga av Syvertsen (2006) og er dokumentert i kapittel 6.1

Kapittel 6

Ny robotapplikasjon

6.1 Programmeringsoppsett - STK500

Før programmet var klart til å kjøre på roboten vart mikrokontrolleren vart kobla opp på eit STK500 development kit frå Atmel. Fordelen med dette er

- Enkel tilgang på alle pinnar
- Kretskortet er korrekt oppbygd, så du kan utelukke hardware feil,
- Muligheit til å koble til ein seriellkabel for å kommunisere direkte via USART
- Knappar og LEDs som er nyttige til debugging
- Dedikert straumforsyning so du er ikkje avhengig av ein svær spenningskilde eller batteri

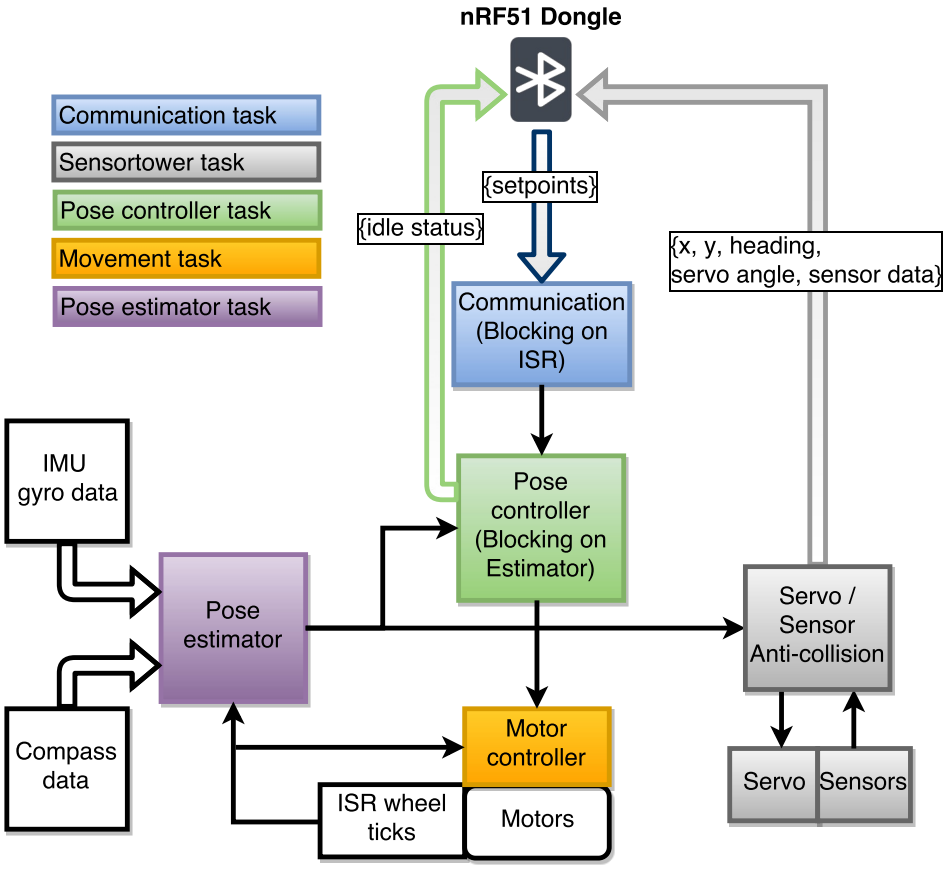
Etter kvart som programmet vart klart til å kjøre på roboten vart programmeringa gjort rett på denne i staden. Ved feilsøking vart det nyttig å ha to plattformer å teste på, for å lettare utelukke feilkilder. Kortet kan også brukast til å "lytte" på USART kommunikasjonen mellom dongel og mikrokontroller.

6.2 Programbeskrivelse

Programmet til roboten har vore utvikla over fleire år. Det var eit poeng å bruke det som fungerte tilsynelatande bra, og så skrive om det som ikkje fungerte.

Med muligheit for multitasking kan vi no legge inn estimator, regulator og andre oppgåver med absolutte perioder, som separerte oppgåver. I tillegg er det lett å definere grensesnitta mellom oppgåvene då kommunikasjon mellom må gå enten via ein køfunksjon (xQueue), eller ein global variabel. Dei globale variablane har ein prefiks med g, til dømes "gVarNamn". Desse må beskyttast med muteksar.

Det ferdige programmet er illustrert som blokkskjema i figur 6.1.



Figur 6.1: Programstruktur til robot

Periodane til taskane er forklart i kapittel 9. Forklaring til taskane

- Communcation (Blå): Sender "handshakes" og tar seg av innkommande kommunikasjon
- Sensor tower (Grå): Tar målingar og sender pose og avstandsmålingar til serveren
- Controller (Grønn): Rekner ut pådrag ut i frå settpunkt og robotens pose
- Movement (Gul): Gjer pådraga om til spenning for motorane, og sikrar at dei kjører like fort
- Estimator (Lilla): Rekner ut robotens posisjon og heading (pose)

I programmet har taskane ein prefiks "vMain" og ein postfiks "Task", som spesifisert av Barry (2009) gjer denne namnekonvensjonen det enkelt for brukaren å sjå returtypen til funksjonen, samt kva fil funksjonen er definert i.

Nokon taskane kan køyre uavhengig av andre, slik at ein kan til dømes deaktivere posisjonsestimatoren og kontrolleren utan at da gjer at resten av koden ikkje fungerer. Dette gjer det enkelt for brukaren då programmert vert delt opp i moduler.

Pause

Roboten har pause modus, denne vert aktivert og deaktivert ved å sende {S,PAU}\n og {S,UNP}\n. Gjer at sensor tårn tasken vert satt på pause. Estimator og kontroller kjører fortsatt i bakgrunnen, men oppdatert posisjon vert ikkje sendt til sentralen. Om roboten får sett-punkt vert det utført, og den vil sende "idle" når den har komt fram, eller om den er i ferd med å kollidere (Sjå kapittel 8).

6.3 Tilstander

Roboten har fleire tilstandar den kan være i utanom avskrudd, det går an å lese tilstanden på roboten ved å sjå på lysdiodene og sensortårnet. Dette er nyttig for å verifisere at roboten oppfører seg som planlagt.

Kommunikasjonen er ikkje tilkopla server

Posisjonsdata er tilbakestilt til 0, roboten måler heading frå kompass og rekner ut bias på gyro sensoren. Grønn LED blinker kvart 2. sekund, servo står i 0. Motoren er av. Dongelen "advertiser" over bluetooth.

Figur 6.2: LED: Advertiser

Tilkopla men ikkje verifisert

Handshake med info om roboten vert sendt ut ein gang i sekundet, grønn LED lyser, gul LED blinker i det handshake meldinga blir sendt. Sensor tårnet står i 0 grader.

Figur 6.3: LED: Handshakes

Tilkopla og verifisert

Alle taskene kjører. Estimatoren bruker gyro, enkoder og kompass til å rekne ut ny posisjon kvert 30ms. For kver gang eit nytt posisjonsestimat er laga, gir estimatoren ut ein semafor til kontrolleren, som sjekker om det er komt nye settpunkt frå kommunikasjonstasken. Deretter vert høveleg rørsle bestemt i kontrolleren, og sendt

vidare til movement tasken, som styrer motorane. Movement tasken går med 20 ms periode. Samstundes går sensor tasken med ein periode på 200 ms, der den setter servoen i 0-90 grader og tek målingar frå alle sensorane, og sender oppdateringane til serveren via UART. Grønn LED lyser. Dersom posisjonsestimatoren inkluderer kompasset vil gul LED lyse ein kort periode medan roboten står i ro.

Figur 6.4: LED: Tilkopla og verifisert

I tillegg har sensor tårnet tre tilstandar medan roboten er aktiv, sjå tabell 6.1.

Tabell 6.1: Sensortårnets tilstander

Sensortårn	Robotens rørsle
Står rett fram	Inaktiv, er ikkje verifisert
Inkrementerer med små steg	Venter på nye settpunkt (idle)
Står i ro i ein vinkel	Roterer eller pause om gul LED lyser
Inkrementerer med større steg	Kjører framover eller bakover

Tilkopla, verifisert og i pause

Roboten er aktiv, men sensor tasken er sett på hold, slik at det ikkje vert sendt oppdateringar til serveren. Roboten kan framleis kontrollerast og vil rekne ut nye posisjonsestimat. Grønn og gul LED lyser. Sensortårnet held posisjonen.

Figur 6.5: LED: Pause

Stack overflow hook

Alle LEDs lyser litt svakare enn vanleg, ingen andre tasker kjører. Roboten må feilsøkast.

Figur 6.6: LED: Stack overflow

Initaliseringsfeil

Kompass eller IMU feil ved initalisering. Roboten må feilsøkast.

Figur 6.7: LED: Feil

6.4 Intern kommunikasjon og synkronisering

For 1-til-1 og n-til-1 kommunikasjon er det brukt FIFO kø med 1 plass. For 1-til-n kommunikasjon er det nytta global variabel som ein resurs. For å beskytte resursar er det brukt mutekser. For å synkronisere er det brukt semaforer. Tabellane syner implementasjonen.

Tabell 6.2: Implementerte RTOS køar

Namn	Sender	Fra	Til
movementQ	Ønska rørsle	Fleire	Movement task
poseControllerQ	Strukt med settpunkt	Kommunikasjon	Pose control
scanStatusQ	Noverande rørsle	Movement task	Tower task
actuationQ	Ønska pådrag	Pose controller	Movement task

Tabell 6.3: Implementerte RTOS mutekser

Namn	Beskytter
xPoseMutex	Globale pose variablar
xUartMutex	USART buffer via printf()
xTickMutex	Globale tikk variablar

Tabell 6.4: Implementerte RTOS semaforer

Namn	Fra	Til
xControllerBSem	Estimator	Kontroller
xCommandReadyBSem	USART ISR	USART task

6.4.1 Avbrotsrutinar

USART interrupt buffer recieve

Når USART bufferen er klar med ny data vert dette kopiert inn til ein buffer, og det vert sett signal til kommunikasjons-tasken om å behandle denne så fort som kernelen tillet det.

Enkoder tikk

Aukar eller minkar variabelen som teller tikk for kvert representattive hjul, avhengig av retninga som er satt at dei skal gå. Det vil sei at dersom roboten ikkje har fått beskjed om å gå nokon spesiell retning, vert tikka ikkje registrert, då det ikkje går an å sei kva retning hjulet går. Retninga vert ikkje satt til stopp når motorane skal stoppe, slik at siste rørsla bestemmer bidraget ved oversving.

nRF51 pin 19

Dongelen setter denne pinnen låg dersom den er tilkobla ein sentral. Den vert sett høg så fort dongelen ikkje lenger er tilkobla. På roboten vert denne sjekka for kvar gang før ein melding blir sendt over USART. Dersom vi mister tilkoblinga under kjøring, nullstiller alle programmet alle status-variablerm stopper motorane, og dongelen advertiser på nytt.

Kapittel 7

Ny kontroller og estimator

7.1 Posisjonskontroller

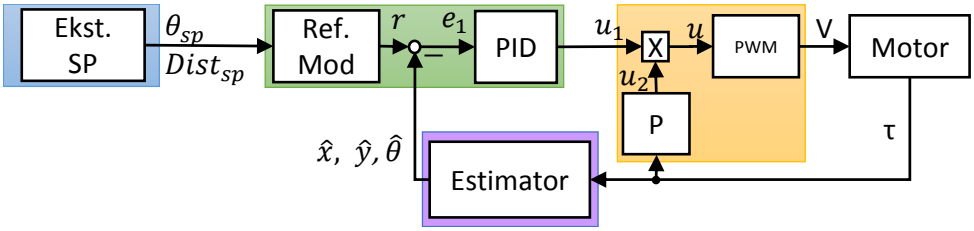
For å styre posisjonen til roboten har eg lagt inn ein PID regulator saman med ein referansemodell, og ein til P / PI regulator som samanliknar tal tikk (τ) frå hjula til roboten. Vi får settpunkt i form av vinkel og distanse frå sentralen, og desse blir tatt inn i referansemodellen. Referansemodellen vert samanlikna med den estimerte headinga, eller posisjonen, og avviket mellom går gjennom PID algoritmen, før outputen vert sendt vidare til movement kontroll tasken. Der vert PID outputen sendt til motorane via ein til regulator, og gjort om til spenning i form av ein PWM verdi. Regulatorsløyfa er illustrert i figur 7.1, fargane representerer taskane frå figur 6.1. Likningssett (7.1) beskriver regulatoren:

$$
\begin{aligned}
r(t) &= r(t) + K_R(Sp - r(t)) \\
e_1(t) &= r(t) - y(t) \\
u_1(t) &= K_P \cdot e_1(t) + K_I \cdot \int e_1(t)dt + K_D \cdot \dot{e}_1(t) \\
e_2(t) &= \Delta\tau_h - \Delta\tau_v \\
u_2(t) &= \begin{cases} 1, & \text{if } |e_2(t)| < 1, \\ 0, & \text{else} \end{cases} \\
u(t) &= u_1(t) \cdot u_2(t)
\end{aligned}
\tag{7.1}
$$

Notasjon:

$$
\begin{aligned}
Sp &= \text{Eksternt settpunkt i form av } \theta_{sp} \text{ og } Dist_{sp} \\
y(t) &= \text{estimert pose: } \hat{x}, \hat{y}, \hat{\theta} \\
K_R &= \text{Forsterking til referansemodell, mindre enn 1 og større enn 0} \\
K_P, K_I, K_D &= \text{PID tuningvariabler}
\end{aligned}
$$

Referansemodellen gjer at vi kan ha ein større proposjonalforsterking en vi normalt ville hatt, og roboten vil akselerere og deakselerere, noko som vil minke integrert feil ved spinning. Når vi tar inn nye sett-punkt vert referansemodellen initialisert med posisjonen til roboten. For å rekne xy koordinat om til distanse er det brukt Pythagoras læresetning, sjå (7.2).



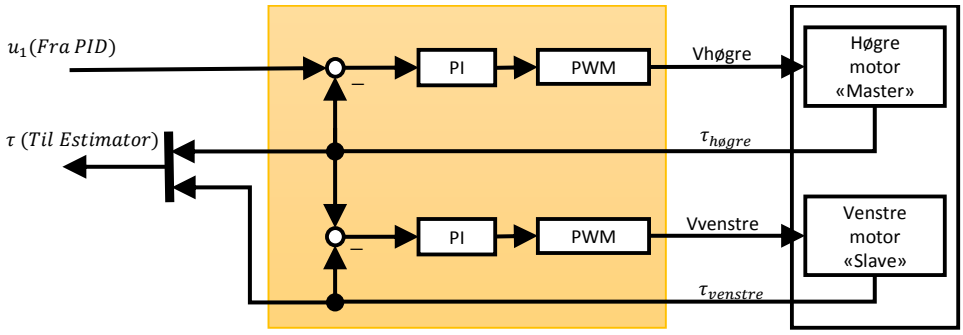
Figur 7.1: Blokkskjema for posisjon-regulator, motorstyring og estimator

På grunn av at tala vert kvadrert so må koordinata konverterast frå MM til CM, då variablane som skal holde verdiane fekk overflow ved kvadrering av millimeter. Dette fører til at roboten berre kan regulere med cm presisjon i distanse. Estimatoren jobbar fortsatt med millimeter, men estimata vert gjort om til cm før dei verdt sendt til serveren.

$$e_{xy}(t) = r_{xy}(t) - \sqrt{(x_{init} - \hat{x})^2 + (y_{init} - \hat{y})^2} \tag{7.2}$$

7.1.1 Kontroller for Arduino robot

For å styre Arduino roboten måtte kontrollalgoritmen utvidast. Utfordringa var at motorane hadde ekstremt høg hastighet i forhold til motorane som er brukt på AVR roboten. Det ein stor forskjell i hastigheita på venstre og høgre motor, i tillegg når vi skal styre motorane må vi ha ei viss spenning for å få motorane til å begynne å gå, men for å halde konstant låg fart må dei ha ein lågare spenning. Løysinga vart å implementere ein PI regulator for kvert hjul, der eit av hjula vart "master" og den andre vart "slave" og regulerer etter hastigheita til den andre motoren. Endringa er vist i figur 7.2.



Figur 7.2: Blokkskjema for motorstyring på Arduino robot

7.2 Forbetring av posisjonsestimat

I 2009 skreiv Tusvik Tusvik (2009) at det var ein vesentleg forskjell på hjulfaktoren til roboten når den roterte og kjørte rett fram, og løyste dette ved å implementere to forskjellige hjulfaktor-konstantar. I testane sine viste Ese (2015) at estimatet til roboten vart vridd i det roboten roterte. Tidleg testing i denne oppgåva verifiserte resultata som vart konkludert i begge rapportane. Eg fann ut at når roboten kjører i ei rett linje er nøyaktige nok til at det burde fungere tilfredstillande. For å korrigere absolutt posisjon, (x, y) innomhus er det fleire kjente metodar, nokon av dei er gjennomgått av Shalam & Rodriguez (2011), men dette vart ikkje prøvd i denne oppgåva. Basert på at translatorisk posisjonsestimat var godt nok, fastslo eg at for å forbetre estimatoren var det viktigast å forbetre estimatet til headinga.

For å forbetre heading estimatet vart det implementert ein gyro som måler vinkelhastigheita til roboten. Målingane frå gyro er uavhengig av underlaget og friksjonen til hjula, og vert samansveisa med målingane frå enkoderane ved hjelp av eit komplementær-filter (likn 7.4). Resultatet er eit estimat for headinga til roboten. For å korrigere estimatet vart målingane frå det elektromagnetiske kompasset kjørt gjennom eit kalmanfilter.

På grunn av magnetiske anomaliar er kompasset upåliteleg, og ein funksjon for å prøve å unngå å gjere avlesingar i forstyrra område vart utvikla, og ser ut til å fungere. Meir detaljar rundt dette er beskriven i delkapittela under. Pseudokoden til estimatoren står på side 42.

7.2.1 Sensor fusion og kalmanfilter

Roboten posisjonerer seg ved å bruke “Dead reckoning” metoden, metoden er drøfta i kapittel 5 i “Where am I” av Borestein et al. (1996). Her er svakheitane med odemetri utgreidd i detalj. Dei er delt opp i systematiske og ikkje-systematiske problem. I korte trekk går det på at verkelegheita ikkje er ideell. Meir om dette er beskriven på side 130 i dokumentet. Her det og beskriven problem med IMU og kompass til mobile robotar. Det som er verdt å ta med seg er at akselerometeret gir store utslag/forstyrrelsar ved ujamt underlag og vibrasjonar. Ein gyro, sjølv med eit extended kalman filter (EKF) gir drift på 1-3 grader per minutt. “Where am I”, kapittel (5.4.2).

Det går an å minke drifta ved å finne meir optimale konstantar for forholdet mellom tikk og avstand, men dette er avhengig av underlaget roboten kjører på, og den einaste måten å få bukt med dette på er å implementere ein ekstra sensor som kan måle absolutt verdi for å korrigere. Dette gjer det elektroniske kompasset, som kan gi absoluttvinkel til magnetisk nord.

Enkoderane gir ut eit interrupt for kver gang eit tannhjul passerer ein optisk sen-

sor. Dei måler ikkje retninga hjula går, derfor må pådraget bestemme kva bidrag enkoderane skal gi. Modelleringa og notasjonen til roboten er skreven i kapittel 2.9.

For å oppsummere:

- Enkoderane gir oss $\dot{x}_r$ og $\dot{\theta}_r$ ved å måle farta på venste og høgre hjul ut i frå tikk, men er utsett for usikkerhet grunna uoptimale konstantar og antagelsar som ikkje er ideelle
- Akselerometeret gir oss $\ddot{x}_r$ og $\ddot{y}_r$ direkte, men er utsett for støy, spesielt ved vibrasjonar og ujamnt underlag
- Gyro gir oss $\dot{\theta}_r$ direkte, men er utsett for drift/bias
- Kompasset gir oss θ relativ til magnetisk nord. Gir varierande målingar og er utsett for magnetiske anomaliar

For å finne biasen til gyroen tek eg gjennomsnittet av eit tal målingar. Dette vert gjort når roboten initialiserer, og biasen vert trekt frå alle målingane i etterkant.

Det er derfor viktig at roboten får stå i ro eit par sekund før den starter. Det er bevisst lagt inn ei forsinking etter roboten vert tilkobla til den starter.

7.2.2 Varians

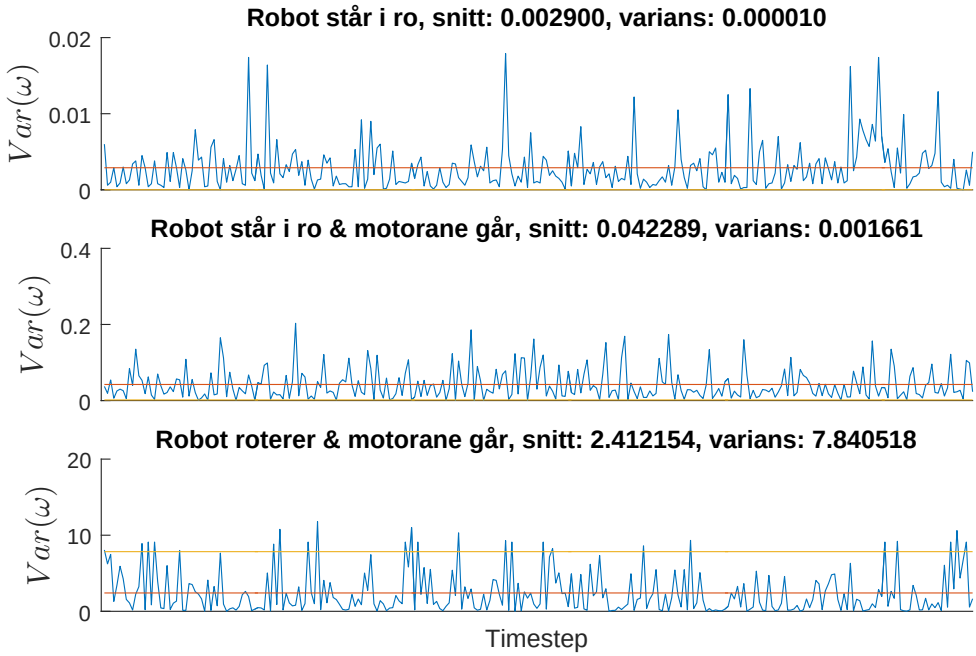
I kalman filteret bruker vi variansen til sensorane. For å finne variansen til gyroen vart det laga ein funksjon på roboten som tok målingar frå gyroen, rekna ut gjennomsnittet og variansen. Det vart gjort tre forsøk, resultata er vist i figur 7.3. Først står roboten i ro, så går motorane men roboten var heva slik at den sto i ro. Siste er at roboten roterte rundt for eigen maskin.

For å finne eit tal for variansen til enkoderane brukte eg buelengde formelen, $\theta = \frac{S}{r}$ og antok at vi kan i værste fall få 2 tikk feil, 1 for mykje på eit hjul, og 1 for lite på det andre hjulet.

$$Var(\omega_{enk}) = \frac{2 \cdot \texttt{WHEEL\_FACTOR\_MM}}{\texttt{WHEELBASE\_MM}} \tag{7.3}$$

For å bruke variansen i kalmanfilteret summerte eg variansane til sensorane, og ganga med periode tida.

Variansen til kompasset er oppgitt i databladet til å være 2 grader, men på grunn av magnetisk anomali vart variansen som vert brukt i kalmanfilteret justert kraftig opp, og blir ein av tuningvariablane til estimatoren.



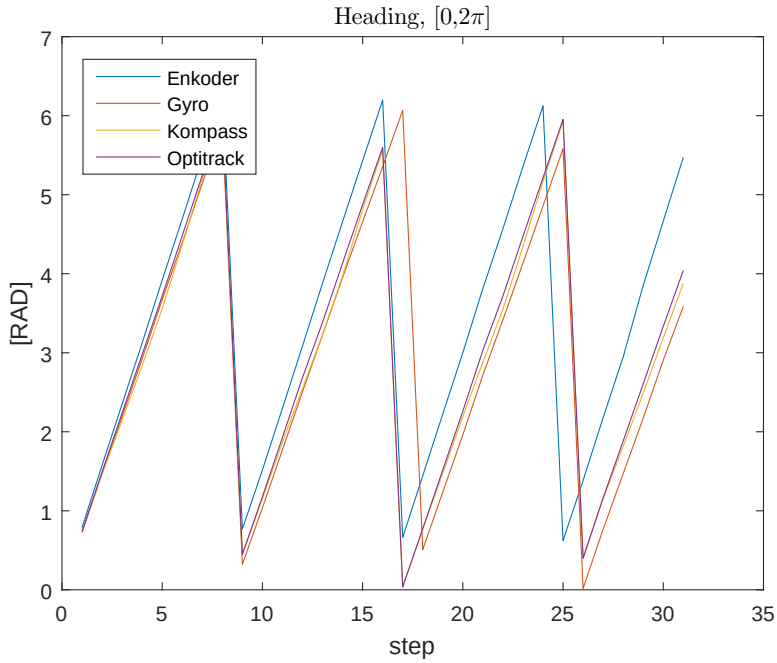
Figur 7.3: Eksperiment for å finna høveleg verdi for gyrovarians

7.3 Testing med kamera for absolutt posisjonering

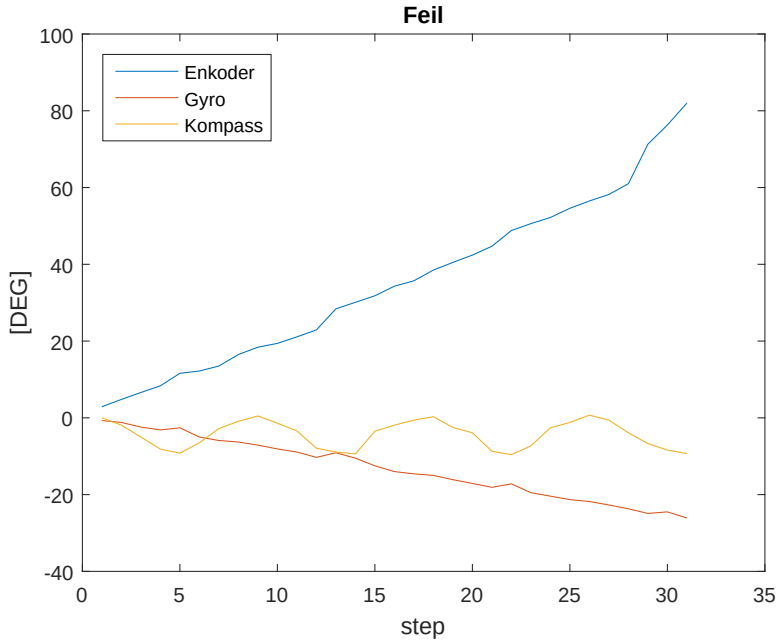
Etter at eg fekk tilgang til slangelaben kunne eg samanlikne ekte målingar med estimata.

