NTNU
Det skapende universitet

Eurobot 2010
Navigasjonssystem

Kristin Holst Haaland

Master i teknisk kybernetikk
Oppgaven levert: Juni 2010
Hovedveileder: Sverre Hendseth, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Det skal bygges en robot som skal delta i Eurobot 2010 i Rapperswil-Jona i Sveits 27.-30. mai. I år er det fem masterstudenter og to Eksperter i Team-grupper som jobber sammen. Denne oppgaven går ut på å lage et velfungerende navigasjonssystem for roboten. Dette innebærer:

- Forstå det gamle systemet som ble brukt i fjor og året før, da dette er dårlig dokumentert.
- Benytte seg av deler av dette, eventuelt lage det meste nytt for å få et godt grunnlag for et generelt navigasjonssystem.
- Legge til egenskaper som er nødvendige for denne konkurransen spesielt.
- Ha et godt samarbeid med resten av teamet da det er roboten som helhet som er viktigst, ikke et enkelt delsystem.
- Sørge for kontinuitet i Eurobot-prosjektet. For at neste års deltakere enkelt skal kunne sette seg inn i årets systemer, vil det bli lagt vekt på oversiktlig kode og god dokumentasjon.

Oppgaven gitt: 25. januar 2010
Hovedveileder: Sverre Hendseth, ITK

Forord

Denne rapporten er skrevet i løpet av vårsemesteret 2010. Rapporten beskriver navigasjonssystemet på en robot som hadde som mål å delta i Eurobot Open 2010. Rapporten er skrevet av:

- Kristin Holst Haaland

Jeg vil benytte anledningen til å takke alle teammedlemmene for et godt samarbeid. Det har blitt utvekslet mye kunnskap underveis og vi har lært mye av hverandre.

Jeg vil også takke vår sponsor Kongsberg Gruppen ASA for økonomisk støtte på prosjektet og Sverre Hendseth som har vært veileder for oppgaven min.

Kristin Holst Haaland

Trondheim, juni 2010.

Sammendrag

Eurobot Open er en internasjonal robotkonkurranse som arrangeres hvert år. Det er flere hundre lag av studenter og uavhengige organisasjoner som deltar. NTNU deltar for 10. år på rad. I 2010 er tittelen på konkurransen "Feed the World" og stedet er Rapperswil-Jona i Sveits.

Det var flere svakheter med det tidligere navigasjonssystemet, og det har ikke vært fokusert på å fornye dette systemet de siste årene. Da årets konkurranse stilte strenge krav til nøyaktighet, ble det sett på som nødvendig å redesigne navigasjonssystemet.

Denne oppgaven har tatt for seg design av navigasjonssystemet, som er delt inn i posisjonsestimering og posisjonsregulering. Designet har blitt implementert i C++ og testet på testrobot og på roboten som ble brukt under konkurransen. Navigasjonssystemet er en viktig del av en autonom robot, men det viktigste er at hele systemet fungerer godt sammen. Det ble brukt en del tid på samkjøringstester før konkurransen, spesielt med AI for å få systemene til å fungere sammen.

Det viste seg gjennom en dårlig gjennomført konkurranse at det var blitt brukt for liten tid på samkjøring. Navsys fungerte heller ikke helt optimalt. Det ble tatt et valg om å ikke benytte seg av motorcontroller da systemet oscillerte selv etter betydelig tuning. Dette valget er det satt spørsmål ved i ettertid da det har vist seg å være vanskelig for eksempel å kjøre i sirkelbane. Det er allikevel ønskelig at senere Eurobot-prosjekter baserer navigasjonssystemet sitt på arbeidet fra denne oppgaven.

Innhold

Figurer

Tabeller

Terminologi

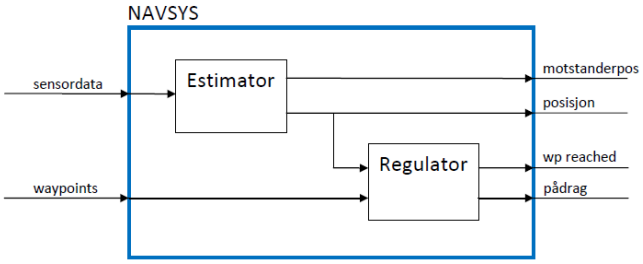
AI	Artificial Intelligence - kunstig intelligens
PID-regulator	Regulator med proporsjonal, integral og derivatledd [5]
Homologering	Kvalifiseringsrunde før Eurobot-konkurransen
Line-of-sight-regulator	Regulator basert på skipsteori [4]
Odometri	Måler hjulrotasjon for å estimere posisjon
CAN-bus	Controller Area Network - meldingsbasert protokoll

Kapittel 1

Innledning

Denne rapporten beskriver arbeidet som er gjort med å utvikle og teste et navigasjonssystem for en autonom robot med tanke på deltagelse i Eurobot Open 2010. Konkurranseregler er forklart i kapittel 2.1.

Navigasjonssystemet er naturlig inndelt i en posisjonsestimator og en regulator for ønsket posisjonsendring. Dette er illustrert i figur 1.1.



Figur 1.1: Blokkdiagram for navigasjonssystemet

Fjorårets navigasjonssystem er kort beskrevet i kapittel 2.2. Det var hensiktsmessig med et nytt design i tillegg til noen utvidelser. Kapittel 3 beskriver teorien som har blitt benyttet, og alternative måter å lage en regulator på. Kapittel 4 gir en innføring i hvordan design og inplementasjon er utført. Testene som er gjennomført er beskrevet i kapittel 5, som tar for seg tester og testresultater. Resultatene blir diskutert i kapittel 6.

Dette prosjektet har vært mer tverrfaglig enn noen gang, og det har følgelig gått med en del tid til kommunikasjon og teamarbeid. Dette har preget prosjektet og

undertegnede sin oppgave i så stor grad at det er valgt å ta med som et eget kapittel; kapittel 9.

Vedlagt denne rapporten er dokumentasjon av systemet laget for senere prosjekter i tillegg A og elektroniske vedlegg i tillegg B.

Da undertegnede var eneste nye student som kom inn før masteroppgaven, og som dermed ikke har skrevet prosjektoppgave var det mye å sette seg inn i som resten av masterstudentene allerede hadde oversikt over. Dette kan sies å både ha vært en fordel og en ulempe. Fordelen har vært at det har vært mange å spørre, og det har i de fleste tilfeller vært mulig å få svar på spørsmål om tidligere system, hva som ble gjort i høst og planer for prosjektet tidlig i semesteret. Ulempen med å komme inn så sent var at det tok såpass lang tid å sette seg inn i tidligere systemer. Dette var noe resten av teamet med masterstudenter hadde tatt seg tid til i høst.

Det var to prosjektoppgaver på fremdriftssystemet i høstsemesteret og det var et ønske om å hele veien være tidlig ute med ferdigstillelse av mekanikk. Planen var å ha en testplattform klar for testing tidlig. Av ulike årsaker ble dette utsatt flere ganger. Da det var planlagt testing på testplattform med det nye fremdriftssystemet ble det valgt å vente med oppstart av testing til dette var ferdigstilt. Det var ikke planlagt at det skulle gå så lang tid, og i ettertid er det lett å være etterpåklok og tenke at det gamle systemet burde vært benyttet i første del av semesteret.

I tillegg til lite testing før testplattformen stod klar, var det ugunstig å begynne å teste navsys med en lite testet motorcontroller. Dette ble ikke klart før det var prøvet og feilet mange ganger og det ble funnet ut at problemet ikke lå i navsys. Etter en diskusjon med andre i teamet ble det planlagt å forsøke å styre systemet uten dedikerte motorcontrollere. Dette fungerte til en viss grad, selv om beslutningen nok ble tatt litt raskt, og det kan sees i ettertid at motorcontrollere bør være en del av systemet. Dette står det mer om i kapittel 4.2.2.

Kapittel 2

Bakgrunn

2.1 Konkurransen

Eurobot generelt

Eurobot Open er en internasjonal robotkonkurranse som foregår i Europa hvert år i mai. Eurobot har utviklet seg fra Frankrike og den første Eurobot Open ble arrangert i La Ferté-Bernard i Frankrike i 1998. NTNU ved Institutt for Teknisk Kybernetikk har deltatt hvert år siden 2000. Konkurransen er åpen for studenter og uavhengige organisasjoner fra hele verden. Eurobot sin visjon er å oppmuntre og fremme ingeniørkunst og interesse for roboter. Konkurransen skal være "fair play" og de konkurrerende lagene er vanligvis villige til å diskutere teknologien bak og hvordan robotene deres fungerer.

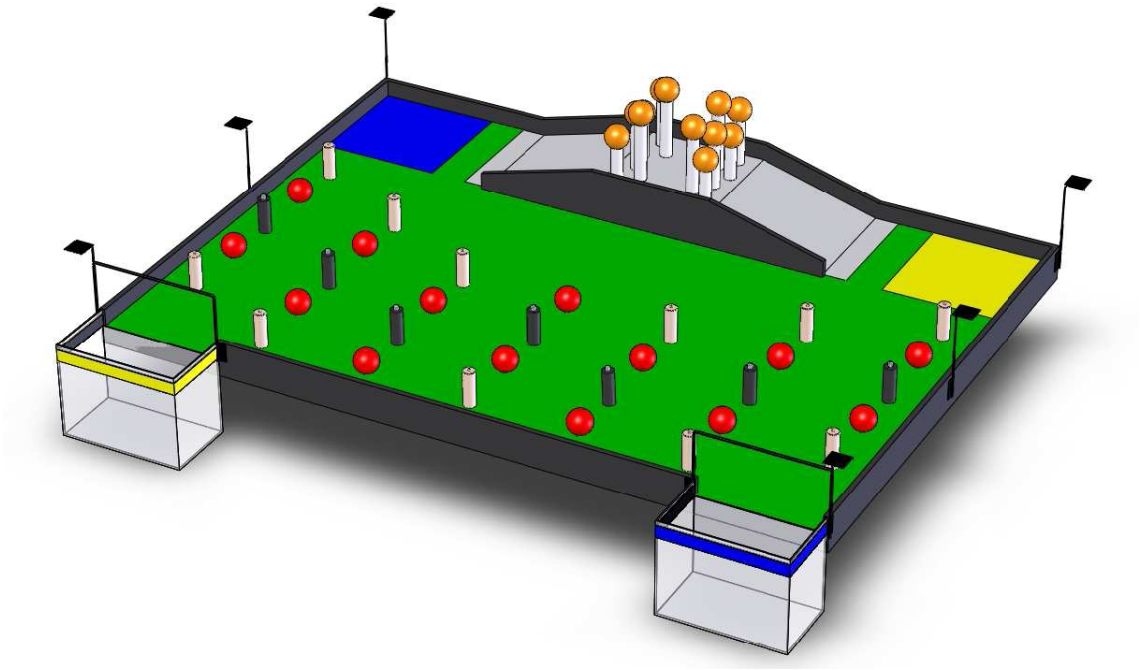
Konkurransen går ut på at to og to roboter konkurrerer mot hverandre på et spillebrett som er 2×3 meter. En kamp varer i 90 sekunder og robotene kjører helt autonomt. Det konkurreres om å sanke flest poeng. Reglene forandres fra år til år.

Årets konkurranse

Årets konkurranse ble avholdt i Rapperswil-Jona i Sveits 27.-30. mai 2010. Årets tema var "Feed the World" [3], og konkurransen gikk ut på å plukke frukt og grønnsaker. Spillelementene forestiller appelsiner, tomater og mais. Det er i tillegg syv svarte sylindre som er hindringer og er skrudd fast i brettet. Spillebrettet er vist i figur 2.2. Årets lagfarger er gult og blått.

Figur 2.1: Temalogo for konkurransen

Startområdet til hvert av lagene er indikert med en farget firkant i starthjørnene. Målbeholderene er plassert på motsatt side av brettet. Spillelementenes vekt avgjør hvor mange poeng de gir. Appelsiner, som er de oransje ballene oppe på "åsen", gir 300 poeng, tomater, som er de røde ballene, gir 150 poeng og mais, som er hvite sylindre, gir 250 poeng. Det er 12 appelsiner, 14 tomater og 11 mais til sammen på brettet. Det er lov å benytte seg av tre "fyrtårn" rundt brettet til posisjonering. Alle regler og størrelser på spillelementer kan finnes i [3].



Figur 2.2: Spillebrettet til årets konkurranse

Homologering

Hvert lag må gjennom en kvalifiseringsrunde for å delta i konkurransen. Denne kvalifiseringen kalles homologering, fra fransk; homologation, som betyr godkjenning. Homologeringen er delt i to deler; en fysisk og en praktisk. Den fysiske godkjenningen skal teste om roboten er innenfor fysiske størrelseskrav og at lasere er godkjent for bruk i konkurransen. Den praktiske kvalifiseringen skal se om roboten kan levere poeng og at antikollisjonssystemet fungerer tilfredsstillende.

2.2 Tidligere navigasjonssystem

Rammeverket for software på roboten er videreutviklet fra det som ble laget av Gunnar Kjemphol og Kristian M. Knausgård i 2007 [11]. Softwaren er delt inn i fire prosesser [18]:

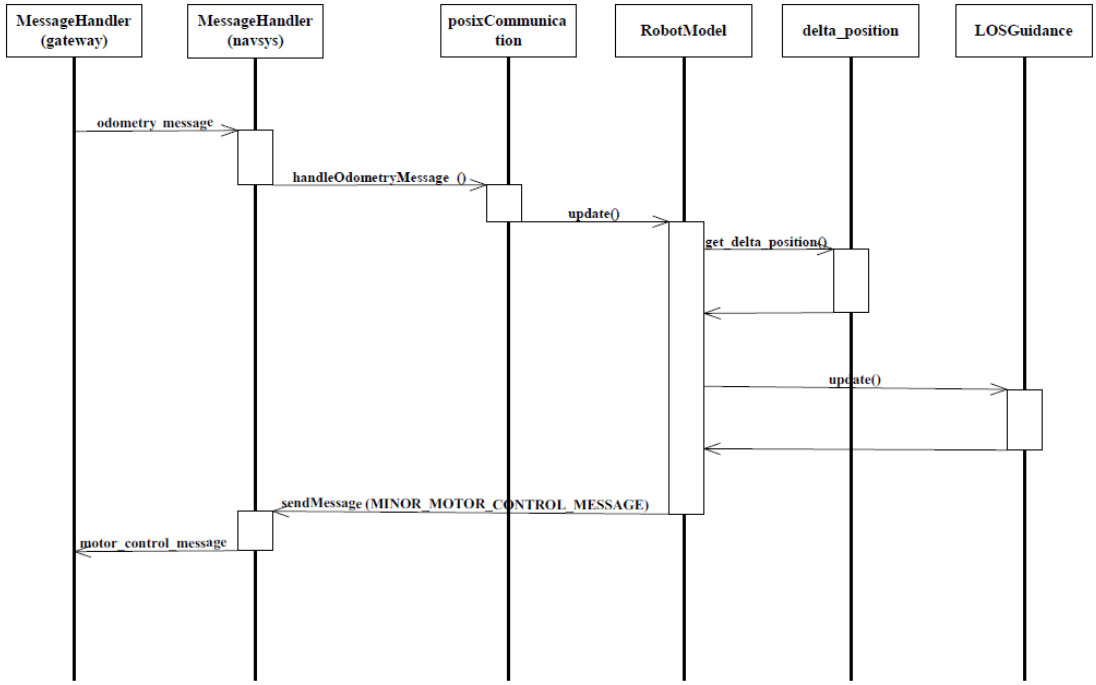
- Gateway
- Navsys
- Plansys
- Cvsys

Disse fire hovedprogrammene har blitt beholdt også i år, og det har vært fire forskjellige personer med hvert sitt ansvarsområde.

Navigasjonssystemet som er brukt de siste årene er også i hovedsak utviklet i 2007 [11, 18]. I motsetning til plansys og cvsys, har det i 2008 og 2009 bare blitt gjort små forandringer på navsys.

