Polyteknisk midtvejsprojekt

Software til automatiseret bordfodbold

Af:
Anders Chr. Myrup (s032097)
Martin Ørding-Thomsen (s032108)

Vejledere:
Nils Andersen
Ole Ravn

5. februar 2007

Danmarks Tekniske Universitet
Ørsted•DTU, Institut for Automation
Building 326, DK-2800 Kongens Lyngby

Resumé

Baseret på eksisterende hardware til en bordfodboldrobot har vi udviklet software, som gør robotten i stand til at spille bordfodbold. Den har i sig selv en begrænset rolle som underholdning, men mange af de discipliner der skal beherskes for at få den til at fungere, bliver brugt i andre sammenhænge. F.eks. kan vision og motorstyring benyttes i forbindelse med industriel produktion.

Vi har konstrueret en AI på baggrund af, hvordan vi selv spiller bordfodbold. Denne kan vha. et kamera og billedgenkendelse bestemme boldens position på trods af, at kameraet ryster flere cm til hver side. En computer kommunikerer med motor-controllere, som ved brug af billige motorer kan styre stængerne på den ene side af et bordfodboldbord. Robotten har vundet 70% af dens kampe og dens spilleniveau vurderes på en skala mellem 0 og 10 som 7. Det er lykkedes at få robotten til at spille på højt niveau på trods af lave omkostninger til hardware.

Indhold

Kapitel 1

Indledning

1.1 Generelt om bordfodbold

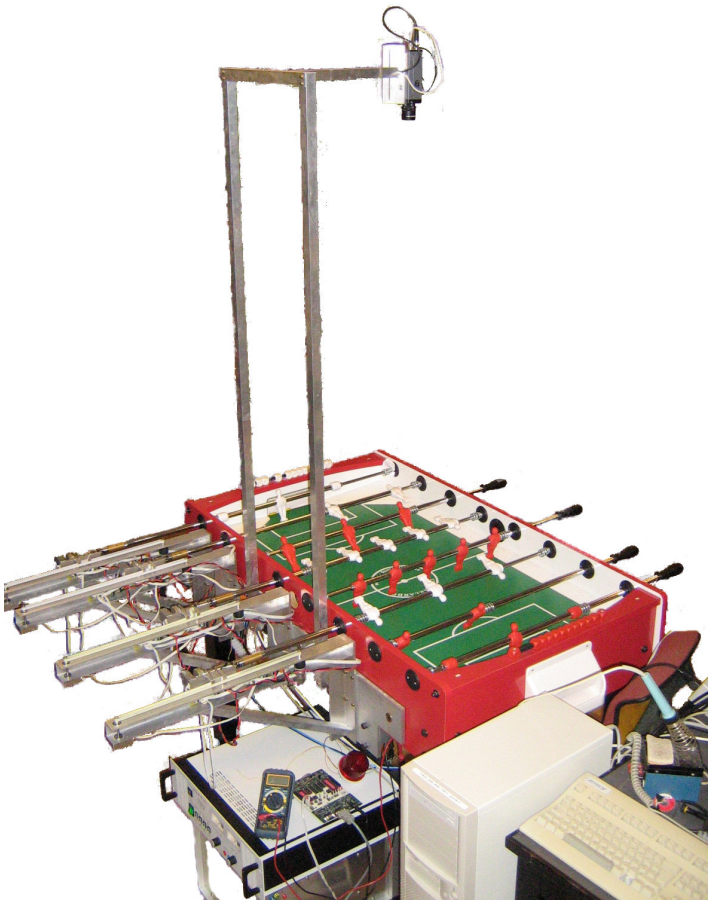
Bordfodbold er et spil, som er inspireret af fodbold. Målet med spillet er således at få en bold til at gå flere gange i modstanderes mål end i ens eget mål. Det spilles på et bord på ca. 1 m gange 70 cm og med en ca 3 cm stor bold. Det kan se ud som på figur 1.1.

Figur 1.1: Et bordfodboldbord.

Spillerne er små og lavet af plastik. De styres ved at skubbe og hive i de stænger, de er sat fast på. Stængerne kan også drejes, hvorved spilleren skyder til bolden. På hver side af bordet er der typisk en eller to personer og disse styrer et sæt spillere hver og spiller på den måde mod hinanden.
Når der scores tages bolden ud af målet og den starter på midten igen. Der spilles oftest til et fast antal point er opnået for en side, f.eks. 10.

1.2 Mål med projektet

Vi skal programmere en robot, som kan styre spillerne på den ene side af bordet, sådan at man kan spille bordfodbold mod robotten. Vi har overtaget en robot fra tidligere projekter, men den eksisterende software gør den ikke i stand til at spille. Selve projektet består således mest af programmering, men også af mindre ændringer af hardwaren. På figur 1.2 ses robotten. Den har motorer til styring af stængerne og et kamera hængt op med et stativ for oven.



Figur 1.2: Bordfodboldrobotten

Vores successkriterie for projektet er at robotten kan vinde over en 7-årig, og spille kampen til ende uden afbrydelser. Det er rimeligt at have så lave ambitioner, da robotten har sværere ved at se bolden end os og fordi det er meget svært at få robotten til at reagere fornuftigt i alle situationer.

1.3 Relevans

Projektet er relevant fordi det bl.a. omfatter følgende emner:

- Samspil mellem hardware og software i forb. med styring af DC motorer.
- Billedgenkendelse - lokalisering af et objekt, kompensation for støj, rystelser og fremmedlegemer og brug af fikspunkter.
- Kunstig intelligens - valg af intelligenstype og strukturering af denne.

Disse emner bliver også brugt i mange andre sammenhænge. Lokalisering af objekter vha. billedgenkendelse og styring af motorer på baggrund af lokaliseringen, kan f.eks. bruges i forbindelse med produktion på transportbånd i industrien.

Vi har en del domæneviden omkring bordfodbold i kraft af flere års erfaring. Konkret har det ført til en placering på plads 33-48 ved DM i 2004 [5].

1.4 Afgrænsning

Da vi er informatikere uden stor ekspertise indenfor elektronik og mekanik, fokuserer projektet på softwaren og vi foretager ikke store ændringer af hardwaren.

Der findes forskellige typer bordfodbold med varierende materialer, spillerkonfigurationer og regler. Inden for de grundlæggende typer af borde findes endvidere forskellige dimensioneringer af enkeltdelene. Vores robot skal kun kunne spille på et bestemt bord.

Desuden findes der et stort regelsæt [14], hvoraf vi ignorerer langt de fleste regler. Dette skyldes at mange af reglerne er irrelevante i forhold til vores projekt (såsom §1.4.10 om straf til en spiller, hvis hans mobiltelefon ringer under spil).

Kapitel 2

Eksisterende hardware og software

Da vores projekt bygger videre på tidligere projekter, introduceres eksisterende hardware og software.

2.1 Hardware

Vi har overtaget et bordfodboldbord med hardware der kan bevæge hele den ene side af bordet.

Over bordet er et kamera monteret, der kan benyttes til at detektere spillets forløb.

Kameraet og hardwaren, der styrer bevægelsen, er sluttet til en computer, hvorpå et hoved-program kæder kamera-input og bevægelsen sammen.

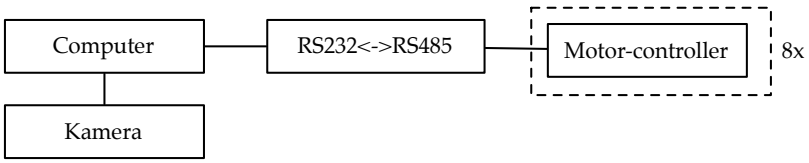
En detaljeret specifikation af hardwaren er at finde i bilag A.

2.1.1 Bevægelse

Der er fire stænger på hver side, og de kan roteres (ved spark) samt bevæges sidelæns. Sidelæns bevægelse kan kun foregå indenfor et fast interval, afgjort af hvilken stang der bevæges. Af de fire stænger (målmand, forsvar, midtbane og angreb) kan eksempelvis forsvar bevæges ca. 30 cm længere sidelæns end midtbanen.

Da hver stang har to frihedsgrader, er der i alt 8 motorer. Til at styre den enkelte motor, har de hver en motor-controller, som er forbundet med computeren via en converter mellem de

to serielstandarder RS485 og RS232.



Figur 2.1: Hardware tilslutning

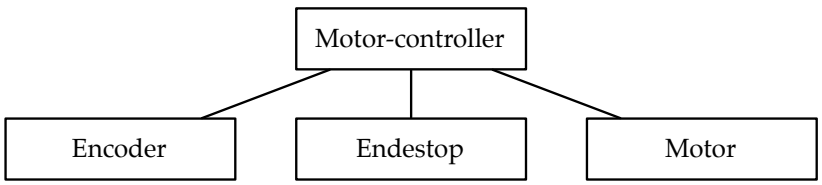
Motor-controllerne er oprindeligt udviklet til SMR, der er en lille multi-funktions robot på hjul, udviklet på Ørsted•DTUs institut for automation. Motor-controllerne regulerer motorsspænding vha. pulsbreddemodulation[8].

2.1.1.1 Encoders

Til at finde positionen for hver akse er encoders med en opløsning på 2000 ticks sluttet til motorcontrollerne[17]. Disse positioner skal nulpunktskalibreres (eller *resettes*) i motorcontrollerne, og dette opnås på forskellige måder for de to akser, da bevægeligheden for den ene som skrevet er afgrænset. På encoders til rotationsbevægelsen er der indekspuls, som er høj når mændene står lodret.

2.1.1.2 Endestop

Hvis afgrænsningen af den lineære bevægelse ikke overholdes, vil stængerne køre ind i siderne på bordet. For at kunne stoppe motorerne når det sker, er der monteret kontakter i yderpunkter for bevægelsen, som er forbundet til motor-controllers.



Figur 2.2: Motor-controlleren

2.1.1.3 Motor

Robotten har DC motorer til styring af stængerne.

I bordfodbold er det, som det gælder for mange sportsgrene, en fordel at kunne bevæge sig hurtigt. Det er også en fordel at skyde hårdt. Gennemsnitshastigheden på vores hårdeste

skud er målt til ca. 7.4 m/s, hvor robottens er målt til 6.6 m/s. Dette konkluderede vi var tilstrækkeligt hurtigt[3].

Der er ikke foretaget hastighedsmålinger af sidelænsbevægelsen, selvom hastigheden af denne type bevægelse også spiller en stor rolle. Det blev konkluderet i [3], at den samlede hastighed var tilfredsstillende.

2.1.2 Kamera

Kameraet er fastgjort den ene side af bordet på et ca. 1.2 m højt stativ, jf. figur 1.2. Det er forbundet til et frame grabber kort i computeren, hvorved vi kan indlæse billeder 25 gange i sekundet (FPS). Da disse billeder er interlaced kan vi i praksis få 50 FPS ved at benytte både det billede der fås når kameraet scanner opad og når det scanner nedad.

2.1.3 Computer

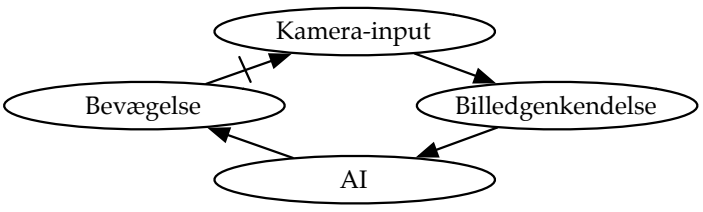
Computeren er en 450 MHz Pentium III med 128 MB RAM, der kører en realtime Linux distribution. Trods den høje alder har denne maskine ikke udgjort en flaskehals under vores forrige projekter.

2.2 Software

2.2.1 Hovedprogram

Igennem [3] og [4] har vi lavet et framework, med henblik på senere videreudvikling af robotten. Dette er skrevet i C++ og er først og fremmest en wrapper for kommunikationen med motor-controllere og input fra kameraet. Inputtet bliver til dels behandlet, idet en primitiv boldgenkendelsesalgoritme er implementeret.

Hovedstrukturen af programmet er en løkke, hvor vi kalder hver iteration for en *frame*.



Figur 2.3: Løkken i hovedprogrammet

Der er først et kald til grabber kortet, der giver et billede fra kameraet. Dette kald er breaking, dvs. programmet venter indtil et billede er klar fra kameraet. Dernæst køres billedgenkendelsen, og til sidst AI, som flytter stængerne (se figur 2.3).

Der er ikke implementeret nogen form for AI i det eksisterende system.

2.2.2 Brugerinterface

Vi har i [3] skrevet en grafisk brugerflade til windows, der via en socket forbindelse med computeren i robotten kan vise positionen hvor bolden er genkendt og hvor encoderne har aflæst at robottens stænger befinder sig under spil. Dette egner sig især til debugging under udvikling af billedgenkendelseskomponenter.

2.2.3 Motor-controller

Software til motor-controllerne er skrevet i C. Den har en kommunikationsdel, der kan modtage ønskede positioner fra computeren, og sende den position, softwaren har registreret vha. encoderinput, tilbage. Selve reguleringen af bevægelsen foregår hvert 2. ms på motorcontrolleren [4].

Nulpunktskalibreringen på rotationsaksen fungerer ved at stangen drejes, indtil indexpulsen er høj, men sidelænsbevægelsen nulpunktskalibreres i softwaren ved brug af endestop.

Kapitel 3

Motorstyring

Der er en del problemer med den eksisterende motorstyring, der bør udbedres, før den er moden til at blive anvendt i en større sammenhæng. En del af disse problemer er påpeget i forrige projekter, og andre er opdaget under udvikling af dette projekt. Følgende mangler vil blive håndteret:

- Motorerne bliver varme, selvom de ikke er i spil.
- Motorerne bliver varme under spil.
- Rotationsbevægelsen kan blokeres, hvis bolden sidder fast mellem en fod og gulv.
- Størstedelen af endestoppene sidder ikke ordentligt på.
- Fejlregistrering af impulser fra encoder.

3.1 Varme udenfor spil

Under udarbejdelse af dette projekt, fandt vi det bemærkelsesværdigt, at motorerne bliver varme, selvom de ikke har været i spil.

Analyse Varmeudviklingen tyder på at motor-controlleren afsætter effekt i motoren selvom stangen ikke ønskes bevæget, dvs. at stangen enten er udenfor spil eller er ved den ønskede position under spil.

I hidtidige implementeringer af motor-controllersoftwaren, sættes pulsbredden til en halv periode udenfor spil, dvs. spændingen er positiv i lige så lang tid som den er negativ.

Dette er ikke tilstrækkeligt, da der afsættes effekt i motoren så længe H-brocontrolleren[1] på motor-controllerboardet er aktiveret.

Design/Implementering Hvis pulsbredden beregnet af reguleringen er så lav at stangen ikke vil kunne igangsætte bevægelse, slås H-bro controlleren helt fra, dermed sikres at der ikke afsættes effekt i motoren. Grænserne for hvilke pulsbredder der kan igangsætte bevægelse er fundet i bilag B.1.1.

Test Målinger med oscilloskop (se bilag B.1.2) viser at der ikke længere sendes spænding ud til motoren, hvis den er ved sin ønskede position.

3.2 Varme under spil

I [3] nævnes det at motorerne godt kan blive meget varme under spil, da de bliver kørt hårdt, hvilket muligvis senere kan blive et problem. En del af varmen, som udvikles under spil er fjernet i forrige afsnit. Den kraftige varmeudvikling under spil, skyldes primært at de målte skudhastigheder i [3] er opnået ved at sende maksimalt 12 V ud til motorerne selvom motor-typen kun er konstrueret til en maksimalbelastning på 7.2 V. Denne mishandling af motorerne er bibeholdt af følgende årsager:

- Ved 7.2 V opnås skudhastigheder på 0.5 m/s, hvilket er utilfredstillende[3].
- Motorerne er designet til hobbybrug, og er derfor meget billige.
- Man kan undgå at motorerne bliver for varme ved at give dem pauser ind i mellem. Først efter 30 minutters spil bliver motorerne så varme at man ikke kan røre ved dem. Spilletid på 30 min svarer til 2-3 spil. Altså mere end et enkelt spil, som vi stiller som minimumskravet.

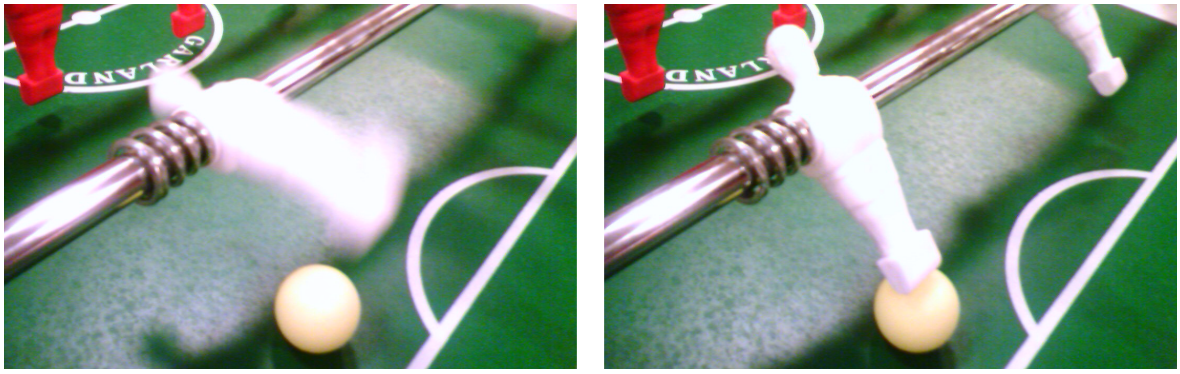
Man kan argumentere for, at en tilfredsstillende skudhastighed også vil kunne fås ved en spænding mellem 7.2 og 12 V, som dermed ville være mere skånsomt for motorerne, men vi har valgt at vægte hastighed over kontinuerlig drift. Mange testkampe (bilag F) har vist at minimumskravet på et enkelt spil er opfyldt.

Motorerne bør skiftes ud med nogle kraftigere motorer, før bordet kan anvendes til kontinuerlig drift.

[1] H-brocontrolleren er af typen HIP-4081AIP

3.3 Blokering af rotationsbevægelsen

Hvis bolden ruller ind under en stang imens denne roterer, kan bolden sætte sig fast imellem fod og gulv, som det ses på figur 3.1.



Figur 3.1: Til venstre: Bold ligger stille og stangen drejes så foden nærmer sig gulvet. Til højre: Stangen forsøger at stadig at bevæge sig mod gulvet, men bolden blokerer bevægelsen.

Dette registrerer motor-controlleren ikke, og der kan derfor fortsat være fuld spænding på motoren, hvilket kan skade mekanikken og pga. de store strømme også motor-controller boardet.

Analyse For at være sikre på at bevægelse ikke forsøges fortsat når denne er blokeret, må vi først kunne registrere en blokering.

En blokering kan kendes ved at stangen ikke bevæges selvom der er spænding på denne igennem længere tid. Alternativt kan den hastighed for bevægelsen som beregnes på boardet anvendes til at registrere hvornår bevægelse ikke finder sted.

Efter en blokering er fundet, er det ønskeligt at spillet dels fortsætter uden at hardwaren skades, dvs. uden at den implicerede stang drejes i blokeringsretningen, og dels at det fortsættes uden indgriben fra brugeren. At sætte spillet igang igen kan enten overlades til AI i hovedprogrammet, eller også kan der på motor-controlleren drejes 2π i modsat retning, hvor der dermed vil være stor chance for at bolden skydes væk fra blokeringsstedet. Førstnævnte løsning er mere tidskrævende at implementere, men mere elegant, da det ikke vil være hensigtsmæssigt at skyde baglæns i blinde og dermed risikere at forårsage et selvmål.

Design En tæller holder styr på i hvor lang tid, der ikke er registreret bevægelse i, mens bevægelse har været *ønskeligt*. Her betyder at bevægelse er ønskeligt, at den nuværende position til et vilkårligt tidspunkt ikke er den ønskede position. Denne tæller nulstilles når bevægelse registreres, men også når der fra hovedprogrammet er modtaget besked om at der skal flyttes modsat bevægelsesretningen (årsagen er at finde i bilag B.2).

Et interval, der definerer hvor mange encoderimpulser der skal til før *bevægelse* har fundet

sted skal findes, da man kan forestille sig at en blokeret stang vil aktivere encoderen, hvis den bevæges bare en lille smule mens den presser mod bolden, da opløsningen på encoderne er meget høj. Desuden skal det evalueres hvor lang tid der skal til uden bevægelse for at det erklæres at udgøre en blokering. Sammen er disse værdier afgørende for om semi-reset aktiveres på de rigtige tidspunkter. Der skal helst ikke gå for lang tid fra en blokering reelt sker, til at H-broen afbrydes, og semi-reset må ikke aktiveres når der ikke reelt er blokering.

Når det omtalte stykke tid er gået uden bevægelse, erklæres dette at være en blokering. Derfor stoppes al bevægelse på aksen øjeblikkeligt. Denne tilstand, hvor al bevægelse på aksen er låst, kaldes herefter *semi-reset*, da det er en form for reset uden nulpunktskalibrering. Her skal hovedprogrammet på computeren tilkendegive, at den har registreret en blokering før aksen låses op.

Hermed overlades videre håndtering af blokeringer til AI, som derefter kan forsøge at få bolden ud af den låste position.

Implementering Tiden der går før semi-reset aktiveres, er sat til 0.2 ms, da denne værdi sikrer at en velfungerende stang har haft nok tid til at accelerere, og at H-broen sandsynligvis vil afbrydes før komponenter overopheder.

For at computeren holdes opdateret vedr. tilstandsskifte, sendes både stangens position og en tilstandsidentifikator tilbage, hver gang computeren spørger om stangens position.

Et nyt kald over serielkommunikationen muliggør deaktivering af semi-reset tilstanden.

På boardet sidder en lysdiode. Dennes funktion omskrives, så den er tændt hvis stangen er ikke nulpunktskalibreret eller hvis semi-reset tilstanden er aktiveret.

Test Bolden placeres så den forårsager en blokering. Idet stangen drejes ned i bolden, slås H-broen fra og lysdioden på boardet tændes, hvilket indikerer at boardet er gået i semi-reset tilstand. Tilstanden bliver registreret på computeren, og det bemærkes at det ikke længere er muligt at flytte stangen, på trods af at nye positionsønsker sendes ud fra computeren. Semi-reset afbrydes herefter fra computeren, hvilket indikeres ved at lysdioden slukkes og stangen kan påny bevæges.

Under længere tids kørsel er desuden testet om semi-reset tilstanden kan udløses uden blokering, hvilket ikke skete under tests.

3.4 Løse endestop

Endestoppene sidder meget løst, og de skal enten fastgøres ordentligt eller fjernes helt [4]. Hvis endestoppene fjernes, skal sidekollisioner håndteres på anden vis. Dette problem er analogt med blokering af rotationsbevægelsen, idet der er tale om en blokering, som bør føre til

afbrydelse af motoren.

Da der er implementeret en løsning for blokering med bolden, vurderer vi at det er nemmest at fjerne endestoppene og anvende samme løsningsmetode til registrering af blokeret bevægelse som for blokering med bold. Normalt sørger hovedprogrammet for at undgå kollisioner med siderne, så det sker kun sjældent.

Test Ved at sende en ønsket position uden for det bevægelige interval, ses det at stangen kører hårdt ind i siden, hvorefter H-broen slås fra og lysdioden på boardet tændes, for at indikere at boardet er gået i semi-reset tilstand.

Desuden testes tilsvarende over længere tid om semi-reset tilstanden udløses uden blokering, hvilket ikke skete under tests.

3.5 Fejlregistrering af impulser fra encoder

Erfaringer har vist at det antal impulser der sendes fra encoderne ikke altid stemmer overens med det antal, der modtages på motor-controlleren[3]. Impulser kan gå tabt eller falske impulser kan genereres, og derved mister motor-controlleren orienteringen i sin bevægelse. Dette sker for rotationsbevægelsen, og i mindre grad for sidelænsbevægelsen. Det er et problem, da det f.eks. kan ske at målmanden vender på hovedet, når den tilhørende motor-controller tror den dækker sit mål.

Fænomenet er beskrevet i [1], [3] og [4], men de direkte årsager er ikke fundet. Problemet ligger formentlig i hardwarekonfigurationen, da softwaredelen, der implementerer interrupts fra encoder og registrering af bevægelse er meget enkelt opbygget. Da vi ikke er interesseret i at udføre større ændringer af motor-controllerhardware, men istedet fokuserer på at få software og hardware til fungere som ét kampdygtigt produkt, vil vi bringe eksisterende hardware til at virke tilfredsstillende, med vores mål for øje.

To faktorer har været medvirkende til at minimere problemet:

- Kompensering i motor-controllersoftware.
- Udskiftning af strømforsyning.

3.5.1 Kompensering i motor-controllersoftware

Skulle fejl opstå under registreringen af impulser, vil der være nogle ting, der kan afsløre dette for motor-controllerne. I visse tilfælde kan der kompenseres for disse fejlregistreringer i motor-controllersoftwaren.

Rotationsbevægelsen Som skrevet er der indekspuls på rotationsaksen. Hvis denne bliver høj på et tidspunkt hvor den nuværende position ikke er 0, indikerer dette uoverensstemmelser mellem den faktiske position og den registrerede.

Det antages at indeks impuls interrupts ikke er falske, og de har derfor større troværdighed end den registrerede nuværende position. Den nuværende position justeres til 0 når indekspulsen er høj. Denne nulpunktskalibrering vil foregå under spil hver gang et skud foretages.

Sidelænsbevægelsen Der sidder ikke en indekspuls på encoderne til sidelænsbevægelsen, men her nulpunktskalibreres istedet efter spillets sider. Hvis en stang rammer ind i en side under spil pga. fejlregistrering af impulser, vil semi-reset tilstanden aktiveres. Semi-reset tilstanden for sidelænsbevægelsen vil kun aktiveres, hvis der er kollision med siderne, eller hvis fremmedobjekter forekommer i spillet, da det er fysisk umuligt for sidelænsbevægelsen at blive blokeret af bolden. Derfor antager vi at der er tale om kollision med en side når semi-reset tilstanden aktiveres og istedet for at låse aksen og gå i semi-reset tilstand justeres den nuværende position, afhængigt af hvilken af siderne den nuværende position er nærmest, og spillet fortsættes.

Test Begge kompenseringsmetoder er testet i bilag B.3.1, hvor det konkluderes at de fungerer efter hensigten.

3.5.2 Udskiftning af strømforsyning

Før dette projekt har robotten ikke været i funktion med alle stænger på samme tid. Tidligere er kun kørt med én stang af gangen, der fik leveret strøm af en enkelt 13.8 V 10 A strømforsyning. Motor-controlleren er specificeret til minimum ca. 10 V, men nye forsøg viser at spændingen til boardet falder helt ned til 9 V og svinger med op til 6.4 V med denne strømforsyning (se bilag B.3.2).

Dette kunne skyldes at strømforsyningen ikke regulerer hurtigt nok, men også at der trækkes en for stor strøm. Strømmen som motortypen principielt maksimalt kan trække, fås udfra målinger af den indre modstand for motortypen i [16].

$$U = RI \Leftrightarrow I = \frac{U}{R} \Rightarrow I = \frac{12\text{V}}{0.4\Omega} \approx 30\text{A}$$

Derfor er en 10 A strømforsyning ikke anvendelig til forsyning for en hel stang, men dog måske en enkelt motor, da denne strøm højst vil blive trukket i peaks.

Når spændingen svinger så meget kan det også tænkes at støjen faktisk opfattes af microcontrolleren som falske impulser, i højere grad end der er tale om tab.

En enkelt 12V 48A strømforsyning blev anskaffet til brug på alle stænger. Det er antaget at 48A er tilstrækkeligt, da vi ikke får brug for at skyde flere steder på banen samtidigt, og

det ellers skulle være uheldigt hvis flere motorer trak peakstrøm samtidigt. Ved udskiftning til denne strømforsyning oplevede vi en betydelig reduktion af impulstab og/eller de falske impulser der før observeredes. Målinger er foretaget, hvori det undersøges, hvor meget spændingen falder med den nye strømforsyning (se bilag B.3.2), med tilfredstillende resultater. Spændingen falder til 11.4 V og svinger med 1.8 V. Disse målinger sammenholdt med observationerne peger på en klar sammenhæng mellem de falske impulser (eller impulstab) og spændingsfaldet over boardet.

3.6 Konklusion

Motorstyringen er nu klar til anvendelse i større sammenhæng. Varmeudviklingen i motorerne er mindsket, så det er muligt at spille minimum én kamp uden overophedning.

Når en stang blokeres, registreres det af motor-controlleren og systemet stopper sig selv, inden hardwaren ødelægges.

Impulstab er minimeret til et acceptabelt niveau, og fejlene akkumuleres ikke længere over tid, da nulpositionen kalibreres under spil.

