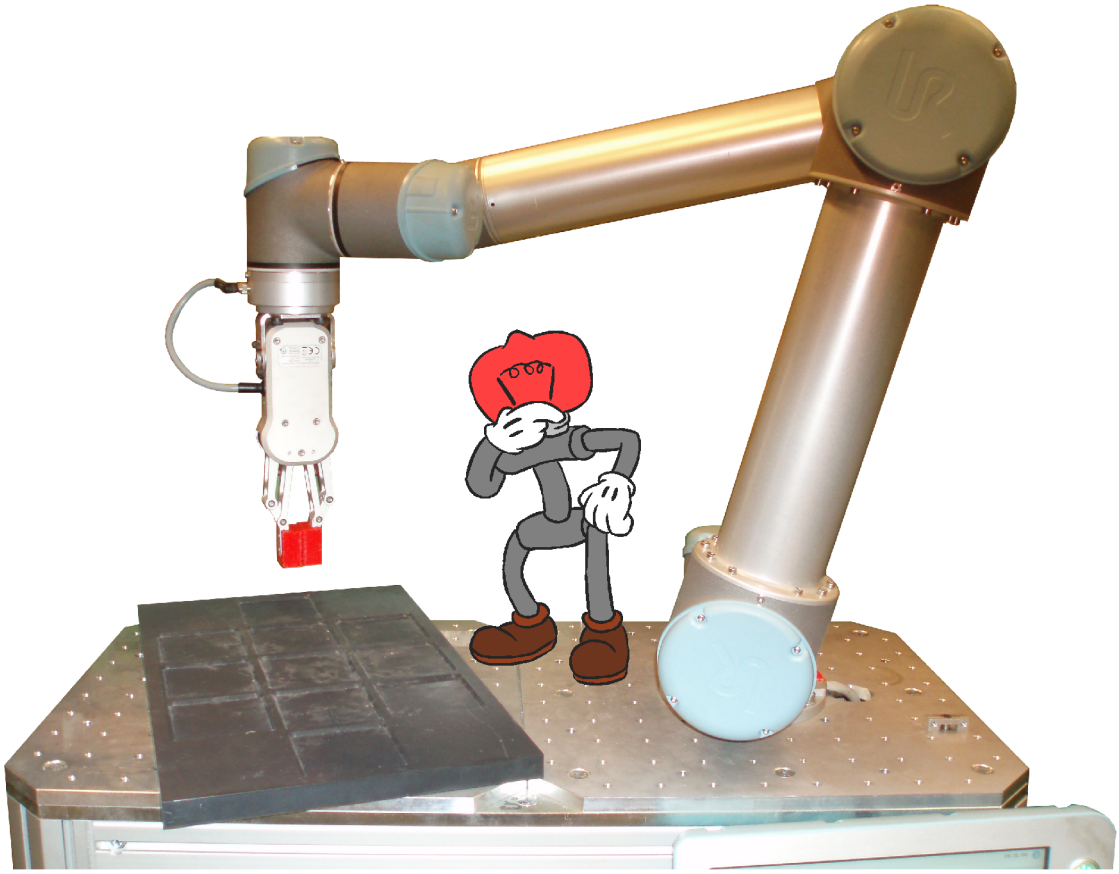
Den lille modstander



Virksomhedsteknolgi - VT4

Michael Fly Pedersen

AALBORG UNIVERSITET

Titel: Den lille modstander

Semester: VT4
Projekt periode: 1. Okt. - 1. Feb 2017

Vejleder: Ole Madsen
Bivejleder: Rasmus Skovgaard Andersen

Michael Fly Pedersen

Oplagstal: 4
Bilagsantal og -art: 1 CD
Antal sider eksl. appendiks: 57
Antal sider inkl. appendiks: 62

Synopsis:

Dette projekt tager udgangspunkt i en robot der ønskes at blive gjort i stand til at spille kryds og bolle med brugeren.
Denne rapport vil af denne grund gennemgå de enkelte trin der er nødvendige for at løse denne opgave.
Som første punkt kræver robotten en strategi til at spille kryds og bolle, hvorfor denne vil blive analyseret og implementeret i robotten.
Til at styre robotten gøres der brug af Matlab, hvorfor programmerne der tillader PolyScope og Matlab at kommunikere herefter vil blive gennemgået.
Dette er efterfulgt at den fysiske kalibrering af robotten.
Som sidste del bliver det gennemgået hvordan kameraet kalibreres, samt processen med at identificere de enkelte brikker på spillebrættet.
Ved projektets afslutning var robotten i stand til at spille kryds og bolle med brugeren, dog i et begrænset omfang, ved blot at observere hvilke træk brugeren lavede.