Navsys er den prosessen som bahandler estimering av robotens posisjon og tar seg av navigering på spillebrettet [18]. Tidligere bestod navsys av tre hovedklasser; RobotModel, OdometryObservation og LOSGuidance. Main.metoden ligger i PinpointedRuntime.cpp og er i hovedsak en tom while-løkke. Systemet bruker interruptbasert MessageHandler og kjøres ca 100 ganger i sekundet, hver gang det mottas en posisjonsmelding fra gateway. Et sekvensdiagram for regulatorsløyfen kan sees i figur 2.3, hvor OdometryObservation brukes i *delta_position* for posisjonsestimering.

Waypoints sendt til navsys var i form av en liste med mellom-waypoints og et ende-waypoint. Dette kom fra plansys sin ruteplanlegger som beregnet rute fra nåværende estimert posisjon til ønsket sluttposisjon. Listen bestod av forskjellig typer waypoints og hadde egen tag som sa hvilket ende-waypoint den var på vei til.



Figur 2.3: Sekvensdiagram for fjorårets navigasjonssystem

2.2.1 Posisjonering

Sensorer som har blitt benyttet tidligere har vært odometri som relative målinger og et trianguleringssystem som absolutt posisjonering. Nøyaktigheten på odometrimålingene fra fjorårets robot kan sees i tabell 2.1, tatt fra prosjektoppgaven til Christian W. Kjølseth og Øystein Wergeland [13].

	Distanse, 500 mm		Rotering 90°	
Forsøk	Frem	Tilbake	Frem	Tilbake
1	500	0	89,7	0,8
2	500	1	89,7	0,1
3	500	0	89,9	0,0
4	501	0	90,2	0,0
5	500	0	90,0	0,5
Gj.snitt	500,4	0,3	89,72	0,3

Tabell 2.1: Nøyaktighet på odometrimålinger fra gammelt system

Målingene ser her veldig nøyaktige ut, men det nevnes at testen ble utført forsiktig. Avstanden kjørt er relativt kort for å kunne bestemme nøyaktigheten på dette

systemet.

Det ble i fjor også testet ut et redundant system til odometri. Dette var en lasermodul som hadde to lasere vendt ned mot bakken, den ene i rotasjonsaksen og den andre langt bak på roboten. Dette var for å gjøre de relative målingene så nøyaktige som mulig da det har blitt sett på som vanskelig å benytte seg av fyrtårnene til absolutt posisjonering. Dette ble testet ut, men ikke benyttet seg av da det var vanskelig å posisjonere nøyaktig på bunnplaten.

Det absolutte posisjoneringssystemet besto av tre fyrtårn rundt spillebrettet i tillegg til det roterende tårnet på roboten; Wall-E. Dette systemet ble godt testet før fjorårets eurobot-konkurranse, og ble ansett som meget nøyaktig med en vinkel-oppløsning på $0,25°$. Systemet er dokumentert i [14]. Absolutt posisjonering ble ikke brukt under konkurransen da det var vanskelig å posisjonere fyrtårnene rundt spillebrettet tilstrekkelig nøyaktig.

Det ble laget en avansert posisjoneringsalgoritme basert på et partikkelfilter med en observasjonsmodell som skulle sørge for et best mulig posisjonsestimat ut fra tilgjengelige målinger [11, 7]. Dette har riktignok ikke blitt bygget videre på da det ble ansett som like fornuftig å la absolutte posisjonsmålinger overstyre de relative, og la de være grunnlaget for videre posisjonering [10].

2.2.2 Regulering

Reguleringsalgoritmen som har blitt brukt er en "line-of-sight"-regulator som er hentet fra [4]. Denne algoritmen lager en vektor fra fartøyet til et punkt på veien for at den skal tvinge fartøyet tilbake til å følge banen når det er ute av kurs. Denne er riktignok meget forenklet, men den er likefullt basert på skipsoppførsel. Algoritmen vil blant annet aldri velge å rygge til et waypoint da fremdriftsretningen er fremover. Dette har ført til at flere modifiseringer er lagt til etter hvert. Dette er muligens noe av det som har gjort programmet vanskelig å sette seg inn i. LOSGuidance er et eget objekt som regner ut ønsket rotasjons- og translasjonshastighet [10].

Selv om navsys de siste årene har blitt relativt uoversiktlig og vanskelig å sette seg inn i, har det klart å føre roboten fra A til B. Dette førte til at navsys heller ikke ble prioritert før fjorårets konkurranse. Tidligere års posisjonsregulatorer har ikke fått mange dedikerte tester. Det er ikke lett å teste denne isolert da testene stort sett går ut på nøyaktighet til estimatoren og hastighet/akselerasjon til fremdriftssystemet. Navigasjonssystemet er hvert år sagt til å fungere tilstrekkelig, eventuelt med mulige justeringer. De siste årene har det ikke vært fokus på annet enn små justeringer. Det

har derfor vært ønskelig med et redesign eventuelt nydesign av navigasjonssystemet for Eurobot2010.

Kapittel 3

Teori

Et navigasjonssystem er et system som skal holde oversikt over egen posisjon og vise hvordan man kan komme til en ønsket posisjon. I dette tilfellet er egen posisjon estimert fra forskjellige sensorer. Systemet er i hovedsak delt inn i to oppgaver: Posisjonsestimering og posisjonsregulering.

3.1 Estimator

Estimatoren er den delen av navigasjonssystemet som tar inn målinger og beregner estimert posisjon. Den estimerte posisjonen beregnes fra relative posisjonsmålinger og/eller absolutte posisjonsmålinger.

Relative posisjonsmålinger gir en hastighet eller akselerasjon og må integreres opp for å kunne beregne posisjon. Dette gjør så feil akkumuleres opp over tid og gir drifting av målingene. Denne typen sensorer er allikevel regnet for å være nøyaktige over relativt korte perioder, og oppdateringsraten regnes som relativt hurtig.

Absolutte posisjonsmålinger gir ny posisjon ved hver måling, og gir ikke drifting i resultatene. Det er flere mulige feilkilder også ved disse målingene. For å oppnå best mulig estimat er det ofte hensiktsmessig å utnytte begge målingstypene.

3.1.1 Relative posisjonsmålinger

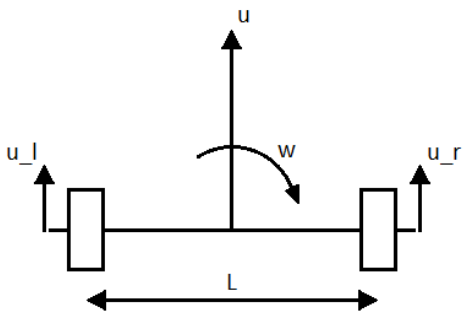
Odometrimålinger er relative posisjonsmålinger da de gir forandring i posisjon mellom hver måling. Odometri har vært brukt som posisjonsmåling på eurobot i mange år,

og brukervennligheten tilsier at det er verdt å fortsette med. Målingene fås fra to løpehjul plassert med en viss avstand parallelt med drivhjulene. Robotens rotasjonsakse bør ligge midt mellom løpehjulene.

Estimatoren med input kun fra odometrimålingen får en ny relativ posisjonsmåling hver gang den kjører. Den regner ut ny posisjon med utgangspunkt i forrige posisjon. Dette impliserer at startposisjon og -vinkel må være nøyaktig. Målingene blir summert opp og ved nøyaktige odometrimålinger får man et godt posisjonsestimat gjennom en hel kamp. Avstanden og vinkelen roboten har forflyttet seg siden forrige måling finnes fra ligninger 3.1

$$\begin{aligned} \Delta x_{body} &= \frac{u_{left} + u_{right}}{2} \\ \Delta \psi_{body} &= \frac{u_{left} - u_{right}}{L} \end{aligned} \tag{3.1}$$

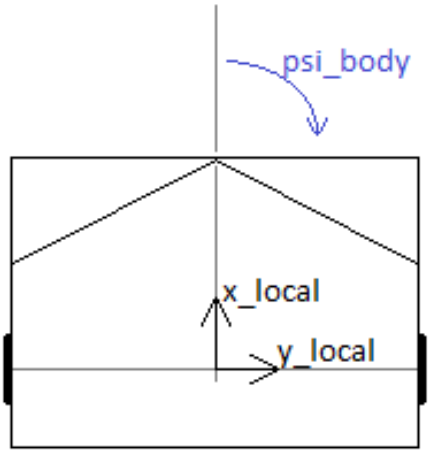
hvor u_{left} og u_{right} er avstand odometri-løpehjulene har kjørt siden forrige måling. Δx_{body} og $\Delta \psi_{body}$ er avstands- og vinkelforandring i robotens *body*-koordinatsystem. En illustrasjon av denne omregningen kan sees i figur 3.1.



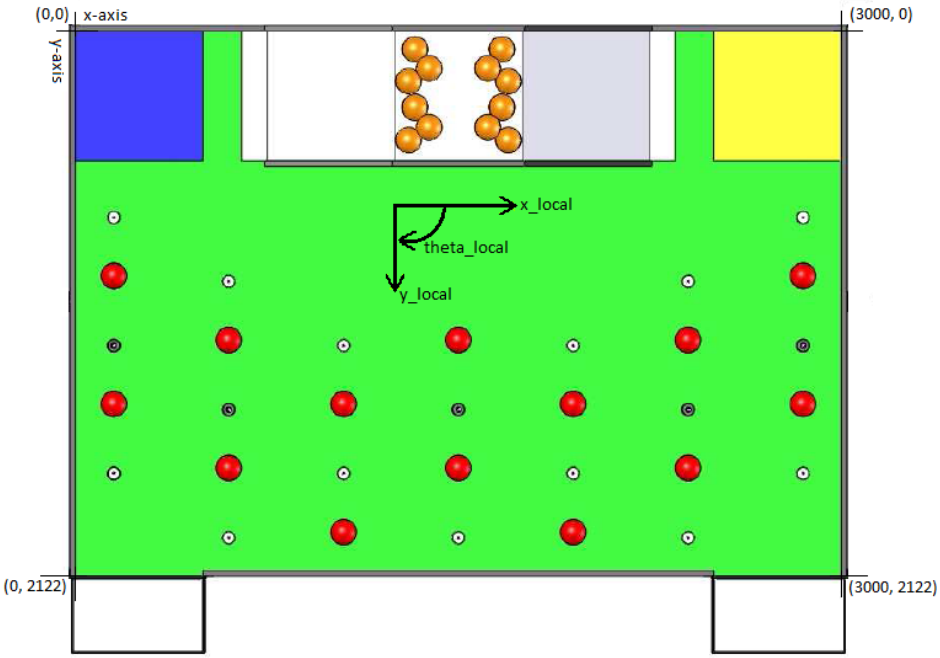
Figur 3.1: Oversikt over omregning av posisjonsforandring

Systemet opererer med to koordinatsystemer. Et system sitter på brettet og kalles i oppgaven for *local*, se figur 3.3. Det andre sitter på roboten, med origo i rotasjonssenter, og kalles i oppgaven for *body*, figur 3.2. Posisjon og vinkel på spillebrettet er definert i *local*-koordinatsystemet, med heading som retningen roboten står plassert i forhold til x-aksen påbrettet.

Utfra koordinatsystemene i figur 3.3 og figur 3.2 kan man finne en homogentransform som gir ny posisjon i *local*-koordinatsystemet [9]. Denne kan sees i ligning 3.2.



Figur 3.2: Koordinatsystemet på roboten



Figur 3.3: Koordinatsystemet på brettet

$$
\begin{aligned}
H_k^i &= H_{k-1}^i * H_k^{k-1} \\
&= \begin{bmatrix} cos(\theta_{local}) & -sin(\theta_{local}) & 0 & x_{local} \\ sin(\theta_{local}) & cos(\theta_{local}) & 0 & y_{local} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} cos(\psi_{body}) & -sin(\psi_{body}) & 0 & dcos(\frac{\psi_{body}}{2}) \\ sin(\psi_{body}) & cos(\psi_{body}) & 0 & dsin(\frac{\psi_{body}}{2}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

$$
H_k^i = \begin{bmatrix} cos(\theta_{local} + \psi_{body}) & -sin(\theta_{local} + \psi_{body}) & 0 & dcos(\theta_{local} + \frac{\psi_{body}}{2}) + x_{local} \\ sin(\theta_{local} + \psi_{body}) & cos(\theta_{local} + \psi_{body}) & 0 & dsin(\theta_{local} + \frac{\psi_{body}}{2}) + y_{local} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}
$$

H_{k-1}^i er homogentransformen fra *local* (i) til robotens forrige posisjon og H_k^{k-1} er homogentransformen fra robotens forrige posisjon til den neste. d er avstanden roboten har forflyttet seg og ψ_{body} er vinkelforandring.

3.1.2 Absolutte posisjonsmålinger

Absolutt posisjonering kan fås fra lasertårn, som det var tenkt i fjor. Det er ønskelig med et system som kan oppdatere seg raskt nok så det kan brukes også i bevegelse. Fjorårets system var avhengig av at roboten stod stille for at det skulle gi fornuftige målinger. Da årets robot ikke er tenkt å stå i ro tilstrekkelig lenge iløpet av en kamp, er det nødvendig med videreutvikling av dette systemet. Konkurransereglene tillater tre fyrtårn plassert rundt kanten av spillebrettet.

Datasynet på roboten har blitt godt utviklet før dette årets konkurranse og en mulig løsning er å benytte seg av informasjon om avstand til spillelementer i kombinasjon med estimert posisjon fra odometrimålinger. Det er nødvendig å vite omtrentlig posisjon da algoritmen til datasyn har behov for å relatere ny informasjon med antatte posisjoner på spillelementer relativt til roboten.

Det ble gått til innkjøp av en lidar, en sveipende laserstråle som gir avstand og vinkel til objektene i nærheten. Den fungerer på samme måte som radar. Tanken bak dette innkjøpet var at den kunne brukes til absolutt posisjonering, men da det viste seg at det ikke ble prioritert plass til den på roboten ble ikke dette videreutviklet.

3.2 Regulator

Regulatoren tar inn estimert posisjon og ønsket posisjon og gir ut et pådrag til fremdriftsystemet. På roboten er det to frihetsgrader; kjøring rett frem og rotasjon. Det er også to pådrag for fremdrift; ett signal til hvert av fremdriftshjulene. Dette gir en todimensjonal regulator som er delt inn i translasjon og rotasjon. Denne inndelingen kan finnes i [1], i tillegg til at det er sånn det har blitt gjort på Eurobot tidligere.

Det er flere måter å beregne ønsket bevegelse for å minke posisjonsavviket. De siste årene har det vært brukt en såkalt "line-of-sight"-algoritme hvor regulatoren er basert på skipsteori. Det blir hele tiden beregnet en linje fra nåværende posisjon til en posisjon langs ønsket vei, se [4]. Det har vært argumentert for at denne har tatt hensyn til værfenomener som bare oppstår på sjøen, og derfor ikke er helt ideel på et brett i en eurobotkonkurranse.

Regulatoren kan utformes som et optimaliseringsproblem hvor avviket mellom estimert posisjon og ønsket posisjon er det som ønskes minimalisert. I dette tilfellet vil objektfunksjonen være posisjonsavviket. Begrensninger på problemet er blant annet pådragsbegrensninger og akselerasjonsbegrensninger i tillegg til at roboten ikke kan bevege seg i y-retning alene. Mer om denne fremgangsmåten kan leses i [6].

Det er ikke nødvendig med avanserte algoritmer for å styre roboten til ønsket posisjon. Da det har vært ønskelig før årets konkurranse å kunne kjøre i oppsatte sirkelbaner er det en mulighet å styre translasjons- og rotasjonsbevegelse separat. To separate PID-regulatorer styrer hver sin del av systemet og blir slått sammen til felles pådrag til motorene. Regulatorene kan vektes forskjellig avhengig av avstand og retning til neste waypoint. Sirkelkjøring oppnås ved at rotasjonshastigheten er proporsjonal med translasjonshastigheten.