Kapitel 4

Billedgenkendelse

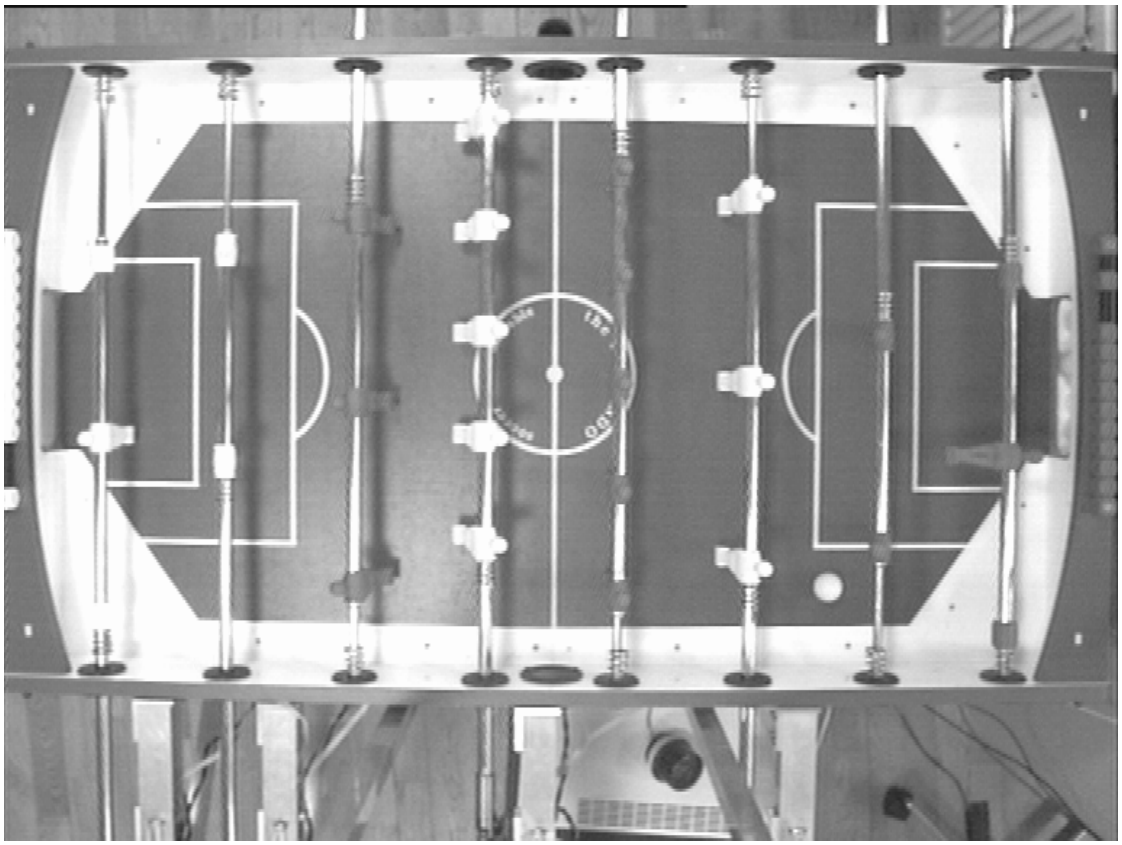
For at kunne spille bordfodbold, er det nødvendigt at kunne følge med i hvor bolden befinder sig på banen.

Fra kameraet får vi et billede 25 gange i sekundet. Disse billeder er i sidste ende blot byte arrays og siger som udgangspunkt intet om boldens position på bordet. Derfor skal billeddataene analyseres for at finde denne position. Det gør vi i to tempi: vi finder bordets position på billedet og finder boldens position på billedet. Deraf kendes boldens fysiske position i forhold til bordet. Hovedprogrammet bruger det meste af den tid det kører, på at køre disse algoritmer igennem.

4.1 Om billedet

Billedet fra kameraet er defineret med x-aksen stigende mod højre, y-aksen stigende nedad og nulpunktet værende den øverste pixel til venstre.

Farverne i billedet er defineret i YUV systemet beskrevet i [7]. Der benyttes tre farveværdier Y, U og V til at definere en enkelt farve. Y angiver hvor lys en farve er og UV angiver tilsammen selve farvenuancen. Hvis man viser hver farvekanal som et sort-hvid billede med sort svarende til den laveste værdi i vores format og hvid svarende til den højeste, så ser vores billede ud som vist i figur 4.1 og 4.2. Når vi herefter refererer til mørke eller lyse farver på u-kanalen, menes hhv. lave og høje u-værdier. Det tilsvarende gælder for v-kanalen. Hver kanal har en opløsning på 1 byte per pixel og kan derfor angive 256 nuancer.



Figur 4.1: Y-kanalen fra kameraet

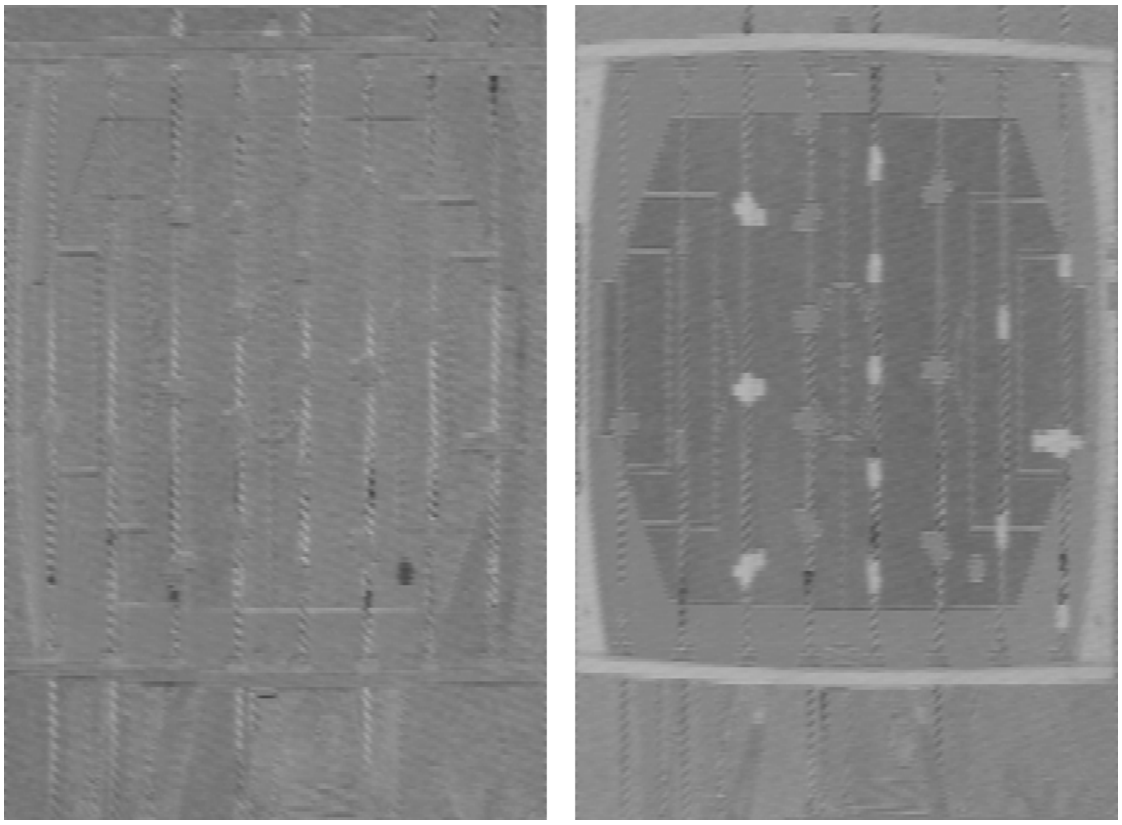
4.1.1 Interlacing

Billedet fra kameraet er interlaced. Dvs. at det optager hver anden linje først og derefter de resterende linjer. Denne forsinkelse mellem de to sæt linjer gør, at billedet egentlig indeholder to billeder, med det ene taget lidt senere end det andet. Effekten af dette kan ses på figur 4.3, hvor der ses to bolde på samme billede.

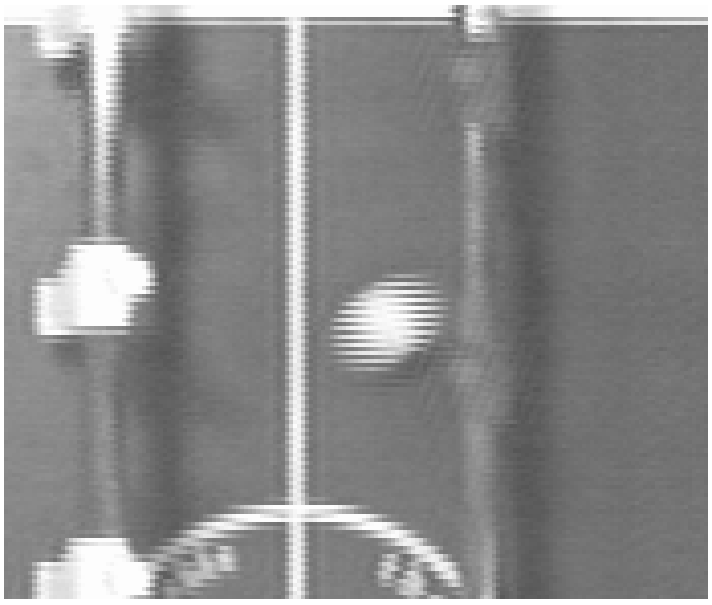
Derfor blev billedet i [4] delt op i to, med det ene sæt linjer i det ene billede og det andet sæt i det andet billede. Derved får vi to billeder per frame.

4.1.2 Deformering

Billedet kan være roteret en smule, idet kameraet ikke nødvendigvis sidder helt lige. Ved korrekt justering af kamerastativet, kan rotationen dog minimeres, således at spillefladens rette sider svarer til vandrette linjer pixels i billedet. Yderligere kan det tænkes, at kameraet ikke ser lodret nedad, men en smule skråt. Dette vil føre til, at et vandret rektangel i virkeligheden ser ud som en trapez på billedet. Dette kalder vi trapezeffekten. I bilag C.1 måles den samlede maksimale virkning af rotationen og trapezeffekten. I forhold til en retvinklet tolkning af billedet giver disse effekter samlet højst en afvigelse på 2 pixels, hvilket svarer til 7 mm



Figur 4.2: Til venstre: u-kanalen. Til højre: v-kanalen fra kameraet. Disse kanaler har på x-aksen den halve opløsning af y-kanalens, fordi øjet ikke er lige så følsomt over for disse.



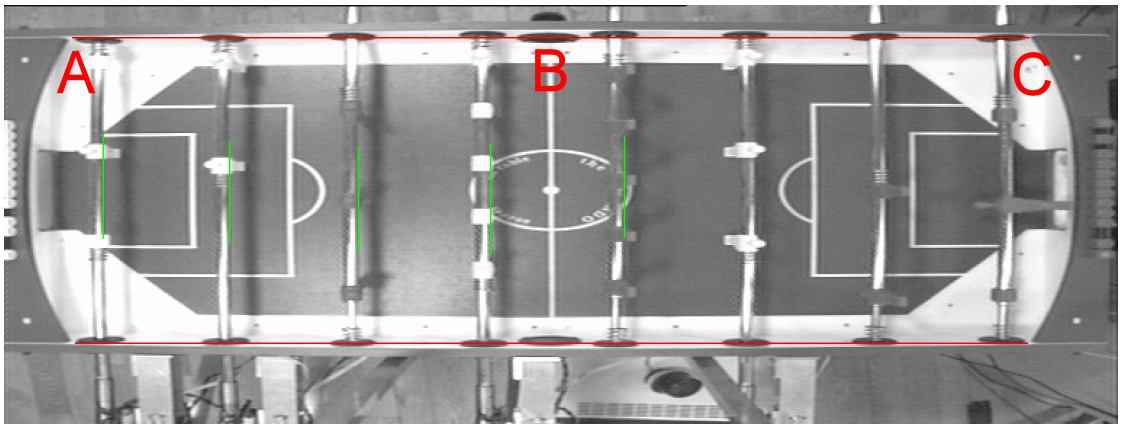
Figur 4.3: I midten en bold i bevægelse på den interlacede y-kanal.

på spillefladen. Effekten kan dog kun være så stor i visse hjørner af bordet, hvor spillerne alligevel ikke kan nå bolden, så på den interessante del af spillefladen drejer det sig snarere

om 0 til 1 pixel. Det er så lidt, at vi fravælger at tage højde for rotationen og trapezeffekten i softwaren.

Pga. linsens form er billedet tøndeforvrænget. Dvs. at rette, parallelle linjer i virkeligheden ser ud som siderne på en tønde set fra siden på billedet. Den maksimale størrelse af denne effekt er målt i bilag C.1 til 4 pixels. Ved kanten af bordet er forskellen på y-koordinaten for et hjørne og for midten af kanten altså 4 pixels. Hjørnet og kanten kan være A og B som vist på figur 4.4. Ved i kalibreringen at benytte et gennemsnit for midt- og hjørnepunktets y-koordinater, vil den maksimale afvigelse for ethvert punkt på billedet være 2 pixels, altså 7 mm. Fejlen vil dog kun være så stor visse steder i kanten af billedet.

Vi har desuden målet en forvrængning af billedet langs bordet (se bilag C.1). Afstanden mellem par af røde linjer på figur 4.4 er ens i virkeligheden, men ikke på billedet. Det ville det være en fordel at slippe af med disse forvrængninger af billedet, men det er ikke noget vi har prioriteret højt nok til at gøre.



Figur 4.4: Tøndeforvrængningen ses ved afstanden mellem den vandrette linje og bordets kant i områderne A, B og C. Mellem de røde linjer er der samme afstand i virkeligheden men ikke på billedet.

Billedgenkendings- og kalibreringsalgoritmerne antager derfor at rette linjer i virkeligheden svarer til rette linjer på billedet. Desuden antager de, at kanterne på spillefladen svarer til vandrette og lodrette linjer på billedet.

4.1.3 Konvertering af farvekanaler

Man kunne forestille sig, at det med de YUV farvekanalere var svært at genkende de ting vi skal på billedet, eller i hvert fald at det var lettere i et andet format, f.eks. RGB, CMYK eller HSL. Hvis det var tilfældet, kunne vi konvertere billedet til det format i hver eneste frame. Det er ikke så besværligt. At konvertere til RGB foregår f.eks. vha. en lineartransformation [7]. Problemet er, at det tager tid i hver eneste frame at gøre det, hvilket giver programmet mindre tid til at lede efter bolden, o.a. Vi vil derfor se, hvor meget vi kan opnå med de rå billeder og undersøger ikke sagen nærmere.

4.1.4 Filtre

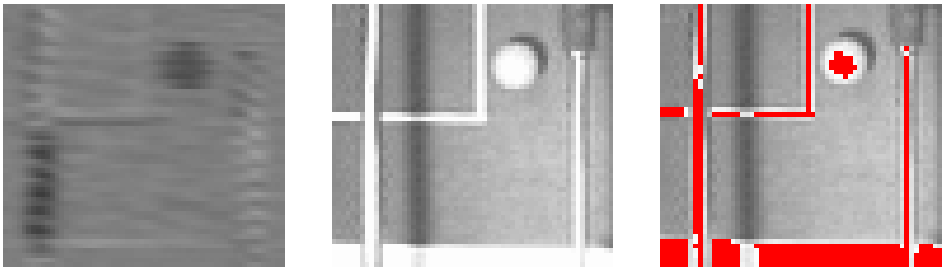
Der findes filtre, altså algoritmer til at ændre billeder. Disse kan reducere støj, gøre billedet skarpere, ændre farverne og meget andet. Det kan være, at man ved at køre et filter på billedet, kunne gøre det lettere at genkende bolden og mændene. Igen er problemet med filtre at det tager tid at køre dem - tid, som kunne være bedre brugt på at lede efter bolden uden filtre. Det viser sig at være unødvendigt at bruge filtre, så det gør vi ikke.

4.2 Boldgenkendelse

4.2.1 Valg af boldfarve

Bordfodbolde kan købes i mange forskellige farver [9]. For lettest muligt at kunne genkende bolden, gælder det om at vælge en god farve til den. Hvilke farver der er lettest at genkende, analyseres i bilag C.2. Analysen bygger på, hvilke farver der er unikke på hvilke kanaler og især på, om de kan forveksles med genskin fra stængerne. Konklusionen i analysen er at gul er en god farve at vælge, men blå er en anelse bedre.

Hvis man vælger en gul bold, vil det være nødvendigt geometrisk at udmaske de områder på billedet hvor stængerne er, for ikke at forveksle genskin i stængerne med en bold. Forfatterne til [1] kom frem til at denne farve var bedst, dog uden en dybdegående analyse. De regnede med, at man kunne bruge en gul blod og samtidig undgå den geometriske udmaskning. De lavede en farvebaseret udmaskning på baggrund af at genskinnet normalt er meget lyst på y-kanalen. Det er rigtigt nok lyst, men det er bolden også på denne kanal. Det kan ses på figur 4.5, hvor den lyseste farvenuance der optræder på y-kanalen, 253, er erstattet med rød. Det ses tydeligt at en stor del af bolden har samme lyse nuance. Altså vil en udmaskning baseret på farvenuancen også udmaske bolden.



Figur 4.5: Genskin kan optræde mørkt på u-kanalen som vist til venstre. På y-kanalen i midten optræder det lyst, men det gør bolden også. Til højre er y-kanalens lyseste farve erstattet med rød.

En blå bold kan forveksles med genskin på u-kanalen, men dette kan tilsyneladende udmaskes på baggrund af genskin på v-kanalen. Altså ville man sandsynligvis kunne undgå så meget geometrisk udmaskning som er nødvendig med en gul bold.

Vi havde i forvejen nogle gule bolde og boldgenkendingsalgoritmen var fra forrige projekter

indrettet til at genkende en gul bold. Derfor har vi ikke prøvet at benytte en blå bold, men dette kunne være et logisk næste skridt. Dog kan man sige, at en blå bold ikke vil virke så "naturlig" for modspilleren at spille med, da gule bolde er mere udbredte. At bruge en blå bold er lidt at indrette spillet efter robotten og ikke omvendt.

4.2.2 Algoritme

Selve boldgenkendelsesalgoritmen har vi overtaget fra de tidligere bordfodboldprojekter. Vi har i dette projekt udvidet den med følgende

- Algoritmen tager højde for billedets kalibrering
- Der tages højde for parallakseforskydning
- Boldens position finjusteres vha. udregning af dens massemidtpunkt

Selvom boldgenkendelsesalgoritmen var implementeret inden dette projekt, beskrives den alligevel her, da det er nødvendigt for forståelsen af vores udvidelser.

Lokalisering af bolden foregår ved, at en algoritme undersøger pixels overalt på spillefladen og returnerer positionen for den, der mest ligner midtpunktet af en bold. Selvom bolden i praksis hopper nogle gange, antages det at den befinder sig i planet, da ingen spillere udnytter at den hopper. Denne algoritme benyttes som udgangspunkt, men med en række forbedringer.

På u-kanalen står bolden frem som en mørk plet. At udregne, hvor meget en pixel ligner boldens midpunkt, gøres ved at tælle hvor mange af de omkringliggende pixels, der er mørkere end en vis tærskelværdi. Desuden benyttes største- og mindsteværdier på y- og v-kanalerne som yderligere verifikation.

4.2.2.1 2 gennemkørsler

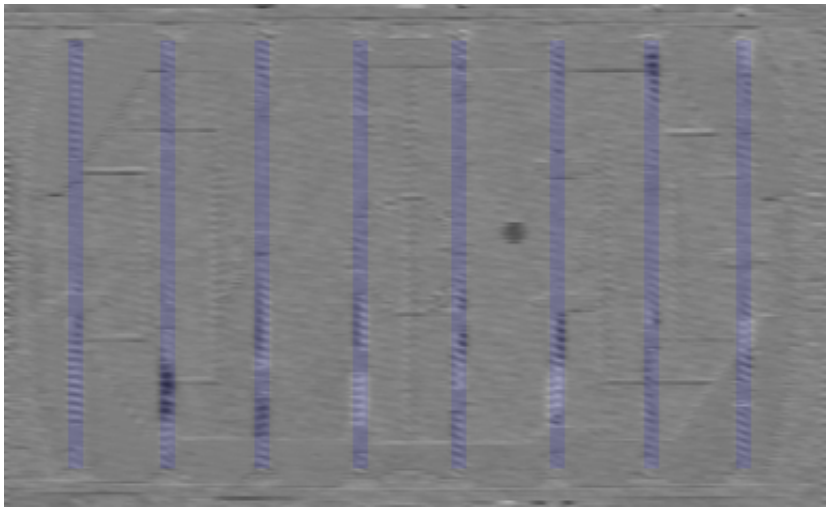
For at optimere algoritmens hastighed køres den igennem to gange. Først for hele spillefladen, men hvor der kun undersøges et lille område omkring hver pixel - altså et område som er mindre end boldens størrelse på billedet. Alle de pixels der ligner "tilstrækkeligt" meget huskes og bagefter køres algoritmen igennem igen. Denne gang kun for disse pixels, men til gengæld for områder på størrelse med bolden. Det område der ligner mest, må være et billede af bolden.

4.2.2.2 Udmaskning af stænger

Da genskin fra stængerne kan ligne bolden på alle tre kanaler, maskes stængerne væk i algoritmen. Algoritmen ignorerer altså de områder, hvor stængerne er. Det er vigtigt at gøre

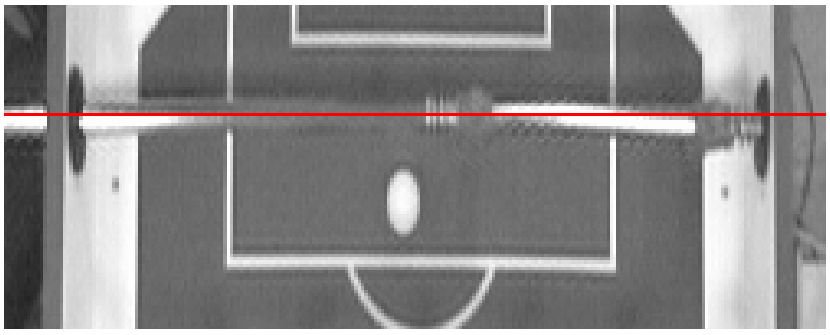
dette præcist, for hvis der maskes for meget væk, bliver bolden meget let skjult under stangen. Hvis der maskes for lidt væk, vil algoritmen nogle gange returnere forkerte boldpositioner. Det sker især, når bolden er skjult og der er genskin på et område af en stang, som ikke er masket helt væk.

Erfaring viser, at modspilleren kan udnytte manglende afmaskning ved at skjule bolden, når han har den under kontrol i angrebet. Hvis forsvaret da bevæger sig til siden, for at dække for noget genskin, som det tror er en bold, kan modspilleren let score.



Figur 4.6: Under de rektangulære masker vist med blåt, ledes der ikke efter bolden. Derved fjernes de mørke områder, der ellers kunne ligne bolden.

Det er svært at lave en ordentlig maske for stængerne, idet bordet på billedet er tøndeformet. Derfor ser især de yderste stænger meget buede ud, som vist på figur 4.7. Derved er det ikke optimalt blot at maske rektangulære områder ud. Sådanne vil typisk maske for meget og/eller for lidt væk. For de yderste stænger vil det være en fordel at benytte en buet maske. Det har vi dog ikke prioriteret højt nok til at implementere, men benytter blot rektangulære masker som dem vist på figur 4.6.



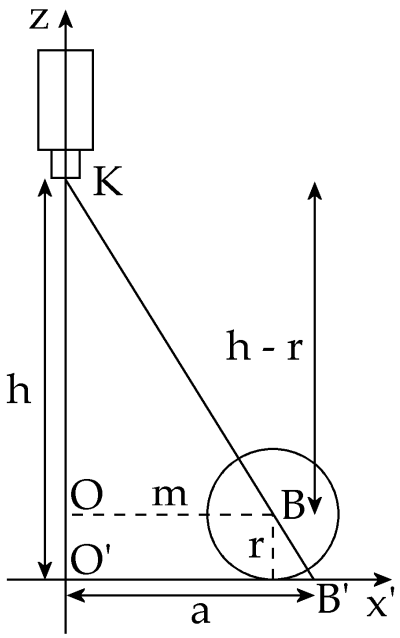
Figur 4.7: Billedets tøndeeffekt er tydeliggjort med den røde linje.

4.2.2.3 Massemidtpunkt

Som algoritmen er beskrevet indtil nu, vil boldens præcision aldrig blive bestemt med højere opløsning end ca. 3.5 mm, idet hver pixel svarer til 3.5 mm på spillefladen. Dette kommer vi udenom ved at udregne et massemidtpunkt for bolden, når dens midtpunktspixel er fundet. Midtpunktspixlen bruges som et estimat for positionen og denne finjusteres ved massemidtpunktsberegningen. Detaljer ang. beregningen af dette findes i bilag C.3.1.

4.2.2.4 Parallakseforskydning

Når bolden ikke ligger fuldstændigt lodret under kameraet, er der parallakseforskydning på den fundne boldposition. Bolden ser ud som om den er længere væk end den er. Det er illustreret på figur 4.8. Bolden er afstanden m fra kameraets position, men ses som værende i afstanden a. h er kameraets højde over bordet og r er boldens radius.



Figur 4.8: Boldens genkendte position er parallakseforskudt, da den ikke ligger lodret under kameraet.

Når boldens position er fundet, kompenseres for denne effekt. Der er tale om ligedannede trekanter KOB og $KO'B'$, så m findes nemt:

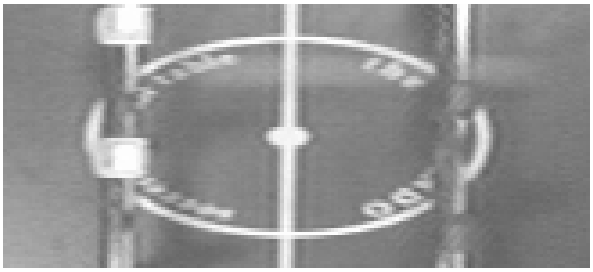
$$\frac{h}{a} = \frac{h-r}{m} \Leftrightarrow m = \frac{(h-r)a}{h}$$

m benyttes da i stedet for den fundne position. Dette gøres for både x- og y-aksen.

4.2.2.5 Konturalgoritme

Vores algoritme benytter ikke det faktum at bolden er rund. Man kunne godt konstruere en mønstergenkendelsesalgoritme, som ledte efter noget cirkelformet med den rigtige farve på spillefladen. Dog er det ikke så nemt, da algoritmen ikke skal nøjes med at genkende cirkler:

- Hvis bolden er delvist skjult af stænger og mænd, er den ikke længere cirkelformet. Der skal altså laves en algoritme som også kan finde cirkelkalotter, cirkelkiler og vilkårlige andre cirkeludsnit.
- Desuden er en bold i høj fart aflang og udtværet på billedet og altså heller ikke cirkelformet, som det ses på figur 4.9. Vores algoritme tager dog heller ikke højde for dette.



Figur 4.9: En bold i høj fart. Bolden ses aflang og udtværet på billedet.

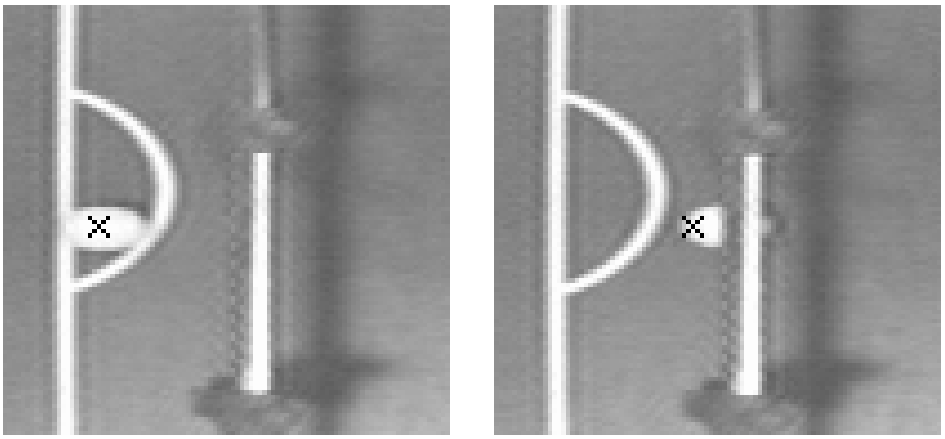
At lave en algoritme der kan dette lyder umiddelbart sværere end vores algoritme. I øvrigt konkluderede [2] også vha. en primitiv prototype af en konturalgoritme (som ikke tog højde for cirkeludsnit), at det ikke var en fordel at finde bolden på denne måde.

Hvis bolden er delvist skjult, vil vores algoritme ikke finde boldens præcise position. Den vil blot finde midtpunktet af det kameraet kan se. Dette kan ses på figur 4.10, hvor det fundne midtpunkt er længere til venstre end boldens egentlige midtpunkt. Når bolden er delvist skjult er det oftest under en stang, hvilket gør at den fundne position er rykket i x-aksens retning, dvs. mod eller væk fra fodboldmålet. Dette ikke så kritisk, idet mændene dækker op foran bolden ved at stille sig foran den fundne position. Hvis forskydningen var i y-aksens retning, altså sidelæns i forhold til målet, ville det være et større problem.