AALBORG UNIVERSITET
Studienævnet for Industri og Global Forretningsudvikling
Fibigerstræde 16
DK 9220 Aalborg Øst
Tel +45 99 40 93 09 Fax +45 98 15 16 75
swe@me.aau.dk http://www.industri.aau.dk/

Indhold

Kapitel 1

Indledning

1.1 Indledning

Ny teknologi bliver ofte fremstillet på messer, hvor de enkelte firmaer konkurrerer om at tiltrække gæsternes opmærksomhed.

Nogle firmaer vælger at demonstrere deres robotters kapabilitet ud fra simulering af normal arbejdsgang, hvor arbejdsmønstret og emnerne, robotten arbejder med, er lette at visualisere i en reel produktion. Et eksempel på denne type demonstration kan ses på fig: 1.1, hvor robotten anvendes til at identificere emner og løfte dem op på et transportbånd.

Figur 1.1: *Eksempel på demonstration der simulere en produktionslinje. [Youtube Klip 1]*

Andre firmaer benytter mere kreative opstillinger, hvor fokus er at demonstrere robottens egenskaber frem for at vise hvad den kan benyttes til. En demonstration der fokuserer på robottens hastighed vil dermed have robotten til at bevæge sig imellem en række punkter, så hurtigt som muligt, uden at udføre en egentlig arbejdsopgave. På fig: 1.2 på næste side vises et eksempel på denne type demonstration. Her flyttes, og roteres, arbejdsfladen hurtigt imellem de 2 robotter, hvilket demonstrere præcisionen ved at sænke en cylinder ned i hullerne på den.

Endeligt er der firmaer, der forsøger at interagere med gæsterne. For nogle firmaer vil denne interaktion være begrænset til at vælge robottens arbejdsopgave, mens andre opstiller deciderede konkurrencer imellem robotten og gæsterne. Et eksempel på denne type demonstration kan ses på fig: 1.3 på den følgende side, hvor robotten spiller kryds og bolle imod gæsterne.

En af de robotter, der er i stand til at interagere med gæsterne, kaldes Baxter. Denne robot er fremvist i Chicagos museum for videnskab og industri.

Det blev observeret at der var lavet flere videoer, hvori Baxter enten lavede strategiske fejl eller brød spillets regler. Udsnit fra disse videoer kan ses på fig: 1.4 på næste side.

Figur 1.2: *Eksempel på demonstration der viser robottens egenskaber. [Youtube Klip 2]*

Figur 1.3: *Eksempel på demonstration der interagere med gæsterne. [Youtube Klip 3]*

(a) *[Youtube Klip 4]*

(b) *[Youtube Klip 5]*

Figur 1.4: *Eksempler hvilke typer fejl Baxter laver.*

1.2 Initierende problemstilling

Det var ud fra ønske om at forbedre Baxters præstation, at det initierende problem blev opstillet:

Hvordan kan en robot blive i stand til at spille kryds og bolle med brugeren?

1.3 System beskrivelse

Til at løse denne problemstilling blev en opstilling stillet til rådighed, hvilket igennem denne rapport vil blive henvist til som "den lille modstander" .

Dette system, som vist på fig: 1.5 på den følgende side, vil blive beskrevet for at identificere dets evner og begrænsninger.

1.3.1 UR5 robot

Robotten, der anvendes af "den lille modstander" , er lavet af firmaet Universal Robots og går under navnet UR5 på grund af dens last kapacitet på 5 [kg]. [UR5 User Manual]

Denne type robot er en såkaldt kollaborativ robot, hvilket betyder at den er i stand til at måle eksterne kraftpåvirkninger. For robotter produceret af Universal Robots sker dette ved at måle de interne kraftpåvirkninger der er krævet for at holde en given position, frem for brug af eksterne sensorer.

Da robotten ikke vil være afskærmet, giver dette en ekstra sikkerhedsfaktor eftersom den vil være i stand til at gå i nødstop ved for høje kraftpåvirkninger.

Udover dette automatiske nødstop, vil dens kollaborative egenskaber ikke blive undersøgt nærmere.

Som vist på fig: 1.6 på side 5 er robotten der anvendes til "den lille modstander" fastspændt på en vogn, hvilken virker som robottens arbejdsflade.

1.3.2 RG2 griber

Griberen, der er påmonteret robotten, er lavet af firmaet On Robot og går under navnet RG2. [RG2 User Manual]

