NTNU
Det skapende universitet

Eurobot 2008

Odd Erik Mørkrid
Kristian Ivar Øien

Master i teknisk kybernetikk
Oppgaven levert: Juni 2008
Hovedveileder: Sverre Hendseth, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Oppgavetekst

Det skal designes og bygges en robot som skal delta i Eurobot Open 2008 i
Heidelberg, 21. til 25. mai. Dette innebærer:
- Videreutvikling av et absolutt posisjoneringssystem for å rette opp feilen som oppstår ved bruk av odometri.
- Utvidelse av navigasjonssystemet, i den grad det er nødvendig, for å kunne manøvrere best mulig under konkurransen.
- Implementering av AI på grunnlag av den teoretiske utledningen i prosjektet, samt lage strategier for årets konkurranse.
- Forbedring av antikollisjonssystemet; software og hardware, slik at roboten kan unngå kollisjoner med en annen robot uten å bli stående fast.
- Utvikling av software for kamera slik at det er mulig å identifisere og plukke opp baller som ligger tilfeldig rundt på spillebordet.
- Organisering og oppfølging av to EiT-gruppers arbeid med å lage en modul for opplukking og sortering av baller, samt sammenkobling av denne med resten av roboten.
- Ferdigstilling og sammensetting av hele systemet i god tid før avreise, slik at roboten kan gjennomgå omfattende testing i et miljø lignende det
man møter i konkurransen.
I tillegg skal det fokuseres på å tilrettelegge for neste års deltagelse ved å lage moduler som kan brukes videre, dokumentere og samle kildekode etc. på ett sted, og videreføre kunnskap direkte til de personene som skal overta.

Oppgaven gitt: 07. januar 2008
Hovedveileder: Sverre Hendseth, ITK

Oppgave

Kandidatenes navn:	Odd Erik Mørkrid og Kristian I. Øien
Fag:	Teknisk kybernetikk
Oppgavens tittel (norsk):	Eurobot 2008

Oppgavens tekst:

Det skal designes og bygges en robot som skal delta i Eurobot Open 2008 i Heidelberg, 21. til 25. mai. Dette innebærer:

- Videreutvikling av et absolutt posisjoneringssystem for å rette opp feilen som oppstår ved bruk av odometri.
- Utvidelse av navigasjonssystemet, i den grad det er nødvendig, for å kunne manøvrere best mulig under konkurransen.
- Implementering av AI på grunnlag av den teoretiske utledningen i prosjektet, samt lage strategier for årets konkurranse.
- Forbedring av antikollisjonssystemet; software og hardware, slik at roboten kan unngå kollisjoner med en annen robot uten å bli stående fast.
- Utvikling av software for kamera slik at det er mulig å identifisere og plukke opp baller som ligger tilfeldig rundt på spillebordet.
- Organisering og oppfølging av to EiT-gruppers arbeid med å lage en modul for opplukking og sortering av baller, samt sammenkobling av denne med resten av roboten.
- Ferdigstilling og sammensetting av hele systemet i god tid før avreise, slik at roboten kan gjennomgå omfattende testing i et miljø lignende det man møter i konkurransen.

I tillegg skal det fokuseres på å tilrettelegge for neste års deltagelse ved å lage moduler som kan brukes videre, dokumentere og samle kildekode etc. på ett sted, og videreføre kunnskap direkte til de personene som skal overta.

Oppgaven gitt: 01.01.2008
Besvarelsen leveres: 11.06.2008

Utført ved Institutt for teknisk kybernetikk
Veileder: Sverre Hendseth

Forord

Denne rapporten er, iløpet av våren 2008, skrevet av:

- Odd Erik Mørkrid
- Kristian Ivar Øien

Det var av praktiske årsaker mest naturlig å levere en felles besvarelse, da vi har samarbeidet om de fleste oppgavene. Prosjektet er i hovedsak finansiert av midler fra vår hovedsponsor, Kongsberg Gruppen ASA.

Vi vil gjerne benytte anledningen til å takke deltagerne i Eksperter i Team for utviklingen av sorteringsmodulen, Lars Vråle for hjelp med kamerakalibrering, komponentverkstedet for hjelp og støtte med komponenter, det mekaniske verkstedet på instituttet for hjelp med mekaniske løsninger, og ikke minst Sverre Hendseth som har veiledet oppgaven.

Odd Erik Mørkrid Kristian Ivar Øien

Trondheim, 10. juni 2008

Sammendrag

Eurobot Open er en internasjonal robotkonkurranse for studenter og uavhengige organisasjoner, som arrangeres i Europa i mai hvert år. Institutt for teknisk kybernetikk har deltatt hvert år siden 2000, gjennom prosjekt- og diplomoppgaver. I 2008 foregikk konkurransen i Heidelberg, Tyskland, under tittelen "Mission to Mars". Oppgaven gikk i korte trekk ut på å plukke opp og samle steinprøver, i form av innebandyballer, i et eget depot.

Dette arbeidet har hatt til hensikt å fullføre roboten ved å utvikle og gjennomføre de systemene som trengs for å delta i konkurrasen. Gjennom prosjektoppgaven, høsten 2007, har undertegnede utviklet en del basisfunksjonalitet på roboten, som det nå er bygget videre på.

Posisjoneringen av roboten er grundig gjennomgått, og det har vært fokus på å utvikle et absolutt posisjoneringssystem basert på triangulering med 3 faste sendere. Systemet er basert på et ferdig konsept, men det har vært jobbet mye med hardware og ny software har blitt utviklet. Til slutt har det hele blitt testet, noe som har vist presise posisjoneringsresultater. Avlesning av vinklene til trianguleringen viste seg imidlertid å ta litt tid, slik at dette må gjøres når roboten står stille på bordet. Pga. tidsbruk og taktikk ble det kun brukt én sender under selve konkurransen.

Navigasjonssystemet har stort sett blitt videreført fra tidligere, men det er lagt til mye ny funksjonalitet som gjør manøvreringen på spillebordet mer fleksibel. Det kan bla. nevnes rygging, hastighetsstyring og avstandsregulering mot kant. Endringene har fungert bra og vist seg svært nyttig i konkurransesammenheng.

Den kunstige intelligensen har blitt basert på en rekke tilgjengelige strategier, der alle har den samme oppbygningen. Fokus har vært på en enkel og strukturert AI der robusthet og repetérbarhet har vært nøkkelordene. Rammeverket med en overordnet styring og fleksible strategier fungerte bra både under testing og konkurranse. Testing av AI ble i utgangspunktet gjort mot en simulator, noe som er mer effektivt enn å teste mot den fysiske roboten.

Antikollisjonssystemet er basert på fjorårets system og en rekke endringer har blitt gjennomført. De viktigste endringene er å ikke benytte Ir til kol-

lisjonsdeteksjon, i tillegg til å legge til antikollisjonslogikk i AI. Tiltak som deaktivering av antikollisjon i definerte soner, og utarbeidingen av en unna-manøvringsalgoritme har gjort systemet mer robust enn tidligere.

Datasynet er videreutviklet med utgangspunkt i fjorårets kode og benytter Hough-transformen til å finne baller. Ballene er sirkler i et bilde. Kameraet klarte fint å gjenkjenne baller foran roboten, både når roboten stod stille og når den var i bevegelse. Kamerakoden la utgangspunktet for den ene AI-strategien som var å lete etter baller på bordet.

Underveis har det vært gjort en del adminstrativt arbeide, som organisering av EiT, økonomi, reise til Tyskland etc. I tillegg ligger det mye arbeid bak å koble alle delmodulene sammen i roboten på en fornuftig måte. Arbeidet har tatt mye tid, men dette har vært helt nødvendig for å kunne framlegge et fungerende system til slutt.

Omfattende testing og resultater fra konkurransen viser at totalsystemet på roboten er veldig robust. Kommunikasjonen mellom modulene har fungert bra og mange feil ble luket bort under testingen. Dessverre greide ikke roboten å hevde seg i konkurransen grunnet tilfeldige feil. Roboten vant 2 av 5 kamper, og tapte de resterende pga. en skrue som falt av dekselet, en startsnor som ikke trigget startinterruptet og "jamming" av baller i sorteringsmodulen. På tross av et dårlig resultat i konkurransen er undertegnede godt fornøyd med arbeidet som er gjort, og tror at fundamentet for neste år skal være bra. Feilene som oppstod var tilfeldige og vanskelige å gardere seg fullstendig mot.

Innhold

Kapittel 1

Innledning

Denne rapporten beskriver arbeidet som er gjort med å utvikle en robot som skal delta i konkurransen Eurobot Open 2008. I tillegg til å bygge selve roboten og de systemene som trengs for å få denne til å virke, må det organiseres reise til Tyskland og gjøres administrativt arbeide i forbindelse med økonomi, påmelding o.l. Gjennom faget Eksperter i Team (EiT) er det to grupper som skal hjelpe til med design av en viktig modul; sorteringsmodulen. Denne modulen skal ta inn baller og sortere dem. Ellers ønskes det fokus på å forbedre posisjoneringssystemet og implementere en god kunstig intelligens (AI).

Oppgaven er først og fremst veldig praktisk, og størstedelen av arbeidet vil derfor vises gjennom et ferdig robotprodukt eller bilder/videoer av dette. Selve rapporten er ment som en dokumentasjon av hvordan roboten virker, samtidig som den forsøker å gi et inntrykk av hva det har vært jobbet med og fokusert på gjennom semesteret. Siden arbeidet består av en del uavhengige oppgaver, er det valgt å dele rapporten inn i en del enkeltkapitler som hver inneholder en egen rapportstruktur.

Hovedområdene i rapporten er posisjonering, navigasjon, kunstig intelligens, antikollisjon, kommunikasjon og datasyn. I tillegg er det gjort mange andre mindre oppgaver, som er samlet under praktisk arbeid. I dette kapittelet er det også tatt med en del utenomfaglig arbeid som organisering, økonomi etc.

På et overordnet nivå finnes det kapitler som bakgrunn, testing og resultater, konklusjon og videre arbeid, der innholdet gjelder generelt for hele arbeidet.

Kapittel 2

Bakgrunn

2.1 Eurobot-konkurransen

2.1.1 Generelt

Eurobot Open er en internasjonal robotkonkurranse for studenter og uavhengige organisasjoner fra hele verden. Konkurransen stammer fra Frankrike og arrangeres årlig i mai måned, et sted i Europa. Eurobot Open har blitt arrangert siden 1998 og NTNU (Institutt for teknisk kybernetikk) har vært representert med et lag hvert år siden 2000.

Lag fra i underkant av 30 nasjoner har deltatt de siste årene. Hver nasjon kan maksimalt stille med 3 lag, men inkludert de nasjonale kvalifiseringene har rundt 350 lag vært involvert hvert år. Frankrike er desidert størst med rundt 150 lag, men også andre land som f.eks. Tyskland og Italia har omfattende nasjonale kvalifiseringsrunder.

Institutt for teknisk kybernetikk har som tidligere nevnt representert NTNU i Eurobot siden 2000. Prestasjonene har vært noe varierende, men stort sett ganske bra, hvis man tar i betraktning lagstørrelse og ressursbruk i forhold til mange av de store utenlandske lagene. Nivået i finalene har generelt vært veldig høyt i og med at mange av lagene har gjennomgått flere runder med nasjonal kvalifisering og dermed kunnet videreutvikle sin robot. Tabell 2.1 viser NTNUs

tidligere plasseringer i Eurobot (hentet fra Kjemphol og Knausgård [1] og http://www.eurobot.org/).

År/konkurransenavn	**NTNUs plassering**	**Antall lag i finalen (antall lag totalt)**
2000 Fun Fair	8.	13
2001 Space Odyssey	13.	18
2002 Flying Billiards	17.	25
2003 Heads or Tails	16.	32
2004 Coconut Rugby	21.	41
2005 Bowling	11.	50
2006 Funny Golf	44.	50 (350)
2007 Robot Recycling Rally	25.	39 (ca. 350)

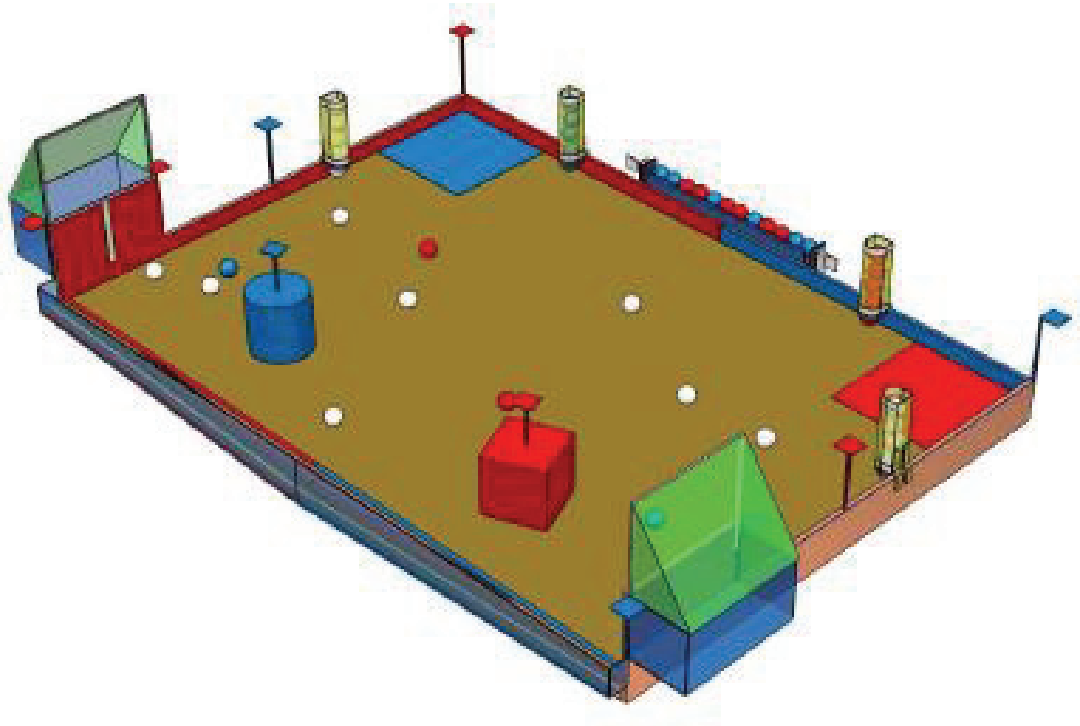
Tabell 2.1: NTNUs tidligere plasseringer i Eurobot

2.1.2 Reglement 2008

Navnet på årets konkurranse er "Mission to Mars", og historien går ut på at man skal samle steinprøver med potensielt levende organismer på den røde planeten. Lagene skal prøve å samle flest mulig prøver og oppbevare de avkjølt i hver sine depoter.

Den praktiske gjennomføringen av konkurransen er ganske lik det som har vært gjort de siste årene. Utgangspunktet er spillebordet på 3 x 2.1 meter, der de to konkurrerende robotene starter i hvert sitt hjørne. Robotene skal operere fullstendig autonomt og har tilsammen 90 sekunder på seg til å skaffe flest mulig poeng. Hvert lag skal samle steinprøver med sitt lags farge (rød eller blå) og bringe dem til sitt frysekammer. Steinprøvene er representert ved innebandyballer i ulike farger (13 blå og 13 røde). Det finnes også hvite baller på bordet (19 stk), som skal forestille is. Figur 2.1 viser årets spillebord og plassering av ballene og depotene. De store røde og blå boksene er de to konkurrerende robotene.

Plasseringen av ballene kan varierere noe fra kamp til kamp, slik at man ikke nødvendigvis kan plukke på samme steder hver gang. Det finnes imidlertid fire faste dispensere, der man kan plukke ut én og én ball. Det finnes også en horisontal dispenser med røde og blå baller som kan utløses mekanisk slik at disse faller ut på bordet. Resten av ballene ligger spredt på bordet. Roboten må selv finne ballene og frakte dem over til ett av de to depotene diagonalt på andre siden av bordet fra startposisjonen. Én mulighet er å legge ballene i en såkalt fryseboks, der man får 2 poeng per ball med riktig farge. Et annet alternativ er å bruke en renne der man kan legge baller ved siden av hverandre. Siden denne ikke er avkjølt, kan man skaffe ekstrapoeng ved å legge isballer mellom



Figur 2.1: Illustrasjon av spillebordet for Eurobot Open 2008

de fargede ballene. Dette var en enkel oppsummering av årets reglement, som i sin helhet ligger på http://www.eurobot.org/.

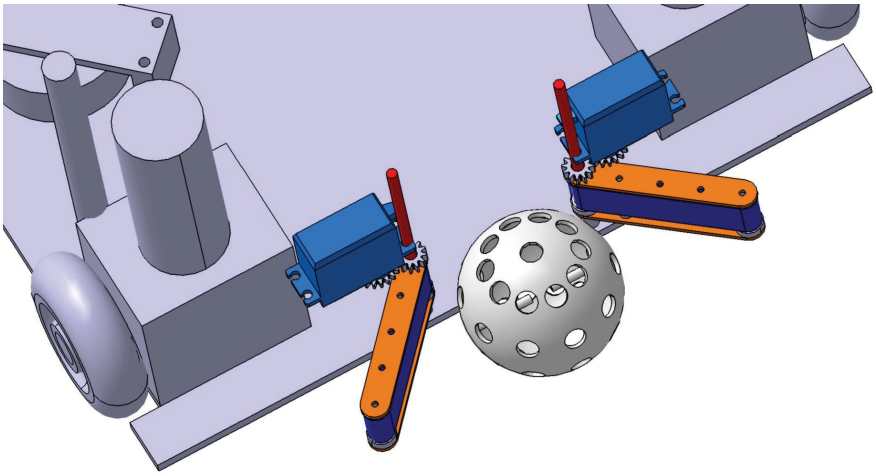
2.2 Sorteringsmodul laget av EiT

Underveis i arbeidet har det også vært hentet inn noe ekstern hjelp fra faget "Eksperter i Team" (EiT). Disse 10 studentene ble oppdelt i to grupper, og har bidratt med en meget viktig modul i roboten: Sorteringsmodulen. De har igjen delt opp modulen i to delmoduler som beskrives kort under. Hele dette arbeidet er forøvrig grundig dokumentert i deres fagrapporter, EiT 2008 gruppe 1 [2] og 2 [3].

Begge delmodulene er styrt av hvert sitt kretskort. Disse to er helt like og er koblet til CAN-bussen for kommunikasjon med resten av roboten. Kommunikasjonen er nærmere beskrevet i kapittel 8.

2.2.1 Innmating av baller

Den første delmodulen sørger for å få baller fra bordet inn i roboten. I hovedsak består denne av to armer med roterende gummibånd som kan klype sammen og dra baller inn, se figur 2.2. Armenes vinkel og hastighet kan settes direkte via en CAN-melding. Delmodulen inneholder i tillegg en nærhetsdetektor, basert på Ir, som registrerer at en ball kommer inn. I tillegg har den en fargesensor som kan sjekke om ballen er rød, blå eller hvit.

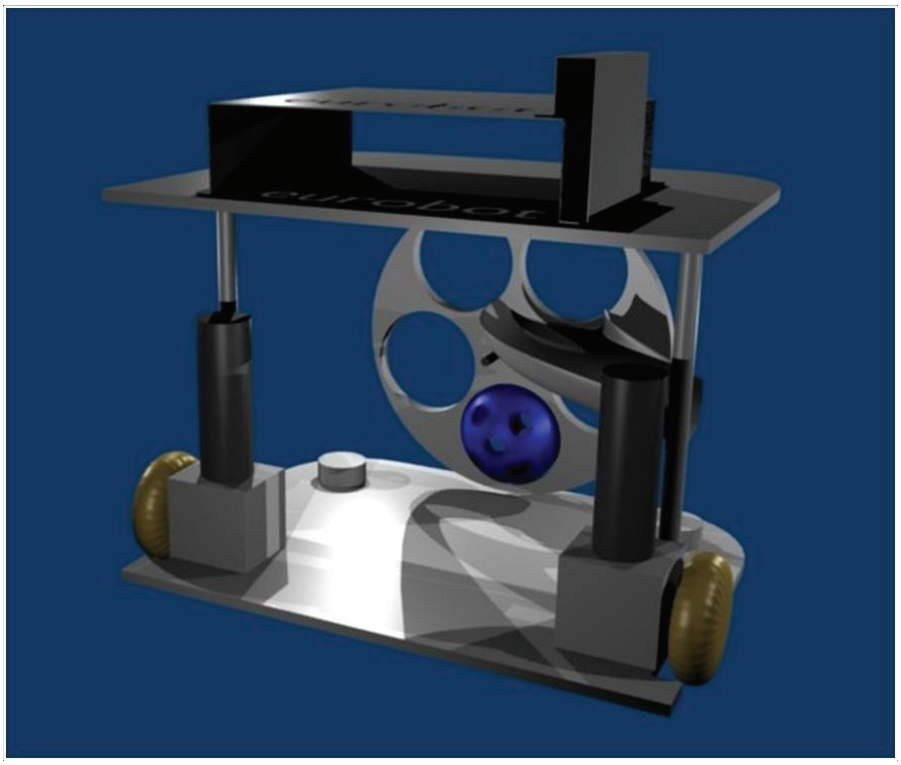


Figur 2.2: Konseptskisse for innmatingsarmer

2.2.2 Sorteringshjul

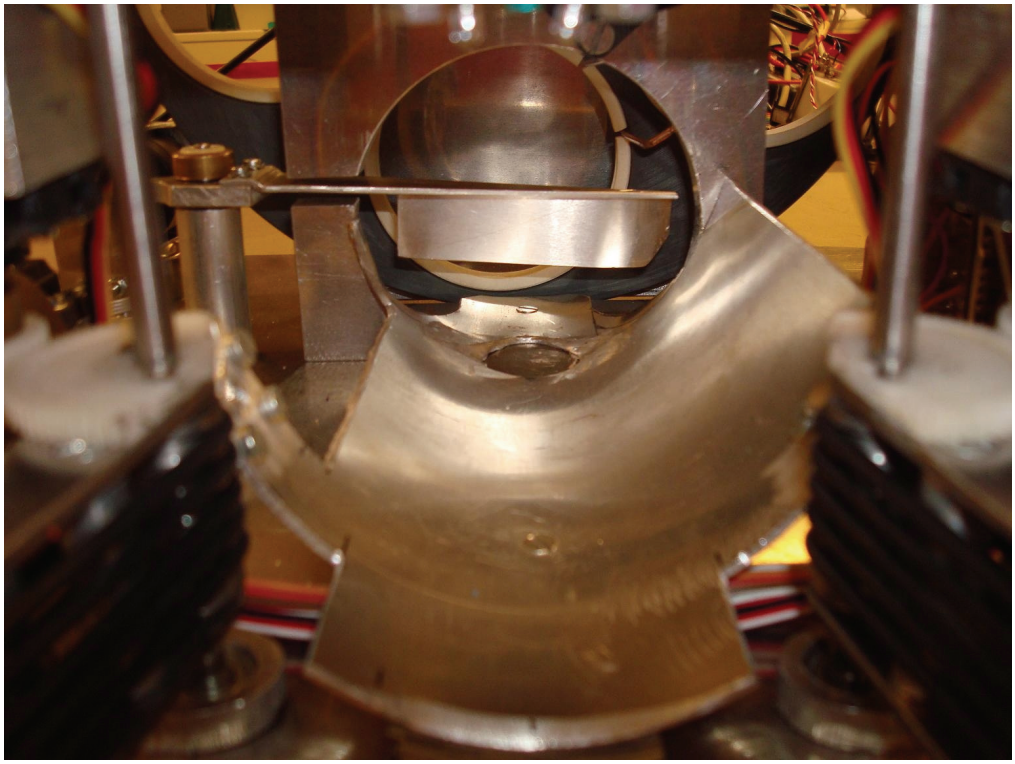
Den andre delen av sorteringsmodulen har som oppgave å plassere ballene i et magasin, samt sørge for at ballene kan slippes ut i sortert rekkefølge. Magasinsløsningen er laget med et vertikalt hjul som er posisjonsregulert, se figur 2.3. Hjulet kan maksimalt inneholde 5 baller ihht. reglene, og disse blir sortert i det de puttes inn. For å få ballene inn og ut av magasinet blir det brukt to armer styrt av servoer. Den første er plassert langt ned i roboten, rett innenfor innmatingsarmene, og har som oppgave å dytte ballene inn i magasinet. Armen for utmating er plassert bak øverste del av hjulet. Denne settes ut slik at alle baller i hjulet som passerer blir dyttet ut på en renne, som fører dem ut foran på roboten. Ved å kjøre hjulet rundt med utmatingsarmen ute, vil dermed ballene trille ut og falle ned foran roboten.

Disse to delmodulene er helt avhengig av å samarbeide for å fungere skikkelig. Timingen og kommunikasjonen mellom dem er viktig, noe man må være veldig oppmerksom på når man skal forholde seg til delmodulene som én enhet. Figur



Figur 2.3: Konseptskisse for sorteringshjul

2.4 viser overgangen fra innmatingsarmene, i forkant, til armen som dytter ballene inn i magasinet, stående på tvers, i bakkant.



Figur 2.4: Overgangen mellom de to delene i sorteringsmodulen

Kapittel 3

Posisjonering

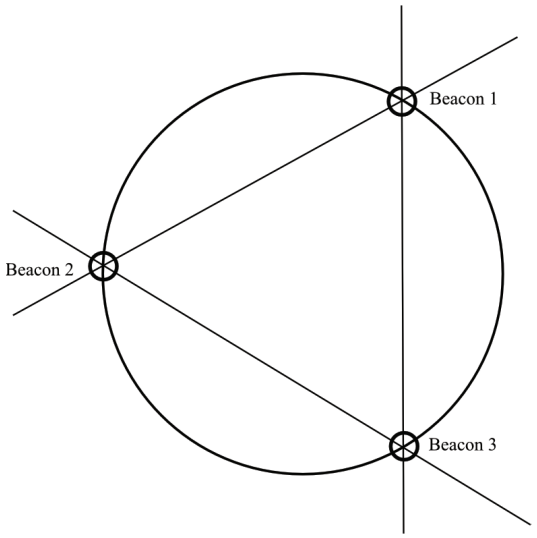
Posisjonering er en veldig viktig del av roboten. For å kunne manøvrere til de ulike elementene på spillebordet er man helt avhengig av å ha et godt posisjonsestimat. Dette kapittelet beskriver de to modulene som gir navigasjonssystemet posisjonsmålinger underveis. Posisjoneringsmodulene bruker to forskjellige prinsipper; odometri og triangulering, og gir henholdsvis relative og absolutte posisjonsmålinger. Det er først og fremst lagt vekt på det absolutte posisjoneringssystemet, siden dette er nyutviklet og ikke har vært i bruk tidligere. Odometrien har kun gjennomgått enkelte justeringer og er fullstendig beskrevet i Mørkrid og Øien 2007, [4].

3.1 Teori og bakgrunn

3.1.1 Triangulering med 3 sendere

Triangulering er mye brukt for å beregne posisjonen til et objekt som kan benytte seg av aktive sendere. For å finne nøyaktig posisjon til objektet, er det tilstrekkelig å kunne beregne vinkelen til to faste punkter. Utfordringen dukker imidlertid opp når man ikke vet sin egen orientering, og dermed bare kan beregne vinklene relativt til de faste punktene. Dette er situasjonen for roboten, og gjør det derfor nødvendig å ha vinkelmålinger til 3 ulike punkter. Det er helt nødvendig at alle 3 senderne er synlige for å kunne beregne robotens posisjon. En generell begrensning for alle trianguleringsmetoder er at det finnes enkelte punkter der posisjoneringen ikke er absolutt. Punkter på sirkelen som dannes av de 3 senderne vil være et slikt tilfelle, og langs randen av denne er det altså ikke mulig å regne ut posisjonen nøyaktig. I tillegg kommer de rette linjene mellom to og to sendere, der man mister en dimensjon i utregningen. Alle problematiske linjer er illustrert i figur 3.1. Videre beskrives kort noen

metoder for posisjonering med 3 sendere, hentet fra artikkelen til Esteves, Carvalho, og Couto 2003 [5].



Figur 3.1: Punktene langs disse linjene er ikke mulig å posisjonere korrekt ved hjelp av triangulering

Standard geometrisk triangulering

Geometrisk triangulering er en velkjent og mye brukt metode for å regne ut posisjon og orientering til et objekt i planet. Utregningen er forholdsvis enkel og ukomplisert, se framgangsmåte i figur 3.2. Metoden virker å være en standardløsning, og er den som stort sett er beskrevet hvis man søker gjennom litteratur etter løsninger for å finne posisjon ved hjelp av triangulering. Det finnes imidlertid noen ulemper som er lette å overse, men som den nevnte artikkelen påpeker og beskriver i detalj. I tillegg til de vanlige, generelle trianguleringsbegrensningene vist over, trekkes det fram følgende begrensninger ved geometrisk triangulering:

1. Senderne må nummereres i fast rekkefølge mot klokken
2. Posisjonen kan bare måles korrekt innenfor triangelet som dannes av de 3 senderne

Den første av disse har liten praktisk betydning, da senderne i vårt tilfelle hele tiden står fast, og nummereringen kan velges deretter. Punkt nummer 2 gir derimot større grunn til bekymring. Slik senderplattformene er satt opp på spillebordet, vil en stor del av området som roboten kan bevege seg i, falle utenfor dette triangelet. I praksis betyr dette at denne trianguleringsmetoden ikke egner seg særlig godt for situasjonen på spillebordet.

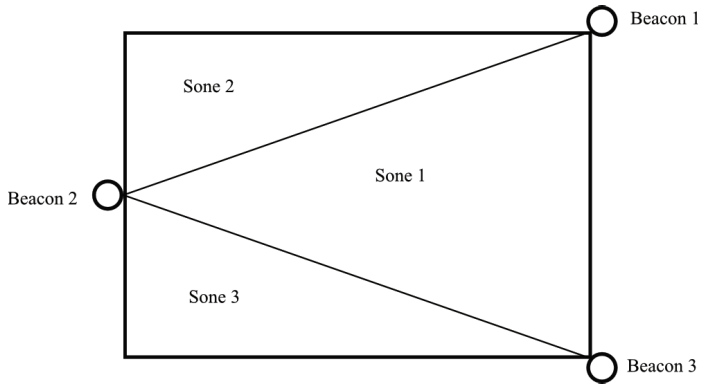
Geometric Triangulation Algorithm

1. Properly order beacons.
2. Let $\lambda_{31} = 360^{o} + (\lambda_1 - \lambda_3)$
3. Let $\lambda_{12} = \lambda_2 - \lambda_1$
4. Let ϕ be the angle between the positive x-axis and the line formed by the points of beacons 1 and 2.
5. Let σ be the angle between the positive x-axis and beacons 1 and 3, plus ϕ.
6. Let $\gamma = \sigma - \lambda_{31}$
7. Let $p = \frac{L_{31} \cdot \sin\lambda_{12}}{L_{12} \cdot \sin\lambda_{31}}$
8. Let $\tau = \tan^{-1}\left[\frac{\sin\lambda_{12} - p \cdot \sin\gamma}{p \cdot \cos\gamma - \cos\lambda_{12}}\right]$
9. Let $L_1 = \frac{L_{12} \cdot \sin(\tau + \lambda_{12})}{\sin\lambda_{12}}$
10. $x_R = x_1 - L_1 \cdot \cos(\phi + \tau)$
11. $y_R = y_1 - L_1 \cdot \sin(\phi + \tau)$
12. $\theta_R = \phi + \tau - \lambda_1$

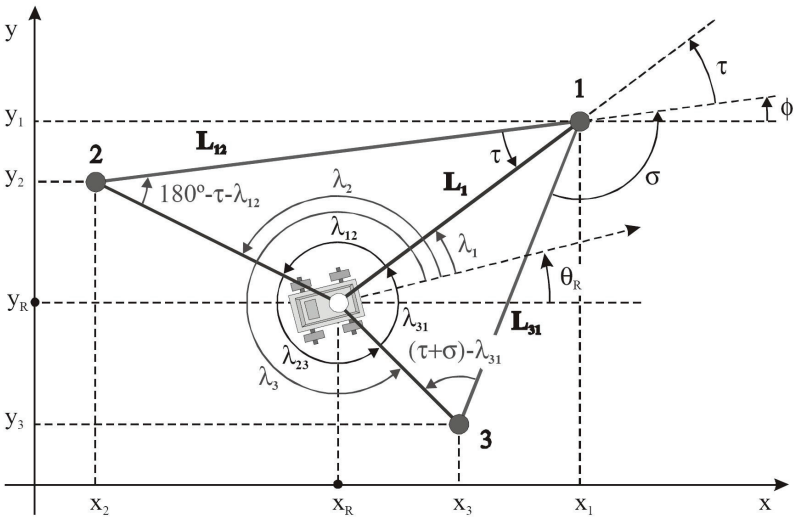
Figur 3.2: Algoritmebeskrivelse for standard geometrisk triangulering

Forbedret trianguleringsmetode

Esteves, Carvalho, og Couto 2003 [5] beskriver også disse begrensningene som problematiske i et mer generelt miljø, og har forsket videre for å finne en bedre metode som kan brukes i flere sammenhenger. Løsningen har blitt å dele opp det aktuelle posisjoneringsområdet i 4 soner. I tilfellet med et rektangulært spillebord vil bare 3 av disse sonene falle innenfor dette, se figur 3.3. Sonene avgjøres av om vinkelen mellom to av senderne, sett fra roboten, er større eller mindre enn 180°, og gir grunnlag for litt ulike geometriske utregninger. Alle vinkler og lengder er definert ut fra artikkelen og er vist i figur 3.4.



Figur 3.3: Inndeling i soner for triangulering



Figur 3.4: Definisjon av alle vinkler og lengder

Det kan settes opp følgende sammenhenger for hver av sonene:

Sone 1:

$$\frac{L_{31}}{\sin \lambda_{31}} = \frac{L_1}{\sin(\tau + \sigma - \lambda_{31})}$$
$$\frac{L_{12}}{\sin \lambda_{12}} = \frac{L_1}{\sin(180^\circ - \tau - \lambda_{12})}$$

Sone 2:

$$\frac{L_{31}}{\sin \lambda_{31}} = \frac{L_1}{\sin(\tau + \sigma - \lambda_{31})}$$
$$\frac{L_{12}}{\sin(360^\circ - \lambda 12)} = \frac{L_1}{\sin(-180^\circ + \tau + \lambda_{12})}$$

Sone 3:

$$\frac{L_{31}}{\sin(360^\circ - \lambda 31)} = \frac{L_1}{\sin(\lambda_{31} - \tau - \sigma)}$$
$$\frac{L_{12}}{\sin \lambda 12} = \frac{L_1}{\sin(180^\circ - \tau - \lambda_{12})}$$

Dermed kan man regne ut L_1, som tilsvarer avstanden fra roboten til sender 1, og videre finne robotens x-posisjon, y-posisjon og retning:

$$x_R = x_1 - L_1 \cos(\phi + \tau)$$
$$y_R = y_1 - L_1 \sin(\phi + \tau)$$
$$\Theta_R = \phi + \tau + -\lambda_1$$

Dette utgjør hovedprinsippene for utregning av posisjonen, men det er naturlig nok en del andre tilleggsdata som må regnes ut etter hvert. Hovedtrekkene i en framgangsmåte for utregning, som foreslåes i artikkelen, er vist i figur 3.5.