En oplagt udvidelse af algoritmen er at tjekke, om bolden er delvist skjult. Når det er tilfældet, kunne en undersøgelse af boldens kontur give et bedre estimat af boldens position. Dette er vigtigt, hvis man vil håndtere bolden med stor præcision, idet robottens egne stænger også tit delvis dækker bolden, når kameraet som vores er over bordet.

4.2.2.6 Optimering vha. tidligere kendt position

En optimering af den gennemsnitlige køretid, som vi har fravalgt at implementere er, at boldfindingsalgoritmen kunne nøjes med at lede efter bolden på et mindre område af spillefladen.



Figur 4.10: Til venstre en udækket bold og til højre en halvt dækket bold. Den af programmet genkendte position er indikeret med et kryds.

Ud fra den genkendte position fra forrige billede og evt. også fra viden om dens hastighed, kunne algoritmen få et rigtig godt estimat for boldens position. Algoritmen kunne som udgangspunkt nøjes med at lede efter bolden i dette område og kun lede på hele spillefladen, hvis den ikke blev fundet i området, eller hvis dens forrige position slet ikke var kendt. Derved kunne man i de fleste frames reducere den tid det tager at køre boldfindingsalgotmen.

To argumenter taler imod det. Dels er det en stor fordel, hvis denne fulde gennemkørsel, kan nå at ske inden det næste billede kommer ind, sådan at den fundne position ikke er forældet med det samme. Altså skal algoritmen kunne køres igennem på den fulde spilleflade i løbet af samme størrelse tidsrum, uanset om vi laver denne optimering eller ej. Desuden får vi ikke så meget ud af at algoritmen gennemsnitligt kører hurtigere, da vi alligevel er tvunget til at vente på næste billede.

Et andet argument imod denne form for optimering er, at hvis noget andet end bolden fejlagtigt bliver genkendt som bolden, så kan det blive ved med at blive genkendt som bolden i lang tid, idet algoritmen ikke kigger på resten af bordet. Disse to forhold gør tilsammen, at vi undlader at benytte denne form for optimering, selvom den ville være meget nem at implementere.

4.3 Genkendelse af modspillerens mænd

4.3.1 Grundlæggende genkendelse

Det kan i visse tilfælde være en fordel at vide, hvor modstanderes mænd befinder sig - først og fremmest for at skyde uden om dem, når de står foran bolden. At kende deres vinkler er mindre vigtigt, da de alligevel næsten altid peger nedad.

Derfor skal der laves en algoritme til at finde ud af, hvor de røde mænd er. Det er det samme som at finde ud af, hvor langt stangen de sidder på, er hevet ud. Dette må ske ved at genkende

mændene på stangen, eller i hvert fald en enkelt af dem, da der er ikke andre ting på stangen, der kan genkendes og give positionen. Når man kender en enkelt mands position og man ved hvilken mand det er, giver stangens og de andre mænds position sig selv.

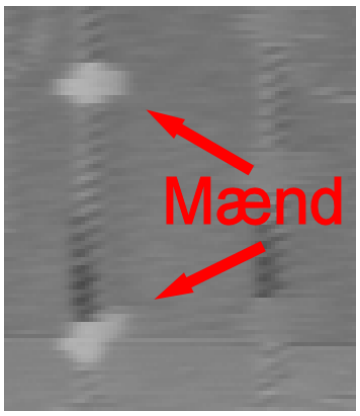
For at genkende en rød mand, er det lettest at bruge v-kanalen. Det kan ses på figur 4.11. Mændene står tydeligt lyse frem. De andre kanaler er ikke værd at overveje i denne sammenhæng.

Stængerne er i x-retningen altid samme sted forhold til banen og mændene kan derfor kun bevæge sig med en enkelt frihedsgrad. Derfor er mændene grundlæggende nemmere at genkende end bolden, som kan bevæge sig med to frihedsgrader.

Forudsat at systemet ved, hvor stængerne er, kunne en skitse til en mandfindingsalgoritme være, at undersøge stangen fra den ene ende til den anden i små skridt. For hvert skridt kan det beregnes, om manden er på den pågældende position. Så snart manden er fundet, afsluttes algoritmen og stangens position udregnes ud fra mandens position. Grundlæggende virker det som en, måske lidt ineffektiv algoritme, som vil fungere efter hensigten. Den kunne f.eks. gøres hurtigere, ved at benytte forrige position som gæt på næste position. Da vi vil vinde meget lidt med en smartere algoritme, tænkes der ikke nærmere over alternativer.

Manden der ledes efter skal helst være den, som er tættest på robotten. Det skyldes at visse modstandere står med hovedet lænet ind over bordet, og derfor kan skygge for de mænd der er tættest på dem selv. Altså starter algoritmen nedefra i billedet (høje y-værdier) og undersøger stangen i små skridt opad (aftagende y-værdier).

For hver position der undersøges, skal det udregnes om manden er på denne position. På figur 4.11 ses det, at manden er indikeret ved et et lyst område, der oven- og nedenfor grænser op til mørkere områder.



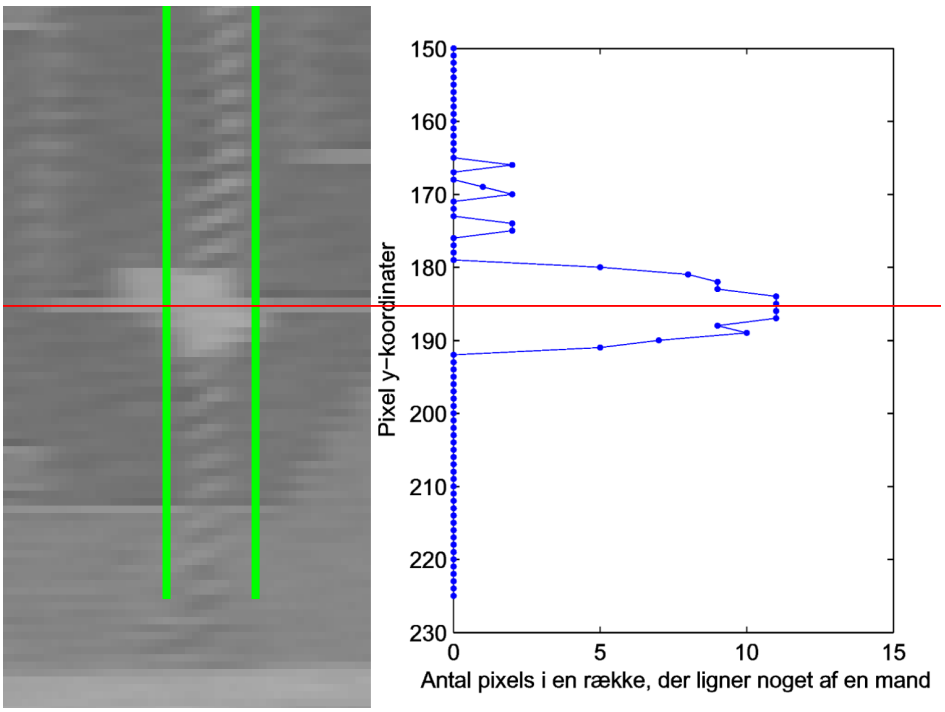
Figur 4.11: På v-kanalen er de røde mænd lyse. Det virker oplagt at lede efter noget lyst i stangens retning.

I en vis bredde omkring stangen må algoritmen se, hvor lyst der er og bruge dette til at udregne, hvor meget dette område ligner en mand. Et mål for, hvor meget en enkelt pixel ligner, er at dens farveværdi skal være en anelse lysere end gennemsnittet for hele stangen og dens nærliggende område. Antallet af pixels i en enkelt række, der ligner en mand, giver

summeret et mål for, hvor meget hele rækken ligner en mand. Dette mål kaldes i det følgende for *likeness-værdien*.

Det vil give mening at undersøge enkelte rækker af pixels ad gangen. Likeness-værdier ses på figur 4.12 holdt op mod billedet på v-kanalen. Værdierne ligger højt, som en bule på grafen, hvor manden er og er ca. nul på resten af grafen. Denne "bule" på grafen indikerer altså, hvor manden er og vil blot blive omtalt som bulen i det følgende.

På figur 4.12 ses omkring pixel række 165 til 175 også lidt støj, der ligner en mand. Fejlmålinger pga. støj kan minimeres ved at antage, at en mand er større end en tærskelværdi - både i likeness og bredde.



Figur 4.12: Der søges på v-kanalen efter en mand mellem de to linjer vist med grønt. Ud for hver række pixels er likenessværdien vist på grafen. Mandens massemidtpunkt blev fundet ved $y = 186$, den røde linje.

4.3.2 Analyse af algoritmedesign

Ved at undersøge mange rækker, kan en algoritme finde manden. Der er flere måder at gøre dette på.

4.3.2.1 Tærskel på opadgående side af bulen

En algoritme kunne finde likeness-værdien for hver række og stoppe så snart den finder en værdi større end en bestemt tærskel. Grænsen skulle i så fald ligge på den opadgående del

af grafen på figur 4.12 og algoritmen vil aldrig undersøge den nedadgående del. Midtpunktet af manden vil ligge en smule længere oppe i billedet end det fundne punkt. Et lille konstant antal pixels kan lægges til for at korrigere. Denne algoritme er simpel at implementere, men pga. støj vil det fundne midtpunkt hoppe lidt op og ned i billedet, idet det er lidt tilfældigt hvilken række pixels der først når over grænsen.

4.3.2.2 Første nedadgående del på bulen

En anden mulighed er at undersøge værdier på både den opadgående og den nedadgående del af grafen. Ved at lokalisere hele denne bule på grafen, burde det være muligt at finde et mere præcist mål for midtpunktet, end den ovenfor beskrevne algoritme. Mandens midtpunktet kan da defineres som det første sted på grafen, hvor værdierne begynder at aftage. Dette fungerer, så længe der kun er en enkelt top på grafen og denne top ligger midt på bulen. Algoritmen er følsom overfor støj, idet forskellige rækker pixels nær midten af manden pga. støj kan skiftes til at antage den største likeness-værdi. Det kan ses på figur 4.12, hvor grafens bule har to toppe.

Både denne metode og den ovenfor beskrevne tærskelmetode giver præcision som er diskretiseret efter pixelstørrelsen. Da hver pixel svarer til ca. 3.5 mm, kan mændenes position ikke bestemmes mere præcist end et helt antal gange 3.5 mm.

4.3.2.3 Massemidtpunkt for bulen

En lignende algoritme kunne lokalisere bulen og finde massemidtpunktet for den. Dette vil gøre algoritmen mindre følsom overfor støj og vil desuden komme udenom pixel-diskretiseringen. I princippet kan metoden altså bestemme mændenes position mere præcist end indenfor 3.5 mm.

Det ses på figur 4.12, at massemidtpunktsmetoden bestemmer mændenes position mere præcist end den foregående metode, da denne ville have antaget, at den første lille top på grafen omkring $y = 190$ var over mandens midtpunkt, men det er den jo ikke.

4.4 Kalibrering

4.4.1 Hvad der er brug for

Bold- og mandgenkendelsesalgoritmerne fungerer kun, hvis det til ethvert tidspunkt vides, hvor spillefladen befinder sig på billedet. De bygger på nogle konstante forhold mellem kameraets position og bordets position. Dvs. at spillefladens position og størrelse skal kendes. Desuden skal positionen af modstanderens stænger i x-retningen kendes.

Med øjemål kunne vi godt finde bordets position på billedet og nedskrive pixlernes koordinater

i programkoden én gang for alle. Al den kode vi overtog fra tidligere projekter brugte denne fremgangsmåde. Det er dog en uholdbar løsning, da en lille drejning af kameraet eller et vrid af stativet vil fejlagtiggøre programmets opfattelse af virkeligheden, idet bordet ikke længere er, hvor programmet tror. Vi valgte dengang at nedprioritere kalibrering, da robusthed ikke var væsentligt.

Kalibrering er altså nødvendig. Den kan foregå med forskellig hyppighed afhængigt af, hvor ofte det er nødvendigt. Det er relevant da det tager tid at kalibrere og vi er begrænsede på det punkt. Mulighederne er:

- En enkelt kalibrering ved spillets opstart.
- Kalibrering en gang i mellem. Måske nogle gange i minuttet eller når man ved måling finder ud af, at det er nødvendigt.
- Kalibrering i hver eneste frame.

Ud af de størrelser der skal kendes, kan kun y-positionen af spillefladen rykke sig betydeligt mens der spilles, da kamerastativet er langt mere stabilt udformet i x-retningen end i y-retningen. Rystelserne skyldes en kraftig vibrering af kameraet, som altid foregår når der spilles på bordet. Vibrationerne er så hurtige, at kalibering af y-positionen er nødvendig i hver eneste frame. Resten af størrelserne er konstante mens der spilles. Det har vi gennem testkampe i bilag F erfaret ved, at robotten ikke bliver dårlig til at ramme bolden, efterhånden som der spilles.

4.4.2 Kalibrering ved opstart

4.4.2.1 Muligheder

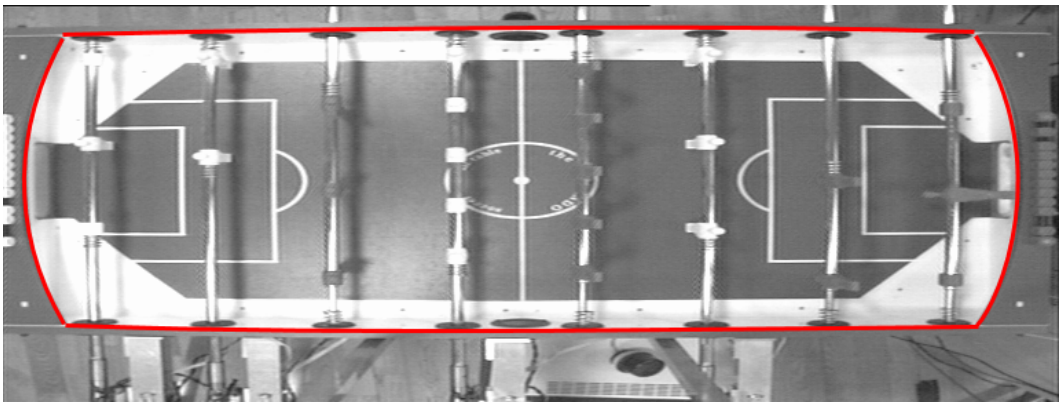
Startkalibreringen skal finde spillefladens position og størrelse. Desuden skal den finde stængernes positioner. Kalibreringen må foregå ved på billedet at finde træk ved bordet, som vi præcist kender placeringen af i forhold til spillefladen. Da det i sagens natur ikke helt er til at vide, hvor disse træk skal findes, skal de nødvendigvis være letgenkendelige. Generelt kunne man lede efter

- Eksisterende træk ved bordet, såsom form og farve.
- Tilføjede træk, f.eks. klistermærker eller maling.

Begge dele vil fungere lige godt, men dog med en lille fordel ved at bruge eksisterende træk, idet kunstige træk nemmere kan ændres, uden det er meningen, f.eks. ved at tilskuere hiver klistermærker af.

Man kunne sagtens benytte eksisterende træk ved bordet til lokalisering af spillefladen som det blev gjort i [2]. Her blev farveovergangen mellem de hvide sider og de røde kanter på bordet

brugt til kalibrering og det endda i hver eneste frame. Billedet ses med farveovergangene angivet med rødt på figur 4.13



Figur 4.13: Kalibrering af billedet kunne foregå ved at benytte de med rødt optegnede farveovergange.

Det virker lidt kompliceret at gøre, idet kalibreringen på den ene akse nemmest kan foregå, når den anden allerede er kalibreret. Hvis algoritmen starter med at finde den øverste farveovergang, vides det ikke hvor på x-aksen stængerne befinder sig. Derfor kan stænger og mænd let være i vejen, så algoritmen i stedet for den hvide side ser en stang. Hvis algoritmen derimod først skal finde farveovergangen i en af siderne, vil den buede kant og målet tilsvarende genere målingen. Derfor faldt valget på at benytte klistermærker.

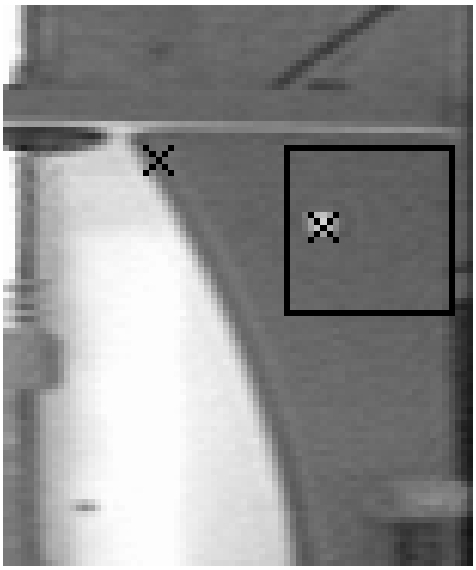
4.4.2.2 Kalibrering vha. klistermærker

På hvert hjørne af bordet påsættes et 1 cm^2 stort klistermærke og ved genkendelse af alle 4 findes bordets position på billedet. Den røde flade, klistermærkerne sidder på, er på y-kanalen meget mørk i forhold til klistermærket og dette mønster med et lyst rektangel med en mørk farve hele vejen omkring sig er let at genkende. Disse klistermærker kan ses på hjørnerne af bordet på figur 4.13.

For at gøre søgningen robust, ledes kun i et fast defineret lille område, hvor vi på forhånd må antage at hjørnet er - der er ingen grund til at lede midt på billedet. Dette område definerer vi rektangulært, da det er det nemmeste at arbejde med. Centrum for hvert rektangel defineres som centrum for det sted klistermærkerne sad i billedet en gang. I mellemtiden er kameraet blevet vredet og drejet, men ikke mere end at klistermærkerne stadig er inde for deres respektive rektangler. Et klistermærke og dets søgerektangel ses på figur 4.14.

For at forhindre, at kalibreringsalgoritmen finder noget der ligner et klistermærke på gulvet, er søgerektanglerne ikke større end de er. At gøre rektanglet større i retning indad mod bordet ville ikke gøre søgningen mere robust. Ved en stor drejning af kameraet ville klistermærkene for den ene ende af bordet da blive indenfor rektanglet, men i den anden ende ville de forsvinde udenfor.

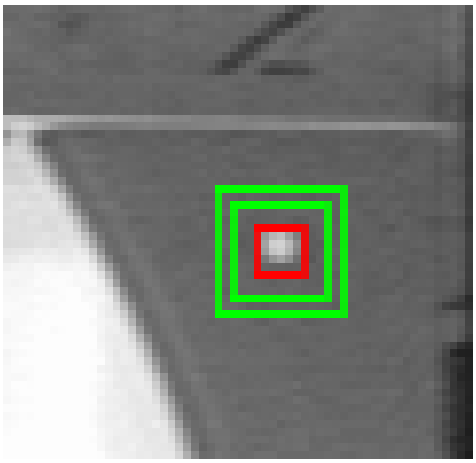
For hver pixel i søgerektanglet undersøges det om den pågældende pixel er et muligt centrum



Figur 4.14: Genkendelse af klistermærket er vist med et kryds. Skråt op til venstre for dette er spillefladens ene hjørne fundet. Omkring klistermærket er tegnet et rektangel, som er det område der gennemsøges, for at finde klistermærket.

for et klistermærke. Så snart et centrum er fundet, afsluttes søgningen.

For hvert af disse mulige klistermærkecentrummer udføres følgende: Da et klistermærke tælles til at fylde ca. 5 gange 5 pixels på billedet, tælles det hvor mange pixels inden for 5 gange 5 pixels der ligner et klistermærke. Hvis en pixel er lidt lysere end gennemsnitspixlen indenfor søgerektanglet, ligner den et klistermærke. For hver pixel i søgerektanglet knyttes derved en værdi, som siger hvor meget det ligner et klistermærkecentrum. Yderligere kræves at de omkringliggende pixels er mørke. Det tjekkes i en kvadratisk rand af pixels i en vis afstand fra centrum. Et resultat af en sådan søgning er vist på figur 4.15.



Figur 4.15: Imellem de grønne linjer er alle pixels mørke og det kan derfor være et centrum. Indenfor det røde kvadrat findes det maksimale antal lyse pixels i denne søgning. Derfor er dette et klistermærkecentrum og algoritmen afsluttes for dette hjørne.

Det bemærkes at klistermærkerne sidder meget yderligt på bordet, og derfor står tilskuerne ofte med hænderne på dem. Hvis de gør det mens kalibreringen foregår, mislykkes kalibreringen og programmet udskriver derfor en fejlmeddelelse for at gøre opmærksom på, at det ikke gik godt.

4.4.2.3 Konvertering

Selve spillefladen er ikke givet ved det rektangel, der udspændes mellem de 4 klistermærker. På figur 4.14 ses forskellen på et klistermærkes position og hjørnet af spillefladen. På begge er indtegnet et sort kryds. Konverteringen er implementeret ved at benytte et en gang for alle opmålt forhold mellem klistermærkernes afstand og spillefladens hjørners afstand. Det er meget simpelt og nærmere beskrevet i appendiks C.3.2.

4.4.2.4 Stængerne

At finde stængernes x-positioner foregår ligesom når spillefladens position skal findes. Forhold mellem stængernes positioner og hjørnerne er opmålt én gang for alle og benyttes, når spillefladens hjørner er fundet. Det er trivielt og nærmere beskrevet i C.3.3.

4.4.3 Kalibrering hver frame

4.4.3.1 Rystelser nødvendiggør kompensering

Kameramonteringen forårsager store rystelser af kameraet. Det skyldes, at kamerastativet ikke har nogen afstivning og rystelser fra bordet derfor meget nemt forplanter sig til det. Stativet bliver sat i svingninger og får kameraet til at svinge fra side til side. Dette får bordet til at hoppe op og ned på kameraets billede og besværliggør derfor billedgenkendelsen. Uden at lave præcise målinger kom vi i [3] frem til, at størrelsen af disse svingninger under normalt spil drejede sig om flere centimeter i hver retning, altså en størrelse lignende boldens. Derfor konkluderede vi, at noget måtte gøres, for at komme rystelserne til livs. Der er tre oplagte løsningsmuligheder på dette problem:

- At afstive kamerastativet, f.eks. ved at montere stænger fra toppen af stativet, ned til den modsatte side af bordet. Det vil reducere rystelserne.
- At hænge kameraet for sig selv, enten i loftet eller på et stativ ved siden af bordet. Det vil næsten fjerne rystelserne.
- Vha. billedgenkendelse at lokalisere kendte træk ved bordet og derved kompensere for rystelserne uden fysiske ændringer af robotten.

Kompensering for rystelserne vha. software må siges at være den mest elegante løsning, dog med den ulempe, at billedet bliver en anelse udtværet pga. bevægelsen. Vi implementerede i

[3] en algoritme til dette og kom derved frem til, at kompensering var praktisk mulig. Pga. kamerastativets udformning behøver kompenseringen kun foregå i y-aksens retning. Desuden er svingningerne hurtige - en frekvens på flere gange i sekundet, så der skal kalibreres i hver eneste frame.

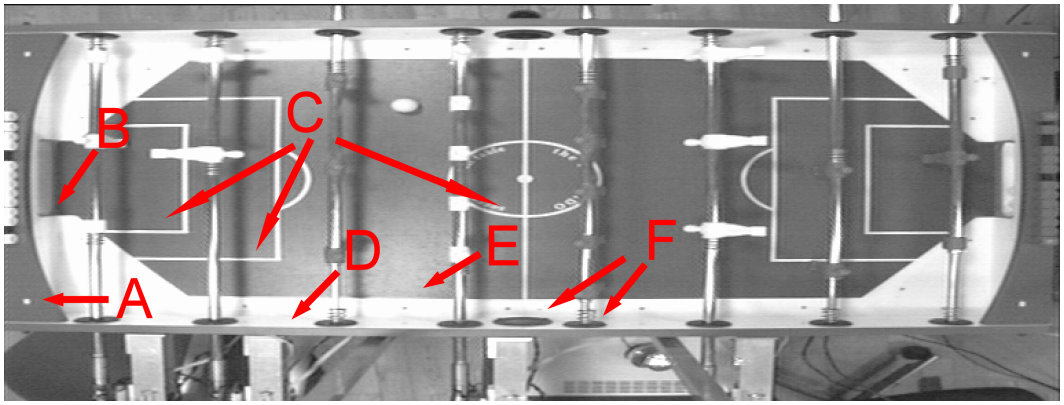
Vi har bygget videre på algoritmen fra [3] og foretaget disse ændringer:

- Algoritmen tager nu højde for den øvrige kalibrering af billedet.
- Der benyttes nu andre farveovergange på billedet, for at gøre kalibreringen mere stabil.
- Algoritmen kompenserede for lidt og det er nu rettet som beskrevet i bilag C.3.4.

4.4.3.2 Der benyttes farveovergange på billedet til kompensering

Strategien for kompensering af rystelserne er i hver frame at finde positionen for en farveovergang på billedet. Ved opstart, hvor det må antages at kameraet ikke ryster, lokaliseres farveovergangen første gang og benyttes i alle efterfølgende frames til at finde ud af, hvor meget billedet har rykket sig. Den nøjagtige fysiske placering af farveovergangen på bordet betyder altså ikke noget, så længe den aldrig flytter sig mens man spiller.

De mest oplagte træk ved bordet til brug ved kalibrering, er vist med pile på figur 4.16. De er alle farveovergange, som skifter i den lodrette retning på billedet.



Figur 4.16: Kalibrering i y-retningen kunne foregå med de på figuren viste farveovergange.

Hver enkelt farveovergang samt argumenter for og imod dem er beskrevet i tabel 4.1. Alle har til fælles at farveovergangen er skarp - farven skifter fra lys til mørk med højst en enkelt pixel mellemtone imellem sig.

De to bedste farveovergange ses at være C og D. Deres fordele er:

- Der er rig mulighed for redundante målinger af disse typer farveovergange.
- Tilskuere står kun sjældent i vejen.

A	Eksisterende klistermærke i hjørnet	Tilskuere står tit med hænderne på klistermærkerne.
B	Siden af målet	Målmændene kan skygge for farveovergangene og desuden afhænger farven i målet meget af lysforholdene, som det også kan ses på billedet.
C	Linjer på banen	Bolden og spillerne kan skygge for farveovergangene. Der er til gengæld mange af disse farveovergange, så målingen kan gøres meget redundant.
D	Overgang fra hvid side til rød kant	Spillerne kan være lidt i vejen og besværliggøre målingen. Til gengæld eksisterer mange af disse farveovergange på bordet, så der kan laves redundante målinger.
E	Overgang fra grøn bane til hvid side	Argumenterne er som i D, men med den ulempe at antallet af disse farveovergange er mindre, så der ikke kan laves så meget redundans. Desuden kan også bolden være i vejen.
F	Overgang fra hvid side til sorte skiver	Argumenterne er også her som i D, men der kan ikke måles på et så bredt område som i D, da den hvid-sorte overgang til skiven er meget smallere.

Tabel 4.1: Kort beskrivelse af hver farveovergang i figur 4.16 samt fordele og ulemper ved disse.

- Farverne der skiftes fra og til er ikke meget afhængige af lysforholdene. Kun ved meget stærk belysning bliver farveovergangen svær at se.
- Farveovergangene er meget brede. Det reducerer risikoen for fejlmålinger skyldet støj, smuds og objekter som er i vejen.

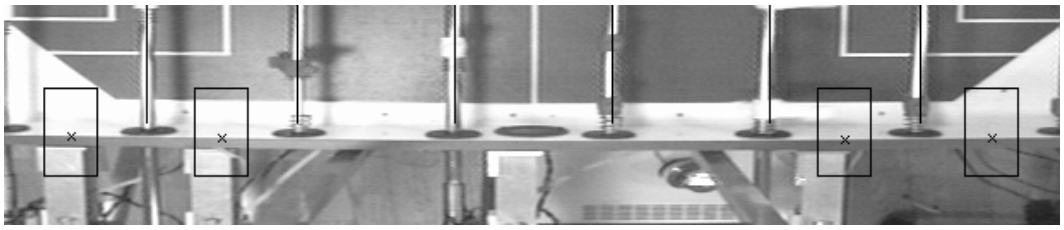
Bolden kan genere målinger af C, men ikke af D. Derfor vælger vi at benytte den hvid-røde farveovergang fra bordets side til bordets røde kant til kalibreringen.

4.4.3.3 Kompenseringsalgoritme

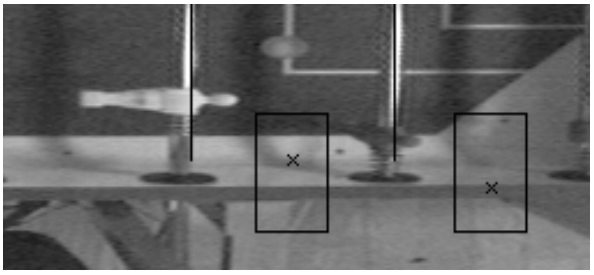
For at finde en enkelt farveovergang undersøges et rektangel, som er defineret i forhold til bordets position og den kendes jo som udgangspunkt pga. startkalibreringen. Rektanglet søges igennem fra top til bund indtil farveovergangen er fundet.

Dette gøres for fire rektangler og de kan ses på figur 4.17. Hvert rektangel er optegnet med sort og i hver af disse er tegnet et kryds, hvor farveovergangen er fundet. Positionen trækkes fra det måleresultat, der blev fundet første gang kalibreringen kørte. Dermed haves for hver farveovergang et mål for, hvor meget billedet har rykket sig.