Eg fekk då bekrefta følgjande:

- Enkoderane har ein aukande bias, og storleiken på denne er avhengig av friksjonen til underlaget. Ved låg friksjon er biasen stor, og ved høg friksjon er den mindre $\implies$ `WHEEL_FACTOR_MM` er avhengig av underlaget
- Gyroen har ein tendens til å vise for liten vinkelfart. Sjå figur 7.4 og 7.5. Eg prøvde å gange gyroen med ein konstant, men eg konkluderte med at forholdet mellom ekte vinkelfart og målt vinkelfart ikkje er konstant når farta ikkje er konstant
- Frå målingane ser det ut som at det fins eit optimalt forhold mellom gyro og enkodermålingane. Dette forholdet er avhengig av rotasjonshastigheita til roboten og underlaget hjula kjører på, og derfor ikkje lett å finne med mindre ein har ei "ekte" referanse å alltid samanlikne med



Figur 7.4: Forskjellige estimat og ekte heading



Figur 7.5: Differansen mellom estimat og ekte heading

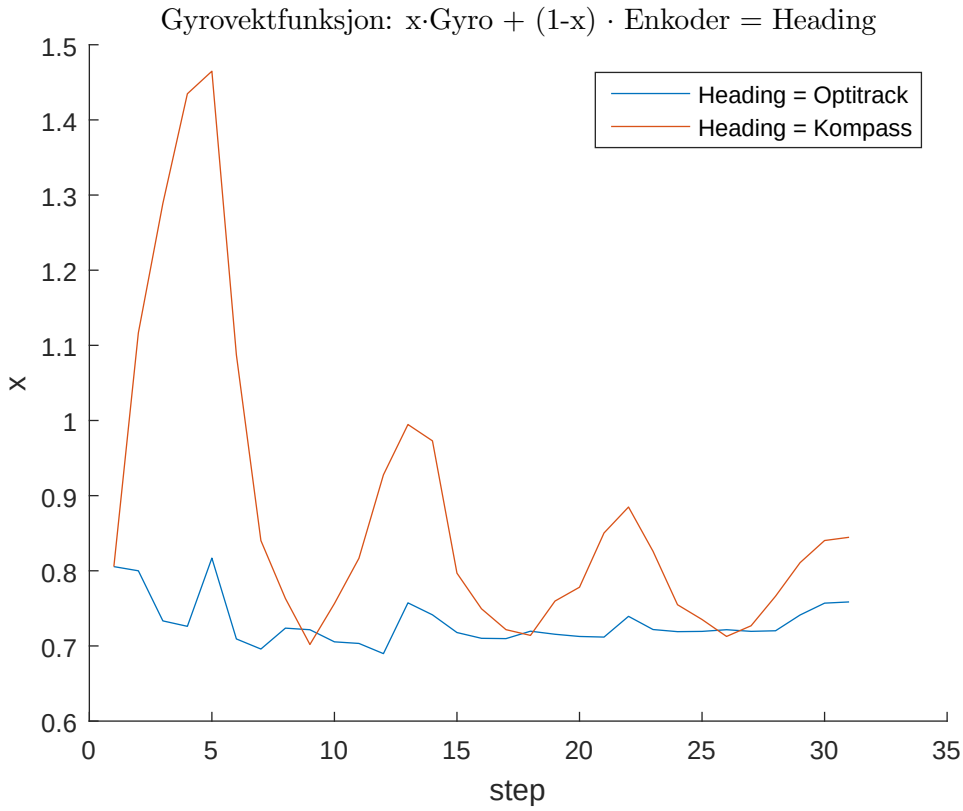
Når vi har ei absolutt referanse kan vi kalibrere estimatoren for underlaget. Ved å samansveise målingane frå enkoder og gyro lager eg eit estimat for å estimere headinga. Eg kaller estimatet frå enkoderane for $\hat{\theta}_{enk}$, estimatet frå gyroen for $\hat{\theta}_{gyro}$ og forholdet for α

$$\theta = \alpha \cdot \hat{\theta}_{gyro} + (1 - \alpha) \cdot \hat{\theta}_{enk} \tag{7.4}$$

der θ er headinga til roboten. Vi kan då finne α ved å snu på likninga.

$$\alpha = \frac{\theta - \hat{\theta}_{enk}}{\hat{\theta}_{gyro} - \hat{\theta}_{enk}} \tag{7.5}$$

Ut i frå målingane konvergerte α mot 0.75, (blå linje, fig.7.6). Eit forsøk ved å kjøre på eit anna underlag etter kalibreringa viste at når friksjonen endra seg vart estimata dårlegare igjen, på grunn av ein α som ikkje er optimal lenger.



Figur 7.6: Gyrovektfunksjon med kompass og ekte heading som mål

Adaptivt komplementærfilter

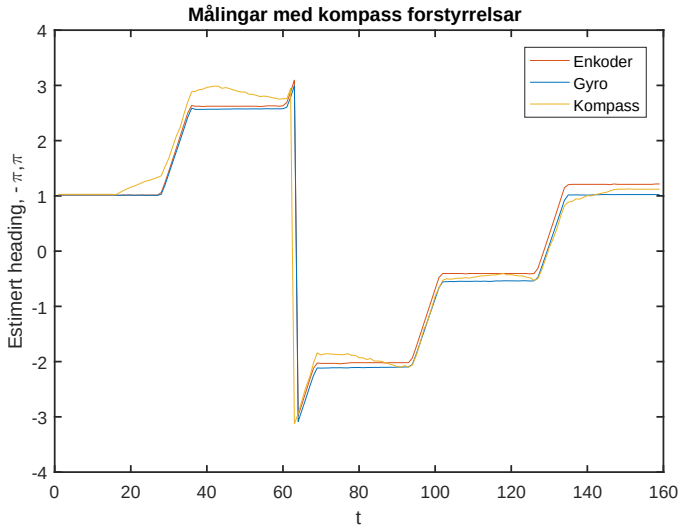
Dersom vi kunne brukt kompassmålingane til å finne den korrekte headinga kunne vi også funne α og slik korrigert parameteren til hjula i forhold til friksjonen på gulvet, slik at roboten automatisk finner rett forhold mellom enkoder og gyro.

Funksjonen for dette vart implementert men ikkje prøvd ut skikkelig. Figur 7.6 viser korleis forholdet vart dersom det hadde blitt rekna ut i frå kompass, kontra ut i frå absolutt målingar, for kvert enkelt steg, men α større enn 1 er problematisk. Same metoden kan kanskje nyttast mellom akselerometer og enkoder estimat i translatorisk rørsle, og saman kan dei adaptivt generere meir optimale konstantar, og oppdage om roboten sklir.

7.3.1 Kompassbehandling

Kompasset er svært utsatt for magnetiske forstyrrelsar. Dette var lett å slå fast etter ein enkel test der roboten vart satt til å kjøre i ein firkant ved å kjøre motorane ei viss tid for å ikkje være avhengig av resultata til estimatoren.

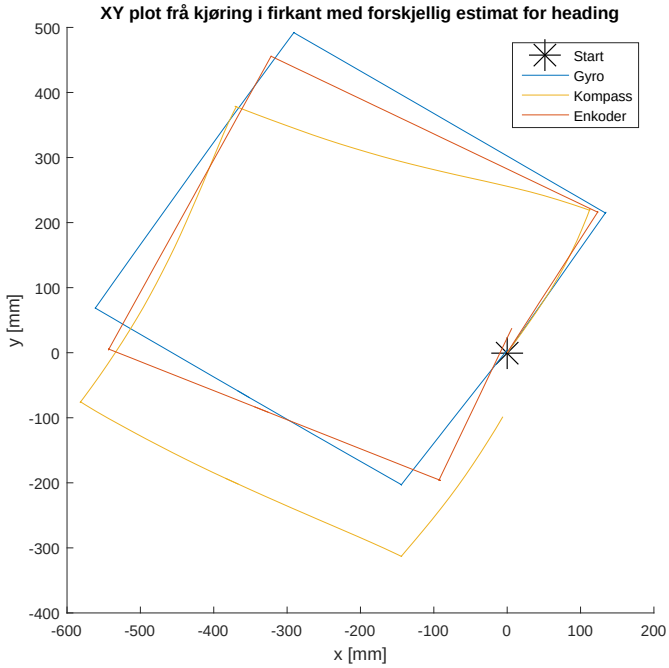
Roboten kjørte ikkje i ein perfekt firkant, men godt nok til å visuelt kunne samanlikne med estimerte verdiar og ekte posisjon. Enkoderane vart brukt for alle tilfelle når det gjalt x og y, men headinga vart estimert av dei tre sensorane individuellt. På figur 7.7 ser vi at når roboten kjører inn i eit forstyrra område vert vinkelen heilt feil.



Figur 7.7: Heading frå forskjellige sensorar

Konsekvensen av dette kan vi sjå i figur 7.8, der vi ser at bana til roboten kurvar

seg. Det er heilt feil, og det er ikkje akseptabelt at roboten skal plotte at den kjører i ei kurve når den eigentleg kjører rett fram. Vi ser også konsekvensen av feilen på xy koordinat når headinga vert estimert feil er roboten sin sluttposisjon er på tre forskjellige plassar.



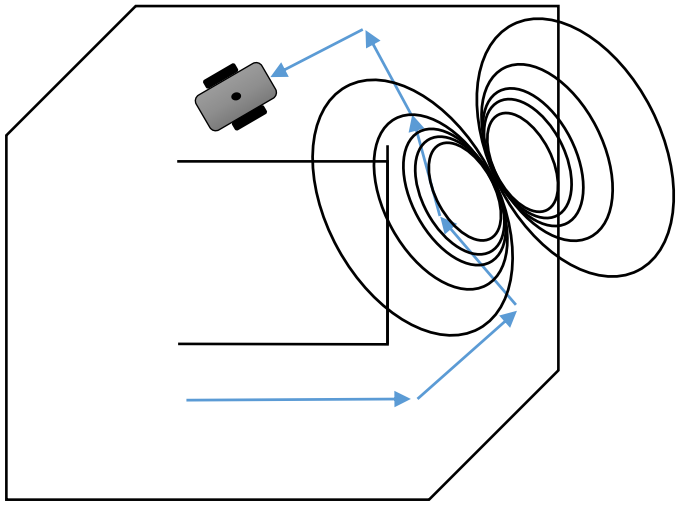
Figur 7.8: Sammenlikning av headingestimatorer

7.4 Estimator task og pseudokode

For å unngå at roboten skal vise at den kjører i ei kurve ved forstyrrelsar vart det implementert fleire sjekkpunkt i estimatoren.

- For at vi ikkje skal ta med forstyrra målingar laga eg ei rutine for å sjekke at kompasset er påliteleg. Rutina antar at når roboten har kjørt i ei rett linje, skal ikkje målinga frå kompasset endre seg frå start til slutt meir enn variasjonen, (2 grader).

- Når roboten *ikkje* roterer, skal kun enkoderane brukast til å oppdatere estimatet for heading. Slik slepper vi unna variasjonane frå gyroen og kompasset når roboten kjører rett fram. Dersom roboten roterer, vil automatisk gyro bli brukt.



Figur 7.9: Illustrasjon av kompass støy

Ein svakheit med sjekken på kompasset er at om roboten kjører med små avstander inni eit forstyrra område, kan det være at målingane frå kompasset er innafor variansen. Konseptet er illustrert i figur 7.9. Ein svakheit med å ignorere gyro målingane er at vi ikkje plukkar opp veldig små endringar.

Når kompasset er "påliteleg" og roboten står i ro, oppdaterer vi estimatet med kompasset.

For å unngå plutselige endringar er variansen til kompasset satt opp mykje. Denne variabelen kan brukast til å tune estimatoren, og den heiter `COMPASS_FACTOR` i koden. Kalman-likningane er implementert som vist i kapittel 4 frå Brown & Hwang (2012).

Det endelige resultatet er vist i kodesnutt 7.1 på side 42.

Kodesnutt 7.1 Estimator task pseudokode

```
/* HENT MÅLEDATA FRÅ SENSORAR */
dVenstre = differanse for venstre hjul sidan sist iterasjon
dHøgre = differanse for høgre hjul sidan sist iterasjon
dRobot = (dVenstre + dHøgre) / 2
dEnkoderVinkel = (dHøgre - dVenstre) / WHEELBASE_MM
gyrZ = les gyro data, ignorer heading viss gyro dataen er veldig liten
kompass heading = les kompass data og trekk frå inital offset

/* Samansveising av vinkelsensorverdiar */
dHeading = gyroVekt * gyrZ + (1 - gyroVekt) * dEnkoderVinkel

/* PREDIKTER */
xhatt = sistXhatt + (dRobot * cos(sistHeading + 0.5 * dHeading))
yhatt = sistYhatt + (dRobot * sin(sistHeading + 0.5 * dHeading))
estimert heading = sistHeading + dHeading
filterKovarians += varians(gyro + enkoder)

/* Sjekk om kompasset er upåliteleg */
kjøreStatus = ta i mot status frå pose kontroller task
if (kjøreStatus seier vi er i ferd med å kjøre rett fram)
    gammel kompass heading = kompass heading
else if (kjørestatus seier vi har komt fram)
    differanse i kompass = kompass heading - gammel kompass heading
    if (differansen i kompass er mindre enn variansen til kompasset)
        kompasset er påliteleg
    else (differansen er større enn variansen)
        kompasset er ikkje påliteleg



/* KORRIGER */
feil = kompass heading - estimert heading
if (roboten står i ro og kompasset er påliteleg)
    kalmanGain = filterKovarians / (filterKovarians + COMPASS_FACTOR)
else (roboten kjører rett fram eller rett bak)
    kalmanGain = 0

estimert heading  += kalmanGain*(kompass heading - estimert heading)
filterKovarians = (1 - kalmanGain) * filterKovarians;

/* OPPDATER GLOBAL POSE */
global heading = estimert heading
global x = xhatt
global y = yhatt
Vent i 30 millisekund absolutt-tid
```

Kapittel 8

Nye funksjoner og løysingar

8.1 Kompasskalibrering

Dersom det vert gjort endringar på roboten, kan nye komponent gi innverknad på kompassmålingane. For å kalibrere kompasset er det lagt inn ein eigen definisjon som må endrast når mikrokontrolleren skal programmerast.

Når "`COMPASS_CALIBRATE`" er definert vert ein eigen kompass-kalibrerings-task kompilert og starta. Pose kontroll, estimator og sensor-tårn taskane vert ikkje aktivert. For å sette i gang kalibrering må roboten få `"{S,CON}"` frå sentral. Den vil då rotere ca 360 grader, og skrive ut ny verdi for x og y kompass-offset til den posisjonen den sto i. Det kan vera lurt å sjekke at offsetverdien er lik fleire plassar, for å sikre at kompasset ikkje vert feil kalibrert i eit område med magnetisk anomali.

8.2 Overgangen mellom $-\pi$ og π

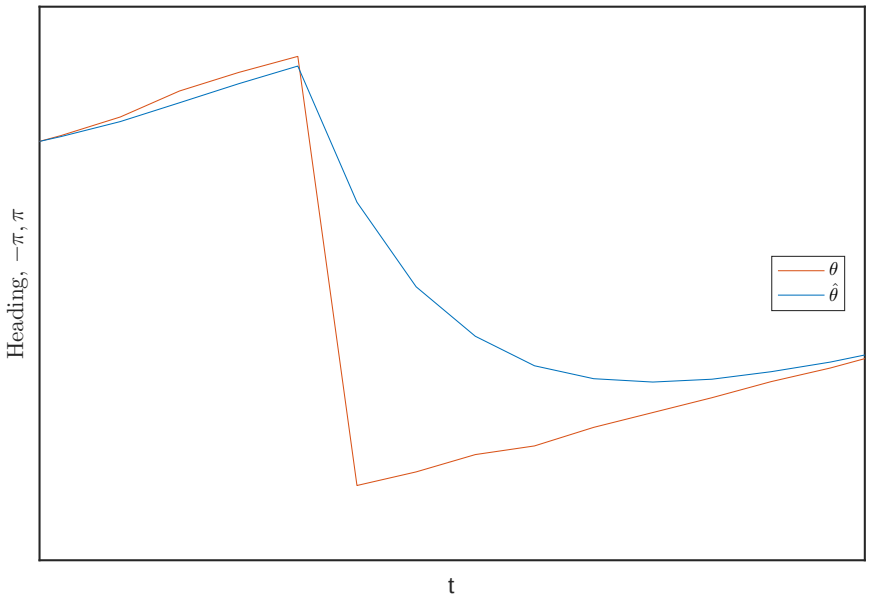
Ved samanlikning av vinklar hadde eg problem med at estimatet prøvde å integrere til rett verdi, når målingane gjekk "rundt". Til dømes når roboten står med ei heading der variansen til kompasset gjorde av vi vippa mellom -pi ogs pi. Dette vart løst ved noke enkle "if" sjekkar i ei løkke. Sjå kodesnutt 8.1. Denne funksjonen må kjørast kvar gang to vinklar skal samanliknast.

Problematikken er illustrert i figur 8.1, roboten roterer med konstant fart medan vi samanliknar kompass og estimat. Ved "vippepunktet" ser vi at estimatet må integrere til korrekt heading, og i tidsstega mellom er estimatet alle vinklar.

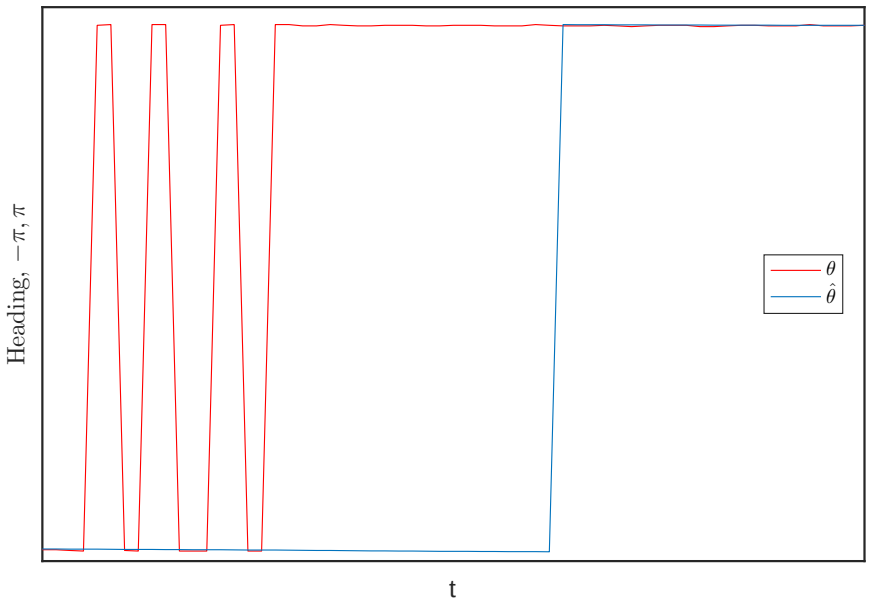
Etter korrigeringa, kan vi sjå at estimatet hopper rett til korrekt verdi i figur 8.2.

Kodesnutt 8.1 Sikring av overgangen mellom to vinklar som vert samanlikna

```
do
    if(vinkel større enn pi)
        vinkel -= 2*pi
    else if (vinkel mindre enn pi)
        vinkel += 2*pi
while(absoluttverdien til vinkel større enn pi)
```



Figur 8.1: Problem med å estimere headinga ved overgang frå $-\pi$ til π



Figur 8.2: Heading problemet er korrigert

8.3 UART interrupt og parsing

Tidlegare vart kommunikasjonen kjørt gjennom fleire løkker med switch case strukturar, i lange rekker med if-setningar på hexadecimal.

Det er ingen grunn til å skrive "0x0A" framfor "\n" eller "0x77" framfor "w", anna enn å gjera det vanskelig for andre å forstå kva ein ynskjer å oppnå. Etter koden har gått gjennom kompilatoren vert det heilt likt.

C har eit string-bibliotek som er utvikla, testa og optimalisert for formålet. For å tolke innkommande meldingar er det brukt "token" funksjonaliteten frå string biblioteket. Denne lar deg definere eit sett med symbol, som funksjonen då vil bruke til å dele opp meldinga og du kan slik lese rett verdi inn til rett variabel.

Implementasjonen er kun brukt i tilfellet der vi får oppdateringsmeldingane. Dei er nødd å være i følgjande strukur for at rett variabel skal få rett verdi: {U,vinkel,distanse}.

Dersom ei meir generell behandling skal implementerast må ein skrive ei utvida funksjon for dette. Den eksisterande rutina vil fungere som eit bra utgangspunkt til dette.

8.4 Printf og flyttal

"printf()" er brukt for å sende stringar av characters over USART. Det er i tillegg skreven ein eigen USART funksjon som heiter `vUSARTsendS(char *s)`. Printf er beskytta med ein mutex, medan vUSARTsendS ikkje er det. I initialisering før scheduleren er starta bruker vi vUSARTsendS, og ved feil som til dømes stack overflow, der RTOSet feiler.

Når koden vert kompilert i Debug mode kan printf også sende flyttal. Denne delen er fjerna under "Release Mode" då det ikkje vert brukt under normal drift.

8.5 Lågnivå antikollisjon

For redundans er ei lågnivå antikollisjonsrutne implementert. Denne tek målingar som allerede er henta, og stoppar motoren dersom avstanden er veldig kort. Sjå kodesnutt 8.2.

Dersom det er noko som er for nært vert det sendt eit stoppsignal til motorkontrolleren, og vidare vil roboten sende idle melding til sentralen, forhåpentligvis vil roboten få nye koordinat som ikkje sender den på kollisjonskurs igjen.

I utgangspunktet var det meint å bruke sinus/cosins for å finne avstanden til eit objekt foran, men dette vart fjerna for å kun ha heiltalsutrekningar. Ulempa er at minimumsavstanden vert sirkulær foran roboten, fordelen er at overheaden ved antikollisjonsrutina er liten. Sidan det er skjeldan at roboten faktisk kolliderer vart låg overhead prioritert.

Kodesnutt 8.2 Lågnivå antikollosjonsrutine

```
if (servovinkel ≤ 30°)
    objekt = foroverSensor * cos(servovinkel)
else if(servovinkel større eller lik 60°)
    objekt = høgreSensor * cos(270° + servovinkel)
else
    objekt = 0
if ( 0 < objekt < minimumsavstand)
    send stopp til kontroller
```

8.6 Tic Toc

For å ta tiden på funksjoner har eg definert to makroar som heiter tic og toc, der desse setter ein pinne høg og låg. Om vi set oscilloskop på pinnen så kan vi lese av kor lang tid det tok å kjøre funksjonen. Ved å ta fleire forsøk kan vi finne ei tilnærming på worst-case execution time (WCET).

8.7 Debug mode

I AS6 går det an å velge kva konfigurasjon du vil kompilere og laste opp på mikrokontrolleren. Dette har eg tatt nytte av ved å legge inn ein sjekk før alle debug-meldingar

Kodesnutt 8.3 Ved debug modus, skriv melding

```
#ifdef DEBUG
    printf("Debug message");
#endif
```

Dette er nyttig fordi då kan ein veksle mellom “Release” og “Debug” ved eit tastetrykk, utan å gjere andre endringar.

Kapittel 9

Scheduling

9.1 Fastsetting av periodetidene

Periodetidene vart bestemt av dynamikken til systemet.

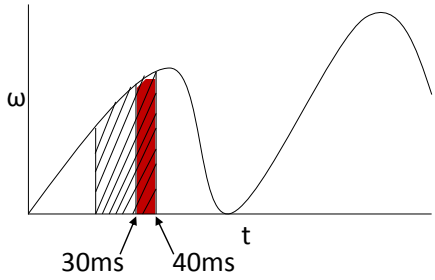
- SensorTowerTask: trade-off mellom stabilitet og respons, servoen må ha tid å svinge inn, og meldingane må rekke å nå fram før neste blir sendt. 200ms
- MovementTask: må høg frekvens då den behandler interrupt verdier, og styrer motorane. 20ms
- PoseControllerTask og PoseEstimatorTask har lengre kjøretid grunna flyttalsbehandling, men bør være raske nok til å styre roboten effektivt. 30ms

CommunicationTask blokkerer på ISR frå USART.

9.2 Valg av prioritetar

Applikasjonen er delt opp i fem tasker som går samtidig, i tillegg er det tre ISR som går regelmessig. Roboten kjørte først med round robin scheduling, og dette fungerte bra. Det vart likevel bestemt at prioritert scheduling skulle prøvast. I FreeRTOS er prioritet satt der eit høgare tal er høgare prioritet. På AVR er eit lågare tal for interrupt vektoren høgare prioritet. Prioriteten på avbrotsrutinene er hardware bestemt og går ikkje an å endre. Prioritetane på taskane vart først sett opp etter rate monotonic prinsippet, men estimator og UART tasken vart gitt høgare prioritet.

1. SensorTowerTask: Inkrementerer servoen med sensorane på i rett vinkel, tek IR målingar frå sensorane, og sender det til dongelen via USART. Denne har fått lågast prioritet, 1
2. PoseControllerTask: Tar inn pose frå estimatoren, og settpunkt frå kommunikasjonen. Denne vert blokkert heilt til posisjonsestimatoren gir den ein semafor, og har fått prioritet 2
3. CommunicationTask: Handterar USART ISR, og sender informasjonen videre til rett task. All informasjon til roboten går via denne tasken, og den har fått prioritet 3
4. MovementTask: Tar inn pådrag og rørsle frå Pose kontroller, samt enkoder verdiar, passer på at rørsla er optimal, og har fått prioritet 4
5. PoseEstimatorTask: Rekner ut ny Pose vha enkoder, gyro og kompass. Denne tasken har fått høgast prioritet, 5



Figur 9.1: Integrering av vinkelhastigheit, problem med ukorrekt tid

9.2.1 Prioritetsarv

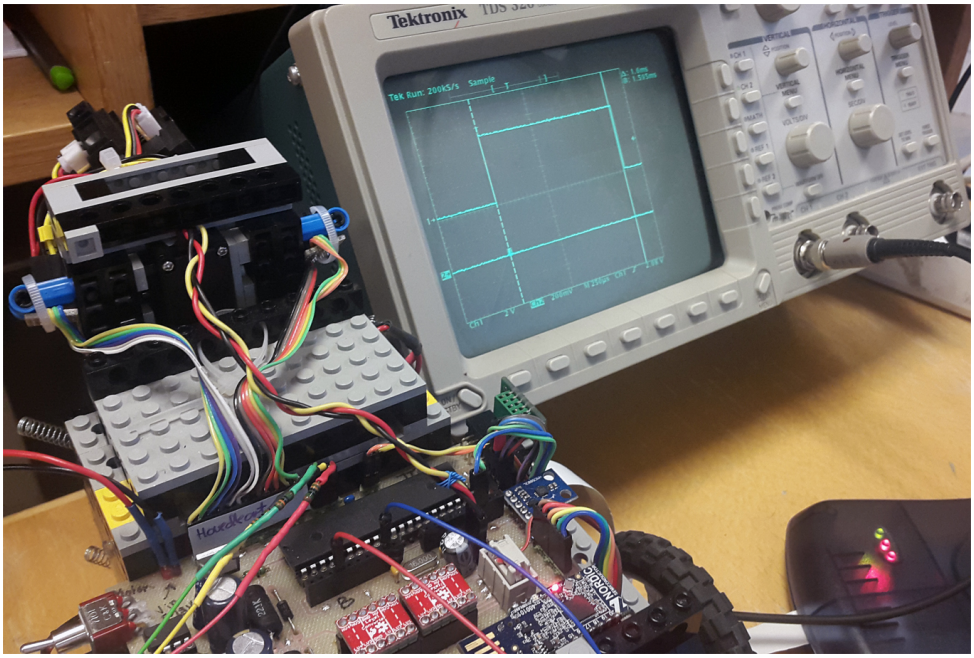
Både SensorTowerTask og PoseEstimatorTask bruker dei globale posisjonsestimata. Desse er beskytta med ein muteks. Prioritetsarv i FreeRTOS gjer at SensorTowerTask får høgast prioritet i det PoseEstimatorTask prøver å ta muteksen, som beskriven i Stallings (2015). Når SensorTowerTask gir tilbake muteksen, vert den pre-empta av PoseEstimatorTask fordi denne har høgare prioritet. For oversikt over synkroniseringsvariabler sjå kapittel 6.4 på side 30.