Omregning fra ønsket translasjonshastighet i mm/sek og rotasjonshastighet i rad/sek til ønsket hastighet på hvert av fremdriftshjulene blir som i ligning 3.3 tatt fra [1].

$$\begin{aligned} u_l &= u_{body} + \frac{L}{2} w_{body} \\ u_r &= u_{body} - \frac{L}{2} w_{body} \end{aligned} \tag{3.3}$$

hvor u_{body} er translasjonshastighet og w_{body} er rotasjonshastighet. L er avstand mellom odometrihjulene.

Kapittel 4

Design og implementasjon

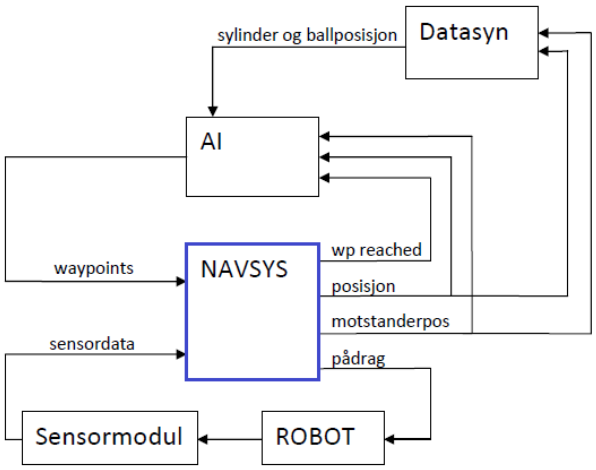
Navigasjonssystemet som er utviklet i år har tatt utgangspunkt i det tidligere systemet og beholdt det overordende klassesystemet. Det har også kommet noen store forandringer og koden er mer eller mindre fullstendig skrevet på nytt. Blant annet kjøres estimatoren bare etter oppdatert odometridata og regulatoren er helt byttet ut. Dokumentasjon av implementasjon finnes i tillegg A.

4.1 Navsys på roboten

Et overordnet blokkdiagram kan sees i figur 4.1. Her er sammenheng mellom andre delsystemer illustrert. Fra AI får navsys en liste med waypoints som består av to elementer. Disse har AI beregnet ut fra sin strategi om optimal kjørerute. AI oppdaterer waypointslisten hver gang et waypoint er nådd eller ved strategiforandring. Systemet kan også forandre strategi hvis roboten har satt seg fast. Navsys må derfor takle å få nye waypoints både etter å ha kommet frem til et waypoint og når roboten er underveis.

Navsys tar inn sensordata fra sensormodulen. Sensorer som er brukt er odometri for relativ posisjonering og lasertårn for absolutt posisjonering. Odometrimålingene kommer samtidig fra hver av encoderskivene og gir antall step kjørt siden kampstart. Fra lasertårnet kommer informasjonen som avstand og vinkel til hvert tårn. Tårnene det gis informasjon om er fyrtårnene som er plassert rundt brettet for posisjonering i tillegg til det som er plassert på motstanderroboten.

Navsys setter pådrag på fremdriftssytemet basert på input til regulatoren. Pådrag gis separat til hver av motorene som PWM-styrte spenningssignal. Posisjon og



Figur 4.1: Overordnet blokkdiagram

motstanderposisjon blir sendt til AI og datasyn hvor det blir brukt til motstanderunngåelse, strategiplanlegging og bildebehandling. *waypointReached* sendes til AI når et waypoint er nådd for å få oppdatert waypointliste.

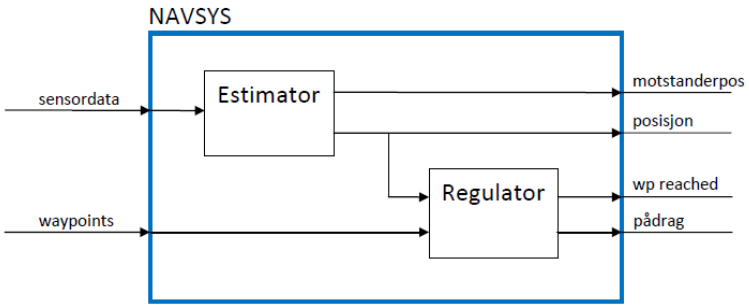
4.2 Inndeling av systemet

Det har vært naturlig å dele systemet inn i tre hovedklasser; RobotModel, Estimator og Regulator. Dette er en videreutvikling av det gamle systemet som også hadde tre tilsvarende klasser. Estimator tar inn sensordata fra sensormodul og estimerer robotens posisjon og motstanderposisjon. Regulator styrer roboten inn til ønsket waypoint. Det er disse som til sammen utgjør hovedfunksjonene i navsys. Dette kan sees i et mer detaljert blokkdiagram i figur 4.2. RobotModel er en overordnet klasse som blant annet håndterer kommunikasjon.

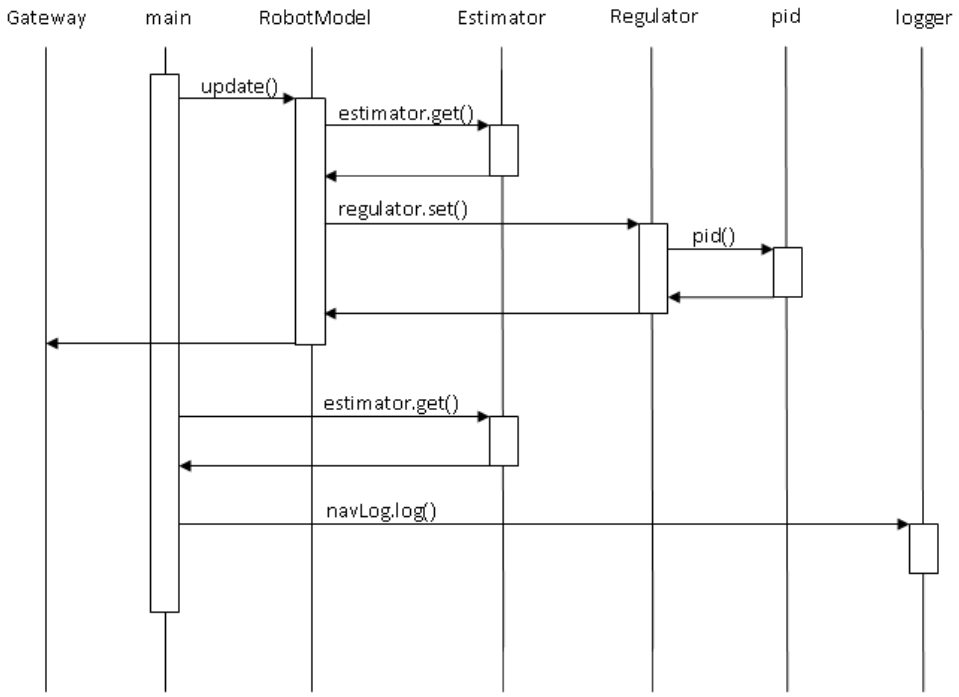
Sekvensdiagrammet i figur 4.3 viser hvordan programflyten går gjennom systemet. Estimatoren kjører hver gang det kommer ny odometriinformasjon, mens regulatoren oppdaterer seg med en frekvens på ca 100 Hz.

4.2.1 Estimator

Estimatoren består i hovedsak av en set()-metode som tar inn antall step odometriløpehjulene har kjørt siden forrige måling. Dette blir omregnet til posisjonsforandring



Figur 4.2: Blokkdiagram for navsys



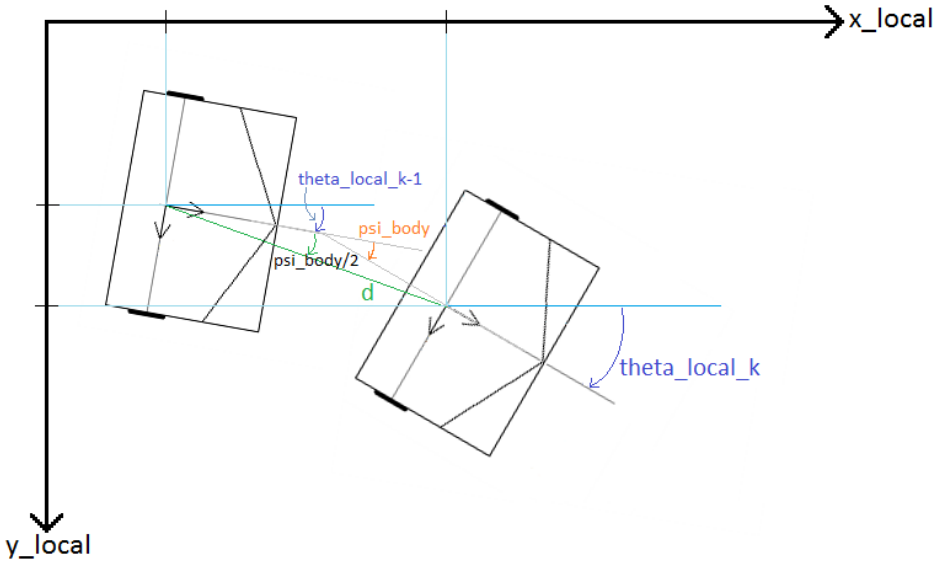
Figur 4.3: Sekvensdiagram for navsys

og vinkelforandring på roboten, ligning 3.1, og oppdaterer posisjons- og vinkelestimatet ved ligningene 4.1 - 4.3,

$$x_{local_k} = x_{local_{k-1}} + dcos(\theta_{local_{k-1}} + \frac{\psi_{body}}{2}) \tag{4.1}$$

$$y_{local_k} = y_{local_{k-1}} + dsin(\theta_{local_{k-1}} + \frac{\psi_{body}}{2}) \tag{4.2}$$

$$\theta_{local_k} = \theta_{local_{k-1}} + \psi_{body} \tag{4.3}$$



Figur 4.4: Koordinatsystemene og hvordan de hører sammen

som er utregnet fra ligning 3.2. Vinklene i utregningen er visualisert i figur 4.4. Set-metoden blir kjørt hver gang det kommer en ny odometri-måling. Dette skjer i communication.cpp hvor all kommunikasjon blir behandlet.

Det ble stilt spørsmål om hvordan odometri-samplingen burde utføres. Det er i hovedsak to måter som ble vurdert. Enten sampling etter gitte tidsintervaller eller sampling etter et gitt antall step på encoderskivene. Dette ble diskutert med andre lag under konkurransen i Sveits. Det er fordeler og ulemper med begge måtene; tidssampling kan gi avrundingsfeil ved lave hastigheter da roboten ikke nødvendigvis kjører 100 % rett frem. Ellers vil tidssampling gi nøyaktig hastighet ved tilstrekkelig høye hastigheter. Stepsampling kan settes til å oppdatere informasjon etter veldig korte avstander, og kan derfor bli veldig nøyaktig også ved lave hastigheter. Dette vil gi tilsvarende høy oppdateringsrate hvis hastighetene blir store. Det ble diskutert

og kommet frem til at forskjellen er minimal, og at det er andre feilkilder som gir større avvik. Det ble derfor besluttet at odometrimålingene skulle gjøres etter faste intervaller; hvert 10 ms.

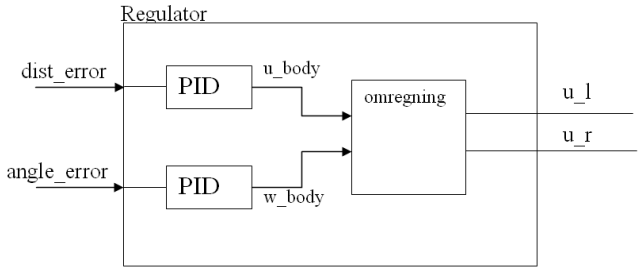
Estimatoren blir oppdatert tilsvarende ofte som odometrimålingene. Da det ble oppdaget at odometrioppdateringene ikke skjedde med helt jevn frekvens, ble det besluttet å legge estimatorens set-metode til å kjøre rett etter at odometrimålignene mottas. Dette var for at odometri-hastigheten ikke skulle kunne gi null når roboten var i bevegelse da to beregninger noen ganger ble gjort uten at det hadde kommet en ny odometrimåling i mellomtiden. Samplingsfrekvensen ble beholdt til å være 0,01, det vil si 100 Hz. Selv om frekvensen på mottatte odometrimålinger varierte noe, var samplingsfrekvensen på sensorkortet tilstrekkelig konstant [8].

4.2.2 Regulator

Valg av regulatorstruktur

Etter fjorårets prosjekt ble det ytret et ønske om et nytt navigasjonssystem. Da tidligere regulator bestod av en skipsbasert "line-of-sight"-algoritme, var det ønskelig med noe som tar utgangspunkt i en robot som beveger med hjelp av hjul. En optimaliseringsalgoritme for å minimere avstand mellom estimert og ønsket posisjon ble vurdert. Dette ble ansett som en god mulig løsning da algoritmen vil gi ut optimalisert bevegelse. En ulempe for denne måten å løse problemet på er at AI og navsys er to separate systemer. Det hadde vært ideelt for optimaliseringsalgoritmen å få informasjon om hindringer på brettet i tillegg til ønsket ny posisjon. Dette var det ikke lagt opp til. AI sender ønsket kjørebane, og navsys har ansvar for å følge denne. For å holde det enkelt var det naturlig å velge en dobbel PID-regulator; en som justerer avstand og en som justerer vinkel. Regulatorstrukturen kan sees i figur 4.5. Det er mulig å vekte pådraget fra de forskjellige regulatorene forskjellig avhengig av avstand til waypoint.

For å gjøre systemet oversiktlig og enkelt å sette seg inn i, ble regulatoren delt mellom en pid-klasse og en Regulator-klasse. Pid-klassen er en liten klasse som inneholder pid-parametre, setpunkt og en run-metode som kjører selve regulatoren. Både avstandsregulatoren og vinkelregulatoren bruker denne klassen, som kjøres fra Regulator-klassen. Omregningen fra u_{body} og w_{body} til u_l og u_r skjer i RobotModel før pådraget sendes via Gateway.