Derefter frasorterer en algoritme `bestYOffset` konstrueret i [3] fejlmålinger, ved at sammenligne disse tal for forskubbelsen. På baggrund af antallet af tal der ligner hinanden ses det, hvilke resultater der skal bruges. Dermed frasorteres fejlmålinger, som f.eks. kan skyldes en skygge eller en spiller. Fejlmålinger kan let forekomme, som det f.eks. ses under ugunstige lysforhold på figur 4.18.



Figur 4.17: Kalibrering i y-retningen foregår med de farveovergange, der er indrammet med kasser. Hvert kryds i en kasse angiver, hvor algoritmen har fundet farveovergangen. Algoritmen stopper, når den når kanten og det er derfor ligegyldigt, om der ligger mærkeligt farvede ting på gulvet.

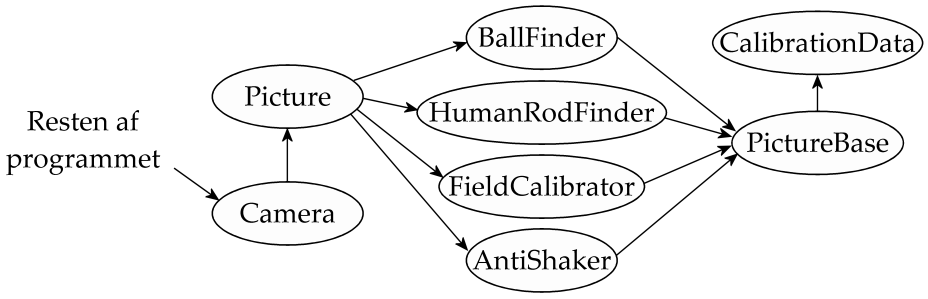


Figur 4.18: Et lyst område over en næsten vandret skygge er skyld i en fejlmåling.

Af de tal der benyttes, tages et gennemsnit og dette siger, hvor meget billedet har rykket sig i forhold til udgangspositionen. Det benyttes til at kompensere for kameraets rystelser. Nærmere detaljer om algoritmen er i bilag C.3.4.

4.5 Implementering

Koden til billedgenkendelsen er opdelt i en række klasser, der beskrives her. Overordnet benytter klasserne hinanden som vist på figur 4.19. Programmet beder `Camera` om et billede og den returnerer da et `Picture` objekt. På dette kan man f.eks. bede om boldpositionen og `Picture` objektet spørger da `BallFinder`. Den gemmensøger billedet, som den har adgang til vha. `PictureBase` og kalibreringsdataene i `CalibrationData`.



Figur 4.19: Klassediagram for den del af programmet, der har med billedet at gøre. Pilene viser, hvilke klasser der bruger hinanden.

Camera sørger for at hente det rå billede fra kameraet og holder styr på, hvilket `Picture` objekt, der skal benyttes. Den grundlæggende funktionalitet er stort set uændret siden [1].

Picture klassen benyttes til at tilgå de funktioner der har at gøre med billedet. Først og fremmest wrapper den funktionalitet fra de klasser, den benytter, nemlig de fire næste:

BallFinder finder bolden på billedet. Algoritmen i 4.2.2 benyttes.

HumanRodFinder lokaliserer modstanderens stænger på billedet som beskrevet i 4.3.

FieldCalibrator kaliberer billedet ved spillets start, ved brug af algoritmen beskrevet i 4.4.2.2.

AntiShaker foretager den kalibrering der skal til, for at kompensere for rystelser som beskrevet i 4.4.3.

CalibrationData indeholder data om, hvor spillefladens hjørner og stængerne er på billedet. Desuden gemmes også de nødvendige information om farveovergangenes placeringer til brug af `AntiShaker`.

PictureBase indeholder `CalibrationData` og funktionalitet til at tilgå de enkelte pixels på billedet. Desuden har denne klasse grundlæggende funktionalitet til at tegne på billederne og gemme disse på harddisken, hvilket mest benyttes til fejlsøgning.

4.6 Test

4.6.1 Genkendelse af bolden

De rå data for følgende tests kan ses i bilag C.4.

4.6.1.1 Stilleliggende bold

Bolden placeres i 6 yderpositioner på bordet. Position 1-4 er de fire hjørner og position 5 og 6 er midt på i x-retningen, ude ved de to kanter i y-retningen. Målepunkterne er altså arrangeret ligesom hullerne på et billardbord. I hver position genkendes positionen 5 gange og der tages et gennemsnit af hvert sæt målinger. Positionen måles også op manuelt og sættene af målinger sammenlignes. Fejlen på de genkendte positioner er i mm:

Position	1	2	3	4	5	6
x	-1.9	-1.1	2.6	6.2	-3.1	-2.9
y	1.5	-5.4	-0.9	1.9	-0.5	-4.9

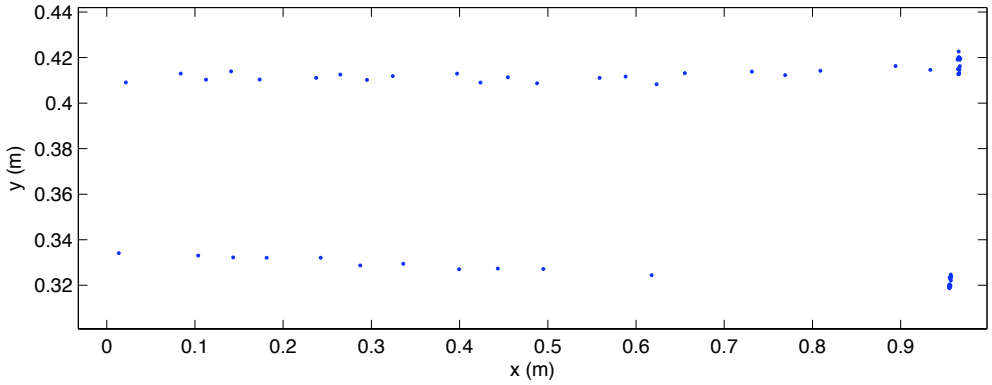
Altså genkendes bolden maksimalt 6 mm fra dens rigtige position. I enkelte frames kan fejlen dog være større pga. støj og rystelser. I langt de fleste tilfælde er fejlen dog langt mindre,

da de andre målte fejl dels er mindre og da målingerne er foretaget i bordets yderpositioner, hvor fejlene må antages at være størst.

4.6.1.2 Bold i bevægelse

Det testes om bolden kan genkendes når den bevæger sig. Vi placerer den foran den røde forsvarsmand og skyder til den. Det gentages ved forskellige hastigheder og ud af graferne for dens genkendte positioner på figur 4.20 ses det, hvornår bolden ikke bliver genkendt. På det øverste skud i figuren genkendes bolden hele vejen fra højre mod venstre, kun afbrudt hvor stængerne er i vejen. De 6 steder, hvor bolden mangler på grafen for det skud, passer fint med at der er 6 stænger imellem det sted, hvor skuddet startede og målet i den anden ende.

På det nederste bevæger bolden sig hurtigere og endda for hurtigt i starten, til at den kan genkendes. Længere mod venstre er farten aftaget tilstrækkeligt, til at boldens position bliver genkendt 9 gange i træk. Ud af de rå data i bilag C.4 ses det at stængerne ikke var i vejen denne gang. Her måles farten til 2.4 m/s. Altså genkendes bolden, når farten højst er 2.4 m/s. Da genkendingsalgoritmen i princippet ikke gør forskel på lodret og vandret, antages det at denne grænse ikke er retningsbestemt.



Figur 4.20: Bolden skydes to gange fra højre mod venstre. Nederst er skuddet i starten for hurtigt til at positionerne kan genkendes.

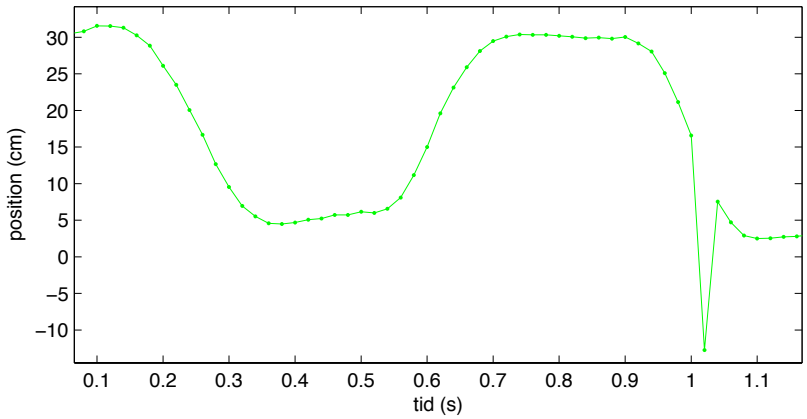
4.6.2 Genkendelse af mændene

Alle modspillerens stænger blev hevet til positionen 0 på y-aksen, dvs. så langt de kunne komme til den ene side. Her blev de genkendt af algoritmen 20 gange og vi fandt fejlen i forhold til deres rigtige position, 0. I rækkefølgen målmand, forsvar, midtbane og angreb var fejlene 3.6, −5.2, −2.9 og −3.5 mm. Stængerne blev derefter hevet så langt de kunne den anden vej, deres rigtige positioner blev målt og de blev genkendt af algoritmen. Her var fejlene 4.2, −0.1, 1.1 og −2.2 mm. I disse yderpositioner må den største fejl forekomme, så vi kan konkludere at den største fejl i genkendelsen af modspillerens positioner er 5 mm. Ved helt præcis justering af kameraets rotation, kan denne usikkerhed dog reduceres. Standardafvigelsen for

de 8 fundne positioner lå mellem 0.9 mm og 1.3 mm og gennemsnittet af standardafvigelserne var 1.1 mm.

Man kunne også teste hvilken indflydelse mændenes rotation har på genkendelsen af deres position. Vi har dog set igennem brugergrænsefladen, at effekten er lille, så det undersøger vi ikke nærmere.

Det testes at manden kan genkendes mens den er i bevægelse. Det gøres ved at hive forsvarsstangen frem og tilbage nogle gange med varierende hastighed. Det viser sig, at der skal høj fart til, før genkendelsen går galt. På figur 4.21 ses stangens genkendte position over tid. På den opadgåene del af grafen omkring $t = 0.6$ s, er hastigheden 1.8 m/s over 6 frames. Der genkendes manden ved alle frames, og positionerne ser fornuftige ud. På den nedadgående del af grafen omkring $t = 1.0$ s går genkendelsen galt i en enkelt frame. Her er hastigheden målt til 1.9 m/s over 7 frames. Altså genkendes mændene ved hastigheder under 1.9 m/s.



Figur 4.21: Forsvarsstangen hives hurtigt frem og tilbage. Omkring $t = 1.02$s er hastigheden for høj, til at stangens position kan genkendes.

De rå data for begge deltests kan ses i C.5.

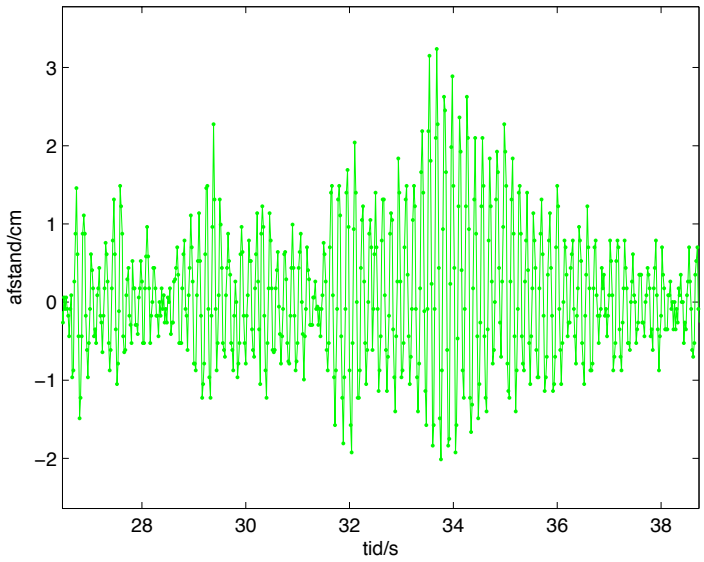
4.6.3 Kalibrering ved opstart

Kalibreringsalgoritmen er testet i C.6 og der viser det sig, at den bestemmer spillefladens position korrekt og er meget robust overfor varierende lysforhold.

4.6.4 Kompensering for kamerarystelser

Under normalt spil viser det sig, at billedet nemt hopper 2-3 cm i forhold til udgangspositionen som det ses på figur 4.22, der viser kompenseringen for rystelserne. I de følgende tests benyttes derfor kameraudsving af samme størrelse.

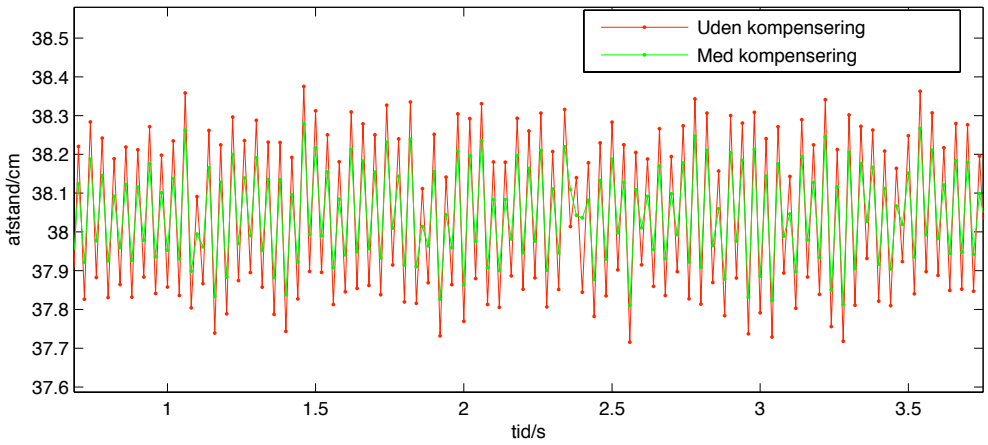
For at vise, at kompenseringen er korrekt, laves følgende forsøgsopstilling: En bold ligger



Figur 4.22: Under normalt spil mod robotten ryster kameraet flere cm til hver side. Rystelsernes størrelse på billedet er vist på andenaksen.

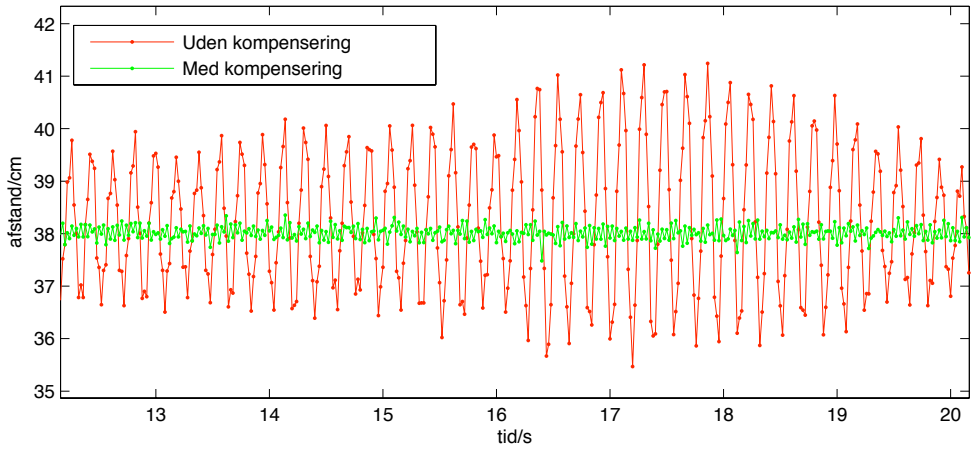
stille på bordet uden der spilles. Den genkendte position skulle gerne være konstant både med og uden kompensering. Kameraet rystes derefter med hånden. Boldens position uden kompensering burde variere meget og med kompensering burde den stadig være konstant. For at rystelserne ikke skal forplante sig til bordet og få bolden til at rykke sig i forhold til spillefladen, er bolden tapet fast til bordet.

Resultatet af testen, hvor bolden ligger stille uden rystelser af kameraet, ses på figur 4.23. Både med og uden kompensering svinger de genkendte boldpositioner 2-3 mm til hver side, hvilket er så lidt at det må siges at være konstant. Det er som ventet.



Figur 4.23: Den genkendte boldposition for en bold der ligger stille og et kamera der ikke ryster.

Resultatet af testen, hvor bolden ligger stille, men hvor kameraet rystes ses på figur 4.24. Uden kompensering svinger den genkendte boldposition 2-3 cm til hver side, ligesom det skete under normalt spil. Med kompensering svinger positionen kun 2-3 mm som da kameraet ikke rystede. Bemærk at de to grafer ikke er skaleret ens.



Figur 4.24: Den genkendte boldposition for en bold der ligger stille og et kamera der ryster. Bemærk at enhederne er forskellige fra dem i figur 4.23.

Det konkluderes, at softwaremæssig kompensering for kamerarystelser fungerer fint og at en ændring af stativet ikke er nødvendig. Dette holdes i skarp kontrast med [1], som konkluderede at *"stativet ikke er stift nok [...] Kamerastativet skal derfor ændres for at komme til at virke optimalt"*. En kompensering i softwaren med det nuværende stativ må siges at være noget mere elegant end en løsning med afstivere til den modsatte side af bordet som [1] foreslog.

4.7 Konklusion

Den eksisterende boldgenkendingsalgoritme er udvidet til at tage højde for parallaksefor-skydning og finjusterer positionen vha. boldens massemidtpunkt. Boldens position genkendes indenfor maksimalt 6 mm plus støj, uanset hvor den er på bordet. Den genkendes ved hastig-heder under 2.4 m/s.

Modspillerens stængers positioner på sidelænsbevægelsen bliver nu fundet af programmet. De genkendes indenfor med fejl på maksimalt 5 mm. Ved hastigheder under 1.9 m/s genkendes de korrekt.

Billedet bliver nu kalibreret i x- og y-retningen ved programmets opstart vha. klistermærker sat på bordet. Derfor behøver man ikke længere foretage manuelle kalibreringer og rette i programmet, hvis der skubbes lidt til kameraet.

Desuden kompenseres softwaremæssigt for kameraets rystelser, så en fysisk ændring eller afstivning af kamerastativet ikke er nødvendig. Boldens genkendte position afviger nu typisk kun 2-3 mm fra den gennemsnitlige, selvom stativet ryster voldsomt.

Kapitel 5

AI

Vi har i det foregående beskrevet, hvordan robotten får informationer om, hvordan spillet ser ud og hvordan robotten har mulighed for at bevæge sig. Disse to ting kædes sammen af en kunstig intelligens, eller AI, der i så vid udstrækning som muligt skal få robotten til at reagere fornuftigt på de informationer, den får.

5.1 Analyse af AI design

5.1.1 Muligheder vi har overvejet

Der er mange muligheder for opbygning af en kunstig intelligens til bordfodboldrobotten. De mest oplagte beskrives herunder. Når der refereres til en tilstand, menes der boldens position og hastighed, samt positionerne af robottens og modspillerens stænger.

5.1.1.1 Søgetræer med minimax

Til enhver tilstand kan minimax-algoritmen [10] i princippet udregne, hvordan stængerne flyttes, for at spille optimalt. Algoritmen tænker fremad i en diskretiseret tid og udregner for hvert tidsskridt alle muligheder for hvad robotten selv og modstanderen kan foretage sig. Rekursivt udregnes alle muligheder igen for det næste tidsskridt. Dette gentages i princippet helt frem til alle forgreninger slutter (ved scoring eller gentagelse af en tidligere position). Dette fører til meget store forgreninger, ligesom i skak, hvor det er umuligt at regne spillet til ende. Vha. gode heuristikker og teknikker såsom alpha-beta pruning [11] kan forgreningerne dog reduceres en del og gør man ikke behøver regne spillet til ende.

5.1.1.2 Udgangspunkt i hvordan vi selv spiller

AI'en kunne tage udgangspunkt i hvordan vi selv spiller bordfodbold. Den defensive del af spillet, altså når man dækker op for bolden, kan da programmeres reaktivt, dvs. uden at computeren gemmer tilstandsinformationer over tid. Ud fra spillets tilstand kan det afgøres hvordan stængerne skal flyttes, for at dække optimalt. I angrebet kan det nok være en fordel at gemme informationer mellem tilstande, således at computeren f.eks. ved, hvilken angrebsfinde den er ved at udføre. Der er altså lagt op til fuldstændigt hardcodede taktikker, dog med mulighed for at udvide med statistik som beskrevet i 5.2.2.

5.1.1.3 Neuralt netværk

Enten som heuristik i forbindelse med et minimax-træ, eller direkte til at styre spillerne, kunne man benytte et neuralt netværk [12]. Metoden er inspireret af, hvordan en hjerne fungerer, dvs. at en række neuroner får noget input eller stimuli om man vil, som baserer sig på spillets tilstand. På baggrund af nogle vægte bliver disse input sendt videre til andre neuroner og i sidste ende får man nogle tal som output.

Disse output kan f.eks. direkte angive, hvor stængerne skal flyttes hen i den givne situation. Man kan også lave netværket, så der kun er et enkelt output og dette siger angiver hvor god tilstanden er. Netværket kan da generere heuristik til brug i en minimax-algoritme.

5.1.1.4 Andre muligheder for maskinel indlæring

Der findes mange andre muligheder for maskinel indlæring [6], men vi har ingen erfaring med dem og kender ikke de principper de bygger på.

5.1.1.5 Hybrid løsning

Man kunne forestille sig hybrider som f.eks. en reaktiv AI i forsvaret og et neuralt netværk i angrebet.

5.1.2 Hvorfor vi valgte at tage udgangspunkt i hvordan vi selv spiller

Der er fordele og ulemper ved alle metoderne, som vi sammenligner her.

5.1.2.1 Minimax

Det vil det være nødvendigt at køre algoritmen igennem hver eneste gang spillets tilstand ændrer sig og det skønner vi vil tage lang tid. For at kunne forudsige spillets gang, kræves der desuden en rigtig god fysisk model, hvilket kan være svært at lave.

5.1.2.2 Udgangspunkt i hvordan vi selv spiller

Denne metode har to store fordele: det er let at programmere og vi er sikre på at få robotten til at spille godt nok til at give modstand mod en nybegynder.

I denne løsningsmodel kan vi nok opnå det samme resultat som med minimax-algoritmen. Vi spiller næppe optimalt, men vi spiller godt [5] i kraft af på forhånd at have kigget frem i tiden og analyseret hvordan vi bedst stiller spillerne. Formålet med minimax-algoritmen er netop at kigge frem i tiden og minimax er derfor unødvendig, hvis robotten sættes til at spille som vi selv gør.

5.1.2.3 Neuralt netværk

En fordel ved denne fremgangsmåde er, at robotten selv kan lære at spille og dermed i teorien kan blive bedre, end hvis hele intelligensen er fast defineret på forhånd. Det har dog også en del ulemper:

- For at blive godt, kræver et neuralt netværk langt mere optræning end vi har tid til at give det. Dog kunne man i stedet træne det vha. simulationer med forudprogrammerede tricks.
- Det er svært at debugge et neuralt netværk, hvis det ikke gør som vi forventer.
- Vi har ingen erfaring med neurale netværk.

Disse punkter gør, at det er usikkert om vi overhovedet vil få et neuralt netværk til at fungere så godt at det vil give nogen modstand.

5.1.2.4 Andre muligheder for maskinel læring

Alternativet beskrevet i 5.1.1.2 virker udmærket, så vi har valgt ikke at bruge tid på at undersøge muligheder for maskinel læring.

5.1.2.5 Hybrid løsning

Det kunne måske have nogle fordele at benytte forskellige algoritmer afhængigt af situationen, men vil kræve dobbelt så meget tid at lave, så det overvejer vi ikke nærmere.

5.1.3 Vores valg

Ud af disse muligheder virker udgangspunktet i, hvordan vi selv spiller som den bedste mulighed. Dens største ulemper er dog, at den ikke selv kan lære nye finder og at hvis modstanderen har fundet en måde at snyde den på, så kan han blive ved med det. Det sidste kan man dog til dels komme udenom vha. statistik på, hvordan modstanderen spiller.

Disse ulemper opvejes rigeligt af, at der er god sandsynlighed for at vi har et produkt, der fungerer ved projektets afslutning og at det er nemt at implementere.

5.2 Design af spillealgoritme

5.2.1 At finde ud af, hvilken mode man er i

Afhængigt af spillets tilstand spiller vi på forskellige måder. Disse måder kalder vi *modes*. Groft set kan man dele spillet op så man spiller offensivt når man selv har kontrol over bolden og defensivt i alle andre tilfælde. Spillemåden kan deles yderligere op idet vi dækker op på forskellige måder afhængigt af

- hvilken af modspillerens stænger bolden er ved
- hvilken af ens egne stænger der har kontrol over bolden
- hvilken hastighed bolden har

Desuden angribes der på forskellig vis afhængigt af, hvilken af ens egne stænger der har bolden.

Hver gang spillets tilstand ændrer sig, bør den kunstige intelligens finde ud af, hvilken af alle disse modes der skal benyttes. Derefter kan spillelogikken for den pågældende mode flytte spillerne.

5.2.2 Statistik

Forsvaret laves reaktivt fordi der til enhver tilstand er en måde at stille spillerne på, som giver stor chance for at forhindre et mål. Dette vil typisk være at stille spillerne i en lige linje

mellem bolden og målet. Den strategi fungerer godt mod nybegyndere fordi de normalt blot skyder til bolden så snart de får den. De prøver altså ikke at trække bolden til siden først eller bruge bander til at ramme målet.

Hvis bordet skal spille godt mod øvede spillere, vil det være en fordel undervejs i spillet at opsamle statistiske oplysninger om, hvilke måder modspilleren skyder på. Dvs. hvilke sammenhørende skudvinkler og skudpositioner modstanderen typisk gør brug af, om modspilleren trækker bolden til siden inden han skyder, og i givet fald hvor langt og om modspilleren benytter bander til at ramme målet.

Statistikkens nyttighed bygger på at øvede spillere kan benytte visse skudfinder, men ikke mange forskellige. Oftest bruger han dem han kan og de andre finder forsøger han end ikke at bruge. Det giver sig typisk udslag i at han næsten hver gang trækker bolden den ene vej inden han skyder til den, men aldrig den anden vej. Da kan den opdækkende spiller med fordel stille sine spillere lidt længere til omtalte side, således at modspilleren får sværere ved at trække bolden forbi. Det kan også være, at modspilleren, når han har bolden og sin spiller i en given position, næsten altid rammer samme sted i målet. Så kan man med fordel stille målmanden det sted hvor han plejer at ramme, allerede inden man ser at bolden er ud for det sted eller har retning mod det.

Da robotten først og fremmest skal spille godt mod nybegyndere, vælger vi at udelade statistik fra dens kunstige intelligens. Men det er en oplagt udvidelsesmulighed.

5.2.3 At finde boldens hastighed

Forudsat at systemet indenfor en vis nøjagtighed kan finde boldens position til en række tidspunkter, vil man kunne finde et estimat for boldens hastighed. Ved at antage, at bolden har bevæget sig i en ret linje det sidste korte stykke tid, kan man benytte en af flere metoder til at finde hastigheden.

Den simpleste metode er at benytte det rette linjestykke mellem de to sidst fundne boldpositioner. Det divideres med længden af tidsrummet imellem disse, for at få hastigheden. Metoden har den fordel, at en ændring i boldens retning med det samme vil føre til en ændring i den opfattede hastighed. Derfor kan robotten hurtigt reagere på ændringer i boldens hastighed. Til gengæld er metoden meget følsom overfor støj, hvilket er svært at undgå, uanset hvordan vi måler boldens position.

Alternativt kan man benytte mindste kvadraters metode til at finde hastigheden over længere tid. Denne metode er god til at udjævne støj, men for at gøre det, er det til gengæld nødvendigt at bruge flere end 2 boldpositioner til at bestemme hastigheden.

5.2.4 At forudse boldens position

Det er nødvendigt at forudse boldens position frem i tiden. Vigtigst må det være, at man kan se nok fremad til at kunne nå at flytte mændene ind foran bolden, når den kommer. Dels

for at skyde til den og dels for at forhindre at modstanderen scorer et mål. Relevant er det også, at kunne forudse boldens position i forbindelse med sammenspil og angrebsfinder, men mindre væsentligt, idet man godt kan score mange heldige mål mod nybegyndere, men det er svært at lave heldig opdækning.