9.2.2 Prioritet på estimator

Estimatoren bruker den faste periodetida til å rekne ut vinkelhastigheita sidan sist sampling. Sidan målingane som kjem frå gyroen er vinkelhastighet per sekund, må måleverdien multiplisserast med samplingstida. Dersom estimatoren får avvik i tida kan det føre til ein bias i heading estimatet, denne fekk derfor høgast prioritet. Figur 9.1 viser eit døme der vi har varierande vinkelfart, og skravert område er vinkelfarta som vert integrert, det raude område er det som vert integrert men ikkje skulle vore med. Dersom tida vert lengre enn antatt, vil vi då få eit større bidrag enn vi eigentleg burde ha, då det ser ut som vi har rørt oss fortare enn vi eigentleg har. Den praktiske innverknaden dersom estimatortasken vert blokkert er ikkje målt.

9.3 Kjøretidsanalyse og teoretisk kapasitet

For å finne ei tilnærming på kapasiteten på prosessorane vart det gjort to analyser. Det er viktig å merke seg at dette kun gir oss ein peikepinn på den teoretiske kapasiteten. Antagelsar baserer seg på ein forenkla task modell, som gjer at analysa ikkje er direkte overførbar til vårt system.



Figur 9.2: Tidtaking av kjøretid

9.3.1 Metode for å finne tidparameter

Ved å bruke tic/toc funksjonen går det an å få eit anslag til kjøretida til tasken ved å bruke eit oscilloskop, sjå figur 9.2. For å ta tiden på kommunikasjonstasken og interrupt rutinene relatert til kommunikasjon laga eg eit matlab-skript som sendte meldingar til roboten, eller kobla til og fra bluetooth kommunikasjonen i faste intervall.

For å få eit anslag på WCET må også avbrotsrutinene takast med. Kernelen bruker Timer 3 på begge mikrokontrollerane. Denne har prioritet 32 på begge robotane, som betyr at dei andre avbrota har høgare prioritet enn tikk genereringa til FreeRTOS. Kontekstbytte og kernelbehandling er ikkje medtatt i utrekningane, og det er antatt at denne er neglisjer-bar.

Svakheita med å måle dette er at det er ikkje mulig å garantere at observert måling er den ekte WCET. For å analysere det nøyaktig må vi ha ein modell av prosessoren, pipelines, cache, minnetilstand og alt som hører til, og dette er ikkje ein del av skopet då det er veldig omfattande.

Periodetida til dei ikkje-periodiske taskane er “worst case” og ikkje tilfellet kvar gong. Til dømes er perioden til tikk interrupta satt til 0.8ms men dette er berre tilfellet

når motoren kjører på full fart.

Resultat etter tidtakinga kan du sjå i tabellane 9.2 og 9.3.

Noken av taskane er sporadiske eller aperiodiske, avhengig av viktigheita at dei vert kjørt innan deadline. Ein sporadisk/aperiodisk task med periode T=15ms kjem ikkje bli kjørt meir enn ein gong per 15ms. I realiteten skjer det skjeldnare, men responstidanalysa tar utgangspunkt i WCET og viss testen passerer så er vi trygg på at teoretisk maksimum rate er opprettholdt.

nRF status pin interrupten er ikkje tatt med i berekninga då denne ikkje inngår som ein del av drifta, dersom denne slår inn skal roboten enten starte eller stoppe og dei andre taskane skal ikkje kjøre.

Tabell 9.1: Tidtaking av meldingstid

Forsøk	Lengste [ms]	Kortaste [ms]
1	7705	(103)
2	8706	604
3	6794	1003

For å finne perioden til meldingane som vert sendt mellom sentral og robot vart det lagt inn ein metode for å måle lengste og kortaste tid på serveren. Resultatet er vist i tabell 9.1.

På første forsøket ein er kortaste tid svært kort i forhold til dei to andre forsøka. Dette var fordi at funksjonen som tok tida, også tok tida fra serveren sender pause til roboten, og då svarer roboten med ein gong. Dei to andre andre kortaste tidene er det som ein normalt vil ha ved sending av nye settpunkt til robotane.

9.3.2 Utilization test

Den første analysa som vart gjort var "Utilization test" (11.1) fra Burns & Wellings (2006).

$$\sum_{i=1}^{N} \left(\frac{C_i}{T_i} \right) \leq N(2^{1/N} - 1) \tag{9.1}$$

Testen er "necessary but not sufficient" men tek utgangspunkt i "The simple task model", rate monotonic scheduling, og krever at vi har følgjande eigenskapar:

- Ingen delte resurser
- Deadline er lik perioden
- Statiske prioritetar
- Kortare periode = høgare prioritet
- Kontekstbytte har ingen innverknad

Dette vil sei at testen eigentleg ikkje er gyldig for vårt system då vi har delte resurser, og ei anna type prioritering, og tasker som er aperiodiske og sporadiske, og kontekstbytte sansynlegvis har ei innverknad. Likevel er testen enkel nok for å sjå kor vi havner ytelsemessig.

Innsett med verdiane for AVR og Arduino robotane frå tabellane 9.2 og 9.3 får vi då ein utility på henholdsvis 0.1044 og 0.0948, begge mykje mindre enn 0.7241 som viser at vi er langt under grensa.

9.3.3 Response Time Analysis

Den andre analysa som vart gjort responstidsanalyse. Igjen er denne basert på antagelsar som ikkje stemmer med vårt system men analysa gir oss ein peikepinn på kva taskar som eventuelt får "dårleg" tid, fordi at vi tek omsyn til prioritetsrekkefølgja.

Responstidsanalyse, (11.5 fra Burns & Wellings (2006)).

$$\omega_i^{n+1} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{\omega_i^n}{T_j} \right\rceil C_j \tag{9.2}$$

Framgangsmåte:
Rekn ut ω_i^{n+1} *viss* $\omega_i^n = \omega_i^{n-1}$ *sett* $R_i = \omega_i^n$*, viss* $R_i < D_i$ *blir tasken ferdig å kjøre innan neste periode.*

Utrekningane er vist i vedlegg J frå side 111 til 120, og resultata er oppsummert i siste kolonna i tabellane 9.2 og 9.3.

Tabell 9.2: Periodar og kjøretid på tasker og avbrotsrutiner, AVR robot

Task	Prioritet	Tasknamn	T [ms]	C [ms]	B [ms]
A	ISR 1	Venstre enkoder tikk	15	0.0017	0
B	ISR 2	Høgre enkoder tikk	15	0.00164	0.0017
	ISR 3	nRFStatusPin	NA	0.009	
C	ISR 20	USART RX	103	0.00296	0.0034
D	5	PoseEstimatorTask	30	1.70	0.0098
E	4	MovementTask	20	0.043	1.7098
F	3	CommunicationTask	103	0.14	1.7528
G	2	PoseControllerTask	30	0.255	1.8894
H	1	SensorTowerTask	200	7.30	2.1784

Tabell 9.3: Periodar og kjøretid på tasker og avbrotsrutiner, Arduino robot

Task	Prioritet	Tasknamn	T [ms]	C [ms]	B [ms]
A	ISR 3	Høgre enkoder tikk	0.8	0.00061	0
B	ISR 4	Venstre enkoder tikk	0.8	0.00061	0.00061
	ISR 5	nRFStatusPin	NA	0.0068	
C	ISR 51	USART RX	103	0.0034	0.0122
D	5	PoseEstimatorTask	30	1.66	0.0058
E	4	MovementTask	20	0.041	1.6658
F	3	CommunicationTask	103	0.13	1.7068
G	2	PoseControllerTask	30	0.187	1.8368
H	1	SensorTowerTask	200	5.88	2.0324

Notasjon

- T_n: Periodetid
- C_n: Kjøretid, målt med oscilloskop, antatt å være WCET
- B_n: Teoretisk blokkeringstid, kor lenge tasken vert blokkert av andre taskar.

Vi ser at alle taskene får kjørt seg ferdig innafor periodetida si, estimator tasken vert i teorien blokkert i maks 0.0098 ms for AVR roboten og maks 0.0058 ms for Arduino roboten, noko som sansynlegvis ikkje er nok til å utgjere ein forskjell. Tasken med lågast prioritet, som tar målingar og skanner vert blokkert i 2.1784ms for AVR roboten og 2.0324ms for Arduino roboten.

Vi kan derfor slå fast at det sannsynlegvis er mykje kapasitet til overs med tanke på kjøretid på begge robotane.

9.4 Konsekvens dersom utrekningane er feil

Dersom antagelsane er så feil at utrekningane ikkje stemmer, vil det gi utslag i "starvation", taskane med låg prioritet vil få problem med å kjøre ferdig innan neste gang dei skal kjøre. Sannsynlegvis vil dette gi utslag i pose kontroller tasken, eller sensor tower tasken. På pose kontroller tasken vi vil merke det ved å til dømes sjå at roboten kjører lenger enn den skal, eller venter lenge mellom kvar gang den skal røre seg. På sensor tower tasken vil vi sjå at servotårnet ikkje beveger seg med ei fast rytme, at inkrementeringa på tårnet ikkje stemmer overens rørsla til roboten, og at meldingane som går til serveren får ei auka periode.

Eit døme på korleis dette ser ut går det an å sjå i videoen til Halvorsen (2014), på 1:30, der er rørsla til motorane styrt av ei interrupt rutine.

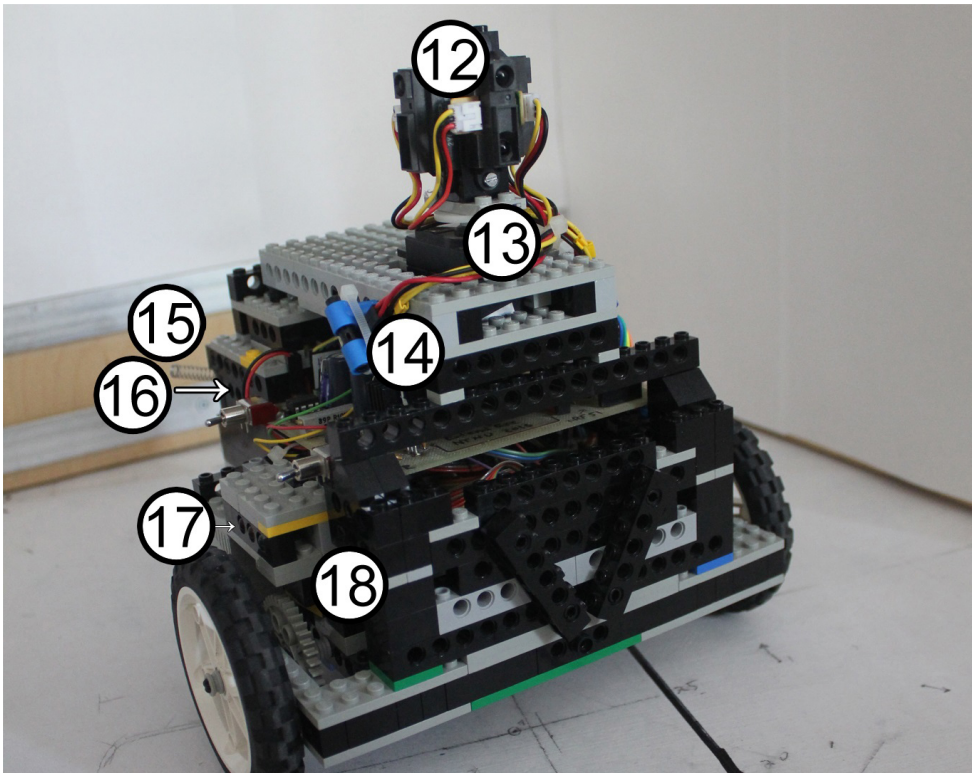
Eigentleg skal ikkje tårnet gå medan roboten kjører, men som videoen viser, og som forklart i Tusvik (2009) gjer det det likevel. Periodetida på tårnet er varierande, sansynlegvis på grunn av sanntidsproblem som har oppstått forsøk på multitasking ved bruk av interrupt rutiner som er for lange.

Gitt at vi ikkje får starvation er desse problema no løyst, timinga vert behandla av sanntids-kernelen til FreeRTOS.

Kapittel 10

AVR Robot

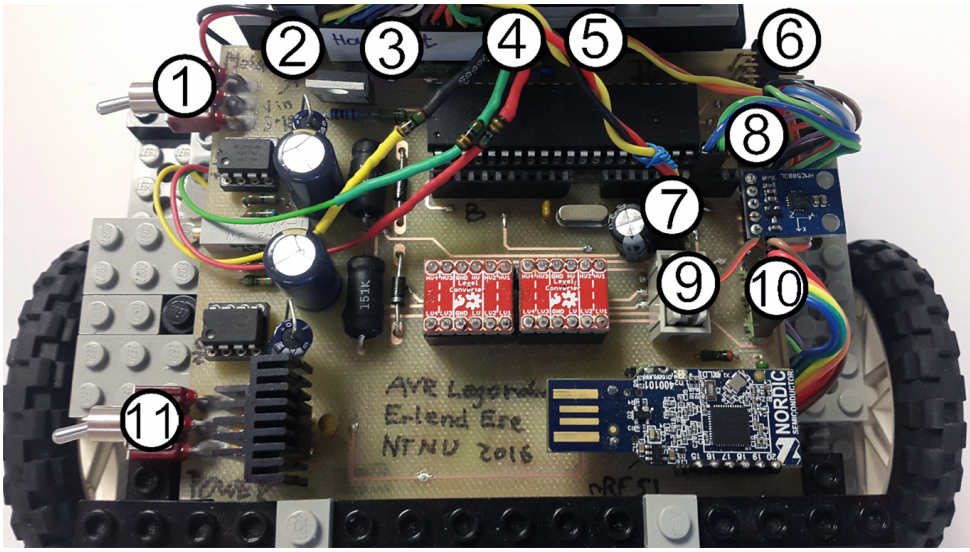
10.1 Oversikt AVR Roboten



Figur 10.1: AVR Robot vår 2016

Forklaring til bilde med nummer (figur 10.2 og 10.1)

12. Tårn med 4 stk IR sensorar
13. Servo
14. Skøytetilkoplingar til IR sensorar
15. Stålfjør til lading av batteri
16. Batteri (sjå pil, inn i roboten))
17. status LEDS
18. Enkoderar til motorane (inn i roboten)



Figur 10.2: AVR Robot, nytt kretskort vår 2016

1. Brytar til motor og servo
2. 2x1, Spenning inn frå batteri
3. 12x1 Tilkopling IR sensorar
4. 3x1 Tilkopling LED
5. 2x1 Tilkopling høgre motor
6. 5x2 JTAG for programmering og debugging
7. 3x1 Tilkopling servo
8. 2x1 Tilkopling venstre motor og 3x2 Tilk. enkoderar
9. 2x1 Tilkopling motorspenning, 9V
10. 6x1 Tilkopling IMU
11. Brytar til mikrokontroller

Spenningsregulatoren er montert med kjøleribber ved brytaren i punkt 11, kompasset står mellom punkt 8 og 10. nRF51 dongelen står nederst til høgre på kretskortet.

Dei raude firkanta korta i midten er level konverterar, for 5/3 volt. Det er plass å kople ein USB skjøteledning på nRF51 dongelen for debugging.

10.2 Endringar

Som referanse til den tidlegare versjonen av roboten, sjå figur 2.1 i Ese (2015).

Motorane var tidlegare kobla på 5 volt regulatoren som var meint til mikrokontrolleren, den vart flytta over til ein eigen 9v switch-regulator, og har fått sin eigen kontakt på kortet. Fordi at tannhjula til roboten hadde veldig liten klaring til gulvet, vart hjula bytta ut til nokon som var større. Roboten kjører no mykje fortare.

Roboten vart ombygd slik at toppen lett kan løftast for å komme til kretskortet.

Endringane som vart gjort på sensor-tårnet er beskriven i kapittel 5 på side 25.

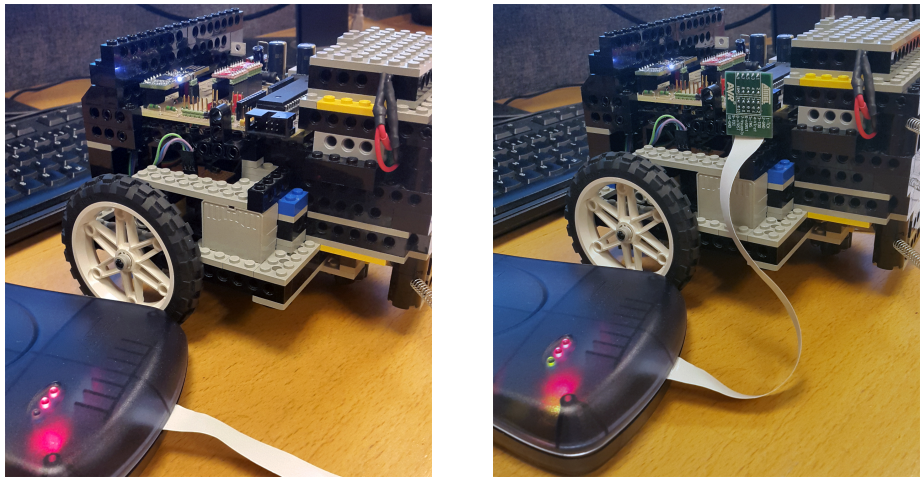
10.2.1 Nytt kretskort

På grunn av ny hardware og manglande utganger på det opprinnelige kretskortet måtte det lagast nytt kretskort til AVR roboten. RS232 chipen vart fjerna og det vart i staden laga eit footprint til for å setje nRF51 Dongelen rett på kortet. Det er montert to ekstra kondensatorar på 47μF og $22p$F ved headeren til henholdsvis sensorane og servoen for å filtrere vekk støy. I tillegg er det lagt fram fleire headers enn det var tidlegare, slik at det går an å komme til for å måle med logikkanalysator eller oscilloskop.

Dokumentasjonen frå tidlegare var mangelfull, og det gamle kortet hadde ein del avvik som vart korrigert. Kondensator-verdiar til switch-regulatorane var $22n$F i staden for 22μF, som beskriven i prosjektrapporten til Naess (2008). Resistorane som sto på det gamle kortet vart lodda av, målt, dokumentert og flytta over til det nye kortet. Det var ikkje avkoplingkondensator på mikrokontrolleren, men det var lagt inn ein kondensator foran ein pinout på kretskortet, som var nær nok til å berge mikrokontrolleren frå brown-outs. Dette vart korrigert ved å legge inn avkoplingskondensatorar på 100nF mellom VCC og jord på MCU, samt beskyttelse av resetlinja, som spesifisert i Atmel sin hardware instruks, AVR042.

Tilkoblinga til IMU går via ein headers og ledning, då IMU chipen er montert i bunn på roboten, mellom hjula, sjå figur 5.2. Kommunikasjon til kompasset går via I^2C. MOSFETen som styrer spenninga til IR sensorane er kopla opp med ein 1k formotsand og ein 10k pulldown mostand som er god praksis for å unngå oscillering. IMU og kommunikasjons-modulen har spenningsnivå på maks 3.3 volt, på grunn av dette må det nyttast logisk nivå-konverterar. Det vart brukt to ferdige modular til dette.

Kortet som står på i roboten er første versjon, og har ein del provisoriske løysingar som er lagt inn i endeleg versjon av kretskortet. Det endelige kretskortet er frest ut



Figur 10.3: Programmering via JTAG grensesnitt med AVR JTAGICE mkII

og klar til å bli lodda, og ligg i komponentkassen til AVR roboten. Skjematikken til den endelige versjonen av kretskortet ligg i vedlegg C på side 100.

10.2.2 Bytte av mikrokontroller

Som foreslått i Ese (2015) vart mikrokontrolleren bytta ut med ein nyare versjon ATmega1284p, som er fysisk lik, men har meir data og program-minne. Ved bytte av MCU vart det 16KB RAM i staden for 2KB, og klokkefrekvensen kan aukast til 20mhz mot tidlegare 16mhz, i tilleg til at vi har fleire funksjoner tilgjengelig på pinnane. Merk at kondensatorane til krystallen må byttast om denne byttast.

10.2.3 Spesifikke programendringar

Servodrivar

På grunn av den nye mikrokontrolleren kunne vi bruke ein 16-bit PWM til å styre servoen, og har no 1150 nivå å justere 0-90°, med 8-bit PWM hadde vi berre 18 nivå å styre servoen mellom 0-90°.

For å ikkje gjera flyttalsutrekningar online vart det laga ein tabell for å justere tårnet med 1 grad oppløsning. Tabellen vart laga i matlab, basert på målingar og lineær regresjonsanalyse. Skriptet til dette er lagt ved på DVD.

Soft-PWM

For å hastighetsregulere motorane til AVR roboten laga eg ein soft-PWM ved å bruke eigenskapane til FreeRTOS. Det går an å kalle delay funksjonen med dynamisk

ventetid. Sidan programvaren ikkje fryser som den ville gjort uten eit RTOS, kan vi så lage ein PWM liknandes funksjon. Metoden er vist i pseudokoden 10.1.

Det viste det seg at det var liten forskjell mellom å kjøre med 40% og 100% pådrag, men det gir ein fin effekt saman med referansemodellen som gjer at du akselererer og de-akselererer.

Kodesnutt 10.1 Soft-PWM

```
hent pådrag frå pose kontroller, lovlig verdi er 40 - 100%
pådrag =  PERIOD_MOTOR_MS * pådrag / 100
if (pådrag ≠ 0)
    Kjør motor
    taskDelay(pådrag)
    if(pådrag ≠  PERIOD_MOTOR_MS)
        Stopp motor
    taskDelay(PERIOD_MOTOR_MS - pådrag)
else(vi ønsker ingen pådrag)
    Stopp motor
    taskdelay(PERIOD_MOTOR_MS)
```

10.3 Programmering og debugging

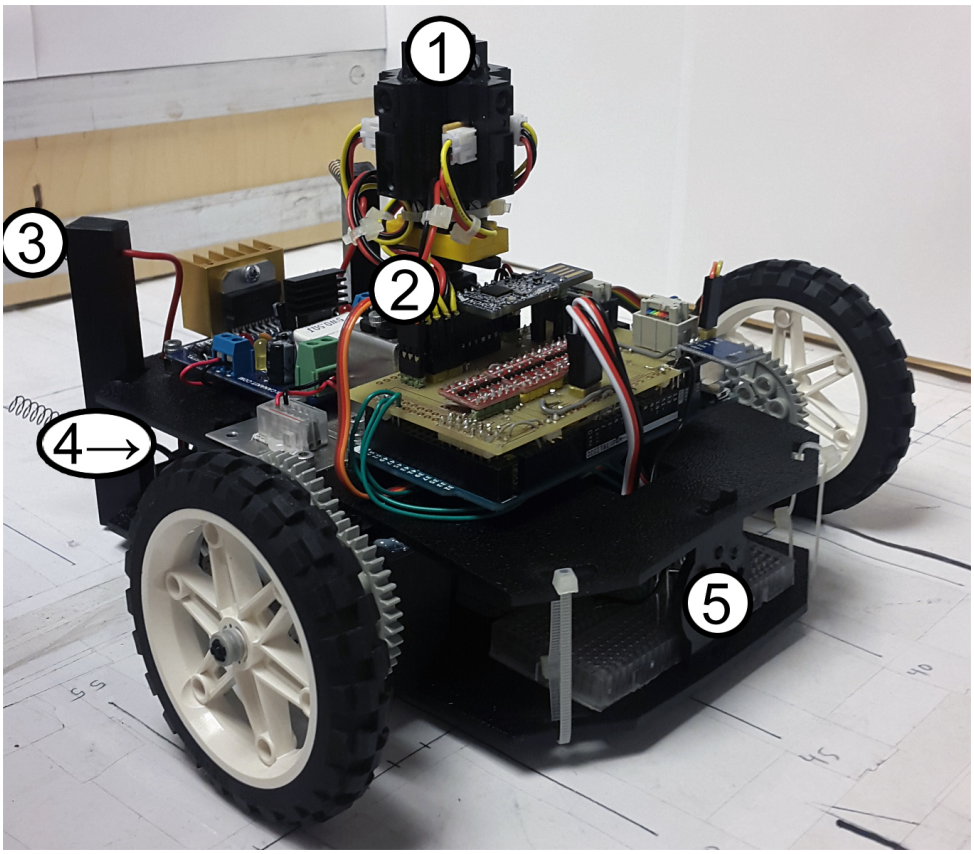
For å programmere AVR roboten bruker vi JTAG, på det nye kretskortet kan vi koble programmeraren rett i ein sokkel på kretskortet, sjå figur 10.3. Det er plass til å koble ein USB ledning til nRF51 dongelen, slik at det skal gå an å debugge både robot og kommunikasjon samtidig.

JTAG gjer at vi kan gå gjennom koden stegvis mens den kjører, men det endrer sanntidseigenskapane til roboten. Debugging også gjerast ved å lese av lysdiodane. Kva diodane signaliserer vert forklart i kapittel 6 på side 28.

Kapittel 11

Arduino

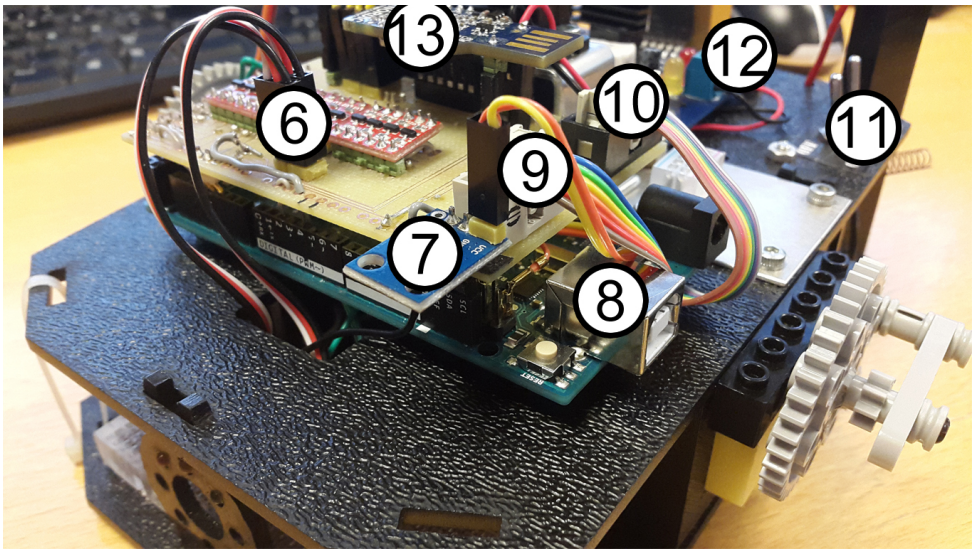
11.1 Oversikt Arduino roboten



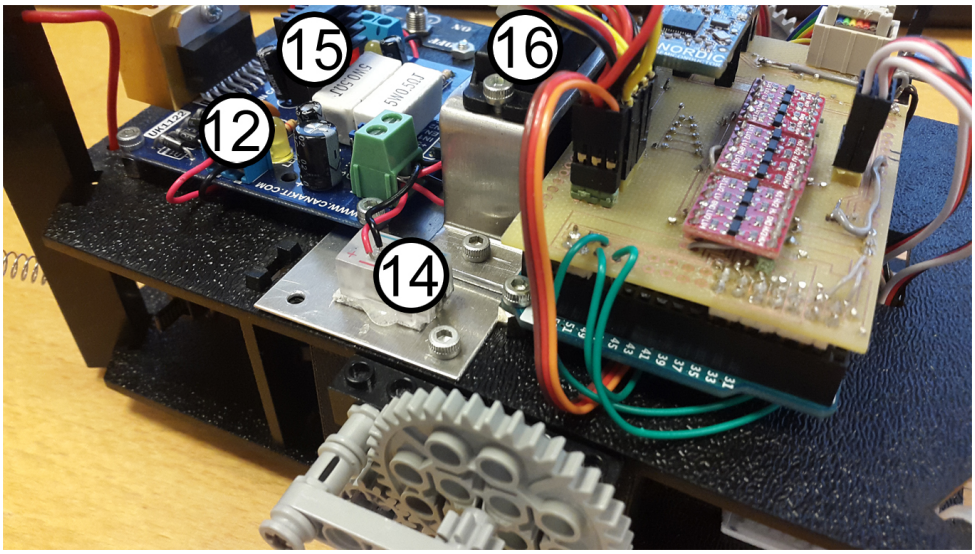
Figur 11.1: Arduino robot 2016

Forklaring til figurane 11.1, 11.2 og 11.3:

1. Tårn med 4 stk IR sensorar
2. Tilkopling IR sensorar og servo
3. Stålfjør til lading av batteri
4. Batteri (sjå pil, inn i roboten)
5. status LED
6. 3x2 Tilkopling enkoderar



Figur 11.2: Arduino kretskort med kompass



Figur 11.3: Arduino kretskort med tilkobling til spenningsforsyning

7. Kompass
8. USB inngang til programmering
9. 3x2 Tilkopling IMU
10. 5x2 Tilkopling Motorstyring
11. Brytarar til Arduino og ladefjør
12. Tilkopling til motor frå motorkort
13. nRF52 USB Dongel
14. Breadboard bit med muligheit for tilkopling med batteri. (slå av brytar 11)
15. Spenningsregulator
16. Servo

11.2 Endringar

11.2.1 Nedgiring

Motorane til roboten var veldig kraftige, og hjula hadde veldig lite friksjon. Det vart gjort ein test der roboten var satt til å rotere kontinuerleg mot klokka, og etter eit par sekund, satt til å kjøre rett fram.