Figur 4.5: Regulatorstruktur

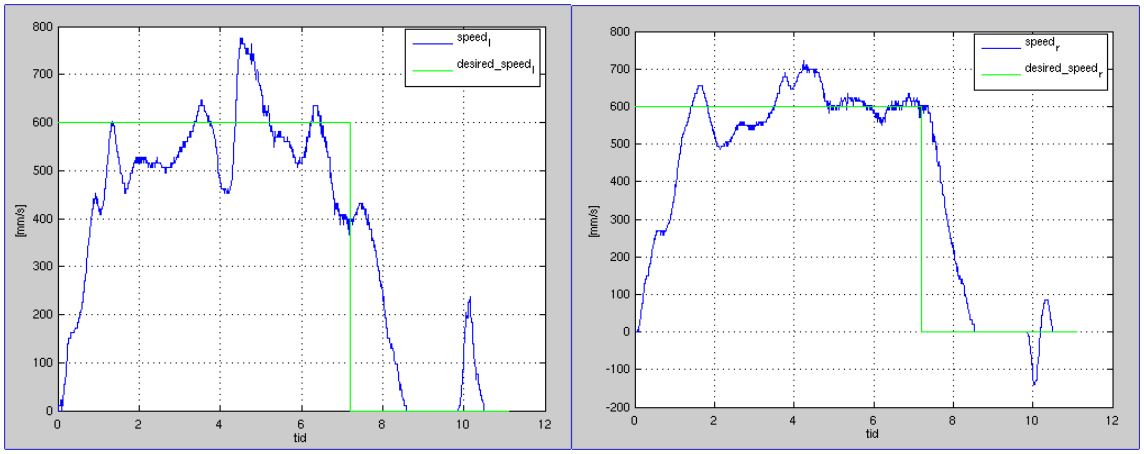
Pådrag

Det var planlagt at output fra navsys skulle være en hastighetsreferanse på hver av fremdriftsmotorene. Dette ble implementert som illustrert på figur 4.5 hvor u_l og u_r i dette tilfelle er ønsket hastighet langs bakken på hvert av hjulene. Input til regulatorene var $dist_{error}$ som er avstandsfeil i mm og, i dette tilfelle, $angle_{error}$ som vinkelfeil i radianer. Omregningen ble gjort ved hjelp av ligning 3.3

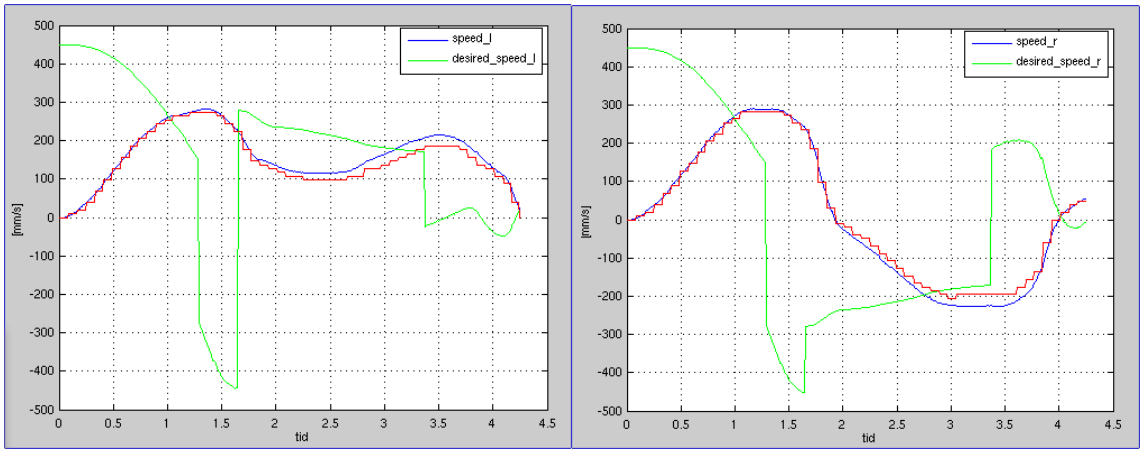
Hastighetsreferanse ble sendt som pådrag til motorcontrolleren, som ble implementert av Stian Søvik i hans prosjektoppgave [15]. Tanken bak dette oppsettet var bra, men det var vanskelig å gjennomføre samkjøringen optimalt. Det ble observert at roboten tilslutt kom seg til riktig posisjon, men det var oscillasjoner og usikkerheter som viste seg å være vanskelige å tune bort. Ved hjelp av logdata ble det laget plot av hastighet fra odometrimålingene i forhold til referansehastighet. Plot fra test med fast referansehastighet for venstre og høyre motor kan sees i figur 4.6. Plot fra test med ønsket hastighet fra navsys kan sees i figur 4.7.

Det kan argumenteres for at det er mulig å tune denne regulatoren tilstrekkelig. Det ble allikevel ikke ansett som undertegnedes oppgave. Løsningen ble derfor å styre PWM-signalene inn på motoren direkte fra navsys. Det vil si at motorcontrollerne ikke ble brukt. Denne beslutningen ble tatt under tidspress og det har i ettertid vist seg at deler av systemet har blitt vanskeligere å styre. Friksjon mot spillebrettet og sirkelkjøring har skapt problemer underveis. På grunnlag av dette er det ønskelig å gjeninnføre motorcontrollere. Det har i ettertid blitt forslått å øke frekvensen på motorcontrolleren betraktelig for å få en raskere respons. Det er mulig dette vil gjøre tuningen enklere.

Systemet som ble brukt bestod av to regulatorer som på figur 4.5. Det som ble forandret var at vinkelfeilen, $angle_{error}$, ble regnet om til mm. Dette ble gjort ved at error i radianer ble gjort om til error i mm langs rotasjonsradien: $error_{ny} =$



Figur 4.6: Motorhastighet i forhold til fast referansehastighet



Figur 4.7: Motorhastighet i forhold til referansehastighet fra navsys

$error_{gammel} * L/2$. I tillegg ble omregningen i RobotModel forandret fra ligning 3.3 til ligning 4.4. Hvor u_l og u_r ikke lenger er hastighetsreferanse, men spenningspådrag som PWM-signal.

$$\begin{aligned} u_l &= u_{body} + w_{body} \\ u_r &= u_{body} - w_{body} \end{aligned} \tag{4.4}$$

PID-regulatoren som brukes i Regulator er enkel og uten modifikasjoner, ligning 4.5. Error ble valgt til å være avstanden fra waypointet til estimert posisjon. Dette var praktisk da estimert posisjon ligger i *robot*-koordinatsystemets origo. Det ble brukt to regulatorer som sammen skulle gi pådrag til hver av motorene. Alle begrensninger

på pådraget ble gjort etter at det var delt opp til høyre og venstre motor. Det vil si at alle begrensninger ligger i RobotModel, og er på de oppdelte signalene til hver av motorene.

$$u = K_p\,(error) + K_i \int (error)\,dt + K_d * \frac{d}{dt}\,(error) \tag{4.5}$$

4.2.3 RobotModel

RobotModel er den klassen som setter i gang regulatoren når et waypoint har blitt oppdatert. Hvis waypoints enda ikke er sendt, eller hvis det kommer en tom liste, vil systemet kjøre i en tom while-løkke. Etter hver gang regulatoren kjøres, regnes translasjon- og rotasjonspådrag om til pådrag på hver av motorene før det blir sendt over gateway. Etter omregning til pådrag u_l og u_r blir pådragsbegrensningene påført. Dette sikrer at det ikke sendes for høye pådrag og at pådragsforandringene ikke er for store. Hvis pådraget til en av motorene er mindre enn pådraget som skal til for å overkomme friksjonen i systemet, vil pådraget økes. Forholdet mellom pådragene beholdes i den grad det er mulig.

4.2.4 Kommunikasjon

All kommunikasjon mellom navsys og andre moduler behandles i communication.cpp. Den kjører som en egen tråd og behandler meldinger når de kommer. På denne måten kan estimatoren kjøres fra communication.cpp hver gang det kommer ny odometri-informasjon. Posisjon og motstanderposisjon sendes på samme måte direkte til datasyn hver gang de oppdateres. Metoder for sending av annen informasjon er opprettet i communication.h og kjøres i RobotModel.

4.3 Spesialtilfeller

4.3.1 Sirkelkjøring

Årets spillebrett hadde flere hindringer enn det har vært tidligere. For at roboten skulle kunne kjøre rundt raskest mulig, og lengst mulig unna hindringene var det ønskelig å ha mulighet for sirkelkjøring. Waypoints fått fra AI sier om roboten skal

kjøre direkte til neste waypoint eller om den skal kjøre i en sirkelbane til neste waypoint. Det er illustrert i figur 4.8 hvordan kjørebanen er tenkt.

Figur 4.8: Kjøring med sikrelbaner

For at roboten skal klare å holde en sirkelbane må vinkelhastigheten være proporsjonal med translasjonshastigheten. Hvis det skal kjøres langs en sirkel med radius R må forholdet være:

$$\omega = \frac{1}{R} * u$$

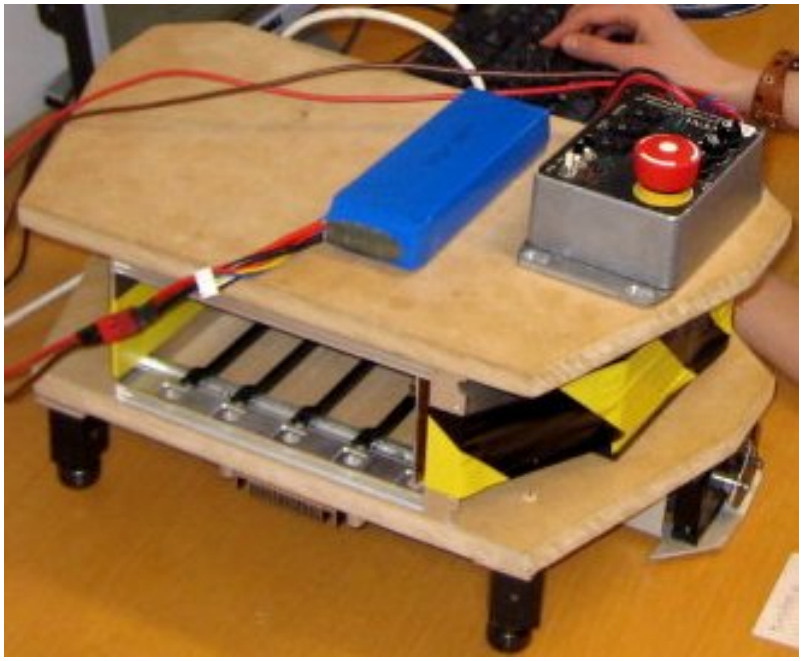
hvor ω er vinkelhastighet og u er translasjonshastighet. Dette ble implementert i Regulator ved at ønsket rotasjonspådrag adderes med en konstant ganger translasjonspådraget.

4.3.2 Bakke

Den største utfordringen på årets spillebrett var bakken. For å kunne plukke de mest poenggivende spillelementene; appelsinene, måtte roboten opp en lite bakke. På bakgrunn av dette ble fremdriftssystemet spesielt utformet og rotasjonsaksen var plassert langt bak på roboten [17]. Det ble laget en appelsin-plukke-modul og testplattformen ble forsøkt kjørt opp bakken. Det viste seg å være en utfordring da det var lite som skulle til for å gjøre systemet ustabilt. Det ville vært mulig med mer tuning av systemet, men den faktiske roboten fikk tyngdepunket såpass høyt og så langt bak at det var fare for at hele roboten tippet bakover ved bakkekjøring. På grunnlag av dette ble henting av appelsiner nedprioritert.

Kapittel 5

Test og testresultater



Figur 5.1: Testplattform

Da testplattformen kom på plass, var det tid for testing. Denne testfasen foregikk stort sett fra den stasjonære PC'en på kontoret. Testplattfomen kan sees i figur 5.1 og består av bunnplate med fremdriftssystem, odometrihjul og motorkort for pådrag til motorene. Det er satt inn et sensorkort som tar imot odometrimålinger og sender dem til PC'en via CAN. Bryterpanelet er for å enkelt kunne bryte strømmen med nødstopp-bryteren.

5.1 Testing på testplattform

5.1.1 Delsystemer

Det første som ble gjort ved testingen var at retningen på odometrihjul og motorer ble funnet. Det var ikke gitt at de var plassert i intuitiv retning.

	Venstre	Høyre
motor	forover	bakover
odometri	bakover	forover

Tabell 5.1: Retning på odometrihjul og motorer

Dette ble enkelt implementert med forskjellig fortegn ved mottak av odometridata og før sending av motorreferanse.

Da input og output til systemet var riktig, ble estimatoren testet for feil med vinkler og koordinatsystemer. Eventuelle feil ble rettet opp i, men nøyaktigheten ble ikke testet før sammensetting av den roboten som skulle brukes, se kapittel 5.2.

Regulatoren bestod av to separate pid-regulatorer og det var naturlig å dele opp testen i to deler; en en translasjonsdel og en rotasjonsdel. For å teste og justere translasjonsregulatoren ble rotasjonsregulatoren koblet fra og roboten kjørte en avstand rett frem. Dette gjorde så tuning av regulatorparametre var enklere. Det samme ble gjort med rotasjonsregulatoren. Da begge regulatorene var tilstrekkelig tunede, ble de testet sammen.

Det ble laget et loggingssystem til gateway av Ole Lillevik i [8] som logget alle relevante data under testing. Loggingsystemet viste seg å være veldig nyttig og ble flittig brukt. Logdata ble plottet i matlab.

5.1.2 Testplan

Etter at enkelt-regulatorene var tilstrekkelig tunet, var det på tide å begynne og slå dem sammen. Testplanen ble derfor:

1. Translasjonsbevegelse. Kjør rett frem til waypoint en meter unna med begge regulatorer. Waypointet er ikke retningsbestemt.

2. Rotasjonsbevegelse. Roter 180° med begge regulatorer aktive.

3. Rotasjonsbevegelse. Roter 180° med bare rotasjonsregulator.
4. Rotasjon + translasjon. Plasser waypoint bak robot, 50 cm unna. Først ren rotasjon, så kjøring rett frem.
5. Translasjon + rotasjon. Plasser et retningsbestemt waypoint 50 cm foran robot med retning 90° mot høyre (og senere venstre) i forhold til robotens start-posisjon.
6. Kjør gjennom en waypoints-liste.

5.1.3 Testresultater

Det var ikke så enkelt å måle nøyaktigheten til systemet på testroboten, men det var heller ikke meningen, da den nye roboten ville komme til å ha andre størrelser og mål. Det viktigste var at systemet skulle klare seg gjennom en liste med waypoints. Det som avgjorde om roboten traff waypointet var om den kom seg innenfor *circle-of-acceptance* til det gitte waypointet. I denne testen ble *circle-of-acceptance* satt til å være 3 cm. Resultatene er oppgitt i tabell 5.2.

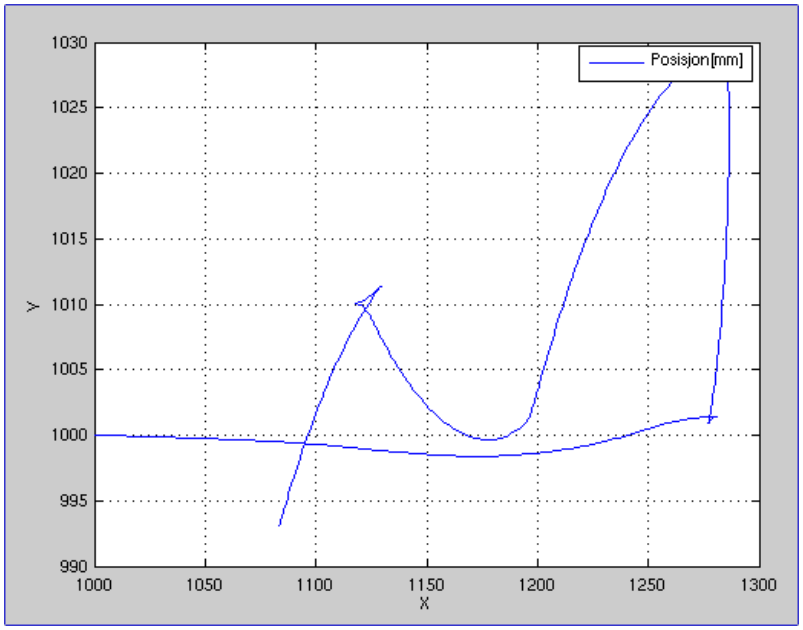
Test	Status	Kommentar
1	OK	
2	nei	Bør kunne skru av translasjonsregulator ved rotasjon
3	OK	Fungerte bedre enn test 2, roterte 180° på ca 2 cm
4	OK	
5	OK	Fjernet feil sluttpunkt til en vinkel på grunnlag av denne testen
6	OK	En film av denne delen av testen ligger i tillegg B

Tabell 5.2: Testresultater med testplattform

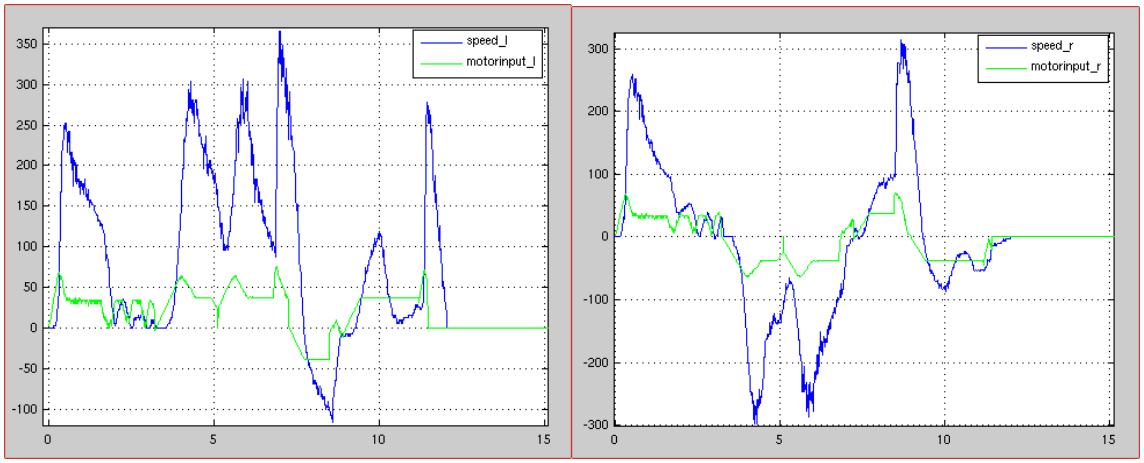
Det nevnes i [15] at tidligere navigasjonssystem har hatt en hastighetsbegrensning. En hastighetsbegrensning kan finnes også i årets system, men vil mest sansynlig ikke nås på spillebrettet da hindringene er mange. Kjøring til waypoint lenger enn en meter unna vil sjelden eller aldri skje.