Til at forudse fremtidige boldpositioner benyttes en lineær fremskrivning af positionsvektoren mht. hastighedsvektoren. Lad boldens positionsvektor være $\vec{p_0}$ og dens hastighedsvektor være $\vec{v}$. Da finder vi boldpositionen $\vec{p_1}$, som hører til tiden Δt efter nuværende tidspunkt, ved

$$\vec{p_1} = \vec{p_0} \cdot \Delta t$$

Metoden er let at programmere, men tager ikke højde for boldens opbremsning pga. friktion. Den tager heller ikke højde for at bolden kan ramme bordets sider og mændene eller at dens bane kan blive afbøjet pga. de skrå kanter der er på bordet. Hvis robotten skulle spille lige så godt som mennesker, skulle den også tage højde for disse forhold, for det gør mennesker - f.eks. kan vi godt regne ud at bolden kommer ned igen, når den ruller op ad en skrå side og vi har derved mere tid til at stille mændene rigtigt, end hvis vi brugte den beskrevne metode. Dog er vores metode god nok:

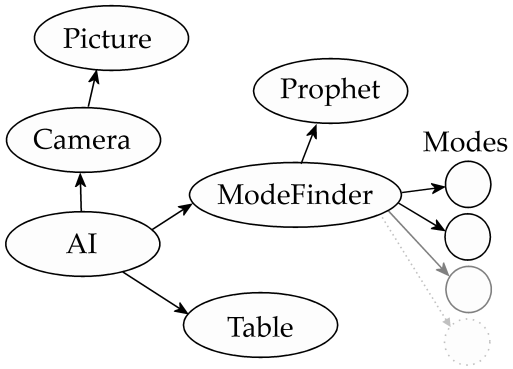
- Begyndere bruger sjældent banderne bevidst i deres spil.
- Når bolden begynder at rulle ned fra et skråt hjørne, går der normalt over et sekund inden bolden når ned til forsvaret, hvilket er rigeligt til at stille sig i vejen og gribe den.
- At beregne boldens udgangsvinkel ved kollision med en mand, præcist nok til at det overhovedet giver mening at gøre det, kræver stor præcision i genkendelsen af boldens og mandens hastigheder og positioner. Boldens genkendte position kan afvige helt op til 6 mm som fundet i 4.6.1.1, hvortil skal lægges støj og afvigelse i de genkendte mænds positioner, og det er næppe præcist nok.

5.3 Implementering

Vi har implementeret en `AI` klasse, som håndterer overordnede del af den kunstige intelligens, beskrevet ved denne pseudokode:

```
for hver frame
   vent på et nyt billede (Picture) fra kameraet (Camera)
   genkend bolden
   genkend modspillerens mænd
   find den mode der skal benyttes
   aktivér denne
loop
```

Der findes en `ModeFinder` klasse til at finde den mode robotten bør være i. Vi har lavet en række `Mode` klasser som flytter stængerne afhængigt af de nærmere omstændigheder ved situationen (hvilken af mændene der kan nå bolden, hvilke vinkler der bør dækkes for, osv.) `Table` klassen sørger for den videre kommunikation med stængerne som beskrevet i [3]. Et simpelt klassediagram over sammenhængen mellem klasserne er vist på figur 5.1. `Prophet` klassen finder boldens hastighed ud fra positioner som beskrevet i 5.3.3.



Figur 5.1: Et simpelt klassediagram for den del af programmet der vedrører den kunstige intelligens. Pilene angiver, hvilke klasser der benytter hinanden.

5.3.1 ModeFinder

Vi har implementeret en `ModeFinder` klasse som beskrevet i 5.2.1. Ved først og fremmest at kigge på boldens position og hastighed, vælger denne klasse hvilken mode AI'en er i. Modefinderen returnerer derefter det tilsvarende `Mode` objekt. I pseudokode foregår det således:

```
hvis en stang er blokeret
   brug SemiResetter mode
ellers hvis bolden har været væk i et stykke tid
   brug NotFound mode
ellers
   hvis bolden er bag egen midtbane
      hvis der er kontrol over bolden og den er ved forsvaret
         brug DefenseAttack mode
      ellers
         brug DefenseDefend mode
   ellers hvis bolden er ved midtbanen
      hvis bolden er under kontrol og ved egen midtbanespiller
         brug CenterAttack mode
      ellers
         brug CenterDefense mode
   ellers // så er den i angrebet eller ved modspillerens forsvar
      hvis bolden er under kontrol og ved egen angrebsspiller
         brug OffenseAttack mode
      ellers
```

```
brug OffenseDefend mode
```

Det er en meget simpel struktur og C++ koden til dette er næsten lige så simpel.

5.3.2 Modes

Der er grundlæggende seks forskellige modes: et par for forsvaret, et par for midtbanen og et par for angrebet. For hvert af disse områder er der en angrebsmode til når ens egne mænd har bolden under kontrol og en forsvarsmode til de resterende situationer. At have bolden under kontrol defineres af os som at den er tæt på ens egen stang og bevæger sig langsomt.

Når bolden blokerer en stang, benyttes en `SemiResetter` mode og når bolden ikke er blevet genkendt i løbet af de sidste 1-2 sekunder har vi en `NotFound` mode.

5.3.2.1 DefenseAttack og CenterAttack

Disse modes er de simpleste og håndterer at bolden er under kontrol ved hhv. forsvaret eller midtbanen. I `DefenseAttack` løftes angrebet og midtbanen, så de ikke kan være i vejen for bolden. Derefter findes ud af, hvilken mand der skal sparke til bolden. Manden bevæges rundt om bolden, hvis denne ligger bag manden, hvorefter der sparkes til den. I `CenterAttack` sker det samme, blot løftes kun angrebet for ikke at stå i vejen for bolden.

I `DefenseAttack` undersøges det, om modstanderen har en spiller lige foran bolden. Hvis det er tilfældet, rykkes forsvarsmanden lidt til siden inden der sparkes, således at bolden skydes skråt forbi modspillerens mand. Hvis man skyder bolden lige ud, kan modspilleren ellers let gribe bolden.

5.3.2.2 OffenseAttack

Denne mode håndterer, at bolden er under kontrol ved angrebet. Ligesom ved `DefenseAttack` og `CenterAttack` skydes til bolden med den nærmeste mand, men her sker det kun, hvis bolden er ca. ud for målet. I de andre tilfælde prøves det at føre bolden ind foran målet, ved at flytte en af mændene rundt om bolden og puffe den til siden.

5.3.2.3 DefenseDefend

Når forsvaret dækker op for modstanderen, bruges denne mode. Den er den mest komplicerede af dem alle. Den opdeler situationen i mange mindre situationer, afhængigt af boldens position og hastighed:

Når bolden bevæger sig langsomt ved angrebsstangen, stiller forsvaret og målmanden sig ud for bolden.

Når bolden bevæger sig hurtigt mod målet, stiller målmanden og forsvaret sig der, hvor bolden vil være, når den når til deres stænger. Den ser altså frem i tiden.

Disse første to tilfælde er samlet i `DefenseBasic`, og benyttes af forsvaret i alle Defense modes. De er mere komplicerede og beskrives derfor nærmere i bilag D.1.1.

Når bolden er i nærheden af målmanden, skyder han til den på samme måde som beskrevet i 5.3.2.1. Dvs. at målmanden går rundt om bolden, hvis den ligger bag ham og skyder den derfor ikke ind i målet.

Når bolden har retning væk fra baglinjen, f.eks. efter at modstanderen har ramt forbi målet, og banden på baglinjen får bolden til at ryge tilbage, tjekkes det om forsvaret kan nå at stille sig i vejen for bolden, inden den ryger væk. Hvis den kan, gør den det og undgår dermed at modstanderen får fat i bolden. Herefter vil bolden ligge stille ved forsvarsstangen og lidt senere vil `DefenseAttack` blive brugt til at skyde til bolden.

Generelt skal det siges om forsvarsstangen, hvorpå der sidder to mænd, at der opdækkes med den samme mand hele tiden, nemlig den tættest på robotten selv. Derved undgås det hul, der ellers vil opstå når der skiftes til at dække med den anden mand. Detaljer om, hvordan det kan lade sig gøre findes i bilag D.1.1.

5.3.2.4 CenterDefend

Midtbanen forsvarer mod bolden ved, hvis den har retning mod midtbanestangen, at stille sig der hvor bolden vil krydse stangen. Ellers stiller den sig blot ud for bolden.

5.3.2.5 OffenseDefend

Ligesom i `DefenseDefend` stiller angrebsmændene sig der hvor boldens vej vil krydse stangen, når bolden ellers er på vej mod denne med god fart på. Ellers stiller den sig blot i en lige linje ud for bolden. Mændene på de andre stænger prøver også at dække målet samtidigt.

5.3.2.6 NotFound

Hvis bolden ikke er blevet set i de sidste 1-2 sekunder, benyttes denne mode. Hvis bolden sidst blev set tæt hos modspilleren, antages det at den stadig er der og at han blot skjuler den. Da gøres intet og AI'en venter blot på at bolden skal komme til syne igen. Hvis den derimod sidst blev set nær ens egen stang, flyttes den pågældene stang til siden og bolden vil da muligvis komme til syne, og en anden mode vil tages i brug. Den prøver kun dette et

kort stykke tid, hvorefter den står stille, for så er bolden sandsynligvis ude af spillet og det vil ikke gavne, at stangen kører frem og tilbage uden bolden er på bordet.

5.3.2.7 SemiResetter

Hvis en stang blokeres, aktiveres denne mode. Hvis den sidder fast på rotationsaksen, ligger bolden sandsynligvis under manden. Da vil manden rotere hele vejen rundt og dermed skyde til bolden, hvis bolden ligger foran stangen. Hvis bolden ligger bagved, flyttes stangen blot til siden, hvorved bolden sandsynligvis vil komme i spil igen. Hvis en motor-controller, som styrer sidelænsbevægelsen er i semi-reset mode, rykkes stangen blot den modsatte vej og spillet kan fortsætte.

5.3.3 At finde boldens position og hastighed

En algoritme til at beregne mindste kvadraters metode er implementeret i `Prophet` klassen. Vi vælger at benytte de sidste 4 boldpositioner til at foretage den lineære regression mht. x-positionen og tiden. Det tilsvarende gøres for y-positionen og tiden.

Ud fra 2 hensyn valgte vi, at programmet skulle huske 4 boldpositioner:

- Det er et mål, at der benyttes så få positioner som muligt, så ændringer i hastigheden opfattes så hurtigt som muligt.
- Der skal huskes nok positioner til at der ikke er så meget støj på den opfattede hastighed, at spillerne flyttes på en uhensigtsmæssig måde.

Vi valgte dette tal ud fra startgæt ved at køre programmet fire gange, hvor det huskede 2, 3, 4 og 5 positioner. For 2 og 3 bevægede spillerne sig ret vildt rundt som reaktion på store opfattede boldhastigheder. Her var der altså for meget støj i forhold til hvad AI'en kunne håndtere. Ved 4 og 5 huskede positioner var der meget lidt forskel på den måde robotten spillede, men det så meget mere kontrolleret ud end ved 2 og 3. 4 boldpositioner var altså det mindste antal der gav så lidt støj på de opfattede hastigheder, at robotten opførte sig fornuftigt.

Resultatet afhænger selvfølgelig af, hvordan resten af AI'en er programmeret - den kunne godt gøres mere robust over for ustabile boldhastigheder. Desuden vil en boldgenkendelsesalgoritme, som er mere robust overfor støj selvfølgelig gøre udsvingene i de opfattede boldhastigheder mindre og dermed mindske behovet for et højt antal huskede boldpositioner.

5.4 Test

Overordnet set fungerer det hele. AI'en stopper skud, og scorer mål. Dog er der visse detaljer vi har set fungerer særligt godt:

- Robotten får meget hurtigt overblik over situationen. Ofte overrasker den ved at skyde til bolden så snart den er i nærheden af en mand. Specielt på midtbanen er den rigtigt hurtig til at finde den mand der skal skyde til bolden, en ting mennesker tit er dårlige til, da der her er mange mænd og de hver kun kan dække et lille område.
- Når bolden er på vej ude fra siden mod målet, skal målmanden stå helt i yderpositionen og indstille sin vinkel rigtigt, for at redde bolden, da denne ellers vil trille foran eller bag målmanden og give mål til modstanderen. Målmanden redder bolden næsten hver gang i disse situationer, selvom det kræver præcision og bolden let kan være dækket af målmandens stang i det område.

Under udviklingen af AI'en er de enkelte detaljer i modes og modefinderen selvfølgelig blevet afprøvet løbende, og de ser ud til at fungere. Dog har vi ikke foretaget systematiske tests af disse. I stedet har robotten spillet en lang række kampe mod forskellige modstandere og vundet 73 % af dem. Det må derfor siges, at AI'en fungerer tilfredsstillende. En resultatliste over disse kampe findes i F.

5.5 Konklusion

Vi har implementeret en AI til at styre bordfodboldrobotten. Den er baseret på, hvordan vi selv spiller og deler derfor spillet op i forskellige tilstande - modes, afhængigt af boldens position og hastighed. I de defensive modes er AI'en reaktiv, dvs. at den kun reagerer på spillets tilstand og prøver at stille sig foran bolden. I de offensive modes har AI'en yderligere et par tilstandsvariable, så den f.eks. kan huske det, hvis den er igang med at sparke til bolden.

Der benyttes mindste kvadraters metode til at finde boldens hastighed ud fra de forudgående positioner. Boldens fremtidige positioner forudses vha. en lineær fremskrivning.

AI'en fungerer rigtigt godt, da den har vundet næsten 3/4 af sine kampe.

Kapitel 6

Systemtest

6.1 Simulator

Da vi er to om at udvikle robotten, er vi ofte færdige med noget implementering og klar til test, samtidigt. Da vi kun har én robot, kan der let blive rift om robotten. For at undgå denne ressurseknaphed, skrives en simulator, der kan agere bordfodboldbord overfor vores computer. Den skal altså erstatte kameraets og encodernes funktion som virkelighedsaflæser og give realistisk respons på de handlinger AI'en udfører. Herved kan AI'ens opførsel aflæses kun ved brug af computeren.

Denne simulator vil også være uundværlig, skulle en større del af hardwaren ikke fungere i en periode. Specielt ved projektets start var vi ikke sikre på, hvor godt hardwaren ville fungere og hvor driftsikker vi kunne gøre den. Med en simulator kunne vi derfor undersøge om AI'en virkede, uafhængigt af hardwaren.

Såfremt den fysiske model bag simulatoren kan efterligne virkeligheden tilstrækkeligt præcist, vil der være visse fordele ved at benytte denne frem for tests udenfor simulatoren. Eksempelvis kan det være nemmere at teste spilstrategier i denne, da den muliggører fuldstændige replikationer af spilsituationer. Derudover kan boldgenkendelse og motorstyringskode udelukkes som fejlfaktorer ved udvikling af AI'en.

Simulatoren benyttes ikke ved normalt spil mod robotten, men kun under udvikling og fejlsøgning. Da den er en integreret del af programmet er den dog en del af slutproduktet og beskrives derfor kort i selve rapporten.

6.2 På klientsiden - en vilkårlig Windows PC

I [3] lavede vi en grafisk brugergrænseflade, igennem hvilken robottens hovedprogram kunne sende informationer om, hvor den havde genkendt bolden og om, hvor dens stænger var placeret. Dette program viser sig nyttigt i sammenhæng med simulatoren. Vi har udvidet programmet, så man gennem det kan sætte startbetingelserne (boldposition, hastighed og spillerpositioner) for simulationerne.

Vi har udvidet brugergrænsefladen med et nyt vindue, som kan ses sammen med selve 3D visualiseringen på figur 6.1. På det nye simulatorvindue kan en boldposition og hastighedsvektor sættes vha. to klik, som i vinduet vises med en rød linje. Fra simulatorvinduet kan også positioner og vinkler for modstanderens stænger sættes.

6.3 På serversiden - i robottens hovedprogram

På serversiden skal socket interfacet kunne modtage input fra brugeren og sætte simulationer igang. Herefter skal en fysisk model beregne nye bold- og spillerpositioner i hver frame udfra disse startbetingelser. I denne model foretages en række approksimationer for at minimere udviklingstiden for den. Vi har ikke prøvet at måle, hvor realisisk simulationen er, men har baseret det på skøn. Det er godt nok, da simulatoren grundlæggende er et hjælpeværktøj til at finde de groveste fejl i AI'en. Detaljerne vedr. hvordan simulationen er opbygget, hvilke approksimationer vi foretager osv., er beskrevet i bilag E.

6.4 Demonstration af robottens spilleevne

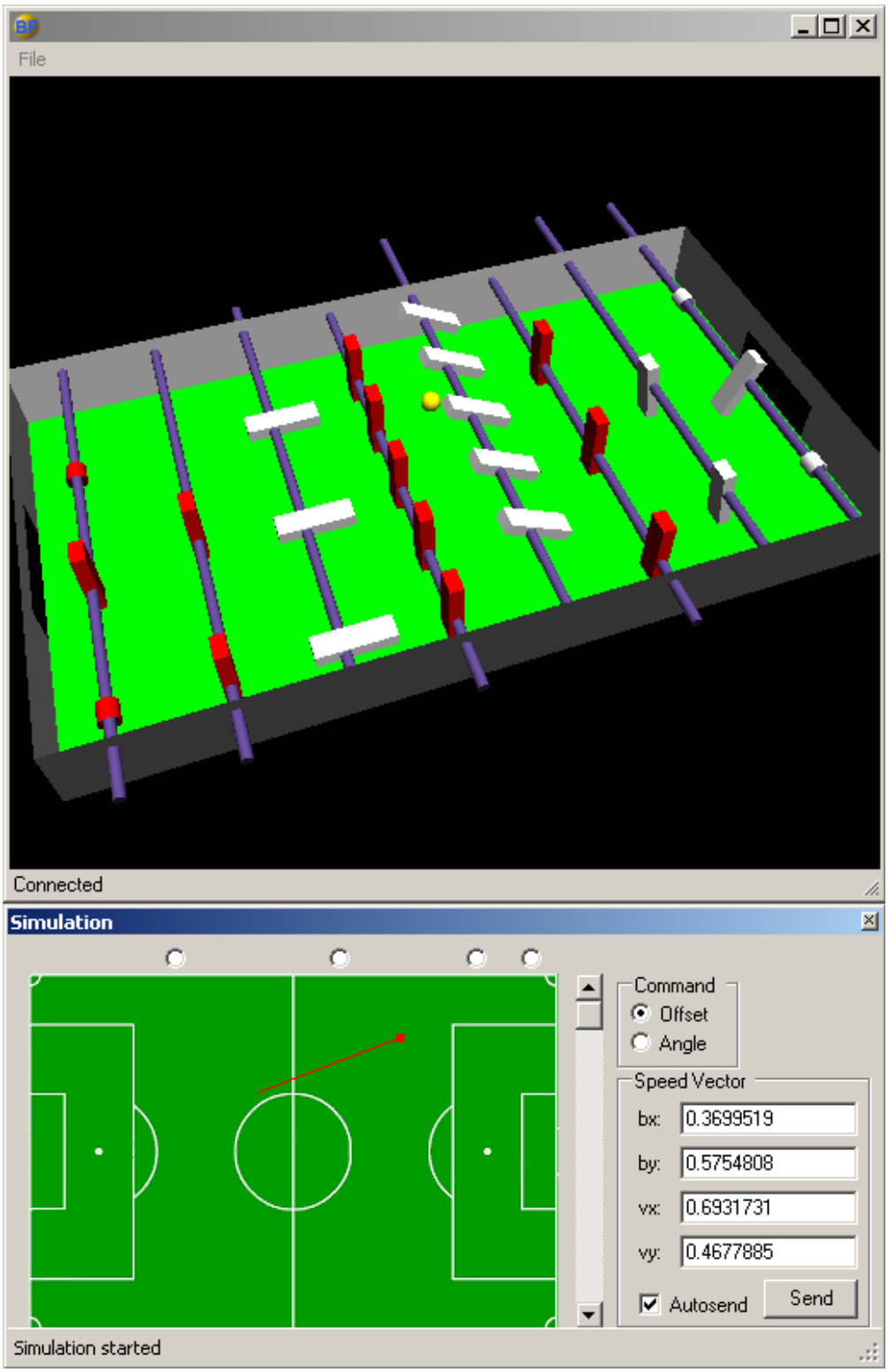
6.4.1 Generelt

For at estimere robottens spilleniveau og teste den generelt, har den spillet en række kampe mod forskellige mennesker. Hver kamp blev afsluttet når den ene spiller nåede 10 point. Modspillerne havde variende spilleniveau, alder og spillefrekvens, hvor der med spillefrekvens menes, hvor ofte vedkommende spiller bordfodbold.

Det bemærkes at kampene blev spillet løbende, mod projektets afslutning og at robotten derfor har spillet med forskellig kode under kampene. Det er kommenteret yderligere i F.

6.4.2 Spilleniveau

For at kunne sige lidt mere om, hvor godt robotten spiller, end blot at opgive andelen af vundne kampe, har vi for alle modspillere noteret hvor gode de er til at spille bordfodbold. Der findes et Elo rating system til dette, ligesom de systemer der benyttes i skak, backgammon



Figur 6.1: Brugerfladen under en simulation. Det øverste vindue viser boldens og mændenes position og har kun undergået kosmetiske forbedringer i dette projekt. Det nederste vindue, simulatorvinduet, er en ny feature.

og andre spil. Det er nærmere beskrevet på [15], men det lader ikke til at dette system benyttes i Danmark, og slet ikke af tilfældige modspillere på instituttet.

Derfor har vi opfundet vores eget rating system, som består i at modspilleren inden kampens start blev bedt om at angive, hvor god han selv syntes han var på en skala fra 0 til 10. For i så stor udstrækning som muligt at gøre denne subjektive skala ens for alle modstanderne, sagde vi til dem at 0 svarer til et spædbarn og 10 svarer til en spiller i verdensklassen.

Det har undervejs vist sig at de spillere der giver sig selv 5 har meget variende kampresultater. Det kan meget vel skyldes, at disse spillere kun har en meget vag idé har om hvor godt de spiller i forhold til andre, og derfor blot giver sig selv det midterste tal i rækken.

6.4.3 Aldersgrupper

For at kunne sammenligne resultater på en meningsfuld måde, er modspillerne også delt op i aldersgrupper.

Børn er ofte ikke så hurtige og har typisk heller ikke stor teknisk indsigt. Den manglende tekniske indsigt gør, at de har svært ved at snyde robotten, ved f.eks. at skjule bolden. Desuden har har de ofte meget begrænset erfaring med bordfodbold og angiver derfor et højere spilleniveau, end det de i virkeligheden har. Derfor har de fået en aldersgruppe for sig selv.

Unge mennesker kan reagere hurtigt og har ofte spillet bordfodbold en del gange før. De får en aldersgruppe for sig selv.

Lidt ældre mennesker reagerer normalt ikke særlig hurtigt og spiller kun sjældent bordfodbold, så de får også en aldersgruppe. Grænsen mellem denne aldersgruppe og den forrige er svær at sætte fornuftigt. Man kan sige at når folk bliver ca. 35 har de en tendens til at have børn og har derfor ikke tid til at spille bordfodbold. Derfor sætter vi grænsen til 35 år.

Gruppenavn	Aldersinterval
A	0 - 17 år
B	17 - 35 år
C	35+

6.4.4 Resultat

Data vedr. de kampe robotten har spillet kan ses i bilag F Robotten har spillet 33 kampe og vundet 24 af dem, hvilket svarer til 73%. Mod spillere i aldersgruppe A har den vundet alle 6 kampe, hvilket er rigtigt tilfredsstillende i forhold til ambitionen om at kunne vinde over en 7-årig.

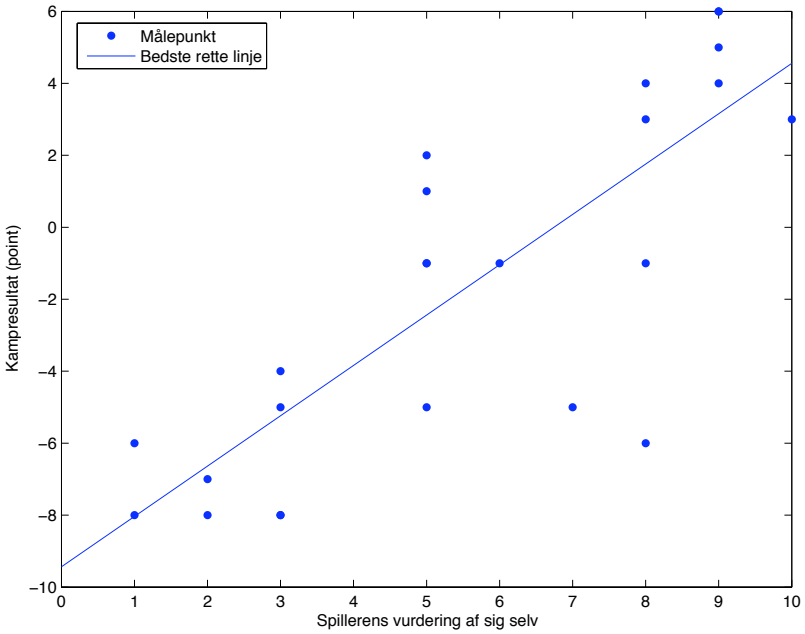
Af robottens 26 kampe mod personer i gruppe B har den vundet 17, altså 65%. I denne

gruppe forventede vi, at modstanderne gennemsnitligt er bedst, så at den har vundet mere end halvdelen af sine kampe mod dem var overraskende og viser at robotten giver nok modstand til at være sjov at spille mod.

Robotten kan vurderes på den 0-10 skala der indførtes i 6.4.2. Det gør vi ved at sammenholde kampenes resultat med den vurdering spillerne giver sig selv. Kampens resultat definerer vi som spillerens antal mål minus robottens. Derved er det for spilleren bedst tænkelige resultat 10 og det værste -10. 0 svarer til uafgjort, hvilket ikke kan forekomme, da der ikke kan scores to mål på en gang. Dog er dette resultat interessant fordi det svarer til at modspilleren og robotten spiller lige godt. Altså hvis en person gennemsnitligt har 0 i resultat mod robotten, scorer de lige mange mål mod hinanden, hvilket vi definerer som at de er lige gode.

Når folk angiver hvor gode de er, må det forventes at de sammenligner sig med personer i samme aldersgruppe, da de oftest spiller mod disse - i hvert fald hvad gruppe A og B angår. At sammenligne spilleniveauer på tværs af aldersgrupper er derfor ikke nyttigt. Vi har flest kampresultater fra aldersgruppe B og vælger derfor i det følgende kun at sammenligne disse resultater.

På figur 6.2 har vi sammenholdt modspillernes spilleniveau med deres kampresultater. Punkterne ligger på noget der minder om en ret linje i kartesiske koordinater. Den bedste rette linje gennem punkterne er givet ved $k = 1.3991m - 9.4337$, hvor k er kampens udfald og s er modspillerens spilleniveau. For den rette linje er korrelationskoefficienten $r = 0.82$. Der er altså en sammenhæng mellem hvor gode folk selv tror de er og kampenes udfald.



Figur 6.2: Spilleniveauer sammenholdt med kampresultater. Bemærk at der kan ligge flere punkter oven i hinanden.

Af den bedste rette linjes ligning fremgår det at

$$k = 1.3991m - 9.4337 = 0 \Rightarrow m = \frac{9.4337}{1.3991} = 6.7427 \approx 7$$

dvs. at robotten gennemsnitligt vil spille lige op mod spillere der synes de selv har 7 i spilleniveau. Udregningens rigtighed kan verificeres ved på figur 6.2 at se, at den rette linie skærer $k = 0$ mellem 6.5 og 7. Det må siges at være rigtig fint.

De data vi har indsamlet om, hvor ofte modspillerne spiller bordfodbold, har vi ikke fundet brug for.

6.5 Yderligere tests

Hver gang et billede er parat fra kameraet skal programmet, inden det næste billede er parat, to gange nå at genkende bolden og mændene og kompensere for rystelser. Desuden skal det også benytte AI'en til at finde ud af, hvordan stængerne skal placeres. Hvis programmet ikke når at gøre alle disse ting hurtigt nok, er det ingen katastrofe - robottens program går ikke ned eller holder op med at spille. Men robotten vil spille dårligt, da den bl.a. vil være længere tid om at opdage ændringer i boldens position og hastighed.

Vi lavede i [3] en klasse, `Profiler` til at måle den tid hver frame tager. Den kan også tage tid på enkelte af de ting der sker i en frame, men det er ikke så interessant her. For at teste, om systemet foretager billedgenkendelse osv. hurtigt nok, måler vores `Profiler` den tid hver eneste frame tager. Hvis den er længere end den tid det tager at få et nyt billede ind, bliver det skrevet til terminalen.

Vi har fremprovokeret at en frame tager for lang tid, og derved testet, at der skrives en fejlbesked i det tilfælde. Ved normalt spil skriver den ikke disse fejlbeskeder. Vi kan konkludere at systemet altid gennemfører billedgenkendelse osv. indenfor den krævede tidsramme.

Kapitel 7

Konklusion

Vi fandt aldrig en 7-årig som robotten kunne spille imod. Til gengæld har robotten vundet ca. 3/4 af kampene den har spillet mod teenagere og DTU-studerende og bliver på en skala fra 0 til 10 bedømt til 7. Man må derfor gå ud fra, at den også ville have vundet over en 7-årig, som det var vores ambition.

For at nå dertil har vi foretaget en masse udvidelser og ændringer af det system vi overtog i starten af projektet. Hardwaren er ændret, så man ikke længere oplever flere impulstab end at man normalt kan spille en kamp uden afbrydelser mod robotten. Dog er vi med den nuværende hardware begrænset til at spille $\frac{1}{2}$ - 1 time pga. varmeudviklingen i motorerne. Vi har indsat sikkerhedstjek, så motorerne stopper hvis de blokeres, og undgår på den måde at hardwaren overophedes.