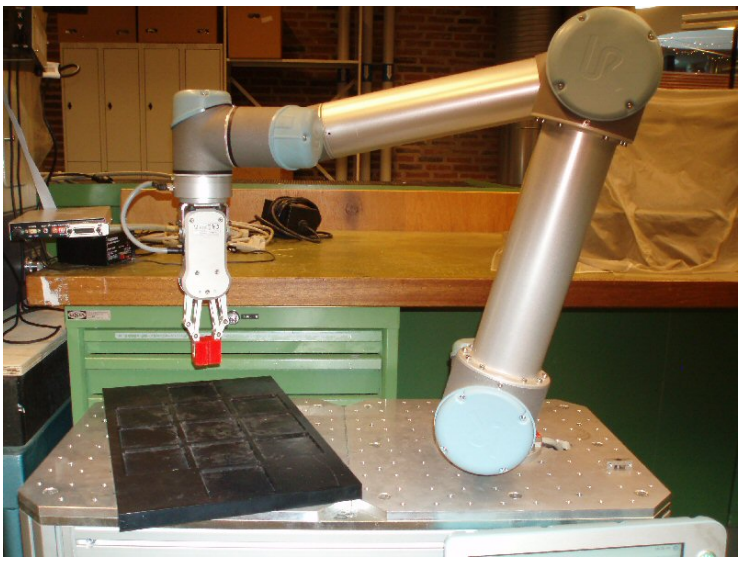
Denne griber er designet til at blive tilsluttet UR robotter og er derfor let at tilslutte samt at styre via UR interfacet.

Fingrene, der benyttes af denne griber, er konstrueret af Aalborg Universitet.

På grund af at disse fingre har andre dimensioner end dem der medfulgte griberen, skulle der tages højde for den ændrede gribe bredde.

Griberens udformning, som vist på fig: 1.7 på side 5, gør at dens gribepunkt varierer afhængigt af hvilken gribe bredde et givent emne kræver. Eftersom tappen på alle brikkerne har de samme dimensioner, var det ikke nødvendigt at tage højde for det varierende gribepunkt.

Figur 1.5: *"den lille modstander"* .



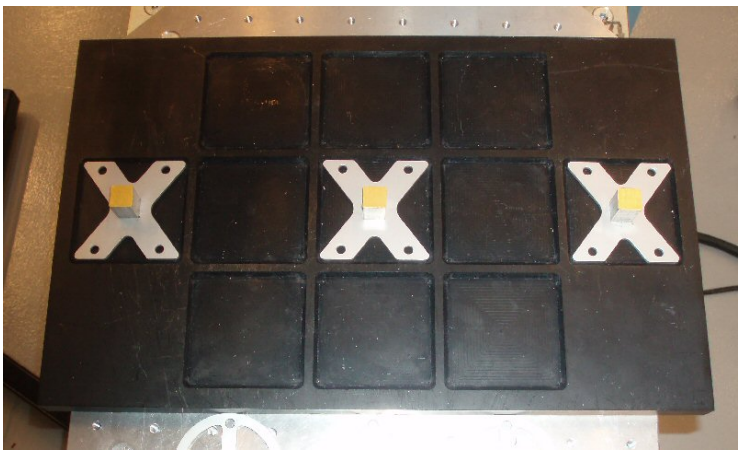
Figur 1.6: *UR5 robotten og den vogn den er fastspændt på.*

(a) *Åben* **(b)** *Lukket*

Figur 1.7: *RG2 griberen i åben og lukket tilstand.*

1.3.3 Spillebræt

Spillebrættet der anvendes af "den lille modstander" er konstrueret af Aalborg Universitet.



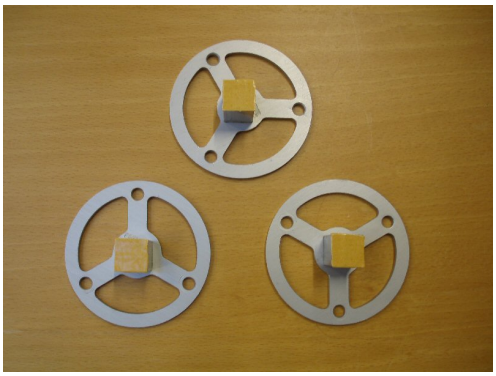
Figur 1.8: *Spillebrættet der anvendes af "den lille modstander"* .

Som vist på fig: 1.8 har spillebræt 11 udfræsede felter, hvoraf 9 af disse er arrangeret i et 3×3 gitter. Det er dermed anvendt i projektet under antagelsen af at kryds og bolle er et tilstrækkeligt kendt, hvorfor de 2 ekstra felter ikke vil skabe forvirring.

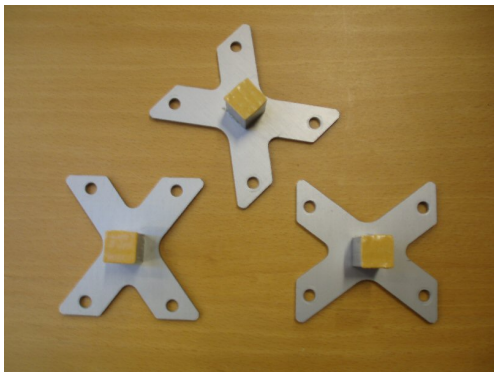
Spillebrættet er ikke fastspændt på "den lille modstanders" bord, eftersom det er lavet ud af en tyk metalplade. På grund af dets vægt blev det vurderet at friktionen imellem spillebrættet og bordet var tilstrækkelig høj til at det ikke ville blive flyttet ved et uheld.