Den forbedrede metoden har som sagt en del fordeler i forhold til den geometriske trianguleringen og egenskaper kan kort oppsummeres slik:

1. De tre senderne kan stå i hvilken som helst rekkefølge
2. De tre senderne kan stå hvor som helst i planet, så lenge de ikke har sammenfallende posisjoner
3. Både vinkelen mellom sender 1 og 2, og sender 3 og 1 kan være større enn eller lik 180°
4. Algoritmen virker korrekt over hele området (bortsett fra de kjente linjene som begrenser enhver tre-senders algoritme)

Generalized Geometric Triangulation Algorithm

1. If there are less than three visible beacons available, then return a warning message and stop.
2. $\lambda_{12} = \lambda_2 - \lambda_1$
3. If $\lambda_1 > \lambda_2$ then $\lambda_{12} = 360^o + (\lambda_2 - \lambda_1)$
4. $\lambda_{31} = \lambda_1 - \lambda_3$
5. If $\lambda_3 > \lambda_1$ then $\lambda_{31} = 360^o + (\lambda_1 - \lambda_3)$
6. Compute L_{12} from known positions of beacons 1 and 2.
7. Compute L_{31} from known positions of beacons 1 and 3.
8. Let ϕ be an oriented angle such that $-180^o < \phi \leq 180^o$. Its origin side is the image of the positive x semi-axis that results from the translation associated with the vector which origin is (0, 0) and ends on beacon 1. The extremity side is the part of the straight line defined by beacons 1 and 2 which origin is beacon 1 and does not go by beacon 2.
9. Let σ be an oriented angle such that $-180^o < \sigma \leq 180^o$. Its origin side is the straight line segment that joins beacons 1 and 3. The extremity side is the part of the straight line defined by beacons 1 and 2 which origin is beacon 1 and does not go by beacon 2.
10. $\gamma = \sigma - \lambda_{31}$
11. $\tau = \tan^{-1}\left[\dfrac{\sin\lambda_{12} \cdot (L_{12} \cdot \sin\lambda_{31} - L_{31} \cdot \sin\gamma)}{L_{31} \cdot \sin\lambda_{12} \cdot \cos\gamma - L_{12} \cdot \cos\lambda_{12} \cdot \sin\lambda_{31}}\right]$
12. If $\begin{cases}\lambda_{12} < 180^o \\ \tau < 0^o\end{cases}$ then $\tau = \tau + 180^o$
13. If $\begin{cases}\lambda_{12} > 180^o \\ \tau > 0^o\end{cases}$ then $\tau = \tau - 180^o$
14. If $|\sin\lambda_{12}| > |\sin\lambda_{31}|$ then $L_1 = \dfrac{L_{12} \cdot \sin(\tau + \lambda_{12})}{\sin\lambda_{12}}$
15. else $L_1 = \dfrac{L_{31} \cdot \sin(\tau + \sigma - \lambda_{31})}{\sin\lambda_{31}}$
16. $x_R = x_1 - L_1 \cdot \cos(\phi + \tau)$
17. $y_R = y_1 - L_1 \cdot \sin(\phi + \tau)$
18. $\theta_R = \phi + \tau - \lambda_1$
19. If $\theta_R \leq -180^o$ then $\theta_R = \theta_R + 360^o$
20. If $\theta_R > 180^o$ then $\theta_R = \theta_R - 360^o$

Figur 3.5: Algoritmebeskrivelse for forbedret triangulering

3.1.2 Utregning av vinkler fra kjent posisjon

For å kunne kontrollere at posisjonsmålinger fra trianguleringssystemet er fornuftige, og dermed kunne forkaste usannsynlige eller feilaktige posisjonsmålinger, kan det være hensiktsmessig med en omregning fra posisjon til sendervinkler. Dette er en mye enklere operasjon, og krever ingen avansert algoritme som i tilfellet over. Sendervinklene, λ_n, er gitt av:

$$\lambda_n = -\arctan(b_y^n - y_R, b_x^n - x_R)\frac{180}{\pi} - \theta_R$$

der b_x^n og b_y^n er henholdsvis x- og y-posisjon til den enkelte sender, mens x_R, y_R og psi_R representerer robotens posisjon.

3.1.3 Spesialtilfelle: Utregning av posisjon inntil veggen

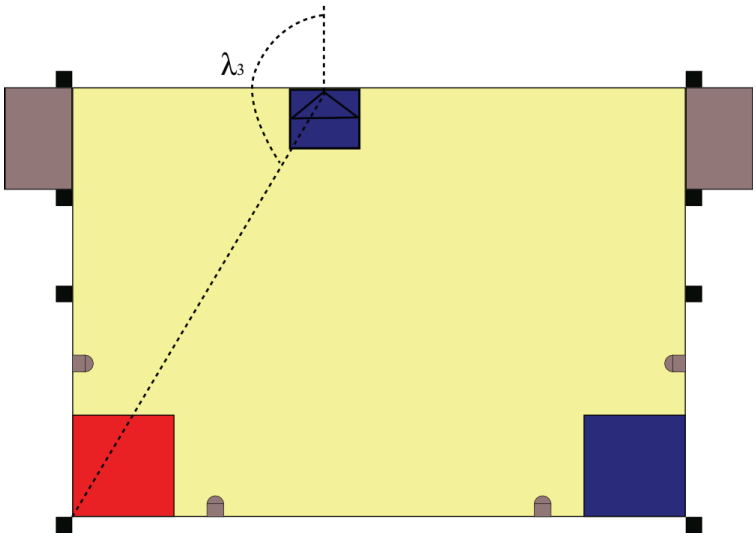
I årets konkurranse vil det i stor grad være nødvendig at roboten plasseres inn mot en vegg, f.eks. i forbindelse med tømming av baller. I denne situasjonen vil roboten sannsynligvis stå stille i en liten periode, noe som tilsier at det egner seg meget godt til å foreta posisjonsmålinger. At roboten står inn mot veggen, gjør at både vinkel- og x- eller y-posisjonen er kjent i punktet. Dermed trenger vi bare målinger fra ett av tårnene og burde kunne oppnå et rimelig nøyaktig posisjonsestimat.

Utregningen kan enkelt gjøres utenom trianguleringsalgoritmen ved hjelp av enkel geometri. Figur 3.6 viser et eksempel der roboten tømmer baller i det ene depotet, og vinkelen til en av senderne er λ_3. Her er y-posisjonen og retningen kjent, mens x-posisjonen kan finnes slik:

$$x_R = y_R \tan(180^\circ - \lambda_3) \tag{3.1}$$

3.1.4 Eksisterende hardware og software

Et posisjoneringssystem basert på 3 aktive sendere er avhengig av en modul på roboten som kontinuerlig kan lese av vinklene til hver av dem. I forbindelse med en masteroppgave i høst, se Knausgård 2007 [6], er det gjort et grundig arbeid med å utvikle et konsept for avlesning av sendere. Det er satt igang utvikling av hardware og noen av prinsippene for kommunikasjon mellom modulene er testet i separate forsøk med annen hardware. Selvom dette arbeidet gir et



Figur 3.6: Eksempel på spesialtilfelle ved tømming

godt utgangspunkt for videre utvikling, er det mye som gjenstår både når det gjelder hardware og software. Det er verdt å merke seg at kretskortene fra denne oppgaven ikke var fullstendige, både med hensyn på komponenter og koblinger. I tillegg finnes det heller ikke noe ferdig software som er testet på denne hardwaren.

Posisjoneringstårnet består i utgangspunktet av to kretskort som er plassert i et rør av pleksiglass. Ir-sensorer er plassert på det øverste kortet, heretter kalt sensorkortet, og dette kortet roteres rundt ved hjelp av en motor. Dette kortets eneste funksjon er å si fra hver gang en av senderne observeres. På undersiden sitter hovedkortet i posisjoneringstårnet og har som oppgave å styre rotasjonen av sensorkortet, ta i mot senderobservasjoner, lese av sensorkortets vinkel og behandle og distribuere posisjonsdata på CAN-bussen. Begge kortene er etset og de fleste av komponentene er loddet på, men ingen av delene har vært programmert eller testet.

For at det roterende sensorkortet skal få strøm er det laget et sleperingsystem som fungerer utmerket. Det hele passer perfekt inn i pleksiglassrøret der man ved å sette spenning på motoren kan se at sensorkortet roterer fint og får tilstrekkelig spenning på inngangen. Kommunikasjonen mellom de to kortene er planlagt å gjøres via Ir, kodet vha. IrDA-standarden, noe som er blitt testet ut med to prototypekort.

3.2 Odometri

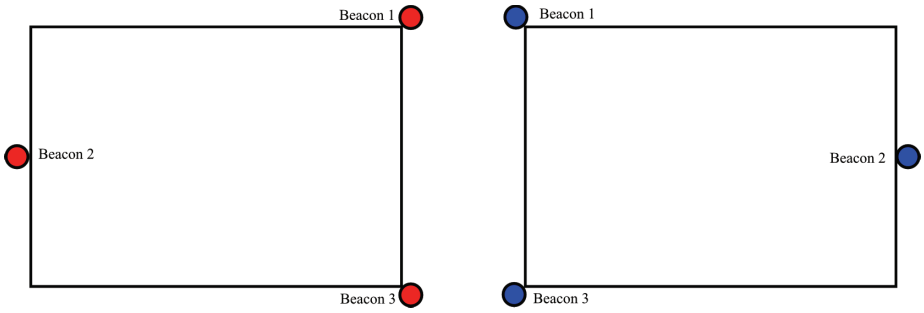
Odometrien var som nevnt velfungerende, men det har likevel vært nødvendig å gjøre noen små justeringer. Først og fremst har o-ringene på løpehjulene blitt byttet oftere enn tidligere, senest ved avreise til Tyskland. Det er observert at disse blir relativt fort slitt og at gripeevnen reduseres betydelig. Overraskende nok ble det også observert spinn på løpehjulene med tilnærmet helt nye o-ringer. Etter nærmere undersøkelser viste det seg at fjærene som dytter løpehjulene ned mot underlaget hadde ulik og ganske liten kraft. Disse ble byttet ut med kraftigere fjærer, som gav mye bedre press mot underlaget. Det anbefales å gjøre en grundig sjekk på det mekaniske oppsettet ved hver forandring eller demontering av odometrioppsettet.

Ved tidligere testing har det vist seg at posisjoneringsfeilen bygget seg opp til å bli ganske stor etter relativt kort tid, selv uten synlig spinn på løpehjulene. Ved nøye inspeksjon av software-implementasjonen ble det avslørt et par svakheter. For det første har det blitt gjort noen minimale forenklinger fra teoretiske ligninger til ferdig C++-kode. Grunnen har vært at disse gir betydelig besparelse på utregningshastighet og skulle tilsynelatende ha lite å si for resultatet. Ved testing viste det seg at nøyaktige utregninger ikke vil sinke systemet i særlig grad, da PC-en er forholdsvis lite belastet. Samplingstiden for odometrimålinger kunne også skrues opp, uten at systemet på noen måte ble overbelastet. I og med at hele teorien er avhengig av stor nok samplingstid, anses dette som en betydelig forbedring av nøyaktigheten. Posisjonsmålingene har i senere tid vist seg å bli bedre og mer stabile, uten at dette har vært systematisk sammenlignet med tidligere. Antagelig har både de software-messige forandringene og justeringene på det mekaniske oppsettet bidratt til bedre ytelse.

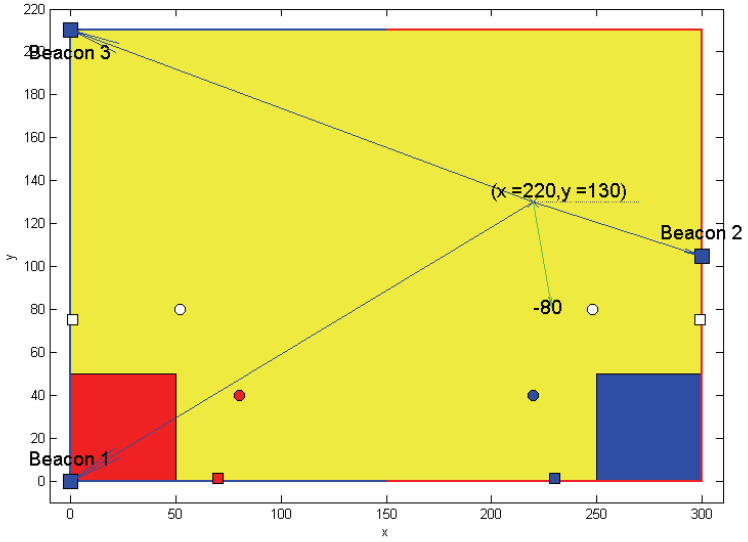
3.3 Simulering og verifikasjon av trianguleringsteorien

I og med at trianguleringsmetoden har ganske mange ligninger, blir det etter hvert ganske komplisert å få oversikt. Det ble funnet hensiktsmessig i første omgang å implementere det hele i Matlab og simulere mange ulike posisjoneringssituasjoner der. Matlab-miljøet gjør det enkelt å sette inn ligningene, samt at man raskt kan få opp intuitive og grafiske diagrammer av simuleringene. Først og fremst ble det viktig å kunne fastslå at algoritmen fungerer korrekt for de 3 sonene, samtidig som den må fungere for begge lag. Senderoppsettene til de to ulike lagene er vist i figur 3.7. Under simuleringsimplementasjonen viste det seg problematisk å finne riktig uttrykk for de vinklene som ikke var regnet ut i artikkelen. Figurene var i flere tilfeller ganske uklare og ikke generelle nok

til å kunne sette opp gode uttrykk for alle parametre som trengtes i utregningen. Testingen underveis ble veldig viktig og etter hvert så det ut til å fungere for alle mulige posisjoner på spillebordet. Det er verdt å legge merke til at denne implementasjonen muligens ikke er like generell som algoritmen beskrevet over, men den skal likevel kunne brukes for alle mulige tårnoppsett på et rektangulært spillebord som brukes i Eurobot. En eksempelsimulering er vist i figur 3.8, basert på 3 tilfeldig valgte sendervinkler. Under testingen ble det også kontinuerlig verifisert at tilbakeregningen fra posisjon til vinkler gav samme verdier som inndataene.



Figur 3.7: Senderoppsett for henholdsvis rødt og blått lag



Figur 3.8: Eksempelsimulering i Matlab

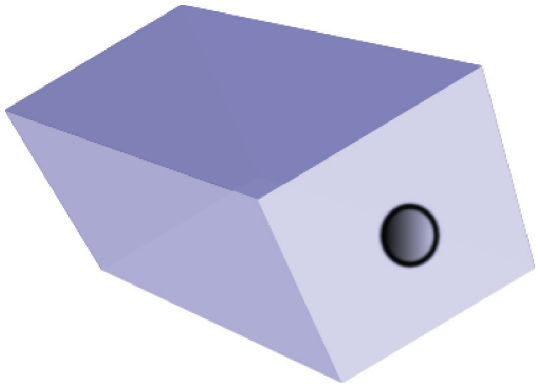
3.4 Posisjoneringstårn

3.4.1 Fullføring og videreutvikling av kretskort til posisjoneringstårn

Sensorkort

Sensorkortet som var laget, viste seg etter en del testing å ikke fungere i det hele tatt. En del av koblingene til mikrokontrolleren viste seg å være dårlige. Likevel ble det ikke oppnådd kontakt før både mikrokontroller og et par andre støttekomponenter ble erstattet med nye. Dessverre manglet det i tillegg noen tilkoblinger i forbindelse med den trådløse kommunikasjonen, og det ble derfor besluttet å lage et nytt tilsvarende kort med noen korreksjoner. I denne forbindelse ble det også satt på en status-diode, slik at det er mulig å se at sensorkortet er påslått. Med noen midlertidige tilkoblinger kunne det opprinnelige kortet fint fungere som reservekort ved behov. Kretsskjema for ferdig sensorkort er vist i vedlegg A.

Siden man er avhengig av å få nøyaktige vinkelmålinger, er det viktig at synsvinkelen til sensorene er så smal som mulig, men at man samtidig er sikker på å kunne observere senderne stabilt. Som løsning på dette er det laget en slags hette som passer på toppen av sensoren. Denne er skåret ut i plast og har et kikkehull på 5 mm og en lengde på omtrent 2 cm, se figur 3.9.



Figur 3.9: Hette for skjerming av Ir-sensor

Når det gjelder software er det blitt utviklet et nytt system for avlesning av sendere. Det er valgt å kombinere bruk av en innebygd timer på mikrokontrolleren, med interrupts på pinnen som er koblet til sensorutgangen. Slik som senderene er bygget opp, er det tilstrekkelig å kunne måle lengden på én innkommende puls, se forøvrig avsnitt 3.4.2 for en nærmere beskrivelse av sendingen. Ved å starte timeren på stigende og stoppe den på fallende flanke,

kan man måle tiden på én periode av sendersignalet. Med en liten toleransegrense, sjekkes det om pulslengden tilsvarer en av tre forhåndsdefinerte lengder for senderne. For å forsikre seg om at det ikke kommer enkeltforstyrrelser innenfor disse lengdene blir det kun registrert målinger når det kommer 5 like perioder på rad. Ved hver måling sendes den respektive senders nummer til hovedkortet via Ir. På grunn av softwaren i mikrokontrolleren og antall interrupts-innganger er det bare valgt å bruke én sensor, i motsetning til det opprinnelige konseptet som foreslo 4. Det skulle likevel bli tilstrekkelig antall målinger, og man kan også delvis kompensere for dette ved å sette opp hastigheten på sensorkortrotasjonen.

Underveis i utviklingen oppstod det en del problemer med minnekapasitet på mikrokontrolleren. En Atmega8, som blir brukt i dette tilfellet, har ikke veldig stor kapasitet og for å få en tilstrekkelig oppdateringsrate krever softwaren mye. Siden det kun kreves ganske enkel tallbehandling, ble det ved å modifisere kompileringsprosessen fjernet støtte for negative tall, desimaltall og en del matematiske funksjoner. Forandringene og diverse kommentarer på dette er vist i Makefilen til sensorkortet, som finnes sammen med all annen mikrokontrollerkildekode, se evt. vedlagt CD.

Hovedkort

Hovedkortet var i større grad fungerende og trengte bare noen omkoblinger på et par komponenter som var koblet feil. Det ble brukt mye tid på å konfigurere og sette opp Ir-kommunikasjonen mot sensorkortet. Korrekt hastighet på UART og IrDA-kontroller er veldig viktig, og det viste seg vanskelig å få dette til å stemme eksakt med de to ulike kretskortoppsettene. Etter hvert ble alle krystaller byttet ut slik at de var identiske på begge kort. Til slutt ble kommunikasjonen veldig stabil, selv med høy rotasjonshastighet på sensorkortet. Likevel ble sendingen gjort enklest mulig slik at det ikke skulle oppstå overbelastning i noen ender. Som hentydet i beskrivelsen av sensorkortet, sendes det kun ett tall mellom 1 og 3 for hver senderobservasjon.

Koblingen mellom hovedkortet og kvadraturtelleren for avlesning av vinkel ble også byttet ut, da denne var veldig skjør og hadde dårlig kontakt på flere av pinnene. Gjennom testing har det blitt erfart at stort sett alle problemer med kvadraturtelleravlesning skyldes nettopp denne overgangen. Det viste seg også å være et problem at o-ringen mellom motoren og hovedakslingen på sensorkortet ble slitt. Denne ble byttet ut med en større ring, noe som også førte til mindre belastning på motoren.

Drivere for motorstyring, kvadraturtelleravlesning, IrDA-kommunikasjon og kommunikasjon mot CAN-bussen er ganske standard, og ble funnet ved å lete gjennom kildekoden til tidligere kretskort som har vært brukt i Eurobot-sammenheng. Det har imidlertid vært brukt en del tid på å koble alt sammen

og få det til å fungere på det gjeldende hardware-oppsettet.

Etter at alle delkomponenter var implementert og testet, kunne man konsentrere arbeidet rundt selve oppgaven til hovedkortet, nemlig det å ta imot målinger fra sensorkortet og tolke disse riktig. For hver mottatte måling fra sensorkortet blir vinkelen regnet ut ved å hente målinger fra kvadraturtelleren. Litt ustabile målinger gjør at disse vinklene vil variere en del og enkelte forstyrrelser gjør at man plutselig opplever å få vinkelmålinger fra hvor som helst, dvs. opp mot 180°feil. For å forhindre dette er det laget et rundbuffer med vinkelmålinger for hver av senderene. Den oppdaterte vinkelen til hver sender finnes ved å regne ut gjennomsnittet av dette bufferet. For å unngå opplagt feilaktige målinger godtar bufferet kun nye vinkelmålinger som ligger innenfor ±5°av dette snittet. Dermed antar man i utgangspunktet at senderen ikke kan flytte seg mer enn 5°mellom to målinger. Dette krever imidlertid at buffrene må initieres ut fra navigasjonssystemets aktuelle posisjonsestimat hver gang man ønsker å starte målinger. En initiering av målinger oppdaterer buffrene og venter på et minste antall godkjente vinkelmålinger før gjennomsnittet av buffrene sendes tilbake til navigasjonssystemet. Dermed kan navigasjonssystemet alltid vite at posisjoneringstårnet faktisk har fått et minste antall målinger i de antatte vinkelområdene for senderene, dersom det får tilbake en vinkelmålingsoppdatering. På denne måten vet man at vinklene er oppdaterte og kan brukes til posisjonsutregning. I verste fall får man ikke målinger, og det aktuelle posisjonsestimatet blir beholdt. Uansett kan man ikke gardere seg mot at utsikten til en sender er blokkert av den andre roboten, men man kan håpe på at dette ikke er tilfellet ved neste måling.

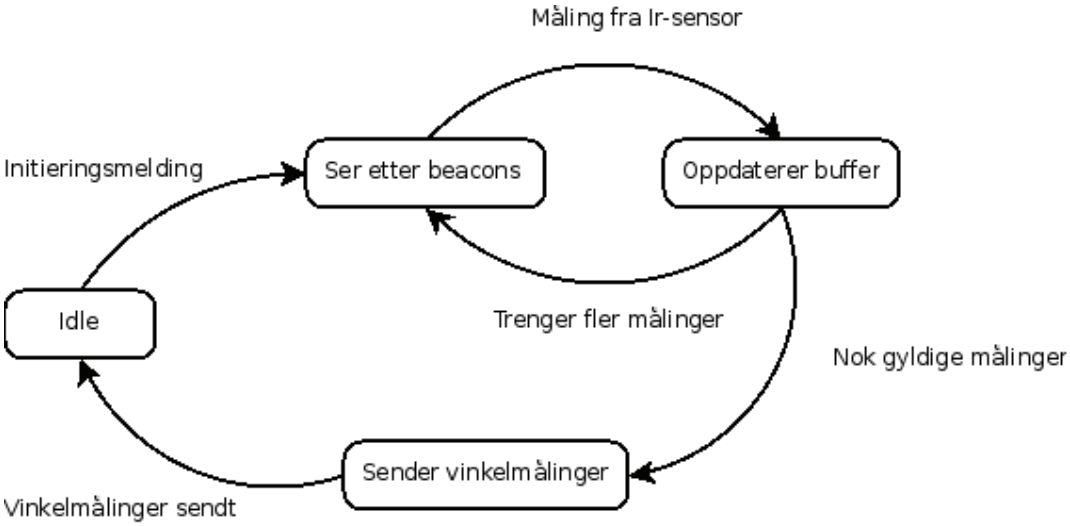
Hovedkortets oppførsel kan oppsummeres i 4 tilstander, også vist i figur 3.10:

1. **Idle** - Tårnet står stille og venter på forespørsel om å foreta målinger.

2. **Ser etter sendere** - Navigasjonsystemet gir beskjed om at en måling skal foretaes, buffrene stilles til antatte sendervinkler, rotasjon av sensorkortet settes igang og hovedkortet venter på senderobservasjoner.

3. **Oppdaterer buffer** - Hovedkortet har fått en måling og sjekker vinkelen. Dersom vinkelen godtas ift. gjennomsnittsverdien på det aktuelle bufferet, legges den inn og en teller for antall gyldige målinger inkrementeres. Dersom antall gyldige målinger overskrider minimumsgrensen, hopper man til den siste tilstanden, hvis ikke går man tilbake til 2.

4. **Sender vinkelmålinger** - Oppdaterte vinkelmålinger hentes ut fra buffrene og pakkes i en CAN-melding som sendes tilbake til navigasjonssystemet.



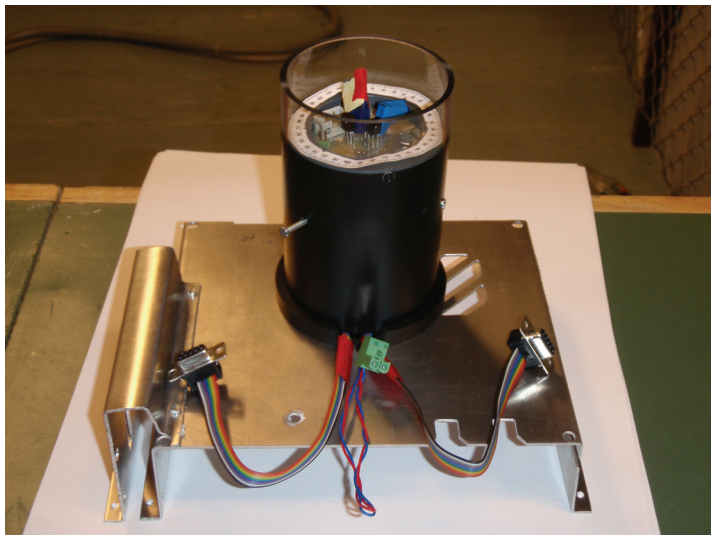
Figur 3.10: Tilstandsdiagram for hovedkort i posisjoneringstårn

Tårnutforming

Eurobot-reglene spesifiserer klart hvor stort posisjoneringstårnet kan være, og hvor det kan plasseres. Høyden ble beregnet i forbindelse med utformingen av resten av roboten, og heldigvis var sensorhøyden riktig tilpasset slik at det ble korrekt i forhold til senderplattformene rundt bordet. Noe som var viktig når tårnet skulle festes, var at vinkelen ble riktig i forhold til kvadraturtelleren og at det stod ca. midt på roboten. Resultatet ble en sylinder som hadde samme diameter som den ytre på tårnet, slik at tårnet kunne settes ned og roteres fritt i festet, se figur 3.11. En liten skrue på siden gjør at man kan sette tårnet fast, når vinkelen er helt riktig. I tillegg ble det laget et hull på baksiden for CAN-buss-ledninger og strøm til motoren, som vist i figur 3.12. I denne figuren kan man også se at det er lagt på en slags kompassrose langs kanten øverst i tårnet, slik at man enkelt kan få en viss oversikt over alle vinkler i forhold til tårnet.

I reglene er det også spesifisert at et eventuelt posisjoneringstårn samtidig skal ha en plattform for en evt. motstandersender. For å tilfredsstille dette ble det laget en 8 x 8 cm plate på toppen, som ble dekket av borrelås. Tårnet danner dermed en stabil plattform, samtidig som lokket burde forhindre forstyrrelser fra en motstandersender.

Figur 3.11: Fleksibelt feste for posisjoneringstårn på toppen av roboten



Figur 3.12: Utføring av ledninger for kommunikasjon og strøm

3.4.2 Trianguleringssendere

For å kunne sende ut gjenkjennbare Ir-signaler til posisjoneringstårnet, er det nødvendig med 3 separate sendere. Spillebrettet er utstyrt med 4 plattformer til hvert lag, som kan brukes til dette. Hver plattform er 350 mm høy og man har lov til å plassere en sender på maksimalt 80x80x160 mm oppå denne.

Viktige designspesifikasjoner for senderne:

- Strømforsyningen må være enkel og liten, samt lett å bytte ut
- Senderen må kunne slåes enkelt av og på
- Diodene må være synlige fra hele spillebrettet
- Diodene må kunne operere på 450 kHz og i tillegg kunne identifiseres forskjellig fra hverandre

Tilsvarende sendere ble laget i forbindelse med et tidligere posisjoneringssystem, se Garsjø og Platou [7], men kun hardware til et av disse er tilgjengelig. Disse gav inspirasjon til det nye designet, men andre ønsker om funksjonalitet og utforming gjorde det nødvendig å starte på nytt.

Versjon 1

Styringen av senderen gjøres enklest ved hjelp av en mikrokontroller, og tidligere erfaring gjør det naturlig å lete blant Atmels mikrokontrollere. Senderene ble derfor basert på ATtiny13, en 8-bits AVR mikrokontroller fra Atmel som innehar tilstrekkelig funksjonalitet for dette. For å drive en stor nok strøm gjennom diodene, er det fornuftig å bruke en transistor som bryter for diodene. Denne styres ved hjelp av et pulsbreddemodulert (PWM) signal fra en av utgangene på mikrokontrolleren.

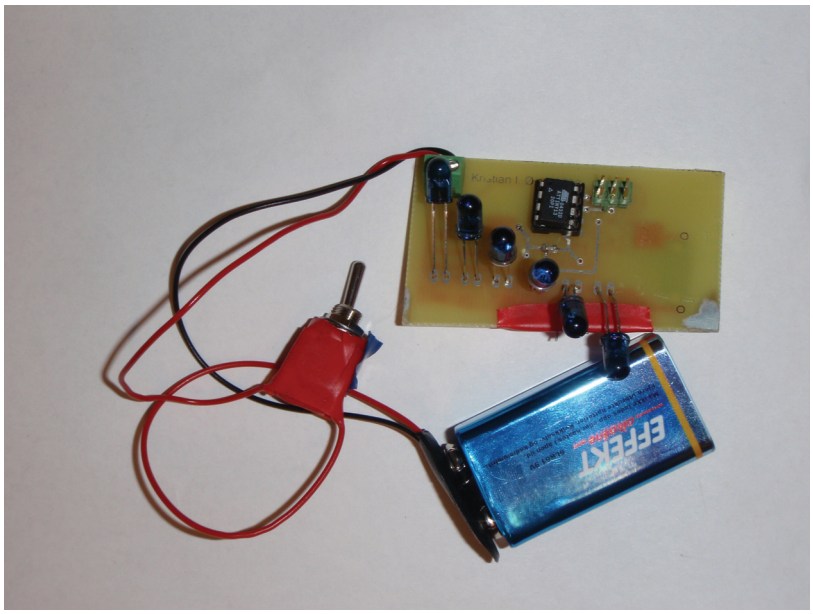
Ir-diodene ble valgt ut fra at de kunne operere på 450 kHz og at de hadde riktig bølgelengde i forhold til sensorene på posisjoneringstårnet. Ved lesing av databladet til sensorene, viste det seg at TSAL6400 passet perfekt til formålet. Ir-diodene har begrenset spredning, og det ble derfor besluttet å bruke 6 stk på hver av de 3 senderne. Disse må stilles inn i litt forskjellige vinkler for å dekke hele spillebordet, men det er også å foretrekke at de står mest mulig samlet for å unngå unødvendige feilkilder i systemet. Løsningen ble å sette alle diodene på en vertikal linje der hver av dem har litt forskjellig vinkel. For å kunne stille inn hver diode er de montert med det ene beinet over det andre, slik at de enkelt kan bøyes til den ene eller andre siden.

Siden sendemodulen må være selvforsynt, trengs det en enkel strømforsyning. Et vanlig 9-volts batteri ble vurdert som gunstig, med tanke på at en del dioder skal forsynes i tillegg til mikrokontrolleren, og at batteriene lett skal

kunne byttes ut. Spenningen er også tilstrekkelig til å forsyne 3 dioder slik at man kan ha 2 rekker i parallell (tilsammen 6 dioder). En liten bryter på spenningsinngangen gjør det enkelt å slå kortet av og på.

For å spare plass ble det brukt en ISP-header til overføring av kode fra datamaskinen, istedenfor JTAG som er brukt på de fleste andre kort på roboten.

Ferdig sender er vist i figur 3.13.



Figur 3.13: Første versjon av sender

Versjon 2

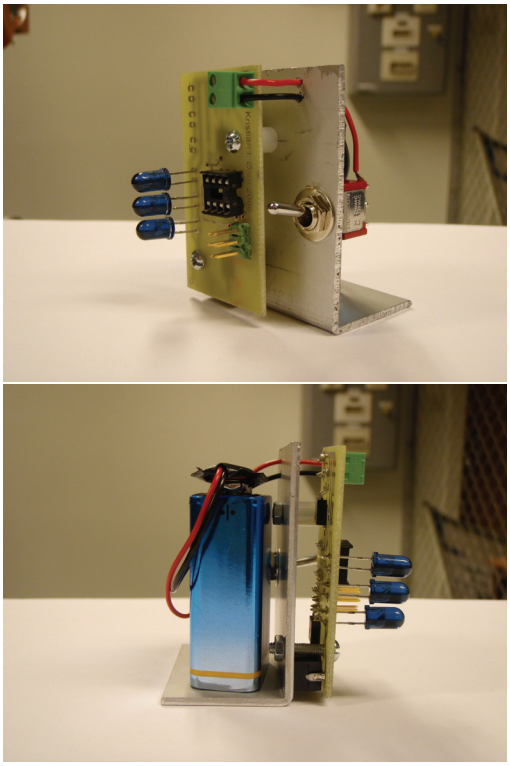
Under testing viste det seg at den første versjonen ikke var helt optimal. For det første klarte ikke transistoren å operere raskt nok, slik at strømmen slapp tilstrekkelig gjennom Ir-diodene. Dette ble løst ved å velge en transistor med noe mindre gate-kapasitans. Strømmen gjennom diodene viste seg også å være i overkant stor, slik at motstanden fort ble veldig varm og at målingene hadde unødvendig stor rekkevidde. Ved å sette på en noe større motstand og bytte over til en motstandstype som tåler høyere effekt, ble risikoen for at noe skulle gå i stykker redusert.

Erfaringsmessig gjorde mye testing med 3 ulike tårn det vanskelig å holde orden på batteristatus for hvert kort og hvilke kort som til enhver tid stod på. Derfor ble det innført en statusdiode på versjon 2, som lyser så lenge senderen er påslått. I tillegg er det laget en spenningsdeling med to motstander, som gjør at dioden lyser svakere etter hvert som batterispenningen faller.

Kretsskjema vises i vedlegg A og alle filer og kode finnes på vedlagte CD.

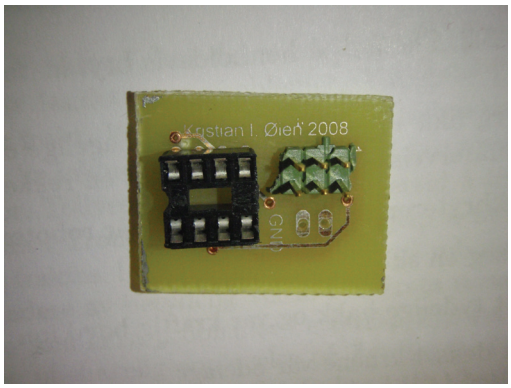
Senderfeste

En siste, men veldig viktig del av designprosessen, gikk ut på å feste kretskortet slik at det kunne stå stødig oppreist på plattformene rundt bordet. Det ble laget et feste i aluminium som var bøyd i en 90-graders vinkel. Kretskortet ble skrudd fast i denne, samtidig som bryteren ble innfelt ved siden av. For at dette skal stå stødig, brukes batteriets vekt som støtte på baksiden. Egen festeanordning og ledningsopplegg for 9-Volts kontakt, gjør det enkelt å skifte batteri raskt. Undersiden av senderfestet er dekket med den myke siden av en vanlig borrelås, slik at det enkelt kan festes og fjernes fra plattformene på spillebordet. Figur 3.14 viser en av de ferdige senderene.



Figur 3.14: Ferdig utviklet sender

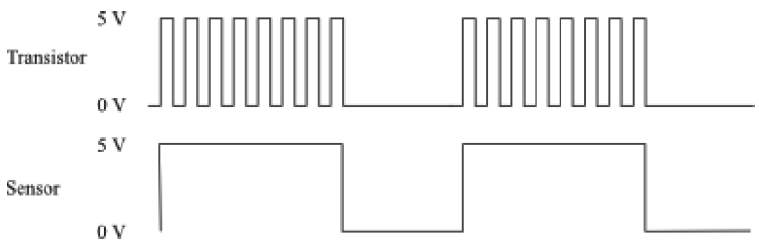
Siden det kan være litt vanskelig å komme til programmeringsinngangen når kretskortet er festet og koblet til, er det laget et lite ekstra programmeringskort, se figur 3.15. Mikrokontrollerne er festet med socket slik at de enkelt kan taes inn og ut, og det finnes reservemikrokontrollere som nå kan klargjøres separat med programmeringskortet.