Billedgenkendelsesdelen i softwaren er udvidet med kalibrering og kompenserer nu så godt for rystelser, at en ændring af kamerastativet ikke er nødvendig. Boldgenkendingsalgoritmen er blevet finpudset, men hæmmes stadig af at bolden kan skjules under stængerne. Positionen af modspilleres stænger kan nu også genkendes.

Vi har programmet en AI til at styre stængerne. Den baserer sig på, hvordan vi selv spiller, ved at dele spillet op i mange forskellige tilstande, og fungerer først og fremmest reaktivt. Til udvikling og fejlfinding af AI'en har vi implementeret en model af bordet, som kan simulere spillet inklusiv kollision med mændene.

Litteratur

[1] Sølvhøj, J. & Breiting, M., *Vision og styring til bordfodboldrobot*, december, 2003.

[2] Lindahl, M. & Laursen, M. & Olsen, J., *Bordfodbold*, 23. december, 2004.

[3] Myrup, A. C. & Ørding-Thomsen, M. *Analyse af hardware til bordfodbold 1*, 20. februar 2006

[4] Myrup, A. C. & Ørding-Thomsen, M. *Analyse af hardware til bordfodbold 2*, 7. september, 2006

[5] Hovedstadens Bordfodboldforening, *DM Resultater 2004*, 15. januar 2007, http://hobofo.dk/hobofo/downloads/pafiledb.php?action=file&id=25

[6] Wikipedia, *Machine learning* , 22. januar 2007, http://en.wikipedia.org/wiki/Machine_learning

[7] Wikipedia, *YUV* , 22. januar 2007, http://en.wikipedia.org/wiki/YUV

[8] Wikipedia, *Pulse-width modulation* , 4. februar 2007, http://en.wikipedia.org/wiki/Pulse-width_modulation

[9] Spil-Aps, *Bordfodbolde Alm.Plastbolde 5 stk. Standard bolde med mønstre.* , 22. januar 2007, http://www.spil-aps.dk/vare/148.htm

[10] Paul Pinto, AI depot, *Minimax Explained* , 22. januar 2007, http://ai-depot.com/LogicGames/MiniMax.html

[11] Paul Pinto, AI depot, *Alpha-Beta Cutoff* , 22. januar 2007, http://ai-depot.com/LogicGames/MiniMax-Cutoff.html

[12] Ingrid F. Russell, University of Hartford, *Neural Networks* , 22. januar 2007, http://uhavax.hartford.edu/compsci/neural-networks-tutorial.html

[13] Dr. Dan Boye, Davidson College, *Rolling Friction* , 24. januar 2007, http://webphysics.davidson.edu/faculty/dmb/PY430/Friction/rolling.html

[14] Regeludvalget for Dansk Bordfodbold Forbund 2005, Dansk Bordfodbold Forbund, *Sportsregler*, 25. januar 2007, http://hobofo.dk/hobofo/downloads/pafiledb.php?action=file&id=47

[15] Bonzini USA, *Player Ranking/Rating System*, 18. august 2000, http://www.bonziniusa.com/foosball/tournament/TournamentRankingSystem.html

[16] Jensen, P.K., *Optimisation of components for autonomous vehicles*, 31. januar, 2001.

[17] National Instruments, *Optical Encoder Fundamentals* , 4. februar 2007, http://zone.ni.com/devzone/cda/tut/p/id/4672

Bilag A

Hardware

A.1 Specifikationer

A.1.1 Motor

- Graupner Speed400
- 7.2 Volt
- 44 Watt

A.1.2 Strømforsyning

Ved projektets start havde vi en Altai 7-10 A, 13.8 V DC strømforsyning.

Ved projektets afslutning havde vi en EA-PS 9060-48 0-60 V, 0-48 A DC strømforsyning.

A.1.3 Encodere

Encoderskiverne har en opløsning på 2000 ticks med fuld kvadratur, men vi benytter kun den halve opløsning, da microcontrolleren ellers ikke kan håndtere alle interrupts [1].

A.1.4 Kamera

Kameraet er et JAI-2040:

- Farvekamera
- Opløsning 752x582 PAL
- 25 FPS interlaced/50 FPS
- 1/2" CCD

A.1.5 Computer

- Real time Red Hat Linux
- Pentium III 450 mHz
- 128 MB Ram

A.2 Ting der på sigt bør ændres ved hardwaren

Hastigheden på den lineære bevægelse er begrænset af at Speed400 motorerne er underdimensionerede til dette formål. Sættes reguleringen op, bliver motorerne for hurtigt varme.

De rød-sorte strømforsyningsledninger til controllerkortene er kun $0.75mm^2$, hvilket er underdimensioneret i forhold til de strømme der løber igennem dem. Det fører til et unødvendigt stort spændingstab, hvilket kan undgås ved at benytte korrekt dimensionerede ledninger.

Fladkabler er beregnet til brug i et stillestående system som f.eks. inde i en computer, og ikke til et bevægeligt system som bordfodboldrobotten. Det har givet problemer i form af løse forbindelser i kablernes stik. Der bør findes egnede kabler i stedet.

Lineært drives stængerne af tandhjul og tandremme. Disse tandhjul har en meget lille radius, hvilket fører til metaltræthed i de metaltråde der er i tandremmene. Tandhjul med større radius vil forlænge levetiden for tandremmene. Evt. kan tandremmene udskiftes med kæder.

Man kunne overveje at udskifte computeren, da den er meget tæt på at være overbelastet. Hvis vores program kører og en anden er logget ind over over SSH samtidigt, viser vores profiler at programmet ikke kan følge med.

Under programmering af et motor-controller board kan converteren brænde af, hvis RS485 stikket fra motor-controller boardet ikke er taget ud. Dette burde ikke ske, da computeren og strømforsyningen til bordet er jordet. Vi har ikke debugget dette fænomen, da det ikke sker når man sørger for at tage RS485 stikket ud før programmering.

Bilag B

Motorstyring

B.1 Varme udenfor spil

B.1.1 Laveste spænding der resulterer i bevægelse

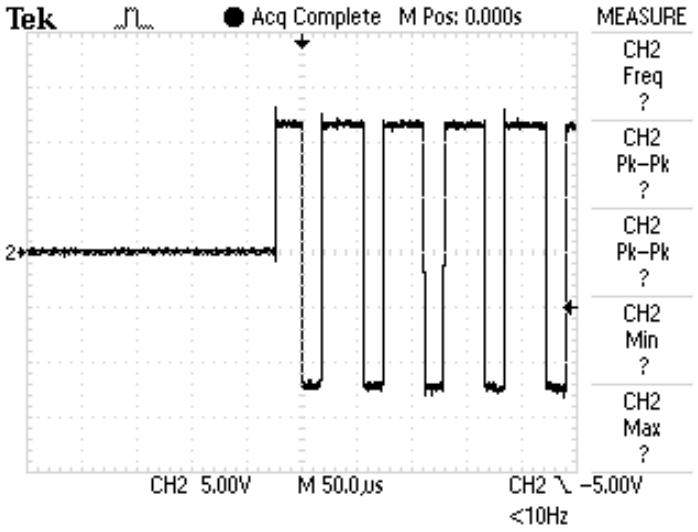
En motor-controllerversion, der sætter pulsbredden via kommunikationsinterfacet skrives. Med kolde motorer testes forskellige pulsbredder omkring 128 (hvor pulsbredde forholdet er en byte uden fortegn) og det erfares hvilken værdi der minimum skal til for igangsætning af bevægelse.

Testen gentages for alle stænger, og heraf fås et minimum interval, som alle stænger kunne igangsættes ved på 128 ± 8, der kan gælde for alle stænger på begge akser.

Det kan godt tænkes at pulsbredder tættere på 128 ville kunne have effekt på hastigheden så snart stængerne er kommet i bevægelse og derfor ikke længere vil være påvirket af den store statiske friktion. Det vil dog være for kompliceret i forhold til udbyttet at undersøge, hvilke pulsbredder, der er acceptable for forskellige hastigheder.

B.1.2 Test efter implementering

På figur B.1 ses en måling med oscilloskop hvor H-broen slår fra.



Figur B.1: Måling på motor, der rykkes en hel encoderomgang. Bemærk at spændingen er tæt på 0 før bevægelsen igangsættes, hvilket indikerer, at H-broen er slået fra.

B.2 Blokering af rotationsbevægelsen

Hvis man forestiller sig at den ønskede position inden for kort tid sættes, så den skiftevis er på hver side af den nuværende position, så kan det ske at motoren ikke når at bevæge stangen før den får ordre på at reversere. Her vil tælleren inkrementeres i hver frame, på trods af der ikke er tale om en blokering. Løsningen på dette er også at nulstille tælleren, når der fra hovedprogrammet er modtaget besked om at der skal flyttes modsat bevægelsesretningen.

B.3 Fejlregistreringer af impulser fra encoder

B.3.1 Test af kompensering i motor-controllersoftware

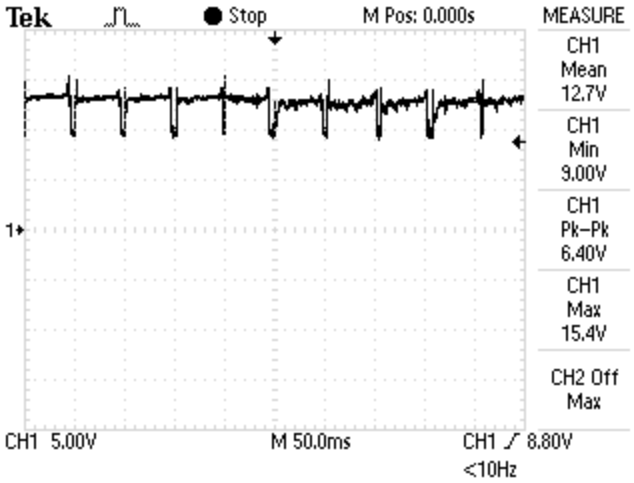
Rotationsbevægelsen Fejlregisteringer kan observeres med det blotte øje, hvis stangen drejes i intervallet $[\frac{\pi}{2}; \frac{3\pi}{2}]$ og tilsvarende for negative vinkler, da vi ikke sætter vinkler i disse intervaller noget som helst sted i hovedprogrammet.

Et spil igangsættes og der spilles indtil synlige fejlregistreringer forekommer. Herefter føres bolden hen til den implicerede stang, og den vil herefter forsøge at skyde. Under skuddet passeres indekspulsen, og derfor drejer stangen ikke tilbage op i intervallet efter skuddet.

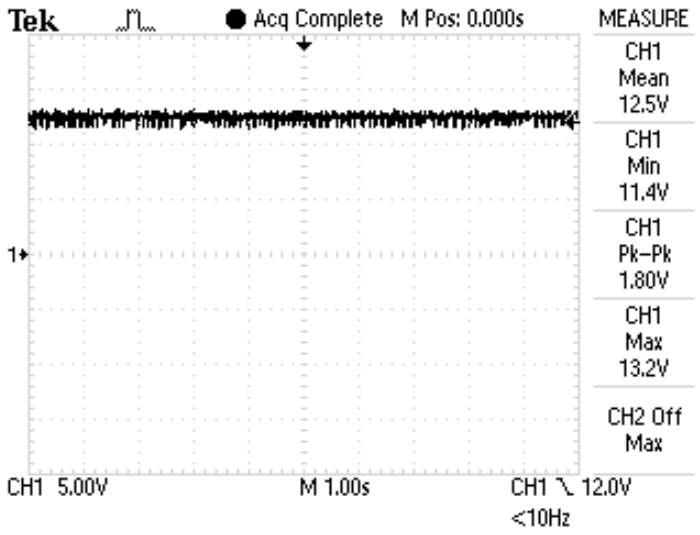
Sidelænsbevægelsen Fejlregistreringer kan observeres med det blotte øje, ved at en stang presser mod en side, og desuden ikke kan bevæge sig helt ud til den anden side. Et spil igangsættes og der spilles indtil synlige fejlregistreringer forekommer. Bolden holdes helt ud til den side stangen skubber imod, og det erfares at stangen ikke længere presser imod siden efter et kort øjeblik. Bolden føres til modsatte side og det konstateres, at stangen pånyt kan bevæge sig helt ud til den anden side.

B.3.2 Måling af spændingen med forskellige strømforsyninger

Af figur B.2 og figur B.3 ses en forøget stabilitet af spændingen over en enkelt stang.



Figur B.2: Måling på strømforsyningen til to motor-controllere dvs. een stang, med 13.8 V 10A strømforsyning. Bemærk at spændingen svinger med 6.4 V og helt ned til 9V.



Figur B.3: Måling på strømforsyningen til to motor-controllere dvs. een stang, med 12 V 48A strømforsyning. Bemærk at spændingen svinger med 1.8 V og kun ned til 11.4V.

Bilag C

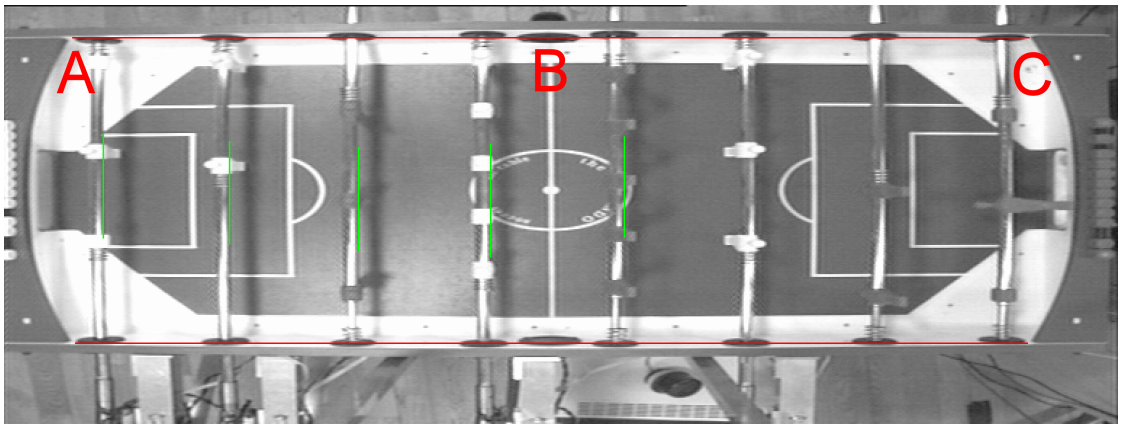
Billedgenkendelse

C.1 Rotation-, trapez- og tøndeeffekter

Det undersøges, hvor stor indflydelse kameraets rotation og trapezeffekt har i forhold til en retvinklet tolkning af billedet. Dette måles vha. en kant på bordet, som burde være lige på billedet. Forskellen i y-retningen på dens to ender måles. En lige linje parallelt med denne indtegnes med rødt på figur C.1, for nemmere at kunne se, hvor skæv linjen er. Dette gøres for to linjer på billedet. For den øverste linje har venstre endepunkt A på linjen y-koordinaten 21 og C til højre 19. For den nederste linje er værdierne hhv. 233 og 233, hvilket giver en maksimal afvigelse på 2 pixels. Da disse linjer ligger i udkanten af spillefladen og løber i hele spillefladens længde, kan ingen punkter på banen afvige mere end 2 pixels fra den retviklede tolkning af billedet.

For de samme linjer i billedet måles tøndeeffekten. For C, midt på den øverste linje er y-værdien 16 og på den nederste 236. Forskellen i y-værdi for midten af linjen og gennemsnittet i yderpunkterne er 4 og 3 pixels for hhv. den øverste og den nederste linje.

Ens afstande i virkeligheden er ikke nødvendigvis ens på billedet. Ellers ville der ikke være en tøndeeffekt. Denne effekt kan også tænkes at kunne ses på afstandene mellem stængerne, så afstanden mellem forskellige par af nabostænger ikke er den samme, selvom de er det i virkeligheden. Forskellige par af nabostængers afstand måles ved at måle positionen af den højre side af stængerne. Målestederne er vist med grønne linjer på C.1. Deres x-koordinater er, fra venstre mod højre: 22, 109, 198, 288 og 390, hvilket giver indbyrdes afstande på 87, 89, 88 og 92 pixels. Det må forventes at vi har opmålt stængernes positioner med maksimalt 1 pixels unøjagtighed. I x-retningen på y-kanalen svarer en gennemsnitspixel på spillefladen til $\frac{1.215\text{m}}{676\text{pixels}} = 1.80\text{mm}$. Den største forskel i målt afstand på par af stænger er 5 pixels, hvilket altså svarer til 9 mm. Hvad denne afstand egentlig repræsenterer for os i forhold til boldgenkendelsen, er svært at sige, for den angiver jo ikke en maksimal størrelse på fejlen for den



Figur C.1: Måling af billedets rotation. Målelinjer er indtegnet med rødt.

genkendte boldposition. Størrelsen hænger dog lidt sammen med denne og vi kan konkludere at den helt sikkert er stor i forhold til boldens diameter som vi i [3] målte til 3.46 mm. Dog er det i x-retningen, som vi ikke er meget følsomme overfor. I praksis har det vist sig at robotten spiller fint denne fejl til trods.

C.2 Valg af boldfarve

For lettest muligt at kunne genkende bolden, må den have en anden farve end de andre objekter på bordet. De mest oplagte farver at bruge må være primær- og sekundærfarverne (rød, gul, blå, orange, lilla og grøn) plus hvid og sort. Nuancer af disse farver er ikke interessante, da de vil være svære at skelne fra de øvrige. Af disse farver findes grøn, hvid og rød allerede på bordet i form af underlaget og mændene. Genkendelse af de resterende farver testes vha. kameraet.

På bordet lægges ting af forskellig farve som vist i tabel C.1. De placeres fra venstre mod højre i den rækkefølge, de er skrevet i tabellen.

Farve	Objekt
Gul	bold
Blå	æske
Orange	gulerod
Lilla	kaffekandelåg
Sort	oplader

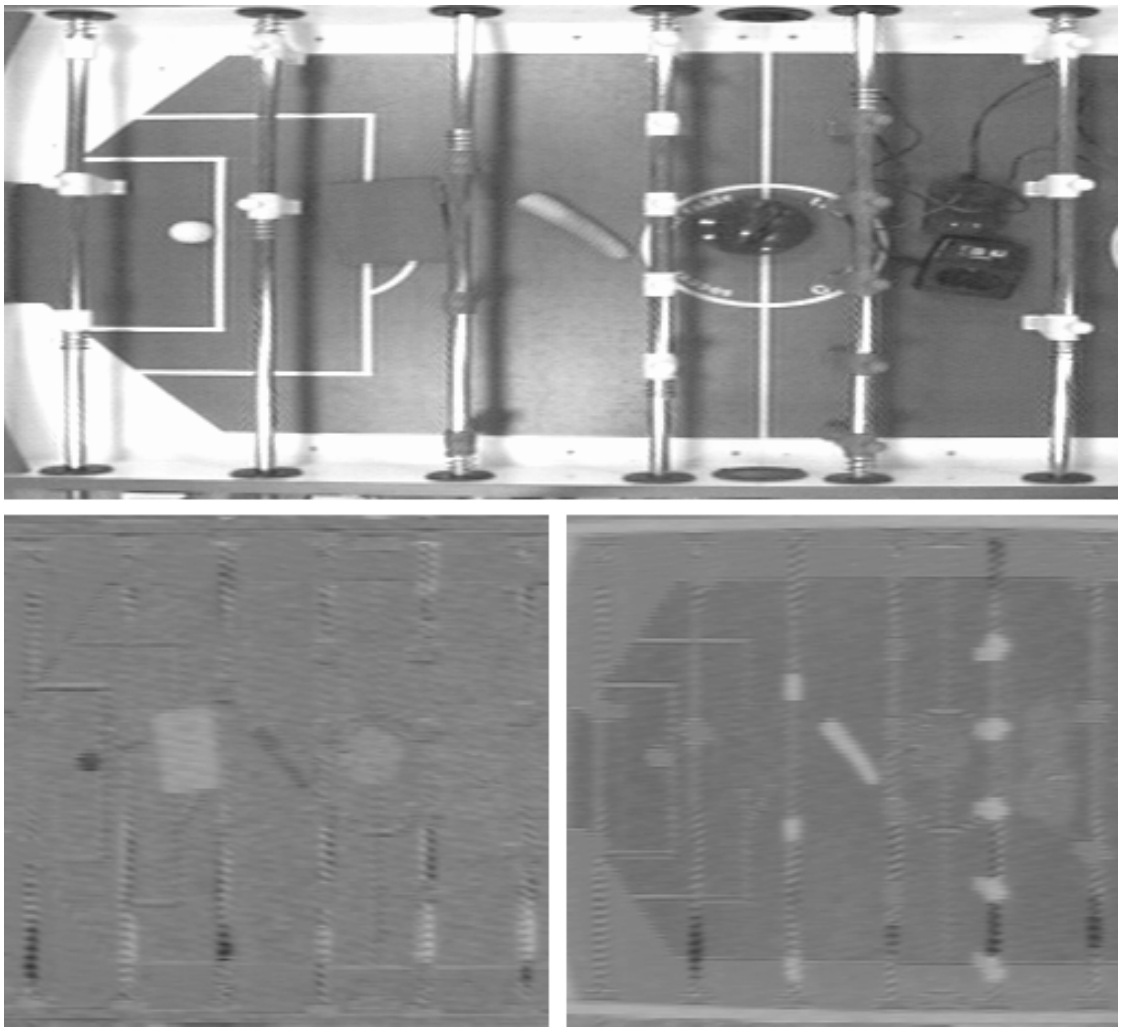
Tabel C.1: På bordet placeres disse objekter, for at finde en farve, der er let at genkende.

Billeder fra de forskellige kanaler fra kameraet er vist i figur C.2. Her ses det at

- Den gule bold står tydeligt frem på u-kanalen, hvor den dog kan forveksles med genskin fra stængerne. Den står også ret tydeligt frem på y-kanalen.
- Den blå æske er tydelig på u-kanalen, men på ingen af de andre. Denne kan på u-kanalen

også forveksles med genskin, men genskinnet giver sig typisk udslag i en høj værdi på v-kanalen.

- Guleroden er nem at se på v-kanalen men kan dog forveksles med de røde mænd på denne kanal.
- Det lilla kaffekandelåg og den sorte oplader står klart frem på y-kanalen, men ingen af de andre. Dog kan de under de rette lysforhold ligne de skygger der er på banen.



Figur C.2: Objekter af forskellig farve er lagt på bordet, for at finde ud af hvor let de kan genkendes. Øverst er y-kanalen, nederst til venstre u-kanalen og til sidst v-kanalen

Hver farve står ret klart frem på en af kanalerne, men kan også forveksles med andet på banen. Derfor analyseres valget af boldfarve yderligere: I modsætning til skyggerne og de røde mænd, er genskinnet fra stængerne ikke et stort problem, idet programmet kan finde ud af, hvor de er. Derfor kan boldfindingsalgoritmen blot lade være med at lede efter bolden i disse områder. Man kan altså lave en geometrisk udmaskning af stængerne.

De to bedste muligheder er altså en blå eller en gul bold. Den gule står ret klart frem på to

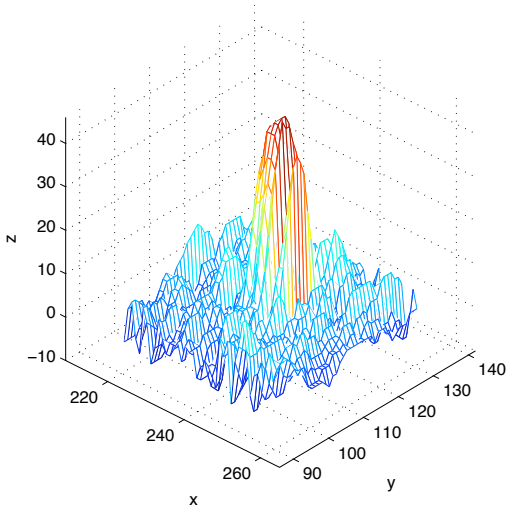
kanaler i modsætning til den blås ene kanal. Da den gule bold kan forveksles med genskin af stængerne, vil man være nødt til at foretage en geometrisk udmaskning af disse.

Det meste af det genskin, den blå bold kunne forveksles med på u-kanalen, kunne udmaskes på baggrund af farven fordi det også optræder ret tydeligt på v-kanalen. Altså kunne man nok undgå den groveste del af en geometrisk udmaskning af genskinnet, hvis man brugte en blå bold.

C.3 Algoritmemæssige detaljer vedr. billedgenkendelsen

C.3.1 Massemidtpunktsberegning

I et område på lidt over boldens størrelse rundt om midtpunktspixlen defineres en værdi for hver pixel til udregning af massemidtpunktet. Værdien på u-kanalen kan ikke benyttes direkte, idet bolden er karakteriseret ved lave værdier og fordi området udenom bolden ikke har værdien nul. Derfor vendes fortegnet på u-kanalens værdi og nulpunktet justeres, så de pixels der ikke viser en bold har ca. nul som værdi (billedets støj gør, at det ikke er præcis nul). Da haves et x-y område med værdierne på z-aksen som det vist på figur C.3. I det beskrevne område udregnes massemidtpunktet mht. z for både x- og y-aksen. Dette returneres som boldens position.



Figur C.3: Boldens massemidtpunkt findes ud fra værdier som disse, hvor bolden ses i midten. For at minimere effekten af støjen rundt om bolden, tages massemidtpunktet kun af et område langt mindre end det viste.

C.3.2 Konvertering af klistermærkekoordinatpositioner til spillefladehjørner

På et tilfældigt billede af y-kanalen opmåles det, at klistermærkernes x-koordinater er 24 og 753, hvilket giver en afstand på 729 pixels. Spillefladens hjørner har x-koordinaterne 50 og 729 og derfor en bredde på 679 pixels. Det antages at forholdet mellem disse bredder er konstant, uanset hvordan kameraet er drejet og hævet/sænket, indenfor de snævre justeringsrammer, hvor hele bordet er indenfor billedet. Det må være en rimelig antagelse, da kameraet er 1.2 m over bordet og kun kan rykke sig få cm og dreje sig få grader.

Når der kalibreres, findes klistermærkernes positioner og afstanden mellem disse. Ud fra det fundne forhold, omregnes dette til en spillefladebredde. Det antages, at spillefladens midtpunkt er midt i det område, der udspændes af klistermærkerne.

For y-koordinaterne gøres det samme, men her er klistermærkernes koordinater 37 og 217. Spillefladens hjørner har y = 27 og 226. Her bemærkes det, at spillefladen er bredere end afstanden mellem klistermærkerne, men beregningerne foregår som for x-koordinaten.

C.3.3 Konvertering af klistermærkekoordinatpositioner til stængernes x-positioner

På et tilfældigt billede af y-kanalen måles modstanderens stængers positioner i x-retningen i forhold til spillefladen. Stængerne er 25, 111, 292 og 475 pixels i forhold til højre ende af spillefladen. Robottens stænger antages at fordele sig på samme måde, blot fra venstre side. Spillefladen måles at være 676 pixels lang. Det antages at forholdene mellem afstandene og bordet længde er konstante uanset kameraets placering, da det ikke kan rykkes mere end et par cm.

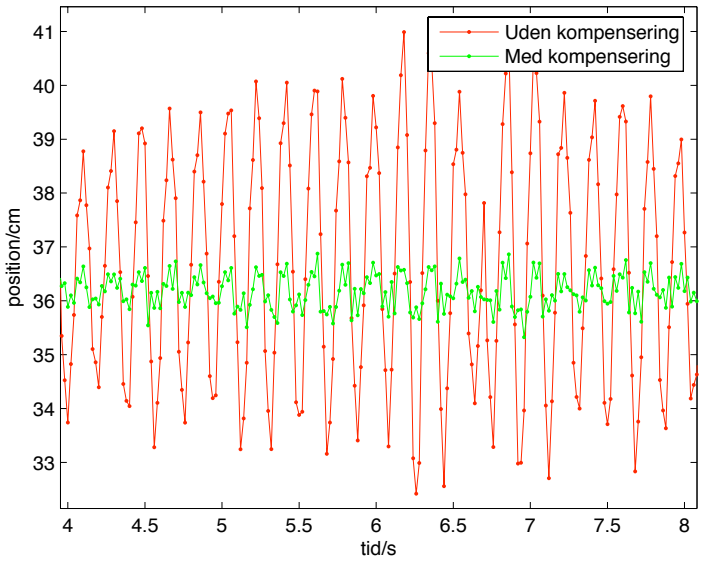
Når der kalibreres, benyttes de målte forhold til at finde stængernes positioner i x-retningen. Koden ligger i `findRodOffsets` funktionen i `FieldCalibrator` klassen.

C.3.4 Kompensering for rystelser

Nogle implementeringsmæssige detaljer vedr. kompenseringen for rystelser beskrives her.

Som udgangspunkt kompenserer vores algoritme beskrevet i 4.4.3.3 for lidt for kameraets rystelser. Vi foretager samme test som beskrevet i 4.6.4, altså genkendelse af en bold der ligger stille, mens kameraet ryster. Den fundne boldposition med og uden kompensering for rystelserne kan ses på figur C.4.