1.3.4 Brikker

Brikkerne, der anvendes af "den lille modstander" , er ligeledes konstrueret af Aalborg Universitet. Begge typer brikker kan ses på fig: 1.9.



(a) *Cirkel*



(b) *Kryds*

Figur 1.9: *Brikkerne der anvendes af "den lille modstander"* .

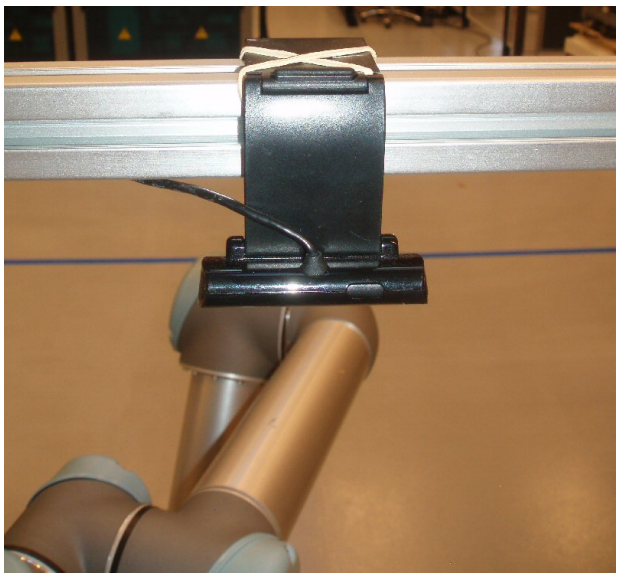
På grund af at der kun var 3 brikker af hver type var "den lille modstander" begrænset til at spille variationen af kryds og bolle der fortsætter til en spiller har vundet. Dette kan i sig selv anses som en fordel, eftersom der af denne grund ikke vil være forvirring om hvilken variation der spilles.

På grund af deres blanke overfalde var det nødvendigt at påhæfte markeringstape på toppen af den tap brikkerne løftes med. Dette skyldes at brikkerne er meget blanke, hvilket gav problemer med at identificere orienteringen af deres tap. Uden denne ændring ville "den lille modstander" dermed ikke have været i stand til at identificere orienteringen af de cirkelformede brikker.

1.3.5 Kamera

Kameraet der anvendes af "den lille modstander" er et standard 720p web kamera.

Dette kamera er opspændt på et stativ, som vist på fig: 1.10, således det er placeret over spillebrættet.



Figur 1.10: *Det fastspændte kamera der anvendes af "den lille modstander" .*

Denne opstilling blev vurderet til at være tilstrækkelig for projektets formål, men blev også anset som værende sårbar over for at blive flyttet ved et uheld. Af denne grund blev det vurderet at "den lille modstander" skulle være i stand til selv at identificere spillebrættets position i billedet, for at simplificere denne process.