Figur 3.15: Kort for programmering av sendermikrokontrollere

Software

Programmeringen av senderne er forholdsvis enkel, da det stort sett bare skal implementeres PWM-kode. Som tidligere nevnt er senderne nødt til å sende på ca. 450 kHz for at sensoren skal kunne oppfatte signalene. I tillegg må de ha et unikt sendemønster, slik at posisjoneringstårnet kan oppfatte hvilken sender det observerer. For å få til grunnfrekvensen på 450 kHz ble det benyttet et PWM-signal til styring av transistoren. Ved enkel testing kunne man lett verifisere at sensorens utgang gikk lav når den ble lyst på av denne Ir-strålingen. Videre ble det forsøkt å slå av og på PWM-utgangen i like intervaller slik at sensoren gav ut en ren firkantpuls som vist i figur 3.16. For at sensorkortet enkelt skal kunne detektere de ulike tårnene, ble det valgt å bruke ulike pulsbredder på signalene. Dvs. at PWM-utgangen står på i en like stor periode hver gang, slik at man kan måle hvor lenge sensorutgangen ligger høy etter hver stigende flanke. Etter en del testing ble perioden til de tre senderene satt til henholdsvis 20, 40 og 60 mikrosekunder. Dette er tilstrekkelig raskt til at posisjoneringstårnet kan oppfatte flere perioder av sendingen, selv om det snurrer forholdsvis raskt rundt. Programmering av hvert kort skjer enkelt ved å forandre på en variabel i koden. I tillegg er hvert enkelt kort fysisk merket med hvilken kode det kjører.



Figur 3.16: PWM-signalets innvirkning på IR-sensoren

3.5 Testing av triangulering

3.5.1 Testplan

Etter at hver enkelt funksjonalitet i posisjoneringstårnet var oppe, kunne hele systemet settes sammen til én enhet. Målet var å teste tårnet i realistiske situasjoner og sørge for at vinkelmålingene ble så robust som mulig. Under testingen har det også blitt lagt vekt på å forutse de mest sannsynlige situasjoner der roboten har behov for å oppdatere sin posisjon.

Følgende parametre kan variereres og har alle innvirkning på resultatene:

- Hastighet på sensorkortets rotasjon
- Antall korrekte pulsperioder på rad før en måling sendes fra sensorkortet
- Størrelse på vinkelmålingsbuffer i hovedkortet
- Antall godkjente målinger i bufferet før vinkelmålingene sendes til navigassjonssystemet

De to første parameterene henger nøye sammen, og vil bestemme hvor mange vinkelmålinger tårnet får per runde. I utgangspunktet virker det fornuftig å stille det inn slik at tårnet leser én vinkel per runde, samtidig som man ønsker at det skal klare å holde så stor fart som mulig. Disse verdiene ble tunet i forkant av testingen, på i overkant av 3 meters avstand (tilsvarende den største aktuelle målingsavstanden på bordet). Sluttverdiene på disse parameterne gav minst én måling per runde, noen ganger to, litt avhengig av hvilken av de tre senderene som ble brukt. Hastigheten på tårnet var ca 3 runder per sekund, som tilsier at det minst vil ta i overkant av 1,5 sekunder å fylle et buffer på 5 vinkelmålinger.

De to siste parameterne er også overlappende i forhold til nøyaktighet versus tidsbruk. Det er valgt å sette verdien på disse like under testene, da det allerede er satt krav til hvilke målinger som får settes inn i bufferet. Det ble gjort to typer hovedforsøk, der bufferstørrelser på henholdsvis 5, 10 og 15 vinkelmålinger ble brukt.

1. Posisjonering ved hjelp av ett tårn ved tømming, som beskrevet i avsnitt 3.1.3 og figur 3.6.
2. Posisjonering når roboten er plassert et tilfeldig sted i nærheten av midten på bordet.

I begge tilfellene blir roboten plassert i en kjent, oppmålt posisjon på bordet. Det blir i tillegg gjort en tilsvarende simulering i Matlab, som gir de teoretisk utregnede vinklene. Til slutt blir målingene initiert med riktig CAN-melding, og vinkelmålingene blir registrert som resultater.

Bufferstørrelse	Måling 1	Måling 2	Måling 3	Måling 4	Måling 5
5	170°	170°	171°	170°	171°
10	169°	170°	171°	170°	170°
15	170°	171°	171°	169°	169°

Tabell 3.1: Vinkelmålingsresultater fra test 1

3.5.2 Testresultater

Test 1

Roboten er plassert 50 cm fra venstre kant av spillebordet, noe som i følge ligning (3.1) tilsvarer en vinkel, λ_3, på 166°. Til hver bufferstørrelse er det utført 5 vinkelmålinger for å kunne se om målingene er stabile. Resultatene er vist i tabell 3.1.

Det er lett å se at målingene holder seg veldig stabile, uavhengig av bufferstørrelse, noe som tyder på at en bufferstørrelse på 5 er tilstrekkelig. Likevel er resultatet ganske feil i forhold til den teoretiske vinkelen på 166°, da gjennomsnittet av alle målingene er 170,13°(avvik på 4,1°). Dette kan forklares ved at sensoren hele tiden er i bevegelse og at det derfor vil bli en liten forskjell på når sensoren får målingene og når vinkelen blir avlest fra hovedkortet. Det er viktig å merke seg at hver grad feil utgjør rundt 3-4 cm i dette tilfellet, så en korrigering bør absolutt gjøres for det faste avviket. Det ser imidlertid ut til at en målevariasjon på ±1°er nødt til å godtas med dette måleoppsettet.

Sender 1:

Bufferstørrelse	Måling 1	Måling 2	Måling 3	Måling 4	Måling 5
5	19°	19°	18°	19°	18°
10	17°	18°	20°	19°	19°
15	19°	20°	20°	19°	19°

Sender 2:

Bufferstørrelse	Måling 1	Måling 2	Måling 3	Måling 4	Måling 5
5	255°	255°	256°	256°	255°
10	256°	256°	256°	257°	257°
15	257°	256°	257°	255°	255°

Sender 3:

Bufferstørrelse	Måling 1	Måling 2	Måling 3	Måling 4	Måling 5
5	89°	91°	89°	90°	91°
10	90°	91°	90°	91°	91°
15	91°	91°	92°	91°	91°

Tabell 3.2: Vinkelmålingsresultater fra test 2

Test 2

Roboten ble i denne testen plassert tilfeldig på spillebordet, mens senderoppsettet for rødt lag ble brukt som utgangspunkt for trianguleringen. Posisjonen som ble valgt var :

$$x = 160cm$$
$$y = 70cm$$
$$\psi = 60°$$

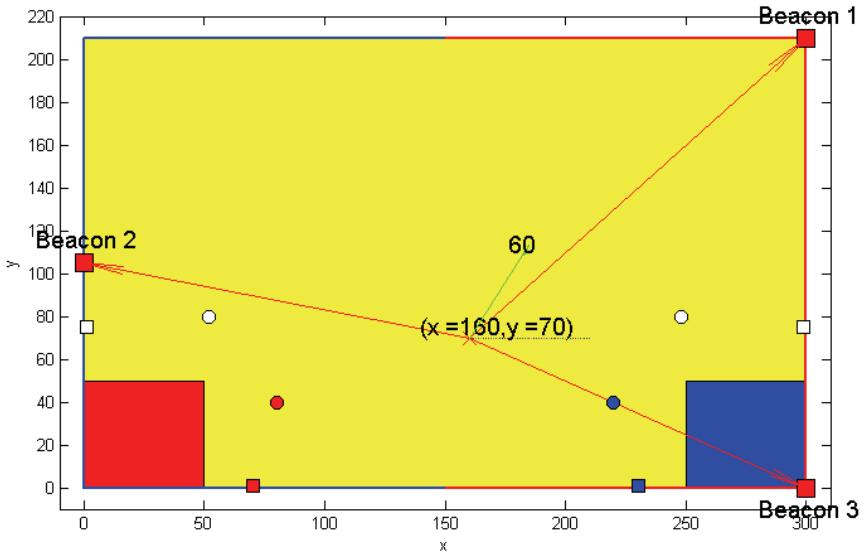
Det ble først foretatt en simulering av denne situasjonen ved hjelp av Matlab, se figur 3.17. Resultatet av simuleringen gav følgende sendervinkler:

- Sender 1: 15°
- Sender 2: 252°
- Sender 3: 87°

Videre ble det foretatt en tilsvarende oppmåling som i den første testen, bortsett fra at det denne gangen ble gjort med alle tre tårnene. For å få sjekket nøyaktigheten på hvert enkelt tårn, ble vinklene i denne testen målt hver for seg. I praksis vil alle tre gjøres samtidig for å spare tid. Tabell 3.2 viser alle vinkelmålingene for de tre senderne.

Igjen kan man se at målingene stort sett ligger ganske stabilt, med en variasjon på ±1°. I punktene under er det tatt gjennomsnittet av alle målingene for hver sender og i tillegg satt opp avviket fra de utregnede vinklene i punktene lenger opp.

- Sender 1: 18,9°, avvik: 3,9
- Sender 2: 255,9°, avvik: 3,9
- Sender 3: 90,6°, avvik: 3,6



Figur 3.17: Matlab-simulering av test 2

Oppsummering av testresultater

Resultatene i de to testene er totalt sett veldig sammenfallende. Det virker som vinkelmålingene konsekvent blir ca. 4°feil og at de har en variasjon på rundt ±1°. Som nevnt i test 1 skyldes nok dette rotasjonen på tårnet, slik at vinkelmålingen skjer en kort tidsforsinkelse etter at sensoren har mottatt signal fra de respektive senderne. Heldigvis virker denne tidsforsinkelsen å være veldig stabil, og man kan derfor korrigere for denne i software. I tillegg kan navigasjonssystemet bruke gjennomsnittet av flere vinkelmålinger, avhengig av hvor lenge man står stille. Målingene ser uansett ut til å bli ganske robuste, og burde være tilstrekkelig nøyaktige til å kunne foreta flere posisjonsoppdateringer i løpet av en match på 90 sekunder. Til slutt en oppsummering av de endringene som ble gjort på grunnlag av testingen:

- Bufferstørrelsen på vinkelmålinger i posisjoneringstårnet ble fastlagt til 5
- Alle vinkelmålinger blir korrigert med et tillegg på 4°før de blir sendt til navigasjonssystemet
- Navigasjonssystemet vil samle opp så mange målinger det er tid til, og bruke gjennomsnittet av disse til utregning av endelig posisjon

3.6 Konklusjon

I arbeidet presentert i dette er det blitt utviklet et velfungerende absolutt posisjoneringssystem. Dette er ment som støtte til odometrien da denne har en tendens til å drifte over tid. Likevel har systemet noen svakheter i forhold til at det tar litt tid å foreta målinger og at roboten helst bør stå stille. Testingen viser imidlertid at målingene blir presise og på den måten kan gi en god korrigering av odometrien som fungerer veldig bra på kortere avstander. Det vil høyst sannsynlig forekomme situasjoner under konkurransen der det er tid til å foreta en absolutt posisjonsoppdatering.

Kapittel 4

Navigasjonssystem

Navigasjonssystemet er bygget opp i C++, og er i stor grad basert på fjorårets Eurobot-prosjekt og arbeidet beskrevet i Mørkrid og Øien 2007 [4]. Etter mange modifikasjoner og utvidelser er hensikten med dette kapittelet å gi en overordnet beskrivelse og dokumentasjon av det systemet som ble brukt under konkurransen i Heidelberg. Systemet er generelt bygget opp og bør absolutt være mulig å bygge videre på.

Navigasjonssystemet har som hovedoppgave å styre roboten rundt på spillebordet. Som hjelpemiddel brukes ulike posisjoneringssystemer, henholdsvis odometri og triangulering, beskrevet i kapittel 3.

4.1 Overordnet struktur og teori

Oppbygningen til navigasjonssystemet er ganske enkel og består i hovedsak av å løse to oppgaver:

- Til enhver tid holde orden på og oppdatere det aktuelle posisjonsestimatetet for roboten på spillebordet
- Utligne forskjellen på nåværende og ønsket posisjon ved hjelp av et regulatorhierarki

4.1.1 Posisjonsoppdateringer

Uansett hvilken tilstand roboten er i, enten den kjører eller står stille, må navigasjonssystemet oppdatere og ha kontroll på den aktuelle posisjonen. Dette foregår i all hovedsak ved at det mottas odometrimålinger kontinuerlig. Disse målingene blir integrert opp i et eget objekt, *OdometryObservation*, og kan hentes ut ved behov. Hovedløkken i navigassjonsystemet leser ut og resetter

disse målingene omtrent 10 ganger i sekundet, og bruker dem til utregning av relativ posisjon som videre blir lagt til det overordnede posisjonsestimatet for roboten.

I tillegg til odometrien vil navigassjonssystemet med ujevne mellomrom bli bedt om å foreta absolutte posisjonsmålinger ved hjelp av posisjoneringstårnet, beskrevet i kapittel 3. Så lenge roboten står stille blir det foretatt målinger, dvs. helt til AI gir beskjed om at roboten skal bevege seg igjen. Alle vinkelmålingene blir samlet opp i et array og til slutt blir gjennomsnittet brukt som grunnlag for utregning av den absolutte posisjonen. Algoritme 4.1.1 viser en forenkling av hvordan dette gjøres i koden. Denne typen posisjonsmåling overstyrer det aktuelle posisjonsestimatet direkte og blir grunnlag for all videre navigasjon.

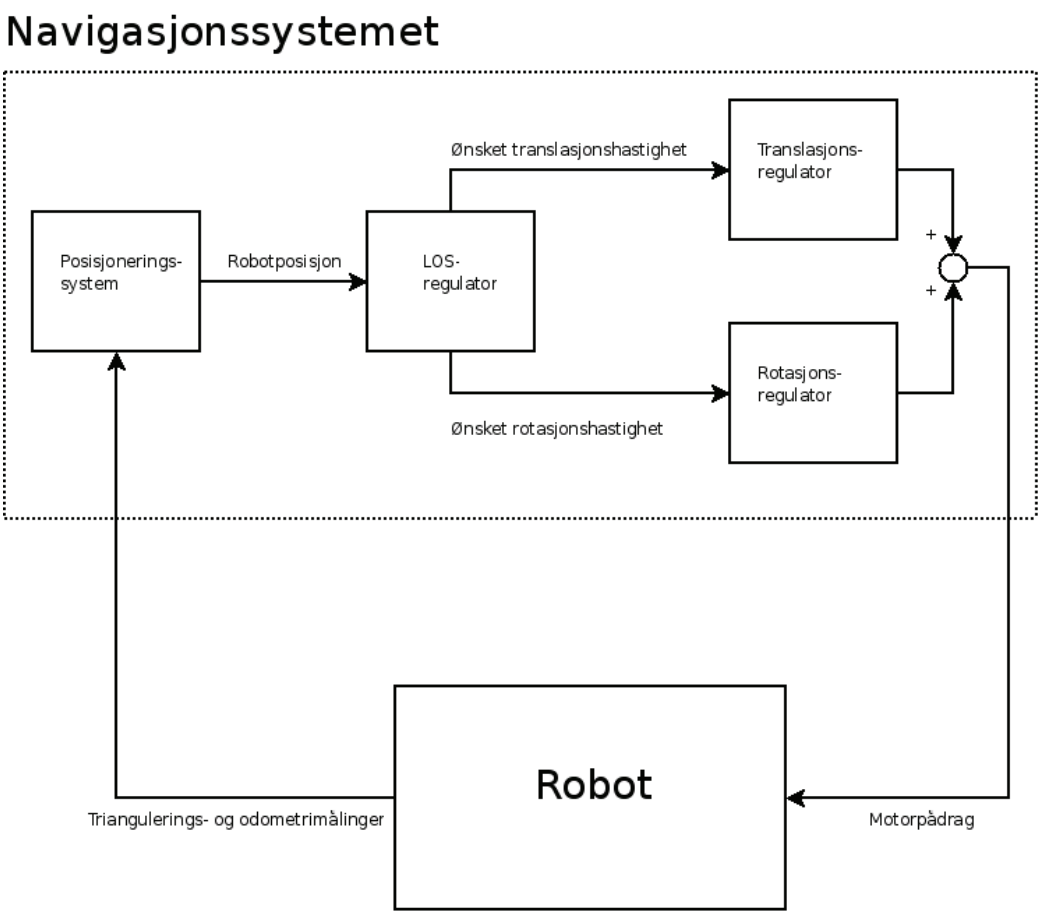
Algorithm 4.1.1: POSISJONSMÅLING(*RobotModel.cpp*)

```
while doingMeasurements
  do
{ calculateCurrentBeaconAngles;
  sendMeasurementRequest;
  while !measurementReceived
    do
    { if robotMoving
        then { doingMeasurements = false;
               numberOfMeasurements = 0;
               break;
    if newMeasurement
        then { numberOfMeasurements++;
               findAverageFromMeasurements;
               calculatePosition;
               updateCurrentPositionEstimate;
```

4.1.2 Regulering av posisjonsavvik

Det er opprettet et eget objekt, *LOSGuidance*, som har hovedansvaret for å regne ut hvilken bevegelse man ønsker for at posisjonsavviket skal reduseres. Ved å inspisere nåværende posisjon i tillegg til ønsket posisjon, som regel i form av neste waypoint, regner modulen ut avvik i avstand og vinkel. En avansert regulator basert på "Line of sight"-teori (LOS), viderebrukt fra Kjemphol og Knausgård 2006 [1], beregner ønsket translasjons- og rotasjonshastighet. Utenfor dette objektet, blir de ønskede hastighetene sendt videre inn i to separate PID-regulatorer som beregner hver sin andel av pådraget til motorene. Når dette pådraget er sendt til motorene, begynner sløyfen på nytt. Man kan si at LOS-regulatoren utgjør en ytre sløyfe i reguleringssystemet, mens

PID-regulatorene for henholdsvis translasjons- og rotasjonsregulering utgjør en indre sløyfe. Oppsettet illustreres i figur 4.1.



Figur 4.1: Reguleringsstrukturen i navigasjonssystemet

Når posisjonsavviket er mindre enn gitte grenser, sørger LOS-regulatoren for at waypointets ønskede slutt-heading blir oppnådd før det til slutt sendes melding om at waypointet er nådd. Alle regulatorer blir nullstilt mhp. intergralvirkning ol. slik at navigasjonssystemet kan fortsette med neste waypoint i køen, eller bare vente på videre beskjed.

4.2 Implementering av ekstra funksjonalitet

Utenom noen mindre forandringer som er gjort på hovedstrukturen i navigasjonssystemet er det innført ny funksjonalitet, som gir helt nye muligheter for kontroll. Systemet har blitt mer fleksibelt for å kunne håndtere nye situasjoner i forbindelse med konkurransen.

4.2.1 Innføring av rygging

En begrensning ved den LOS-baserte reguleringsalgoritmen, er at den aldri velger å rygge til et waypoint. Det er kun definert én retning med framdrift, men i noen sammenhenger kan det være hensiktsmessig at roboten rygger rett bakover isteden for først å måtte snu rundt. I enkelte tilfeller er dette også helt nødvendig, f. eks. når roboten står inntil kanten av bordet i forbindelse med plukking/utslipping av baller. For å gjøre dette mulig er det lagt inn støtte for å angi at roboten skal rygge til et spesielt waypoint. Hvis navigasjonssystemet får beskjed om å rygge, vil det sørge for at roboten setter pådrag bakover mot punktet istedenfor å kjøre framover.

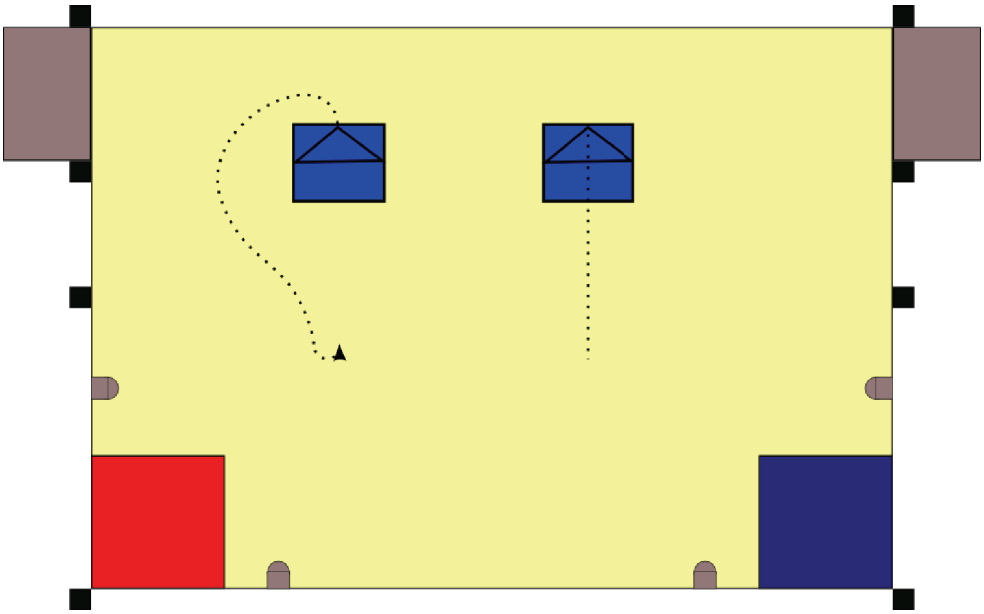
Dette er løst ved å legge til 180°på headingen slik at algoritmen tror at roboten hele tiden står motsatt vei. Samtidig blir alle motorhastigheter invertert slik at de går motsatt vei av det algoritmen regner ut. På denne måten vil roboten kjøre like raskt og presist som ved vanlig kjøring, bare at den nå rygger istedenfor å kjøre framover. Til slutt ble det også nødvendig å gjøre en endring slik at vinkelopprettingen ved ankommet waypoint ble riktig i forhold til ønsket slutt-heading.

Ryggefunksjonaliteten er blitt grundig testet og har etter hvert vist seg å bli helt nødvendig for robotens funksjonalitet. Det er viktig å merke seg at rygging angis som en uavhengig egenskap ved et waypoint og at alle relative og absolutte posisjoner skal angis på samme måte som ved vanlige waypoints.

Et eksempel på hvor besparende ryggingen kan være, er illustrert i figur 4.2. I begge tilfellene har roboten fått et relativt waypoint med koordinatene $x = -1$ m, $y = 0$ m og $\psi = 0°$, men forskjellen er at roboten til høyre har blitt bedt om å rygge. Banen til venstre er omtrentlig gjenskapt etter å ha observert denne testen på bordet, og er et resultat av LOS-systemets utregninger. Ryggemanøveren sparer mye tid, samtidig som odometrien klarer seg bedre og posisjonsestimatet dermed holder seg mye mer presist.

4.2.2 Ulike typer waypoints

All manøvrering på bordet, med unntak av reguleringsmetodene beskrevet i avsnitt 4.2.4, skjer ved at det legges til et waypoint. I utgangspunktet blir waypointene lagt i en kø, slik at roboten kjører til dem i samme rekkefølge som de blir lagt inn. Det er imidlertid også gitt mulighet til å avbryte gjeldende waypoint ved å legge inn nye waypoints foran i køen. Alle de ulike oppgavene som skal løses krever støtte for flere typer manøvrering. Det er valgt å løse dette problemet ved å innføre ulike typer waypoints slik at den øvrige strukturen i navigassjonssystemet blir beholdt. Resultatet ble til slutt følgende waypoint-typer:



Figur 4.2: Eksempel på ryggemanøver

1. Vanlige waypoints

Disse blir stort sett brukt hver gang roboten skal forflytte seg til et punkt på bordet så fort og presist som mulig.

2. Waypoints som skal kjøres til med en valgt hastighet

Noen ganger ønsker man ikke å kjøre med full hastighet fram mot et waypoint, f.eks. når man skal søke med kamera underveis. Denne typen tilbyr muligheten for å angi en makshastighet under kjøringen.

3. Waypoints med gyldighetssjekk

I utgangspunktet har det blitt bestemt at AI har ansvaret for å sjekke at alle waypoints er gyldige. Dette fordi gyldighetskriteriet har vist seg å variere ut fra forhold som AI har kontroll over. Imidlertid ønsker AI noen ganger at navigasjonssystemet skal sjekke de waypointene som er oppgitt i forhold til robotens posisjon, og i disse tilfellene kan waypoints med gyldighetssjekk brukes. Et gyldig waypoint er først og fremst et waypoint som ligger innenfor spillebordet, men det er også viktig å huske på at roboten har en utstrekning. Siden posisjonsmålingens midtpunkt ligger litt foran på roboten, vil utstrekningen være avhengig av om roboten rygger eller kjører fremover. Dette

er tatt med i gyldighetssjekken, i tillegg til en liten margin i alle retninger. Det er også tatt hensyn til at det stikker ut dispensere flere steder på bordet.

4. Ryggewaypoints

Som nevnt er ryggeegenskapen uavhengig av selve waypointet og er derfor strengt tatt ikke en egen type. Likevel brukes type-feltet til å angi dette. Alle waypoints med negativ type betyr rygging, dvs. at et ryggewaypoint med gyldighetssjekk angis med type -3, ihht. waypointnummereringen over.

Etter utvidelsen har et waypoint følgende egenskaper:

ID - en unik identifikasjon av hvert waypoint

TYPE - angir waypoint-typen, se ovenfor

X - x-posisjon på spillebordet

Y - y-posisjon på spillebordet

PSI - ønsket sluttheading i waypointet

CIRCLE OF ACCEPTANCE - tolleranse for posisjonsavvik

4.2.3 Overstyring av posisjonsestimatet

Ettersom AI har god kontroll og oversikt over alt roboten foretar seg, har den også mulighet til å overstyre navigasjonssystemets posisjonsestimat til enhver tid. Et eksempel på dette er at AI setter startposisjonen, siden det kun er denne som vet hvilken side av bordet vi starter på. I tillegg har den kontroll på vinkel og x- eller y-posisjon når roboten står inntil en kant. I noen av tilfellene har AI bare ønske om å oppdatere én eller to av frihetsgradene til roboten, f.eks. x-posisjonen og retningen, og det er derfor lagt inn mulighet for å ignorere parametre under posisjonsoppdateringen. Navigassjonssystemet vil dermed beholde sitt estimat på disse, mens den overstyrer de andre. Dette systemet gjør det mulig for alle slags moduler, som kamera og sensorer, å komme med innspill til det sentrale posisjonsestimatet dersom dette skulle finnes ønskelig.

4.2.4 Regulering av avstand mot kant

Slik årets oppgave er lagt opp er det nødvendig for roboten å kunne kjøre helt inn mot kanten av spillebordet. For å kunne bruke de oppsatte balld-ispenserne trenger roboten å rette seg inn omtrent 10 cm fra kanten, mens man ved tømming av baller er nødt til å kjøre helt inntil kanten. Ved vanlig

posisjonering av roboten kan det bli noe unøyaktig å bruke et waypoint. Dersom posisjoneringen blir litt feil, kan man risikere at roboten blir stående feil i forhold til kanten, eller i verste fall at den blir stående å kjøre inn i kanten av bordet. Som løsning på dette er det innført to enkle avstandsregulatorer i navigasjonssystemet.

Avstandsregulering ved hjelp av avstandssensorer

For å kunne stoppe opp en viss avstand fra kanten er det plassert to avstandssensorer lavt nede, foran på hver side av roboten. Sensorene som brukes er av samme type som antikollisjonssensorene, se kapittel 6. Målingene sendes til AI via antikollisjonssystemet som er koblet til CAN-bussen. AI initierer avstandsreguleringen ved å sende en egen POSIX-melding til navigasjonssystemet, etterfulgt av kontinuerlige avstandsmålinger fra sensorene. Navigasjonssystemet benytter en egen P-regulator og regulerer roboten inn til ønsket avstand. Differansen mellom målingene brukes til å regulere rotasjonshastighet og dermed vinkel, samtidig som avstanden til kanten justeres etter ønske. Navigasjonssystemet sender til slutt melding til AI om at avstand og vinkel er korrekt.

Ønsket translasjonshastighet, u_d, og rotasjonshastighet, r_d, blir regulert på denne måten.

$$u_d = K_u(e_l + e_r)$$
$$r_d = K_r(e_l - e_l)$$

der K_u og K_r er forsterkningen i regulatorene for henholsdvis translasjons- og rotasjonshastighet, mens e_l og e_r er avviket mellom ønsket og gjeldende avstand til vegg for henholdsvis venstre og høyre avstandssensor.

Innkjøring til kant ved hjelp av endebrytere

Ved levering av baller, er det nødvendig å stå helt inntil kanten. For å få til dette er det plassert en endebryter på hver side av fronten på roboten. Disse blir trykket inn når roboten treffer kanten og et interruptsignal blir sendt via CAN-bussen til AI. Bryterne brukes av AI på samme måte som avstandssensorene, bortsett fra at avstandsavvikene, e_l og e_r, er satt til en fast verdi inntil bryterne blir trykket inn. Roboten vil dermed ha samme hastighet helt inn mot veggen.

4.2.5 Mulighet for overstyring av makshastighet

En generell utfordring for bevegelse av roboten på spillebordet er hvilke hastigheter man skal operere med. I utgangspunktet har hardwaren kapasitet til å kunne kjøre på over 30 V motorspenning, noe som antagelig vil tilsvare hastigheter på godt over 1 m/s. Utfordringen er imidlertid at kjøringen må foregå kontrollert. Slik posisjonerings- og antikollisjonssystemet er lagt opp nå, har vi ingen mulighet til å kunne operere kontrollert på spillebordet med slike hastigheter. I all hovedsak er det presisjonen til antikollisjonssystemet som begrenser dette, men det er også usikkert hvor høye hastigheter posisjoneringssystemet tåler. I utgangspunktet er motorene strupet til å kjøre på maks 8 V, med en oppløsning på $2^8 = 256$ nivåer som navigasjonssystemet kan sette. Etter ønske om å kunne øke denne maksgrensen i gitte situasjoner er det innført en ny motorpådragsgrense i software. Samtidig er strupingen i hardware doblet til 16 V, og det er innført et utvidet grensesnitt som tillater $2^9 = 512$ nivåer av pådrag. Ved å sette maksgrensene til 256 vil systemet oppføre seg eksakt som tidligere, men det er nå mulighet for dynamisk å sette høyere hastighet i situasjoner der man har kontroll og ønsker å tillate høyere hastighet.

4.3 Resultater og konklusjon

Siden navigassjonssystemet har blitt utvidet underveis i utviklingen, har det ikke vært gjennomført egen systematisk testing av de ulike nyvinningene. Alle funksjonene har imidlertid blitt utprøvd og verifisert ved implementering og systemene er stort sett såpass enkle at man har god oversikt over hva som skjer. Den virkelige testingen har vært gjort i forbindelse med all prøvekjøring av totalsystemet, der hver enkelt funksjon har blitt benyttet mange ganger i alle mulige slags sammenhenger. Noen opplagte småfeil har blitt funnet, men dette har stort sett vært enkelt å rette opp.

Totalt sett har navigasjonssystemet oppført seg meget presist og robust. Programvaren har aldri feilet direkte og stort sett gitt roboten fornuftig og korrekt oppførsel på bordet. Tuning av regulatorene i systemet utgjør selvfølgelig en stor utfordring og her er det nok ennå noe å hente, ikke minst på manøvrering ved høyere hastigheter.

Kapittel 5

AI - Kunstig intelligens

AI (kunstig intelligens) utgjør en sentral brikke i styringen av roboten; planlegging av arbeidsoppgaver. Det er mange måter å lage en AI på, helt fra en veldig enkel og sekvensiell gjennomføring til en dynamisk og intelligent AI. Dette kapittelet vil presentere arbeidet som er gjort med implementering og testing av årets AI.

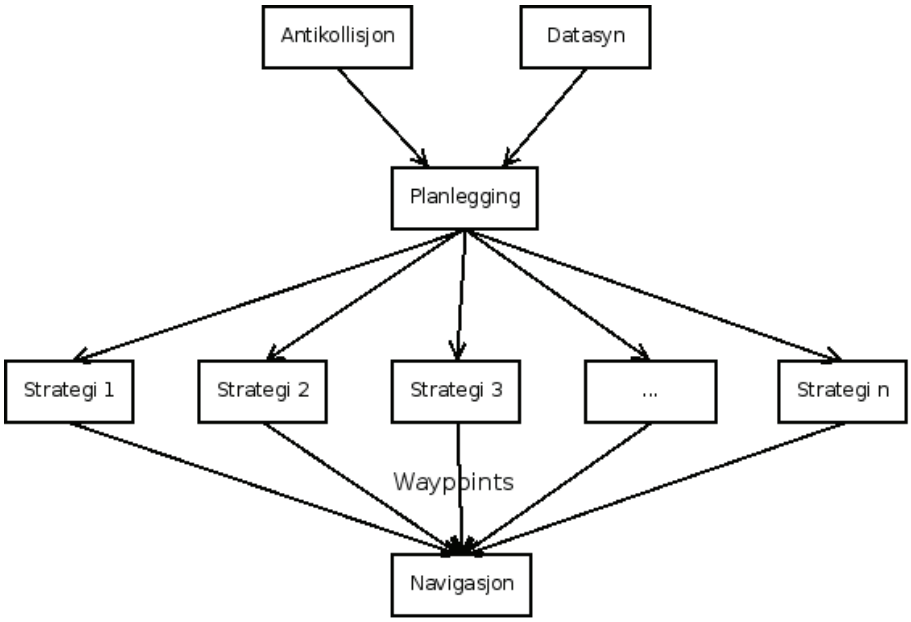
5.1 Bakgrunn

Teorien bak og valg av metode for implementering av AI er beskrevet i prosjektrapporten skrevet av Mørkrid og Øien høsten 2007 ([4]). Denne våren har all fokus vært rettet mot implementeringen. Implementasjonen har blitt en realisering av figur 5.1, som viser skjematisk hvordan AI-strukturen ble definert i prosjektrapporten. Koden er skrevet i C++, der hver strategi er et objekt. AI består av en hovedfil (*planner.cpp*) som inneholder en while-løkke og en timer-tråd. Denne while-løkken representerer den overordnede styringen.

5.2 Overordnet styring

Hovedfilen, *planner.cpp*, har den overordnede styringen og omtales som *Planlegging* i figur 5.1. Denne filen er ment å være generell og skal kunne benyttes i kommende års konkurranser, da det lett kan legges til/fjernes strategier som er tilpasset oppgaven. Den er bygd opp på følgende måte:

- Posix-meldingskøer for kommunikasjon med navigasjonssystemet, gateway og kamera blir opprettet.



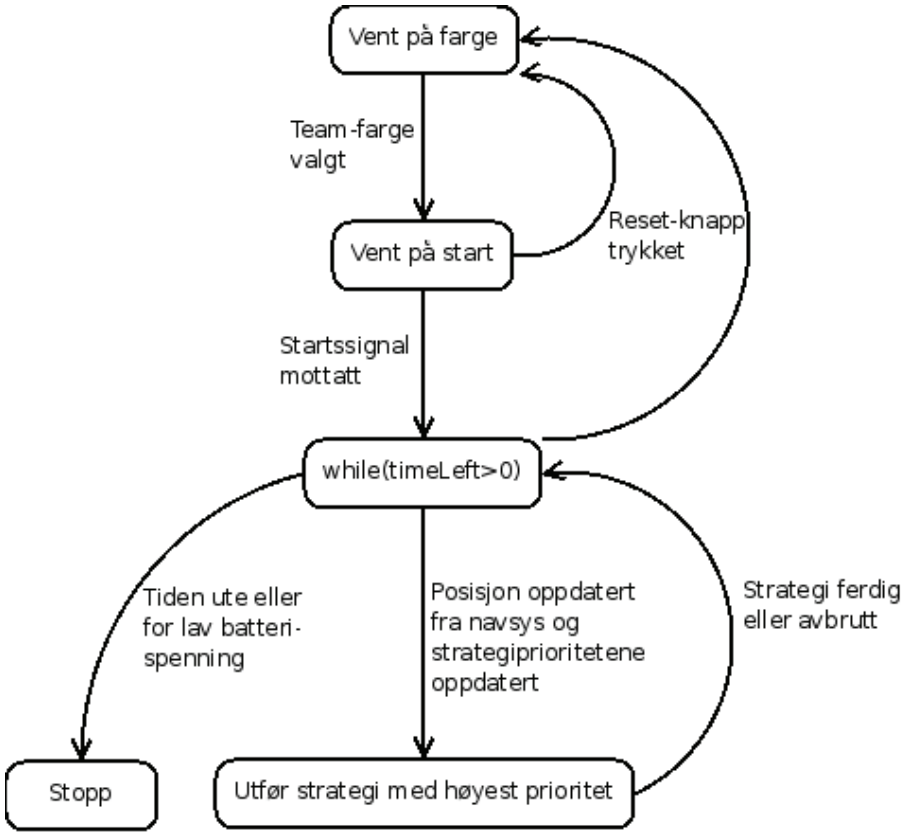
Figur 5.1: Skjematisk oppbygging av AI.