En film av den siste testen ligger i tillegg B. Et XY-plot av denne gjennomkjøringen kan sees i figur 5.2. Pådrag og hastighet til hver av motorene er plottet i figur 5.3. Roboten starter i posisjon $(1000, 1000)$, har første waypoint i $(1300, 1000)$ og siste waypoint i $(1100, 1000)$. Sluttposisjonen har et retningsbestemt waypoint som er 90° på startposisjonen. Man kan se at roboten ikke er helt tunet inn enda, men

den havner på riktig posisjon med rett vinkel. Y-aksen på plottet er omvendt i forhold til på brettet. I denne filmen blir GUI-visualiseringsverktøyet som Kai Olav Ellefsen utviklet i [2] presentert. Dette var et svært nyttig verktøy som ble brukt av undertegnede til å se om roboten klarte å komme seg til riktig waypoint. Fordelen med visualisering kom spesielt godt frem ved sammenslåing av AI og navsys.



Figur 5.2: Plot av posisjon ved waypointkjøring. Se film i tillegg B.

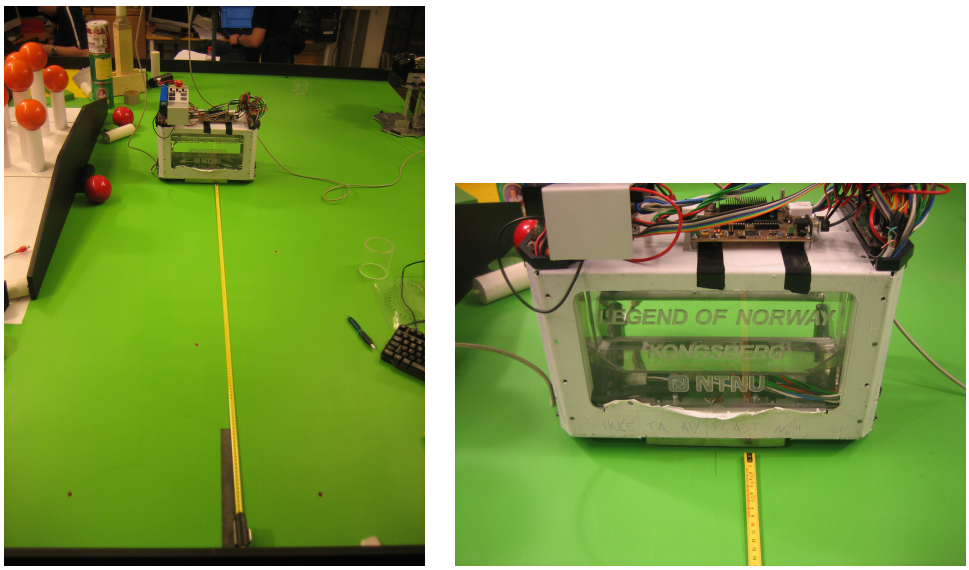


Figur 5.3: Pådrag og hastighet til hver av motorene ved test 6.

5.2 Testing av robot

5.2.1 Delsystemtest på ferdig robot

Odometri



Figur 5.4: Illustrasjon av testing av odometri

Det første som ble gjort var å finne riktige dimensjoner. Målingene fra odometri-løpehjulene er i hovedsak avhengig av to fysiske avstander for at de skal gi nøyaktige posisjonsendringer. Dette er diameter på løpehjulene og avstand mellom dem. De ble målt så nøyaktig som mulig, men det krevdes testing for å få tilstrekkelig nøyaktige målinger. Den første testen bestod i å kjøre 1,5 meter rett frem for å finne nøyaktig diameter på odometrihjulene. Den gamle diameteren ble målt til å være 51,4 mm. Testen ble gjort med pådrag til motorene. Avstand kjørt ble målt opp for hånd etter hver kjøring.

Gjennomsnittet av forholdstallet blir 1,002997. Den nye diameteren ble derfor utregnet til å være $D_{ny} = D_{gammel} \times forholdstall = 51,4 \times 1,002997 = 51,5540\,mm$.

Testen ble gjort på nytt etter at hjuldiameteren var forandret. Dette resulterte i målinger i tabell 5.4. Disse resultatene viser at den utregnede hjuldiameteren på 51,554 mm er tilfredsstillende.

Avstand mellom odometrihjulene ble testet på tilsvarende måte. Roboten ble satt til å rotere ca to runder. Målt vinkel ble gjort med gradskive. Systemet ble stoppet av

Måling [mm]	Odometri [mm]	Forholdstall (måling/odometri)
1534	1529	1,00327
1506	1502	1,00266
1516	1512	1,00265
1529	1524	1,00328
1520	1516	1,00264
1522	1517	1,00330

Tabell 5.3: Test av odometri for å finne riktig hjuldiameter, kjøring rett frem i 150 cm, målt hjuldiameter på 51,4 mm

Måling [mm]	Odometri [mm]	Forholdstall (måling/odometri)
1531	1531	1,00
1519	1519	1,00
1517	1516	1,00067
1512	1512	1,00
1519	1520	0,99934
1524	1525	0,99934

Tabell 5.4: Test av odometri med utregnet hjuldiameter på 51,554 mm, kjøring rett frem i 150 cm

oss, det er derfor lengde kjørt er såpass forskjellig. Det er riktignok forskjellen mellom målt vinkel og beregnet vinkel som er viktig også her. Avstand mellom hjulene var i utgangspunktet målt til å være 338 mm. Dette viste seg å være for mye, og det ble kjørt noen runder før det viste seg at en odometrihjul-avstand på 335 mm fungerte bra. Dette kan sees i tabell 5.5

Måling [radianer]	Odometri [radianer]	Forholdstall (måling/odometri)
18,9892	18,9996	0,99945
13,5612	13,6759	0,99161
16,1966	16,0840	1,00700
14,3433	14,3501	0,99953
12,0567	12,0690	0,99898
13,9234	13,8412	1,00594

Tabell 5.5: Test av odometri for å finne riktig avstand mellom hjulene, rotasjon, målt avstand mellom hjulene: 335 mm

Disse to testene ble gjort for å bestemme fysiske konstanter til odometrisystemet.

Det er derfor ikke meningen å bruke resultatene til å si noe om nøyaktigheten til odometrisystemet. Det er flere feilkilder til denne testen:

- Unøyaktigheter ved oppstilling av roboten ved start
- Målefeil ved måling av endt posisjon
- Avrundingsfeil

Det er allikevel tilstrekkelig for sitt formål.

5.2.2 Test av regulator

Grunnleggende tester ble gått igjennom også her for å forsikre seg om at ikke det nye systemet oppførte seg helt forskjellig fra testplattformen. Det ble deretter testet med en liste av waypoints.

Testplan

1. Kjøring rett frem
2. Ren rotasjon
3. Kjøring rett frem til retningsbestemt waypoint
4. Rygge
5. Kjør i firkant etter en waypointliste
6. Følg en sirkelbane

Resultater

Translasjon og rotasjon ble gjort med en presisjon på under to cm. To cm er mye tatt i betraktning at roboten er så stor som den er og at spillelementene står såpass tett på brettet. Det ble allikevel ansett som et godt mellomresultat da det begynte å haste med sammensying mellom AI og navsys. Resultatene er oppgitt i tabell 5.6.

Den nye roboten viste seg å være enklere å ha med å gjøre enn testplattformen. Støttehjulene tok ikke så hardt, og friksjonen mot bakken ble derfor mindre.

Roboten ble satt til å kjøre i en firkant etter waypoints for å verifisere at den kan kjøre igjennom en waypointliste og for å se på nøyaktigheten til systemet. Plot fra

Test	Status	Kommentar
1	OK	Hadde litt oversving nærme waypoint.
2	OK	Det ble registrert at hjulene ikke tok helt likt.
3	OK	Treffer nøyaktig vinkel, feilmargin på 0,1 radianer
4	OK	Merkbart annen oppførsel ved rygging
5	OK	Se plot. Brukt til å verifisere estimatorens nøyaktighet
6	nei	Bør benytte en hastighetsreferanse.

Tabell 5.6: Testresultater med roboten

denne testen kan sees i figurer 5.5 - 5.7. Utfra XY-plottene i figur 5.5 kan man se at startposisjon og sluttposisjon ligger innenfor en radius på 10 mm. Faktisk start- og sluttposisjon viste seg å være vanskelig å måle nøyaktig og det ble beregnet en feilmargin på ca 3 mm. Sluttposisjon ble ved alle testene målt til å være mindre enn 8 mm unna startpunkt. Det kan på grunnlag av dette ikke bli påstått å være mer nøyaktig enn 6 mm etter en slik firkantkjøring.

Pådrag til hver av motorene og hastighet fra odometrien fra testen i figur 5.6 kan sees i figur 5.7. Man kan se at høyre motor har hatt positivt pådrag hele veien og derfor hatt yttersving i firkanten. Venstre motor har hatt negativt pådrag ved rotasjon. Det kan observeres også her at y-aksen i plottet er speilvendt i forholdt til faktisk koordinatsystem på bordet.

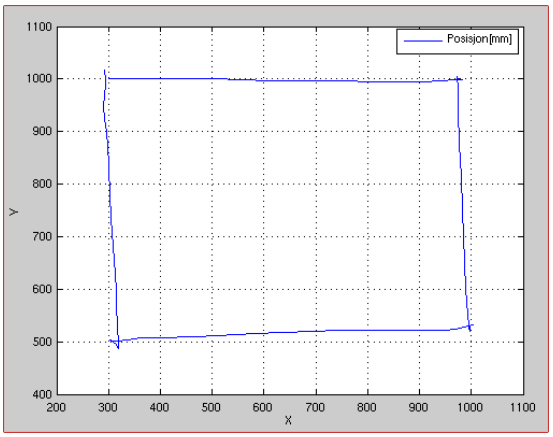
Kjøring langs sirkelbane var ønskelig da årets spillebrett hadde mange hindringer. Testen ble gjennomført ved at roboten skulle kjøre fra et startpunkt i $(300, 1000)$ rett frem til $(500, 1000)$ og kjøre i sirkelbane i 180° til $(500, 1540)$. Resultatet i form at et XY-plot kan sees i figur 5.8.

Det kan sees at dette ikke er optimalt resultat, men det er på grunnlag av knapp tid og nedprioritering av sirkelkjøring at dette var det beste resultatet som ble oppnådd. Det vil aldri forekomme at roboten skal holde sirkelen såpass lenge under faktisk kjøring på brettet. Pådrag og hastighet kan sees i figur 5.9.

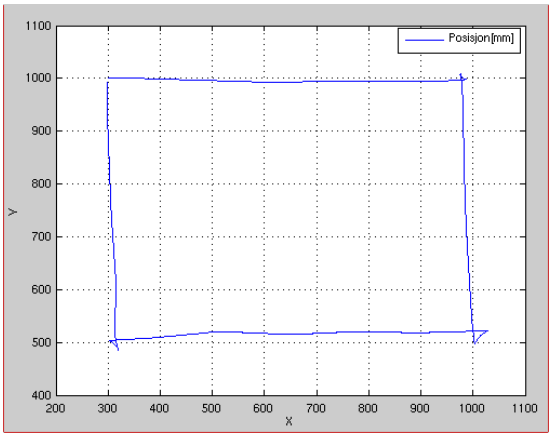
5.2.3 Hele systemet

Sammenslåing mellom AI og navsys på den nesten ferdigstilte roboten var avsluttende del av samkjøringen. Navsys og gateway var blitt testet underveis og AI og datasyn hadde hatt sine tester.

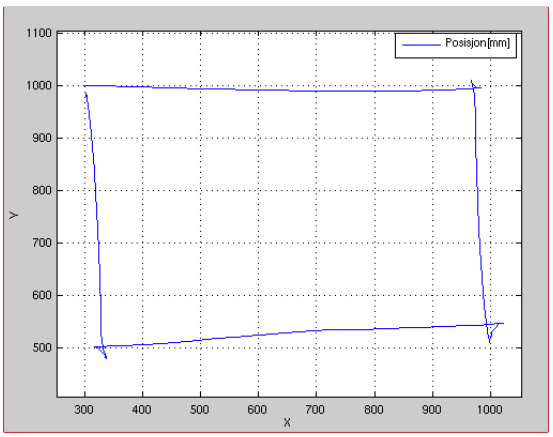
Den store forskjellen med å teste systemet sammen med AI var at waypoints ble