Den fundne boldposition efter kompensering har udsving i samme retning som den fundne boldposition inden kompensering. Det ses på figuren ved at den grønne graf har en tendens til at svinge som den røde graf, dog med mindre amplitude. Efter kompensering er standardafvigelsen for den genkendte bolds position 0.32 cm i tidsrummet 4 - 8 sekunder.



Figur C.4: Der kompenseres som udgangspunkt for lidt for rystelserne - den grønne kurve følger den røde men med mindre udsving. Det laver vi om på.

At der kompenseres for lidt, skyldes sandsynligvis at kompenseringen dannes på baggrund af træk ved bordet, som er tættere på kameraet end selve spillefladen hvor bolden befinder sig. Der kan altså være tale om parallakseforskydning. Vores løsning er følgende:

1. Vores algoritme ser og udregner, hvor mange pixels der skal kompenseres med.

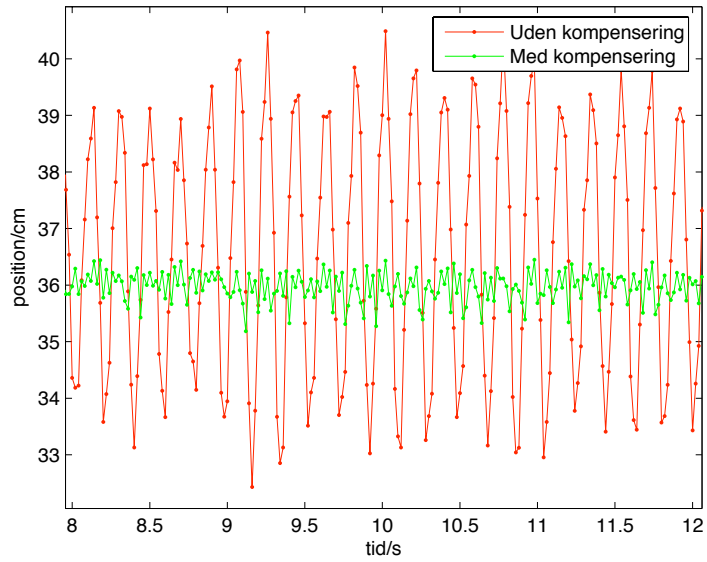
2. Dette tal multipliceres med 1.1 som ren symptombehandling.

3. Det nye tal benyttes til kompenseringen.

Effekten af denne ændring ses på figur C.5, hvor den genkendte position har fået reduceret standardafvigelsen til 0.27 cm. At reducere den gennemsnitlige fejl med 0.5 mm til hver side er ikke alverden, men bedre end ingenting. På figuren er det ikke til at se en systematisk afvigelse længere.

Faktoren 1.1 blev fundet eksperimentelt, dog uden dokumentation. Hvis faktoren er lavere end 1.05, kan den systematiske fejl ses. For højere værdier end 1.15 ses en overkompensering.

At finde ud af om problemet virkeligt skyldes parallakseforskydning har vi pga. problemets størrelse prioriteret lavt og derfor ikke gjort.



Figur C.5: Nu kompenseres der nok for rystelserne - den grønne kurve følger ikke den røde.

C.4 Test af boldgenkendelsen

En stillestående bold blev placeret på følgende 6 positioner, hvor hver række angiver 5 målinger af én koordinat. Hvert par rækker angiver x- og y-koordinater for én boldposition:

```
x1    0.1314    0.1315    0.1302    0.1313    0.1313
y1    0.1264    0.1275    0.1288    0.1278    0.1267
x2    0.1523    0.1515    0.1514    0.1518    0.1522
y2    0.5871    0.5860    0.5866    0.5872    0.5862
x3    1.1051    1.1042    1.1060    1.1048    1.1079
y3    0.5591    0.5586    0.5591    0.5597    0.5588
x4    1.0828    1.0839    1.0832    1.0824    1.0837
y4    0.1272    0.1275    0.1283    0.1276    0.1287
x5    0.6133    0.6141    0.6139    0.6138    0.6141
y5    0.6409    0.6417    0.6418    0.6419    0.6413
x6    0.6069    0.6069    0.6074    0.6067    0.6076
y6    0.0556    0.0558    0.0560    0.0557    0.0571
```

Af programmet blev boldens position igennem 5 målinger genkendt på disse positioner:

```
x1   13.3000
y1   12.6000
x2   15.3000
y2   59.2000
x3  110.3000
```

```
y3   56.0000
x4  107.7000
y4   12.6000
x5   61.7000
y5   64.2000
x6   61.0000
y6    6.1000
```

I den test hvor bolden var i bevægelse, er de rå data for boldens genkendte x- og y-positioner:

```
0.9661    0.4132
0.9658    0.4194
0.9648    0.4149
0.9649    0.4196
0.9652    0.4127
0.9645    0.4191
0.9661    0.4155
0.9674    0.4197
0.9662    0.4132
0.9659    0.4202
0.9666    0.4146
0.9657    0.4227
0.9660    0.4127
0.9661    0.4200
0.9671    0.4162
0.9670    0.4190
0.9336    0.4146
0.8942    0.4163
     0         0
0.8090    0.4142
0.7690    0.4123
0.7311    0.4138
     0         0
0.6553    0.4132
0.6232    0.4083
0.5880    0.4117
0.5586    0.4111
     0         0
0.4877    0.4087
0.4546    0.4114
0.4236    0.4090
0.3971    0.4130
     0         0
0.3243    0.4119
0.2951    0.4102
0.2647    0.4125
0.2374    0.4111
     0         0
0.1733    0.4104
0.1410    0.4140
0.1124    0.4103
0.0838    0.4130
     0         0
0.0216    0.4091
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
0.9565    0.3232
0.9549    0.3198
0.9564    0.3232
0.9552    0.3187
0.9572    0.3238
0.9557    0.3191
0.9566    0.3247
0.9561    0.3200
0.9568    0.3241
0.9551    0.3194
0.9559    0.3235
0.9551    0.3200
0.9568    0.3221
0.9555    0.3204
```

```
0.9556    0.3234
0.9563    0.3196
0.9561    0.3228
0.9550    0.3188
     0         0
     0         0
     0         0
     0         0
     0         0
0.6177    0.3244
     0         0
0.4949    0.3272
0.4434    0.3274
0.3994    0.3270
0.3362    0.3294
0.2874    0.3287
0.2426    0.3321
0.1813    0.3321
0.1433    0.3322
0.1037    0.3331
     0         0
0.0137    0.3341
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
     0         0
```

hvor 0 angiver, at bolden ikke blev fundet i den pågældende frame. Hastigheden udregnes som

$$v = \frac{0.4949\text{m} - 0.1037\text{m}}{8 \text{ frames}} 50 \text{ frames/s} \approx 2.4 \text{ m/s}$$

C.5 Test af genkendelse af modspillerens mænd

Fra venstre søjle mod højre ses de genkendte positioner for modspillerens målmands-, forsvars-, midtbane- og angrebsstang, da stængerne var hevet mod 0-positionen.

```
0.00229951  −0.00622992  −0.00438772  −0.00498037
0.0046855   −0.0044917   −0.0034801   −0.0009048
0.00354162  −0.00587672  −0.00375591  −0.00464572
0.00408654  −0.00391411  −0.00189849  −0.00227848
0.00358378  −0.00543007  −0.00394084  −0.00434899
0.00446333  −0.00444282  −0.00081173  −0.00231598
0.0037686   −0.00593581  −0.00389365  −0.00438033
0.00484557  −0.00470116  −0.000670022 −0.00306082
0.00425166  −0.00610981  −0.00253799  −0.00378854
0.00438512  −0.0037944   −0.000506254 −0.00354137
0.00325888  −0.00645646  −0.00381288  −0.00463297
0.00410487  −0.0048874   −0.00180693  −0.00418194
0.00227247  −0.00606333  −0.00360543  −0.00373777
0.00411849  −0.00376798  −0.00253979  −0.00307116
0.00221188  −0.00605725  −0.00466981  −0.00371743
0.00416037  −0.00406257  −0.00189924  −0.0043901
0.00238892  −0.0061552   −0.00420754  −0.00374322
0.00410478  −0.0043785   −0.00274266  −0.00242701
```

```
0.00241986  −0.00638862  −0.00439083  −0.00462161
0.00401076  −0.00460945  −0.00257146  −0.00183643
```

De tilsvarende positioner, da stængerne var hevet så langt de kunne den anden vej:

```
0.191591  0.353331  0.118584  0.174323
0.192481  0.353604  0.11899  0.17618
0.189844  0.351772  0.118317  0.173825
0.192393  0.353734  0.119208  0.176656
0.189918  0.352563  0.118063  0.175427
0.192304  0.354127  0.119481  0.177488
0.189592  0.351434  0.11739  0.175508
0.191267  0.354118  0.120351  0.177314
0.189145  0.351845  0.118778  0.17669
0.192036  0.353044  0.119789  0.177431
0.189343  0.351794  0.119216  0.175841
0.19188  0.354021  0.119902  0.178043
0.190636  0.351895  0.119105  0.1744
0.192051  0.354086  0.119893  0.175906
0.190321  0.351598  0.119303  0.174595
0.193273  0.352488  0.120945  0.176012
0.191651  0.352737  0.118595  0.173628
0.192838  0.354693  0.119317  0.176741
0.189683  0.353298  0.117002  0.173959
0.192112  0.352812  0.119849  0.176755
```

Stængernes positioner blev her manuelt målt til 18.7, 35.3, 11.8 og 17.8 cm.

For testen, hvor forsvarsmanden blev hevet frem og tilbage er de rå y-positioner angivet her. Der er 1/50 s mellem hver måling.

```
0.2850
0.2913
0.3048
0.3082
0.3156
0.3153
0.3132
0.3027
0.2884
0.2612
0.2350
0.2006
0.1667
0.1267
0.0955
0.0694
0.0554
0.0457
0.0449
0.0469
0.0508
0.0524
0.0574
0.0573
0.0616
0.0600
0.0657
0.0810
0.1117
0.1502
0.1961
0.2313
```

```
0.2592
0.2811
0.2949
0.3008
0.3037
0.3035
0.3034
0.3021
0.3006
0.2989
0.2995
0.2982
0.3005
0.2916
0.2807
0.2509
0.2115
0.1659
-0.1272
0.0754
0.0472
0.0291
0.0250
0.0254
0.0273
0.0281
0.0303
0.0282
0.0304
0.0279
0.0294
```

Hastighederne blev udregnet på følgende måde:

$$v1 = \frac{0.259152 - 0.0810292}{5 \cdot 1/50} = 1.7812 \tag{C.1}$$

$$v2 = \frac{0.280657 - 0.0472051}{6 \cdot 1/50} = 1.9454 \tag{C.2}$$

Hvor $v1$ var hastigheden for hvilken genkendelsen lykkedes i alle frames og $v2$ var hvor den mislykkedes for en frame.

C.6 Kalibreringstest

Bordets kalibrering er testet ved forskellige belysningsforhold. Dels i vores lokales normale arbejdsbelysning, dels udelukkende med indendørs vinterdagslys og dels med kraftig belysning.

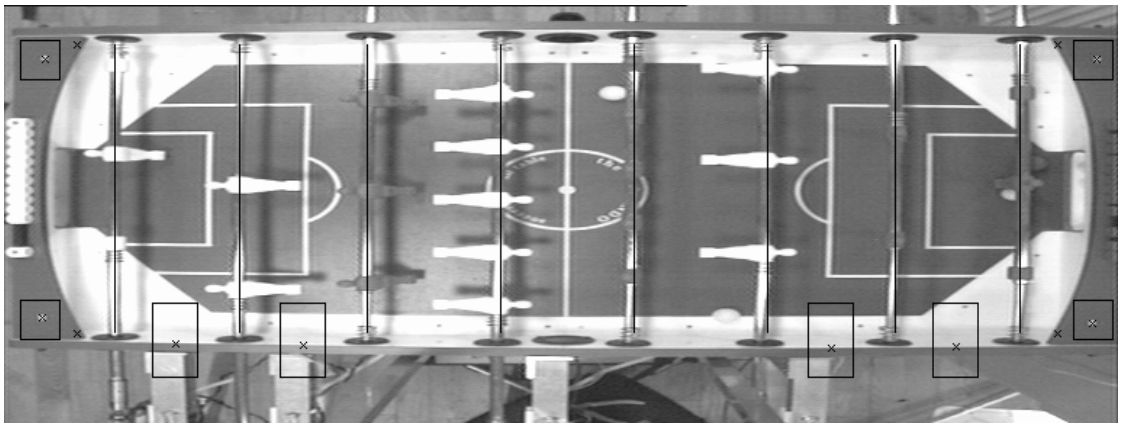
De værste af disse lysforhold er så dårlige at bolden ikke kan genkendes, så hvis kalibreringen fungerer under disse lysforhold, så fungerer kalibreringen godt nok.

Forsøget blev udført ved at indstille lyset først og derefter starte programmet. Ved opstart gemmer det et billede af kalibreringen. På disse billeder tegner programmet et kryds på hvert klistermærke, det har fundet. Hvis alle fire har et kryds på midten af sig, er lokalisering af spillefladen gået godt. Rektanglet i hvert hjørne viser, hvor programmet forventer at klistermærket befinder sig.

Desuden indtegner programmet linjer, hvor den estimerer at stængerne befinder sig. Hvis disse ligger oven i stængerne, er lokalisering af stængerne gået godt.

C.6.1 Normal belysning 1

I denne test bruges loftbelysningen i vores lokale plus dunkelt lys fra vinduerne en januar eftermiddag. På figur C.6 ses kalibreringsbilledet. Da det har krydser i alle fire hjørner og stængerne har påtegnede linjer, fungerer kalibreringen.



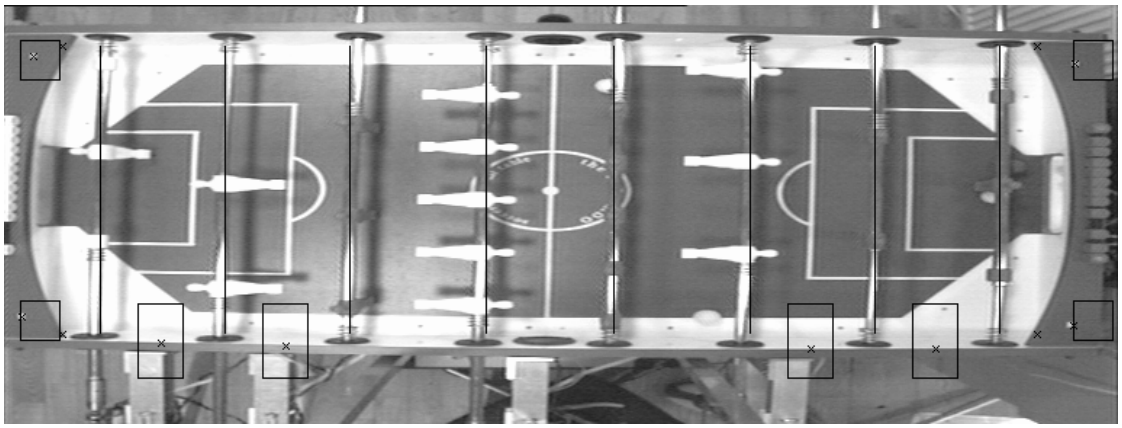
Figur C.6: Kalibrering ved normal belysning

C.6.2 Normal belysning 2

Kameraet blev drejet en smule til denne test, for at forsikre os om, at krydserne ikke bare bliver sat samme sted hver gang. Belysningen er den samme som i C.6.1. På figur C.7 ses kalibreringsbilledet. Da det har krydser i alle fire hjørner og stængerne har påtegnede linjer, fungerer kalibreringen.

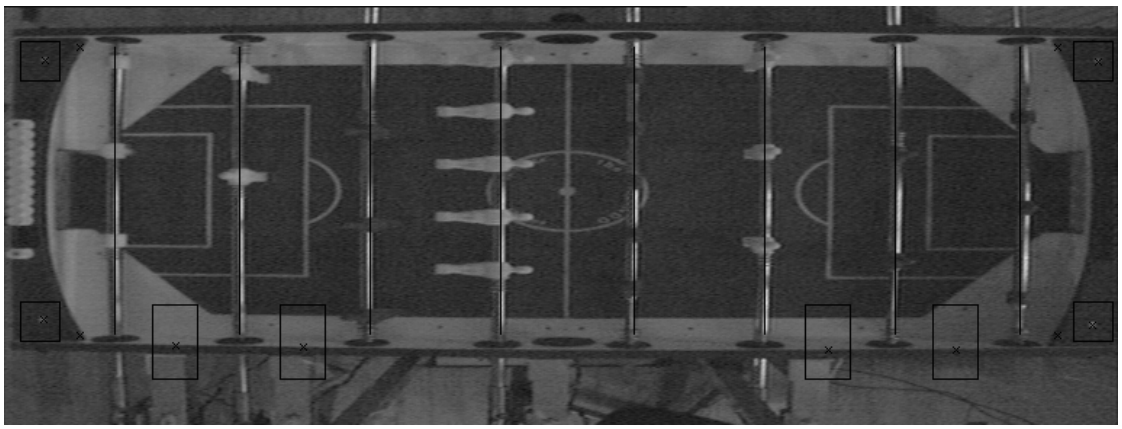
C.6.3 Mørk belysning

Lyset i lokalet er slukket og det eneste lys kommer fra vinduerne som beskrevet i C.6.1. På figur C.8 ses kalibreringsbilledet. Da det har krydser i alle fire hjørner og stængerne har



Figur C.7: Anden kalibrering ved normal belysning

påtegnede linjer, fungerer kalibreringen.



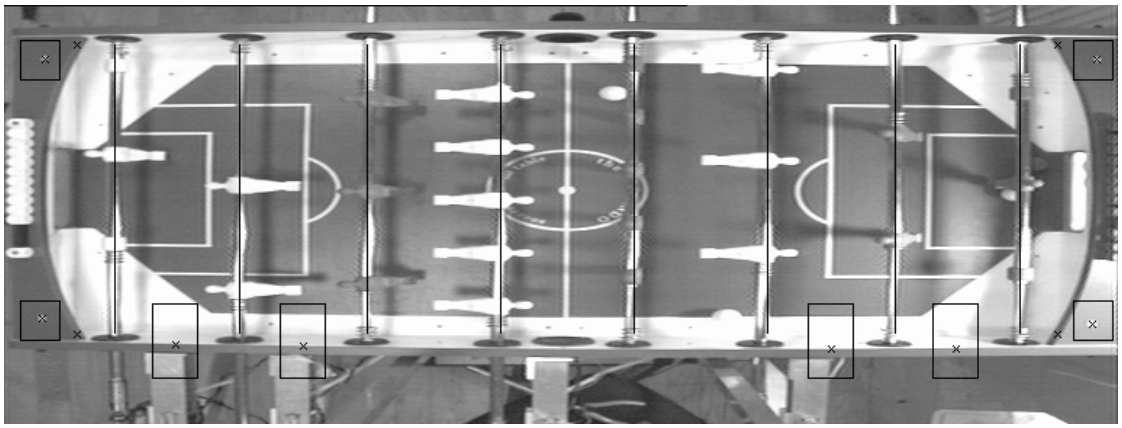
Figur C.8: Kalibrering ved mørk belysning

C.6.4 Lys belysning

Lyset i lokalet er tændt og ved det nederste højre hjørne lyser en en 60 W lampe skråt ovenfra direkte mod klistermærket i 60 cm afstand, for at teste om kalibreringen fungerer ved kraftig belysning. Denne belysning er valgt ud fra at hvis lyset er kraftigere, ser kameraet hele området som hvidt, hvilket umuliggør kalibrering med vores metode. På figur C.9 ses kalibreringsbilledet. Da det har krydser i alle fire hjørner og stængerne har påtegnede linjer, fungerer kalibreringen.

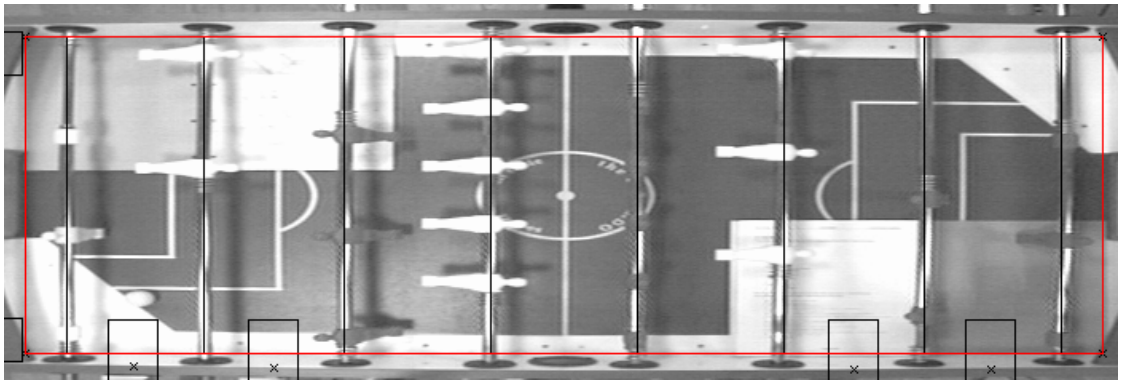
C.6.5 Spillefladens position

Det tjekkes at spillefladens bliver bestemt korrekt ud fra klistermærkernes position. Det gøres ved at lægge papstykker i bordets hjørner og se at farveovergangen mellem lys og mørk på billedet ligger oveni den kalibrerede kant for spillefladen. Denne kant vises med rødt på figur



Figur C.9: Kalibrering ved stærk belysning

C.10. Det ses at pappet slutter, hvor den røde kant går. Altså fungerer omregningen fra klistermærkernes koordinater til spillefladens hjørner.



Figur C.10: Pappet lægges helt op til bordets kanter for at se hvor spillefladen slutter. De slutter samme sted som det røde rektangel, så udregningen af spillefladens position er korrekt.

C.6.6 Konklusion

Kalibreringen fungerer under de lysforhold vi kan komme ud for.

Bilag D

AI

D.1 Modes

D.1.1 DefenseBasic

Som beskrevet i 5.3.2.3, benyttes denne mode til at stille målmanden og forsvaret, så de dækker for bolden. Den funktionalitet, der er beskrevet i omtalte afsnit er simplificeret en del og her er detaljerne.

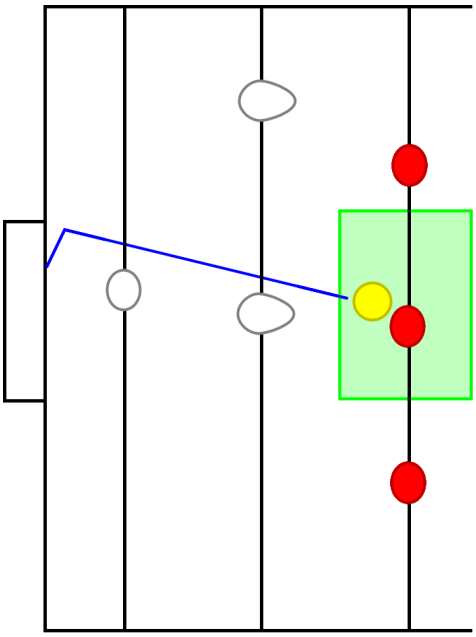
D.1.1.1 Når mændene stilles foran bolden

Uanset situationen, opdækkes der altid med samme forsvarsmand, nemlig den tættest på robotten selv. Det gøres for at undgå det skift af mand, der ellers uundgåeligt må ske en gang imellem. Når der skiftes mand, vil der jo være uopdækket et kort stykke tid, hvilket modstanderen kan udnytte.

Baseret på hvordan vi selv spiller, opdeles opdækningen i 5 cases afhængigt af boldens position: 2 yderzoner, en midterzone og 2 zoner beliggende mellem midterzonen og yderzonerne.

I midterzonen vist med grønt på figur D.1, dækker forsvaret og målmanden lidt forskudt op foran bolden. Forsvaret er drejet lidt fremad, for at dække en større vinkel for bolden. Der opdækkes altid med den af forsvarsmændene der er tættest på robotten selv, som vist på figuren. Modspilleren kan, ved at ramme den ene stolpe, være heldig at den derefter bander ind i målet, men ellers er der ingen huller.

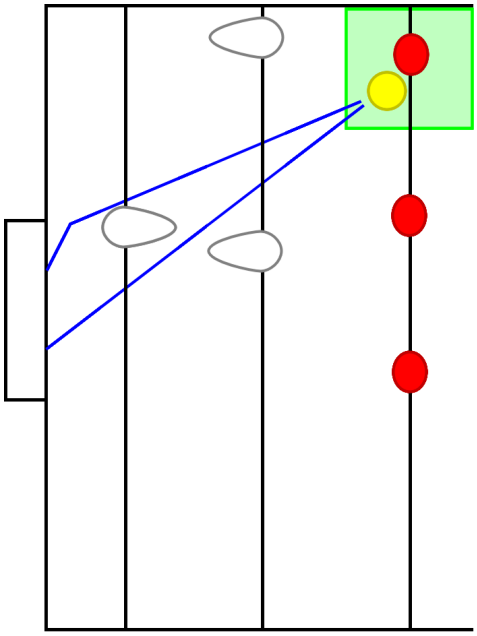
I yderzonerne dækkes med forsvaret vendt bagud og målmanden vendt fremad. Da dækkes



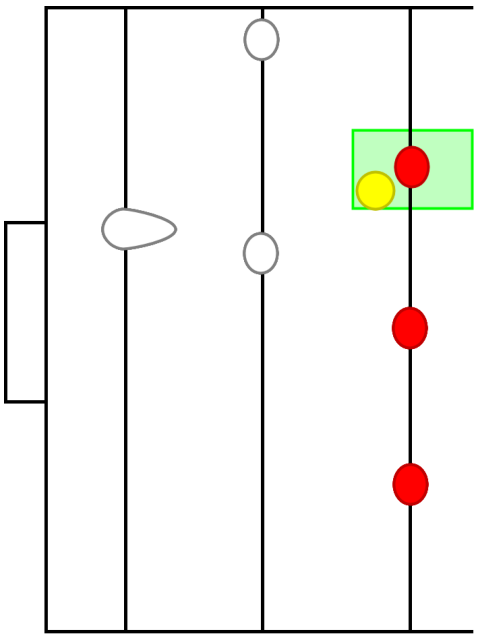
Figur D.1: Opdækning af midterzonen. Det eneste mulige direkte skud på mål er vist med en blå linje.

det meste af målet. Dog er der et lille hul mellem fødderne på de to mænd, hvilket man kan ramme, hvis man er rigtig god eller heldig. Desuden kan man med samme dygtighed eller held også ramme stolpen bag målmanden og derved bande den i mål. For begge dele gælder, at hvis det skuddet er for langsomt, kan målmanden nå at stille sig i vejen og blokere bolden når den kommer. Selvom der er små huller i denne opdækning, er det altså ikke et stort problem. Den ene yderzone er vist på figur D.2 og den anden ser tilsvarende ud, men situationen er dog ikke helt spejlvendt, da der opdækkes med samme forsvarsmand i begge situationer.

De to mellemzoner er karakteriseret ved, at mændene står i samme position på y-aksen, som de gør i yderpositionerne. Dog justeres vinklen på forsvarsmændene således, at der ikke er huller i opdækningen. Dvs. at yderst i mellemzonen er manden drejet bagud, ligesom i yderzonen, og inderst i mellemzonen er den drejet fremad, ligesom i midterzonen. I positionerne imellem disse benyttes mandens position på y-aksen til at lave en lineær interpolation mellem disse to vinkler og denne vinkel benyttes af manden. Området det drejer sig om er vist på figur D.3.



Figur D.2: Opdækning af den ene yderzone, vist med grønt. De to mulige direkte skud på mål er vist med en blå linjer.



Figur D.3: Opdækning af den ene mellemzone, vist med grønt. Ingen direkte skud på mål er mulige.

Bilag E

Simulator

Simulationen af bordfodboldbordet er beskrevet i detaljer her.

E.1 Bordets fysiske tilstand

Vi definerer bordets fysiske tilstand som

- boldens position i 2 dimensioner
- boldens hastighed i 2 dimensioner
- robottens stængers positioner og vinkler
- modstanderens stængers positioner

For en realistisk fysisk model ville tilstanden yderligere inkludere boldens position og hastighed i opadgående retning og dermed blive en 3D simulation. Boldbevægelser i denne dimension er dog ikke nær så interessante som dem i x,y-planet. I vores model flytter bolden sig derfor kun i et enkelt plan og hopper aldrig, som det ellers ofte sker i virkeligheden.

Tilstanden kunne også inkludere boldens rotationshastighed, men det har vi approksimeret væk. Det er en meget grov approksimation, da man i spillet tydeligt kan se en effekt af at bolden ruller. I E.3.1 argumenteres der for at simulationen kan bruges på trods af denne udeladelse.

Stængernes bevægelse kunne gøres mere realistisk ved i tilstanden at inkludere stængernes lineære hastighed og deres drejehastighed. Da kunne modellen opdatere deres hastigheder på

baggrund af accellerationer. Det gør vi ikke, men opdaterer blot positionerne på baggrund af en konstant hastighed. Accelerationen foregår altså momentant i vores model. Dette er en nem løsning, men ser realistisk ud, da mændene i virkeligheden accellererer meget hurtigt og bevæger sig lidt i ryk.