- En startkommando blir sendt til gateway for å gjøre systemet klart til start.
- Systemet venter på valg av farge.
- Initialposisjonen blir sendt til navigasjonssystemet.
- Strategiene blir lagt til AI.
- Timeren blir klargjort og systemet venter på startmelding.
- En while-løkke går så lenge tiden ikke er ute, eller så lenge ikke batterispenningen er under den grensen som er satt.

En skjematisk oppsummering av punktene over er vist i figur 5.2.

While()-løkken består hovedsaklig av 6 hovedpunkter:

- **Sjekker for reset-melding**: Dersom reset-knappen på roboten er trykket, skal man gå inn i en ventetilstand der det på nytt ventes på team-farge og startmelding. I tillegg resettes alle moduler på roboten. Dersom reset er trykket mens en strategi er aktiv, vil strategien bli avbrutt og reset-rutinen vil bli kjørt.
- **Ber om posisjon fra navigasjonssystemet**: Ber om posisjonen for å kunne oppdatere strategiprioritetene med den mest oppdaterte posisjonen.



Figur 5.2: Figuren viser de grunnleggende tilstandene i AI. tilstandsmaskinen beskriver i korte trekk innholdet i planner.cpp.

- **Oppdaterer prioritetene til strategiene**: Prioritetene oppdateres på bakgrunn av de variabler som definerer prioriteten. Dette er nærmere beskrevet under.
- **En strategi velges på bakgrunn av prioritetene**: Den strategien med høyest prioritet blir valgt. Dersom to strategier har like høye prioriteter, vil den som ligger først i vektoren bli valgt. Derfor er antikollisjonsstrategien lagt først i vektoren, slik at denne skal bli valgt først dersom en annen strategi har like høy prioritet. Antikollisjonsstrategien kan også lett gis en høyere prioritet enn alle andre strategier.
- **Den valgte strategien blir kjørt**: *PerformStrategy()*-funksjonen i den valgte strategien blir kjørt helt til den blir avbrutt eller har kjørt ferdig.
- **Til slutt sjekkes det om batterispenningen er for lav**: Dersom batterispenningen er for lav, vil spenningsvarslerkortet sende en CAN-melding til gateway, som videresender denne til AI. Når AI mottar en slik melding, vil variabelen *lowBattery* settes til **TRUE**.

Pseudokoden under viser i korte trekk hvordan programmet ser ut.

Algorithm 5.2.1: PLANLEGGER(*planner.cpp*)

```
while true
  do
  { if reset
      then { stop robot;
             wait for team color;
             wait for start message;
             send init position;
             reset strategies;

    request position;
    update strategies;
    get all priorities;
    perform the strategy with the highest priority;

    if lowbattery
      then {shut down the system;
```

5.2.1 Timer

Timeren kjøres som en egen Posix-tråd. Denne passer på å sende stopp-melding til gateway når tiden er ute. I tillegg er det lagt inn en sikkerhetsmargin på 1 sekund slik at systemet vil stoppe etter 89 sekunder i stedet for 90.

5.3 Oppbygning og filstruktur

Det første som skjer når AI startes er at det opprettes et *PlansysObservation*-objekt. Dette objektet inneholder en vektor (*observers*) som lagrer unna alle strategi-objektene. Når et strategiobjekt blir opprettet, blir det lagt til i *observers*-vektoren. Både *PlansysObservation*-objektet og strategiobjektene arver fra *Observation*-klassen, som inneholder metoder som er felles for alle strategiene. Dermed kan man via *observers*-vektoren kalle disse metodene på alle strategiene. *PlansysObservation* inneholder også alle variabler som er felles for alle strategiene.

PosixCommunication.cpp sørger for kommunikasjonen mellom AI og henholdsvis gateway og navigasjonssystemet. Som navnet tilsier er denne kommunikasjonen basert kun på Posix-meldingskøer.

Balls.cpp er en *ball*-klasse som har oversikten over antall baller som er i dispenserne, hvor mange som befinner seg i roboten og hvor mange fargede og hvite baller som skal plukkes opp. I tillegg lagres fargen på den sist sorterte ballen.

Waypoints.cpp inneholder alle fast posisjonerte waypoints, disse lagres i en waypoint-array. Med fast posisjonerte waypoints menes de waypoints som definerer plasseringen til fast posisjonerte baller, dispensere, avleveringsrenne osv.

5.3.1 Prosjektspesifikke filer og metoder

Mye av AI har blitt laget spesifikt mot årets konkurranse, men det er meningen at strukturen skal være på plass slik at det prosjektspesifikke enkelt skal kunne byttes ut med nye prosjektspesifikke filer og metoder. Alle strategiene og *balls*-klassen er laget med tanke på Eurobot 2008. Selve strukturen på strategiene derimot, kan gjerne brukes videre.

5.3.2 Generelle filer og metoder

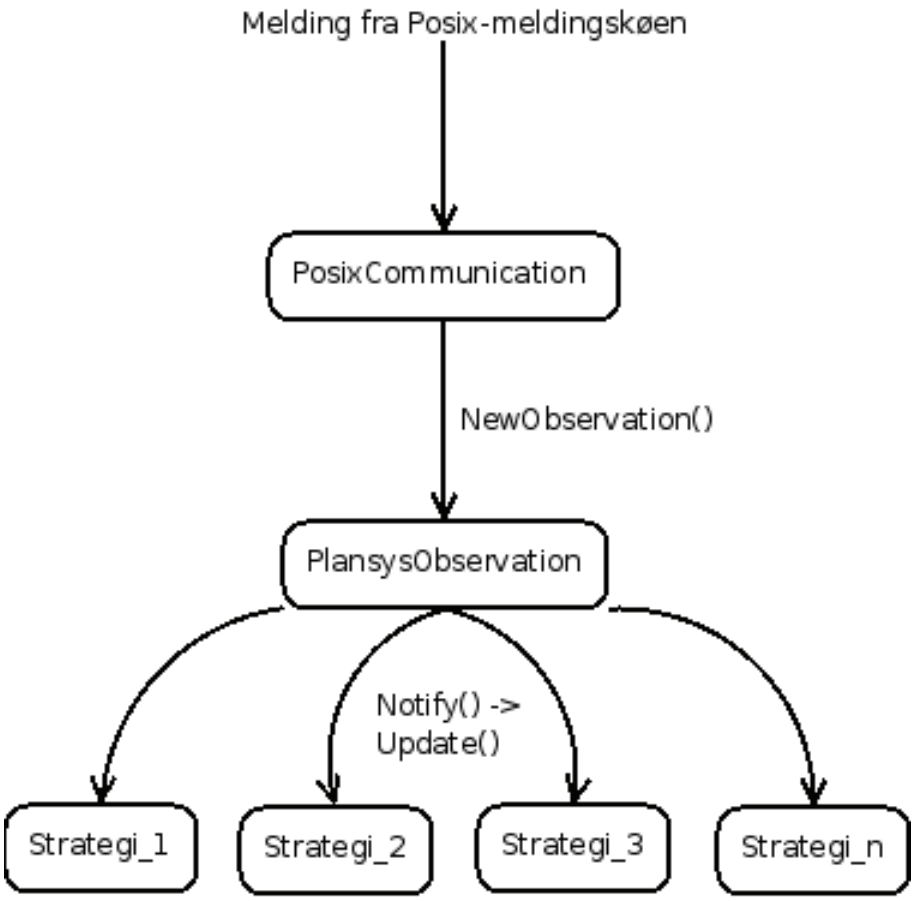
PlansysObservation-klassen inneholder en del variabler og metoder som er spesifikke i forhold til årets konkurranse, men klassen i seg selv er generell og kan lett brukes videre. Det samme gjelder *PosixCommunication* og skjelettet til strategiene.

5.3.3 Kommunikasjon via Posix-meldingskøer

Når AI mottar Posix-meldinger i *PosixCommunication.cpp*, vil disse mottaksfunksjonene kalle *newObservation*-funksjonen i *PlansysObservation.cpp*. Denne funksjonen oppdaterer de relevante variablene i objektet, og informerer strategiobjektene om dette via *Notify()* som kjøres fra Observation-klassen. *Notify()*-funksjonen kjører *Update()* på alle strategiene som ligger i *observers*-vektoren. På denne måten oppdateres alle strategier til en hver tid. Sekvensen er vist i figur 5.3.

5.4 Strategiene

AI-strukturen er basert på bruk av strategier som beskrevet i Mørkrid og Øien [4]. Disse strategiene har dynamiske prioriteter, og en strategi vil velges på bakgrunn av denne prioriteten, som i de fleste tilfeller er avhengig av mange ulike variabler. AI består av 10 strategier som er nærmere beskrevet under:



Figur 5.3: Kommunikasjonshierarkiet i AI.

- Antikollisjonsstrategi
- Antikollisjons-supportstrategi
- Startstrategi
- Container-strategi
- Blå-hvit-dispenserstrategi
- Rød-hvit-dispenserstrategi
- Rød-blå-dispenserstrategi
- Søk-etter-ball-strategi
- Cooled-containerstrategi
- Horisontal-dispenserstrategi

De to siste blir i utgangspunktet ikke brukt, da det satses på å sortere/avlevere ballene i renna, noe som gir flere poeng. Samtidig vurderes tidsmangel som et argument for å ikke ta i bruk den horisontale dispenseren. I tillegg er det en større utfordring å få tak i ballene fra denne dispenseren enn fra de vertikale.

Alle strategier er bygget opp på samme måte slik at det skal kunne være lett å fjerne eller legge til en strategi. Hver strategi inneholder følgende metoder:

- **Konstruktor**: All initialisering skjer her (blir kjørt når strategiobjektet blir opprettet).
- **Destruktor**: Frigjør minne når objektet blir "drept".
- **Reset**: Re-initialisering av strategien dersom reset-knappen har blitt trykket.
- **Update**: Oppdatering av essensielle strategivariabler basert på meldingskøen.
- **UpdatePriority**: Oppdatering av den dynamiske strategiprioriteten.
- **GetPriority**: Returnerer prioriteten til strategien.
- **PerformStrategy**: Kjører tilstandsmaskinen som utgjør selve strategien. Tilstandsmaskinen er en while-løkke som består av en switch-case. Dette blir nærmere forklart for hver strategi under.

5.4.1 Prioritet

Prioritetene er definerte i området 0-9, det vil si at en prioritet 0 er lavest og 9 høyest. Prioriteten til den enkelte strategien er dynamisk og er avhengig av ulike faktorer:

- Avstanden til dispenserne
- Hvor mange baller igjen i dispenserne
- Hvor mange baller i magasinet i roboten (av den gitte fargen)
- Kollisjon

Faktorene blir vektet, slik at én faktor kan være viktigere enn en annen.

Tiden vil også innvirke på prioriteten til en strategi. Når tiden begynner å renne ut, vil de fleste strategiene få prioritet 0, mens Container-strategien vil få prioritet 9 dersom roboten inneholder noen baller. Noen variabler er viktige for å avgjøre når en strategi skal regnes som "done" eller "completed". Forskjellen går på at noen strategier kan velges flere ganger. *strategyDone* og *strategyCompleted* virker derfor inn på prioriteten. I begge tilfeller settes

prioriteten til 0, men i *strategyDone*-tilfellet vil prioriteten kunne settes på nytt på et senere tidspunkt. Hvis *strategyCompleted* er satt vil prioriteten forbli 0.

En del tid har gått med til å tune vektingen av de ulike faktorene. Avstandsfaktoren er ikke like viktig som innholdet i dispenserne. Samtidig er det ganske viktig hvor mange baller som er av de ulike fargene i magasinet i roboten. Derfor må disse tre faktorene vektes ulikt slik at roboten velger strategier på en hensiktsmessig måte.

Et eksempel på en dispenserstrategi-prioritet:

```
priority =  (((maxDistance-distance)/maxDistance)*0.2 +
            internalState*0.10 +
            robotState*0.06 +
            extraBall*0.06)*9.9;
```

Prioriteten i dette tilfellet består av 4 ledd. Det første leddet vekter prioriteten i forhold til den maksimale avstanden roboten kan befinne seg fra dispenseren. Dersom avstanden er veldig liten, vil dette leddet bli tilnærmet 1*0.2 = 0.2. Det vil si at dette leddet kan maksimalt utgjøre 20% av den totale prioriteten. Det neste leddet, *internalState* er rett og slett lik antall baller som er igjen i dispenseren. Dersom dispenseren er full, vil *internalState* være 5, og totalt vil da dette leddet utgjøre 50% av prioriteten. Det tredje leddet, *robotState*, er et forholdstall som blir satt med utgangspunkt i hvor mange baller det er av den aktuelle fargen i robotens magasin. Denne delen av prioriteten kan utgjøre maksimalt 30% (da robotState kan være maksimalt 5). Dersom roboten ikke har plukket opp like mange baller som den skulle av den andre fargen, vil *extraBall* settes lik antall baller som mangler av den fargen. På denne måten vil man kunne sørge for å fylle opp magasinet uansett. Også denne kan være maksimalt 5. Summen av alle leddene kan bli maksimalt 1, og siden prioriteten er en *int*, vil prioriteten bli rundet ned fra 9.9 til 9.

5.4.2 Antikollisjonsstrategi

Dette er den strategien som er vanskeligst å implementere. Utfordringen ligger i å gjøre fornuftige valg i tilfelle kollisjon. Valg av handling baseres først og fremst på hvilke sensorer som detekterer kollisjon. På bakgrunn av sensorene vet AI hvor kollisjon er detektert; venstre, høyre eller midt på. Ulike valg blir tatt som følge av dette, i tillegg taes hensyn til hvor på bordet roboten befinner seg. Prioriteten til denne strategien er 0 så lenge det ikke detekteres noen kollisjon, for så å bli satt til 9 ved kollisjon. I antikollisjonsstrategien vil roboten rygge bakover en gitt avstand som er avhengig om det er frontkollisjon eller ikke. Type kollisjon blir lagret, og blir brukt i antikollisjons-supportstrategien. Antikollisjonsstrategien består av 2 tilstander, *ADJUST* og *COLLISION*. Under konkurransen ble kun *COLLISION*-tilstanden brukt. Denne tilstanden har

2 if-setninger, henholdsvis **if** *(!lastWaypointSent)* og **if** *(lastWaypointReached)*. Den første vil slå til når strategien startes, og *lastWaypointSent* vil settes til **TRUE** så fort kollisjons-waypointet har blitt sendt. Når waypointet er nådd, sender navigasjonssystemet melding om dette, slik at *lastWaypointReached* blir satt til **TRUE**. Tilstandsmaskinen er da ferdig med å kjøre, og *strategyDone* blir satt **TRUE**, slik at strategien avsluttes.

ADJUST-tilstanden var ment som en nedbremsingstilstand før absolutt stopp ved kollisjon. Tanken var å ta i bruk avstandsregulatoren beskrevet i avsnitt 4.2.4. På denne måten kunne man bremse farten ved kollisjonsdeteksjon, slik at roboten ikke bråstoppet. Det ble derfor i utgangspunktet tatt i bruk to grenser for kollisjon. Den øverste grensen *COLLISION THRESHOLD* ble definert noe høyere enn *COLLISION STOP*, slik at nedbremsingen skulle startes ved den øverste grensen, og at roboten skulle stoppe ved *COLLISION STOP*-grensen (skifte tilstand fra ADJUST til COLLISION). Denne nedbremsingstilstanden ble ikke benyttet under konkurransen fordi det under testing ble observert at roboten stoppet for sent. Roboten har såpass stor hastighet at målingene faller fort fra *COLLISION THRESHOLD* til *COLLISION STOP*. Dermed rekker ikke roboten å reagere fort nok, og kjørte i mange tilfeller på hindringen. Dette ble derfor ikke regnet som robust nok til å bli tatt i bruk under konkurransen.

5.4.3 Antikollisjons-supportstrategi

Prioriteten til denne strategien er i hovedsak avhengig av *supportNeeded*-variabelen. Denne variabelen settes av antikollisjonsstrategien, og når denne er satt er prioriteten 9, ellers 0. Denne strategien utfører selve manøvreringen forbi en motstander. Den leser av variabelen som beskriver type kollisjon, og på bakgrunn av denne velges venstre eller høyre side til forbikjøring. Hvordan dette gjøres er nærmere beskrevet i kapittel 6. Når strategien er kjørt ferdig settes *supportNeeded* til **FALSE**, og AI vil fortsette med den opprinnelige strategien.

Antikollisjons-supportstrategien består av 3 tilstander; *INIT*, *SUPPORT* og *ANGLE FIX*. I *INIT*-tilstanden oppdateres posisjonen og support-waypointene, som danner forbikjøringsbanen, blir utregnet. Neste tilstand er *SUPPORT*, der alle forbikjørings-waypointene blir sendt. Når alle waypointene er nådd, vil headingen bli rettet mot det opprinnelige waypointet i *ANGLE FIX*-tilstanden. Dersom kollisjon oppstår underveis, vil alle waypointene i *SUPPORT*-tilstanden bli slettet, og roboten vil rygge 20 cm bakover, for så å snu seg mot det opprinnelige waypointet i *ANGLE FIX*-tilstanden. Når strategien er ferdig, settes *supportNeeded* til **FALSE**, og AI fortsetter med den opprinnelige strategien.

5.4.4 Startstrategi

Dette er en oppstartsstrategi som definerer hva roboten skal gjøre i starten av kampen. Denne strategien vil ha maksimal prioritet (9) i oppstarten og 0 når den er ferdig. Denne strategien er basert på å plukke opp den nærmeste fargede ballen pluss den nærmeste hvite ballen (disse to har fast posisjon). Når disse to ballene er plukket opp eller dersom AI timer ut fordi en av, eller begge, ballene ikke har blitt plukket opp, avsluttes strategien ved at *strategyDone* blir satt til **TRUE**. Dermed vil prioriteten bli satt til 0 og denne strategien vil ikke bli valgt igjen.

Startstrategien har 2 tilstander; *INIT* og *FIRST WHITE BALL*. I den første tilstanden vil roboten kjøre til den fargede ballen og plukke den opp. Dersom den ikke plukker opp noen ball vil tilstanden time ut, og neste tilstand velges. *FIRST WHITE BALL* oppfører seg på samme måte.

Pseudokoden for startstrategien er gjengitt i algoritme 5.4.1, som i utgangspunktet viser strukturen på tilstandsmaskinen for en hvilken som helst strategi. Timeout er ikke tatt med for enkelhets skyld.

Algorithm 5.4.1: STRATEGIIMPLEMENTASJON(*StartStrategy.cpp*)

```
while true
    switch(state)
        case INIT:
            if not lastWaypointSent
                position arms 45 degrees;
                send first colored ball waypoint;
                lastWaypointSent = true ;
            if lastWaypointReached
                position arms 90 degrees;
                lastWaypointReached = false ;
                send first white ball waypoint;
            if newBall
                check the color of the ball;
                newBall = false ;
                wait for the belts to send the ball inside the robot;
                state = FIRST_WHITE_BALL;
        case FIRST_WHITE_BALL:
            if lastWaypointReached
                position arms 90 degrees;
                lastWaypointReached = false ;
            if newBall
                check the color of the ball;
                newBall = false ;
                wait for the belts to send the ball inside the robot;
                position arms 45 degrees;
                strategyDone = true
```

5.4.5 Container-strategi

Denne strategien har en prioritet avhengig av hvor mange baller som er i magasinet. Når magasinet er fullt er prioriteten 9, slik at roboten vil kjøre til standard-containeren og tømme magasinet. Etter at magasinet er tømt, blir prioriteten satt til 0. Waypointet til containeren blir satt et lite stykke fra veggen. Når dette waypointet er nådd, vil roboten benytte seg av avstandsregulatoren beskrevet i avsnitt 4.2.4. Den vil altså kjøre mot kanten helt til begge endebryterne har trigget. Dermed vet man at roboten står helt inntil kanten og man kan sette y-posisjonen og headingen nøyaktig. I tillegg initieres tårnavlesning slik at x-posisjonen kan bli oppdatert. På denne måten har vi nullet ut eventuelle odometrifeil.

Når tiden begynner å renne ut og det er kun 15 sekunder igjen av kampen, vil alle andre strategier enn denne få prioritet 0, i tillegg til at container-strategien får prioritet 9. Dette er for å sikre at eventuelle baller i magasinet skal bli sortert før tiden er ute.

Container-strategien består av 4 tilstander: *INIT*, *ADJUST*, *RELEASE BALLS* og *REVERSE*. I *INIT*-tilstanden settes det første waypointet som beskrevet over. I *ADJUST* kjøres avstandsregulatoren, og når begge endebryterne har trigget, tømmes magasinet og man skifter tilstand til *RELEASE BALLS* hvor man venter på beskjed om at sorteringsmodulen er ferdig med å tømme. Når denne meldingen er mottatt, skiftes tilstanden til *REVERSE* og roboten rygger 30 cm tilbake og snur seg mot bordet.

5.4.6 Dispenserstrategiene

Dispenserstrategiene er i utgangspunktet helt like, bortsett fra fargen på ballene. De består alle av de samme tilstandene; *FIRST RUN*, *INIT*, *DISPENSER*, *PICK UP BALL* og *REVERSE*. *FIRST RUN* brukes til å sjekke om det er første gangen strategien blir kjørt. Hvis ikke settes et hjelpe-waypoint 70 cm fra og pekende mot dispenseren. Dette for å sørge for at roboten kommer rett inn mot dispenseren, og ikke fra sidene. Første gangen man kjører til dispenserne kommer man rett mot dem pga. startstrategien. Derfor trengs ikke hjelpe-waypointet første gangen. Så fort dette waypointet er nådd, skiftes tilstanden til *INIT* der det virkelige dispenserwaypointet blir satt. Deretter velges *DISPENSER* eller *PICK UP BALL* alt etter om avstandsregulering (se avsnitt 4.2.4) skal benyttes eller ikke. Dersom avstandsreguleringen er deaktivert, blir neste tilstand *PICK UP BALL* og roboten er klar til å plukke opp baller fra dispenseren. Når roboten har plukket opp det antall baller den skulle, eller den har funnet ut at dispenseren er tom, skiftes tilstand til *REVERSE* og roboten rygger bakover en definert lengde.

Blå-hvit/Rød-hvit-dispenser-strategi

Prioritetene til disse to strategiene er avhengige av hvor mange baller som er igjen i dispenseren, hvor mange hvite baller som allerede er i roboten og avstanden til dispenseren. Selve strategien går i utgangspunktet bare ut på å kjøre til en hvit dispenser og plukke opp baller.

Rød-blå-dispenser-strategi

Strategien går ut på å kjøre til den røde eller blå dispenseren, avhengig av *teamColor*. Prioriteten blir satt tilsvarende de hvite dispenserne.

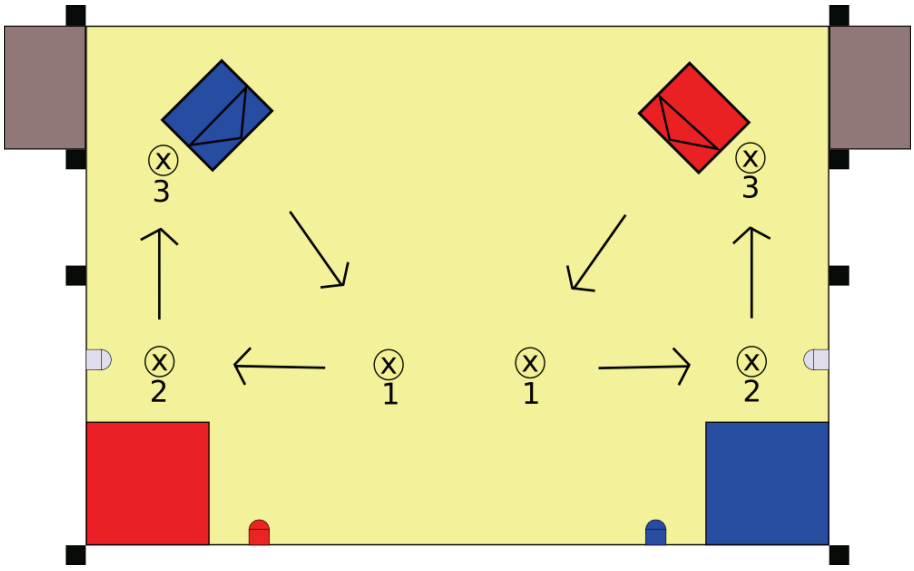
5.4.7 Søk-etter-ball-strategi

Denne strategien hadde i utgangspunktet en prioritet avhengig av antall gjenværende baller i dispenserne. Dette fordi det ikke var ønskelig å kjøre denne strategien før dispenserne var tomme. Når dispenserne var tomme startet søket etter baller på bordet. Til dette benyttes kamera (se kapittel 7). Kameraet tar et bilde, og dersom en ønsket ball finnes innenfor et akseptert område vil et waypoint bli sendt til navigasjonssystemet. Dette blir gjort så lenge det er plass til flere baller i magasinet. Etter hvert ble prioriteten til strategien satt mer fleksibel, slik at strategien kunne kjøres når man selv ønsket det. I forhold til konkurransen vil det si en hardkodet høy prioritet etter å ha kjørt den første avleveringsrunden.

Strategien har i alt 4 tilstander: *WAYPOINTS*, *GO TO BALL*, *PICK UP BALL* og *COLLISION*. *WAYPOINTS*-tilstanden har det navnet den har, fordi roboten kjører rundt i en bane bestemt av forhåndsdefinerte waypoints (som er lagret i en waypoint-array). Roboten kjører fra waypoint til waypoint og søker samtidig hele tiden etter baller. Figur 5.4 viser hvordan roboten leter etter baller for henholdsvis blått og rødt lag. De 3 kryssene tilsvarer 3 waypoints som er de faste søkewaypointene. Denne strategien vil alltid velges etter container-strategien, slik at roboten vil alltid starte i avleveringshjørnet, som vist i figuren. De 3 waypointene er valgt slik for å sørge for at roboten søker i det området det er mest sannsynlig at det ligger baller, samtidig vil roboten aldri kjøre langt unna standardkontaineren.

Dersom roboten oppdager en ball, vil aktivt waypoint bli slettet og neste tilstand blir *GO TO BALL*. Her sendes ball-waypointet og roboten kjører mot ballen. Når ballen er nådd, skiftes tilstand til *PICK UP BALL* og ballen blir plukket opp. Her er det lagt inn en timeout i tilfelle den bommer på ballen. Når ballen er plukket opp eller tilstanden har timet ut, vil neste tilstand bli *WAYPOINTS*. *COLLISION*-tilstanden blir brukt dersom kollisjon detekteres. Da vil aktivt waypoint bli slettet, roboten rygger tilbake 30 cm og snur seg 90 grader. Deretter velges neste waypoint i waypoint-arrayen. Grunnen til at ikke antikollisjonsstrategien blir kjørt ved kollisjon, er at man ønsker å ha større kontroll på hva roboten foretar seg i denne strategien, og fordi det waypointet den ønsker å nå ikke er et viktig waypoint. Det er kun en del av en søkerute, og det virker derfor naturlig å hoppe til neste waypoint istedenfor å tviholde på det forrige. Strategien kjøres til det er 15 sekunder igjen av kampen. Da avbrytes den og Container-strategien overtar (såfremt roboten har noen baller i magasinet).

Under søkingen settes hastigheten ned, ved at waypoints som blir sendt til navigasjonssystemet er av en annen type enn vanlige waypoints. For nærmere beskrivelse, se avsnitt 4.2.2, waypoint-type 2.



Figur 5.4: Søkeruten til roboten for henholdsvis blått og rødt lag.

5.5 Waypoints

Det skilles mellom lokale og robotrelative waypoints som beskrevet i navigasjonskapittelet (kapittel 4). Dette benyttes mye av AI, og det er 3 metoder i *PosixCommunication* som blir brukt til å sende waypoints til navigasjonssystemet:

- **sendRobotWaypoint()**: Denne metoden tar inn message-handleren til navigasjonssystemet og et ferdig definert waypoint i lokale koordinater.
- **sendRelativeRobotWaypoint()**: Denne metoden tar inn det samme som over, men dette waypointet er definert relativt til roboten.
- **sendReverseWaypoint()**: Tar inn et rygge-waypoint, og input til metoden er *x*, *y*, *psi*, *circle of acceptance* og om waypointet er relativt eller lokalt. Metoden bruker videre en av de to foregående.

5.6 Inputs som styrer AI

AI behandler ulike inputs for valg av strategi og framgangsmåte. Disse er:

- Kamera, kapittel 7
- Kollisjonsdetektorer, kapittel 6
- Navigasjonssystemet, kapittel 4

- Endebrytere, kapittel 4
- Sorteringsmodul, kapittel 2
- Spenningsvarslerkort, kapittel 9

5.7 Simulering

Roboten har vært under utvikling hele semesteret. EiT-gruppene har disponert den mye, og navigasjonssystemet har vært under utvikling. Derfor har det vært vanskelig å teste AI mot den fysiske roboten. Av praktiske årsaker meldte det seg derfor et behov for en simulator. Lester Solbakken, stipendiat (innen kunstig intelligens) ved IDI, tipset om et open-source simulator-bibliotek kalt "Player-Stage".

5.7.1 Player

"Player" tilbyr et nettverksgrensesnitt til et stort utvalg av robot- og sensorhardware. Player sin klient/server-modell tillater styringsprogrammer for roboter å bli skrevet i et hvilket som helst programmeringsspråk med en nettverkstilkobling til roboten.

5.7.2 Stage

"Stage" simulerer roboter som beveger seg i et to-dimensjonalt miljø. Ulike sensormodeller er integrert: Sonar, laser, kamera og odometri.

5.7.3 Player-Stage

Kombinasjonen av Player og Stage definerer simulatoren som er brukt til testing av AI. Installasjonen av Player og Stage er beskrevet på hjemmesiden til Player-Stage-prosjektet, se [8]. For at simulatoren skal ha noen funksjon må den oppføre seg som navigasjonsdelen av software. På denne måten skal AI fungere på samme måte, uavhengig av om det benyttes en simulator eller den faktiske, fysiske roboten.

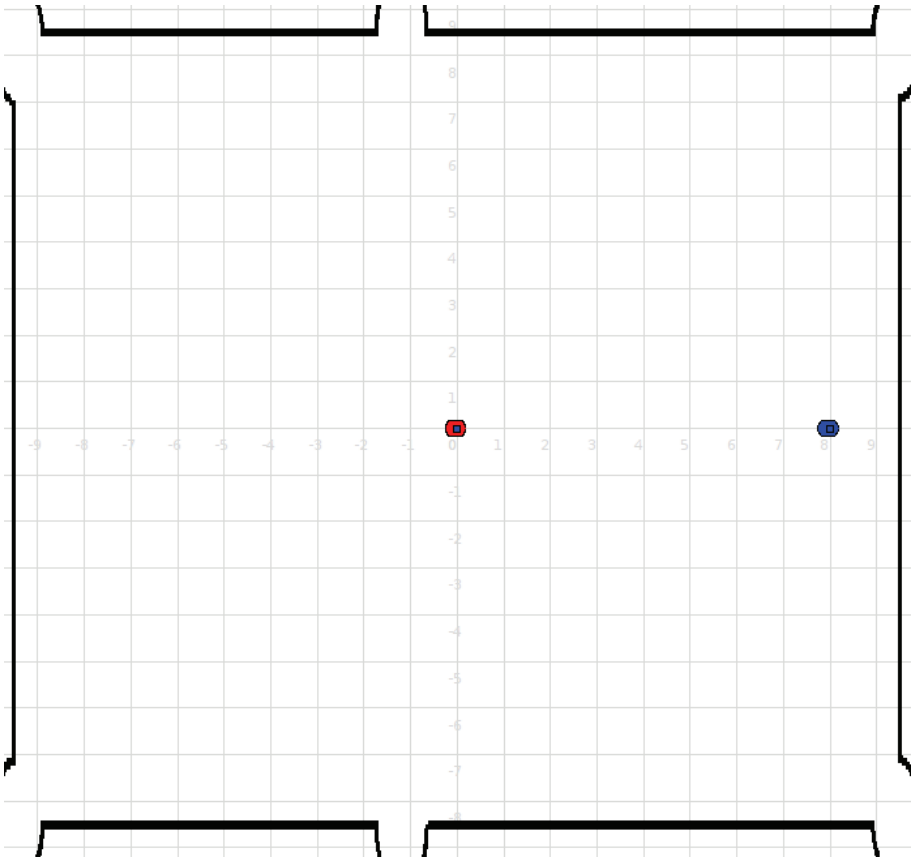
Selve simulatoren startes ved å kjøre *player* med en gitt konfigurasjon:

```
$ player simple.cfg
```

der *simple.cfg* inneholder konfigurasjonen til roboten og hvilket *map* som skal brukes. Robotkonfigurasjon kan f.eks. være odometri, laser og sonar. *map* blir definert ut fra en *world*-fil, i dette tilfellet *simple.world*. I denne filen defineres

størrelsen på vinduet, hvor roboten skal starte og navnet på png-filen som brukes til å definere bordet.

Når *player* er startet med en gitt konfigurasjon, popper vinduet som er vist i figur 5.5 opp.



Figur 5.5: Skjermbilde av simulatoren.

For konfigurasjon som passer skjermbildet, se vedlegg B.

Nå er altså selve simulatoren oppe, men det må skrives en driver før simulatoren gjør som AI vil. Denne driveren kjøres parallelt som et c++-program. Det er denne driveren AI kommuniserer med. De viktigste punktene i driveren er:

- **PlayerClient robot("localhost")**: Oppretter en robotklient.
- **SonarProxy sp(&robot,0)**: Oppretter en sonar for antikollisjon.
- **Position2dProxy pp(&robot,0)**: Oppretter et navigasjonssystem.
- **pp.GetXPos(), pp.GetYPos(), pp.GetYaw()**: Ber navigasjonssystemet om posisjon.

- **pp.GoTo(xPos,yPos,yaw)**: Ber navigasjonssystemet om å kjøre til en gitt posisjon.
- **pp.ResetOdometry()**: Resetter odometrien.
- **sp[0], .., sp[n]**: Leser ut data fra sonaren, sp[0] tilsvarer sonarsensor 1.
- **robot.Read()**: Oppdater alle målinger fra roboten.

Et eksempel på et program som benytter punktene over er oppsummert i algoritme 5.7.1 under. Metoden *checkCollision()* leser ut sp[0] osv, som beskrevet over. Det samme gjelder *checkPosition()*, som leser ut posisjonen fra odometrien.