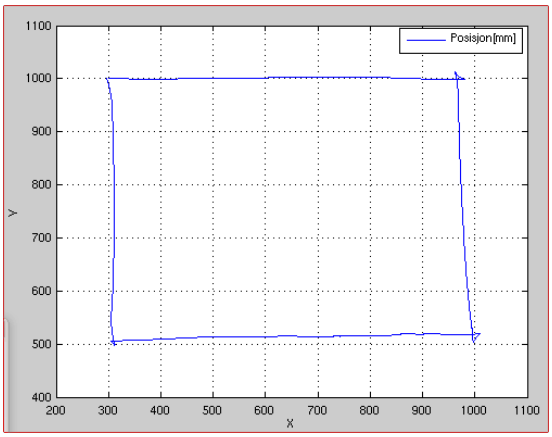
(a) Firkanttest nr 1



(b) Firkanttest nr 2

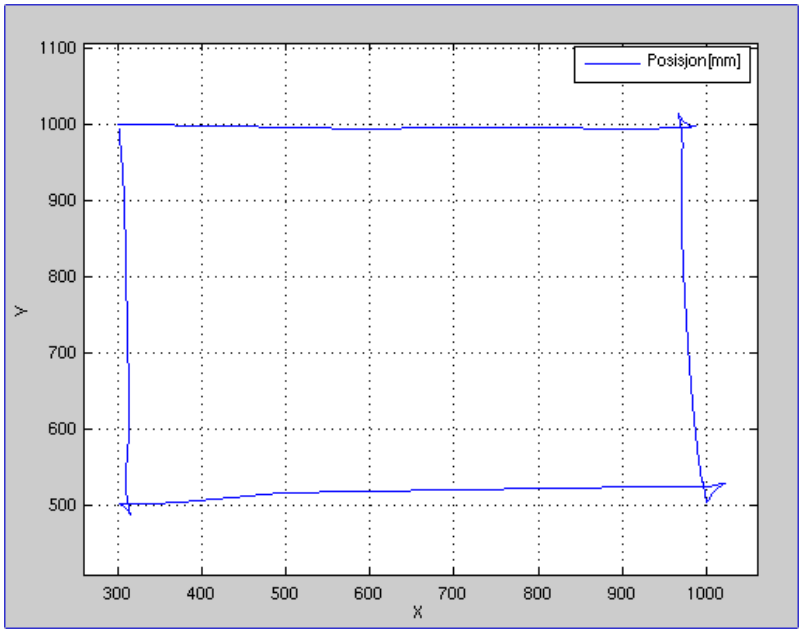


(c) Firkanttest nr 3

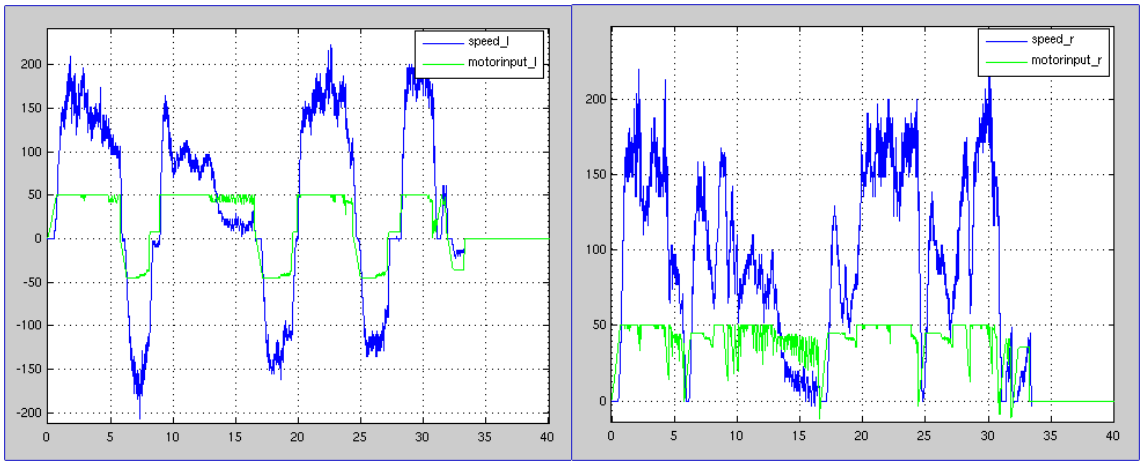


(d) Firkanttest nr 4

Figur 5.5: Testkjøring i firkant

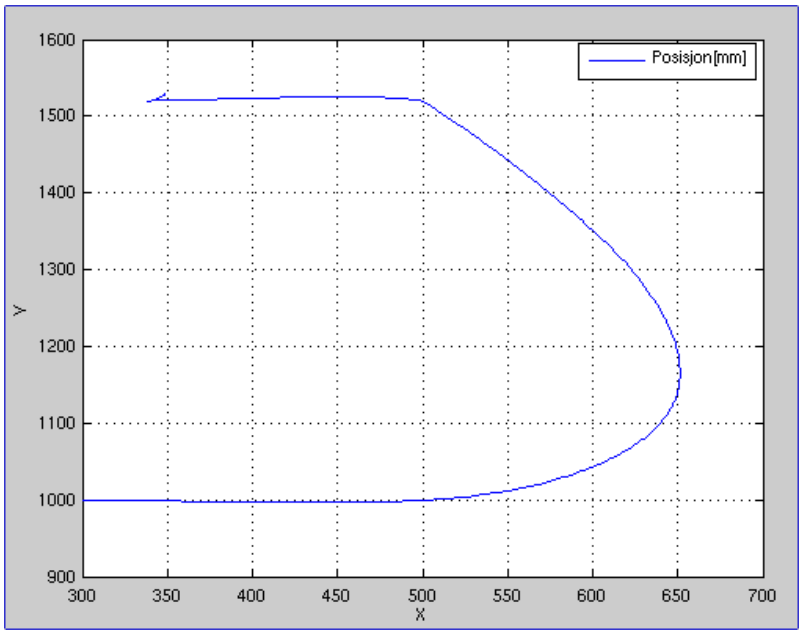


Figur 5.6: Plot av posisjon ved kjøring i en firkant

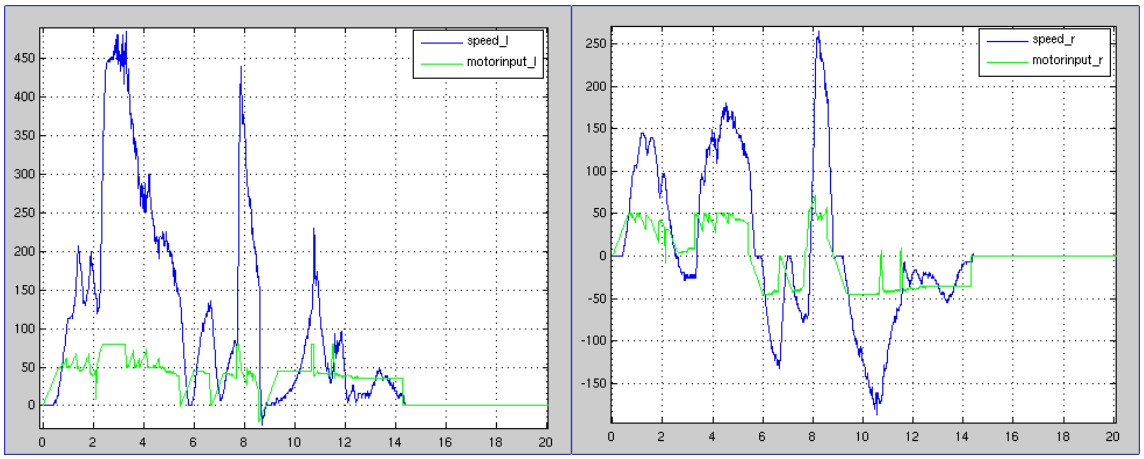


(a) Pådrag og hastighet, venstre (b) Pådrag og hastighet, høyre

Figur 5.7: Pådrag og hastighet fra testkjøring i firkant



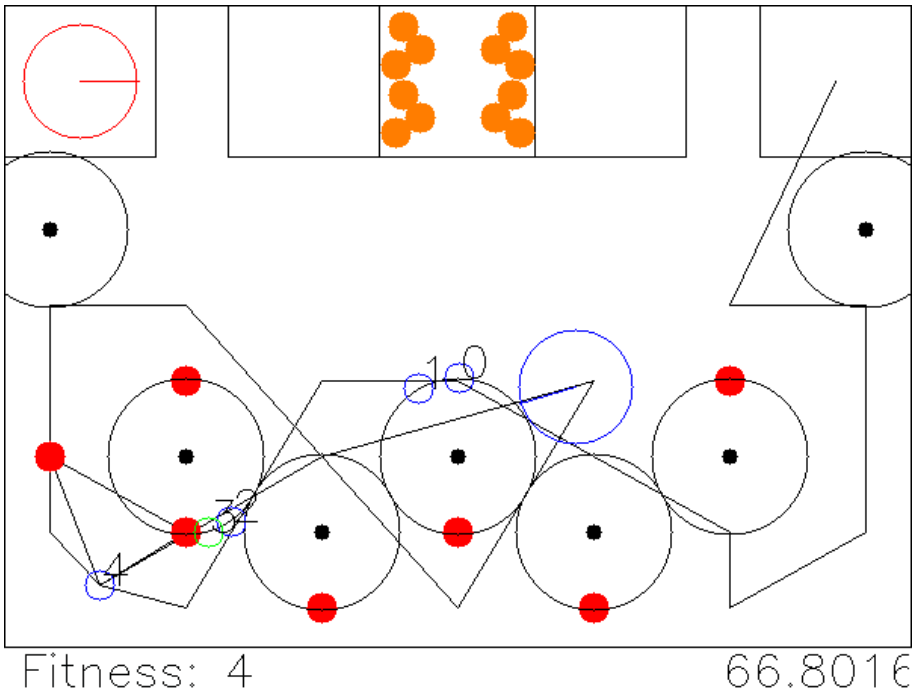
Figur 5.8: Plot av posisjon ved kjøring på sirkelbane



(a) Pådrag og hastighet, venstre (b) Pådrag og hastighet, høyre

Figur 5.9: Pådrag og hastighet fra sirkelkjøringstest

oppdatert ofte og *waypointReached* måtte sendes tilbake når et waypoint var nådd. To systemer ga flere mulige feilkilder, og utskrifter og plotting av testene var viktig. Kai Olav Ellefsen sitt visualiseringssystem fra [2] ga stor forståelse av hvordan roboten oppførte seg og hvorfor. En figur av visualiseringssystemet kan sees i figur 5.10, hvor de små blå sirklene viser waypoins som ligger i waypointlisten.



Figur 5.10: Illustrasjon av GUI med visning av waypoints som sendes til navsys

Testplan

- Kommunikasjon: Uten pådrag
  - Sending og mottak av waypoints
  - Sending og mottak av posisjon og motstanderposisjon
  - Sending og mottak av *waypointReached*
- Waypoints: Kjøring etter waypoints på brettet uten spillelementer
  - Testing av kommunikasjon med pådrag
  - Test av spesielle waypoints: stopp, rygging, sirkelkjøring
- Milepæltest: Levering av poeng. Noen elementer satt ut på brettet
- Kjøring i 90 sekunder som i en kamp
- Kjøring med fullt oppsatt spillebrett
- Testing i Sveits

Resultater

Resultater fra de to første testene er organisert i tabell 5.7 og 5.8. Kommunikasjonen fungerte og kjøring til enkelt-waypoints på brettet var tilfredsstillende.

Sending og mottak av:	Status	Kommentar
Waypoints	OK	Fremdriftsmodul koblet fra
Robotens posisjon	OK	
Waypoint reached	OK	
Motstanderposisjon	OK	
Med pådrag	OK	Test av overstående med fremdriftl tilkoblet

Tabell 5.7: Samkjøringstest med navsys og AI

Waypoints:	Status	Kommentar
Stopp	OK	
Rygging	OK	
Sirkelkjøring	OK	Delvis
Bakke	nei	Robotdesign

Tabell 5.8: Test av spesielle waypoints

Milepæltesten "levering av poeng" ble gjort med mais på bordet. Roboten kjørte over tre mais og kjørte til leveringsområdet hvor mais-leveringsmodulen sørget for at to av maisene datt ned i målkurven.

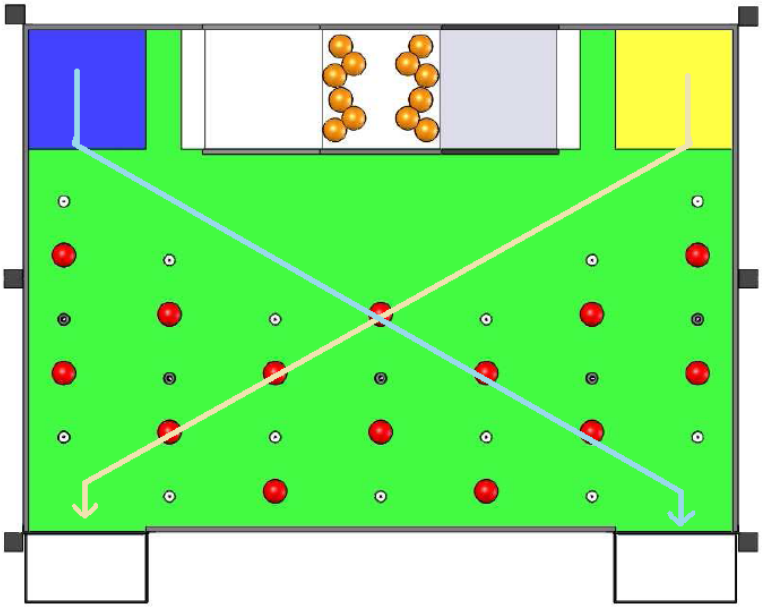
På samme måte som roboten kjørte da den leverte for første gang, skulle den kjøre på brettet og plukke mais. Testen var lagt opp til å vare i 90 sekunder, som en ordentlig eurobot-kamp. Det meldte seg et behov for absolutt posisjonering da roboten kjørte seg fast mellom en hvit mais og veggen, og odometrihjulene forskjøv seg. Ved gjentatte tester av samme slaget fikk den noen fine gjennomføringer som burde vært filmet. Roboten fikk vanligvis levert to ganger. Det var riktignok ikke alltid roboten stod i riktig posisjon så utskyvningsmekanismen klarte å få alle sylindrene i mål. Roboten må være nøyaktig plassert mot målområdet for at også mais som ligger parallelt med kjøreretningen skal ramle over målstreken.

Kjøring med fullt oppsatt spillebrett viste seg å være en utfordring for roboten som er litt for bred over rumpa. Da roboten i første omgang ikke var tilstrekkelig bra tunet, viste det seg at det var veldig lett å kjøre på sorte hindringer. Slike påkjørsler førte ofte til en innføring av feil i posisjonsestimatet. Roboten ble derfor tunet tilstrekkelig

før samme test skulle gjennomføres på nytt. Dette ble av tidligere nevnte årsaker ikke gjort før avreise til konkurranse i Sveits.

Reisen til Sveits kom litt for tidlig, og systemet var ikke helt oppe og gikk før avreise. Det var riktignok ikke noe å gjøre med, og vi dro nedover med en demontert robot i bagasjen og et lite håp i baklomma. Etter ankomst ble roboten tilslutt montert ferdig, og systemet var klart til testing. Det viste seg at det var mye av testingen som stod igjen. Etter at navsys og AI hadde fått roboten i gang, var det flere feil som oppstod.

Det gikk ikke veldig lang tid før vi innså at vi måtte ha fokus på å komme igjennom homologeringen, det vil si en godkjenningsrunde som roboten må igjennom for å kvalifisere seg til selve konkurransen. Homologeringsprosessen står det mer om i kapittel 2.1. Det ble besluttet at roboten skulle kjøre raskeste vei til målområdet for å levere poeng. Den skulle bare plukke noen få tomater for å få denne delen godkjent. En illustrasjon av denne kjøringen kan sees i figur 5.11. Dette kjøremønsteret var enklere planlagt enn utført. Det ble trangt og ofte kollisjon med de sorte hindringene. Det var heller ikke helt heldig å levere tre baller da de noen ganger truet med å tippe hele roboten bakover ved levering. Det hele toppet seg da roboten levde sitt eget liv da den eksterne PC'en, som ble brukt som skjerm og tastatur, ble koblet fra. Etter mange nederlag innså teamet tilslutt at tiden hadde rent ut.



Figur 5.11: Ønsket kjøring ved homologering

Kapittel 6

Diskusjon

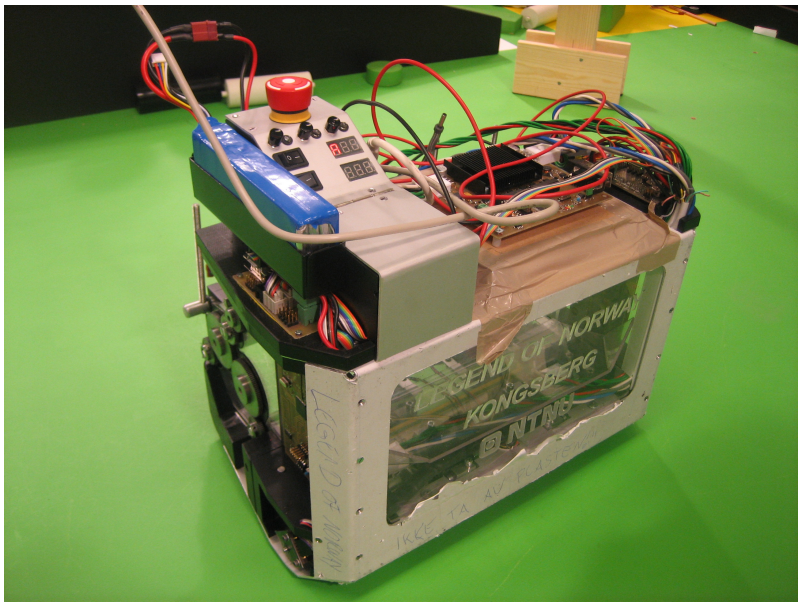
6.1 Testplattform

Testplattformen som vist i figur 5.1 har vært et viktig verktøy i utviklingsprosessen til navsys. Da den endelige roboten ikke ble ferdig før like før vi dro, har systemet blitt testet ut på denne. Det har riktignok ikke vært prioritert høy nøyaktighet da dette ikke var relevant før roboten var på plass. På plottet i figur 5.2 og på filmen som ligger i tillegg B kan man se at roboten takler å kjøre til waypoints lagt i en liste. Regulatoren er enda ikke helt tunet inn, men roboten kommer frem.