Roboten roterer fort, når den blir satt til å kjøre rett fram, spinner hjula volsomt, og roboten "drifter" i aukande sirkler. Det er lagt ved eit opptak av dette på DVDen.

Det vart forsøkt å setje ei lågare spenning på motorane. Motorane treng ei viss spenning for å setje i gang, og ei lågare spenning for å halde konstant fart. Dette vart veldig vanskelig å regulere. Vi bestemte derfor at vi skulle prøve å gire ned motorane med LEGO tannhjul. Nedgiringa vart $\frac{16}{40} \cdot \frac{8}{40}$, og roboten har no eit meir stabilt tempo.

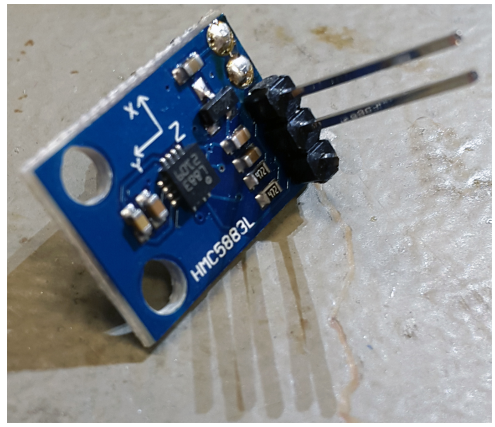
Endra senter for rotasjon

På grunn av nedgiringa hjula lenger fram på roboten, og senterpunktet for rotasjon har flytta seg. Offseten frå rotasjons-senterpunktet og senter på sensorane vert det ikkje tatt omsyn til i applikasjonen på serveren, og når roboten roterer vert kartet litt feil.

11.2.2 Kompass

På grunn av magnet-enkoderane var det eigentleg ikkje meininga at Arduino roboten skulle ha kompass. På grunn av at koden var enkel å implementere, vart kompasset montert likevel. Kompasset fungerer fint når det vert kalibrert, men det kan også enkelt fjernast dersom det gir uforutsette problem, då det er festa med lange pinnar som går gjennom kretskortet, figur 11.4.

NB! Merk at kompasset står opp-ned i forhold til AVR roboten, og i $Atan2$ funksjonen er x og y bytta om for at kompasset skal gi samme målingar.



Figur 11.4: Kompass med lange pinner

11.2.3 Spesifikke programendringar

Forskjellen på pinnar og portar og tilhørande register er endra i `defines.h`

servodrivar

Det vart brukt ein 16bit PWM til denne for å kunne gjere små endringar i posisjonen til sensortårnet. Same framgangsmåte som på AVR roboten.

FreeRTOS

På grunn av ein forskjell i arkitekturen mellom ATmega1284p og ATmega2560 fungerte ikkje portfilen til FreeRTOS som vart laga av Ese (2015). Ein "generell" konfigurasjon til AVR og Arduino dukka opp web: feilipu (2016), og denne vart implementert. Denne konfigurasjonen er unødvendig kompleks og har veldig mykje ekstra funksjonalitet, men lar brukaren velge mellom fleire timers, og passer til dei fleste versjonane av AVR / Arduino.

Det vart testa å bruke watchdog som RTOS timer, men det fungerte ikkje då minimumstida for eit "tikk" i FreeRTOS vart 18ms.

Motor regulator og PWM

Eg oppdaga at motorane på Arduino roboten kunne hastighetregulerast ved å bruke PWM på enable pinnane. To 8bit PWM utgangar vart brukt til dette. Vidare vart motor-regulatoren utvida som beskriven i kapittel 7 på side 33. Ut i frå målingar frå enkoder såg dette ut til å fungere bra, og ein PI regulator optimaliserte pådraget slik at motorane hadde same tikk-rate.

Interrupt handling

Ned-giringa gjorde at vi fekk veldig mange interrupt tikk frå enkoderen. Magnetskiva er festa direkte på akslingen til motoren. Frå akslingen er motoren gira ned internt i motoren, og utfor av oss. Resultatet er at vi får fryktelig mange tikk. Med motoren på fullt pådrag fekk vi 1250 tikk/s mot 66 tikk/s på AVR roboten, og 2475 tikk per omdreining på hjulet, mot 198 før nedgiring.

Når vi forsøkte å kjøre roboten rett fram gjekk det bra ca 70 cm, men så skar roboten ut til sida. Ifølge enkoderen drog roboten til venstre, medan den drog til høgre i virkeligheten.

Eg antok at dette var fordi interrupt vektorane har forskjellige prioritetar, og vi fekk så mange tikk, at ein interrupt pre-empta den andre. Høgre interupt har høgare prioritet enn venstre. ISR rutina manipulerte ein 16 bits variabel, etter å ha tatt to sjekkar på ein annan variabel, noko som kan ta litt meir tid enn nødvendig når vi har så mange avbrot. Antagelsen stemte godt overeins med det vi såg, og det vart prøvd å optimalisere interruptrutina.

Den nye metoden inkrementerer kun ein 8bit unsigned int. Motor-movement tasken, som har 20 millisekund periode, behandler interrupt variabelen, og setter den til 0 etterpå. Med 20 ms periode får vi maks ein inkrementering på 25 på ISR variabelen, så det er litt å gå på. Tikk variabelen vert så skalerert 10, og vi fekk då ca 247 tikk per omdreining på hjulet.

Etter dette vart implementert, vart det ei responstid på nesten ein tredjedel av det den var tidlegare, 610ns mot 1700ns.

11.3 Uforutsette problem

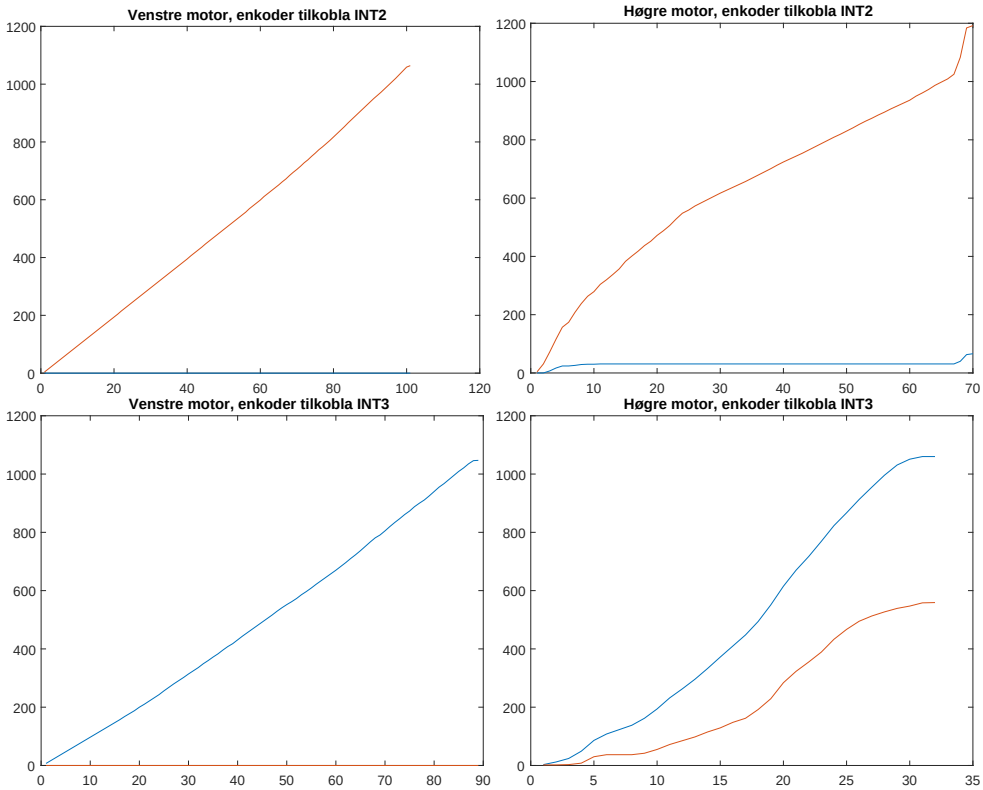
11.3.1 Ødelagt enkoder

Sjølv om ISR vart optimalisert, gjekk roboten fortsatt skeivt. Mistanken gjekk då over på enkoderen.

Eg gjorde ein test der eg kobla ut ein motor og tilhørande enkoder, og plotta verdien til begge enkoderane medan motoren gjekk. Motorane vart kjørt til enkoderverdien oversteig 1000 tikk.

Forventinga er at ein enkoder skal auke linjært, og tida den bruker på å nå 1000 skal være ca lik for alle tilfella. Den andre enkoderen som ikkje er tilkobla skal vise 0 for alle tilfella.

Resultata er vist i figur 11.5.



Figur 11.5: Rådata frå enkoder

På venstre motor er resultata omtrent som forventa.

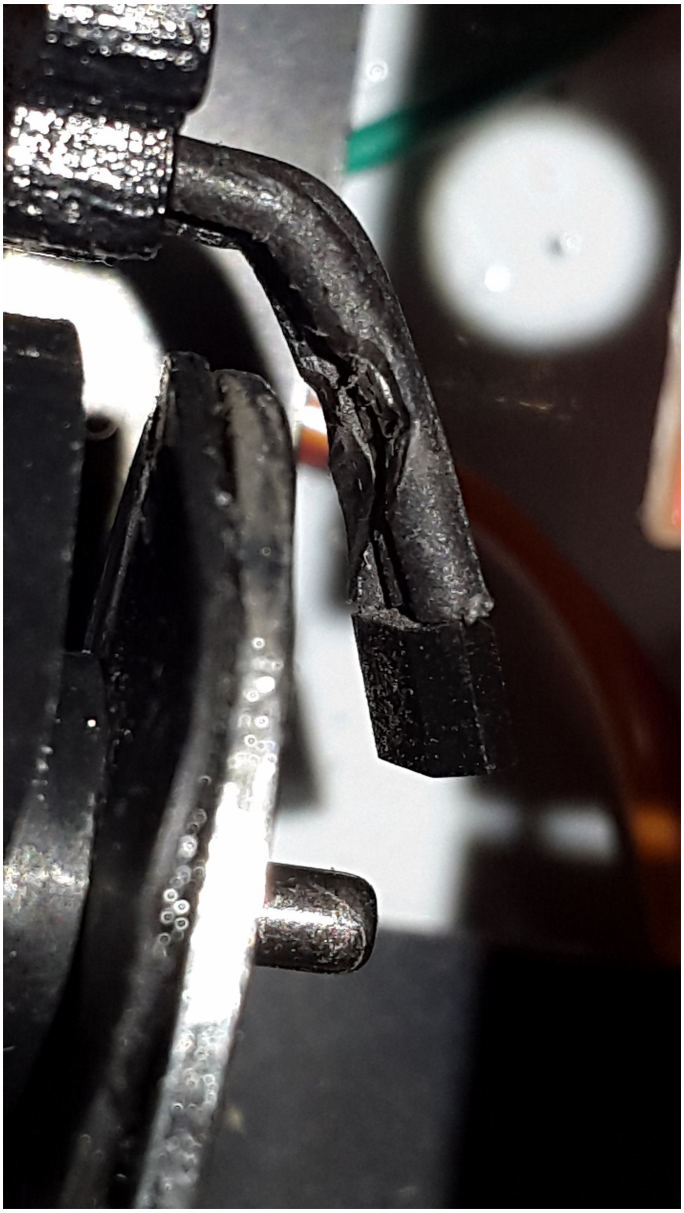
For høgre motor ser vi noko interessant. I begge tilfella aukar begge enkoderverdiane, sjøl om berre ein enkoder er tilkobla. Aukinga er ikkje linjær.

Årsaka til dette viste seg å være at sensoren hadde låg heilt inntil magnetskiva som roterer, når eg løfta vekk sensoren såg eg at isolasjonen var slipt vekk, sjå figur 11.6.

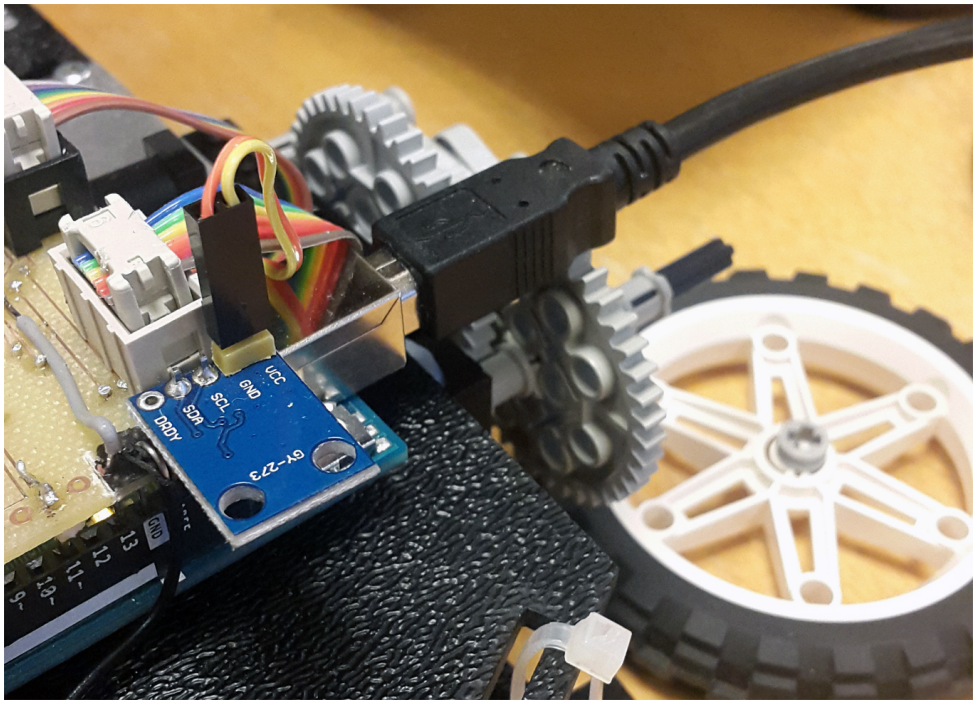
Sannsynlegvis har kortslutninga generert så mykje straum at det har gitt innverknad på den andre interrupt pinnen og gjort at begge har fått flanker.

Dette forklarer problema som oppsto, og fortel meg at regulatoren sannsynlegvis fungerte som tenkt, då tikka var like men roboten likevel kjørte skeivt.

Det var ikkje tid å få tak i ein ny enkoder.



Figur 11.6: Ødelagt magnetsensor ved roterande magnetdisk



Figur 11.7: Programmering av AVR robot via USB

11.3.2 Ødelagt avstandssensor

Thon (2016) oppdaga at IR sensoren slutta å fungere. Eg testa å bytte ledningen mellom to sensorar, og det såg ut til å være sensoren som var problemet. Sensoren vart ikkje bytta.

11.4 Programmering og debugging

For å programmere Arduino roboten er USB koplinga brukt, som vist i figur 11.7. Venstre hjul må av. Eg satt opp AS6 til å programmere roboten framfor Arduino IDE. Rettleiing til korleis dette blir gjort er gitt i Rødseth & Andersen (2016). I framtida er det nok å foretrekke å programmere denne med JTAG, dette kan gjerast utan inngrep på Arduino Mega kortet. JTAG gir mulighet for å lese av registera til roboten, og gå gjennom koden stegvis.

Kapittel 12

NXT Robot

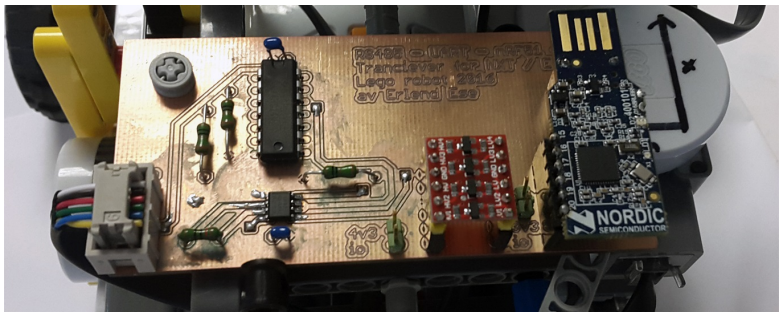
12.1 RS485 - USART Tranciever

For å få NXT roboten til å kommunisere med sentralen vart det laga eit kretskort med ein RS485 transciever, sjå figur 12.1. Skjematikken ligg i vedlegg. Løysinga vart inspirert av ein blogg (web: github Stefans 2016) som hadde gjort noko liknande for å kople USART sensorar til NXT. Dessverre vart det ikkje tid å prøve ut løysinga.

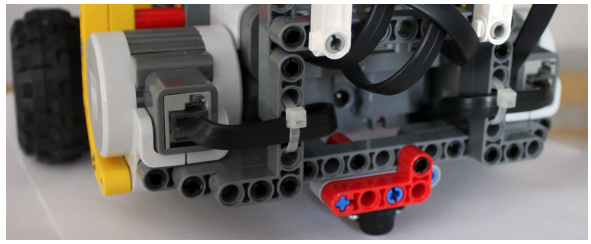
Om løysinga ikkje fungerer tilfredstillande er det også kjøpt inn I^2C til SPI bridge og ein kan slik kommunisere med nRF51 dongelen.

Det vart montert eit nytt slepehjul på roboten, sjå figur 12.2.

Det er skreve litt i kapittel 17 på side 91 om kva som må gjerast videre med NXT roboten.



Figur 12.1: Kretskort med nRF51 dongel



Figur 12.2: Nytt slepehjul

Del III

Resultat

Kapittel 13

AVR roboten

13.1 Eksperiment

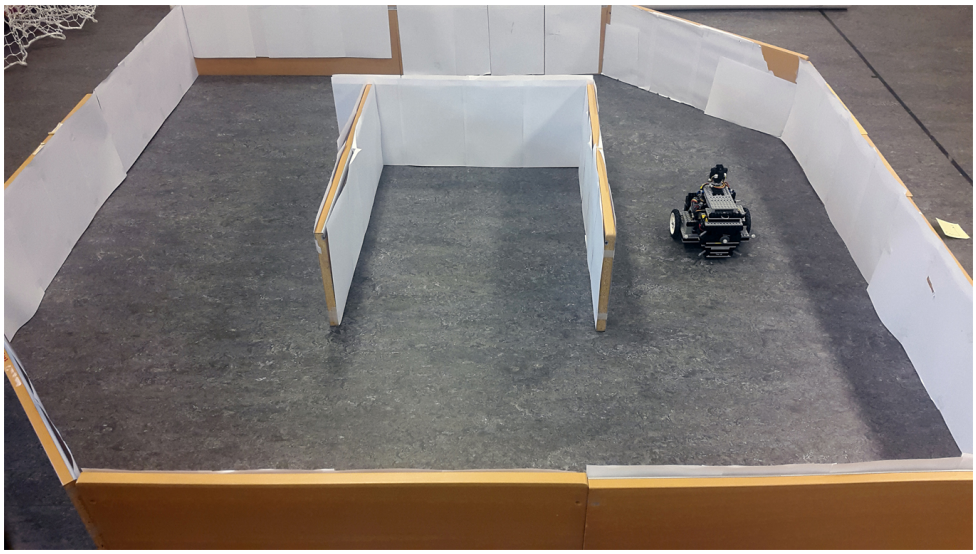
For å teste AVR roboten vart det gjort tre forskjellige eksperiment:

- Nytt forsøk av testen utført av Ese (2015)
- Manuell kjøring i slange labben med optitrack motion tracking system
- Samanlikning med og uten kompass

Storleiken på sirkelbana, som alle tidlegare testar har blitt utført i er ca 1.77m^2. Når roboten kartlegger eit så lite område, så er det veldig avgrensa kor mykje feil estimatoren kan å bli.

Den einaste testen som er gjort tidlegare med tracking system har vore gjort innanfor ein kvadratmeter Syvertsen (2005) s.38, og det vart derfor vanskelig å samanlikne testane vi gjorde. Det var ikkje nokre andre tidlegare testar å finne som var større enn denne sirkelbana, så testane som er utført på større baner står for seg sjølv.

Dei to siste testane vart gjort i slange-laben, som er utstyrt med eit optitrack system som bruker mange kamera og reflektorar til å nøyaktig posisjonere robotane.



Figur 13.1: Testoppsett i slangelab

13.2 Samanlikningstest i sirkulær bane

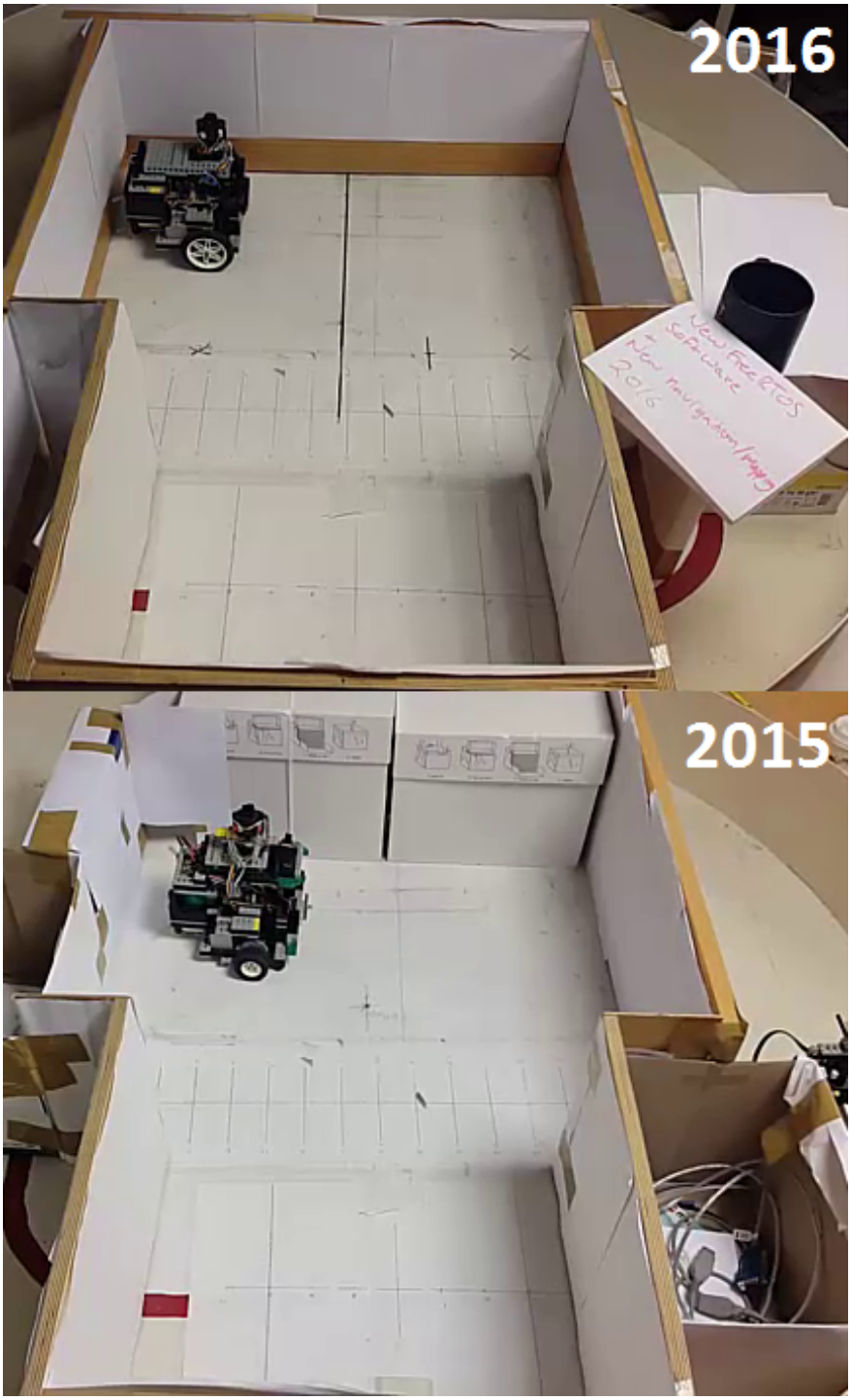
For å sjekke ytelsen til AVR roboten gjorde eg den same testen som vart utført før jul, i sirkelbana, sjå figur 13.3. Roboten vart styrt av server applikasjonen og kartla området. Før jul såg vi at vi fekk ei forskyving av kartet som auka medan roboten roterte. Hypotesa då var at dette enten skulda estimatorfeil eller sanntidsproblematikk. Etter å ha vurdert enkoder estimatet opp mot ekte verdi, er det sannsynleg at dette var skulda estimatorfeil.

Resultatet etter testen ser vi i figur 13.2. Det er blitt lagt ved ein video som samanliknar testen fra i fjor og år. På grunn av at roboten kjører med 9 volt på motorane, at hjula er større, og fordi tårnet roterer heile tiden, bruker roboten kortare tid på å kartlegge. På denne enkle testen brukte roboten ca 3 minutt på å kartlegge området før jul. Same området vart kartlagt på under 1 minutt i år.