Algorithm 5.7.1: PLAYER-STAGE(*simulator.cpp*)

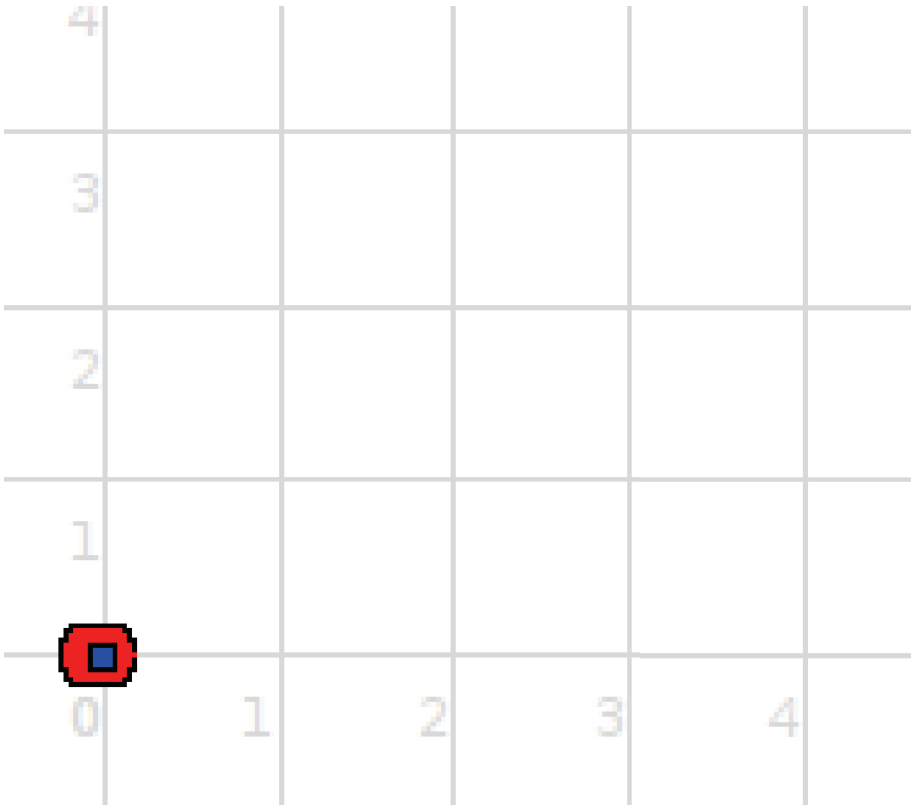
```
PlayerClient robot("localhost");
SonarProxy sp(&robot,0);
Position2dProxy pp(&robot,0);
while true
(   robot.Read();
    checkCollision();
    checkPosition();
    while not waypointReached or not waypointReceived
    (       robot.Read();
            checkCollision();
            checkPosition();
            if waypoint reached
                waypointReached = true ;
    )
    send waypoint reached to AI;
)
```

I tillegg til de metodene som er nevnt i algoritmen over, vil en del andre metoder kjøres etter forespørsel fra AI. Dette gjelder hovedsaklig *setWaypoint()*-metoden som kaller *pp.GoTo(xPos,yPos,yaw)*, som ber simulatoren om å kjøre til den gitte posisjonen.

Praktisk informasjon

Da det tok litt tid å sette seg inn i hvordan man setter opp et eget bord med rett skalering, ble dette nedprioritert, og det bordet som er vist i figur 5.5 ble brukt. Det viste seg også å være litt vanskelig å få satt farten til roboten, så den kjører med standardhastighet, noe som gjør roboten veldig treig. Derfor ble bare en brøkdel av brettet benyttet, og denne brøkdelen tilsvarer det faktiske

spillebordet. I utgangspunktet er bordet i simulatoren på 16x16 ruter, men kun 4x3 ble tatt i bruk, se figur 5.6.



Figur 5.6: Skjermbilde av simulatoren, zoomet inn på aktivt område.

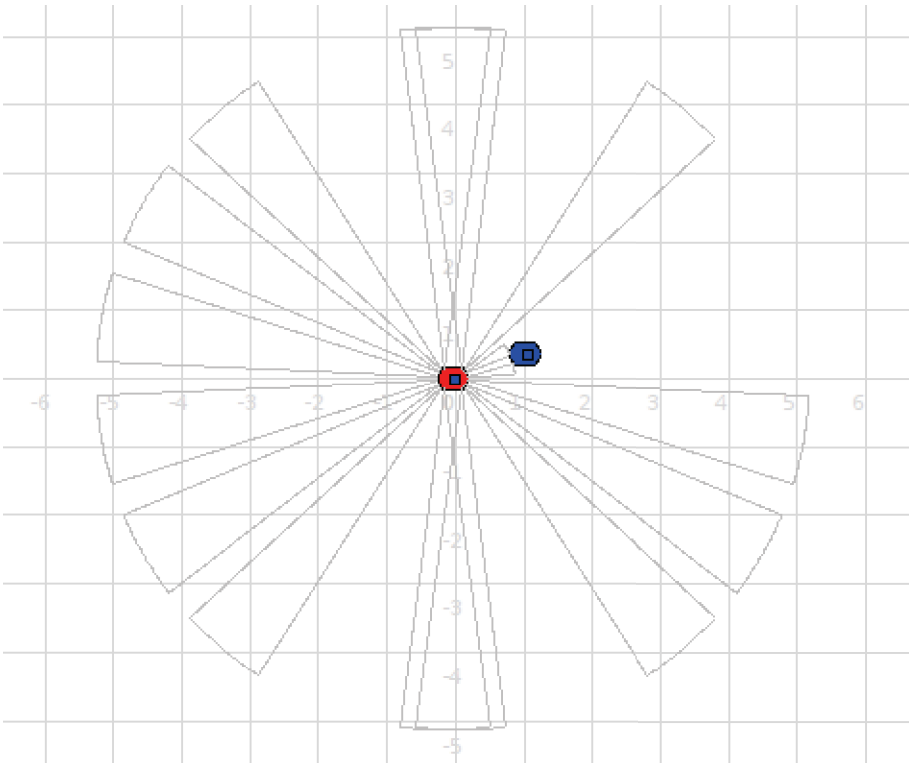
Figur 5.7 viser hvordan sonaren fungerer. Man kan se at den trigger på den blå roboten ved å se på utstrekningen til sonarbølgene.

5.8 Testing og resultater

Da simulatoren var satt opp og grensesnittet definert, var det klart for testing av systemet. En testprosedyre ble gjennomført for å teste at AI oppfylte de krav man hadde satt. Disse kravene vil bli presentert først. Deretter følger en gjennomgang av testene.

5.8.1 Kravspesifikasjon

Det er viktig å definere hvilke krav man ønsker å sette til oppførselen til AI. Dette er krav som utgjør utgangspunktet for testing av software. Desto mer



Figur 5.7: Skjermbilde av simulatoren, med fokus på sonar. Legg merke til at sonaren har trigget på den blå roboten.

gjennomført testingen er, jo sikrere er vi på at AI fungerer som den skal. De viktigste kravene er:

1. Roboten skal startes ved hjelp av fargevalg og en startsnor. AI må derfor sette riktig team-farge i software, basert på hvilken farge som ble valgt. Deretter skal selve AI-en starte i det startsnoren blir nappet ut.

2. Software skal resettes ved trykk på reset-knappen. Dette innebærer at all kode skal initialiseres på nytt og AI skal vente på valg av farge og start. I tillegg skal timeren restartes ved start.

3. Startstrategi skal velges som første strategi, og denne skal ha prioritet 9.

4. Roboten skal plukke opp tilsammen 5 baller før den kjører til standard-kontaineren og tømmer magasinet. 2 av disse ballene skal være de 2 nærmest plasserte (fast posisjonerte, en farget og en hvit), de 3 andre plukkes fra de to dispenserne som er plasserte ved starthjørnet.

5. Antikollisjon skal aktiveres dersom en motstanderrobot kommer i veien. Det vil si at roboten skal stoppe, for så å kjøre til siden eller bakover.

6. AI skal velge den avbrutte strategien etter antikollisjonsstrategien. Altså skal roboten fortsette som før.
7. Når roboten har plukket opp 5 baller skal den velge Container-strategien og kjøre til standardkontaineren.
8. Roboten skal tømme magasinet fullstendig før den velger neste strategi.
9. Når roboten kjører mot en dispenser skal den bruke kollisjonsdetektorer til å justere avstanden til dispenseren, slik at roboten stopper en gitt avstand fra dispenseren.
10. Når roboten kjører mot standardkontaineren, skal den kjøre helt til de to endebryterene på fronten av roboten trigger. På denne måten er man sikker på at roboten har kjørt helt inntil veggen.
11. Roboten skal kunne detektere baller på bordet ved hjelp av kamera, og skal på bakgrunn av dette sette ut waypoints til navigasjonssystemet. Dermed skal roboten kunne plukke opp ballen dersom denne har riktig farge.
12. AI skal passe på tiden slik at når 90 sekunder er gått skal alle aktuatorer stoppes.
13. Når spenningen på batteriene som er tilkoblet spenningsvarsleren, kommer under et spesifisert nivå, skal pc-en slås av.

5.8.2 Testresultater

Simulatoren ble brukt til testing av kravspesifikasjonen. Et grensesnitt ble definert slik at simulatoren oppførte seg slik den fysiske roboten ville gjort med samme inputs fra AI. Ikke alle kravene kunne bli testet med simulatoren. De kravene som ikke ble testet her, presenteres og testes i sammenheng med testing av totalsystemet, se kapittel 10.

1. Spenningsvarslerkortet ble brukt for å velge teamColor og for å starte roboten/simulatoren. På denne måten fikk man samtidig testet dette kortet. Problemer med CAN-bussen gjorde at valg av farge og start ble hardkodet inn i software. Fordi fokus lå på AI og simulatoren, ble problemet med CAN-bussen lagt litt på is. Det ble observert at simulatoren mottok riktig startposisjon når teamColor var valgt.
3. Startstragien ble valgt og hadde prioritet 9, roboten kjørte mot den første ballen som planlagt.
4. Opplukking av baller måtte hardkodes, siden simulatoren ikke hadde noen opplukkingsmodul eller baller. Derfor ble det definert noen midlertidige metoder i AI som plukket opp baller og reduserte antall dispenser-

baller basert på input fra bruker. Når roboten nådde waypointet måtte antall baller skrives inn i konsollen, og det ble valgt om ballen(e) kom fra dispenser eller ikke. Roboten kjørte og plukket opp de to første ballene, for så å ta baller fra de to dispenserne.

5. Stage (simulatoren) har innebygget sonar, slik at antikollisjon kunne testes med simulatoren. I utgangspunktet var antikollisjonsstrategien veldig enkel, og systemet ble testet ved at roboten stoppet når en motstanderrobot kom i veien. Dette fungerte bra, roboten ble stående i ro til man fjernet motstanderroboten igjen.

6. Den avbrutte strategien startet på nytt etter kollisjonen og dette tydet på at prioritetene var satt riktig.

7. Etter å ha plukket opp 5 baller, ble container-strategien valgt, og roboten kjørte mot det gitte waypointet.

8. Tømming av magasinet ble gjort, og prioritetene til de andre strategiene ble justert på bakgrunn av at magasinet i roboten var tomt.

12. Det var vanskelig å tune hastigheten til roboten i simulatoren til å bli lik den faktiske, slik at tiden ble økt til langt mer enn 90 sekunder for å rekke å teste opplukking og tømming. Når tiden var ute, stoppet AI simulatoren. Dette så derfor ut til å fungere bra.

5.9 Konklusjon

Denne måten å bygge opp AI på, altså ved bruk av strategier med dynamiske prioriteter, har vist seg å være en veldig strukturert og enkel løsning. Det at AI er lite kompleks gjør den mer robust, og man har hele tiden kontroll på hva roboten foretar seg. Hver enkelt strategi er relativt kort og konsis, og den overordnede styringen er enkel å ha kontroll på. Fleksibiliteten i forhold til å legge til eller fjerne strategier er helt klart en fordel ved denne typen AI. På denne måten kan AI lett tilpasses et nytt prosjekt der man kun behøver å erstatte strategiene. Selve skjelettet i strategiene kan brukes videre. I utgangspunktet er det kun behov for å bytte ut tilstandsmaskinen i en strategi, så har man en ny strategi.

Mangel på tid i forhold til testing mot den fysiske roboten resulterte i behov for simulering. Uansett er simulering en fordel, da dette er en mye mer effektiv måte å debugge systemet på. Mye tid gikk med til debugging av AI vha. Player-Stage-simulatoren. Dette viste seg å være vel investert tid, og det anbefales å bruke denne simulatoren videre.

Kapittel 6

Antikollisjon

Årets antikollisjonssystem bruker fjorårets system som utgangspunkt. I fjor var systemet ustabilt, noe vi måtte ta hensyn til og forbedre. Det var flere grunner til at systemet ikke fungerte optimalt, blant annet kommunikasjonen mellom AI og kollisjonsdetektorene. I dette kapittelet vil det eksisterende systemet bli grundig gjennomgått og analysert. I tillegg vil forbedringer presenteres, og til slutt omfattende testing av det endelige systemet.

6.1 Bakgrunn

6.1.1 Eksisterende hardware

Antikollisjonssystemet er utviklet av Kristian M. Knausgård og består av to hovedkort, ett master- og ett slavekort. Disse kortene kommuniserer seg imellom via CAN-bussen. Hvert av disse kortene kan ha opptil 4 sensorkort koblet til, og disse sensorkortene kommuniserer med hovedkortet via TWI-buss. Sensorkortene består av ultralyd- og Ir-sensorer. Maks antall sensorer er dermed 8x2; 8 sensorkort med 2 sensorer på hvert kort.

6.1.2 Eksisterende software

Antikollisjonssystemet er altså oppdelt i 2 hovedkort. Det ene er definert som master, det andre som slave. Masterkortet vil alltid vente på oppdateringer fra slavekortet. Når disse oppdateringene har kommet vil masteren ha en komplett oversikt over alle sensormålingene. I utgangspunktet legges ultralydmålingene og ir-målingene i adskilte bytes. Fra masteren til AI legges gjennomsnittet av ultralyd og ir for hvert sensorkort i hver sin byte i en CAN-melding. CAN-

meldingen med gjennomsnittsmålingene sendes når målingene er under en gitt verdi.

6.1.3 Svakheter ved eksisterende system

Under fjorårets konkurranse oppstod det mange rare situasjoner som følge av kollisjon. Det var mye på grunn av for lite tid til testing. AI hadde ingen logikk i forbindelse med antikollisjonen. Sensorkortene var programmerte til å sende CAN-meldinger når avstanden ble mindre enn en definert grense. Når AI mottok en slik melding, gikk den ut fra at det var kollisjon, og stoppet dermed roboten umiddelbart. Dette gjorde at kollisjon ble trigget veldig ofte, og roboten fikk aldri gjort noe fornuftig. Både ultralyd- og Ir-sensorne måler av og til feil, ved at en verdi plutselig kan være for lav slik at AI trigger.

6.2 Hardware

Med utgangspunkt i erfaringene fra i fjor, ble det satt opp en testplan med medfølgende kravspesifikasjon. Denne testingen er basert på de eksisterende sensorkortene, og er ment å klargjøre hva som ikke fungerer. Implementeringen av hardware ble gjort etter å ha gjennomført testingen. Resultatene fra den endelige implementasjonen av hardware er beskrevet senere i kapittelet.

6.2.1 Kravspesifikasjon

1. Ingen sensor skal trigge kollisjon dersom den faktiske avstanden er mer enn 30 cm.
2. Ir og ultralyd skal ha noenlunde like målinger, slik at ikke avviket de to imellom blir for stort.
3. Gjennomsnittet av målingene til de to sensorne skal være så korrekt som mulig, jamfør punktet over.
4. Ved avstander under 30 cm skal sensorne vise nøyaktige målinger, da dette er viktig for å unngå kollisjon eller for å unngå “falske” kollisjoner.
5. Sensorne skal være robuste og i minst mulig grad utsatt for forstyrrelser.
6. Sensorne skal vise fornuftige verdier også når roboten er i fart.

6.2.2 Testplan

Testplanen er basert på kravspesifikasjonen og ble gjennomført i samme rekkefølge.

1. Ett og ett sensorkort settes mer enn 30 cm fra en vegg. Sjekk om det trigger kollisjon.
2. Sjekk målingene til Ir og ultralyd hver for seg og verifiser om avviket mellom disse er akseptabelt.
3. Snittet av målingene skal være et akseptabelt estimat for den faktiske avstanden til veggen.
4. Sjekk at målingene til Ir og ultralyd er nøyaktige under 30 cm.
5. Test hvordan sensorene påvirkes av forstyrrelser.
6. Montér sensorkortene på roboten og sjekk målingene når roboten beveger på seg.

6.2.3 Testresultater

1. Seks sensorkort var tilgjengelig, og ett av disse fungerte ikke. Dermed gjenstod det å teste de 5 fungerende kortene. Testene viste at de stort sett ikke trigget på avstander større enn 30 cm, med unntak av noen målinger. Fjorårets AI ville ha trigget på disse målingene siden den ikke hadde sjekk på om enkeltmålinger hadde stor varians.
2. Testing av de ulike sensorne hver for seg resulterte i et gjennomgående resultat; Ir viser for stor avstand når avstanden er liten, og ultralyd viser for liten avstand når avstanden er stor. Dette skillet ligger mellom 20 og 30 cm. Når avstanden blir større enn 30, viser ultralyd gjennomgående for liten avstand. Den kan f.eks. vise 35 cm, mens den reelle avstanden er 70. Motsatt med Ir, da denne kan vise 15 cm når den faktiske avstanden er 8.
3. Gjennomsnittet påvirkes av at Ir og ultralyd har ulike soner der de fungerer bra. Dette medfører at gjennomsnittet er for høyt ved veldig små avstander, og for lavt ved store avstander. Derfor kan man ikke stole på gjennomsnittsverdiene.
4. Ett viktig krav er at sensorne virker stabilt bra under 30 cm, da det er innenfor denne avstanden det er aktuelt å gjøre preventive tiltak i forhold til kollisjon. Ultralydsensorne viste gode målinger i sjiktet mellom 5 og 30 cm. Ir viste dårligere målinger jo nærmere 0 man kom.
5. Gunnar Kjemphol skriver i sin masteroppgaverapport ([9]) at sensorne mest sannsynlig ble forstyrret av andre roboters sensorer, og/eller lysforhold-

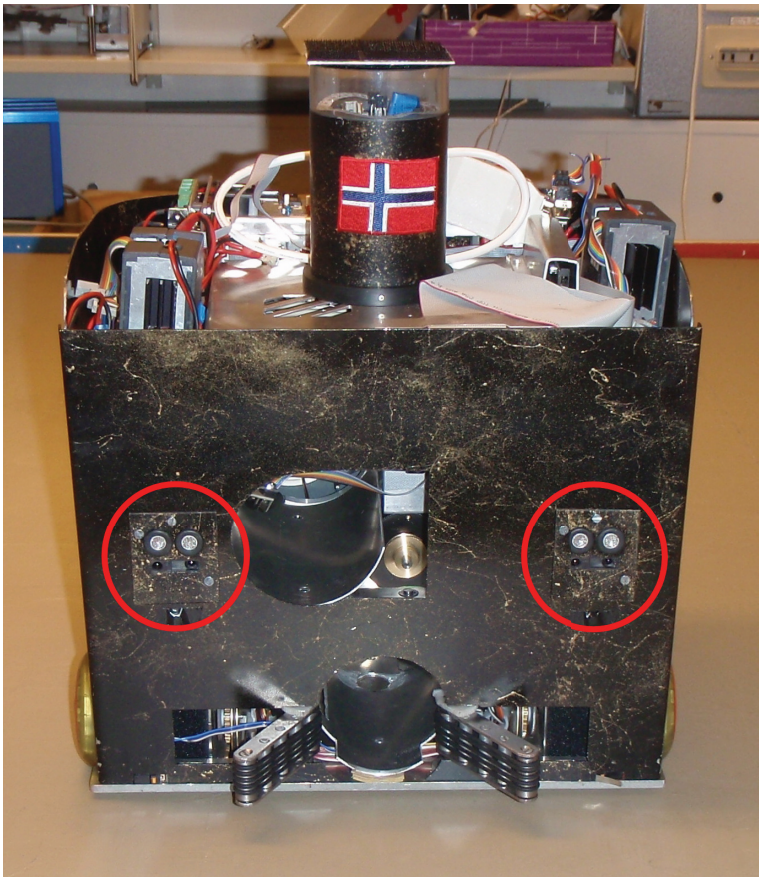
ene i salen. Dermed var det viktig å få testet sensorene for å bekrefte eller avkrefte hans antagelser. Ir-tårnene som benyttes til posisjonering, ble plassert på siden av roboten, mens antikollisjonssensorene var påmontert robotens framside. Det viste seg at Ir-målingene ble svært ustabile fordi de ble påvirket av Ir-tårnene. Ultralydsensorne ble testet ved å stille to sensorer mot hverandre. Resultatene viste at de ikke ble nevneverdig påvirket av hverandre. Konklusjonen ble at ultralydsensorne var mer robuste enn Ir. Følgelig ble bruken av Ir-sensorne forkastet, både fordi de ble lett forstyrret og fordi målingene er dårlige ved små avstander.

6. Sensorkortene ble etter hvert montert på roboten og sensormålingene ble avlest mens roboten kjørte. Målingene var tilfredsstillende, men verdiene fikk noen "peaks" av og til, og da spesielt lave verdier som trigger kollisjon. Det ble bestemt at AI skulle inneholde logikk som avgjorde om det var en kollisjon, og hvor grensene for kollisjonsunngåelse skulle settes.

6.2.4 Endelig hardwareoppsett

Basert på testresultatene ble det valgt å ikke bruke Ir-sensorne. Ellers ble samme sensorkort som i fjor brukt. Kun ultralydsensorne fungerer nå som kollisjonsdetektorer.

Totalt var 3 sensorkort i bruk som kollisjonsdetektorer, 2 framme (figur 6.1) og ett bak (figur 6.2). Kortet bak er felt inn i bakplaten.



Figur 6.1: Sensorkonfigurasjonen framme.

Figur 6.2: Ryggesensoren felt inn i bakplaten.

6.3 Software

6.3.1 Kollisjonsunngåelse

Erfaring fra konkurransen i fjor understreket viktigheten av en god taktikk i forhold til kollisjonsunngåelse. I utgangspunktet baserte Gunnar Kjemphol seg på å stoppe og vente ved kollisjon. Dette viste seg å være en strategi som fungerte dårlig i de tilfeller der motstanderen hadde tenkt på samme måte. Han gikk derfor bort fra den ideen etter hvert under konkurransen, og kom fram til at det kunne være lurt å kjøre rundt istedet for å stoppe. Mye tid har blitt brukt for å finne en lur måte å unngå kollisjon på. Noen viktige punkter ble funnet å være sentrale:

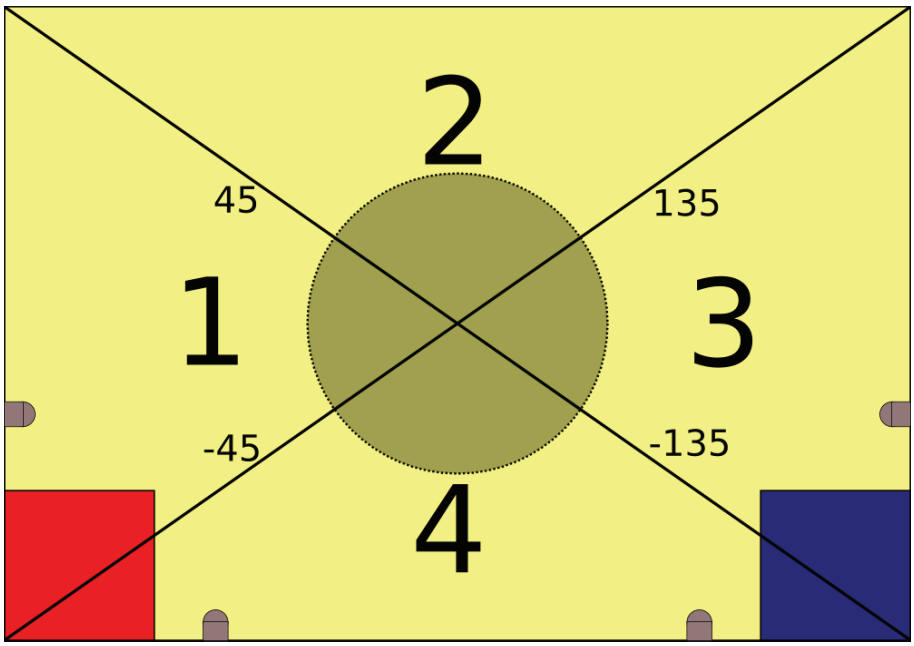
- Rygge tilbake for å få litt luft mellom roboten og motstanderen.
- Snu seg i en eller annen retning for å komme seg unna motstanderen.
- Kjøre rundt for å komme seg dit man ønsket seg i utgangspunktet.
- Prøve å nå det waypointet man siktet seg inn på før kollisjonen. Avhengig av viktigheten for å nå det gitte waypointet, avgjør hvor mange forsøk man skal gjøre før man bytter strategi.
- Skru av antikollisjon mens man rygger bakover for å kunne bestemme seg for strategi før roboten trigger kollisjon igjen.

Disse punktene la grunnlaget for en enkel algoritme for utregning av waypoints som danner en bane rundt motstanderen. Tre typer kollisjoner ble definert; venstre, høyre og front. Algoritmen tar hensyn til hvor kollisjonen detekteres, i tillegg til posisjonen på bordet og hvor det opprinnelige waypointet ligger. Banen rundt motstanderen er en tilnærmet halvsirkel som kan bestå av så mange waypoints man ønsker. Ved kollisjon estimeres posisjonen til motstanderroboten basert på hvilke sensorer som trigger. Dersom den venstre sensoren trigger, estimeres posisjonen til å være 20 cm fram (20 cm forventes å være ca. kollisjonsavstand) og 15 cm til venstre for roboten, tilsvarende for trigging fra høyre sensor. Ved frontkollisjon blir dette estimatet 20 cm fram og 0 cm til siden. Tanken er at posisjonen til motstanderen utgjør sentrum i en sirkel, og at man på denne måten kan legge waypoints rundt med en radius tilsvarende avstanden fra motstanderen til roboten. Denne radiusen kan være minimum 40 cm, og vil i de fleste tilfeller være større. Er den mindre vil man risikere å kjøre i motstanderen på nytt når man prøver å kjøre rundt. Det er 2 utveier ved kollisjon; å kjøre til høyre eller til venstre. Det er flere hensyn å ta ved valg av side. Faktorer som påvirker dette:

- Plassering av det opprinnelige waypointet i forhold til den rette linjen som dannes fra roboten gjennom motstanderen. Dersom waypointet ligger til venstre for linjen, vil det være naturlig å legge banen til venstre.

- Hvor på bordet befinner roboten seg? Kanskje er det ikke plass til å kjøre forbi på venstre siden?
- Headingen til roboten påvirker valg av x- og y-koordinater for de relative waypointene som blir dannet av algoritmen.

Headingen til roboten og hvordan denne påvirker x- og y-koordinater beskrives i figur 6.3. Her ser man at bordet er delt opp i 4 soner, avhengig av headingen til roboten. Sone 1 spenner fra -45 til 45 grader, sone 2 fra 45 til 135 osv. Disse sonene benyttes til å sette de relative waypointene slik at x- og y-retningen blir riktig. I tillegg estimeres avstanden fra motstanderen til bordkanten slik at denne er med på å avgjøre hvilken side man kjører til. I konkurransen ble denne avstanden satt til å være minimum 70 cm for at roboten skulle velge å kjøre til den aktuelle siden.

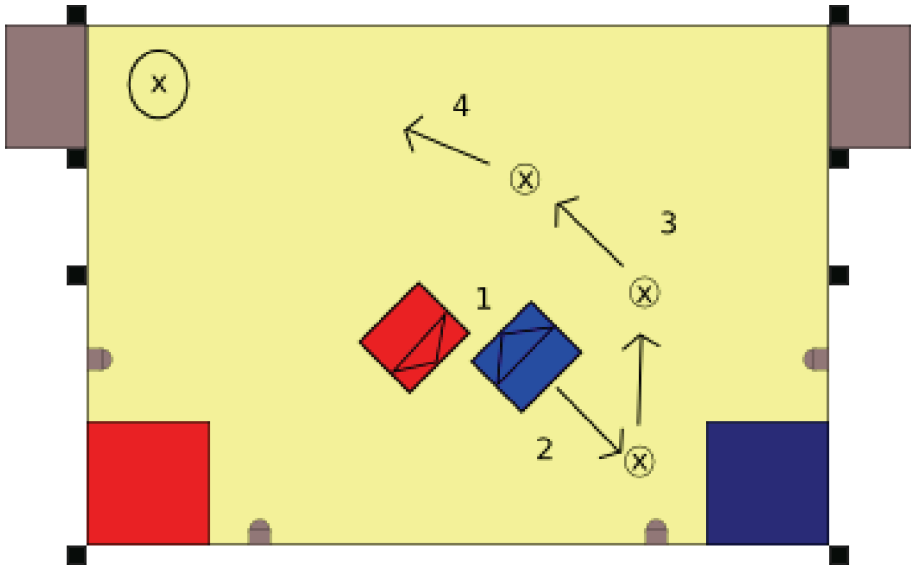


Figur 6.3: De 4 sonene som avgjør om det er venstre-, høyre- eller frontkollisjon.

Algoritme for kollisjonsunngåelse

Algoritmen kan enklest beskrives med utgangspunkt i figur 6.4. Den blå roboten er på vei mot waypointet som er markert oppe i venstre hjørne (tilsvarer waypointet ved standardkontaineren, roboten skal altså tømme magasinet for baller). Motstanderen er rød, og robotene møtes i punkt 1. Den blå roboten detekterer da en venstrekollisjon. Når en kollisjon detekteres, rygger roboten

bakover en definert lengde (30 cm ved venstre-/høyrekollisjon, 40 cm ved frontkollisjon). Når rygge-waypointet nåes i punkt 2, snur roboten seg 90 grader mot den siden den velger å kjøre forbi på. I punkt 3 kjører roboten mot de waypointene som blir generert av algoritmen. I punkt 4 snur roboten seg mot det opprinnelige waypointet og gjenopptar den tidligere aktive strategien. Antikollisjonssystemet er deaktivert mens roboten rygger, men aktiveres igjen når den har snudd 90 grader i punkt 2. Dersom roboten detekterer kollisjon på nytt mens den holder på med punkt 3, vil den rygge 20 cm bakover og snu seg mot det opprinnelige waypointet og strategien avsluttes. En eventuell ny kollisjonsdeteksjon medfører at algoritmen for kollisjonsunngåelse kjøres på nytt.



Figur 6.4: Eksempel på en kollisjonsunngåelse.

6.3.2 Deaktivering av antikollisjon

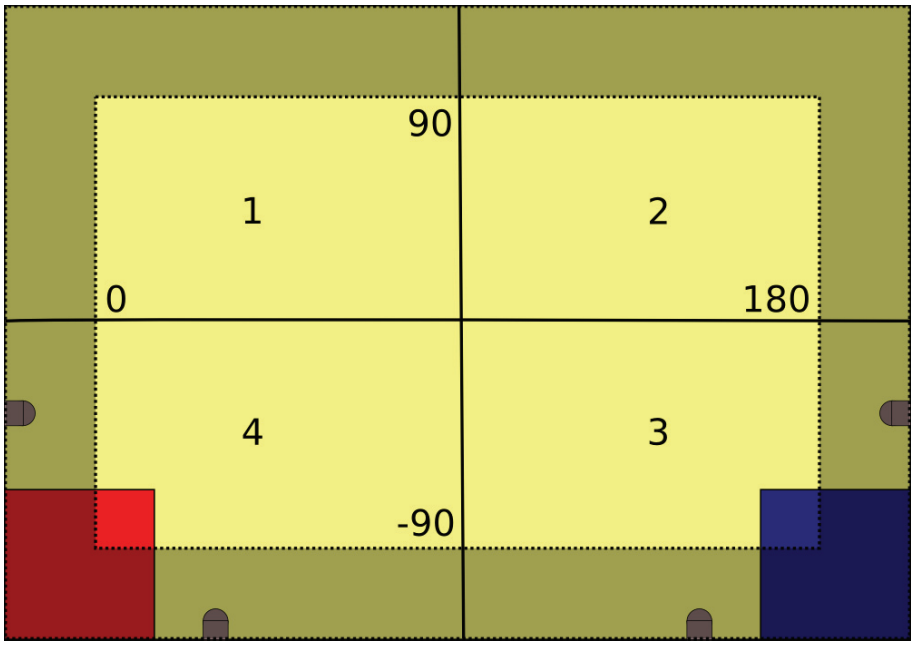
I og med at ultralydsensorene jevnt over viser for små avstander kan disse lett trigge kollisjoner som ikke er reelle. Det kan gjelde dispensere, containere og dommere rundt bordet. En løsning på det problemet er å deaktivere/overse meldinger når roboten befinner seg innenfor et definert område i forhold til bordkanten. Det er lagt til funksjonalitet for lett å kunne deaktivere/aktivere kollisjonssensorene.

Startstrategi

Problemet med trigging av kollisjoner som ikke er reelle, frambrakte ideen om å deaktivere antikollisjonen i startstrategien til AI. Det vil si at antikollisjonssystemet er deaktivert i starten, mens roboten fyller opp magasinet med de 5 første ballene. I det roboten skal snu seg rundt og kjøre for å tømme, vil antikollisjonssystemet aktiveres igjen. Dette anses som trygt da motstanderroboten neppe vil rekke å komme seg til vårt oppstartsområde før vi har begynt på den hvite dispenseren.

Soner

Antikollisjonssystemet deaktiveres når posisjonsmålingene tilsier at roboten er innenfor 30 cm fra bordkanten (se figur 6.5). Dette gjelder kun dersom roboten har en heading som peker mot bordkanten, da kan man gå ut fra at det ikke befinner seg noen motstander imellom. Det er definert 4 soner som vist i figuren. Dersom roboten har en heading som befinner seg innenfor sone 1, vil antikollisjonssystemet deaktiveres dersom den er innenfor de 30 cm fra de to veggene det gjelder. Ved å bruke disse sonene som beskrevet vil roboten kunne kjøre langs veggen med antikollisjonen aktivert, så lenge den ikke peker mot veggen.



Figur 6.5: Her vises de 4 sonene i tillegg til det båndet (skravert felt) rundt bordet der antikollisjon er deaktivert.

6.3.3 Utførte endringer i software

På bakgrunn av resultatene fra testene som ble kjørt på systemet ble det avgjort å ikke bruke Ir, noe som måtte taes høyde for i software. Dette ble fikset ved å sette alle Ir-målinger lik ultralydmålingene, slik at gjennomsnittsmålingene ble indentiske med ultralydmålingene. Dermed får AI kun ultralydmålinger.

I tillegg ble det lagt inn et buffer på mottakssiden i AI, der gyldige målinger ble regnet for å være snittet av et gitt antall målinger. Det ble testet med ulike bufferstørrelser, helt opp til 10. Med bufferstørrelse på 10, viste systemet seg å reagere for tregt. Til slutt endte man opp med bufferstørrelse på 3, noe som fungerte ganske bra under konkurransen. Med et slikt buffer vil plutselige hopp i målingene bli noe jevnet ut.

6.4 Testresultater

Mange tester ble kjørt på det nye antikollisjonssystemet, og resultatene var oppløftende. Hver gang roboten detekterte en kollisjon rygget den tilbake og kjørte rundt. Problemet var at den trigget kollisjoner som ikke fant sted. Under konkurransen ble det observert at den venstre sensoren lett trigget på baller som lå på bordet. Dette gjorde at den utførte unødvendige unnamanøvrer. Derimot kom den seg alltid forbi motstanderen, og kom seg alltid fram dit den skulle. Kombinasjonen av soner med deaktivering av antikollisjon og det faktum at LOS-algoritmen i navigasjonssystemet ikke godtar waypoints satt utenfor bordet (eller for nærme bordkanten), gjorde at den aldri ble stående fast ved kollisjonsunngåelse.

6.5 Konklusjon

Antikollisjonssystemet som ble overtatt fra i fjor var veldig ustabilt, ved at det hadde veldig lett for å trigge kollisjoner. Noen av grunnene til dette var dårlig logikk rundt behandlingen av kollisjonsmeldinger, samt forstyrrelse av Ir-sensorene. Ved å kutte ut bruken av Ir, og kun satse på ultralyd, ble systemet mer stabilt, fordi ultralyd ikke blir like lett forstyrret. I tillegg har bedret logikk på AI-siden gjort at behandlingen av kollisjonsmeldingene har blitt mer fornuftig. Innføring av buffer på målingene og deaktivering av antikollisjon i enkelte soner, har gjort at systemet ikke er så sårbart i forhold til feilmålinger. Selv om det ble observert under konkurransen at kollisjonssensoren foran på venstre side trigget på baller, er dette akseptabelt i forhold til ikke å trigge på en robot, eller som i fjor, da roboten trigget veldig ofte. Kol-

lisjonsunngåelsesalgoritmen har hjulpet roboten til å kjøre en fornuftig rute rundt hindringen, noe som fungerte utmerket under testing.