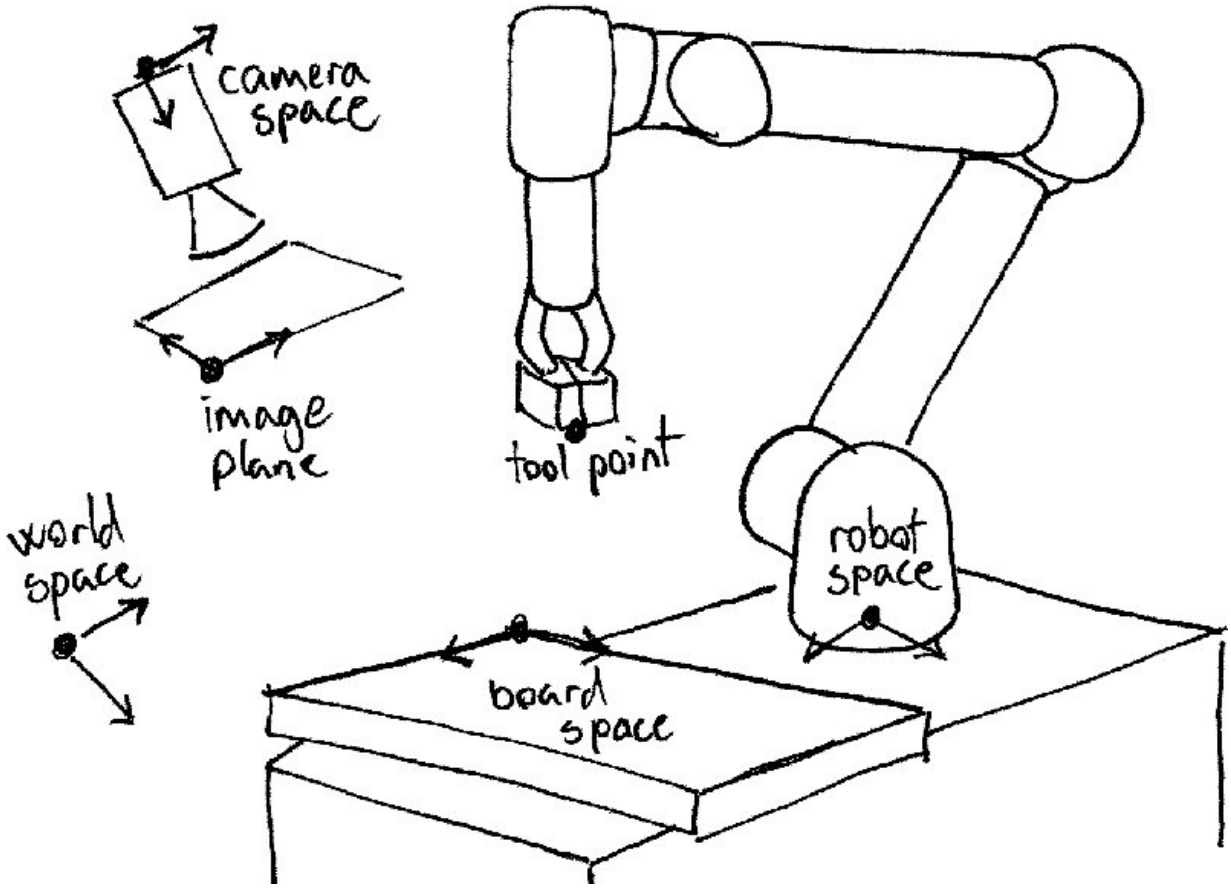
1.3.6 Samlet system

På grund af at "den lille modstander" indeholder flere komponenter er det nødvendigt at kunne konvertere imellem de individuelle koordinatsystemer. En illustration af "den lille modstander" og disse koordinatsystemer kan ses på fig: 1.11 på den følgende side.

1.4 Kravspecifikationer

For at sikre at have et endeligt produkt ved projektets afslutning blev kravspecifikationerne opdelt i flere niveauer.

På det laveste niveau ønskes det at opfylde de absolut minimumskrav for at have hvad der kaldes et *minimal viable product*. Dette betyder at det antages at brugeren opføre sig ordenligt, samt at personalet



Figur 1.11: *Skitse over de anvendte koordinatsystemer.*

er ekstra opmærksomme under opstillingen af “den lille modstander” .

Som et minimum skal “den lille modstander” :

- Være i stand til at benytte perfekt strategi til at spille kryds og bolle.
- Kunne styres via Matlab.
- Være i stand til at skelne imellem brik typer.
- Være i stand til at identificere spillets tilstand.

På niveauet over minimumskravene ønskes det at interaktionen med “den lille modstander” virker naturligt. Dette betyder at “den lille modstander” skal være i stand til at spille kryds og bolle uden input fra bruger eller personale. Yderligere ønskes det på dette niveau at simplificere og automatisere dele af opstillings proceduren.

På dette niveause skal “den lille modstander” :

- Være i stand til at identificere spillebrættets position og orientering.
- Være i stand til at identificere brikkernes position og orientering.
- Kunne spille kryds og bolle uden input fra brugeren.
- Kunne styres af Matlab som en kontroller.

På det højeste niveau antages det ikke længere at brugeren vil opføre sig ordentlig. Det er normalt at være nysgerrig, hvorfor dette niveau kræver at brugeren skal være i stand til at teste “den lille modstanders” grænser.

Hvis dette niveau opnås, skal “den lille modstander” :

- Kunne håndtere at brugeren snyder i kryds og bolle.
- Kunne håndtere at brugeren skubber til robotten.
- Være i stand til at automatisk korrigere for at kameraet bliver flyttet.
- Være i stand til at automatisk korrigere for at spillebrættet bliver flyttet.

1.5 Afgrænsning

Der vil, i denne rapport, ikke blive taget højde for de ekstra sikkerhedsforhold der er nødvendige når uerfarne brugere interagere med en robot.

Dette projekt antager af denne grund at der altid vil være en erfaren bruger til stede, der er i stand til at afværge potentielle farlige situationer samt være i stand til at benytte robottens nødstop.

Alment gældende sikkerhedsforanstaltninger for brug af robotter uden afskærmning er dog benyttet, såsom sænket hastighed, minimering af arbejdsområde og afstand fra flader hvor personer kan komme i klemme.

Kapitel 2

Strategi

I dette kapitel vil strategien “den lille modstander” gøre brug af blive udarbejdet.

Det skal her bemærkes at denne del af strategien kun omfatter planlægningen af hvert enkelt træk. Hvordan “den lille modstander” registrerer hvis tur det er, vil blive gennemgået i senere kapitler.