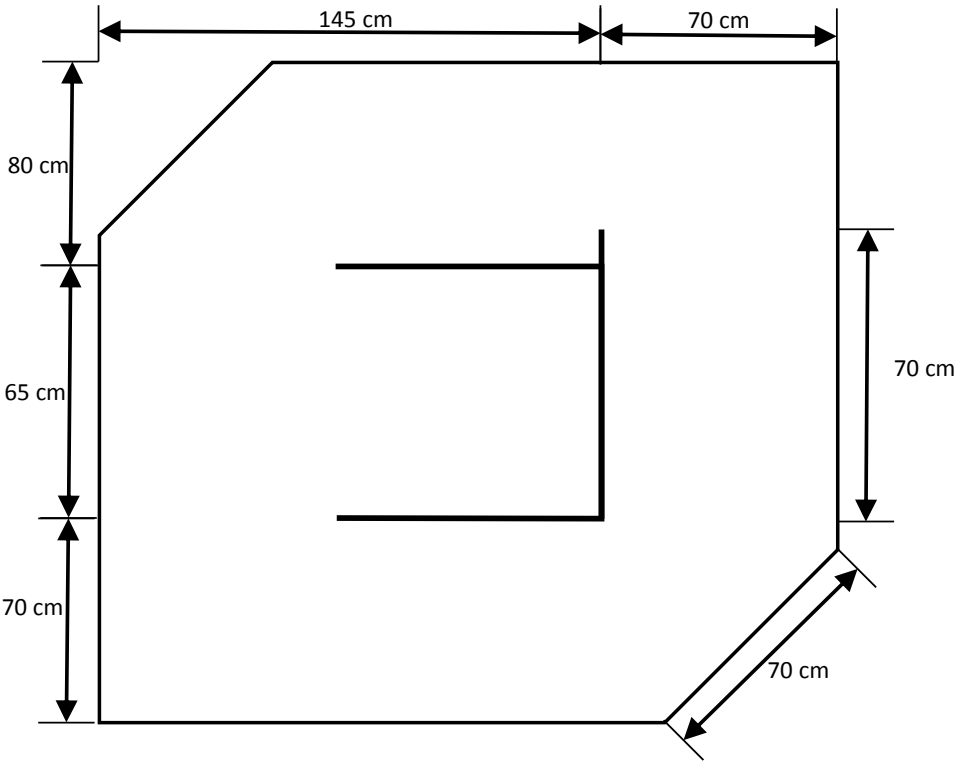
I filmen som er lagt ved, så er opptaket frå i fjor spelt av på 3x hastigheit medan det frå i år er spelt av på normal hastighet, vi ser då at robotane kjører ca. like fort. I tillegg ser estimatorfeilen ut til å være redusert betrakteleg.

Figur 13.2: Resultat, 2015 vs 2016 i sirkelbana på kontoret

Ein svakheit med testen er at roboten roterte mindre mot ein retning enn den gjorde i testen til Ese (2015).



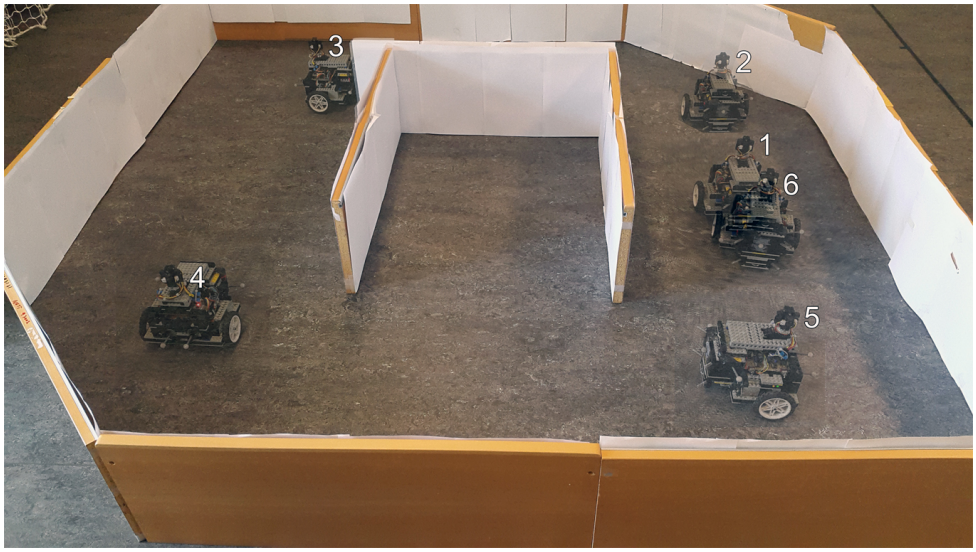
Figur 13.3: Testoppsettet for våren 2016 og hausten 2015



Figur 13.4: Dimensjonar på testoppsett, ca 4.38m^2

13.3 Test med Optitrack motion tracker kamerasystem

Optitrack motion tracker kamerasystem bruker mange kamera til å "måle" nøyaktig posisjon til eit objekt ved å sjå kvar nokre reflektorar står i forhold til kvarandre. Reflektorane reflekterte også målingane frå IR sensorane, dei måtte derfor plasserast som vist i figur 13.6. Optitrack systemet fungerer best om den reflektorane er plassert litt usymmetrisk. Skjermbilda med posisjonen til roboten og den estimerte posisjonen er lagt ved på CDen, men resultata er oppsummert i tabell 13.1.



Figur 13.5: posisjonane til roboten

Tabellens notasjon, alt er oppgitt i centimeter og grader

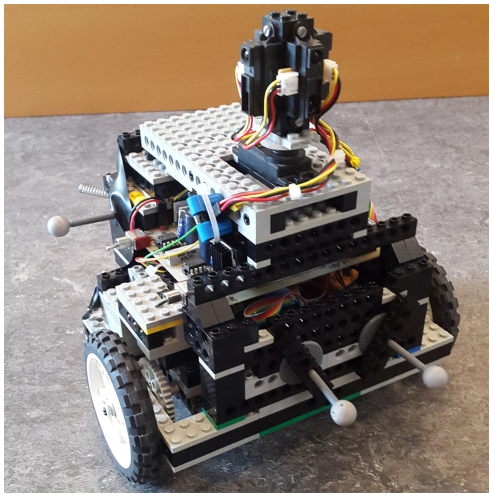
- x, y, θ er ekte posisjon og heading målt av kamerasystemet
- $\hat{x}, \hat{y}, \hat{\theta}$ er roboten sitt estimat av posisjon og heading
- e_x, e_y, e_θ er differansen mellom ekte og estimerte verdier

Tabell 13.1: OptiTrack Motion Capture Systems mot estimator resultat

	Kommando	x	y	θ	$\hat{x}$	$\hat{y}$	$\hat{\theta}$	e_x	e_y	e_θ
1	{U, 0, 60 }	0	0	0	0	0	0	0	0	0
2	{U, 90, 130}	57.9	-0.4	0	58	1	1	0.5	1.4	1
3	{U, 90, 120}	64.3	123.4	89.7	60	129	89	-4.3	5.6	-0.7
4	{U, 90, 130}	-50.8	132.3	181.4	-58	137	181	-7.2	4.7	-0.4
5	{U, 85, 52 }	-61.7	8.4	272.5	-52	9	275	9.7	0.6	2.5
6		-15.8	-2.2	351	-1	4	358	14.8	6.2	6.9

Testen vart utført ved å kjøre roboten i ein firkant inni labyrinten. Rørsla vart avgrensa til ein firkant for å halde det enkelt. Posisjonane roboten sto i er vist i figur 13.5.

Ei svakhet ved testen er at det gjekk lang tid mellom kommandoane var ferdig utført til ny kommando vart gitt. Dette var fordi det ikkje var mulig å gjere sanntidsopptak på testmaskina med programvaren til tracking systemet pga manglande harddiskplass på denne. Bakdelen med at det gjekk så lang tid mellom iterasjonane er at kalmanfilteret



Figur 13.6: AVR Roboten med reflektorar

fekk unaturleg lang tid til å konvergere estimatet mot kompasset, kontra det den vil få under automatisk navigasjon og kartlegging.

På siste iterasjon ser vi at feilen til kompasset er 6.9, mykje meir enn på dei andre iterasjonane. Dette er sansynlegvis fordi at kalmanfilteret ikkje fekk tid til at estimatet fekk konvergere mot kompasset før eg tok skjermbildet, om roboten hadde fått stått litt lenger slik den gjorde dei andre gangane, hadde feilen til headinga sannsynlegvis konvergert mot null.

Vi kan ta med oss følgjande frå denne testen:

- Fordi cutoff er brukt framfor band-limiter i posisjonsregulatoren har roboten ein tendens til å alltid kjøre litt kortare enn den får beskjed om. Dette er likevel ikkje så nøye sidan pose estimatoren er uavhengig av regulatoren, og det kjem fram at vi har kjørt litt for kort
- Kompasset fungerer veldig bra om roboten får stå i ro ei stund
- Det er ein del feil på estimatet som kompasset korrigerer

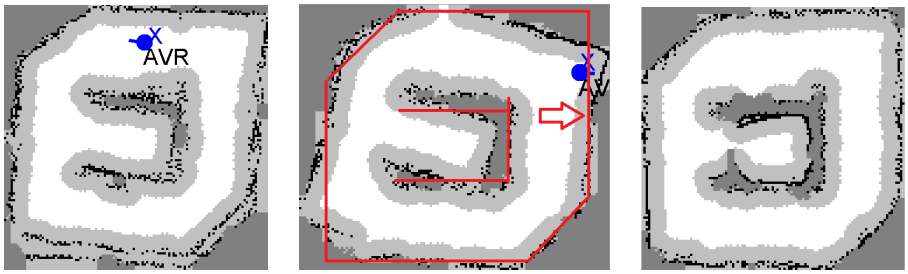
Det vart også gjort ein test der den automatiske navigasjon og kartleggingsalgoritmen i javaprogrammet styrte roboten, dette vart tatt opp saman med sanntidsdata frå optitrack programmet. På grunn av at vi kom seint i gang med testinga og mangel på harddiskplass på datamaskinen som kjører programmvaren på labben, fekk eg ikkje satt opp utstyret sånn at vi fekk logga og kunne analysere målingane på anna måte enn å ta skjermopptak. Resultata etter trackinga vert derfor ikkje vidare analysert.

13.4 Med og utan kompass

Ved for aggressiv konvergens mot kompasset ga plottet ingen meining lenger. Dette var fordi at magnetfelt ga store utslag på headinga til roboten.

Kommentar til figur 13.7. I det midterste plottet kan vi sjå at headinga til roboten har fått eit hopp der den raude pila peiker. Dette er ein vegg som eigentleg er bein, og har fått ein knekk. I dette området var det alltid dårlege resultat frå kompasset. Sannsynlegvis magnetisk anomali i gulvet ein plass, og vi får ein magnetisk anomali. Plottet til høgre ser veldig bra ut. Forskjellen mellom desse to testane er at variabelen for variansen (`COMPASS_FACTOR`) til kompasset i kalman-filteret er veldig stor i den siste testen, så det tar veldig lang tid før målingane frå kompasset får nokon innverknad på estimatet.

Denne måten gjer at estimatet kun konvergerer mot magnetisk nord over lang tid. Medan roboten rører seg rundt i eit område vil kompasset då gjere at den estimerte headinga sannsynlegvis konverger mot korrekt retning.



Figur 13.7: Uten kompass, for mykje kompass, og akkurat passe

Det skal seiast at ikkje alle testane ga like fine resultat som i plottet til høgre, og det vart ved fleire høve resultat som likna meir på plottet til venstre. Dette er avhengig av kor lenge kalman-filteret får tid til å konvergere estimatet mot korrekt heading mellom kvar gang roboten kjører til eit nytt område, initial-posisjonen til roboten i forhold til magnetiske anomali, og kor mange gangar roboten roterer mot høgre kontra mot venstre.

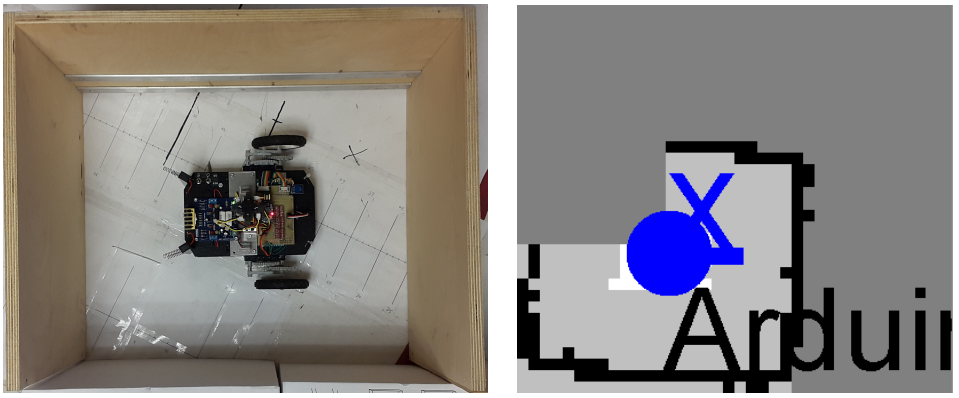
Kapittel 14

Arduino roboten

14.1 Eksperiment

For å verifisere at Arduino roboten fungerte vart det gjort to eksperiment. Grunna den ødelagt enkoderen var det ikkje mulig å teste kjøringa.

1. Kartlegging i firkanta boks
2. Kartlegging saman med AVR roboten



Figur 14.1: Oppsett og resultat etter test i boks

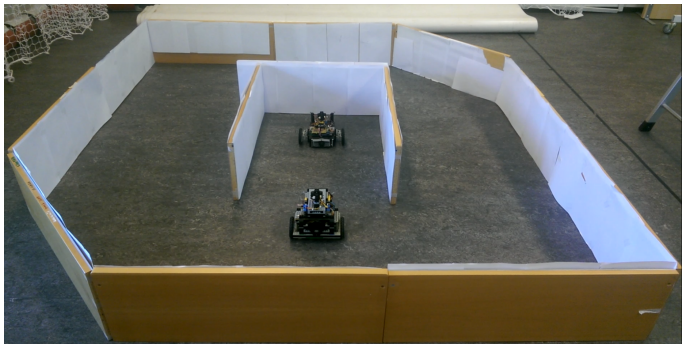
14.2 Test i boks

Resultatet etter plottet samsvarer med boksen den sto i, sjå figur 14.1. Vi ser at kvadranten oppe til venstre ikkje vert kartlagt, dette er på grunn av ein IR sensor som ikkje lenger fungerer som beskreven i slutten av kapittel 11.

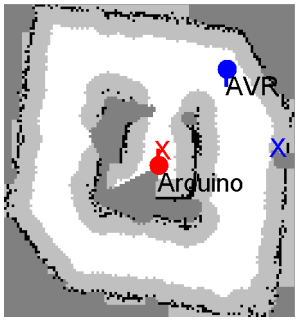
14.3 Samarbeidande navigasjon og kartlegging

Roboten måtte stå i same posisjon under samarbeidande kartlegging. Om enkoderen var i orden er det svært sansynleg at roboten ville våre lik på AVR roboten, då programmet fungerte på lik måte.

Estimatoren for headinga til roboten vart endra til å kun være basert på gyro målingar, og kompasset korrigerer alle estimata heile tiden. Dette gjer at roboten



Figur 14.2: Test-oppsett for samarbeid



Figur 14.3: Samarbeid resultatplott

kan rotere for eigen maskin og estimere headinga. Resultata etter skanningane til Arduino roboten, og estimata som vart generert såg korrekt ut, men dei vart ikkje sjekka mot optitrack systemet.

Bortsett frå problematikken med avstandssensoren som ikkje fungerte, den øydelagte enkoderen til hjula, og offseten til tårnet fungerte programmet til AVR roboten med eit minimum av endringar på Arduino roboten.

Resultata etter samarbeids-testen er vedlagt i rapportane til dei to andre oppgåvene Thon (2016), Rødseth & Andersen (2016). Videoen til dette vart lagt ved deira rapport.

Vedlagt er ein video frå testinga som vart gjort under utviklinga av motor regulatoren til Arduino roboten, denne videoen var ikkje meint for å være ein del av dei endelege vedlegga, men igjen fordi at enkoderen rauk vart det ikkje mulig å vise på nokon annan måte at roboten kan kjøre. På videoen vert differansen mellom motorane korrigert ved samanlikning av tikk.

Kapittel 15

Kommunikasjonen

15.1 Eksperiment

For å teste kommunikasjonen vart det gjort to eksperiment:

1. "Vanleg" navigasjon og kartlegging i labyrint i område med god dekning
2. Kjøring ut av dekningsområde

Den første testen vart utført i slangelabben, vi har god dekning i heile rommet. Datamaskina sto i hjørnet og labyrinten var 5-8 meter unna.

I den andre testen satt vi roboten til å kartlegge i gangen utfor slangelabben, og lot den kjøre til vi ikkje lenger fekk oppdateringar frå den. Rødseth & Andersen (2016) gjorde fleire testar på rekkevidde og frekvens mellom nRF51 sentral og periferi.

15.2 Navigasjon og kartlegging

Generelt fungerer kommunikasjonen svært tilfredstillande og virker redundant. Vi fekk kjørt svært mange testar, kommunikasjonen er robust og fungerer kvar gang ved rett bruk. Det hender at vi får fragmenterte eller forsinka meldingar, men på grunn av måten systemet er bygd opp har dette liten innvirkning om vi mangler eit par oppdateringar innimellom. Den generelle oppdateringsfrekvensen er 5 meldingar per sekund, som forventa. Det er veldig lite forsinkelse frå ein melding blir sendt til roboten utfører handlinga, sansynlegvis under 50ms.

15.3 Testing forbi dekningsområde

Når robotane kjører langt vekk frå sentralen, ser vi ei degradering i form av fragmenterte medlingar, eit døme er vist i kodesnutt 15.1. Dette gjer at det går lengre tid mellom gyldige pakkar vert mottatt.

Ein pakke startar med avsendar og indeksnummer, her er det AVR og 0. Dei to første og to siste meldingane er korrekte, den fjerde frå toppen kan sjå korrekt ut, men den inneheld for mange tal.

Vi ser at det kjem ei idle-status melding midt i som er korrekt. Det er fordi denne vert sendt som ein pakke. Vi kan også sjå at det tilsynelatande kjem meldingar som berre innehelt til dømes "}". Grunnen til dette er at vi kan være uheldige og sende

slutten på stringen, ein "\n" i ein pakke for seg sjølv, og denne kan bli levert rett etter, eller rett før ein anna fragmentert melding.

Etter kvart som dekninga til roboten vert dårlegare får vi fleire og fleire fragmenterte meldingar. Sidan det går an å sjå at pakkene vert fragmentert vart det implementert ein integritetstest på serveren der vi kan sjå andelen fargmenterte meldingar. Dersom robotane kjem langt vekk, kan brukaren eller programmet enkelt sende ein melding til roboten om å snu og kjøre tilbake.

Vi prøvde dette manuelt.

Vi lot roboten kjøre heilt til applikasjonen i programmet nesten ikkje fekk meldingar som vart godkjente til å oppdatere kartet. Vi sendte så ein manuell melding til roboten om å snu, og kjøre tilbake. Denne meldingen vert sendt i ein pakke, og den vart levert til roboten som planlagt i samsvar med bluetooth protokollen Bluetooth-spec (2010). Roboten snudde og kjørte tilbake, og oppdateringane i kartet vart oppdatert igjen så fort dei inn-kommande meldingane var godkjent.

Dersom roboten er så langt vekke at meldinga ikkje når fram, vil den automatisk miste tilkoblinga. Ettersom programmet på roboten heile tiden sjekkar at dongelen er tilkobla, vil roboten då stoppe så fort den mister kontakten. Her er det rom for backtracking.

Kodesnutt 15.1 Døme på fragmenterte meldingar

```
[0]:AVR:{U,-177,-61,-68,49,0[0]:AVR:,0,0,0}
[0]:AVR:{S,IDL}
[0]:AVR:{U,-177,-61,-68,47,0[0]:AVR:-68,46,0,0,0,0}
,0,0[0]:AVR:,0}
{U,-177,-61,{U,-[0]:AVR:,0}
{U,-177,-61,-72,[0]:AVR:,-69,45,0,0,0{U,-177[0]:AVR:,0,43,0,0}
}
{U,-177,-[0]:AVR:,-63,39,0,34,0,0}
,0[0]:AVR:,-74,41,0,36,0,0}
{U[0]:AVR:,-177,-61,-59,38,0,3[0]:AVR:77,-61,-60,37,0,32,0
[0]:AVR:{S,IDL}
{U,-1{U,-177[0]:AVR:0}
,0}
61,-73,42{U,-[0]:AVR:,23,80,0}
[0]:AVR:4,0,22,70,0}
36,0,32[0]:AVR:40,0{U3,0,0}
2,0,0,0}
}
177,-61[0]:AVR:L}0,17,58,0,-177{U,-[0]:AVR:171,-6{S,ID
[0]:AVR:{U,-177,-60,142,29,0[0]:AVR:,16,0,0}
```

Del IV

Drøfting og videre arbeid

Kapittel 16

Diskusjon

16.1 Trådlaus kommunikasjonsmodul

Kommunikasjonsløysinga ved prosjektstart vart implementert for over 10 år sidan. Det har skjedd mykje på "trådlaus-fronten" sidan då. BT1 var ein stor flaskehals i systemet, og vart derfor bytta ut. Løysinga vi gjekk for, nRF51 montert på ein USB dongel, var den som brukte minst straum av alternativa vi såg på, men var litt meir utfordrande å få til å fungere enn viss vi hadde til dømes valgt wifi, som er ein godt utbredt løysing i "maker" miljøet, og har derfor rikeleg med dokumentasjon og eksempel på løysingar.

Ei utfordring med løysinga var storleiken på pakkane som kunne sendast, då pakkane som vert sendt er avgrensa på 23 byte. Lengre meldingar verdt oppdelt i fleire pakker. Dette gjer at vi opplever fragmenterte pakker dersom signalkvaliteten går ned. Vi løyste dette ved å sjekke stringane som vart mottatt på server siden. Løysinga vi brukte gjer at vi har ein viss margin for å kunne kalle roboten tilbake før vi mister kommunikasjonen heilt. Det fins nok betre måtar å løyse dette på. Det er til dømes mulig å gå inn i BLE protokollen og sortere pakkane på mottakarsida før dei vert send vidare til programmet på serveren.

Kommunikasjonen kan no debuggast på fleire måtar, og det er lett for brukaren å lokalisere og korrigere feil eller uønska oppførsel frå dongelen. Dette var eit stort problem tidlegare i prosjekta. Under prosjektet har vi ikkje hatt tilfelle der dongelen på roboten ikkje har fungert, truleg fordi programmet på denne i hovudsak er utvikla av applikasjonsingeniørane til Nordic Semiconductor, og eg har kun gjort små endringar for å tilpasse denne til roboten.

På server sida er programvaren til dongelen basert på to eksempel frå Nordic, men sett saman av ein tredjepart. Programvaren har hatt ein del bugs som vi har fiksa undervegs. Når den skal brukast, må brukaren følgje spesifikke retningslinjer for at dongelen ikkje skal slutte å fungere, så det er rom for forbetring på denne. Til tross for dette, dersom desse retningslinjene vert fult, er det lite problem relativt til slik løysinga var tidlegare.

16.2 Oppgraderingane til AVR roboten

For å få bruke dongelen på AVR roboten var eg nøydd å designe eit nytt kretskort på grunn av IO på mikrokontrolleren ikkje var gjort tilgjengelig på det tidlegare kretskortet. Dette skulle eigentleg vore fort gjort, då skjematikken som var brukt

skulle vera ein del av vedlegga på dei tidlegare prosjekta. Etter litt leiting oppdaga eg at den einaste skjematikken som var tilgjengelig var i dårleg oppløyste bilete vedlagt i rapportane, og dei nyaste designfilene var ein eldre versjon av kretskortet.

Det gjekk derfor mykje tid i å lære seg kretskortdesign og lage eit kort som vart brukbart. Under arbeidet med kretskort kom det fram ein del avvik på det gamle kortet, dette vart korrigert og det vart mindre forstyrrelsar på spenninga frå servo og IR dioder som switcher. Kretskortet som står på er prototypen, men det er produsert eit nytt kretskort der alle endringane på prototypen er tatt med, komponenta må berre loddast på.

Hjula på roboten vart skifta ut fordi at roboten kjørte svært sakte, og tannhjula til motoren hadde veldig lite klaring til gulvet. Sidan motorane kjører på 9V i staden for 5V og hjula er større, så kjører roboten fortare. Den kjører likevel mykje seinare enn Arduino-roboten og NXT/EV3 robotane.

Byttet av mikrokontroller frå ein 8bit AVR til ein anna 8bit AVR vart gjort fordi at i utgangspunktet var det ikkje planar om å lage nytt kretskort, og mikrokonrolleren det vart bytta til passer rett i samme footprint som den gamle. I ettertid ser eg at eg burde heilt klart heller brukt ein 32bit microkontroller når kretskortet måtte byttast likevel. Til dømes kunne den nye nRF52 DK erstatta både dongelen og AVR mikrokontrolleren.

Vi oppdaga at kalibreringa til sensorane truleg har ein bias, og dette viser seg når Thon samanliknar kartet mellom ekte og simulert. Det er då rimelig å anta at den generelle kalibreringa ikkje er god nok likevel, og bør gjerast på nytt for kvar enkelt sensor.

Til IMU vart det brukt SPI kommunikasjon. I etterkant ser eg at eg burde brukt I^2C både på IMU og kompass for å ikkje bruke så mange pinner. Fordelen med SPI er at det går veldig mykje fortare enn med I^2C. Men den praktiske forskjellen er sannsynlegvis ubetydelig.

16.3 Forbetring av posisjonsestimat

Ein av dei svakheitane med roboten tidlegare var at posisjonsestimatet kun baserte seg på verdiar frå enkoderane. Desse er utsett for drift, og forholdet mellom tikk og avstand er heilt avhengig av friksjonen til underlaget. Det har tidlegare vore brukt eit kompass, men dette har vore fjerna i ettertid. I denne oppgåva har eg implementert eit kompass på nytt, og kan no forstå kvifor dei fjerna kompasset tidlegare. Kompasset er svært upåliteleg ved innandørs kjøring.

Alt på roboten som er magnetisk vil gi innverknad på kompasset. I tillegg vert

kompasset forstyrra av stålbjelkar i gulvet, leidningar, bordbein, kabelbruer, ventilasjonskanalar og meir. Kompasset saman med alt dette gjer at estimata på headinga til roboten vert svært dårlege dersom kompasset får gi for mykje bidrag i estimatet.

Eg har laga ein metode for behandling av kompassdataen, men dersom vi lot kompasset oppdatere estimatet for fort, kom kompass-svingingane med og gjorde at kartlegginga vart veldig dårleg fort. Løysinga var å sette opp variansen til kompasset mykje, slik at det berre vert bidrag over tid som gjeld.

I tillegg til å prøve eit kompass på estimatoren, vart ein IMU implementert. I denne vart kun gyroen brukt for å lese av vinkelfarten til roboten. Ved testing i slange-laben med motion capture kamera fant eg ut at under rotasjon, så gir gyroen litt for låg verdi, medan enkoderane gir litt for høg. Det vart prøvd å multiplisere gyroen med ein konstant, men dette fungerte ikkje då biasen til gyroen i forhold til ekte verdi ikkje var konstant ved varierande vinkelfart.

Under enkel testing med måleband fant vi ut at roboten er ganske nøyaktig når det gjeld linjære rørsle. Eg sjekka aldri nøyaktigheita til dette over lang avstand, noko eg burde gjort i slange-labben.

Det vart oppdaga stor forskjell mellom å kjøre på golvbelegget i slange-laben og på ei tre-plate som hadde meir friksjon.

16.4 Ny programvare på roboten

I fjor vart FreeRTOS testa og lagt inn på AVR roboten, men programmet som kjørte på roboten var ikkje tilpassa for å ta nytte av moglegheitene til dette. No er heile programmet omskriven, og delt opp i modular, slik at brukaren enkelt kan aktivere eller deaktivere dei ved behov. Strukturen til programmet er heldt slik at FreeRTOS filene er delvis abstraktert vekk, men all programvare som er relevant til roboten ligg på topp.