Kapittel 7

Datasyn

Årets konkurranse er mer deterministisk enn fjorårets, på den måten at vi har dispensere med faste posisjoner. I utgangspunktet behøves derfor ikke kamera, men når dispenserne er tomme må roboten begynne å plukke fra bordet. Her vil imidlertid ikke ballene ligge i faste posisjoner, i alle fall ikke utover i kampen. Derfor må ballene detekteres på en eller annen måte. Ifjor ble kamera benyttet til å finne flasker og bokser, og dermed eksisterer det allerede kamera og software for gjenkjenning av objekter (og kommunikasjon med AI). Det som er nytt i forhold til i fjor, er at det skal letes etter baller (sirkulære objekter), i stedet for noe mer komplekst som flasker og bokser. Fargene er også annerledes, nå er det hvit, rød og blå i stedet for gul (bokser) og grønn (flasker). Software tilpasset årets konkurranse baseres på arbeidet Lars Vråle gjorde i sin prosjektoppgave. Lars Vråle har også bistått i kalibreringen av kameraet.

7.1 Bakgrunn

Kamerakoden er basert på open-source OpenCV-biblioteket, et datasyn-bibliotek som lett kan taes i bruk i Linux. Dette biblioteket har god funksjonalitet rundt bildegjenkjenning. Gjenkjenning av sirkler er en velkjent problemstilling, og den mest brukte algoritmen i den sammenheng er Hough-transformen.

7.1.1 Hough-transformen

Hough-transformen er en teknikk brukt i bildeanalyse, datasyn og digital bildeprosessering. Formålet med teknikken er å finne gitte objekter i et bilde, basert på en "vote"-prosedyre på bakgrunn av diverse parametre. I hovedsak benyttes

algoritmen til å finne posisjonen til sirkler og ellipser i et bilde. OpenCV-biblioteket har implementert denne algoritmen. Dermed kan denne benyttes til å finne baller i et bilde. Den vanskelige biten er å tune parametrene til algoritmen, slik at algoritmen ikke finner for mange sirkler. Algoritmen returnerer altså posisjonen til alle sirkler den finner. Dette gjelder sentrum i sirkelen pluss sirkelperiferien.

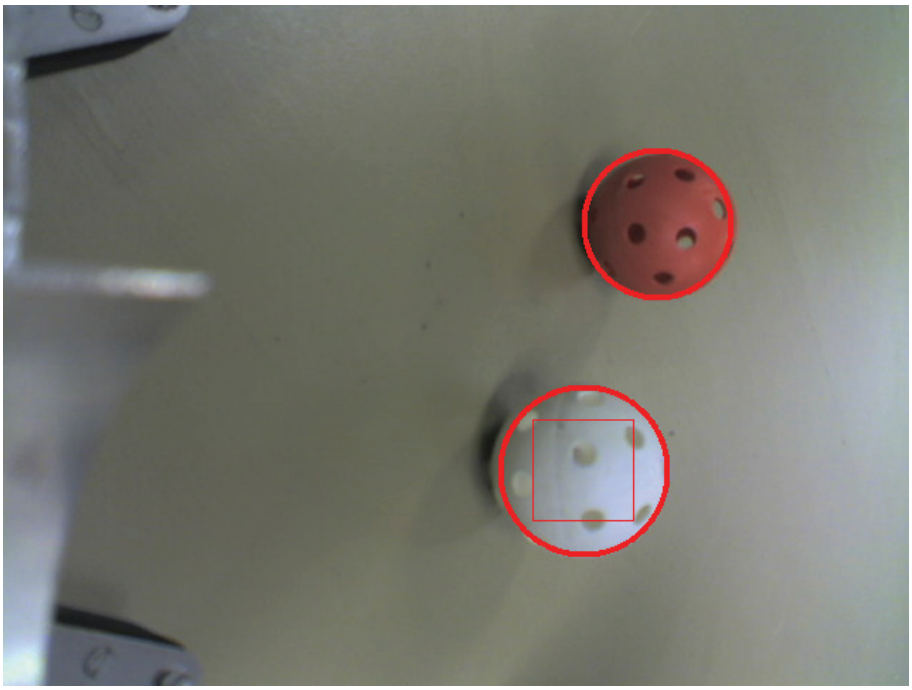
7.2 Hvordan avgjøre om det er en ball eller ikke?

Hver eneste sirkel som returneres av Hough-transformen må undersøkes. Er det en ball eller ikke? Dette er komplisert, da forskjellige lysforhold vil påvirke intensiteten til fargene. I utgangspunktet defineres et kvadrat omsluttet av sirkelen. Dette kvadratet sjekkes deretter for hvilken farge det inneholder. Bakgrunnen for ideen med kvadratet, var at det er enklere å søke gjennom piksler i et kvadrat enn i en sirkel. Dessuten er kvadratet definert såpass stort at det dekker mesteparten av ballen, og vil derfor i aller høyeste grad være representativt for ballen. Man går ut fra at dersom det er f.eks. en blå ball, vil mesteparten av kvadratet være blått. Til å begynne med ble RGB-verdiene for hver enkel piksel sjekket, og de med riktig farge ble summert. Dersom prosentandelen av riktige piksler var høyere enn en definert grense, ville ballen gjenkjennes som enten rød, blå eller hvit. Dersom en ball blir gjenkjent med riktig farge, vil kvadratet tegnes opp. I figur 7.1 ser kameraet to baller fra sin posisjon inne i roboten. Den røde ballen har ikke blitt gjenkjent som rød (intet kvadrat), mens den hvite gjenkjent som hvit. Figur 7.2 viser også 2 baller, men her har både den hvite og den blå ballen blitt gjenkjent. Problemet med denne metoden er at lysforholdene kan variere veldig. Da er det vanskelig å operere med statiske RGB-grenser og en gitt grense for prosentandelen av pikslene i kvadratet som skal være riktig farge. Det viste seg at denne metoden ble for usikker.

7.2.1 RGB-closeness

Utgangspunktet for RGB-closeness-metoden er at man vet (mest sannsynlig) at når man har funnet en sirkel, så er dette en ball. Da vet man også at ballen er rød, hvit eller blå. Derfor opereres det med 3 idealfarger som representerer fargene på de 3 ulike ballene under gitte lysforhold. Det vil si at det må en viss kalibrering til. RGB-verdiene til de 3 ulike fargene blir altså kalibrert og lagret som idealfarger. I koden gjenkjennes disse idealfargene som ROD-ROD, ROD-GRONN, ROD-BLAA for rød, og tilsvarende for blå og hvit.

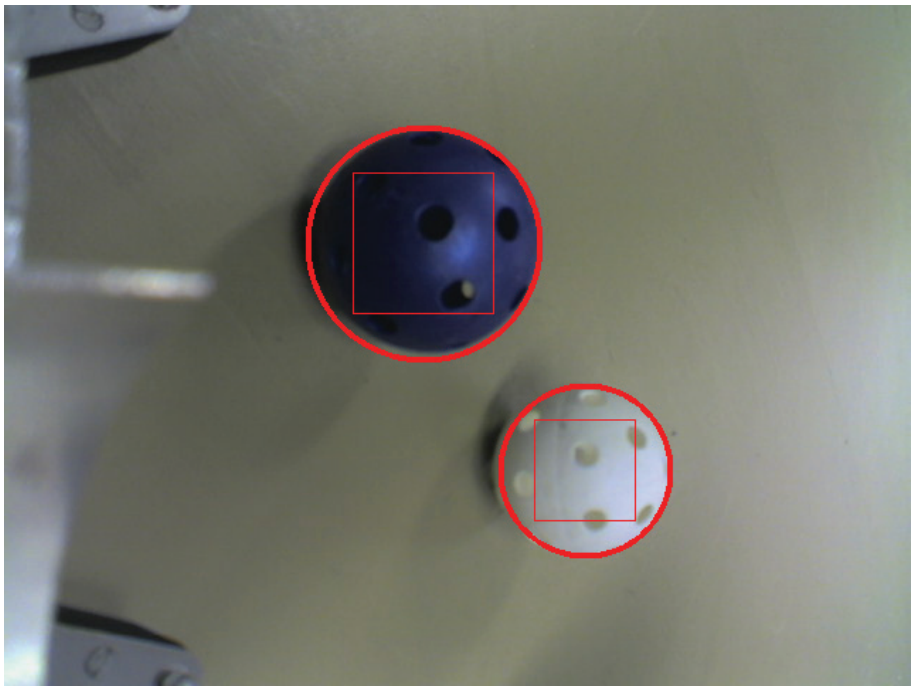
Først tar kamera et bilde, så kjøres Hough-transformen. Denne returnerer alle sirklene som er funnet i bildet. Deretter går man gjennom én og én sirkel.



Figur 7.1: En hvit og en rød ball som er gjenkjent av kamera.

Kvadratet blir definert, og alle pikslene blir gjennomgått. For hver piksel leses RGB-verdiene ut. Så summerer man opp R-, G- og B-verdiene for alle pikslene og deler til slutt på antall piksler. På denne måten får man gjennomsnittsverdien for R, G og B.

Nå kommer "closeness" inn i bildet. "Red-closeness" er definert som avstanden til idealfargen rød. Denne avstanden blir regnet ut vha. Pytagoras for 3 dimensjoner. På denne måten tilsvarer "red-closeness" summen av kvadratene til differansen mellom gjennomsnittsverdien og den ideelle verdien for henholdsvis R, G og B. Dette gjøres på samme måte for blå og hvit, for å sjekke hvilke av "closeness"-verdiene som er minst. Den minste har den minste avstanden til idealfargen, og på bakgrunn av dette konkluderes det med at det mest sannsynlig er den fargen. Pseudokode er vist i algoritme 7.2.1.



Figur 7.2: En hvit og en blå ball som er gjenkjent av kamera.

Algorithm 7.2.1: BALLGJENKJENNING(*objrec.cpp*)

```
for hver sirkel returnert av Hough-transformen
    Definer et kvadrat inni sirkelen
    for hver y-posisjon
        for hver x-posisjon
            meanR += R-verdien til pikselen;
            meanG += G-verdien til pikselen;
            meanB += B-verdien til pikselen;
            antallPiksler ++;

    meanR = meanR/antallPiksler;
    meanG = meanG/antallPiksler;
    meanB = meanB/antallPiksler;
    redCloseness = (meanR - ROD-ROD)*(meanR - ROD-ROD)+
                   (meanG - ROD-GRONN)*(meanG - ROD-GRONN)+
                   (meanB - ROD-BLAA)*(meanB - ROD-BLAA);
    blueCloseness = (meanR - BLAA-ROD)*(meanR - BLAA-ROD)+
                    (meanG - BLAA-GRONN)*(meanG - BLAA-GRONN)+
                    (meanB - BLAA-BLAA)*(meanB - BLAA-BLAA);
    whiteCloseness = (meanR - HVIT-ROD)*(meanR - HVIT-ROD)+
                     (meanG - HVIT-GRONN)*(meanG - HVIT-GRONN)+
                     (meanB - HVIT-BLAA)*(meanB - HVIT-BLAA);
    plukk ut den med minste closeness-verdi;
```

7.3 Kalibrering av posisjon

For å få ut den faktiske posisjonen relativt til roboten må kameraet kalibreres i forhold til plasseringen på roboten. Kameraet ble kalibrert ved hjelp av "Camera Calibration Toolbox for Matlab". Utgangspunktet for kalibreringen er at kameraet tar bilder av et sjakkbrett. For kalibreringsprosedyre, se [10].

7.4 Testing og resultater

Den endelige testingen av kamera ble gjort når roboten forøvrig var ferdig. Dette for å være sikker på at plasseringen av kameraet ble riktig, og for å kunne teste all funksjonalitet. De 5 viktigste kravene til funksjonalitet er definert i kravspesifikasjonen under.

7.4.1 Kravspesifikasjon

1. Kamera skal kunne gjenkjenne en henholdsvis hvit, rød og blå ball som ligger foran roboten.
2. Kameraprogrammet skal kunne gi posisjonen til ballen ut fra bildet som blir tatt.
3. Kameraprogrammet skal kunne ta et bilde og returnere en farge og posisjon dersom en ball er observert.
4. Kameraprogrammet skal plukke ut den nærmeste relevante ballen dersom det ligger flere baller foran kamera.
5. Kamera skal kunne detektere baller mens roboten er i bevegelse.

7.4.2 Testplan

Ut fra kravspesifikasjonen var det greit å sette opp en testplan:

1. Start kameraprogrammet og be dette om kontinuerlig å ta bilder for å identifisere baller (sirkler). Start uten baller foran roboten, legg så én og én ball foran, og se om programmet gjenkjenner ballene. Gjør dette for alle 3 fargene.
2. Ta bilder helt til en ball blir gjenkjent, sjekk så den posisjonen programmet returnerer.

3. Fra AI, be kameraprogrammet om å ta et bilde og let etter ball. Sjekk at AI får svar. Dersom kamera har funnet en ball, sjekk om riktig farge og posisjon er returnert.

4. Sjekk at posisjonen og fargen til den nærmeste og mest relevante ballen blir returnert riktig.

5. Be roboten om å kjøre litt framover og spør samtidig etter bilde. Se om kamera greier å detektere baller som ligger foran roboten.

7.4.3 Testresultater

Resultatene fra gjennomføringen av testene basert på testplanen, dannet utgangspunktet for hvor bra kamerafunksjonaliteten ble:

1. Etter de første testene viste deg seg at RGB-grensene som var satt for gjenkjenning av farge var for strenge. Fargene ble sjelden gjenkjent. I tillegg var Hough-transformen dårlig tunet. Etter å ha forandret tankegang til "closeness", fungerte gjenkjenningen veldig bra. Det eneste problemet som oppstod, var at røde og blå baller til tider kunne bli tolket som hvite. Dette kan skyldes lysforholdene. Sterkere lys vil gjøre at ballen virker lysere enn den er. Om ballen blir tolket som hvit er ikke noe problem, det hadde vært verre dersom en blå ball hadde blitt tolket som rød eller omvendt.

2. Når de to første punktene var ok, ble den faktiske returnerte posisjonen sjekket opp mot virkeligheten. Det viste seg å stemme meget bra, så konklusjonen ble at posisjonen til ballene var tilnærmet eksakt.

3. Med akseptabel gjenkjenning, og posisjon, ble AI satt til å be kameraprogrammet om ballobservasjoner, for så å kontrollere om AI har fått riktig farge og posisjon. Dette ble sjekket ved å be roboten om å kjøre til den ballen som ble observert, og plukke den opp. Resultatene av denne testen var veldig oppløftende, da gjenkjenning av posisjon og farge så ut til å fungere meget bra. Roboten kjørte til ballene og plukket dem opp. Dersom den ikke så noen baller, ble den stående i ro, ellers kjørte den umiddelbart mot ballen.

4. Denne testen ble kjørt ved å legge flere baller foran roboten. Den kjørte så og plukket opp den nærmeste ballen som ble gjenkjent. Dette så også ut til å fungere veldig bra.

5. Det kan være praktisk å kunne bevege på seg samtidig som man leter etter baller. Dette ble testet ved at AI kontinuerlig spurte etter ballobservasjoner mens roboten kjørte bortover mot et waypoint. Det ble først testet med vanlig hastighet. Da greide ikke kamera å detektere baller.

Hastigheten ble så halvert, og da kjørte roboten mot ballene og plukket dem opp.

Det ble også kjørt tester på synsfeltet. Ballene ble lagt gradvis lengre og lengre inn i synsfeltet til kameraet, helt til det ble detektert en ball. Til slutt konkluderte man med et synsfelt på nesten 30 cm i bredden og 50 cm i lengden.

7.4.4 Oppsummering

Søk-etter-ball-strategien i AI benytter seg av resultatene over. Når roboten kjører mellom waypointene i strategien søker den samtidig etter baller.

Det ble testet hvorvidt kamera greier å kjenne igjen baller i dispenserne. I utgangspunktet fungerte det delvis, forutsatt at man tok mange nok bilder. Ved å tune litt på Hough-transformen, oppnådde man at kameraet oppdaget flere sirkler slik at det tilnærmet alltid fant en ball som lå i en dispenser. Problemet nå var at kamera slet med å finne riktig farge på ballen. Det var enighet om at fargen ikke var så nøye da man alltid vet hvilken farge det er på ballene som befinner seg i dispenserne. Det var viktigere å kunne oppdage "tvilsomme" sirkler, det vil si sirkler som antagelig representerer baller. Baktanken med å bruke kamera til å oppdage baller i dispenserne, var at man kunne være sikker på å treffe en dispenser dersom posisjonen var noe usikker. Denne funksjonaliteten ble lagt til før de 2 siste kampene, men pga. de feilene som oppstod, ble den aldri testet ut i praksis under konkurransen.

7.5 Konklusjon

Konkurransen i år med røde, hvite og blå baller gjorde bildegjenkjenning lettere enn i fjor ved at man slipper å lete etter forskjellige objekter. Ved hjelp av Hough-transformen er det greit å finne sirkler i et bilde. RGB-sjekk av fargen på ballen ble først prøvet, men dette feilet og "closeness"-metoden ble valgt. Med denne metoden er man sikker på at man returnerer en ball, og mest sannsynlig med riktig farge. Eneste feil som kan oppstå er at en rød eller blå ball blir tolket som hvit. Dette er ikke noe problem, da man i utgangspunktet skal plukke opp hvite baller også. Derfor viste det seg til slutt at bildegjenkjenningen var veldig robust.

Kapittel 8

Kommunikasjon mellom modulene

8.1 Bakgrunn

Dette kapittelet er ment som et dokumentasjonskapittel for å presentere hvordan de enkelte modulene på roboten kommuniserer. All kommunikasjon på roboten foregår via CAN-buss eller Posix-meldingskøer, som ble implementert på roboten i fjor. Dette er nærmere beskrevet i Kjemphol og Knausgård 2006 [1]. I dette kapittelet vil det gjøres klart hvilke moduler som bruker hvilken kommunikasjon. Det viktigste å legge merke til, er at *gateway* står for oversettelsen mellom Posix og CAN. Derfor vil all kommunikasjon som går inn eller ut fra de interne programmene gå via *gateway*. De interne programmene er:

- AI (*plansys*)
- Navigasjonssystemet (*navsys*)
- Kamera (*cvsys*)

Ytre moduler:

- Sorteringsmodul
- Motordrivere
- Antikollisjonskort
- Spenningsvarslerkort
- Posisjoneringstårn

Kommunikasjonen internt mellom disse foregår via Posix-meldingskøer. Kommunikasjonen med de ytre modulene på roboten derimot, foregår via *gateway*.

Dersom f.eks. AI vil sende en melding til sorteringsmodulen må denne sendes til *gateway* som Posix-melding, deretter vil *gateway* sende den videre som en CAN-melding. På samme måte den andre veien, sorteringsmodulen sender en CAN-melding til *gateway*, og denne blir sendt videre som en Posix-melding.

Ved å legge opp kommunikasjonen på denne måten får man samlet all CAN-logikk på en plass, og de interne programmene slipper å forholde seg til noe annet enn Posix. Posix-meldingssystemet er nærmere beskrevet i masteroppgaven til Gunnar Kjemphol ([9]).

Utover i kapittelet vil de forskjellige modulene og deres kommunikasjon presenteres.

8.2 AI - Sorteringsmodul (CAN)

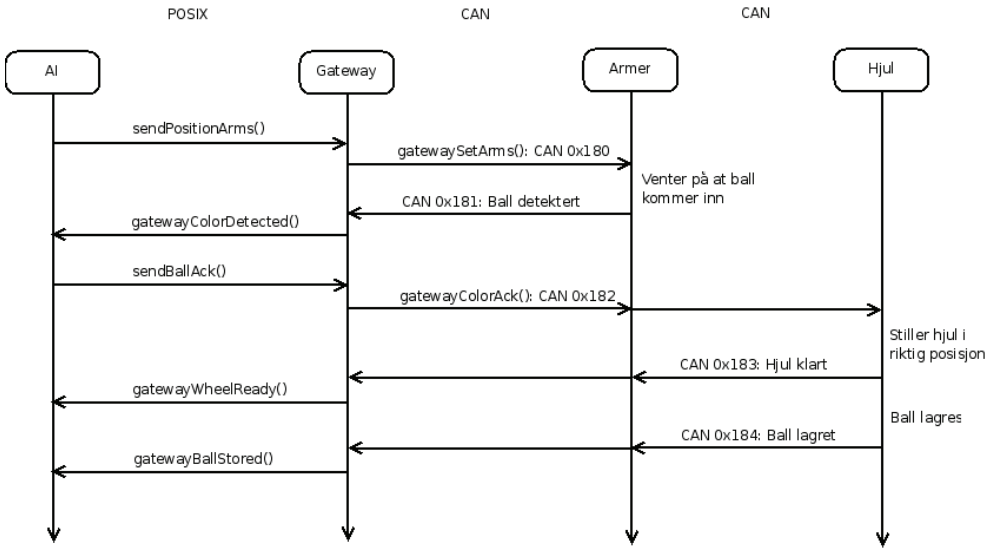
I dette avsnittet presenteres flyten i kommunikasjonen mellom AI og sorteringsmodulen. Det er to tilfeller som er interessante å se på; opplukking av en ny ball og tømming av magasin. Sorteringsmodulen vil fra nå av bli omtalt som enten *armene* eller *hjulet*, fordi den består av nettopp disse to delmodulene (se kapittel 2). De CAN-meldinger som blir sendt mellom AI og sorteringsmodul er følgende:

- **0x180**, AI → armer: Posisjonér armer og bånd. Denne meldingen består av 3 data-bytes. Den første definerer vinkelen til armene (mellom 0 og 90 grader). Byte nummmer 2 og 3 inneholder henholdsvis av/på (0/100) og retning på båndene (0 for innover og 1 for utover).
- **0x181**, Armer → AI: Ny ball med en gitt farge detektert. Meldingen består av kun en data-byte og denne inneholder fargen på ballen.
- **0x182**, AI → armer/hjul: Godta/forkast ny ball med en gitt farge. Data-byte 1 inneholder 1 eller 0, avhengig om ballen aksepteres eller ikke. Byte 2 og 3 inneholder fargen på ballen og den fargen AI tror neste ball har.
- **0x183**, Hjul → armer/AI: Hjul klart for lagring av ny ball.
- **0x184**, Hjul → AI: Ball lagret unna i hjul.
- **0x185**, AI → hjul: Tøm magasin for baller. Meldingen består av en data-byte som sier hvilken farge som skal sorteres først.
- **0x186**, Hjul → AI: Magasin tømt. 5 data-bytes representerer hver de 5 ballene som er sortert, der den siste byten tilsvarer den siste ballen.
- **0x188**, AI → armer/hjul: Reset sorteringsmodulen.
- **0x189**, AI → armer/hjul: Stopp sorteringsmodulen.

To eksempler på kommunikasjonsflyten mellom modulene er ballopplukking og magasintømming som presenteres under.

8.2.1 Ballopplukking

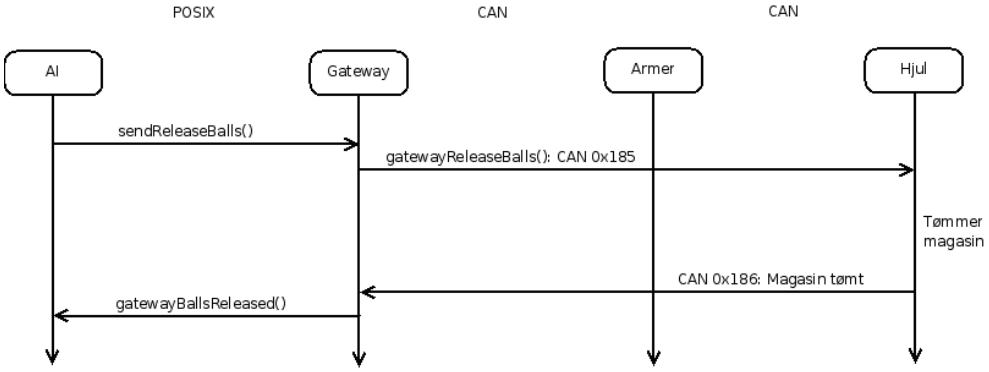
Sekvensen er vist i figur 8.1. Det første som skjer ved en ballopplukking er at AI sender beskjed (0x180) om å posisjonere armene 90 grader ut, slik at disse klemmer om ballen. Samtidig ber AI om at båndene skal gå innover slik at ballen vil bli dratt inn i roboten. Deretter brytes Ir-strålen og armene sender 0x181 med informasjon om farge på ballen. Denne meldingen blir sendt til *gateway*, som videresender denne som en Posix-melding til AI. Dersom den opplukkede ballens farge tilsvarer AI sitt ønske, sender AI 0x182 med databyte 1 = 1 tilbake. Når armene og hjulet mottar denne meldingen, sender hjulet *wheelReady*-melding (0x183) tilbake når hjulet er klart. Når armene mottar *wheelReady*, kjøres ballen inn og den blir lagret i hjulet. Hjulet sender så en *ballStored*-melding (0x184) når ballen er lagret og hjulet har stilt seg klar til neste ball.



Figur 8.1: Sekvensdiagram som viser meldingene som går mellom modulene/programmene ved opplukking av ball.

8.2.2 Magasintømming

Denne sekvensen er kort og grei, men viser nok et eksempel på hvordan kommunikasjonen foregår. Figur 8.2 viser hvordan AI og hjulet samarbeider når magasinet skal tømmes. AI sender en *releaseBalls*-melding (0x185), og hjulet kjører igang og tømmer magasinet fortløpende. Når det er gjort, sendes en *ballReleased*-melding (0x186) tilbake.



Figur 8.2: Sekvensdiagram som viser meldingene som går mellom modulene/programmene ved tømming av magasin.

8.3 AI - Antikollisjonskort (CAN)

Kommunikasjonen mellom AI og antikollisjonskortene er enveis fordi antikollisjons-master-kortet sender sensormålinger til AI uoppfordret. Slavekortet sender kontinuerlige oppdateringer til master-kortet slik at master-kortet hele tiden kan sende ferske målinger til AI. Meldingene som blir sendt er:

- **0x80**: Sensormålinger fra master-kortet til AI.
- **0x81**: Oppdateringer fra slavekortet til master-kortet.

8.4 AI - Spenningsvarslerkort (CAN)

Det går lite trafikk mellom AI og spenningsvarslerkortet. Ved oppstart av totalsystemet vil AI vente på farge og start fra spenningsvarslerkortet. Så fort AI har fått fargen, sender den en melding tilbake som setter riktig farge på spenningsvarslerkortet. Ellers vil et trykk på reset-knappen trigge en restart av totalsystemet, og lav batterispenning vil medføre sending av en *lowBattery*-CAN-melding. De meldingene som inngår i kommunikasjonen er:

- **0x06**: Lav batterispenning, sendt fra spenningsvarslerkortet.
- **0x10**: Startinterrupt trigget, sendt fra spenningsvarslerkortet.
- **0x11**: Reset-knapp trykket, sendt fra spenningsvarslerkortet.
- **0x15**: Team-farge blått trykket, sendt fra spenningsvarslerkortet.
- **0x16**: Team-farge rødt trykket, sendt fra spenningsvarslerkortet.
- **0x134**: Sett blå diode aktiv, sendt fra AI.
- **0x135**: Sett rød diode aktiv, sendt fra AI.
- **0x136**: Reset modul, sendt fra AI.

8.5 AI - Kamera (Posix)

Den enkleste kommunikasjonen i hele totalsystemet foregår her, kun en Posix-meldings-id er definert:

- **MINOR_CV_OBSERVATION**

Denne ene id-en blir brukt til både sending og mottak på begger sider. AI ber kamera om et bilde, og kamera svarer med data.

8.6 AI - Navigasjonssystemet (Posix)

AI og navigasjonssystemet kommuniserer direkte med hverandre gjennom Posix, uten å gå via *gateway*. Meldingsoversikten er noe større her enn for kamerakommunikasjonen:

- **MINOR_NAVSYS_POSITION**: Posisjonsoppdatering, sendt fra navigasjonssystemet.
- **MINOR_NAVSYS_REACHED_WAYPOINT**: Waypoint nådd, sendt fra navigasjonssystemet.
- **MINOR_NAVSYS_SET_WAYPOINT**: Nytt waypoint fra AI (lokale koordinater).
- **MINOR_NAVSYS_SET_WAYPOINT_ROBOT_RELATIVE**: Nytt waypoint fra AI (robotrelative koordinater).
- **MINOR_ROBOT_TO_LOCAL**: Robotrelativ posisjon fra AI, AI ber om å få posisjonen i lokale koordinater.
- **MINOR_NAVSYS_SET_SPEED_LIMIT**: Hastighetsbegrensning fra AI.
- **MINOR_NAVSYS_SET_SPEED**: AI ber om å sette hastigheten.
- **MINOR_NAVSYS_SET_POSITION**: Forespørsel fra AI om å sette posisjonen til det AI ønsker.
- **MINOR_NAVSYS_ADJUST_POSITION**: Avstandsreguleringsmeldinger fra AI.
- **MINOR_BEACON_MEASUREMENT_REQUEST**: AI ber navigasjonssystemet om å starte målinger fra posisjoneringstårnet.

8.7 Navigasjonssystemet - Motordrivere (CAN)

Navigasjonssystemet kommuniserer med motordriverene gjennom *gateway*. Her presenteres kun de CAN-meldingene som blir sendt fram og tilbake. I praksis er det *gateway* som spør om kvadraturtellermålinger fra hjulene, men målingene blir sendt som Posix-meldinger til navigasjonssystemet. CAN-meldingene som blir sendt er:

- **0x70**: Kvadraturtellermålinger fra venstre hjul til navigasjonssystemet.
- **0x71**: Kvadraturtellermålinger fra høyre hjul til navigasjonssystemet.
- **0x110**: Sett motorpådrag fra navigasjonssystemet.

- **0x113**: Navigasjonssystemet ber om kvadraturtellermålinger fra venstre hjul.
- **0x118**: Navigasjonssystemet ber om kvadraturtellermålinger fra høyre hjul.

8.8 Navigasjonssystemet - Posisjoneringstårn (CAN)

Posisjoneringstårnet starter målinger når det får CAN-melding fra *gateway*. Det er kun 2 CAN-meldinger som inngår i kommunikasjonen mellom navigasjonssystemet og posisjoneringstårnet:

- **0x120**: Posisjoneringstårnet returnerer målinger til navigasjonssystemet.
- **0x122**: Navigasjonssystemet ber om målinger fra posisjoneringstårnet.

Kapittel 9

Diverse arbeid

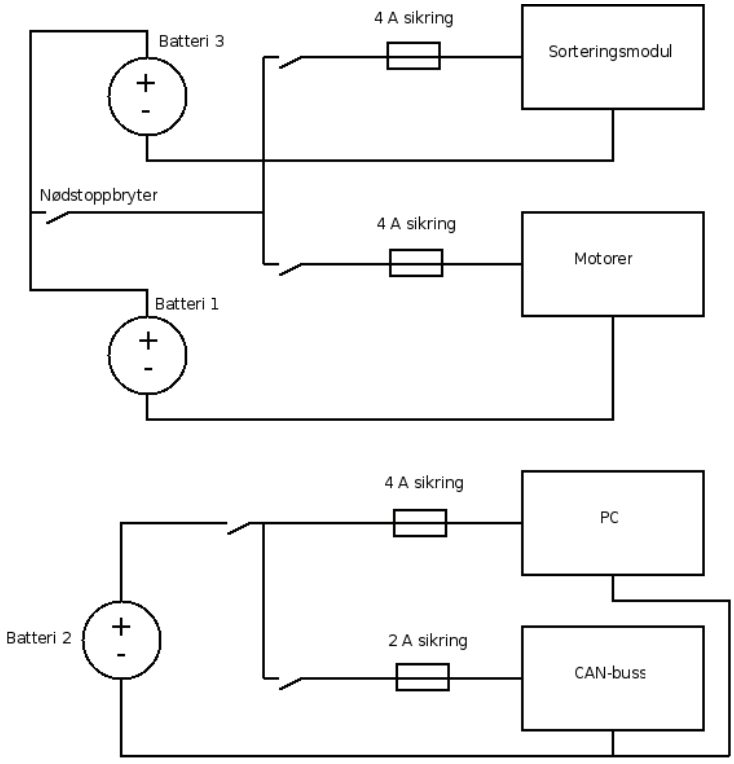
9.1 Fysisk utforming av roboten

9.1.1 Koblingsboks

På roboten er det nødvendig med noen av/på-brytere for å koble strøm til de ulike modulene. Som beskrevet i Mørkrid og Øien 2007 [4], brukes det LIPO-batterier som strømforsyning, og det har etter hvert blitt bestemt å bruke 3. Dette gir god kapasitet/batterilevetid og adskiller strømforsyningen til flere av modulene, slik at de er lite avhengig av hverandre. Batteriene er fordelt slik:

1. Motorer for fremdrift
2. PC og moduler som får strøm fra CAN-buss
3. Sorteringsmodul

Hvert batteri er koblet innom en av/på-bryter og en sikring. I tillegg er det ekstra bryter og sikring for batteriet som skal forsyne både PC og CAN-buss (batteri 2). Det er valgt å samle alt dette på ett sted for enkelt å kunne slå av de ulike delene. I tillegg er det et krav at roboten skal være utstyrt med en nødstoppbryter, som ved nedtrykking skal stoppe alle aktuatorer. Alle aktuatorer vil i dette tilfellet innebefattes av batteri 1 og 3, og disse er derfor koblet direkte via nødstoppbryteren. Oppkoblingen av koblingsboksen er vist i figur 9.1.

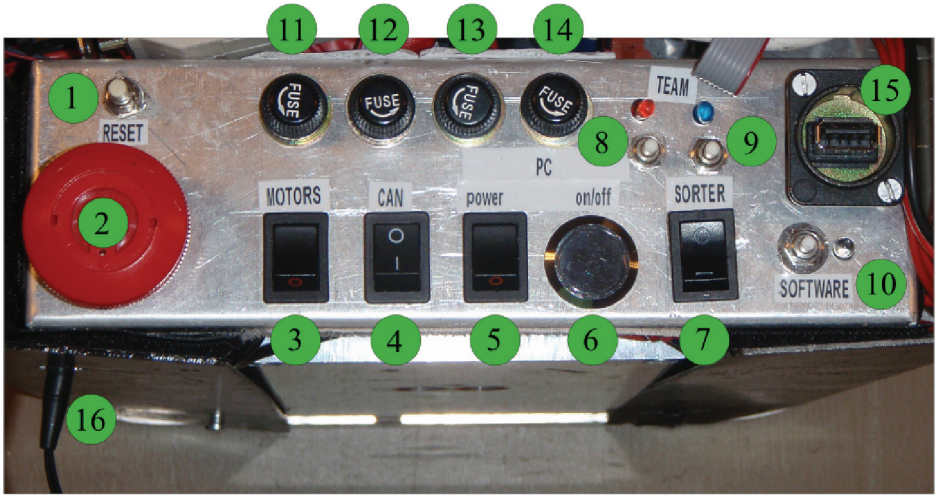


Figur 9.1: Koblingsskjema for koblingsboks

Bryterpanel

Etter hvert har det blitt en del brytere for til- og frakobling av batterier, samtidig som det er nødvendig med noen brytere til andre formål bekskrevet nedenfor. For å skape oversikt og tilgjengelighet, er det valgt å samle alle brytere på toppen av koblingsboksen. Denne er sentralt plassert bak på roboten og danner et bryterpanel for styring av alle modulene. Figur 9.2 viser hele panelet med nummereringer og hvert punkt er kort forklart under.