Vi går i simulationen helt udenom billedgenkendelsen. Simulatoren kunne godt generere billeder til billedgenkendelsesalgoritmerne, men billedgenkendelsen testes lettest med rigtige billeder fra kameraet.

E.2 Opdatering af tilstanden

Programmets hovedløkke kører 50 gange i sekundet. Hver gang opdateres den igangværende simulation med AI'ens ønsker om stængernes positioner. Herefter opdateres bordets fysiske tilstand vha. fremskrivning af differentialligninger efterfulgt af registrering af kollisioner og reaktion på disse. Denne opdatering benytter numerisk løsning af differentialligninger og dens tidsskridt er $\frac{1}{50 \cdot 100} s$. Det er næppe nødvendigt at bruge så små skridt, men da simulationen tager langt mindre tid end boldfindingsalgoritmen, er det helt iorden. Den tager faktisk så meget mindre tid, at vi får programmet til at vente lidt i hver frame, så simultationen ved øjemål kører ca. samme hastighed som det virkelige spil.

Først findes de kræfter $\vec{F}$ der påvirker bolden. Disse er beskrevet i E.3.

Fra fysikken kendes sammenhængene

$$\vec{a} = \frac{\vec{F}}{m} = \vec{v}\,' \tag{E.1}$$

$$\vec{v} = \vec{p}\,' \tag{E.2}$$

hvor m er boldens masse, $\vec{a}$ er dens acceleration, $\vec{v}$ dens hastighed og $\vec{p}$ positionen. Til disse koblede differentialligninger benytter vi Eulers metode til at finde en numerisk løsning. Man kunne også have brugt mere præcise metoder, f.eks. Runge-Kutta, men det er ikke besværet værd at implementere. Vores model indeholder så grove approksimationer, at den mistede præcision ved brug af Euler ikke betyder noget. Desuden er der jo tid til at køre med små tidsskridt.

Når boldens nye position er fundet, tjekkes der for kollisioner med bordets sider og med mændene. Hvordan kollisioner detekteres og håndteres er beskrevet i E.4.

Hvis det kunne lade sig gøre at bolden kom i klemme i vores model, så kan reaktionen ikke være så simpel som vi har lavet den. Der skal i så fald tages højde for, hvilke vinkler bolden bliver klemt med, statiske og dynamiske friktionskoefficienter og meget andet. Det slipper vi for, da bolden ikke kan komme i klemme i vores model. Alle de steder det i princippet kunne ske er disse:

- Det er fysisk umuligt at bolden kan kollidere med to mænd på én gang i planet og vi har derfor også gjort det umuligt i modellen. Dels sidder alle mænd på samme stang længere fra hinanden end en bolds diameter og dels sidder stængerne så langt fra hinanden at to mænd på forskellige stænger ikke kan nå den samme bold, uanset hvilke vinkler de står i.

- Bolden kan heller ikke sidde fast mellem manden og spillefladen i vores model, da vi simplificerer mandens bevægelse til, som boldens, at foregå i planet.

- Yderst på hver stang, mellem mændene og bordets sider er der en fjeder til at forhindre mændene i at ramme siderne. Denne fjeder er længere end boldens diameter og da fjederen i vores model er usammentrykkelig, kan mændene ikke klemme bolden ved bordets sider.

- Målmanden kan i praksis klemme bolden ved målstolpen, hvis han er drejet langt op. Det kan han dog ikke i modellen.

Modstanderens stænger findes i modellen, fordi AI'en reagerer forskelligt, afhængigt af deres positioner. Dog foretager vi ikke kollisionsberegninger for modstanderens stænger, da de stænger for det meste bare er i vejen for bolden, når vi tester AI'en.

E.3 Kræfter

E.3.1 Friktion på bolden

Da tilstanden ikke inkluderer rullehastighed, skøjter bolden i modellen henover banen uden at rulle. Ved høje hastigheder er dette ikke meget forskelligt fra virkeligheden, idet bolden også her skøjter henover banen. Både bolden og underlaget er af glat plastik og der er derfor kun lidt friktion mellem dem. Hvis bolden ikke spinner samtidig, vil kraften have en retning modsat boldens. Men hvis bolden f.eks. spinner til siden, vil kraftens retning være en anden. Under normalt spil spinner bolden ikke så kraftigt at det har nogen nævneværdig effekt. Størrelsen af den dynamiske friktion er

$$F = \mu_d \cdot N$$

hvor N er størrelsen af normalkraften i forhold til underlaget. Når kun tyngdekraften påvirker bolden i nedadgående retning er normalkraftens størrelse $N = m \cdot g$. Vi tager ikke højde for, at mændene i virkeligheden kan presse bolden op- og nedad - de rammer kun bolden i det vandrette plan i modellen.

Ved lave hastigheder har bolden derimod en tendens til at rulle. Når det er tilfældet, optræder rullemodstand i stedet for dynamisk friktion [13]. Denne er også modsat rettet boldens retning og størrelsen af kraften på bolden er

$$F = \mu_r \cdot N$$

Afhængigt af om bolden ruller eller skøjter er der tale om to forskellige friktionskoefficienter. Vi har ikke fundet værdierne af disse konstanter og modellen regner heller ikke ud om bolden ruller eller ej. I stedet har vi opfundet følgende approksimation:

Der er kun én slags friktion og denne er, ligesom både den dynamiske friktion og rullemodstanden, modsat rettet boldens hastighed. Den består af to komponenter - en konstant komponent k_1 og en komponent k_2, som er proportional med boldens fart.

$$F = k_1 + k_2 \cdot |\vec{v}|$$

hvor k_1 og k_2 er positive konstanter. Det kan virke som en underlig approksimering i forhold til at den approksimerer to forskellige konstante friktionskræfter, men boldens bevægelse viser sig at se meget realistisk ud på denne måde. Vi har prøvet at sætte hhv. k_1 og k_2 til 0 og justere på den anden af konstanterne, men det ser forkert ud i forhold til virkeligheden.

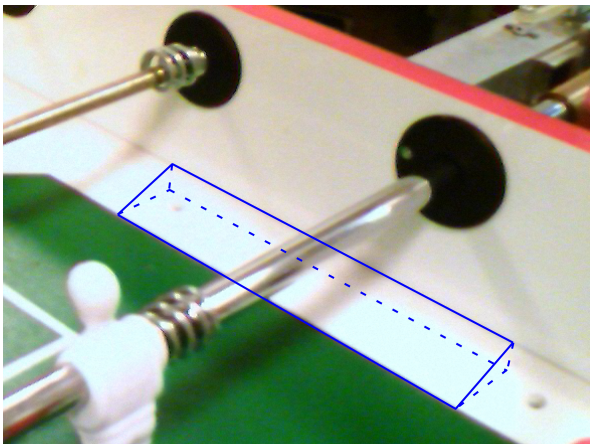
Vi har ikke et godt argument for at approksimationen skulle se sådan ud, men at det ser realistisk ud giver alligevel god mening:

- Når farten er lav, er den approksimerede friktion næsten lig k_1, som svarer til rullemodstanden $\mu_r \cdot N$.
- Når farten stiger, falder sandsynligheden for at bolden ruller og dermed skal der sandsynligvis benyttes dynamisk friktion til udregningen. Da stiger den approksimerede friktion proportionalt med farten og nærmer sig den dynamiske friktion $\mu_d \cdot N$.
- Boldens hastighed vil sjældent blive særlig høj, da den kun accelereres af mændene. Hvis den bliver så høj, at den approksimerede friktion er urealistisk stor i forhold til den rigtige, dynamiske friktion, vil bolden stoppe meget hurtigt, da den rammer en spiller eller bordets sider.

Statisk friktion er implementeret ved at tjekke om boldens fart er meget lille ($|\vec{v}| < 1/10mm/s$), og i så fald sætte farten lig 0. Vi prøvede forskellige værdier større og mindre end denne og fandt at den så mest realistisk ud. Hvis den er sat for høj, stopper bolden for pludseligt og hvis den er for lav, bevæger bolden sig i længere end den gør i praksis. Bolden har en ujævn, mønstret overflade og stopper derfor let, når farten er lav.

E.3.2 Kraft fra skrå områder på bordet

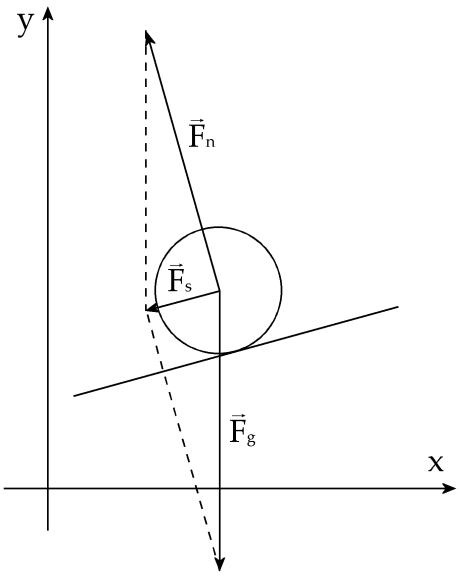
Langs kanten på fodboldbordet er der skrå kanter, der gør at bolden ikke kan ligge stille det pågældende sted, så mændene ikke kan nå den. En af kanterne kan ses på figur E.1.



Figur E.1: En skrå kant nær bordets side. Dens hældning er tydeliggjort vha. blå linjer.

Disse kanter tages der højde for i simulationen, ved at udregne den kraft $\vec{F}_s$, der påføres bolden i retning væk fra bordets side. Denne udregnes vha. kræfternes parallelogram på figur E.2:

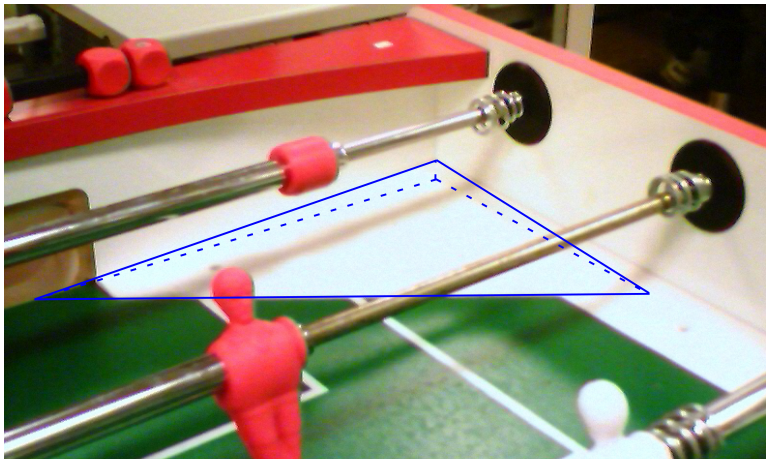
$$F_g = mg \tag{E.3}$$
$$F_n = cos(\alpha)F_g \tag{E.4}$$
$$F_s = sin(\alpha)F_g = sin(\alpha)mg \tag{E.5}$$



Figur E.2: Når bolden er på en skrå kant, påvirkes den med en kraft F_s ind mod bordet.

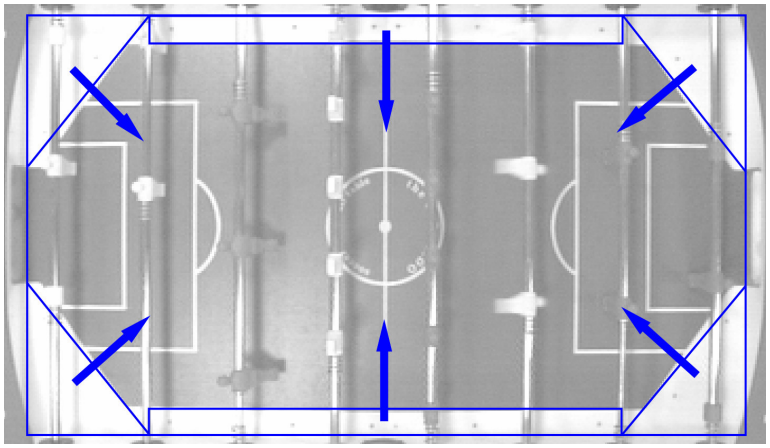
Tilsvarende er der et næsten trekantet område i hjørnet af bordet, som også hælder. Det er vist på figur E.3. Det mangler et hjørne for at være et trekantet område og det har ikke samme hældning over det hele, men vi lader i simulationen som om området er trekantet.

Kraften udad udregnes på samme måde som for de skrå kanter, men kraften er mindre idet hældningen også er det.



Figur E.3: Et skråt område ved bordets hjørne. Hældningen er tydeliggjort vha. blå linjer.

Til at detektere om bolden er indenfor et skråt område undersøges det om boldens midte er indenfor visse rektangler og trekanter på spillefladen. Disses former er forsimplede i forhold til de rigtige områder fordi de helt præcise hældninger ikke betyder noget for brugen af simulationen. Disse områder ses på figur E.4.



Figur E.4: Indenfor de afmærkede rektangler, øverst og nederst, påfører simulationen en kraft på bolden væk fra kanten. Indenfor de afmærkede trekanter i hjørnerne påføres tilsvarende en kraft skråt væk fra hjørnet. Kræfternes retninger er angivet med pile.

Det viser sig at bolden som udgangspunkt accelererer alt for meget når den skøjter op og ned ad de skrå områder, i forhold til virkeligheden. Når bolden med lav fart sættes til at rulle op, vende om og rulle ned ad hjørnernes skrå områder, kan det i hvert fald tydeligt ses. Ved høje hastigheder er kræfterne muligvis mere realistiske - det er ikke til at se. Den for høje acceleration må skyldes, at bolden i simulationen skøjter afsted i stedet for først at rulle den ene vej, bremse op og derefter rulle den anden vej, som det sker i virkeligheden. Rotationsenergien bliver først til kinetisk og potentiel energi, hvilket får den til at rulle højere op, end hvis den blot skøjtede. På vej ned bliver en del af energien tilsvarende til

rotationsenergi igen. Effekten er, at bolden accelererer langsommere end den ville have gjort, hvis den blot skøjtede op og ned. At udregne denne effekt kan gøres med klassisk mekanik, men det gør vi ikke. I stedet multiplicerer vi blot den udregnede kraft med et passende lille tal, så det ser realistisk ud.

E.4 Kollisioner

E.4.1 Generelt om detektion af kollisioner

Bolden vil i løbet af spillet kollidere med bordets sider og med mændene. Vi detekterer om der er kollision, ved at udregne om bolden lapper over med en af bordets kanter eller med af mændenes fødder. Det er en simpel måde at gøre det på. Ved små hastigheder og små tidsskridt i simulationen må metoden være god nok. Ved store tidsskridt eller hastigheder har den dog den ulempe, at bolden kan gå igennem manden, uden det bliver detekteret. Vi målte i [3] hastigheden af både robottens og vores eget hårdeste skud til ca. 7 m/s. Forudsat at bolden ikke bevæger sig hurtigere i simulationen, giver det at bolden i løbet af et enkelt tidsskridt maksimalt rykker sig

$$\Delta s_{bold} = v_{max} \cdot \Delta t = 7\text{m/s} \cdot \frac{1}{50 \cdot 100}\text{s} = 1.4\text{mm}$$

Hvis en mands fødder bevæger sig med samme hastighed den modsatte vej, kan bolden gå igennem en mands fod, hvis denne tykkelse plus boldens diameter tilsammen er mindre end 2.8 mm. Da de tilsammen er over 4 cm, er vi på den sikre side. Dog kan de ske, at en bold som burde snitte en mands hjørne ikke bliver detekteret som kollision. Men det er ligegyldigt i forhold til de andre approksimationer vi foretager.

Når en kollision detekteres, flyttes bolden lidt væk, så den ikke længere overlapper med manden eller bordets side. Derefter udregnes udgangshastigheden for bolden, og simulationen fortsætter.

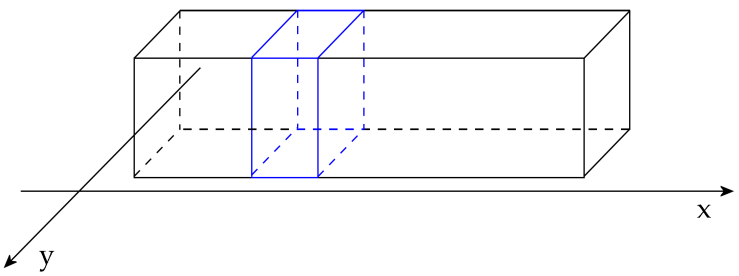
E.4.2 Kollision mellem bordets sider og bolden

Kollision mellem bolden og bordets sider er håndteret simpelt. Bolden rykkes ud på den rigtige side af muren, så der ikke længere er kollision. Derefter vendes den af hastighedens komponenter, der peger ind mod muren, til at pege udad. Den multipliceres med en konstant for at miste fart. Det er ikke en præcis model af hvordan kollisioner foregår, men det er også uvæsentligt. Det ser realistisk ud og er let at lave.

E.4.3 Kollision mellem mand og bold

Når en mand og en bold kolliderer, skulle bolden gerne ændre hastighed. I virkeligheden vil bolden også påvirke manden og derved få denne til at ændre hastighed, men det ser vi bort fra, da mandens hastighed efter den har ramt bolden, ikke er vigtig.

Da bolden i vores model bevæger sig langs jorden, kan bolden ikke ramme andre dele af manden end hans fod og dette kun når foden er tilstrækkeligt tæt på jorden. Foden drejer sig i forskellige vinkler i virkeligheden, men det vælger vi at se bort fra. I stedet bevæger foden sig vandret som en kasse i vores model, da det er nemmere at foretage beregninger sådan. På figur E.5 ses foden som den blå kasse. Når stangen sidelæns står stille, men roterer, bevæger foden sig i vores model i det afmærkede område. Når foden pga. dens vinkel i virkeligheden når så højt op, at den ikke kan nå bolden, er den i vores model helt ude i en af enderne af det område, den kan bevæge sig i. Der vælger vi at lade den forsvinde fra kollisionsberegningerne, sådan at en mand der er vippet op, ikke kan røre bolden.



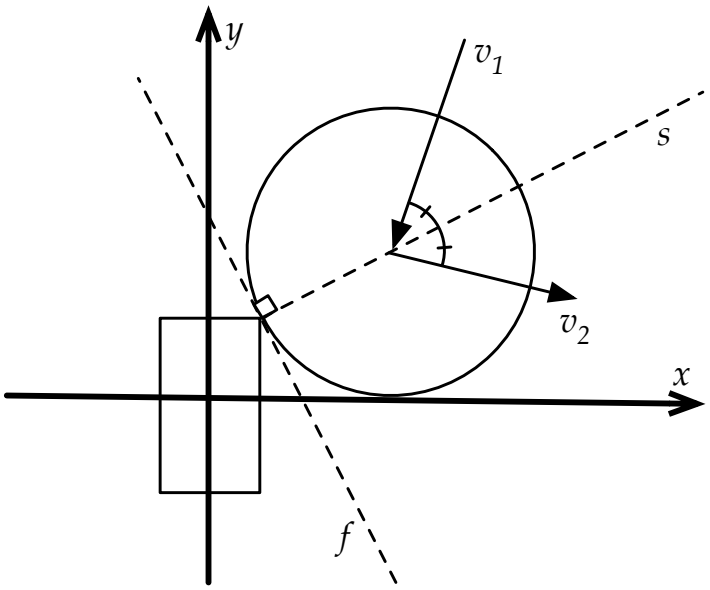
Figur E.5: Når stangen sidelæns står stille, kan mandens fod (blå) i vores model bevæge sig i det afmærkede område

I modellen antager vi, at manden ikke er til at rokke i forhold til bolden, når de kolliderer. Altså kan et bevægende koordinatsystem K defineres med midten af mandens fod som centrum inden kollisionen. Foden vil da forblive at være centrum i dette koordinatsystem efter kollisionen. Ved at regne boldens koordinater og hastighed om til K, kan de følgende beregninger foretages, som om mandens fod stod stille. Derefter omregnes boldens hastighed igen til bordets koodinatsystem. I det følgende forudsættes det, at K benyttes.

Når der er en kollision mellem manden og bolden, afhænger boldens reaktion af, hvor bolden rammer manden. Hvis den rammer denne på et fladt stykke, vælger vi at reaktionen skal foregå som i reaktionen mellem bolden og spillets sider, som beskrevet i E.4.2, altså stort set med indgangsvinkel lig udgangsvinkel.

Når bolden rammer et hjørne, bliver beregningerne lidt sværere. Situationen er som vist på figur E.6. Bolden rører et enkelt punkt på hjørnet af mandens fod, men der er ingen forskel på dette og på at ramme en flade f gående igennem dette punkt, tangentielt på boldens overflade. Udgangshastigheden må være den samme. Vi gør beregningerne så simple som muligt, ved at antage at udgangsvinklen her er lig indgangsvinklen om aksen s. Dog må udgangsfarten være mindre, så denne skaleres ned med en konstant faktor i forhold til indgangsfarten.

I praksis finder vi udgangsvinklen således: Den vektor der udspændes fra kontaktpunktet



Figur E.6: Ved kollision mellem bolden og et hjørne af mandens fod (rektangel), reflekteres bolden som om den havde ramt en flade f.

med manden til boldens midtpunkt, ligger oveni spejlingsaksen. Lad dennes vinkel hedde α. Vinklen for indgangshastighedens modsatte retning, kaldes β. Når forskellen lægges til α findes udgangsvinklen γ, altså

$$\gamma = \alpha + (\alpha - \beta)$$

Når bolden har fået sin nye vinkel og fart og dermed sin nye hastighed v_2, omregnes denne fra K tilbage til bordets koordinatsystem.

E.4.4 Støj

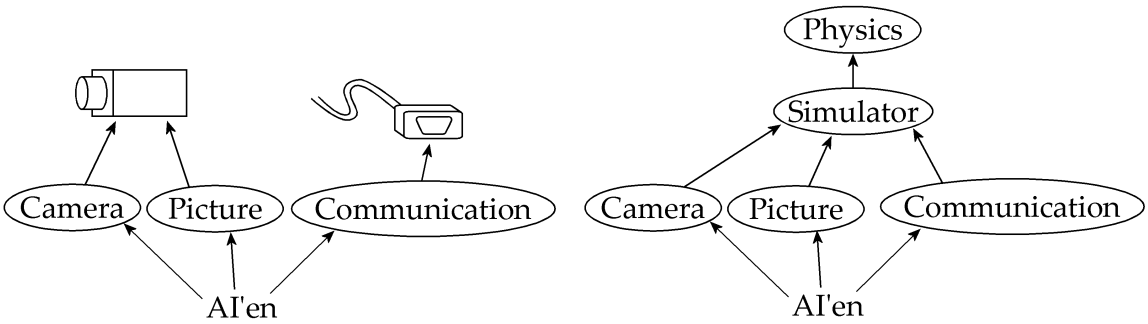
Når AI'en spørger simulatoren om boldens position, kan simulatoren sættes til at tilføje støj til svaret. Derved simuleres den støj der er i forbindelse med boldgenkendelsen. Vi modellerer støjen på x-aksen som en uniform fordeling i intervallet -0.8mm til 0.8mm. Vi har ikke undersøgt, hvilken type fordeling støjen har i virkeligheden, men valgte denne type, da den er lettest at implementere og fordi det ser realistisk ud, når man ser simulationen gennem brugergrænsefladen.

Intervallets størrelse blev bestemt således: En bold lå stille på bordet og dens position blev fundet af programmet. Via brugergrænsefladen så vi støjens størrelse. Samtidigt kørte simulationen på en anden maskine og ved at forbinde en brugergrænseflade til denne, kunne vi sammenligne støjens størrelse i virkeligheden og på simulationen. Vi prøvede forskellige værdier med 0.2 mm forskel og fandt, at udsving på maksimalt 0.8 mm til hver side så mest realistisk ud.

E.5 Implementering

Vi har ikke gjort det muligt at slå simulatoren til og fra efter programmet er kompileret. I stedet bruges en `#define` til at slå simulatoren til, inden der kompileres. Det er lidt mere besværligt, men er i orden, da vi ikke slår den til og fra særligt tit.

De involverede klassers sammenhæng er vist på figur E.7.



Figur E.7: Uden simulator kommunikerer `Camera` og `Picture` med kameraet og `Communication` benytter serielporten, som vist til venstre. Med simulator benytter de `Simulator`, som igen benytter `Physics`. I begge tilfælde er AI'en den samme.

Når simulatoren er slået til, vil de klasser som normalt benytter kameraet og motorerne (`Camera`, `Picture` og `Communication`) snakke med simulatoren i stedet. Altså er der for AI'en ingen forskel på, om den bruger kameraet og motorerne, eller om den bruger simulatoren.

Selve den fysiske simulation og information om bordets simulerede fysiske tilstand ligger i `Physics` klassen. `Simulator` klassen er den eneste klasse, der tilgår denne direkte. Den sørger bl.a. for trådsikkerhed, dvs. den undgår race conditions ved brug af semaforer.

E.6 Test

Vi har ikke lavet objektive tests af, hvor realistisk vores simulation er. Dele af simulationen kunne testes ved at sætte en bold i bevægelse på det fysiske bord og optage bevægelsen. Derefter kunne man starte en simulation med de samme startbetingelser og over tid sammenligne afvigelsen fra den fysiske optagelse. Det skulle gøres mange gange ved forskellige startbetingelser, for at være statistisk signifikant. Det har vi prioriteret så lavt, at det ikke er blevet gjort.

I stedet har vi overalt brugt øjemål til at se om simulationen så realistisk ud. Det er godt nok, da simulationen først og fremmest er et hjælpeværktøj til udvikling af AI'en.

At simulationen er præcis nok, har vi desuden bagefter kunnet verificere ved at den AI vi testede med simulatoren opfører sig stort set på samme måde i virkeligheden.

E.7 Konklusion

I robottens hovedprogram har vi tilføjet en 2D fysisk simulation af bordfodboldspillet. Her kan man teste AI'en uden de fejlkilder der bl.a. er i forbindelse med billedgenkendelsen. Mange steder er der foretaget grove approksimationer i forhold til virkeligheden, men simulationen er alligevel præcis nok til formålet.

Bilag F

Kampresultater

Spilleresultaterne er vist i denne tabel. I resultatkolonnen er robottens point vist først, derefter modstanderens point.

Resultat	Navn	Alder	Spillefrekvens	Egen mening	Dato
10 - 5	Christian Flintrup	B	2/måned	3	9/1-07
10 - 2	-	-	-	-	27/1-07
10 - 6	-	-	-	-	27/1-07
10 - 9	Miya Cara Akselgaard	B	4/måned	6	9/1-07
10 - 3	Nils Andersen	C	2/år	5	11/1-07
10 - 5	Farhad Shariti	B	1/uge	7	12/1-07
7 - 10	Lars Pontoppidan	B	3/måned	8	12/1-07
10 - 2	Izmir Gvozbar	B	1/måned	3	12/1-07
9 - 10	Jakob Ravn	B	1/3. måned	5	12/1-07
10 - 9	Morten Rufus Bias	B	1/uge	8	15/1-07
8 - 10	Jacob Remin	B	1/måned	5	15/1-07
10 - 2	Alice Mao	B	2/år	1	15/1-07
5 - 10	Martin Ørding-Thomsen	B	1/uge	9	15/1-07
4 - 10	-	-	-	-	27/1-07
4 - 10	Anders Myrup	B	1/uge	9	16/1-07
6 - 10	-	-	-	-	27/1-07
10 - 5	Allan Bertelsen	B	1/måned	5	17/1-07
6 - 10	Aske Olsson	B	flere/uge	8	18/1-07
10 - 4	Aske Olsson	B	flere/uge	8	26/1-07
10 - 9	Mads Ingvar	B	1/måned	5	18/1-07
10 - 2	Anders Beck	B	2/år	2	18/1-07
7 - 10	Niklas Boss	B	hver dag	10	18/1-07
10 - 9	Kenneth Ahrensberg	B	2/måned	5	19/1-07
10 - 9	Martin Söderlind	B	2/måned	5	19/1-07
10 - 6	Emil Blach og Nikolaj Møller-Madsen	A	N/A	4 og 3	23/1-07
10 - 6	Rasmus Tulinius og Mads Albech Christensen	A	N/A	6 og 5	23/1-07
10 - 3	Peter Kjærulff	B	1/2. måned	2	23/1-07
10 - 3	-	-	-	-	23/1-07
10 - 4	Bill Riis Ebbesen	B	1/10. år	1	27/1-07
10 - 3	2 * 5. klassebørn	A	N/A	N/A	25/1-07
10 - 5	2 * 5. klassebørn	A	N/A	N/A	25/1-07
10 - 4	2 * 5. klassebørn	A	N/A	N/A	25/1-07
10 - 5	2 * 5. klassebørn	A	N/A	N/A	25/1-07

For langt de fleste kampe er der indsamlet data om modspillerne. De steder, det ikke er gjort er angivet med N/A.

Bilag G

Kildekode

Koden for de mest interessante klasser i hovedprogrammet ligger her. Den komplette kildekode til hovedprogrammet, motor-controllerne og klienten findes på den vedlagte CD.