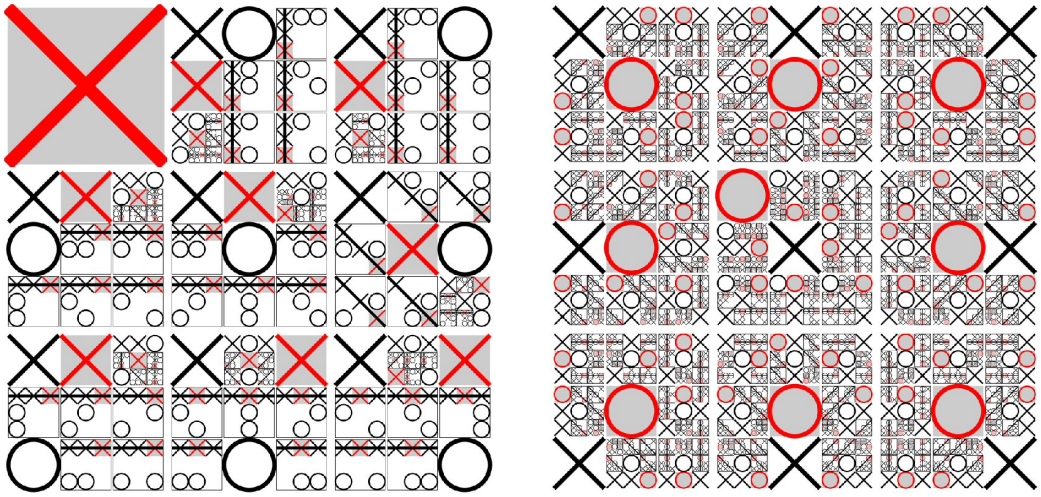
2.1 Metodik

Kryds og bolle er hvad der kaldes et løst spil, hvilket betyder, at ved spillets start er det kendt hvad resultatet vil være, hvis alle spillere spillede perfekt. Yderligere er kryds og bolle et stærkt løst spil.

Kryds og bolle er hvad der kaldes et løst stærk løst spil. Et spil anses for at være løst, når det er muligt at forudse resultatet af et givent spil, før det starter, under antagelse af at alle spillere spiller perfekt. At kryds og bolle er stærkt løst betyder yderligere, at det kan forudses, hvilken spiller der vil vinde, eller om spillet ender uafgjort for en given tilstand af spillet.

Problemet med stærkt løste spil er, at disse ofte er løst ved at gennemgå alle spillets tilstande, hvorefter en række træk kan udvælges til at give det bedste resultat. At et spil er stærkt løst er dermed ikke ensbetydende med, at det er forstået, hvorfor et givent træk er bedst.

For at få en bedre forståelse for, hvorfor et givent træk var godt, blev det af denne grund valgt at beregne et givent træk ud fra spillets tilstand. Min-maks grafer, som vist på fig: 2.1, blev benyttet til at verificere, at de beregnede træk var perfekte.



(a) *Strategi for kryds* **(b)** *Strategi for bolle*

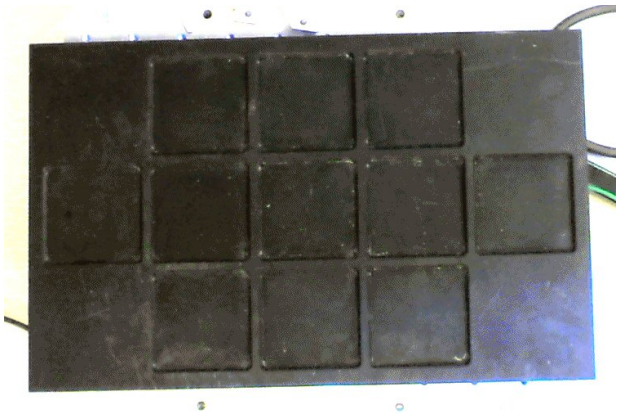
Figur 2.1: *Min-maks grafer af perfekt strategi. [Wikipedia]*

Disse min-maks grafer er beslutningstræer, hvor det bedste og værste scenarie skiftetvist bliver udvalg, hvilket dermed simulerer et spil imellem 2 spillere.

2.2 Introduktion til spillet

For at sikre en fælles forståelse for hvordan kryds og bolle spilles, vil der her blive givet en kort forklaring af spillet og dets regler.

Kryds og bolle er et spil for 2 spillere der spilles på et spillebræt med 9 felter, hvilket er arrangeret i en 3×3 matrice. Spillebrættet der anvendes af "den lille modstander" har 2 ekstra felter, som vist på fig: 2.2, hvilket ikke anvendes af spillet.



Figur 2.2: *Spillebrættet der benyttes af "den lille modstander"* .

Ved spillets start er alle felterne tomme, og hver spiller har hver 3 brikker. Den ene spillers brikker er formet som et kryds, hvorimod den anden spillers brikker er formet som en cirkel.

Spilleren med de kryds formede brikker er den første til at placere en af sine brikker på spillebrættet. Herefter skiftes de 2 spillere om at lægge en af deres brikker i et af de tomme felter på spillebrættet.