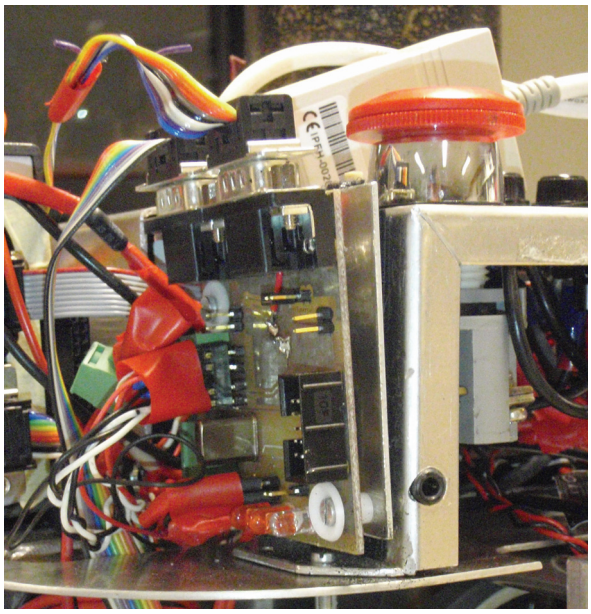
1. **Reset-knapp** - bryter for å resette alle hardwaremoduler som er koblet til CAN-bussen
2. **Nødstoppbryter** - stor og lett tilgjengelig bryter for bryting av strøm til alle aktuatorer, dvs. drivmotorer og sorteringsmodul.
3. **Motorer** - av/på-bryter for strømtilkobling til drivmotorer
4. **CAN-buss** - av/på-bryter for strømtilkobling til CAN-buss
5. **PC power** - av/på-bryter for strømtilkobling til PC
6. **PC on/off** - bryter for å slå på PC
7. **Sorteringsmodul** - av/på-bryter for strømtilkobling til sorteringsmodul
8. **Rød teamfarge** - bryter for valg av rødt lag, med tilhørende statusdiode
9. **Blå teamfarge** - bryter for valg av blått lag, med tilhørende statusdiode
10. **Software** - bryter for oppstart av software, med tilhørende statusdiode
11. **Motorsikring** - 4A sikring for strømforsyning til drivmotorer
12. **CAN-bussikring** - 2A sikring for strømforsyning til CAN-buss
13. **PC-sikring** - 4A sikring for strømforsyning til PC
14. **Sorteringsmodulsikring** - 4A sikring for strømforsyning til sorteringsmodul
15. **USB** - lett tilgjengelig USB-inngang til PC
16. **Startsnor** - mini-jack-plugg for startsnor



Figur 9.2: Oversikt over bryterpanelet

9.1.2 Spenningsvarsler

Spenningsvarslerkortet som ble utviklet høsten 2007, viste seg å være for stort til å passe inn på roboten etter at sorteringsmodulen og koblingsboksen var på plass. Dermed måtte det modifiseringer til for å få plassert kortet på siden av koblingsboksen. Det var hensiktsmessig å plassere det på siden av koblingsboksen pga. ledninger til brytere og dioder. Kortet ble forminsket en del, og nytt kretsskjema ligger i vedlegg A. Et bilde av den ferdige realiseringen av kortet er også vist i figur 9.3.

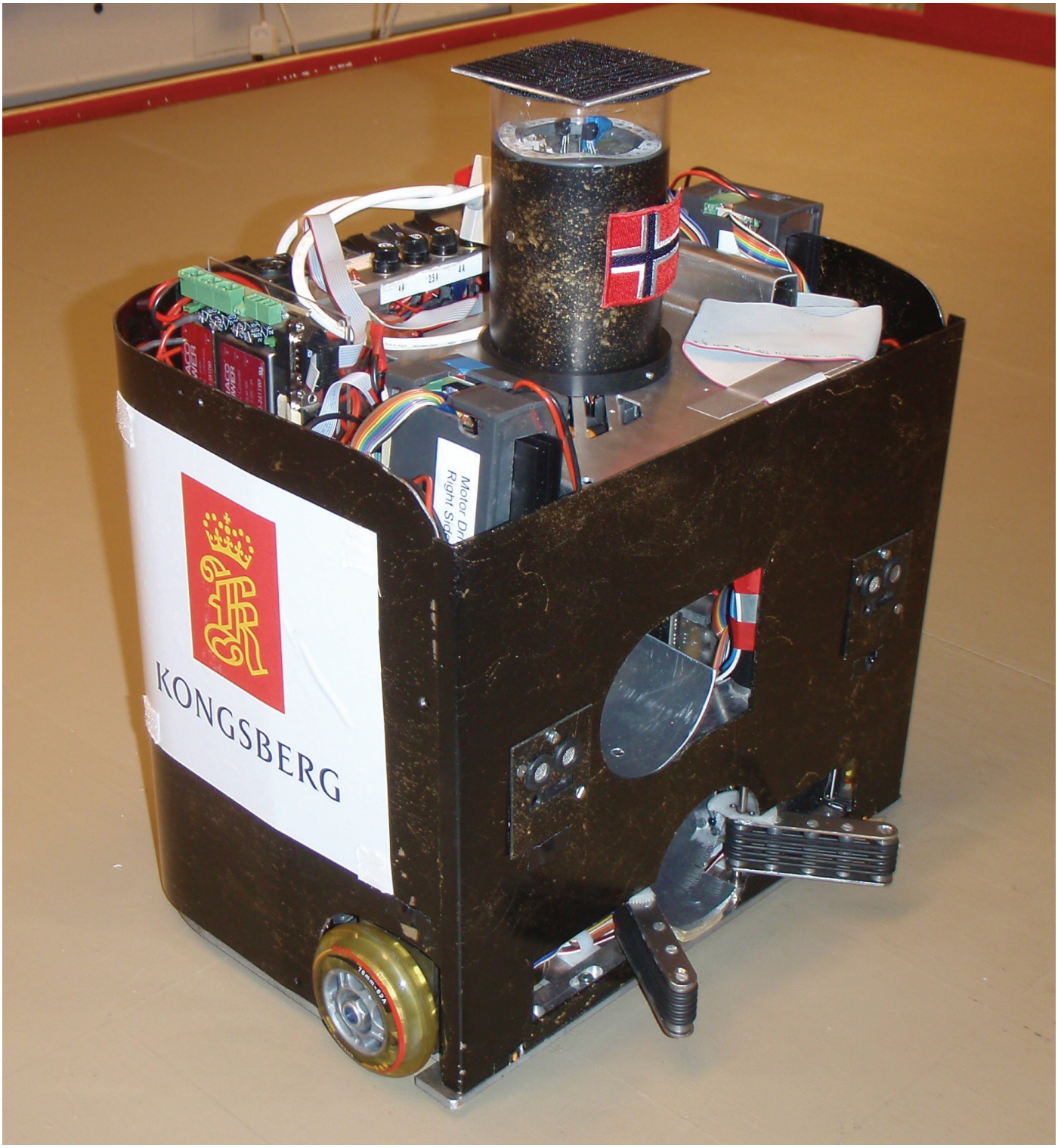


Figur 9.3: Ferdig spenningsvarslerkort

9.1.3 Deksler

Mot slutten av arbeidet med roboten, da alle moduler og deler var som de skulle, ble det konstruert deksler til roboten. Dekslene danner en beskyttelse av all elektronikk og vitale deler, samtidig som det holder ledninger etc. innenfor den lovlige omkretsen. Dekselet gir også flater til å feste startnummer og sponsormerker, samt at det gir roboten et mer helhetlig og ryddig utseende.

Dekslene ble, i samarbeid med EiT, laget med utgangspunkt i fjorårets deksler. Sidedekslene ble tilpasset det litt modifiserte oppsett med hjul o.l., mens det ble laget et helt nytt frontdeksel. Det ble laget fester med tanke på at dekslene skulle være lett å ta av og på, men samtidig sitte ordentlig fast. For å ha en nøytral, lovlig farge som ikke kunne forveksles med noen av spillets elementer, ble det valgt å bruke svart. Alle overflater ble grunnet og lakket med svart spraylakk. Til slutt ble det også dekorert med en slags gullmarmorering, for å kunne skille seg litt ut utseendemessig. For å promotere sponsoren ble det plassert et stort Kongsberg-klistremerke på hver side av roboten. Roboten med ferdige deksler er vist i figur 9.4.



Figur 9.4: Robot med ferdige deksler

9.2 Oppstartsprosedyre

Et gjennomgående problem i Eurobot-sammenheng de siste årene har vært å få til en rask og smertefri oppstartsprosedyre i forbindelse med konkurranse. Ettersom roboten blir ganske komplisert og inneholder mange elementer som skal samarbeide, er det mye som må gjøres riktig ved oppstart. Fjorårets oppstartsprosedyre var relativt problematisk pga. veldig lang oppstartstid på PC-en, i tillegg til at 4 ulike programmer måtte startes fra kommandolinjen. Disse problemene er beskrevet og delvis løst i Mørkrid og Øien 2007 [4], men det er ønskelig å gjøre enda flere forbedringer. For det første er det stressende å måtte koble seg til roboten via en krysset nettverkskabel i forkant av hver match, ikke minst fordi man må skrive en del kommandoer i kommandolinjen for å få igang oppstartsskriptet. I tillegg tar prosedyren lang tid og det er vanskelig å beregne når man må sette igang for ikke å tømme batteriene. Følgende ønsker dannet utgangspunkt for en ny prosedyre:

- Det skal ikke være nødvendig med en ekstra laptop og krysset nettverkskabel ved oppstart
- All softwaren skal kunne startes med et knappetrykk etter at PC'en har startet
- Det skal være enkelt å se når PC'en er klar, når programmene starter og ikke minst at alt har startet riktig
- Hvis en feil oppstår, skal all software kunne resettes

Etter noen undersøkelser, og god hjelp fra Gunnar Rangøy fra EiT, ble det utviklet en løsning som baserer seg på bruk av parallellporten på hovedkortet som inngang/utgang. Utgangspunktet var et tidligere utviklet skript som starter alle de 4 softwaremodulene; gateway, AI, navigassjonssystem og kamera. Ved å lage et enkelt c-program som lytter på en parallellport-pinne, kan man starte skriptet ved å kortslutte denne pinnen mot jord. Tilsvarende kan man sette en utgang høy og koble på en diode og en motstand for å få satt et lys. Det er viktig å merke seg at parallellporten kan være ganske skjør og at man må passe på å bruke motstander for å forhindre store strømmer.

Koden for oppstartsprogrammet er skissert i pseudokoden under. Når programmet er klart vil dioden bli stående å blinke helt til knappen blir trykket. Deretter vil den slukke mens alle programmene starter og lyse igjen når alle er igang, dvs. oppstartskriptet har kjørt ferdig. Programmet har også støtte for reset ved at man holder knappen inne i minst 3 sekunder. Ved reset vil alle prosesser stoppes, og man går igjen inn i starttilstanden der man venter på knappetrykk. Det hele skal være veldig intuitivt å bruke og statusdioden gir hele tiden god tilbakemelding på hva som skjer.

Algorithm 9.2.1: OPPSTARTSPROGRAM(*paraboot.c*)

```
while true
  do
  { system: killall robotSoftware;
  { while buttonNotPressed
    do
    { flashLED;
    { checkForButtonPress;
  { LEDOff;
  { system: start robotSoftware (run script);
  { LEDOn;
  { while buttonNotHeldFor3Seconds
    do
    { if buttonHeldFor3Seconds
      then { LEDOff;
           { break;
           { sendResetSignal;
```

For å bli helt uavhengig er oppstartsprogrammet lagt inn i */etc/rc.local* slik at det starter automatisk ved boot av operativsystemet, uten at man trenger noen innlogging. Straks PC'en er ferdig med å boote, vil statusdioden for software begynne å blinke. Resultatet er at hele oppstartsprosedyren tar rundt 40 sekunder, noe som gir god margin til de 3 minuttene man kan bruke ifølge reglene. Dette ble brukt med stor suksess under testing og i konkurranse.

9.3 Kontinuitet i videre deltagelse

Gjennom årene med Eurobot-deltagelse har det blitt utviklet mange gode konsepter og løsninger, men det har ofte vært vanskelig å få brukt dette i senere konkurranser. Det finnes mange årsaker til dette, både manglende modularitet og dokumentasjon, samtidig som nye oppgaver krever nye løsninger og i mange tilfeller en helt annen utforming av selve roboten. Gjennom de to siste deltagelsene har det imidlertid vært større fokus på å få kontinuitet i Eurobot-arbeidet ved NTNU, og dette er noe man har forsøkt å bidra til i denne oppgaven. Kontinuitet og videreføring av både kunnskap, erfaringer og fysiske moduler, er helt nødvendig for at NTNU skal kunne hevde seg på et høyere nivå i konkurransen. Det er viktig å legge merke til at de beste lagene som regel stiller med mange deltagere, samme deltagere over flere år og god støtte fra sitt faglige miljø på universitetet. Robotene de bygger er ikke nødvendigvis mer kompliserte enn de som bygges på NTNU, ofte tvert imot, men modulene er velfungerende og nøye tilpasset konkurransesituasjonen.

9.3.1 Gjennomgang for neste års lag

Ved overtagelse av Eurobot-prosjektet, er det mye å sette seg inn i dersom man skal ha et ønske om å kunne gjenbruke deler videre. Til tross for velvilje fra en av fjorårets deltagere, Kristian M. Knausgård, var det vanskelig å få oversikt over de ulike delsystemene i roboten. Spesielt var det tungt å komme inn i tenkemåten for den ganske store og kompliserte programvaren på hovedkortet. For å forbedre dette ble neste års deltagere, Kristian Kjølseth og Øystein Wergeland, invitert til en grundig, muntlig gjennomgang av all software og hardware på roboten. Dette vil forhåpentligvis gi dem en god forståelse og innsikt i dagens system, og kan være en fordel for å komme raskere igang til høsten. Det har i hele vår vært en god kontakt og kommunikasjon med disse personene gjennom samarbeidet med EiT. Interessen for å bidra videre hos årets deltagere er absolutt tilstede, men realistisk sett blir dette først og fremst i form av rådgivning og forklaring på moduler som overtas.

9.3.2 Samling av software og andre ressurser

Tidligere har programvare fra Eurobot i større eller mindre grad blitt samlet etter at arbeidet er avsluttet. Som regel følger det med en CD til rapporten, men denne kan ofte være mangelfull og kan lett forsvinne etter å ha vært innom flere personer. Noen år har det også blitt opprettet et eget gruppeområde på skolen, men informasjonen fra år til år blir lett spredt og det kan være vanskelig å få tilgang til dette i ettertid. Enda verre er tilfellet der det har vært opprettet en egen server under prosjektet, som i ettertid er vanskelig å få kontroll over og i det hele tatt finne tak i, hvis man ikke har løpende kontakt med de som opprettet den.

I år har det blitt innført et nytt og mer generelt samlingssted for Eurobot-data. Ved hjelp av orakelansatt og EiT-deltager Henrik Austad er det opprettet et nytt gruppeområde "eurobot" der alle aktuelle data har blitt lagret underveis. På dette området finnes det flere SVN-repositoryer og adskilte mapper for bilder og datablader. SVN har med stort hell vært brukt tidligere, men det har blitt litt problematisk at alt av bilder og store datablader har ligget i hovedrepositoryet slik at det har blitt unødvendig stort og tungvint å få hentet ut en hel versjon. All EiT-kode har også blitt lagt på dette hjemmeområdet slik at man til enhver tid kan hente ut siste versjon, dersom det er nødvendig med endringer. Ved overgang til en ny oppgave vil det antagelig være fornuftig å flytte alt inn i en ny mappe, f.eks. kalt "2008", slik at de aktuelle filene kan ligge lettest mulig tilgjengelig.

9.4 EiT

Gjennom hele semesteret har vi hatt støtte fra 2 EiT-grupper. Disse har hatt som oppgave å lage en sorteringsmodul til roboten, nærmere beskrevet i avsnitt 2.2. Rekrutteringsarbeidet ble unnagjort på forhånd og er nærmere beskrevet i Mørkrid og Øien 2007 [4].

9.4.1 Oppgavegivning

For å dra mest mulig nytte av dette samarbeidet ble det lagt føringer på hvordan oppgaven skulle løses, og hvilke tidsfrister som skulle gjelde. Det var et ønske om å beholde hovedkonstruksjonen på roboten, så her var det bare rom for små justeringer. I tillegg ble det definert omtrentlig hvor ballene skulle inn og ut. For å engasjere begge gruppene best mulig ble det valgt å la dem starte med å utvikle hver sin totalløsning for plukking og sortering av ballene. Etter en stund skulle de slå seg sammen og samarbeide om en felles løsning. Denne samlingsfasen krevde mye oppfølging og gruppene måtte ha en del hjelp for å bli enige om et godt konsept.

9.4.2 Ferdigstilling og testing

Fra starten av prosjektet ble det fokusert på at hele roboten skulle være ferdig på et ganske tidlig tidspunkt, da testing erfaringsmessig tar mye tid. Ovenfor EiT-gruppene ble det tidlig gjort klart at alt måtte fungere innen 1. april. Dette viste seg å bli vanskelig og problemer og feil førte til en forsinkelse på flere uker. Det er på ingen måte dårlig planlegging som førte til dette, men rett og slett oppgavens størrelse og begrensningen av hva man kan kreve av personer som kun er med gjennom et vanlig fag på 7,5 studiepoeng. Mye frivillig og glimrende innsats førte til at det hele kunne testes sammen med totalsystemet og at alt ble ferdig og fungerende til konkurransen.

9.4.3 Konklusjon av samarbeidet

Samarbeidet med EiT har utvilsomt vært veldig suksessfullt og bidraget har kommet til stor nytte og spart oss for mye arbeid. Likevel har det vært en jobb å administrere dette, og kanskje i større grad en man hadde forventet på forhånd. Å sørge for at gruppene har ønsket fremgang, samtidig som arbeidskapasiteten deres varierer med andre plikter, har tatt mye tid. I tillegg har roboten måttet stilles til disposisjon i store deler av utviklingsarbeidet, og dermed forsinket det øvrige arbeidet noe. I testperioden og sluttfasen fram mot konkurransen var det, forståelig nok, mindre hjelp å få til justeringer og

utbedringer av feil pga. eksamen o.l. Disse punktene underbygger viktigheten av å få tak i interesserte og arbeidsvillige studenter. For senere år anbefales det imidlertid å se på om oppgaven bør gjøres noe mindre, eller om man på andre måter kan tilrettelegge bedre for økt arbeidsmengde fra EiT.

9.5 Økonomi

Økonomi og finansiering er beskrevet i Mørkrid og Øien 2007 [4].

Sponsorpengene har de siste årene blitt satt inn på en konto som administreres av instituttet ved kontorsjef Tove K. B. Johnsen. På denne måten har alle utlegg blitt registrert på et sted og ingen penger har forsvunnet i overgangen fra år til år. Ulempene med dette systemet har imidlertid vist seg tydelig under årets arbeid. Instituttet krever at det leveres kvittering med ekstra forklaring/beskrivelse på hvert eneste utlegg. I tillegg blir pengene utbetalt med uforutsigbar forsinkelse og kun til én av de to registrerte personene på Eurobot. Dette medfører at alle faktureringer og utlegg til f.eks. EiT, må gå gjennom disse personene uten noen garanti for at det blir tilbakebetalt før etter en uke eller to. Ettersom det er snakk om betydelige innkjøp i noen perioder, er dette en unødvendig belastning i tillegg til at spesielle komponenter/deler må forklares og forsvares overfor en ikke-faglig tredjeperson.

Hovedproblemet oppstod da sponsorpengene fra Kongsberg skulle overføres til instituttets konto, vha. en faktura fra instituttet. Etter flere uker med purring på utsendelse, ble det gitt tilbakemelding fra økonomiavdelingen om at det skulle betales moms av disse pengene, og at fakturautsendelsen inntil videre var avbrutt. Med flere titalls tusen kroner i utlegg, bl.a. for reise til Heidelberg, ble det besluttet å åpne en personlig konto og be hovedsponsoren om å betale til denne. Ved klarsignal fra Kongsberg, ble det sendt en egen faktura på e-post, og pengene var disponible på egen konto etter én uke. Et system med egen konto har gjort det mye lettere å ha kontroll på aktuell saldo, samt raskt kvitte seg med gjeld og utlegg. Denne økonomistyringsmodellen anbefales videre, men det er viktig å strukturere registrering av utlegg/fakturaer.

Som beregnet i budsjettet, går prosjektet med et betydelig overskudd i år. Dette er ment som starthjelp for neste års prosjekt, da sponsorpengene ofte viser seg å komme ganske sent. Pengene overføres etter ønske til neste års deltagere.

9.5.1 Hovedsponsor

Kongsberg har vært sponsor for Eurobot-prosjektet de siste 3 årene, og samarbeidet har fungert meget godt. De står for finansiering, mens prosjektet fungerer som promotering av bedriften. Arbeidet med dette har i korte trekk gått ut på å promotere bedriften under demonstrasjoner og ikke minst under konkurransen i Heidelberg, skaffe og distribuere reklameeffekter fra bedriften og til slutt sende informasjon om prosjektet og oppdateringer fra arbeidet.

I midten av mai ble det gjennomført et møte med kontaktpersonene i Kongsberg med en tilhørende demonstrasjon av en nesten ferdig robot. Alle parter virket fornøyde, samtidig som det ble opprettet kontakt mellom Kongsberg og neste års deltagere. Kongsberg uttalte at de var interessert i å fortsette samarbeidet neste år.

9.6 Frakt

9.6.1 Toll

I forbindelse med frakt av roboten til Tyskland, ble det diskutert hvorvidt den overstiger verdier som man personlig kan frakte ut og inn av landet. Tidligere Eurobot-rapporter refererer til et Carnet-skjema som nødvendig for slik frakt. Ved å ta kontakt med tollvesenet i midt-Norge kom det fram at et slikt skjema ville koste flere tusen kroner inkludert utfylling, og at dette overhodet ikke var nødvendig for så små verdier som roboten representerer. Tollvesenet anbefalte imidlertid å bruke et personlig utførselsskjema for verdier inntil 5000 kr. På tollbua i Trondheim var det lite hjelp å få med dette, og man måtte oppgi serienummer på alle deler som skulle fraktes. I og med at det meste på roboten er lagd fra bunnen av, vil dette kun gjelde hovedkort og batterier/ladere av betydelig verdi. For årets reise ble det besluttet å ikke fylle ut noe skjema, uten at det skapte noen som helst problemer under reisen. Likevel kan det være smart å undersøke dette enda bedre, så man slipper en ubehagelig opplevelse på flyplassen. Under oppholdet i Tyskland, kom det også frem påstander om at det ikke er lov å frakte LIPO-batterier på flyet. Dette anbefales å undersøke nærmere med flyselskapet før neste års reise.

9.6.2 Pakking

Før reisen ble det brukt god tid på å pakke roboten skikkelig. Eurobot besitter en stor plastkasse, som har rikelig plass til en godt innpakket robot. Ved kontakt med flyselskapet (SAS/Lufthansa) ble det kjent at vekten ble regnet totalt på alle reisende og at man derfor fint kan ha med en stor kolli på over 20

kg. Det er imidlertid også slik at én kolli maksimalt kan være 32 kg for at den skal bli fraktet som vanlig last. Disse reglene bør sjekkes opp hos flyselskapet på forhånd. Uansett har roboten en egenvekt på ca. 15 kg, så det var mulig å pakke litt annet nødvendig utstyr i den store kassen. Roboten ble pakket inn i tykke lag med bobleplast på alle kanter og virket å være godt nok polstret for en røff flyreise. Annet utstyr ble fordelt på de andre reisende, som hadde fått instruks om å holde av noe plass.

9.6.3 Fraktskader

Under utpakking i Heidelberg fikk vi dessverre noen ubehagelige overraskelser. Kassen hadde tydeligvis ikke fått særlig god behandling underveis og det ble registrert en rekke mindre skader, vist i figur 9.5:

- Den ene av de to trykkbryterene som trykkes inn når roboten kjører mot en vegg, var knekt
- Topplaten hadde fått seg en skikkelig bøy
- Bakdekselet hadde fått tydelige hakk og var kraftig bøyd inn
- En del av gjengene for dekselskruene var ødelagt
- Renna for utmating av baller var bøyd kraftig ut av posisjon

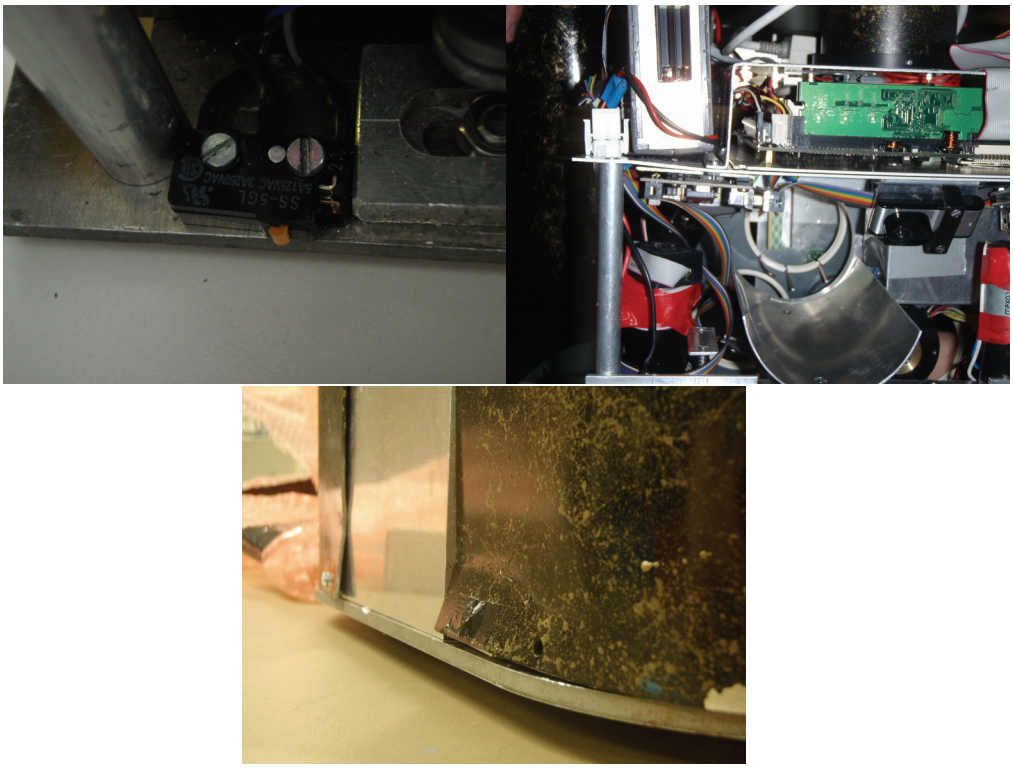
Selvom en del av roboten var blitt bøyd og forskjøvet, var det ikke altfor mye arbeid å justere det noenlunde tilbake. Noen midlertidige løsninger måtte lages, men det meste så ut til å fungere greit under senere testing. Skjevhetene skapte likevel noe bekymring siden enkelte deler i sorteringsmodulen var veldig nøye tilpasset og tilrettelagt på forhånd, for å unngå at baller skulle sette seg fast på vei inn eller ut.

9.7 Reservedeler

Helt fra starten av prosjektet har det blitt satt fokus på reservedeler. Det har vært ønskelig å ha dobbelt opp av alt, slik at hva som helst på roboten kan ryke, uten at det skal være altfor problematisk å erstatte direkte, eller finne en tilsvarende løsning på problemet. Gjennom hele semesteret har det vært nødvendig å bytte deler; alt fra små motstander og dioder, til hele kretskort og servoer. Ved å starte på dette arbeidet tidlig, har man fått rutiner på å skaffe reservedeler, slik at det ikke skulle oppstå noen overraskelser eller mangler rett før avreise.

Ved avreise til Tyskland fantes det reservedeler til alle komponenter og deler på kretskort, servoer, sikringer, kabler, plugger og ikke minst alle mekaniske

slitedeler. Det finnes et ekstra hovedkort til roboten uten nettverksstøtte, men dette ble liggende i Trondheim siden hovedkortet i roboten hadde fungert problemfritt og stabilt i lang tid. Motorer, bl.a. til fremdrift, finnes ikke ekstra, men det har aldri vært noe problem med noen av disse. Fremdriftsmotorene har kjørt uten problemer i nesten 2 år.



Figur 9.5: Skader på henholdsvis bryter, topplate og bakdeksler

Kapittel 10

Testing og resultater

Tidligere år har det blitt såpass lite tid til testing at man aldri har vært sikker på om totalsystemet fungerer. Enkeltmoduler har gjerne vært testet mye, men har blitt koblet sammen rett før konkurransen slik at samspillet aldri har blitt testet nok. I år ble det noe mer tid til testing. Totalsystemet var klart til testing ca. 1 måned før konkurransen. Fokus under alle testeprosedyrer har vært på robusthet og repetérbarhet. I utgangspunktet ønsket man å satse kun på løsninger som hadde blitt grundig testet og som viste seg å fungere riktig i flest mulig tilfeller. Som utgangspunkt for testingen måtte spillebordet fra i fjor tilpasses årets konkurranse.

10.1 Tilpassing av spillebord

For å teste roboten grundig, er det viktig å ha realistiske testforhold. Fra tidligere år finnes det et spillebord på den størrelsen som brukes i Eurobot hvert år. Det var imidlertid behov for å gjøre noen modifikasjoner slik at spillebordet passet til årets konkurranse. Dette arbeidet ble gjort i samarbeid med EiT, og var en fin anledning for å bedre kontakten og samholdet med dem. For det første måtte bordflaten males i ny farge (grå-gul). Det var viktig å lage vertikale dispensere som lignet de i reglene, da det syntes klart at disse burde utnyttes under konkurransen. Dispenserne ble skrudd sammen av trebiter og plastflasker, og det ble laget en børstelignende nedkant i henhold til reglene. Det ble laget to dispensere, som ganske enkelt kunne flyttes når man bytter lagfarge. Standardkontainer for levering er plassert langs den ene langsiden av spillebordet, og kanten her måtte dermed skjæres ned til 2 cm høyde. Selve beholderene på utsiden ble noe nedprioritert, siden disse ikke har noen annen funksjon enn å samle opp ballene. Den horisontale dispenseren i reglene var i utgangspunktet ikke planlagt å utnytte, så dermed ble ikke denne prioritert å lage. Sorteringsbeholderene fra fjorårets konkurranse, med

noen modifikasjoner, ville imidlertid fungere til dette hvis behovet skulle melde seg.

Spillbordet har vist seg svært avgjørende for resultatet av oppgaven. Det har vært utført et hundretalls tester av forskjellige slag her, og utallige justeringer har vært gjort ut fra forholdene på spillebordet. Alt fra regulatorer i navigassjonssystemet til kamera og sorteringsmodul er helt avhengig av å ha blitt testet i et realistisk miljø før det kan fungere presist og riktig. Det anbefales i høyeste grad å gjøre en skikkelig innsats på dette i framtiden. Spillebordet med dispensere er vist i figur 2.1.

10.2 Testing av totalsystem

Mye tid ble brukt gjennom semesteret på å teste enkeltmodulene i roboten. AI ble testet mot en simulator, posisjoneringssystemet ble testet separat, og EiT-modulene ble testet av EiT-gruppene selv. Da mai nærmet seg var det på tide å sammenkoble alle modulene. AI hadde ikke kunne blitt testet skikkelig før alt var på plass, og omfattende testing av totalsystemet var dermed høyst nødvendig.

10.2.1 Testresultater etter testing mot fysisk robot

Den siste måneden før konkurransen ble brukt til testing av totalsystemet på roboten. De punktene i kravspesifikasjonen til AI som ikke ble testet mot simulatoren, ble testet mot roboten slik at alle krav var oppfylt.

Opplukking av baller

Kommunikasjonen mellom AI, armer og hjul skapte mye hodebry under testingen. Timing var stikkordet, og omfattende testing har blitt utført for å få til den rette timingen. AI, armer og hjul kjører hver sin tilstandsmaskin, og dersom ikke timingen er riktig, vil ikke tilstandene til de ulike modulene samsvare. Et av de store problemene som oppstod var "jamming" av baller mellom armene og hjulet. For å minske tidsbruken, ba AI om å kjøre umiddelbart etter at ballen var opplukket istedenfor å vente på at ballen hadde blitt lagret i hjulet. Denne endringenn medførte "jamme"-problemer i tillegg til at armene ikke lystret. En løsning på dette var å innføre en timer i AI, slik at når AI ba armene om å ta inn ballen, så ventet AI et visst antall mikrosekunder før den hoppet videre til neste tilstand. I tillegg ble timeren til armen inni roboten forandret slik at den slår tidligere.

Et annet problem var at beskjeder sendt fra AI kom til armene på feil tidspunkt. Plutselig åpnet armene seg mens en ball var på vei inn. Dette ble fikset ved å legge inn ekstra kode i koden til armene, slik at armene ble tvunget til å være lukket mens en ball ble sjekket.

Det oppstod også til tider problemer med avleveringen av baller. Baller kunne falle ut av hjulet på nedsiden, og baller "jammet" seg i hjulet på vei ut. "Jammingen" var pga. timing-problemer i forhold til når armen som dytter ut ballene skulle slå ut. At ballene falt ut nede ble fikset ved å slå de hardere inn i hjulet ved lagring.

Etter at de ovennevnte problemene ble løst, har kommunikasjonen fungert veldig bra. I de fleste testrundene plukket roboten opp 5 baller og avleveringen fungerte bra. Av og til, typisk 1 av 50 ganger, "jammer" en ball seg på en eller annen måte. Dette er fortsatt et timing-problem mellom de to EiT-modulene. Det er mye mekanikk som skal fungere sammen, og det var derfor vanskelig å gjøre noe med dette.

Avstandsregulering med kollisjonssensorer

To av kollisjonssensorene ble påtenkt en rolle som avstandsregulatorer. Ved å ta i bruk kun ultralydsensorene, ble avstandsmålingene til bordkanten ganske nøyaktige. I AI ble det lagt til kode for å trigge på sensorene når de kom innenfor en gitt avstand til veggen, eller når waypointet foran dispenseren var nådd. Under testing viste deg seg at denne måten å regulere seg inn mot dispenseren på var ganske nøyaktig. Det eneste problemet var at målingene til sensorne kan hoppe litt av og til slik at det ikke alltid fungerte like bra. Litt ustabile målinger, i kombinasjon med en taktikk som ikke medførte behov for denne reguleringen, resulterte i at det ikke ble brukt under konkurransen (mest pga. robusthetshensyn).

Bruk av endebrytere

For at avleveringen av baller skulle være mest mulig robust, ble det tidlig bestemt at roboten burde være sikker på å ha kjørt i veggen før den begynner å avlevere ballene. Endebryterne ble koblet til interruptinnganger på det EiT-kortet som styrer armene. Bryterne ble først testet ved å trykke de inn manuelt og sjekke om det ble sendt CAN-meldinger ved interrupt. Til å begynne med trigget ikke bryterne alltid interrupts. Dette ble ordnet ved å legge til intern pull-up i koden. Etter å ha lagt til intern pull-up, trigget bryterne mange interrupts når de ble trykket inn. Løsningen på det var å legge på kondensatorer over bryterne.

Etter å ha sjekket at bryterne fungerte hver for seg, ble det kjørt en testrunde med roboten. Roboten stoppet fint når begge bryterne hadde trigget, så bruken av endebrytere ble en suksess, og feilet aldri.

Siden roboten står helt inntil veggen, vet man med sikkerhet y-posisjonen og headingen. På denne måten kan dette benyttes til å sette y-posisjonen og headingen i navigasjonssystemet, slik at eventuelle odometrifeil blir visket ut. Denne posisjonssettingen fungerte også veldig bra under testing.

Spenningsvarsling

Spenningsvarslerkortet ble testet ved å koble strømforsyningen rett innpå pluggen på kortet. Startspenningen ble satt til 16 V og ble gradvis justert nedover, helt til PC-en (roboten) fikk beskjed om å skru seg av. Dette fungerte veldig bra etter å ha tunet grensene for når kortet skulle trigge. Til å begynne med var det noen dårlige loddinger på kortet, slik at ad-konverteren ikke fungerte helt som den skulle.

Reset

Reset-funksjonalitet var i utgangspunktet ikke implementert i koden til EiT-modulene, men ble etter hvert implementert pga. behov. Spenningsvarslerkortet og navigasjonssystemet hadde reset-funksjonalitet hele tiden, og et enkelt trykk på reset-knappen medførte resetting av alle moduler. Dermed kunne man trykke feil team-farge, men likevel komme seg unna ved å trykke reset. Under testing har dette vist seg å være veldig robust.

CAN

I løpet av testeprosessen oppstod det en rekke uforståelige problemer. Det kom etter hvert fram at CAN-bussen var skyldig i mange av disse. Det var en rekke grunner til dette:

1. Noen CAN-meldinger sendt fra gateway ble sendt for tett, slik at enkelte meldinger ikke ble sendt ut på bussen. Man må vente et minimum av tid før neste melding kan sendes. Dette problemet ble løst på to måter; legge til en *usleep* før sending og legge en mutex på sendemetoden. På denne måten kan kun én metode kjøre CAN-send-metoden om gangen.
2. Software til de to kretskortene hadde ikke filtrering av CAN-meldingene slik at alle meldinger ble mottatt og behandlet. Ved å sette på filtrering ble disse mindre belastet.