Implementering av nye taskar er enkelt og dersom ein behøver kommunikasjon eller synkronisering mellom taskar, er det veldefinerte grensesnitt i metodane å utføre dette på.

Mykje av programvaren på roboten bar preg av forskjellige måtar å løyse ting på, sidan alt er skreve av ein person no, er det lettare å sjå kontinuiteten i koden. Det er fokusert på å gjere koden intuitiv å forstå i, og kommentarar som refererer til rett plass i databladet.

Framfor infløkte strukturar med hexadecimal tal og variablar som er navngitt med berre ein bokstav, er det valgt variabel og funksjonsnamn som skal gi meining i forhold

til kva eg ønsker at dei skal gjere. Alle funksjonar er prefixa med returtypen og navnet på fila dei ligg i, til dømes heiter funksjonen for gyroskopet "fIMU_readFloatGyroZ". Globale variablar er holdt til eit minimum, dei har fått ein eigen prefix "g" og er kun brukt der det er fleire trådar som skal lese frå same variabel. I høve der ein tråd skal kommunisere med ein anna tråd er det brukt FIFO køar, som vert definert som ein eigen struktur i starten av programmet. Taskane i programmet følgjer same namnekonvensjonen som funksjonane, og alle heiter "vMain..." og er skreven i main.c fila. Dette er gjort for å ikkje abstraktere vekk essensielle delar av programvaren.

Prosessen med å skrive programmet har vore lærerik, og nokon av løysingane som vart brukt i starten av semesteret hadde eg løyst på ein anna måte med lærdomen eg har opparbeidd meg under arbeidet. Programmet fungerer likevel som ønska, og ein kan grave seg ned i detaljane her om ein vil, så det vart tatt ei avgjersla på at programmet er klart så fort det fungerer.

Det er brukt pre-emptive scheduling og prioriteringa av taskane er fastbestemte, estimatortasken har fått høgast prioritet, då gyromålingane bruker tida sidan sist avlesning. Det skal seiast at det ikkje er sikkert at forskjellen hadde vore målbar dersom estimator tasken hadde fått ei lågare prioritet, då dette ikkje vart sjekka.

Det er oppretta ei eiga fil for definisjonar som heiter `defines.h` der tanken er at ein enkelt kan gå til denne fila og omdefinere kva pinnar og register som vert brukt, slik at ein enkelt kan nytte programvaren på ein anna robot, med liknande arkitektur. Det vart likevel eit poeng at dersom ein har veldig mykje fleire eller færre portar på mikrokontrolleren, så er det fleire eller færre register som må endrast, og det må likevel gjerast små endringar i koden andre stader. Men i all hovudsak er det i definisjonsfila ein går først, for å så korrigere etter kvart dersom det ikkje stemmer.

Rørsla til sensortårnet går kontinuerleg, men for å kartlegge meir effektivt, endra eg inkrementeringa til tårnet i forhold til rørsla til roboten. Når den står i ro, flytter tårnet seg med små steg, slik at vi tette målingar. Når roboten kjører så flytter tårnet seg fortare, slik at vi får oversiktsmålingar. Når roboten roterer, står tårnet i ro, slik at vi ikkje gjer fleire målingar på samme plassen, eller måler med enorm forskjell. Dette viste seg å være veldig effektivt.

Det var problem at roboten vart ståande utan å utforske sjølv om den tilsynelatande skulle ha fått beskjed om å gjere dette. Det vart mistanke om at IDLE-meldinga frå roboten, som berre vart sendt ein gong ikkje kom fram. Det vart lagt inn at roboten skulle sende IDLE-melding med fast intervall når den sto i ro. Det vart i ettertid bekrefta at problemet skulda ein bug i server applikasjonen. Bluetooth protokollen garanterer at ein pakke på 23 byte vert levert dersom koblinga er gyldig (Bluetooth-spec 2010). Idle-meldinga når roboten står i ro vart beheldt, men denne bruker berre unødvendig resursar og kan trygt fjernast.

Utrekningane for å finne ut den teoretiske kapasiteten var noko omstendelige, og ein betre tilnærming hadde vore å brukt idle tasken til FreeRTOS, då denne kun vert kjørt når det er "ledig tid" for prosessoren. Det er då viktig at vi ser på "kort nok" tid, og ikkje berre over veldig lang tid, då vi kan få situasjonar der "alt må skje på ein gang" medan det ikkje skjer nokon ting før og etter.

16.5 Implementering på Arduino roboten

Programmet som vart laga til AVR roboten vart implementert på Arduino roboten. Tanken var at arkitekturen var lik nok til at det berre var å bruke same koden og endre registera. Eg oppdaga at arkitekturen ikkje var heilt lik, men etter å ha brukt ei anna portfil til FreeRTOS enn den eg laga i fjor, fungerte det. Bortsett frå dette og at Atmega 2560 har mykje meir register er mikrokontrollerane ganske like. Vi har brukt dei same peripherals på begge, og robotane oppfører seg likt.

Arduino roboten har ein anna motor enn AVR roboten, og denne har mykje høgare effekt. Dette gjorde det svært vanskelig å styre roboten. Vi prøvde å løyse dette ved å gire ned motorane slik at variasjonane gjer mindre utslag. Dette fungerte, men då vart tikkfrekvensen veldig høg, med opp til 25 tikk per 20 millisekund. På grunn av dette måtte ein alternativ tilnærming skrivast på interrupt behandlinga.

Det er kun ein 8bit variabel som vert inkrementert i interruptet, og den vert seinare behandla i tasken som tar seg av styringa til motorane. Denne kjører med 20 millisekund periodetid, og 8 bit variabelen vil ikkje rekke å bli stort større enn 25 før tasken lese av verdien, og nullstiller interrupt variabelen. Skulle det likevel gå lang tid, så kan det gå opptil 204 ms før variabelen overflower, og det tilfellet vil det være meir alvorlige problem som har hindra at motor tasken ikkje får kjørt på så lang tid, enn motorstyringa. Ei betre løysing for å behandle interrupta er truleg å bruke ein hardware counter, der vi kan sette overflow grensa til f.eks 10. Vi vil då få 125 interupt per sekund, som er passelig, utan å bruke noko prosessorkraft på dette. For å få dette til må interrupta flyttast til T0 og T5 pinnane på roboten.

Det kan være nyttig å ta med seg at det faktisk var ein forskjell på venstre motor sin enkoderverdi når den var tilkobla INT2 og INT3, vi ser at den når 1000 på 10 tidssteg / 200ms når den er tilkobla INT2 enn INT3. Det er ikkje sjekka om dette skuldast prioritetsforskjellen på interruptvektorane, eller om det var ein naturleg tilfeldighet. Når magnetsensoren er bytta bør det verifiserast om begge hjula genererer like mange tikk på ein omdreining. Dette vart sjekka før motorane vart gira ned, men ikkje etterpå, og det kan være at den venstre enkoderen ikkje fungerer skikkelig.

Ein avstandssensor og ein enkoder sensor vart øydelagt rett før prosjektslutt. Dersom desse hadde vore i orden, hadde roboten sannsynlegvis fungert bra.

16.6 Integrering med resten av gruppene

Integreringa med dei andre gruppene har gått bra då vi har hatt eit tett samarbeid og god kommunikasjon. Dersom ting har vore uklart har vi hatt ein låg terskel for å spørre kvarandre om hjelp og dette vi heva kvaliteten på kvarandre og prosjektet jamnt over.

Vi har ikkje hatt nokon store utfordringar med integreringa då det har vore klar semje rundt løysningane vi har brukt.

Arbeidet med implementeringa på Arduino roboten hadde ein del utfordringar, og det hadde nok vore lettare om same personen som bygde roboten også programmerte den. Det vart ein del debugging under programmeringa som ikkje var av programmeringsmessige årsaker. Dette likevel fort då eg allereie har feil-søkt mykje på AVR roboten og hadde ein viss kjensle for typiske feil.

16.7 Testinga

Testinga som vart utført i vår er gjort i mykje større areal enn det som har vore brukt tidlegare. I den gamle labyrinten på kontoret er det ca 1 kvadratmeter med areal som roboten kan kartlegge. Estimator-feilen vert veldig liten på eit så lite område. Med forbetringane kan den nye roboten kan kartlegge heile bana på eit minutt.

Testen som vart gjort på kontoret for å samanlikne ytelsen til roboten frå i fjor vart gjort med den nye navigasjonsalgoritmen. Roboten roterte mykje meir då enn no, og dette kan ha gjort at resultatet vart mykje bedre enn det hadde vore dersom robotane følgte den same bana. Likevel viste dei andre testane at estimering med berre enkoder ville gitt store avvik etter ei stund, og kompasset korrigerer for dette.

Under testinga etter vi fekk tilgang til motion tracker laben var det mykje enklare å verifisere og tune regulatorane og estimatoren, og eg burde hatt tilgang her med ein gang eg begynte på arbeidet med dette. Hadde eg komme i gang med dette tidlegare kunne eg brukt meir tid på å funne smarte løysingar.

Den manuelle testen med optitrack systemet vart noko svak, då roboten fekk alt for lang tid mellom kvar iterasjon.

Kapittel 17

Vidare arbeid

17.1 Vidare programutvikling generellt på robotane

- Det er ein liten offset mellom senterpunktet på roboten, akselavstanden og posisjonen på sensortårnet på robotane. Offsetten vert sendt i handshake meldinga til serveren, men det vert ikkje gjort noko med dette. Det går og an å legge inn offset verdien lokalt på roboten

- Implementasjon av meir avanserte reguleringsmetodar, spesielt MRAC (Ioannou & Sun 2012) som adaptivt justerer på regulatorparameter for å få systemet til å følgje ein bestemt modell. Dette er simulert i Canigur & Ozkan (2012), der dei konkluderer med at resultata viser ei forbetring i baneføling

- Ved meir implementering på robotane med FreeRTOS bør ein legge inn idle tasken, og bruke denne til å sjekke ulitilization for å sikre seg at ein ikkje bruker for mykje resurser. Det er viktig at ein tenker på race conditions som beskreven i Stallings (2015), då det kan være mindre kapasitet tilgjengelig enn først antatt dersom ein ser på for stort tidsrom

- Gjer robotane mindre avhengig av serveren og meir autonom ved implementere Line of Sight navigasjon med circle of acceptance, som skreven om i Fossen (2011)

- Ved implementering av navigasjon på roboten kan ein legge inn høgnivå anti-kollisjon og planlegging av nye baner lokalt på roboten

- Det ligg metodar inne i roboten for å sende posisjonen til objekt som er skanna, framfor å sende all måledata over den trådlause koblinga, desse kan brukast framfor å sende rå måledata

- Kalibrering for kvar enkelt sensor på robotane, sidan robotane klarer å regulere posisjonen med god nøyaktighet i linjer rørsle, går det an å lage eit automatisk kalibreringsskript der du starter roboten heilt inntill ein vegg, og den kjører og tek målingar, linjeriserer analogverdiane mot avstand, og lagrer dei kalibrerte verdiane i f.eks EEPROM, eller skriver dei ut til brukaren så han kan legge dei inn sjølv

- Det er mykje kapasitet til overs, dersom ikkje meir funksjonalitet skal implementerast kan ein sjå på filtrering av målingane frå IR sensorane

– Pose reglatoren vart først brukt utan referansemodell. Det vart då gjort ei forenkling, sidan settpunktet er konstant kan vi skrive $\dot{e} = -\dot{y}$. Etter referansemodellen vart implementert er settpunktet ikkje konstant lenger, men derivatleddet i regulatoren vart ikkje endra til $\dot{e} = \dot{r} - \dot{y}$

– I Fossen (2011) er det skreven om å bruke akselerometer som tilbakemelding i PID regulatoren. Sidan vi har dette tilgjengelig kan det forbetre styringa ytterligare

– Ei betre løysing kan være å bruke FreeRTOS `xTaskGetTickCount()` for å sjekke differansen sidan sist iterasjon og bruke dette til å finne vinkelhastigheita per tidssteg. Vi vil få med millisekunds nøyaktighet tida sidan sist gong gyroen vart avlesen. Dette bør vera godt nok for alle praktiske føremål. Dette gjer at estimator tasken ikkje er avhengig av å kjøre som høg prioritet

– Batterisimulatoren er ikkje implementert igjen, men etter å ha inspisert løysinga virker denne ikkje spesielt skjønnsmessig då den er avhengig av at batteriet er full lada i det roboten vert påskrudd, og roboten ikkje blir skrudd av igjen før batteriet er i ferd med å blir tomt. Ei enklare løysing er å måle spenninga på batteriet, og finne cutoff punktet, slik at ein kan gi advarsel om at batteriet bør ladast, litt før cutoff punktet. Ein må ha i bakhovudet at dette er ei forenkla løysing på eit komplekst problem, men sannsynlegvis ei god nok løysing

– Vi kunne brukt akselerometeret saman med enkoderane og på den måten kanskje fått betre estimat på translasjoniske rørsle. Dersom dette skulle vore jobba meir på, ville eg vurdert å montere to akselerometer, eit over kvart hjul og sjå om det vil fungere med ei slags antispinn løysing. Denne løysinga, i tillegg dersom vi kunne stolt meir på kompasset, kunne vi adaptivt finne ein konstant mellom kor korrekt gyroen er i forhold til enkoderane, og på den måten kunne sei noko om kor dårleg friksjonen er

– Kommunikasjonsdongelen er optimalisert for lite straumforbruk, men programmet på robotane er ikkje det. Det går an å implementere "sleep mode" i idle tasken til FreeRTOS for å minke straumforbruket ytterligare, då begge robotane har denne funksjonen i ATmega mikrokontrolleren

– Tidlegare vart det lagt inn "Back-tracking" som baserte seg på at instruksane til roboten vart lagra, sånn at roboten då kan returnere same veg den kom. Dette var då implementert i matlab på serveren, men det er ingen grunn til at det ikkje kan lagrast i ein struct lokalt på roboten. Då kan det nyttast viss roboten mister kontakt med serveren, eller begynner å gå tom for strøm i batteriet, og må returnere

17.2 Forslag til ny robot

- Lage ein robot som baserer seg på nRF52 mikrokontrolleren. Denne har cortex M4 prosessor med ein Floating Point Unit, som gjer den rask på flyttal-utrekningar
- Om vi bruker nRF52 dev-kitet så har vi også ein near field communication (NFC) antenne, og vi kan nytte nær-kommunikasjon med til dømes ladestasjonen eller andre robotar
- Abot robot-platformen vert solgt på Omega Verksted, og er godt egna dersom ein finn på noko lurt for å få tilbakemelding frå hjula
- I staden for 4 stk IR sensorar går det an å kjøpe 1 LIDAR-lite som er rett rundt hjørnet med versjon 3, til nesten same prisen

17.3 Spesifikt for AVR Roboten

- Fjørene som skal lade batteriet på roboten går ikkje via nokon brytar. Det er ein sikring på roboten men det er ikkje kontrollert kvar denne sikringen beskytter. Det bør monterast ein brytar mellom batteri og ladefjør, som på Arduino roboten
- Ved måling ser eg at MCU, sensorar og nRF51 dongelen trekker opptil 0.2A, sidan det står ein linjær regulator på, vil den gjere differansen frå batterispenninga til ønska spenning om til varme. Dette tilsvarer ein $P = U \cdot I = 7 \cdot 0.20 = 1,4W$ "varmeovn" inni roboten, som med ein bivirkning lager rett spenning til mikrokontrolleren. Ein switche regulator kan føre til mykje spart straum
- For den justerbare switch-regulatoren til motorane det går an å bytte ut potmeteret med eit som det går an å justere med mikrokontrolleren, t.d "DS1804-010". Ein kan då styre hastigheita til motorane, med litt triksing går det kanskje an å styre spenninga til motorane individuellt
- Forbetring av kretskrotet som styrer motorane, slik at vi får skikkelig hastighetsregulering individuelt på begge hjula, PD6 og PD7 har 8bit PWM frå timer 2. Sjå vedlegg F på side 106 for kretsskjema
- Dersom ikkje kretskortet skal endrast ytterligare, ligg det eit nytt kretskort klart til lodding. Skjermatikken er vedlagt
- Ved arbeid på kretskortet les "AVR042: AVR Hardware Design Considerations"

17.4 Spesifikt for Arduino Roboten

- Bytte ut den ødelagte sensoren for enkoderen
- Bytte ut den ødelagte IR sensoren
- Implementere ein MOSFET på avstandssensorane for å bruke mindre straum, som på AVR roboten
- Lage eit nytt kretskort for å programmere med JTAG framfor USB, og litt tettare plassering av kontaktane for IR sensorane, då ledningane vert øydelagt etter kvert
- Konstantane som vert sendt i handshaken er dei som gjeld for AVR roboten, Arduino roboten må målast opp slik at rett verdi blir sendt. Dei kan endrast i `defines.h` fila

17.5 NXT Roboten

- For meir sjå kapittel 7 i Ese (2015)
- Teste om RS485-UART tranciever kortet til nRF51 dongelen fungerer. Husk at dongelen får feil dersom det er signal på rx/tx linjene før den er klar for å ta i mot kommunikasjon. Ein veg om dette problemet er å deaktivere debug mode, den vil då restarte kvar gang den krasjer. Med peripheral-softwaren som er brukt i dag vil dongelen advertise med blinkande raudt lys når den er klar. Dersom dette aldri skjer går den sansynlegvis i ein evig krasj-restart-loop
- For å interface med sensorane vart det utvikla eit I^2C-ADC kort av bakken i 2008 Bakken (2008). Ut frå tidlegare rapportar har dette aldri vore testa på NXT roboten, men eit likt kort har vore i bruk på den store mobil-roboten
- Om I^2C-ADC kortet fungerer, kan vi enkelt koble til både IMU og kompass på same bussen, og NXTen vil med litt programmering være på nivå med dei andre robotane
- Motorstyringa på NXT roboten er sannsynlegvis årsaka til at den ikkje fungerte tilfredsstillande, denne bør skrivast om, og enkoderverdiane må ikkje nullstillast før ein er sikker på at roboten har slutta å bevege på seg
- For å programmere roboten bør nxtOSEK eller leJOS testast, det går an å bruke NXC med multi-tasking, men det er ikkje moglegheit for synkronisering ved bruk av tradisjonelle RTOS verktøy
- Motorane bør girast ned

- Sensor tårnet på NXT roboten har ingen måte å posisjonere seg på. Det vart implementert ein fiks hausten 2015 men sannsynlegvis vil tårnet vera utsett for drift, og ein kan ikkje stole på at det alltid vil returnere til 0 grader. EV3 har same problemstillinga, og det vart tenkt over mulige løysingar:
  - nRF dongelen kan styre ein servo med PWM, eller lese av ein rotajsonsenkoder som er festa til tårnet
  - ein ATTiny til ein liten servomodul som NXTen kan kommunisere med via I2C eller UART, dersom ein vil separere kommunikasjonen og skanninga

17.6 EV3 Roboten

- Bygge den opp igjen, i samme stil som NXT roboten er no
- Viss NXT roboten fungerer med I^2C utvidelseskortet kan dei fire mindstorms Sharp sensorane brukast på EV3 roboten i staden
- Kommunisere med serveren enten med nRF51 dongelen som vart kjøpt inn til EV3, eller ved hjelpa av ein vanleg USB BLE dongel. EV3 har ein USB port
- EV3 har allereie ein gyro, og enkoderane på legomotorane er bra, spesielt om ein gjer "flanke hacken". Den treng eit kompass for å korrigere headinga
- Finne ein anna måte å programmere den på enn ved bruk av matlab eller mindstorms programmet. Ev3dev eller lejos er to gratis alternativ. RobotC er eit anna som koster pengar. Eventuelt går det an å programmere direkte på ARM prosessoren, men omfanget rundt dette er ikkje kjent

Kjeldekoden til EV3 bricken er gjort tilgjengelig:
https://github.com/mindboards/ev3sources

17.7 Kommunikasjonen

- Finne ut kvifor seriellporten krasjer når robotane vert frakobla på "feil" måte
- Sentral dongelen er avgrensa til å bruke tre roboter samtidig, det kan være at ein tradisjonell usb dongel med BLE kan koble til fleire mot at ein multiplekser på datamaskinen i staden for på dongelen som vi gjer no, ulempa med dette er at det vert vanskeligare å debugge kommunikasjonen. Det vert nok mulig med fleire einingar etter kvart som Nordic videreutvikler bluetooth stacken

- BLE protokollen har sansynlegvis mulighet for synkronisering av meldingar som vi ikkje har utnytta, og vi kan minimere problemet med fragmenterte meldingar, og heller få beskjed om når det tek for lang tid mellom meldingane frå roboten
- I eit av eksempela til nRF chipen er det vist at det går an å bruke sendestryka til radiosendaren saman med signalstyrken når du tar i mot, og ein finner slik eit estimat av avstanden mellom to noder. Det kan brukast til å finne ei relativ posisjon mellom fleire robotar, og med smarte løysingar kan ein slik korrigere for posisjonsfeil
- Det går an å skrive om meldingsstrukturen og sende data som bytes i staden for å gjere dei om til string av characters, og slik få ned storleiken på dei frå 3-4 pakker til 1 pakke. Det vil bli litt værre å "lytte" på kommunikasjonen, men ein kan få raskare respons og truleg eit meir effektivt system
- Det bør sjekkast om det går an å bruke mesh topologien som vart utvikla av T. Snekvik i 2015 på NTNU. Det heiter "nRF OpenMesh", og er interessant fordi vi kan ha veldig mange robotar, og veldig lang rekkevidde

Since the Bcast-mesh doesn't have any addressing features, the mesh network may be extended to any number of devices (10, 100, 1000, 10000 devices, if you want), but keep in mind that the propagation time and synchronization problems this causes may become an issue in larger networks.
-Trond Einar Snekvik, http://devzone.nordicsemi.com

Snekvik har gjort kjeldekoden for mesh nettverk tilgjengelig:
https://github.com/NordicSemiconductor/nRF51-ble-bcast-mesh

17.8 Andre forslag

- Den store mobil-roboten kan lett inkluderast for å bli ein del av systemet, og kan f.eks bli "arbeidaren" medan dei andre robotane kan være "karleggarane"
- Utvide IR sensortårnet med ein ekstra servo, sånn at vi kan skanne i rommet
- Kamera eller laser for å måle underlaget roboten kjører på for å forbetre posisjonsestimat, som foreslått i Syvertsen (2005) i 2005. Det har komt mykje bra på kamerafronten dei siste 11 åra

Del V

Vedlegg

Tillegg

Beskriving av filvedlegg

A

1_Video

Arduino_Highspeed

Test av Arduino roboten med fullt pådrag før motorane vart gira ned. Roboten roterer først, og så skal den kjøre rett fram. Vi ser at roboten "drifter" i sirkler pga manglande friksjon

Arduino_tick_comparison

Test under utvikling av motorkontroller. Her blir motorane styrt ved å samanlikne tikk for kvart hjul. Videoen er kun med for å vise at det gjekk an å styre roboten før enkoderen vart ødelagt

AVRrobot_test_comparison

Samanlikning av test som vart utført før jul av Ese (2015). Vi ser at den nye roboten er ca 3 gonger raskare og meir nøyaktig.

AVRrobot_mapnav_optitrack

Samansveisa opptak frå kartleggingstest med samtidig tracking via optitrack systemet i slange-labben.

2_Kjelder

Skjermbilete frå dei viktigaste nettsidekjeldene som er referert til ligg her om dei skulle bli fjerna eller flytta

3_Tidlegare prosjektoppgåver

Alle rapporter som har vore tilknytt prosjektet er vedlagt.

4_Kjeldekode

Kjeldekode for AVR roboten, Arduino roboten, nRF51 dongelen for server og robot. Merk at Tor Onshus også har koden i eit bitbucket repository

5_Designfiler

Eagle kretsskjema og utlegg for Abot, AVR roboten, NXT Transceiveren, og footprintet til nRF51 dongelen, 1284p xplained, IMU og logikk konverteren.

6_Datablad

Datablad til ATmega1284p, IMU, kompass, ATmega2560, sensorar, mm

7_Testresultat

Plott og sånt

8_Papers and research

Artiklar o.l. som ikkje vart sitert i rapporten.

Tillegg

B Bruk av robotane

Initialisering

Når robotane skal brukast må dei stå i ro i det dei vert initalisert, altså før handshake meldinga vert bekrefta frå serveren.

Grunnen til dette er fordi at det er funksjoner som finner bias i gyro målingane, samt ein offset til kompass målingane, då roboten alltid starter med heading $\theta = 0$.

LED

Robotane er utstyrt med lysdioder for å vise kva status dei har. Kva diodane signaliserer vert forklart i kapittel 6 på side 28.

Kompass

Kompasset må rekalibrerast ved endringar på roboten. For å gjere dette må ein laste inn programmet på nytt og kommentere inn definisjonen "`COMPASS_CALIBRATE`". Om kompasset ikkje kalibrerast vil det visast ved at estimata på headinga til roboten vert veldig feil.

Når roboten starter vert ein eigen task initialisert som starter ved at brukaren sender {S,CON}. Den roterer 360 grader og skriver ut nye verdiar for x og y offset. Brukaren må notere seg verdiane og når han legger inn igjen "release" programmet må han oppdatere x og y offseten til kompasset i estimator tasken, og fjerne definisjonen "`COMPASS_CALIBRATE`" igjen, og laste opp programmet på nytt.

Tilkobling

Ved manuell styring av roboten via server dongelen er det viktig at brukaren følgjer "rett rekkefølgje" av kommandoar, ellers kan dongelen slutte å svare.

Sett i dongel - opne termite, og velg rett port. Aktiver DTS / DTR og verifiser at oppstartsmeldinga frå dongelen kjem. "`[9]:nrf:{reboot}`", eventuelt ein rekke meldingar om kva kommandoar som kan brukast dersom DEBUG er definert. Om desse ikkje kjem men du burde vera tilkobla dongelen, kan du skrive "reboot" og sjekke om den kjem igjen. Viss det ikkje kjem noko, så er det feil port, eller trøbbel med seriellkommunikasjonen frå pc, bytt COM og/eller USB port

Fjernstyring

Når du har kontakt med dongelen, slå på robot og verifiser at roboten advertiser (Grønn LED blinker, raud LED på dongelen blinker) kjør “scan” i terminalen, og så “list” og oppdaga bluetooth peripherals med navn og indeks nummer dukker opp. Skriv “conn 0” for å koble til indeks 0. Du er no i “kommando modus” og sender meldingar til roboten. For å gå ut av kommando modus skriv “*”

Det går an å koble til med nordic sin bluetooth app, og du kan også lese og sende meldingar til roboten med denne. Når du er tilkobla får du ein handshake etter 1-2 sekund, som starter med {H,....}. Du kan sende {S,CON} for å verifisere og roboten begynner å skanne og sende updates. Om du ikkje vil ha updates kan du sende {S,PAU}. Roboten kan styrast ved å sende {U,vinkel,distanse}. Dersom roboten er på kollisjonskurs kan du stoppe den ved å sende {U,0,0}. Meir om meldingane er skreven i delen om kommunikasjonen.