Testroboten var et vanskelig system å styre da støttehjulene lagde mye friksjon mot underlaget. Gummien på drivhjulene som ble brukt var heller ikke optimal. Bård Jonas Wigestrand testet friksjonen til både testhjulene og de hjulene som til slutt ble brukt på roboten [17]. Resultatene viste at testhjulene ga veldig god friksjon. Grunnen til at det ble dårligere etter hvert var fordi de bestod av en slags skumgummi, som i tillegg til å gjøre at de satt godt på underlaget, tok til seg veldig mye støv. De begynte etter hvert lett å spinne, og akselerasjonen måtte følgelig begrenses. Det skal nevnes at det fungerte betydelig bedre å kjøre på skrivebordet enn på gulvet. Det var heller ikke noe stort problem på spillebrettet med noe redusert akselerasjon.

6.2 Testing av robot

Den nye roboten viste seg å være enklere å ha med å gjøre enn testplattformen. Bård Jonas Wigestrand har i [17] videreutviklet systemet fra prototypemodellen og



Figur 6.1: Illustrasjon av endelig robot, fortsatt uten PC og kameraer

forbedret blant annet hjul og støttehjul da det viste seg at friksjonen ble et problem. Det var flere ting som etter hvert viste seg å ikke fungere så bra etter at systemet i sin helhet ble testet. Mer om dette i kapittel 6.2.3.

6.2.1 Odometri

Beregnet odometrihjulavstand viste seg å være forskjellig fra målt avstand. Forskjellen var ikke mer enn 3 mm, men det er allikevel grunn til å tro at hjulene ikke stod helt parallelt. En skjevhet i systemet er ikke heldig da det blir innført nok en feilkilde. Odometromålingene ble allikevel ansett som gode nok da firkanttesten tilsa at systemet ikke driftet mer enn 6 mm på en firkantkjøring. Det ble heller ikke observert noe drifting i løpet av en 90 sekunders gjennomføring. Robotens posisjon ble også brukt av datasyn hvor det var viktig at estimert posisjon var tilstrekkelig nøyaktig da det systemet bruker robotens posisjon til å finne andre spillelementer på brettet.

Det har i ettertid blitt oppdaget i [12] at den anbefalte nøyaktighetstesten for odometri er å kjøre i en firkant på 4×4 meter begge veier. Hvis testen bare kjøres en vei, kan feil eliminere hverandre og gi lik start- og sluttposisjon. Til tross for dette kan det argumenteres for at firkanttesten i denne oppgaven også kan brukes for å si noe om nøyaktighet da plottene viser kjøremønsteret.

6.2.2 Regulator

Firkanttest

Den såkalte firkanttesten indikerer at odometri og estimator er tilstrekkelig nøyaktig. Roboten ender tilsynelatende opp få millimeter unna startposisjonen ved hver kjøring. Det kan sees fra figur 5.6 at verken rotasjons- eller translasjonsregulatoren er ordentlig tunet. Tuning var det ikke mye tid til da roboten som ble brukt ofte var delvis demontert da det var flere som måtte jobbe med forskjellige deler samtidig. Tyngden og tyngdepunktet på roboten forandret seg derfor stadig vekk og det ble ikke mange tester med et fullstendig system før vi dro til Sveits.

Sirkelkjøring

Det var ønskelig at roboten skulle kunne kjøre deler av løypen sin langs sirkelformede baner. Teorien bak er relativt enkel, se kapittel 4.3. Testen som ble gjennomført viste at gjennomføring i praksis er mer kompleks. Et XY-plot av testen kan sees i figur 5.8. Andre gjennomføringer av testen viste tilsvarende oppførsel. Det som skjer når svingen begynner, ved $(500, 1000)$, er at friksjonspådraget slår inn. Når regulatoren gir ut et pådrag til en av motorene som ligger under pådraget som overgår friksjonen på bordet, vil pådraget øke. Derfor kjører det hjulet som har innersving fortere enn det skal i første delen av sirkelen. Når roboten kommer tilstrekkelig ut av kurs, retter den seg opp og kjører mer direkte på neste waypoint før den roterer til riktig retning der.

Det er antageligvis mulig å fintune regulatoren og få dette systemet til å kjøre i en god sirkel, men det vil være avhengig av underlaget og vekt på roboten. Hvis roboten hadde hatt en hastighetsreferanse ville det antageligvis vært mer overkommelig å tune, og det ville ikke i like stor grad vært avhengig av underlag og last. Det er derfor en mulighet for at dette hadde vært lettere hvis ikke motorcontrolleren hadde blitt valgt bort.

Løsningen på den vanskelige sirkelkjøringssituasjonen ble i samarbeid med AI å plassere flere waypoints langs sirkelbanen. Da roboten aldri skulle kjøre langs en sirkel i 180°, men ofte kortere strekk, ble dette ansett som et godt nok alternativ. Planlagt kjøremønster kan sees i figur 4.8.

6.2.3 Hele systemet

Etter at roboten viste at den kunne kjøre etter waypoints i en liste, var det klart for samkjøring med AI. Roboten begynte å ta form, og de fleste systemene hadde kommet på plass. Kommunikasjonen mellom AI og navsys ble satt opp vellykket og roboten fant veien til waypoints som var generert av strategien til AI.

Milepæltest

Levering av poeng ble sett på som en milepæltest fordi dette var en nødvendighet for at roboten skulle bli med i konkurransen. Dette ble gjort uten sorte hindringer på brettet for å se hvordan roboten oppførte seg under veis og ved levering. Det ble sett på som en vellykket test da den viste at kommunikasjon mellom navsys og AI fungerte og robotens oppførsel var tilfresstillende i en så tidlig fase av samkjøringstestingen. Denne testen indikerte også at kommunikasjon mellom AI og maisleveringsmodulen fungerte da roboten leverte poeng. Da roboten ikke klarte å levere all maisen ned i kurven, ble det klart at roboten trengte å stå nøyaktig plassert i målgården for at man med sikkerhet skulle få ned alle sylindrene.

En liten utvidelse av leveringstesten var å kjøre i hele 90 sekunder. Det ga lovende resultater da roboten hadde noen fine leveringer av mais. Det som derimot ikke var like heldig var at den hadde noen hendelser hvor den kjørte seg delvis fast mellom en hvit mais og veggen. Dette viste behovet for et absolutt posisjoneringssystem da odometrihjulene forskjøv seg ved sammenstøt. Ole Lillevik har i [8] jobbet med et laserbasert fyrtårn-system som det er lagt opp til å kjøre sammen med navsys. Dette er derimot ikke testet sammen da lasertårnet ikke ble ferdigstilt og -testet i tide.

Testing i Sveits

Da det ble klart at fokus skulle være på homologeringen, konkluderte teamet med at det lønte seg å sette opp ett kjøremønster som skulle følges under kvalifiseringen. Kjøremønsteret som ble valgt kan sees i figur 5.11. Dette kjøremønsteret ble valgt da det tilsynelatende var fornuftig å kjøre korteste vei, og uten flere rotasjoner enn nødvendig.

Det viste seg imidlertid at denne løypen var den smaleste passasjen til målområdet hvis det stod flere sorte hindringer på hver side. Dette hadde de fleste konfigurasjonene lagt opp til. Roboten var såpass bred at for å komme igjennom denne passasjen var det bare noen få millimeters klaring på hver side. Denne typen kjøring var ikke noe

som var planlagt tidligere, og bredden på roboten ble ikke sett i sammenheng med bredden på denne passasjen før underveis i homologeringstestingen. Roboten var i utgangspunktet bredere enn det som antall spillelementer på brettet tilsa at den burde være. Dette kom ekstra godt frem underveis i denne testfasen.

Det burde vært prioritert å kjøre som det var planlagt underveis da roboten hadde fått noen centimeter ekstra slingringsmonn på hver side. Da det var flere andre feil som viste seg underveis i denne testingen, var ikke kjøremønsteret avgjørende for at roboten feilet. Dette var en kombinasjon av mange små feil som var et resultat av for lite testing av hele systemet.

Kapittel 7

Konklusjon

Det har i dette prosjektet blitt designet og implementert et navigasjonssystem til roboten som deltok i Eurobot Open 2010. Designet har i grove trekk blitt basert på tidligere års system, men er i hovedsak redesignet. Overordnet består systemet av en estimator som estimerer posisjon og en regulator som styrer roboten til ønsket posisjon.

Estimatoren estimerer posisjonen relativt nøyaktig. Det har blitt testet ut tilstrekkelig med sensordata fra odometri. Det har vist seg at det kan være ønskelig å utvikle systemet til å ta imot absolutte posisjonsmålinger for bedre nøyaktighet da årets konkurranse ga utfordringer i form av mange spillelementer på brettet. Det er ønskelig å benytte seg av et lasertårn-system utviklet i [8].

Regulatoren er valgt å være mindre kompleks enn tidligere år. Dette har ført til at det er mer oversiktlig og lettere å sette seg inn i. Det ble også besluttet at det ikke skulle være noen motorcontroller innenfor posisjonsregulatoren i navsys. Dette har i ettertid vist seg å ikke fungere optimalt. Blant annet ble ønsket sirkelkjøring ikke tilstrekkelig god da det var vanskelig å holde forholdet mellom translasjonshastighet og vinkelhastighet konstant uten hastighetsreferanse.

Systemet har vist gjennom tester at det fungerer. Roboten er riktignok så stor at det settes veldig høye krav til nøyaktigheten på systemet. Dette viste seg spesielt under testing før homologering i Sveits. Det kan lønne seg å holde størrelsen på roboten godt innenfor maksimale begrensninger for konkurransen.

Det har vært et mål å utvikle oversiktlig og godt dokumentert kode. Dette anses som et nådd mål og dokumentasjonspermene som ble laget i forbindelse med fjordårets prosjekt har blitt holdt oppdatert også etter dette årets prosjekt.

Kapittel 8

Forslag til videre arbeid

Estimatoren har bare benyttet seg av sensordata fra odometri. Testene tilsa at det var stor fare for å kjøre på faste og løse elementer på spillebrettet som flyttet odometrihjulene litt ut av posisjon. Det er derfor ønskelig å benytte seg av absolutte posisjonsmålinger i tillegg til relative posisjonsmålinger fra odometri. Lasertårn-systemet som er utviklet i [8] bør være en naturlig utvidelse av nåværende estimator.

Da posisjonsregulatoren ble kjørt uten motorcontroller ble det mer direkte regulering til motorene. Dette viste seg allikevel å ikke være det mest gunstige da hastighets-referansen forsvant. Denne var svært nyttig blant annet for å gjøre sirkelkjøring mulig. Det bør derfor jobbes videre med posisjonsregulatoren i kombinasjon med en motorcontroller.

Systemet som helhet har vist seg å få problemer på grunn av for lite testing. Det anbefales derfor på det sterkeste å sette realistiske tidsfrister underveis som også blir overholdt. Det bør være større avstandsmargin mellom sidekant på robot og hindringer på brettet. Da dette ikke ble prioritert høyt nok ved konstruksjon av årets robot ble det veldig trangt på spillebrettet.

Kapittel 9

Teamarbeid

9.1 Innledning

Antallet personer på dette prosjektet har i år vært mange fler enn tidligere. Dette har ført til mer organisering og mer samarbeid. Da jeg anser dette som en stor del av læringsutbyttet, er det tatt med en såpass stor del i rapporten.

9.2 Hoveddel

Teamet var i vårsemesteret satt sammen av to fra Kybernetikk; Ole Lillevik og undertegnede, to fra Data; Kai Olav Ellefsen og Kai Hugo Hustoft Endresen, en fra Maskin; Bård Jonas Wigestrand og to Eksperter i Team-grupper. Hver person eller Eit-gruppe har hatt ansvar for hvert sitt delsystem;

1. Bård: Mekanisk ansvar og prosjektleder
2. Kai Olav: Kunstig Intelligens
3. Kai Hugo: Datasyn
4. Ole: Gateway og hardwaredesign
5. Kristin: Navigasjonssystem
6. Eksperter i Team: Plukke- og leveringssystem

Denne fordelingen var en samling personer med forskjellig bakgrunn som utfylte hverandre faglig. Det var en stor utfordring å utnytte og samkjøre all faglig kunnskap

som var fordelt på teammedlemmene. Da alle hadde hvert sitt arbeidsområde var det viktig å ikke glemme fokuset på helheten i prosjektet. De fleste av teammedlemmene hadde ikke mer erfaring med å jobbe i team enn det som er oppnådd under studiene. Mange vil derfor påstå at dette relativt store teamet stod på bar bakke når det gjelder teamarbeid og -organisering. Denne utfordringen var en av hovedgrunnene til at jeg ønsket å være med på dette prosjektet.

Ambisjonsnivået til teamet var det ingenting å si på. Da fjorårets NTNU-lag gjorde det så bra, var det selvfølgelig et mål å gjøre det minst like bra. Da årets team i tillegg stod faglig sterkere med hele fem masterstudenter var det ingen grunn til å være tilbakeholdne. Årets Eurobot fra NTNU skulle gjøre det bedre enn noen gang. Og det med en robot full av nyutviklede, høyteknologiske systemer som fungerte feilfritt sammen. Det ble ikke tatt nevneverdig hensyn til at roboten allerede ved semesterstart var begynt å bli forsinket.

Da systemet har hatt en så markant oppdeling, har det vært naturlig å jobbe mye individuelt i startfasen. Dette ble også forsterket av en litt dårlig geografisk plassering av masterstudentene rundt på campus. Datastudentene satt stort sett hos seg selv, mens Bård Jonas Wigestrand fikk plass sammen med kybstudentene. Individuell jobbing ble supplementert med ukentlige møter for alle masterstudentene og en representant fra hver av Eit-gruppene. Dette var for å holde hverandre oppdatert og lage litt lagfølelse. Den individuelle arbeidsperioden i første del av semesteret gikk over i en samarbeidsperiode mot slutten hvor alle masterstudentene satt relativt samlet. Dette var viktig da det tar tid å samkjøre så store systemer, og få ting til å fungere som ønsket. Kommunikasjonen mellom systemer skulle testes og funksjonene til hele roboten skulle fungere som planlagt.

De to siste ukene før konkurransen begynte man å se at tidslinjen hadde sklidd ut. Det ble derfor tatt noen grep rundt arbeidsmetode, og det ble lagt opp til daglige morgenmøter for å vise fremdrift og for å finne mulige løsninger på problemer som oppstod fra dag til dag. På disse møtene ble tavlen brukt til å sette opp arbeidsplan for dagen. Denne kan sees i figur 9.1

Samarbeidet med de andre teammedlemmene var avgjørende for at jobbingen med helheten av roboten skulle fungere. Det har tilsynelatende vært god kommunikasjon igjennom semesteret. Det har riktignok vist seg i ettertid at kommunikasjonen ikke har vært god nok. Enkelte teammedlemmer har vært mer fokusert på at sitt eget system skal fungere ypperlig mens noen har jobbet hardt for å få systemet til å fungere sammen. Det har vært nevnt at prosjektlederen ikke har hatt den erfaringen som skal til for å lede et så stort prosjekt. Dette har riktignok vært vanskelig å løse

Figur 9.1: Illustrasjon av arbeidsplan fra morgenmøte rett før konkurransen

på noen annen måte da ingen av de andre teammedlemmene heller hadde noen store prosjektledererfaringer.

Læringsutbytte i et sånt prosjekt bør poengteres, da teamarbeid er noe man vokser på. Det ble holdt et oppsummeringsmøte i etterkant av konkurransen for å dele erfaringer og synspunkter. Her ble erfaringer fra hele prosjektet diskutert og skrevet ned. Referatet fra dette møtet kan finnes i [16].