3. Feil terminering. Enkelte kort ble terminert under debugging, og terminering ble ikke fjernet før testing av totalsystemet. Dette medførte at en rekke CAN-meldinger aldri kom fram.

Etter å ha ordnet opp i problemene over, har CAN-bussen fungert utmerket. Derfor er det veldig viktig å være klar over disse tingene ved bruk av CAN.

Posisjoneringen

Underveis i testingen ble posisjoneringstårnet brukt med stort hell. Etter hvert viste det seg imidlertid at strategien som ble kjørt, ikke krevde så mange posisjonsoppdateringer. Feilen på odometrien bygger seg opp, og blir størst etter at roboten er ferdig med de to dispenserne og etter evt. kollisjonsunngåelse på veien over bordet. Roboten skal da treffe en container som er over én meter bred, samtidig som bruk av endebrytere gjør at man har stor margin på hvor man kan treffe i y-retning. Etter levering brukes kun en sender, som beskrevet i avsnitt 3.1.3, og da er det såpass begrenset med gjenværende tid at det ikke trengs flere posisjonskorreksjoner. Bruken av det absolutte posisjoneringssystemet begrenser seg derfor til kun å bruke dette ene tårnet under testing og konkurranse. Likevel kan resten brukes i større grad i en annen konkurransesammenheng.

10.3 Konkurransen

Under testing i Trondheim kom det tidlig fram at roboten brukte ca. 1 minutt på å avlevere de 5 første ballene. Dermed hadde man 30 sekunder til rådighet etter første avlevering. Det ble derfor konkludert med at roboten ikke ville få nok tid til å hente flere baller fra dispenserne. I og med at kameratestingen hadde gitt veldig positive resultater, var det et ønske om å kunne benytte kamera til å søke etter baller, for å på denne måten kunne plukke opp baller som måtte ligge i nærheten etter avlevering. Det ble lite tid til testing av kamera før konkurransen, fordi mye av tiden gikk med til debugging av andre feil som kom til overflaten ved testing.

Video av alle kampene ligger på vedlagt CD. Undertegnede oppfordrer leseren til å se på videoen fra den første kampen, da denne på en god måte viser hvordan roboten oppfører seg ved en vanlig gjennomkjøring. Videoen finnes også på internett, se [11].

10.3.1 Homologation

Første utfordring i konkurransen var å kvalifisere seg. Kravet for å bli kvalifisert er at roboten klarer å score poeng, i tillegg til at den skal kunne unngå kollisjon. Disse to kravene blir testet hver for seg, slik at kvalifiseringen ("homologation") innebærer to runder.

Robusthet har vært et stikkord gjennom hele prosjektet, og det var derfor enighet om å videreføre dette til konkurransen også. Derfor ønsket man å gjøre ting så enkelt som mulig under kvalifiseringen, og endte opp med en veldig enkel poengscorings-strategi; plukk opp begge de første ballene og kjør deretter og levér. Dersom roboten bommet på den første ballen, ville den mest sannsynlig ikke bomme på den andre, så på denne måten var man sikker på å levere minst én ball.

Første del av "homologation" gikk ut på å få roboten fysisk godkjent. Krav som maks omkrets, høyde osv. ble sjekket. Hjørnene på fronten av bunnplaten ble ikke godkjent, da disse var alt for skarpe. Dermed måtte disse files ned og avrundes. Etter at det var gjort ble roboten godkjent, og det var klart for de to kvalifiseringsrundene.

Første runden gikk veldig bra. Roboten plukket opp de to ballene og leverte dem. Etter å ha levert de to ballene, ble roboten stående. Dette var hardkodet fordi roboten hadde nå klart å ta poeng, og det var ingen vits i å prøve å ta flere poeng.

Under kollisjonstesten kom det fram at antikollisjonssystemet trengte tuning. Denne tuningen gjaldt først og fremst grensene for når roboten skulle trigge. Roboten kjørte på motstanderroboten flere ganger, så første gjennomkjøring ble altså ikke godkjent, og grensene ble satt opp. Neste forsøk gikk bra og roboten ble godkjent. Men også denne gjennomkjøringen viste at mer tuning måtte til, nå trigget roboten alt for lett slik at den trigget kollisjoner som ikke var kollisjoner.

10.3.2 Kamp 1

Etter "homologation" ble antikollisjonssystemet tunet, slik at roboten så ut til å kjøre bra under testing. Dermed var man klar for første kamp. Strategien var å plukke opp 5 baller og levere disse. Når ballene var levert, skulle roboten bli stående. Dette for å unngå at den skulle finne på noe som ville føre til diskvalifisering eller lignende. Som sagt hadde ikke bruk av kamera blitt testet nok, så man satset heller på robusthet i den første kampen.

Første kampen gikk strålende, roboten plukket opp de ballene den skulle, og leverte dem. Motstanderroboten startet riktig nok aldri, så antikollisjonssys-

temet ble aldri satt på prøve. Det viste seg derimot at den venstre kollisjonssensoren trigget på baller som lå på bordet. Derfor kjørte roboten unnamanøvring for å unngå å kræsje med ballene. Likevel kom den seg til standardkontaineren og avleverte baller. Det at den trigget på baller viste i alle fall at systemet og unnamanøvringsalgoritmen fungerte bra.

Etter første runde lå Legend Of Norway på en oppløftende 9.plass med 17 poeng (13 for 5 sorterte baller, + 4 for seier).

10.3.3 Kamp 2

2. runde startet ca. 3 timer etter første runde, så det var lite tid til forbedringer mellom rundene. Dermed ble det kjørt samme strategi/taktikk i den 2. kampen.

Roboten startet fint og plukket opp de første 2 ballene, for så å ta en ball fra den hvite dispenseren. Da den så rygget tilbake, kjørte den seg fast og ble stående å spinne. Slik stod roboten i ca. 1 minutt, før den avbrøt og kjørte fram mot standardkontaineren. Det viste seg at en skrue hadde løsnet fra dekselet og havnet under roboten. Dermed endte man opp med 1 poeng i denne kampen (1 poeng for å komme seg ut fra starthjørnet). Etter runde 2 lå Legend Of Norway på 19. plass med 18 poeng.

10.3.4 Kamp 3

Runde 3 begynte ganske fort etter runde 2, så her var det absolutt ikke tid til forbedringer. Det ble derimot brukt tid til å passe på at alle skruer satt godt, i tillegg til at de ble teipet.

Kamp 3 utviklet seg på nøyaktig samme måte som den første kampen, og sluttresultatet ble nesten det samme. Den eneste forskjellen var at 2 av ballene kom i feil rekkefølge til slutt. En av ballene spratt oppover i renna slik at neste ball kom før denne. Dermed endte man opp med 10 poeng + 4 for seier; 14 poeng i kamp 3, og Legend Of Norway klatret opp på 16.plass med 32 poeng, 8 poeng bak 9.plassen.

Med dette var første konkurransedag (fredag) over. Hele fredagskvelden og natten gikk med til å teste ut forskjellige videre strategier. Til slutt endte man opp med ballsøkingsstrategien. Ved hjelp av kamera skulle roboten lete etter baller i et definert område etter avlevering, for så å returnere til standardkontaineren når tiden ble knapp. Dette er nærmere beskrevet i kapittel 7.

10.3.5 Kamp 4

Lørdag, siste konkurransedag, skulle bli den store dagen da Legend Of Norway skulle hente inn de tapte poengene fra kamp 2. Laget var ved godt mot, og alle var sikre på at det skulle bli en stor poengfangst. I løpet av fredagsnatten hadde det blitt lagt til mulighet for å sette ut den ene armen under avlevering, slik at ballene ikke skulle kunne sprette oppover i renna. På denne måten var man sikret 13 poeng dersom avleveringen ble gjennomført.

Ting gikk ikke akkurat som planlagt fordi roboten ikke startet når startsnoren ble dratt ut. Fortvilelsen var stor da roboten ble stående og det ble konstantert 0 poeng i fjerde kamp. Grunnen til at roboten ikke startet må ha vært at start-interruptet ikke trigget. Dette skyldes ikke software, og ei heller koblinger mellom mini-jack og interrupt-pinne. Konklusjonen ble at mini-jacken ikke kortsluttet i det vi trakk ut snoren.

Legend Of Norway falt ned på en 25.plass i stedet for den klatringen som var forventet. Etter kampen ble startsnoren og prosedyren rundt starting av programmer testet 20-30 ganger uten at det feilet. Det tunisiske laget kunne informere om at mini-jack av og til ikke kortslutter, typisk 1 av 100 ganger.

10.3.6 Kamp 5

Nå så ting mørkt ut med tanke på åttendedelsfinale, men det var snakk om å klatre så langt som mulig, her skulle det sankes minst 13 poeng.

Optimismen var stor da roboten startet og plukket opp de 2 første ballene, men så viste det seg at ballene hadde "jammet" inne i sorteringsmodulen, slik at det ble 0 poeng også i denne kampen. Årsaken til "jammingen" denne gangen må ha vært at armen som dytter inn ballene, har presset den første ballen mot veggen av renna. På denne måten ble den første ballen liggende i renna, og da det kom enda en ball inn, ble det fullstendig stopp. Dette problemet hadde blitt observert under testing, men svært sjelden, noe som gjorde det vanskelig å debugge.

Dermed, etter 5. og siste kamp, endte Legend Of Norway på 31.plass med 32 poeng (se vedlegg C). Dersom 2 av de kampene som feilet hadde gått bra, hadde situasjonen blitt helt annerledes, da hadde Legend Of Norway endt opp på en 15.plass og vært klar for åttendedelsfinale. Hadde alle kampene gått bra, hadde topp 10 vært et faktum.

10.4 Oppsummering

Resultatene fra testing og konkurranse er gjennomgående positive og forholdsvis like. De feilene som oppstod under konkurransen er feil som kan oppstå, og som er vanskelig å gardere seg mot. Erfaringene fra konkurransen tilsier at alle skruer bør dobbelsjekkes før en kamp slik at man ikke risikerer at de faller av. Når det gjelder startsnor er det tydelig at mini-jack-løsningen kanskje er litt for usikker, så dette bør tenkes gjennom. Mekanikk og bevegelige deler vil alltid kunne feile og er følgelig vanskelig å gjøre noe med. Enkle og robuste løsninger anbefales.

Kapittel 11

Konklusjon

Hovedmålet med denne masteroppgaven har vært å ferdigstille roboten til Eurobot Open 2008. Gjennom et godt samarbeid med EiT-studentene har det blitt utviklet en modulbasert og robust robot.

Det har blitt laget et nytt system for absolutt posisjonering basert på triangulering med 3 faste sendere. Systemet er basert på et tidligere utviklet konsept, men er i stor grad forbedret innenfor både hardware og software. Tester av systemet viste gode resultater, men målingene tok noe lenger tid enn opprinnelig planlagt. Dermed kreves det at roboten står stille under posisjoneringen. Siden odometrien fungerer bra har ikke dette vært noe stort problem, og det finnes flere anledningner til å foreta absolutte posisjonsmålinger underveis i gjennomføringen av konkurransen.

Navigasjonssystemet har stort sett blitt videreført fra tidligere, men det er utviklet mye ny funksjonalitet; støtte for rygging, nye typer waypoints og regulering mot kant. Det utvidede systemet gir mye større fleksibilitet i manøvreringen, samtidig som grunnstrukturen er enkel og oversiktlig. De nye mulighetene har blitt brukt flittig i forbindelse med gjennomføring av konkurransen og har vist seg å fungere veldig bra.

Den kunstige intelligensen bygger på ideene som ble presentert i prosjektoppgaven og baserer seg på en mengde tilgjengelige strategier med dynamiske prioriteter. AI er bygd opp på en strukturert og enkel måte, noe som har gjort debugging lett. En enkel struktur medfører bedre kontroll på hva roboten foretar seg. På grunn av strategienes like oppbygning kan strategier lett fjernes eller legges til. Bruk av simulator gjorde debuggingen av AI enkel og effektiv. Player-Stage er et ferdigutviklet bibliotek for simulering av roboter og ble brukt med hell i årets oppgave. Det anbefales å bruke både simulator og AI videre i neste års konkurranse. Selve rammeverket til AI er såpass bra og nøye gjennomgått at det bør brukes videre. Ved å fjerne årets strategier og legge til

nye har man en AI tilpasset den nye konkurransen.

Årets antikollisjonssystem bygget på fjorårets, men med en god del forbedringer og endringer. Bruken av Ir ble forkastet pga. følsomheten i forhold til forstyrrelser. Sensorkonfigurasjonen var ellers den samme. Ultralydssensorne ga ganske stabile målinger ved korte avstander, men enkelte avvik medførte innføring av buffer på målingene. I tillegg har deaktivering av antikollisjon og en unnamanøvringsalgoritme gjort systemet mer robust.

Ved hjelp av det ferdige biblioteket OpenCv og Hough-transformen har bildegjenkjenningen vært robust. Kamera finner ballene som ligger foran roboten og gjenkjent farge er i de aller fleste tilfeller riktig, bortsett fra de gangene røde og blå baller blir tolket som hvite. Ved hjelp av kamera ble en "søk-etter-ball"-strategi implementert og utprøvd; roboten kjørte med redusert hastighet og kamera klarte å finne baller mens roboten var i bevegelse.

Fokus fra starten av prosjektet har vært robusthet og repetérbarhet og gjennom omfattende testing har totalsystemet vist seg å være veldig robust. Selv om roboten fungerte bra under testing, gikk ikke alt som forventet under konkurransen. Roboten vant 2/5 kamper, der den feilet i 3 av kampene pga. henholdsvis en løs skrue som falt av dekselet, mini-jack som ikke kortsluttet og "jamming" av baller i sorteringsmodulen. Til slutt endte Legend of Norway på 31. plass av 39. Undertegnede er fornøyd med gjennomføringen og mener at resultatet gir et feil bilde av robustheten til systemet, da de feilene som oppstod var tilfeldige og uheldige. For første gang på mange år har NTNU-laget vunnet kamper i Eurobot, bl.a. som resultat av fokus på gjenbruk og robusthet. Ved å gå videre med det arbeidet som er lagt ned, og ha noe mer hell, bør det være mulig å hevde seg langt bedre i årene framover.

Kapittel 12

Videre arbeid

12.1 Posisjonering

Posisjonering av roboten er en av de viktigste faktorene for å oppnå suksess i Eurobot-konkurransen. Systemet som brukes nå, fungerer tilfredsstillende, men det anbefales å utvikle den absolutte posisjoneringen videre. Grunnlaget burde være tilstrekkelig, men målingen av vinkler bør helst bli enda mer presis og ikke minst raskere. Ellers kan grunnoppsettet antagelig brukes i stor grad slik som det er nå.

De siste justeringene på odometrien har gjort denne mer enn bra nok for kortere avstander og skal fungere ypperlig i samarbeid med et raskere absolutt posisjoneringssystem. Likevel bør man være oppmerksom på de feilene og justeringene som er beskrevet i denne oppgaven, slik at man unngår dette på nytt.

12.2 Navigasjonssystemet

Rammeverket for navigassjonssystemet har vært brukt i 2 år, og bør ikke være nødvendig å endre i altfor stor grad. Man bør i størst mulig grad utnytte de funksjonene som finnes, og evt. fortsette å legge til funksjonalitet som beskrevet i denne oppgaven. Potensialet er imidlertid stort mhp. å øke hastigheten på navigeringen. Forutsatt gode antikollisjonssensorer og posisjonering, kan regulatorene endres og tunes til å håndtere langt større hastigheter, samtidig som hardwaren allerede har et stort potensiale.

12.3 Kunstig intelligens

AI har blitt utviklet for å være så generell som mulig med tanke på videre bruk. Rammeverket er laget slik at man enkelt kan fjerne de strategiene som eksisterer og legge til nye. Den overordnede styringen og timeren fungerer på samme måte uansett hvilke strategier som er i bruk. Erfaringene fra testing og konkurranse tilsier at AI fungerer veldig bra. I tillegg er strukturen veldig klar, så det skal være greit å sette seg inn i virkemåten. Det anbefales derfor å benytte årets AI i de kommende års konkurranser.

12.4 Antikollisjon

Hardware har forbedringspotensiale da sensorne er noe ustabile. Derfor anbefales det å bytte ut all hardware. Systemet kan brukes videre, men da må man være klar over de begrensninger som råder. Software og taktikk ser ut til å fungere bra, og kan i aller høyeste grad benyttes videre. Det kan derimot være fordelaktig med tracking av motstanderroboten, da dette gir et enda bedre utgangspunkt for planlegging av kjørerute før kollisjoner forekommer.

12.5 Datasyn

Kamera har fungert bra og det er mye å hente på bruk av det. OpenCv-biblioteket anbefales å bruke videre, både fordi dette er mye brukt og fordi det har god funksjonalitet. Det finnes flere bruksområder for kamera enn det som er presentert i denne rapporten, blant annet tracking av motstanderroboten. Ved å bruke 2 kameraer, kan ett brukes til oppgaven, og ett til tracking. Kode for bruk av kamera kan gjerne baseres på årets kode, da kommunikasjonen med AI er etablert i denne koden. Dersom man skulle få tilgang på arbeidskapasitet og ekspertise kan kamera også brukes i større grad til gjenkjenning av andre objekter og områder for bruk i f.eks. posisjonering.

12.6 Organisering og arbeidskapasitet

Å delta i Eurobot krever veldig mye arbeid. Først og fremst skal det gjøres utrolig mye praktisk gjennom bygging og testing av roboten, samtidig som man er nødt til å passe på alt som skal ordnes i forbindelse med reising og deltagelse i selve konkurransen. Med to ansvarlige personer er det vanskelig å gjøre et skikkelig arbeid på å utvikle den oppgaven man er satt til. Veldig stor del av tiden går med til å organisere konkurransen; påmeldingsskjemaer, store

Eurobot-undersøkelsesskjemaer, design/utforming av A1-plakat og trykking av denne, bestilling/organisering av reising for alle som skal være med, pakking osv. I tillegg kommer felles praktisk arbeid med roboten som innkjøp av deler, sammensetting, kobling, kommunikasjon og testing/utvikling av fellesmoduler (kamera, CAN-buss, strømforsyning m.m.). Til slutt kan det nevnes en del generell organisering av prosjektet som omfatter veiledning/oppfølging av EiT, økonomistyring og mindre demonstrasjoner. Alt dette arbeidet er felles og kommer i tillegg til de hovedområdene man har forsøkt å forbedre/utvikle: Posisjonering og kunstig intelligens. For å få satt enda større fokus på forbedring av enkeltmoduler, som er ment til å brukes gjennom flere år, bør man vurdere å utvide teamet. Dette vil samtidig gi mindre fellesarbeid på hver enkelt og dermed større fokus på selve oppgaven. Generelle moduler som brukes hvert år og som har store potensiale forbedringsområder er:

- Bedre absolutt posisjonering vha. triangulering, gjerne med posisjonering av motstanderen
- Mer robust og raskere kollisjonsdeteksjon
- Større bruk av kamera for å detektere objekter og kjente posisjoner

Her utkrystaliserer det seg 3 ganske klare oppgaver. Oppgaven med kamera er ikke spesielt kybernetikk-relevant og det bør derfor vurderes å trekke inn en person med annen fagbakgrunn, f.eks. fra institutt for datateknikk og informasjonsvitenskap.

12.6.1 Tilrettelegging for ekstern hjelp

Prosjektet har hatt stor nytte av hjelp fra utenforstående. Først og fremst fra EiT-gruppene, men også fra mekanisk verksted og komponentlager. Denne hjelpen er man veldig avhengig av, og hvor heldige man er med arbeidskapasiteten og tilgjengeligheten på disse er veldig avgjørende for det totale resultatet. I år har dette stort sett fungert fint, men det hadde vært en stor fordel om instituttet kunne bidratt med å organisere dedikert hjelp til prosjektet. Eurobot bør ha høyere prioritet hvis man ønsker å få en utvikling. Med all den eksterne pengestøtten og innsatsen som legges ned, er dette en god mulighet til å få frem et flaggprosjekt som i langt større grad kan brukes til promotering av kybernetikkstudiet. Et forslag er derfor at det settes av ressurser i form av dedikert tid eller egne ansvarlige personer, slik at man ikke er avhengige av "goodwill" fra gang til gang.

12.7 Testing i konkurransesammenheng

Det har vært gjort meget omfattende testing av roboten, både av enkeltdeler og det ferdige produktet, og dette har vist seg å være veldig viktig og avgjørende. Likevel har man litt følelsen av å løse oppgaven uten noen som helst input fra omverdenen. Det er et spenningsmoment å se de ordentlige konkurransebordene og å oppdage hvordan andre roboter oppfører seg i konkurransesammenheng. For de landene som har nasjonal kvalifisering er dette en stor fordel, da de får prøvd seg i konkurranse på et tidligere tidspunkt. Under oppholdet i Heidelberg kom det frem at flere av disse landene åpner for at lag fra mindre land kan være med, uten at de har noen innvirkning på de nasjonale resultatene. Muligheten for å delta i en slik kvalifisering, uten å ha noe å tape, er absolutt noe som burde undersøkes. Der kan man teste systemet og få gode erfaringer og korreksjoner til den ordentlige konkurransen.

Bibliografi

[1] Kjemphol og Knausgård. Prosjektoppgave: Eurobot 2007. Technical report, 2006.

[2] gruppe 1 EiT. Fagrapport eurobot 2008. Technical report, 2008.

[3] gruppe 2 EiT. Fagrapport eurobot 2008. Technical report, 2008.

[4] Mørkrid og Øien. Prosjektoppgave: Eurobot 2008. Technical report, 2007.

[5] Carvalho Esteves and Couto. Generalized geometric triangulation algorithm for mobile robot absoulete self-localization. Technical report, IEEE, 2003.

[6] Kristian Muri Knausgård. Eurobot 2007 navigasjon. Master's thesis, Institutt for Teknisk Kybernetikk, Norges Teknisk-Naturvitenskapelige Universitet, 2007.

[7] Garsjø og Platou. Eurobot 2006. Master's thesis, Institutt for Teknisk Kybernetikk, Norges Teknisk-Naturvitenskapelige Universitet, 2006.

[8] Player project. http://playerstage.sourceforge.net/wiki/main_page.

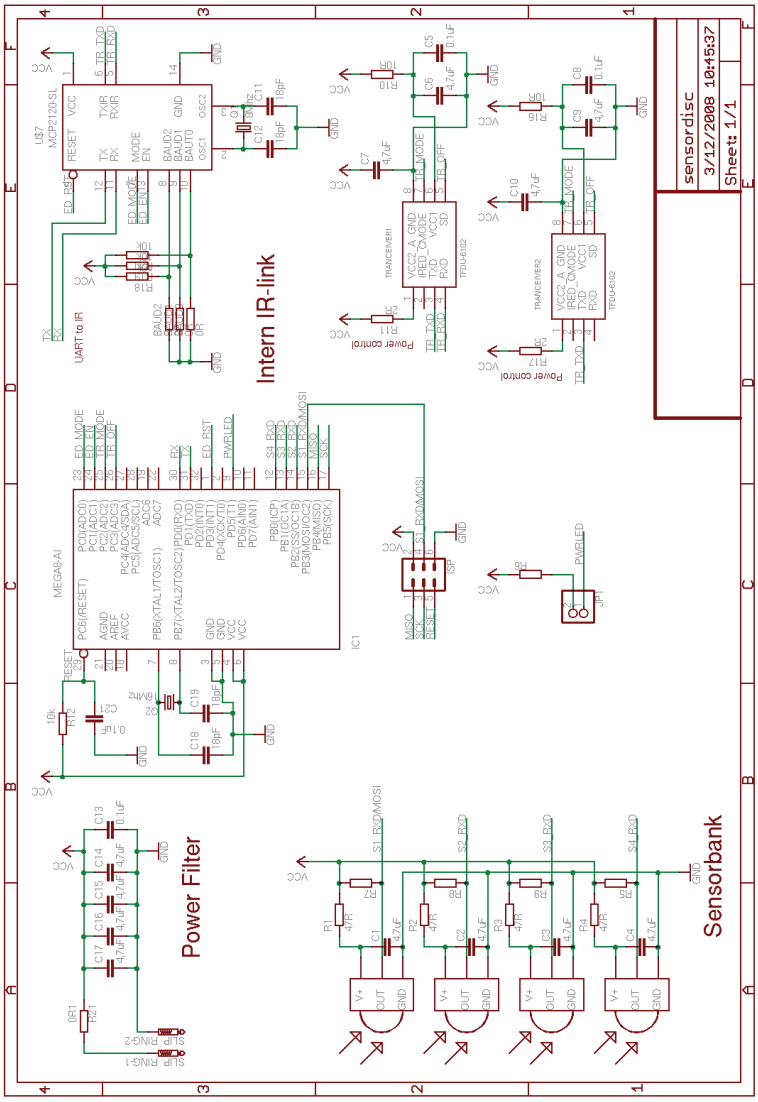
[9] Gunnar Kjemphol. Eurobot 2007. Master's thesis, Institutt for Teknisk Kybernetikk, Norges Teknisk-Naturvitenskapelige Universitet, 2007.

[10] Jean-Yves Bouguet. http://www.vision.caltech.edu/bouguetj/calib_doc/.

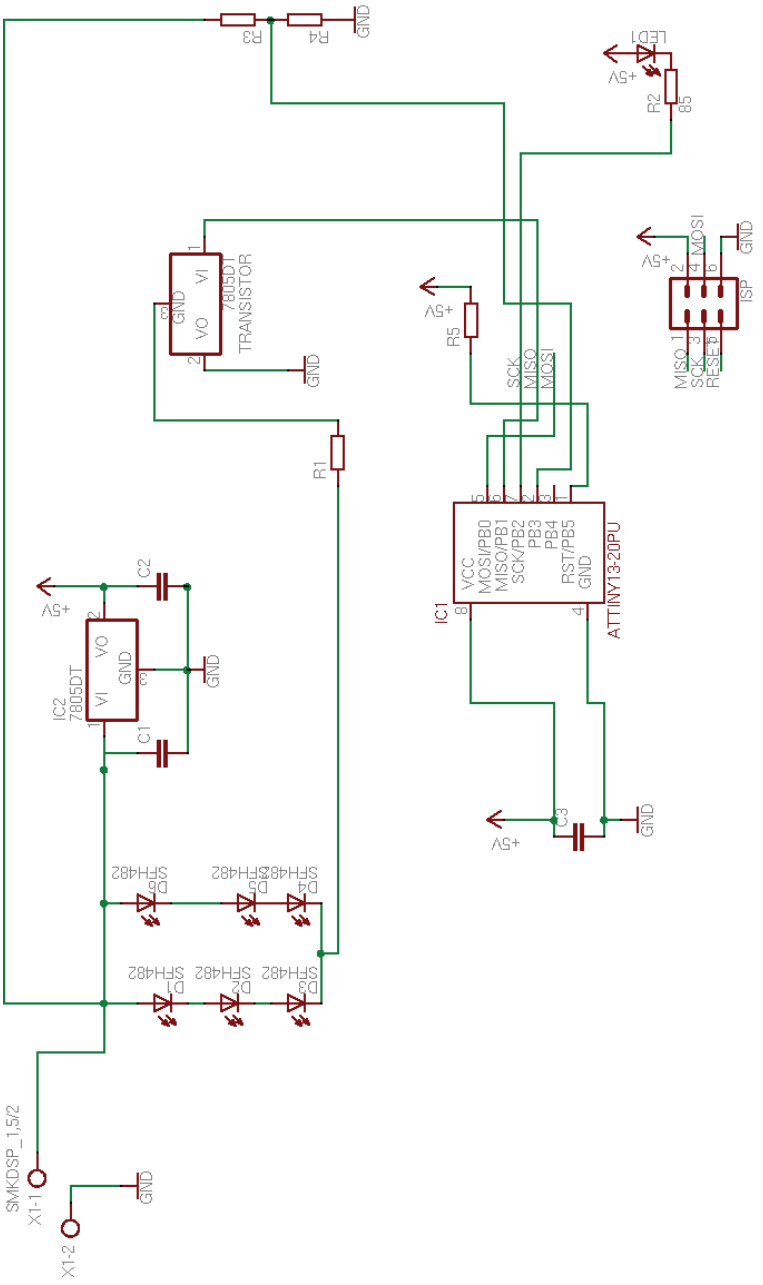
[11] Legend of Norway. http://www.youtube.com/watch?v=dzcd14vuvsc.

Tillegg A

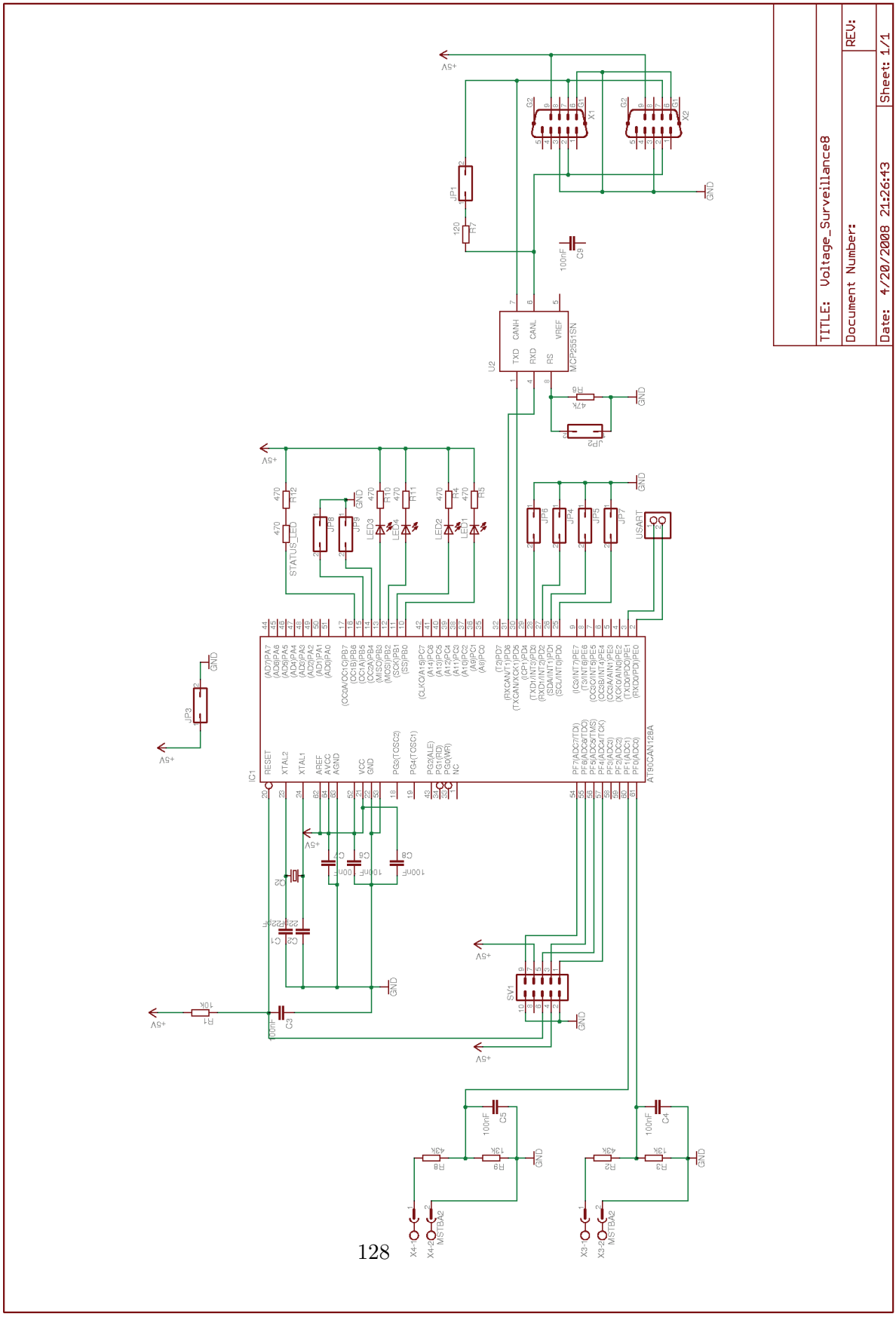
Kretskort



Figur A.1: Sensorkort



Figur A.2: Posisjoneringssender



Figur A.3: Spenningsvarslerkortet

Tillegg B

Player-Stage

simple.cfg:

```
# Desc: Player sample configuration file for controlling Stage devices
# Author:  Richard Vaughan
# Date: 1 December 2004
# CVS: $Id: simple.cfg,v 1.34 2007/11/02 01:11:39 gerkey Exp $

# load the Stage plugin simulation driver
driver
(
  name "stage"
  provides ["simulation:0" ]
  plugin "libstageplugin"

  # load the named file into the simulator
  worldfile "simple.world"
)

# Export the map
driver
(
  name "stage"
  provides ["map:0" ]
  model "cave"
)

# Create a Stage driver and attach position2d and laser interfaces
# to the model "robot1"
driver
(
  name "stage"
  provides ["odometry:::position2d:0" "laser:0" "sonar:0" ]
  model "robot1"
)

# Demonstrates use of a Player "abstract driver": one that doesn't
# interface directly with hardware, but only with other Player devices.
# The VFH driver attempts to drive to commanded positions without
# bumping into obstacles.
driver
(
  name "vfh"
  provides ["position2d:1"]
  requires ["position2d:0" "laser:0" ]
```

```
)
```

simple.world:

```
# Desc: 1 pioneer robot with laser
# CVS: $Id: simple.world,v 1.67 2006/10/05 22:27:29 gerkey Exp $

# defines Pioneer-like robots
include "pioneer.inc"

# defines 'map' object used for floorplans
include "map.inc"

# defines sick laser scanner
include "sick.inc"

# size of the world in meters
size [20 20]

# set the resolution of the underlying raytrace model in meters
resolution 0.02

interval_sim 100
interval_real 100

# configure the GUI window
window
(
  size [ 695.000 693.000 ]
  center [-0.010 -0.040]
  scale 0.028
)

# load an environment bitmap
map
(
  bitmap "bitmaps/table.png"
  size [20 20]
  name "cave"
)

# create a robot
pioneer2dx
(
  name "robot1"
```

```
  color "red"
  pose [0 0 0]
  sick_laser()
  watchdog_timeout -1.0
)

pioneer2dx
(
  name "robot2"
  color "blue"
  pose [8 0 1.57]
  sick_laser()
  watchdog_timeout -1.0
)
```

Tillegg C

Resultatliste

Plass	Team	Poeng
1	rcva	203
2	Helb - inraci	184
3	ISTIA - IUT d'Angers	137
4	μ	111
5	Lily	101
6	RoboRacingTeam	90
7	STARTUP	88
8	Yzro	79
9	DIIT TEAM	77
10	Raging ball	72
11	Viper	71
12	Deimos	71
13	LSI-UC3M	70
14	ROBOCES CORCHOPAN	69
15	León Robots	62
16	Russian Engineering Team	61
17	YUNIMIN	58
18	Ubermaschin	56
19	Robosib	54
20	Germ Warfare	53
21	eu_MUST	52
22	TURAG	51
23	Gyurgyalag	43
24	Uni Heidelberg	42
24	Monastir	42
24	MarsRiders	42
27	Montefiore Team	41
27	Mons Polytech Team 2008	41
29	FELBOT	36
30	Roboterclub Aachen e.V.	35
31	**Legend of Norway**	**32**
32	ArgonautE	31
33	1966	30
34	RallyRobot	17
35	beaRobot	16
36	TRANSROB	14
37	RoBUTE	8
37	Universiti Teknologi Petronas	8
39	AQRA/CGL	5

Tillegg D

CD-ROM

- Kretskort
  - Posisjonering
  - Spenningsvarsler
- Mikrokontrollerkildekode
- Robot:
  - cvsys - Datasyn-kode
  - gateway - gateway-kode
  - include - Felles header-filer
  - navsys - Navigasjonskode
  - plansys - AI-kode
  - RobotFirstMatch3 - Kode som ble brukt fra og med kamp 3 i konkurransen
- Simuleringer
  - PlayerStageSimulator
  - Posisjonering
- Videoer - Kamp 1 til 5