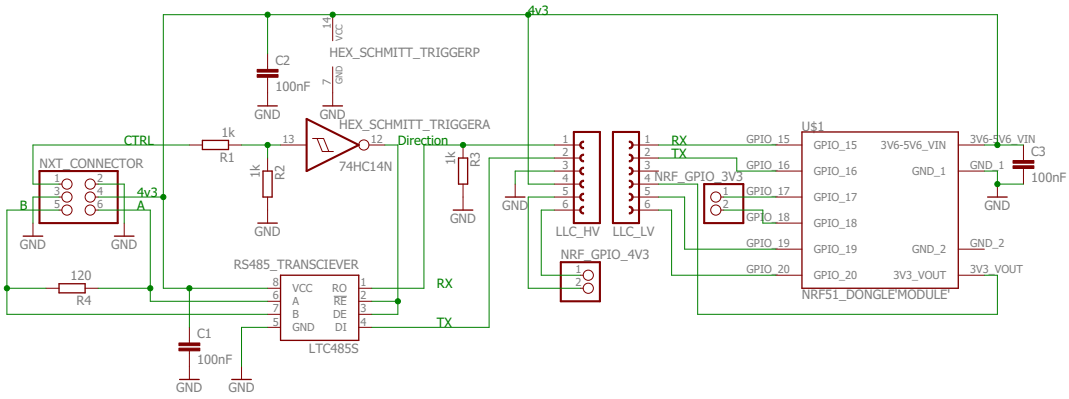
Fråkobling

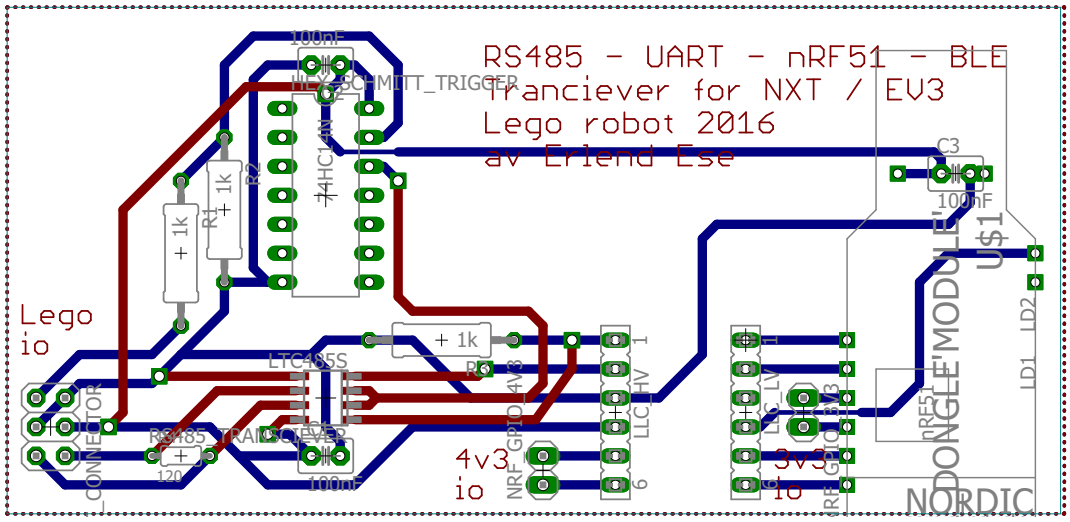
Når du er “ferdig” med roboten, er det viktig at du avslutter på rett måte, ellers kan server dongelen bli uresponsiv over seriellporten, og ikkje fungere før du tar den ut og inn igjen i USB porten. For å unngå dette, kjør “*drop 0” for å gå ut av kommando modus og koble frå roboten i indeks 0. Roboten vil respondere momentant og gå tilbake til advertising mode, servoen går til 0°. Om du berre slår av roboten utan å koble frå, eller berre resetter serverdongelen, vil ikkje dongelen på roboten oppdage det før sentralen timer ut. Når dette skjer hender det at sentralen blir uresponsiv.

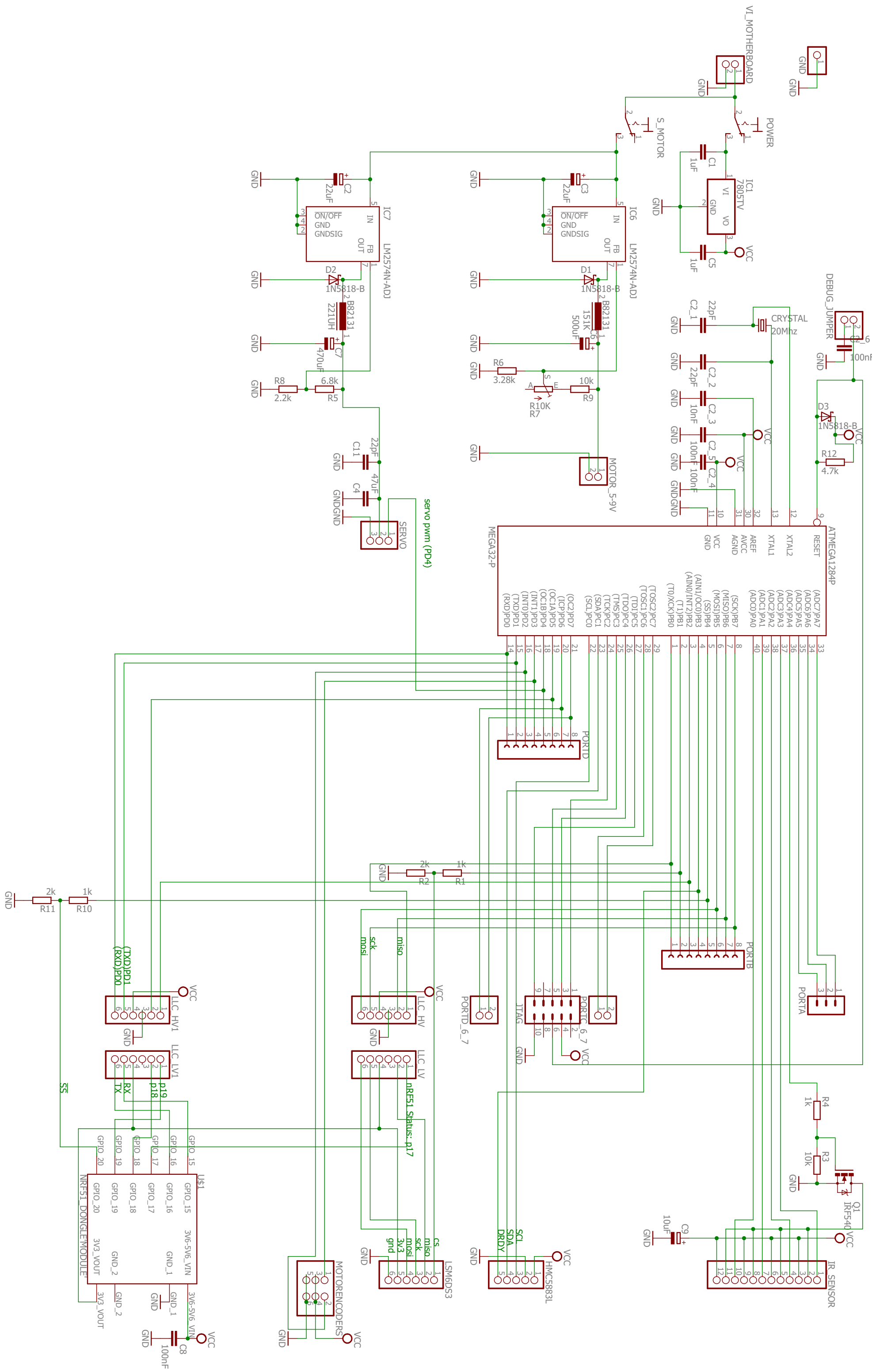
I java applikasjonen vert alt gjort automatisk i rett rekkjefølgje når du klikker på “x” for å lukke vinduet.

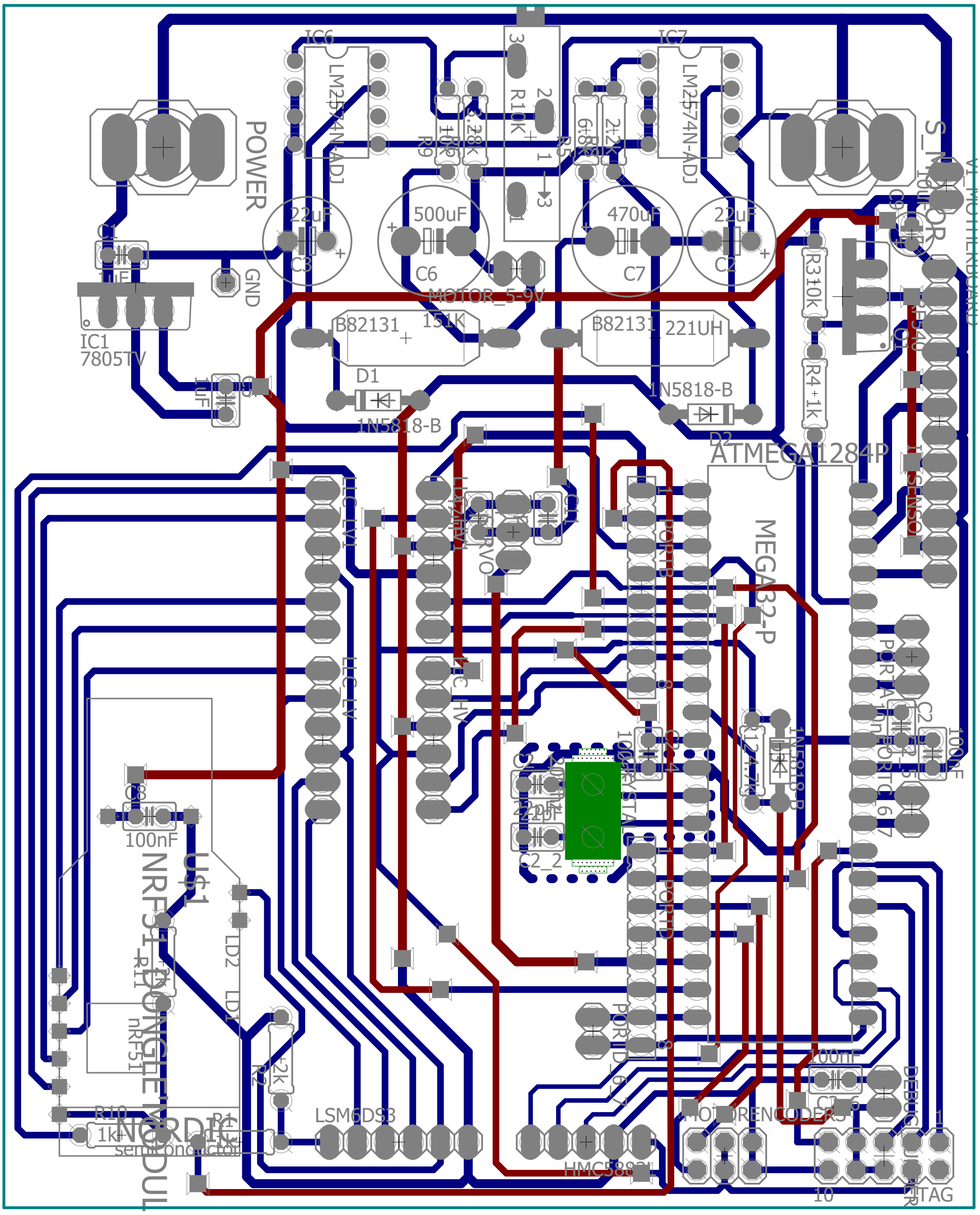
C Kretskort

Vedlagt er utskrift frå sch og brd filene frå eagle. Designfilene er vedlagt på DVD.






VI_MOTHERBOARD
POWER
S_MOTOR
IC1
7805TV
IC6
IC7
LM2574N-ADJ
1N5818-B
B82131
CRYSTAL
20Mhz
DEBUG_JUMPER
MOTOR_5-9V
SERVO
servo pwm (PD4)
MEGA32-P
ATMEGA1284P
PORTD
PORTB
PORTA
PORTC_6_7
PORTD_6_7
JTAG
LLC_HV1
LLC_LV1
LLC_HV
LLC_LV
nRF51_Status: p17
NRF51_DONGLE_MODULE
MOTORENCODERS
LSM6DS3
HMC5883L
IR_SENSOR
IRF540
VCC
GND


IC6
LM2574N-ADJ
IC7
LM2574N-ADJ
POWER
GND
IC1
7805TV
22uF
C3
500uF
C6
470uF
C7
22uF
C2
MOTOR_5-9V
B82131
151K
B82131
221UH
D1
1N5818-B
1N5818-B
D2
ATMEGA1284P
MEGA32-P
S_MOTOR
100nF
LSM6DS3
HMC5883L
MOTORENCODER
JTAG
NRF51_DONGLE
NORDIC
semiconductor
nRF51
LD2
LD1

Tillegg

D Pinouts

ATmega1284 pinout

nRF51 p17	1	PB0	U	(ADC0) PA0	40	distSensLeft
CS IMU	2	PB1		(ADC1) PA1	39	distSensRear
nRF51 p19	3	PB2 (INT2)		(ADC2) PA2	38	distSensRight
	4	PB3		(ADC3) PA3	37	distSensForward
nRF51 p20	5	PB4 (!SS)		(ADC4) PA4	36	distSensPower
SDA IMU	6	PB5 (MOSI)		(ADC5) PA5	35	LED Green
SD0 IMU	7	PB6 (MISO)		(ADC6) PA6	34	LED Yellow
SCL IMU	8	PB7 (SCK)		(ADC7) PA7	33	LED Red
	9	RESET		AREF	32	
	10	VCC		GND	31	
	11	GND		AVCC	30	
	12	XTAL1		PC7	29	motorRightBackward
	13	XTAL2		PC6	28	motorRightForward
nRF51 TxD p16	14	PD0 (RxD)		(TDI) PC5	27	JTAG
nRF51 RxD p15	15	PD1 (TxD)		(TDO) PC4	26	JTAG
encoderPinLeft	16	PD2 (INT0)		(TMS) PC3	25	JTAG
encoderPinRight	17	PD3 (INT1)		(TCK) PC2	24	JTAG
Servo	18	PD4 (OC1B)		(SDA) PC1	23	Kompass
nRF51 p18	19	PD5		(SCL) PC0	22	Kompass
motorLeftBackward	20	PD6		PD7	21	motorleftForward

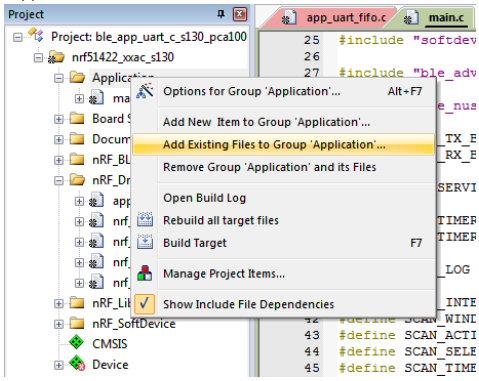
Arduino robot pinner

Høgre motor, framover	PINA2	PINA7	nRF51 p17
Høgre motor, bakover	PINA4	PINC6	nRF51 p18
Høgre motor, enable	PINB7	PINE4	nRF51 p19
		PINC2	nRF51 p20
Venstre motor, enable	PING5	PINH1	nRF51 p16
Venstre motor, framover	PINC7	PINH0	nRF51 p15
Venstre motor, bakover	PINC5		
		PINB0	IMU SS
Venstre enkoder	PIND3	PINB1	IMU SCK
Høgre enkoder	PIND2	PINB2	IMU MOSI
		PINB3	IMU MISO
Servo pin	PINB5		
		PINF0	Venstre IR sensor
Gul LED	PINL1	PINF1	Forvover IR sensor
Grønn LED	PINL2	PINF2	Bakover IR sensor
Raud LED	PINL3	PINF3	Høgre IR sensor
Kompass SCL	PD0	PD1	Kompass SDA

Importering av Keil uVision5 prosjekt

How I imported Marcos non-keil code into keil to be used on a nRF51 dongle (PCA10031)

1. Download and extract Marco's code files to some folder, i.e. next to the SDK folder.
2. Go to SDK/examples/ble_central/ and copy the folder "ble_app_uart_c" and name it something else, i.e. "ble_app_multilink_uart_c"
3. Copy all Marco's .c and .h files into your new folder "ble_app_multilink_uart_c" and replace main.c when prompted.
4. In your folder "ble_app_multilink_uart_c" go to pca10031/s130/arm5_no_packs and open ble_app_uart_c_s130_pca10031.uvprox, this should start keil.
5. Right click "Application" in the project window to left and choose "Add Existing Files to group 'Application'…"



6. Now go back to "ble_app_multilink_uart_c" in the add files window, and add "conn_manager.c" "uart_manager.c" and "util.c" and choose close. Compile and get 0 errors and 5 warnings and you should be all set.

To use uart to PC instead of the Arduino:

a. Open uart_manager.c in keil
b. On lines 58-62 change to following, this sets the pins connected to USB

```
/* UART TX, RX, CTS, RTS pin numbers */
#define UART_TX_PIN          9
#define UART_RX_PIN          11
#define UART_RTS_PIN         8
#define UART_CTS_PIN         10
```

And on line 110 change

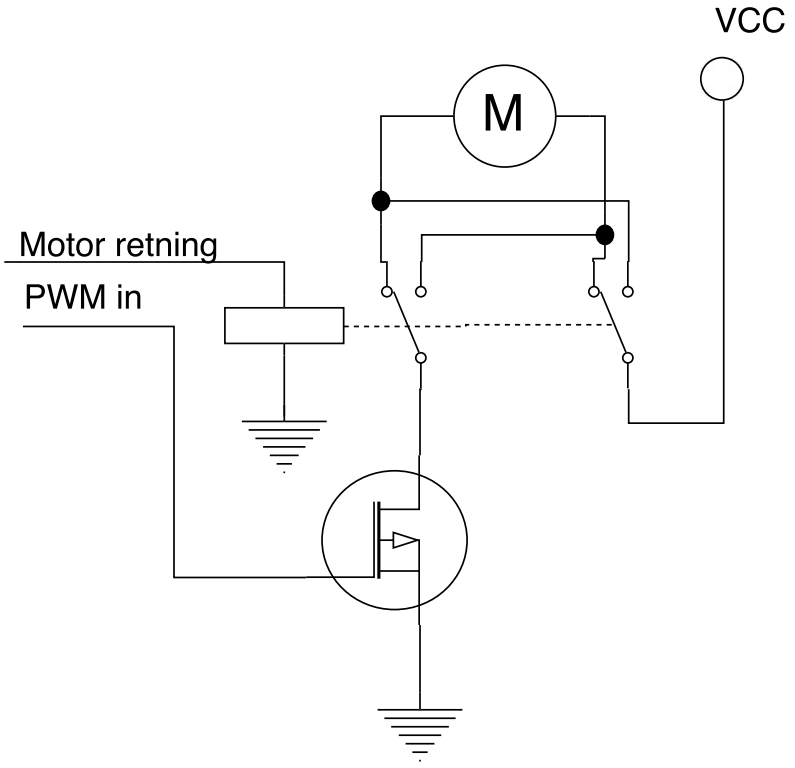
".flow_control = APP_UART_FLOW_CONTROL_DISABLED,"

To

".flow_control = APP_UART_FLOW_CONTROL_ENABLED,"

Load your softdevice (S130 in this case) recompile, load the program in keil, and you should now be set up to use termite with 38400 baud, and RTS/CTS enabled as in the original uart example. Use the commands explained in the git readme to connect to peripherals. I managed to connect to peripherals loaded with the "ble_app_uart" example, without any changes. **Remember to disable "append /lf" in your terminal program.**

Forslag til motorstyring



Tillegg G

Straummålingar

Målingane vart gjort ved å bruke ein laboratorie-forsyning, setje opp roboten i dei forskjellige scenarioa og lese av straumen som vart trekt.

AVR straummålingar

Tilstand	Forbruk	Kommentar
Venter	72mA / 90mA	avhengig av LED
Skanner, står i ro	90-160mA	avhengig av servoposisjon
Kjører uten å skanne	95mA	avhengig av hastigheit
Skanner og kjører	100-200mA	avhengig av servo og hastigheit

Arduino straummålingar

Tilstand	Forbruk	Kommentar
Venter	200mA	
Skanner, står i ro	200-300mA	avhengig av servoposisjon
Kjører uten å skanne	400-600mA	avhengig av hastigheit
Skanner og kjører	400-600mA	avhengig av servo og hastigheit

Tillegg

II Tidsbruk

I tillegg til prosjektstyring ved hjelp av MS project har det vore ført timar på enkeltoppgåver heile semesteret.

For framtidig referanse for studentar som planer å gjere liknande arbeid kan det vera greit å ha ei peikepinn på kor mykje tid ein kan forvente seg å bruke. Det kan være verdt å merke seg at timane som er opplista også dekker tida det tok å setje seg inn i aktuell teori og programvare for å utføre oppgåva. Om ein til dømes har laga kretskort tidlegare er det usansynleg at det tek like lang tid å lage eit tilsvarande kretskort.

Oppgåver	Timer
Dokumentering (rapport)	232
Estimator, teori, programmering, testing	164
Utvikling og testing av nytt RTOS program	119
Nytt kretskort AVR	107
Arduino robot implementering og testing	95
Kommunikasjonsmodul, teori, programmering, testing	93
Kontroller og motorstyring, teori, programmering, testing	67
Integrering og testing av det komplette systemet	39
NXT Kretskort for nRF51 tranciever, design og lodding	25
Møter med gruppene eller veiledar	20
Div. forbetring av AVR robot, sensortårn, ombygging osv	18
Sum	979

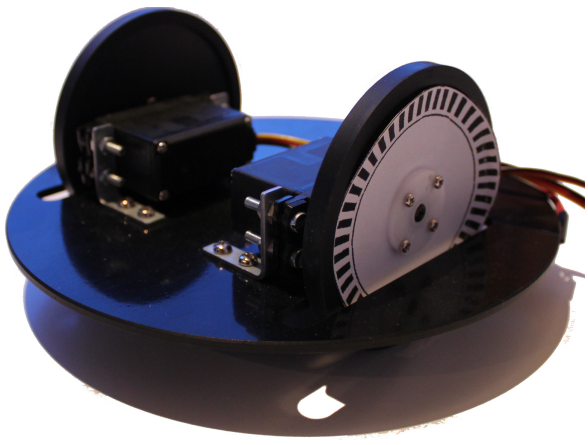
Tillegg I

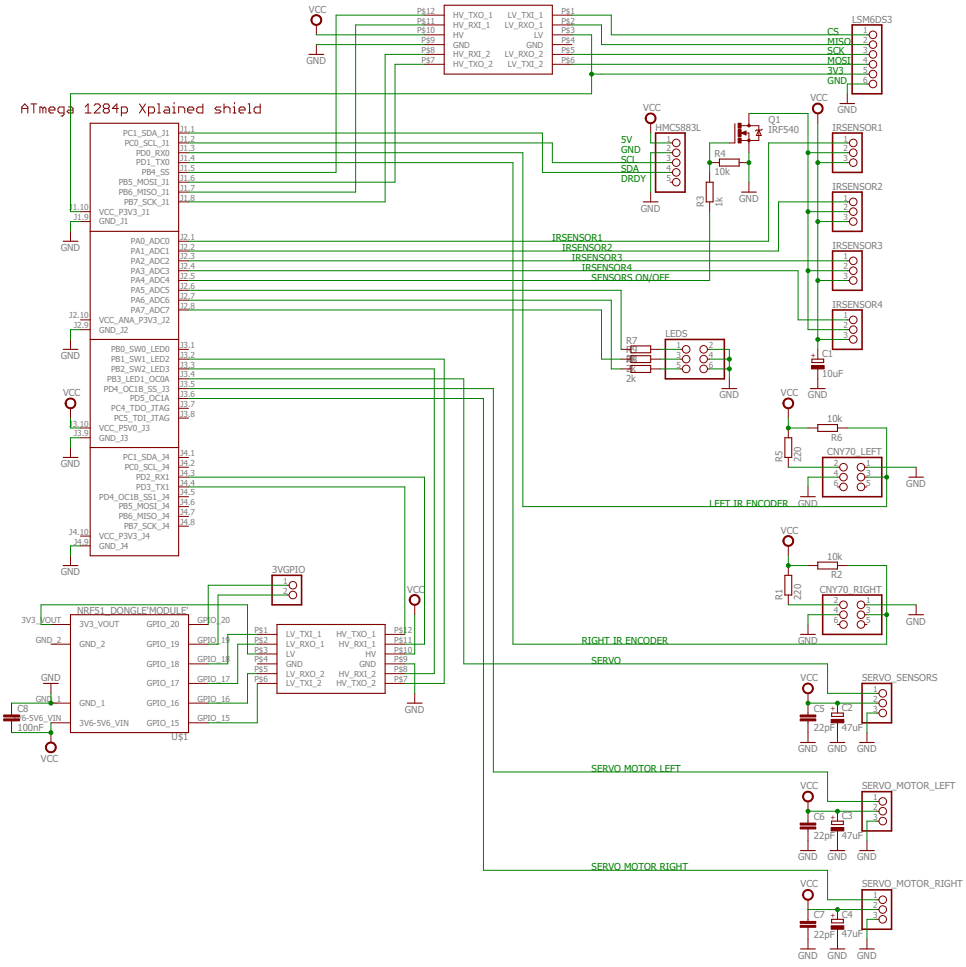
Mr. Abot

Det vart begynt å bygge ein robot frå komponent som var til salg på Omega Verksted. Denne vart ikkje ferdig, men komponentlista og skjematikken er tatt med dersom nokon ønsker å ta arbeidet opp igjen.

- Atmel Mr. Abot kit 300,-
- MEGA1284P Xplained 100,-
- Atmel Inertial One Sensor Board ATAVRSBIN1 100,-
- Sharp GP2Y0A21YK0F, 4 stk, 134,-
- RPR220 Reflective Opto Sensor Photoelectric Switch Sensor, 4 stk, 15,-
- CNY70 Reflective Optical Sensor with Transistor output, 4 stk, 12,-
- nRF51-DONGLE, 150,-
- SG90 9G Servo 15,-
- Sum kostnader: 826,-

Motorane til hjula er ein variant av ein servo, der pulsbredda til styresignalet styrer hastigheita og retningen. Det er ingen tilbakekopling på motoren. For å få tilbakemelding dette vart det klistra eit papir med svarte og kvite streker på hjula, som ein infraraud reflektiv optosensor kan lese.