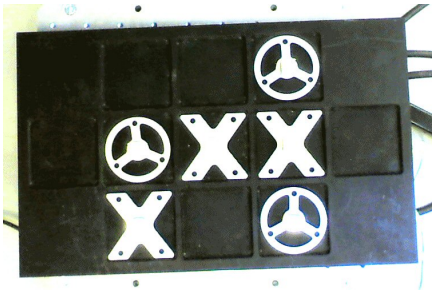
Når alle brikkerne er lagt på spillebrættet skiftes hver spiller til at fjerne en af deres brikker fra spillebrættet, og placere denne brik i et nyt tomt felt.

Vinderen af spillet er den spiller, der er i stand til først at placere sine 3 brikker i samme række, søjle eller diagonal, hvilket herefter vil blive henvist til som en stribe.

2.3 Digitalisering af spillet

På grund af spillets udformning, benyttes en 3×3 matrice til at gemme spillebrættets tilstand. I denne matrice anvendes tallene $0-2$ til at beskrive om et givent felt er tomt, eller om det indeholder en brik udformet som et kryds eller en cirkel. Et eksempel på denne digitalisering kan ses på fig: 2.3 på næste side

Brikkerne uden for spillebrættet var ikke nødvendige at gemme, eftersom deres placering ikke har betydning for spillets strategi. Ligeledes kunne det let beregnes, om brikker skulle tilføres eller flyttes, ud fra antallet af brikker af samme type der var på spillebrættet.



(a) *Virkelig*

0	0	2
2	1	1
1	0	2

(b) *Digital*

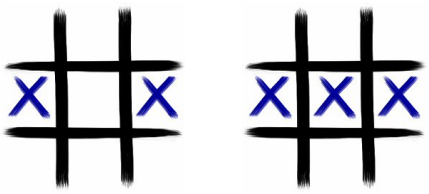
Figur 2.3: *Virkelig og digital repræsentation af spillebrættet.*

2.4 Strategi

Strategien der blev implementeret i "den lille modstander" var baseret på en lektion fra N. J. Wildberger, som benyttede kryds og bolle til at demonstrere logisk tænkning. [Wildberger, N. J.]

Det blev valgt at benytte strategien i denne lektion på grund af den opstillede form af klare regler. Disse regler var opdelt i 2 sæt af 3 regler, hvoraf reglerne for det første sæt lød som følgende:

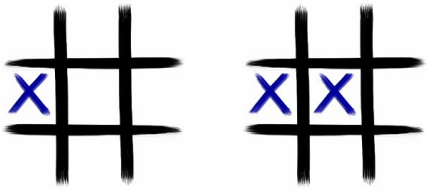
Vind:
Hvis en spiller har 2 brikker i samme stribe, og sidste felt er tomt, bør spilleren placere sin brik i dette felt og dermed binde spillet. Et eksempel på dette træk kan ses på fig: 2.4.



(a) *Før* (b) *Efter*

Figur 2.4: *Eksempel på et vindende træk.*

Trussel:
Hvis ikke det er muligt at vinde spillet, bør en spiller opstille en trussel, der gør det muligt at vinde spillet i næste træk. En trussel kan laves ved at placere en brik i en stribe med 2 tomme felter, hvis det 3. felt indeholder en af spillerens egne brikker. Et eksempel på dette træk kan ses på fig: 2.5.

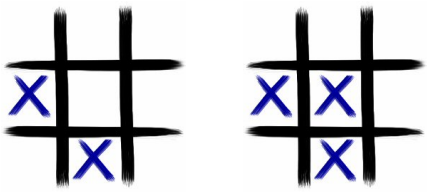


(a) *Før* (b) *Efter*

Figur 2.5: *Eksempel på en trussel.*

Fælde:

Det er muligt for modspilleren at blokere en trussel ved at placere en af sine brikker i det tomme felt i striben. For at gøre det muligt at vinde i næste træk, er det derfor nødvendigt at opstille en fælde, hvilket er scenariet, hvor en spiller har 2 samtidige trusler. Et eksempel på dette træk kan ses på fig: 2.6.



(a) *Før* **(b)** *Efter*

Figur 2.6: *Eksempel på en fælde.*

De resterende 3 regler er en gentagelse af de beskrevne regler, men med fokus på at forhindre modspilleren i at lave de samme træk. En kort version af de 6 regler, listet i deres prioriterede rækkefølge, vil dermed være følgende:

1. Lav et vindende træk.
2. Forhindrer modspiller i at lave et vindende træk.
3. Lav en fælde.
4. Forhindrer modspiller i at lave en fælde.
5. Lav en trussel.
6. Forhindrer modspiller i at lave en trussel.