Ved siden av arbeid med selve prosjektet har teamet brukt endel tid på andre aktiviteter for å promotere prosjektet, Kongsberg Gruppen som sponsor og NTNU. Denne listen beskriver hva undertegnede har deltatt på i vårsemesteret:

- Jentedag 8. mars, ca 3 timer
- Kreator 18. mars, ca 6 timer
- Diverse foredrag for klasser eller andre underveis i semesteret, ca 2 timer

9.3 Diskusjon

Da årets Eurobot-team er større enn noen gang har det vært et høyere krav til samarbeid. Fredagsmøtene var noe som ble videreført fra høstsemesteret. Det var

bra å få en oppdatering for hele prosjektet en gang i uka. Problemstillinger og mulige løsninger ble diskutert på disse møtene.

Et ønske fra prosjektleder var ganske tidlig i semesteret en oversikt over hver enkelt sin masteroppgave. En side som gir informasjon om planlagt arbeid, fremgangsmåte og tidslinje. Dette er det vanlig å lage i begynnelsen av masteroppgaven hos noen institutter. En slik oversikt hadde nok vært nyttig for oss da vi hadde fått et større innblikk i andres oppgave og blitt litt presset til å holde vår egen oversikt oppdatert. Grunnen til at det ikke ble laget var fordi nødvendigheten av et sånt samarbeidsverktøy ikke ble sett før etter konkurransen da vi evaluerte prosessen.

Etter konkurransen ble det holdt et oppsummeringsmøte hvor fokuset var på hvordan samarbeidet hadde gått. Erfaringene som er høstet i løpet av dette semesteret er mange. Den mest åpenlyse er at tidsfrister ikke er ment for å brytes. De fleste tidsfrister i løpet av semesteret har blitt utsatt. Når det tidlig blir en kultur for en avslappet holdning til tidsfrister, er det vanskelig å være streng på det senere i prosjektet. Problemer som har utviklet seg fra de nedprioriterte tidsfristene er blant annet for lite tid til testing. Det ble altfor liten tid til å teste det samkjørte systemet før konkurransen. En ting som ble tatt opp i forbindelse med dette var at det hadde vært lurt å ha samkjøringsperioder underveis, for eksempel burde roboten vært testet igjennom en homologeringsløype tidlig i semesteret. Det hadde vært en mulighet å begynne samkjøringen mellom AI og navsys på testplattformen. Grunnen til at dette ikke ble gjort var fordi roboten var "nesten ferdig" i en periode, og undertegnede hadde nok med å teste navsys.

Det ble reist spørsmål om prosjektlederen vår. Han har vært flink til å inkludere og sette i gang teamet, men det har båret preg av at han er uerfaren. Det har nok vært litt mye ansvar som har vært lagt på en person. Mengden ansvar og arbeid det kom til å bli for en prosjektleder var vanskelig å se før et godt stykke ut i prosjektet. Da Bård Jonas Wigestrand tok på seg prosjektlederansvaret var det uten noe videre tidligere erfaringer. Dette var teamet klar over, og det burde kanskje vært tatt en diskusjon på dette da problemene så smått begynte å vises.

9.4 Konklusjon

Det første man sitter igjen med etter at en konkurranse feiler på den måten det gjorde, er at man har gjort det meste feil. Det har vært mye som har blitt gjort på feil måte, men det er også gjort mye riktig. Fredagsmøtene bør trekkes frem, sammen med morgenmøtene i innspurten. Oppsummeringsmøtet etter konkurransen gjorde

så vi fikk reflektert over ting vi har lært og ting som bør videreføres. Det ble diskutert et ønske om bedre veiledning i forhold til prosjektgjennomføring. Dette kan være noe som bør diskuteres før neste års prosjekt da det nye teamet har et utgangspunkt tilsvarende det vårt team hadde ved prosjektstart.

Bibliografi

[1] James L. Crowley. Mathematical Foundations of Navigation and Perception For an Autonomous Mobile Robot. Technical report, University of Amsterdam, 1995. http://www-prima.inrialpes.fr/jlc/papers/NavFoundations.pdf.

[2] Kai Olav Ellefsen. Dynamic Planning for Autonomous Robotics. Master's thesis, NTNU, 2010.

[3] Eurobot Association. Eurobot 2010 - Feed the World. `www.eurobot.org`.

[4] Thor I. Fossen. *Marine Control Systems.* Tapir Trykk, 2002.

[5] Jens G. Balchen, Trond Andresen og Bjarne A. Foss. *Reguleringsteknikk.* Institutt for Teknisk Kybernetikk, NTNU, 2003.

[6] Jorge Nocedal og Stephen J. Wright. *Numerical Optimization.* Springer, Second edition, 2006.

[7] Kristian Muri Knausgård. Eurobot 2007 Navigasjon. Master's thesis, NTNU, 2007.

[8] Ole Lillevik. Eurobot 2010. Master's thesis, NTNU, 2010.

[9] Mark W. Spong, Seth Hutchinson og M. Vidyasagar. *Robot Modeling and Control.* John Wiley & Sons, 2006.

[10] Odd Erik Mørkrid og Kristian Ivar Øien. Eurobot 2008. Master's thesis, NTNU, 2008.

[11] Gunnar Kjemphol og Kristian M. Knausgård. Eurobot 2007. Technical report, 2006.

[12] Johann Borenstein og Liqiang Feng. Measurement and Correction of Systematic Odometry Errors in Mobile Robots. Technical report, University of Michigan, 1996.

[13] Christian W. Kjølseth og Øystein Wergeland. Eurobot 2009. Technical report, Januar 2009.

[14] Christian W. Kjølseth og Øystein Wergeland. Eurobot 2009. Master's thesis, NTNU, 2009.

[15] Stian Søvik. Eurobot 2010 Fremdriftsystem. Technical report, 2010.

[16] Bård Jonas Wigestrand. Eurobot 2010 - The Project. Technical report, 2010. Prosjektlederrapport.

[17] Bård Jonas Wigestrand. Utvikling og bygging av robot for deltakelse i Eurobot 2010-konkurransen. Master's thesis, NTNU, 2010.

[18] Øystein Wergeland. *Softwareperm*, 2009.

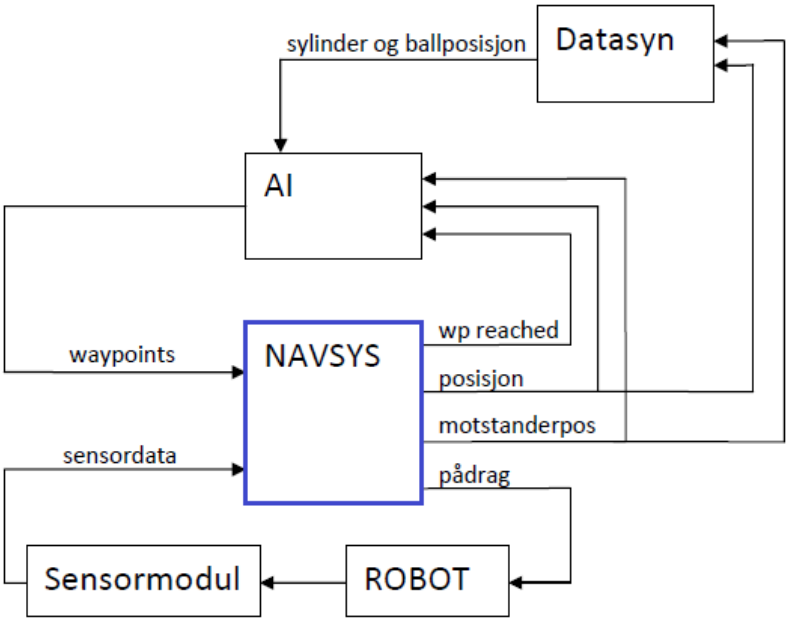
Tillegg A

Dokumentasjon: Navsys

Navsys tar seg av navigering på spillebrettet. Det er her pådrag til motorene blir satt, og posisjonen estimeres. Navsys er i år helt redesignet.

A.1 Meldinger

Navsys kommuniserer med både AI, datasyn og gateway. Meldinger som blir sendt kan sees i figur A.1. Pådrag blir sendt som pwm-signal gjennom gateway til motorene. Sensordata, i form av odometrimålinger, mottas fra sensormodulen gjennom gateway. Odometrimålingene oppdateres ca 100 ganger i sekundet. Navsys sender posisjon og motstanderposisjon til datasyn. Dette blir sendt med en gang det er beregnet da det er viktig for datasyn å motta posisjon så nøyaktig som mulig. Utover dette er det ingen kommunikasjon direkte mellom navsys og datasyn. Navsys og AI sender flere meldinger med hverandre. Posisjon og motstanderposisjon sendes fra navsys til AI ved hver iterasjon av while-løkken i main-metoden. Når roboten har navigert seg innenfor *circle-of-acceptance* til ønsket waypoint sender navsys *waypointReached* til AI. Når AI har mottatt *waypointReached*, eller når det har vært nødvendig å oppdatere strategien, vil det sendes nye waypoints til navsys. Waypoint-listen som sendes består av to waypoints. Det er alltid element 0 i denne listen som er ønsket waypoint.



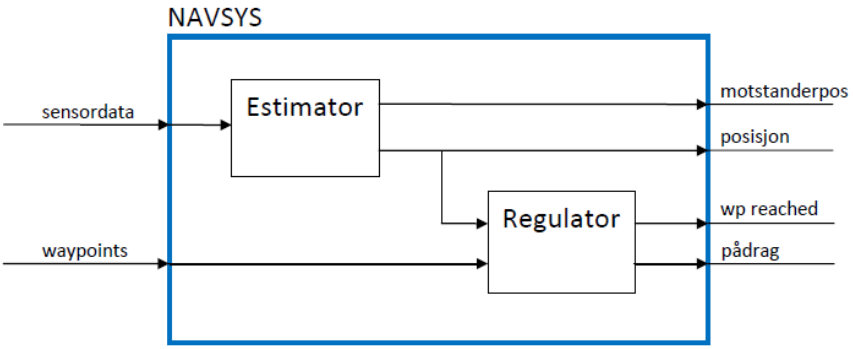
Figur A.1: Kommunikasjon mellom modulene

A.2 Oppbygning

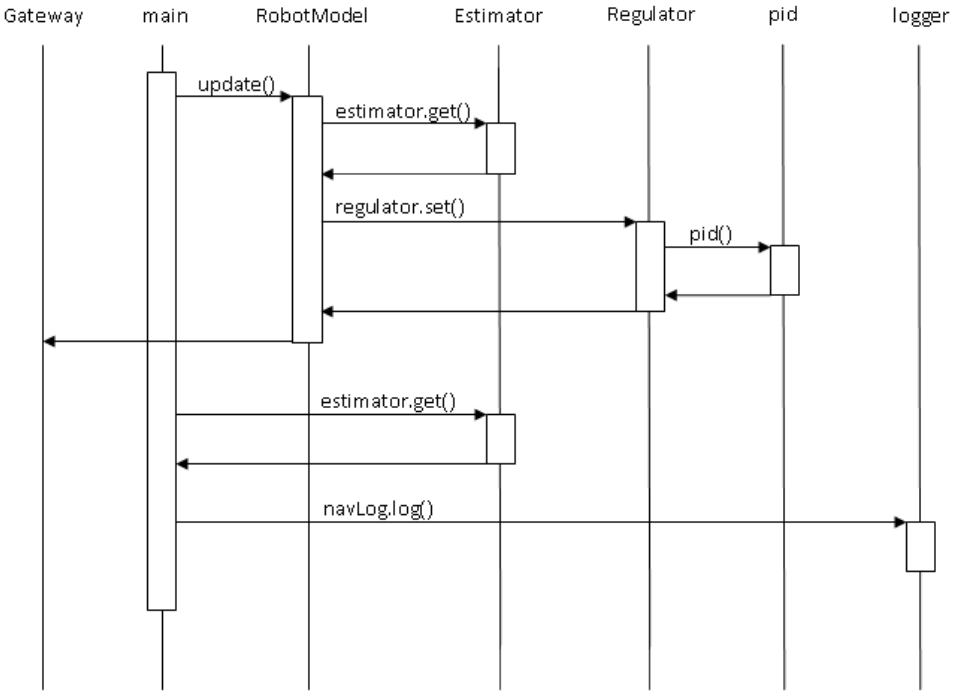
En oversikt over hvordan systemet er oppbygget kan sees i figur A.2. Estimatoren behandler sensordata og estimerer posisjon og motstanderposisjon, mens regulatoren minimerer avstandsfeilen mellom ønsket posisjon og estimert posisjon.

Sekvensdiagram for regulatorsløyfen kan sees i figur A.3. I main-metoden, som ligger i main.cpp, kjøres det initialiseringsmetoder før den setter igang RobotModel. RobotModel henter posisjonsestimat fra Estimator og kjører regulatoren. Dette skjer omtrent 100 ganger i sekundet. Main-metoden starter også logger, som er en klasse som tilhører gateway som logger ønskede data. Loggene har blitt brukt til å lage plot i matlab. Dette har vært svært nyttig for debugging.

Estimatoren kjøres fra kommunikasjonstråden da det er viktig at denne blir oppdatert etter hver gang gateway sender nye målinger.



Figur A.2: Struktur i navsys-modulen



Figur A.3: Sekvensdiagram

A.3 Oversikt over kildekode

main.cpp/.h

Main-metoden kjører initialiseringsmetoder og venter til det er mottatt startposisjon fra AI. Den kjører en while-løkke som kjører updateModel() i RobotModel og setter igang logger. Posisjon og motstanderposisjon sendes til AI herfra.

communication.cpp/.h

Communication.cpp kjører en egen tråd som mottar alle meldinger. De mottatte meldingene settes inn i ønskede objekter. Set()-metoden til Estimator kjører når det er mottatt odometriinformasjon. Da sendes også estimert posisjon til datasyn. Motstanderposisjon sendes til datasyn etter at det er oppdatert. Alle metodene som brukes i programmet er opprettet i denne filen.

RobotModel.cpp/.h

I RobotModel ligger metoden updateModel() som er metoden som kjører Regulator og som sender pådrag til motorkortet. Pådraget fra Regulator er i form av translasjon og rotasjon. Dette blir omregnet til pådrag på venstre og høyre motor. Begrensninger på pådraget i tillegg til å øke pådraget hvis friksjonen er for høy skjer her før det sendes til motorkort.

Estimator.cpp/.h

Set()-metoden i Estimator-klassen kjøres hver gang odometridata blir oppdatert. Den tar inn odometridata som en struct og regner om dette til antall step kjørt siden forrige måling. Dette blir brukt til å berenge robotens rotasjon og lengde kjørt siden forrige måling. Robotens estimerte posisjon blir beregnet fra robotens posisjon ved forrige tidsskritt og forandring fra odometrimålingene.

Regulator.cpp/.h

Denne klassen holder oversikt over neste waypoint og pådrag. Set()-metoden kjører først next_waypoint() som regner om neste waypoint i *local*-koordinater til *body*-koordinater så avstand og vinkel til neste waypoint kan brukes videre. Så sjekker

set()-metoden om ønsket waypoint er nådd. Den tar hensyn til om waypointet er retningsbestemt, om det skal kjøres i sirkelbane eller om det skal rygges til. Hvis waypointet er nådd, sendes waypointReached() til AI. Hvis waypointet ikke er nådd kjøres pid(). Pid() er en metode i Regulator som oppretter to pid-objekter og kjører run()-metoden i pid.cpp på disse.

pid.cpp/.h

Pid er en klasse som består av en enkel regulator med proporsjonal, integral og derivatvirkning. Den har også noen begrensninger på output som kan settes i pid()-metoden i Regulator.

posix_messages.h

Inneholder *defines* til bruk i meldinger.

navsys_structs.h

Inneholder structer som brukes i navsys.

communication_structs.h

Inneholder strukter som blir brukt i kommunikasjon mellom systemer.

messageQueue.cpp/.h

Inneholder funksjoner for mottak av posix-meldinger.

Tillegg B

CD

Vedlagte CD inneholder kildekode til navigasjonssystemet, en film av testroboten og dokumentasjon. Strukturen er:

- Programmvare
  - Kode
  - Dokumentasjon
- Film