ATmega 1284p Xplained shield
LSM6DS3
CS
MISO
SCK
MOSI
3V3
GND
HMC5883L
5V
GND
SCL
SDA
DRDY
Q1
IRF540
R4
10k
R3
1k
IRSENSOR1
IRSENSOR2
IRSENSOR3
IRSENSOR4
SENSORS ON/OFF
LEDS
R7
2k
C3
10uF
R6
10k
R5
220
CNY70_LEFT
LEFT IR ENCODER
R2
10k
R1
220
CNY70_RIGHT
RIGHT IR ENCODER
SERVO
SERVO_SENSORS
C5
22pF
C2
47uF
SERVO MOTOR LEFT
SERVO_MOTOR_LEFT
C6
22pF
C3
47uF
SERVO MOTOR RIGHT
SERVO_MOTOR_RIGHT
C7
22pF
C4
47uF
3VGPIO
NRF51_DONGLE'MODULE'
C8
100nF
U$1
VCC
GND

Tillegg J

Responstidsanalyse, utrekningar

AVR responstidsanalyse

Bokstaven til taskane er frå tabellane 9.2 og 9.3 på side 52.
AVR Task A:

$$D_A = T_A = 15$$
$$\omega_A^0 = C_A = 0.0017$$

$$\omega_A^1 = C_A = 0.0017, \qquad \omega_A^0 = \omega_A^1$$

$$\Longrightarrow R_A = 0.0017, D_A = 15, \qquad R_A \leq D_A \Longrightarrow \underline{\underline{OK}}$$

AVR Task B:

$$D_B = T_B = 15$$
$$\omega_B^0 = C_B = 0.0017$$

$$\omega_B^1 = C_B + \left\lceil \frac{\omega_B^0}{T_A} \right\rceil C_A = 0.0017 + 0.0017 = 0.0034, \qquad \omega_B^1 \neq \omega_B^0$$

$$\omega_B^2 = C_B + \left\lceil \frac{\omega_B^1}{T_A} \right\rceil C_A = 0.0017 + 0.0017 = 0.0034, \qquad \omega_B^2 = \omega_B^1$$

$$\Longrightarrow R_B = 0.0034, D_B = 15, \qquad R_B \leq D_B \Longrightarrow \underline{\underline{OK}}$$

AVR Task C:

$$D_C = T_C = 103$$
$$\omega_C^0 = C_C = 0.00296$$

$$\omega_C^1 = C_C + \left\lceil \frac{\omega_C^0}{T_B} \right\rceil^{1} C_B + \left\lceil \frac{\omega_C^0}{T_A} \right\rceil^{1} C_A$$
$$= 0.00296 + 0.0017 + 0.0017 = 0.0064 \qquad \omega_C^1 \neq \omega_C^0$$
$$\omega_C^2 = C_C + \left\lceil \frac{\omega_C^1}{T_B} \right\rceil^{1} C_B + \left\lceil \frac{\omega_C^1}{T_A} \right\rceil^{1} C_A$$
$$= 0.00296 + 0.0017 + 0.0017 = 0.0064, \qquad \omega_C^2 = \omega_C^1$$

$$\implies R_C = 0.0064, D_C = 103, \qquad R_C \leq D_C \implies \underline{\underline{OK}}$$

AVR Task D:

$$D_D = T_D = 30$$
$$\omega_D^0 = C_D = 1.70$$

$$\omega_D^1 = C_D + \left\lceil \frac{\omega_D^0}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_D^0}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_D^0}{T_A} \right\rceil^{2} C_A$$
$$= 1.70 + 0.00296 + 0.0034 + 0.0034 = 1.7098, \qquad \omega_D^1 \neq \omega_D^0$$
$$\omega_D^1 = C_D + \left\lceil \frac{\omega_D^1}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_D^1}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_D^1}{T_A} \right\rceil^{2} C_A$$
$$= 1.70 + 0.00296 + 0.0034 + 0.0034 = 1.7098, \qquad \omega_D^2 = \omega_D^1$$

$$\implies R_D = 1.7098, D_D = 30, \qquad R_D \leq D_D \implies \underline{\underline{OK}}$$

AVR Task E:

$D_E = T_E = 20$

$\omega_E^0 = C_E = 0.043$

$$\omega_E^1 = C_E + \left\lceil \frac{\omega_E^0}{T_D} \right\rceil^1 C_D + \left\lceil \frac{\omega_E^0}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_E^0}{T_B} \right\rceil^1 C_B + \left\lceil \frac{\omega_E^0}{T_A} \right\rceil^1 C_A$$
$$= 0.043 + 1.70 + 0.00296 + 0.0017 + 0.0017 = 1.7494, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_E^2 = C_E + \left\lceil \frac{\omega_E^1}{T_D} \right\rceil^1 C_D + \left\lceil \frac{\omega_E^1}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_E^1}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_E^1}{T_A} \right\rceil^2 C_A$$
$$= 0.043 + 1.70 + 0.00296 + 0.0034 + 0.0034 = 1.7528, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_E^3 = C_E + \left\lceil \frac{\omega_E^2}{T_D} \right\rceil^1 C_D + \left\lceil \frac{\omega_E^2}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_E^2}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_E^2}{T_A} \right\rceil^2 C_A$$
$$= 0.043 + 1.70 + 0.00296 + 0.0034 + 0.0034 = 1.7528, \qquad \omega_D^3 = \omega_D^2$$
$$\implies R_E = 1.7528, D_E = 20, \qquad R_E \leq D_E \implies \underline{\underline{OK}}$$

AVR Task F:

$D_F = T_F = 103$

$\omega_F^0 = C_F = 0.14$

$$\omega_F^1 = C_F + \left\lceil \frac{\omega_F^0}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_F^0}{T_D} \right\rceil^1 C_D + \left\lceil \frac{\omega_F^0}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_F^0}{T_B} \right\rceil^1 C_B + \left\lceil \frac{\omega_F^0}{T_A} \right\rceil^1 C_A$$
$$= 0.14 + 0.043 + 1.70 + 0.00296 + 0.0017 + 0.0017 = 1.8894, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_F^2 = C_F + \left\lceil \frac{\omega_F^1}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_F^1}{T_D} \right\rceil^1 C_D + \left\lceil \frac{\omega_F^1}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_F^1}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_F^1}{T_A} \right\rceil^2 C_A$$
$$= 0.14 + 0.043 + 1.70 + 0.00296 + 0.0034 + 0.0034 = 1.8928, \qquad \omega_D^2 \neq \omega_D^1$$

$$\omega_F^3 = C_F + \left\lceil \frac{\omega_F^2}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_F^2}{T_D} \right\rceil^1 C_D + \left\lceil \frac{\omega_F^2}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_F^2}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_F^2}{T_A} \right\rceil^2 C_A$$
$$= 0.14 + 0.043 + 1.70 + 0.00296 + 0.0034 + 0.0034 = 1.8928, \qquad \omega_D^3 = \omega_D^2$$
$$\implies R_F = 1.8928, D_F = 103, \qquad R_F \leq D_F \implies \underline{\underline{OK}}$$

AVR Task G:

$D_G = T_G = 30$
$\omega_G^0 = C_G = 0.255$

$$\omega_G^1 = C_G + \left\lceil \frac{\omega_G^0}{T_F} \right\rceil^1 C_F + \left\lceil \frac{\omega_G^0}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_G^0}{T_D} \right\rceil^1 C_D + ...$$
$$\left\lceil \frac{\omega_G^0}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_G^0}{T_B} \right\rceil^1 C_B + \left\lceil \frac{\omega_G^0}{T_A} \right\rceil^1 C_A$$
$$= 0.255 + 0.14 + 0.043 + 1.70 + 0.00296 + 0.0017 + 0.0017 = 2.1444, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_G^2 = C_G + \left\lceil \frac{\omega_G^1}{T_F} \right\rceil^1 C_F + \left\lceil \frac{\omega_G^1}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_G^1}{T_D} \right\rceil^1 C_D + ...$$
$$\left\lceil \frac{\omega_G^1}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_G^1}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_G^1}{T_A} \right\rceil^2 C_A$$
$$= 0.255 + 0.14 + 0.043 + 1.70 + 0.00296 + 0.0034 + 0.0034 = 2.1478, \qquad \omega_D^2 \neq \omega_D^1$$

$$\omega_G^3 = C_G + \left\lceil \frac{\omega_G^2}{T_F} \right\rceil^1 C_F + \left\lceil \frac{\omega_G^2}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_G^2}{T_D} \right\rceil^1 C_D + ...$$
$$\left\lceil \frac{\omega_G^2}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_G^2}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_G^2}{T_A} \right\rceil^2 C_A$$
$$= 0.255 + 0.14 + 0.043 + 1.70 + 0.00296 + 0.0034 + 0.0034 = 2.1478, \qquad \omega_D^3 = \omega_D^2$$

$$\implies R_G = 2.1478, D_G = 30, \qquad R_G \leq D_G \implies \underline{\underline{OK}}$$

AVR Task H:

$D_H = T_H = 200$
$\omega_H^0 = C_H = 7.30$

$$\omega_H^1 = C_H + \left\lceil \frac{\omega_H^0}{T_G} \right\rceil^{1} C_G + \left\lceil \frac{\omega_H^0}{T_F} \right\rceil^{1} C_F + \left\lceil \frac{\omega_H^0}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_H^0}{T_D} \right\rceil^{1} C_D + ...$$
$$\left\lceil \frac{\omega_H^0}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_H^0}{T_B} \right\rceil^{9} C_B + \left\lceil \frac{\omega_H^0}{T_A} \right\rceil^{9} C_A$$
$$= 7.30 + 0.255 + 0.14 + 0.043 + 1.70 + 0.00296 + 0.0153 + 0.0153 = 9.4716, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_H^2 = C_H + \left\lceil \frac{\omega_H^1}{T_G} \right\rceil^{1} C_G + \left\lceil \frac{\omega_H^1}{T_F} \right\rceil^{1} C_F + \left\lceil \frac{\omega_H^1}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_H^1}{T_D} \right\rceil^{1} C_D + ...$$
$$\left\lceil \frac{\omega_H^1}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_H^1}{T_B} \right\rceil^{11} C_B + \left\lceil \frac{\omega_H^1}{T_A} \right\rceil^{11} C_A$$
$$= 7.30 + 0.255 + 0.14 + 0.043 + 1.70 + 0.00296 + 0.0187 + 0.0187 = 9.4784, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_H^3 = C_H + \left\lceil \frac{\omega_H^2}{T_G} \right\rceil^{1} C_G + \left\lceil \frac{\omega_H^2}{T_F} \right\rceil^{1} C_F + \left\lceil \frac{\omega_H^2}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_H^2}{T_D} \right\rceil^{1} C_D + ...$$
$$\left\lceil \frac{\omega_H^2}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_H^2}{T_B} \right\rceil^{11} C_B + \left\lceil \frac{\omega_H^2}{T_A} \right\rceil^{11} C_A$$
$$= 7.30 + 0.255 + 0.14 + 0.043 + 1.70 + 0.00296 + 0.0187 + 0.0187 = 9.4784, \qquad \omega_D^3 = \omega_D^2$$

$$\implies R_H = 9.4784, D_H = 200, \qquad R_H \leq D_H \implies \underline{\underline{OK}}$$

Arduino responstidsanalyse

Bokstaven til taskane er frå tabellane 9.2 og 9.3 på side 52.
Arduino Task A:

$$D_A = T_A = 0.8$$
$$\omega_A^0 = C_A = 0.00061$$

$$\omega_A^1 = C_A = 0.00061, \qquad \omega_A^0 = \omega_A^1$$

$$\implies R_A = 0.00061, D_A = 0.8, \qquad R_A \leq D_A \implies \underline{\underline{OK}}$$

Arduino Task B:

$$D_B = T_B = 0.8$$
$$\omega_B^0 = C_B = 0.00061$$

$$\omega_B^1 = C_B + \left\lceil \frac{\omega_B^0}{T_A} \right\rceil \qquad C_A = 0.00061 + 0.00061 = 0.00122, \qquad \omega_B^1 \neq \omega_B^0$$

$$\omega_B^2 = C_B + \left\lceil \frac{\omega_B^1}{T_A} \right\rceil \qquad C_A = 0.00061 + 0.00061 = 0.00122, \qquad \omega_B^2 = \omega_B^1$$

$$\implies R_B = 0.00122, D_B = 0.8, \qquad R_B \leq D_B \implies \underline{\underline{OK}}$$

Arduino Task C:

$$D_C = T_C = 103$$
$$\omega_C^0 = C_C = 0.0034$$

$$\omega_C^1 = C_C + \left\lceil \frac{\omega_C^0}{T_B} \right\rceil^{1} C_B + \left\lceil \frac{\omega_C^0}{T_A} \right\rceil^{1} C_A$$
$$= 0.0034 + 0.00061 + 0.00061 = 0.00462 \qquad \omega_C^1 \neq \omega_C^0$$
$$\omega_C^2 = C_C + \left\lceil \frac{\omega_C^1}{T_B} \right\rceil^{1} C_B + \left\lceil \frac{\omega_C^1}{T_A} \right\rceil^{1} C_A$$
$$= 0.0034 + 0.00061 + 0.00061 = 0.00462, \qquad \omega_C^2 = \omega_C^1$$

$$\implies R_C = 0.00462, D_C = 103, \qquad R_C \leq D_C \implies \underline{\underline{OK}}$$

Arduino Task D:

$$D_D = T_D = 30$$
$$\omega_D^0 = C_D = 1.66$$

$$\omega_D^1 = C_D + \left\lceil \frac{\omega_D^0}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_D^0}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_D^0}{T_A} \right\rceil^{2} C_A$$
$$= 1.66 + 0.0034 + 0.00122 + 0.00122 = 1.66584, \qquad \omega_D^1 \neq \omega_D^0$$
$$\omega_D^1 = C_D + \left\lceil \frac{\omega_D^1}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_D^1}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_D^1}{T_A} \right\rceil^{2} C_A$$
$$= 1.66 + 0.0034 + 0.00122 + 0.00122 = 1.66584, \qquad \omega_D^2 = \omega_D^1$$

$$\implies R_D = 1.66584, D_D = 30, \qquad R_D \leq D_D \implies \underline{\underline{OK}}$$

Arduino Task E:

$D_E = T_E = 20$

$\omega_E^0 = C_E = 0.041$

$$\omega_E^1 = C_E + \left\lceil \frac{\omega_E^0}{T_D} \right\rceil^{1} C_D + \left\lceil \frac{\omega_E^0}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_E^0}{T_B} \right\rceil^{1} C_B + \left\lceil \frac{\omega_E^0}{T_A} \right\rceil^{1} C_A$$
$$= 0.041 + 1.66 + 0.0034 + 0.00061 + 0.00061 = 1.70562, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_E^2 = C_E + \left\lceil \frac{\omega_E^1}{T_D} \right\rceil^{1} C_D + \left\lceil \frac{\omega_E^1}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_E^1}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_E^1}{T_A} \right\rceil^{2} C_A$$
$$= 0.041 + 1.66 + 0.0034 + 0.00122 + 0.00122 = 1.70684, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_E^3 = C_E + \left\lceil \frac{\omega_E^2}{T_D} \right\rceil^{1} C_D + \left\lceil \frac{\omega_E^2}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_E^2}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_E^2}{T_A} \right\rceil^{2} C_A$$
$$= 0.041 + 1.66 + 0.0034 + 0.00122 + 0.00122 = 1.70684, \qquad \omega_D^3 = \omega_D^2$$
$$\implies R_E = 1.70684, D_E = 20, \qquad R_E \leq D_E \implies \underline{\underline{OK}}$$

Arduino Task F:

$D_F = T_F = 103$

$\omega_F^0 = C_F = 0.13$

$$\omega_F^1 = C_F + \left\lceil \frac{\omega_F^0}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_F^0}{T_D} \right\rceil^{1} C_D + \left\lceil \frac{\omega_F^0}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_F^0}{T_B} \right\rceil^{1} C_B + \left\lceil \frac{\omega_F^0}{T_A} \right\rceil^{1} C_A$$
$$= 0.13 + 0.041 + 1.66 + 0.0034 + 0.00061 + 0.00061 = 1.8356, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_F^2 = C_F + \left\lceil \frac{\omega_F^1}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_F^1}{T_D} \right\rceil^{1} C_D + \left\lceil \frac{\omega_F^1}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_F^1}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_F^1}{T_A} \right\rceil^{2} C_A$$
$$= 0.13 + 0.041 + 1.66 + 0.0034 + 0.00122 + 0.00122 = 1.8368, \qquad \omega_D^2 \neq \omega_D^1$$

$$\omega_F^3 = C_F + \left\lceil \frac{\omega_F^2}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_F^2}{T_D} \right\rceil^{1} C_D + \left\lceil \frac{\omega_F^2}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_F^2}{T_B} \right\rceil^{2} C_B + \left\lceil \frac{\omega_F^2}{T_A} \right\rceil^{2} C_A$$
$$= 0.13 + 0.041 + 1.66 + 0.0034 + 0.00122 + 0.00122 = 1.8368, \qquad \omega_D^3 = \omega_D^2$$
$$\implies R_F = 1.8368, D_F = 103, \qquad R_F \leq D_F \implies \underline{\underline{OK}}$$

Arduino Task G:

$D_G = T_G = 30$
$\omega_G^0 = C_G = 0.187$

$$\omega_G^1 = C_G + \left\lceil \frac{\omega_G^0}{T_F} \right\rceil^1 C_F + \left\lceil \frac{\omega_G^0}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_G^0}{T_D} \right\rceil^1 C_D + ...$$
$$\left\lceil \frac{\omega_G^0}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_G^0}{T_B} \right\rceil^1 C_B + \left\lceil \frac{\omega_G^0}{T_A} \right\rceil^1 C_A$$
$$= 0.187 + 0.13 + 0.041 + 1.66 + 0.0034 + 0.00061 + 0.00061 = 2.0226, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_G^2 = C_G + \left\lceil \frac{\omega_G^1}{T_F} \right\rceil^1 C_F + \left\lceil \frac{\omega_G^1}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_G^1}{T_D} \right\rceil^1 C_D + ...$$
$$\left\lceil \frac{\omega_G^1}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_G^1}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_G^1}{T_A} \right\rceil^2 C_A$$
$$= 0.187 + 0.13 + 0.041 + 1.66 + 0.0034 + 0.00122 + 0.00122 = 2.0238, \qquad \omega_D^2 \neq \omega_D^1$$

$$\omega_G^3 = C_G + \left\lceil \frac{\omega_G^2}{T_F} \right\rceil^1 C_F + \left\lceil \frac{\omega_G^2}{T_E} \right\rceil^1 C_E + \left\lceil \frac{\omega_G^2}{T_D} \right\rceil^1 C_D + ...$$
$$\left\lceil \frac{\omega_G^2}{T_C} \right\rceil^1 C_C + \left\lceil \frac{\omega_G^2}{T_B} \right\rceil^2 C_B + \left\lceil \frac{\omega_G^2}{T_A} \right\rceil^2 C_A$$
$$= 0.187 + 0.13 + 0.041 + 1.66 + 0.0034 + 0.00122 + 0.00122 = 2.0238, \qquad \omega_D^3 = \omega_D^2$$

$$\implies R_G = 2.0238, D_G = 30, \qquad R_G \leq D_G \implies \underline{\underline{OK}}$$

Arduino Task H:

$D_H = T_H = 200$
$\omega_H^0 = C_H = 5.88$

$$\omega_H^1 = C_H + \left\lceil \frac{\omega_H^0}{T_G} \right\rceil^{1} C_G + \left\lceil \frac{\omega_H^0}{T_F} \right\rceil^{1} C_F + \left\lceil \frac{\omega_H^0}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_H^0}{T_D} \right\rceil^{1} C_D + ...$$
$$\left\lceil \frac{\omega_H^0}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_H^0}{T_B} \right\rceil^{7} C_B + \left\lceil \frac{\omega_H^0}{T_A} \right\rceil^{7} C_A$$
$$= 5.88 + 0.187 + 0.13 + 0.041 + 1.66 + 0.0034 + 0.0043 + 0.0043 = 7.91, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_H^2 = C_H + \left\lceil \frac{\omega_H^1}{T_G} \right\rceil^{1} C_G + \left\lceil \frac{\omega_H^1}{T_F} \right\rceil^{1} C_F + \left\lceil \frac{\omega_H^1}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_H^1}{T_D} \right\rceil^{1} C_D + ...$$
$$\left\lceil \frac{\omega_H^1}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_H^1}{T_B} \right\rceil^{9} C_B + \left\lceil \frac{\omega_H^1}{T_A} \right\rceil^{9} C_A$$
$$= 5.88 + 0.187 + 0.13 + 0.041 + 1.66 + 0.0034 + 0.0055 + 0.0055 = 7.9124, \qquad \omega_D^1 \neq \omega_D^0$$

$$\omega_H^3 = C_H + \left\lceil \frac{\omega_H^2}{T_G} \right\rceil^{1} C_G + \left\lceil \frac{\omega_H^2}{T_F} \right\rceil^{1} C_F + \left\lceil \frac{\omega_H^2}{T_E} \right\rceil^{1} C_E + \left\lceil \frac{\omega_H^2}{T_D} \right\rceil^{1} C_D + ...$$
$$\left\lceil \frac{\omega_H^2}{T_C} \right\rceil^{1} C_C + \left\lceil \frac{\omega_H^2}{T_B} \right\rceil^{9} C_B + \left\lceil \frac{\omega_H^2}{T_A} \right\rceil^{9} C_A$$
$$= 5.88 + 0.187 + 0.13 + 0.041 + 1.66 + 0.0034 + 0.0055 + 0.0055 = 7.9124, \qquad \omega_D^3 = \omega_D^2$$

$$\implies R_H = 7.9124, D_H = 200, \qquad R_H \leq D_H \implies \underline{\underline{OK}}$$

Litteratur

Bakken (2008), *Prosjekt: Bygge og programmere ny Legorobot*, NTNU.

Barry, R. (2009), *Using the FreeRTOS Real Time Kernel - A Practical guide*, Vol. 1.0.5, FreeRTOS.

Bluetooth-blogg (2016), *Mesh and Bluetooth*, Bluetooth SIG Inc.

Bluetooth-spec (2010), *Bluetooth Spec. V.4.0*, Bluetooth SIG Inc.

Borestein, J., Everett, H. & L.Feng (1996), *Where am I? Sensors and Methods for Mobile Robot Positioning*, University of Michigan.

Brown, R. G. & Hwang, P. Y. (2012), *Introduction to Random Signals and Applied Kalman Filtering, fourth Edition*, Wiley.

Burns, A. & Wellings, A. (2006), *Real Time Systems and Programming Languages*, Addison Wesley.

Canigur, E. & Ozkan, M. (2012), *Model reference adaptive control of a nonholonomic wheeled mobile robot for trajectory tracking*, IEEE.

Carona, R., Aguiar, A. P. & Gaspar, J. (2008), *CONTROL OF UNICYCLE TYPE ROBOTS, Tracking, Path Following and Point Stabilization*, Universidade Técnica de Lisboa, Portugal.

Ese, E. (2015), *Prosjekt: Fjernstyring av Legorobot*, NTNU.

Fossen, T. I. (2011), *Marine craft hydrodynamics and motion control*, Wiley.

Halvorsen (2013), *Prosjekt: Collaborating robots*, NTNU.

Halvorsen (2014), *Master: Collaborating robots*, NTNU.

Homestad (2013), *Master: Fjernstyring av legoroboter*, NTNU.

Ioannou, P. A. & Sun, J. (2012), *Robust adaptive control*, Dover.

Naess (2008), *Prosjekt: LEGOROBOT - forbedring av eksisterende system*, NTNU.

Rødseth, M. G. & Andersen, T. E. S. (2016), *Master: System for self navigating autonomous robots*, NTNU.

Shalam, U. & Rodriguez, A. (2011), *Indoor Positioning using Sensor-fusion in Android Devices*, Kristianstad University.

Skjelten (2004), *Prosjekt: Fjernnavigasjon av LEGO-robot*, NTNU.

Spong, M. W., Hutchinson, S. & Vidyasagar, M. (2006), *Robot modeling and control*, Wiley.

Stallings, W. (2015), *Operating Systems, Internals and Design principles, eigth Edition*, Pearson.

Steuper (2015), *Prosjekt: LEGO Mindstorms EV3 robot*, NTNU.

Syvertsen (2005), *Prosjekt: Fjernstyring av legorobot*, NTNU.

Syvertsen (2006), *Master: Fjernstyring av legorobot*, NTNU.

Thon, E. (2016), *Master: Mapping og Navigasjon*, NTNU.

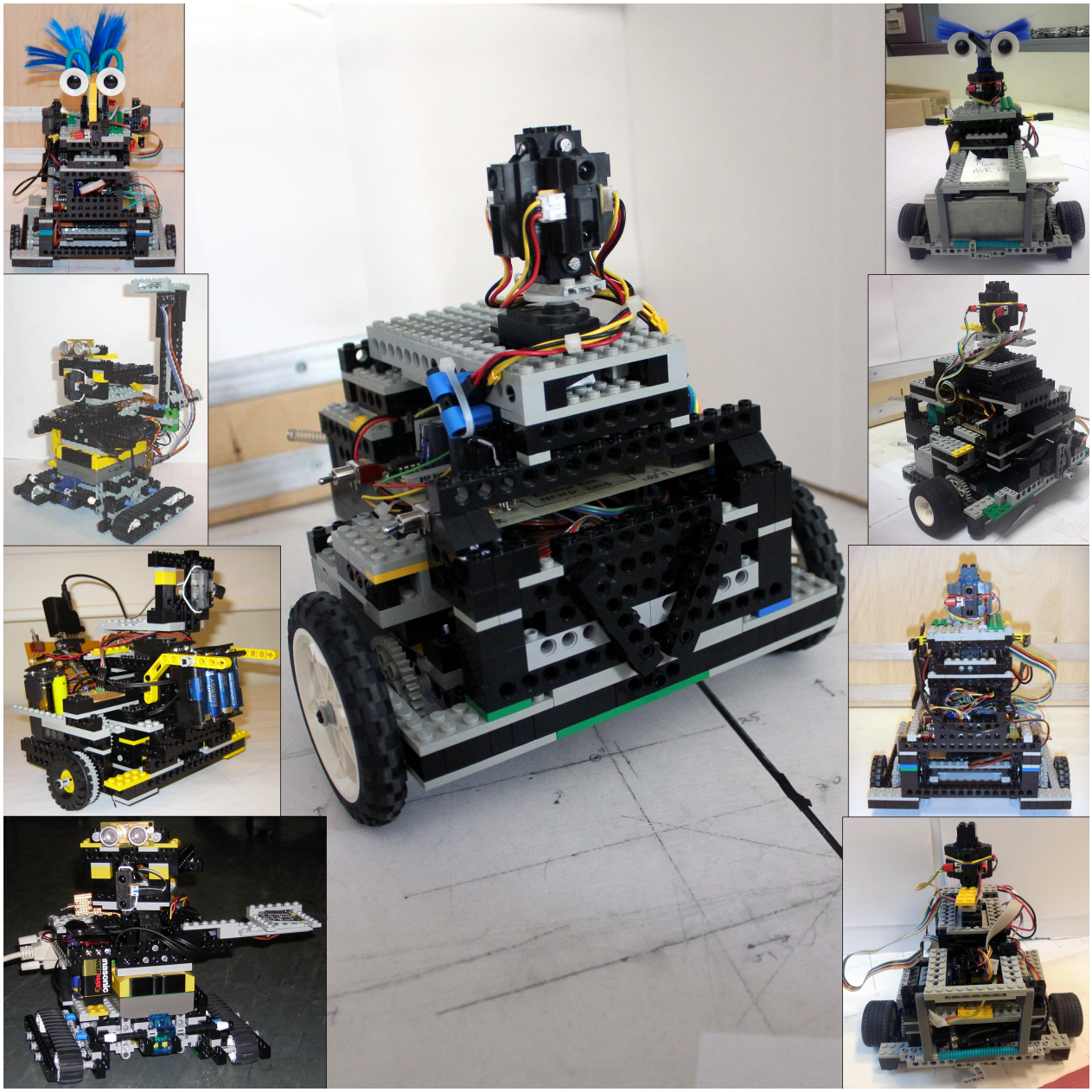
Tusvik (2009), *Prosjekt: Fjernstyring av legorobot*, NTNU.

web: feilipu (2016), 'freertos and libraries for avr atmega with eclipse ide', https://feilipu.me/2011/09/22/freertos-and-libraries-for-avr-atmega/. Besøkt 2016-05-31.

web: github Russi (2016), 'nrf51 multi nus central', https://github.com/marcorussi/nrf51_multi_nus_central. Besøkt 2016-05-31.

web: github Stefans (2016), 'Ev3 uart for nxt', https://github.com/StefansProjects/EV3UARTForNXT. Besøkt 2016-05-31.

web: Hobbylogs (2016), 'Arduino and hmc5883l (with auto calibration)', http://hobbylogs.me.pn/?p=17. Besøkt 2016-05-31.



“Nanos gigantum humeris insidentes.”