2.5 Udvidet strategi

Strategien med disse regler viste sig, under forsøg, ikke at være tilstrækkelige til at gøre "den lille modstander" i stand til at spille perfekt. Af denne grund var det nødvendigt at tilføre følgende regler til den anvendte strategi:

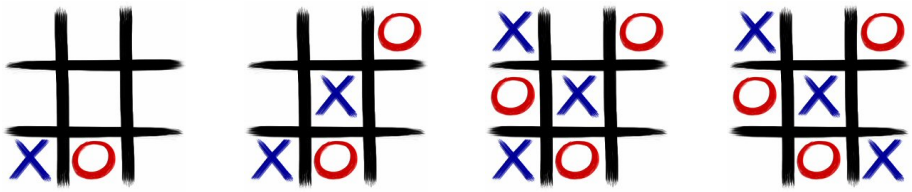
2.5.1 Strategi for første træk

De første træk i kryds og bolle viste sig at være de mest komplicerede på grund af alle de muligheder, de tomme felter giver.

Af denne grund lavede "den lille modstander" ofte et forkert træk, når den skulle placere den første cirkelformede brik. Et af disse scenarier kan ses på fig: 2.7 på næste side.

Årsagen til dette problem skyldes, at felterne "den lille modstander" benyttede, kun var en del af få striber, der kunne benyttes til at vinde spillet. Herudover var en af disse striber blokkeret på grund af modspillerens første træk, hvorfor det var let for modspilleren at blokere de resterende striber.

Ved at isolere denne brik var "den lille modstander" dermed et træk bagud og kunne derfor kun reagere, på hvilke træk modspilleren lavede.



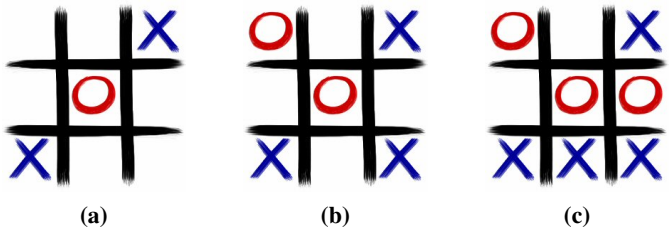
Figur 2.7: *Illustration af hvordan "den lille modstander" (O) taber i første træk.*

For at forhindre dette scenarie blev følgende regel tilført strategien, hvilket sikrede at "den lille modstanders" første brik ikke blev isoleret:

Benyt tomme striber:
Brikker bør placeres i felter, der indgår i flest mulige tomme striber.

2.5.2 Strategi for første fase

I den første fase af spillet, hvor spillerne skiftes til at tilføre en brik på spillebrættet, viste det sig, at der var et andet scenarie, der gav problemer for "den lille modstander" . Dette scenarie er vist på fig: 2.8.



Figur 2.8: *Illustration af hvordan "den lille modstander" (O) taber i første fase.*

I dette scenarie har modspilleren mulighed for at lave 2 forskellige fælder, ved at placere sin brik i et af de resterende hjørner. Problemet med dette scenarie var, at "den lille modstander" forsøgte at blokere en af disse fælder ved at placere en brik i hjørnet, og dermed tvang modspilleren til at lave en fælde.

For at undgå denne problemstilling blev følgende regel tilført strategien:

Undgå dobbelt fælde:
Hvis en modspiller er i stand til at lave to fælder, skal en trussel laves, således at modspilleren er tvunget til at blokere denne. Yderligere skal denne trussel laves således, at modspilleren ikke er i stand til at lave en fælde i næste træk.

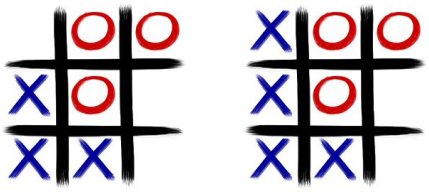
2.5.3 Strategi for anden fase

I anden fase af spillet, hvor spillerne skiftes til at flytte en af deres egne brikker, lavede "den lille modstander" ofte fejl i scenarier, hvor den havde lavet en fælde.

Denne fejl skyldes metoden, der blev benyttet til at beregne et givent træk frem for en reel fejl i strategien.

På grund af kompleksiteten ved at beregne hvilken brik, der skulle fjernes og hvor den herefter skulle placeres, var det valgt at opdele denne beregning i 2 dele. Den første del af processen beregnede i hvilket felt, "den lille modstander" burde placere sin brik, mens den anden del af processen beregnede, hvilken af de oprindelige 3 brikker der skulle fjernes.

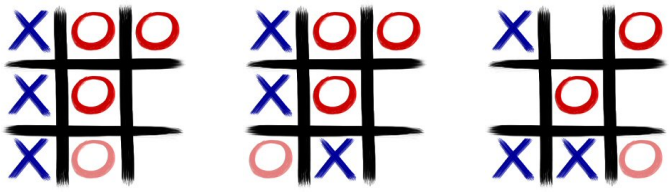
Første trin i et eksempel hvor "den lille modstander" altid ville lave fejl kan ses på fig: 2.9.



(a) *Før* **(b)** *Efter*

Figur 2.9: *Illustration af hvordan "den lille modstander" (X) lægger en 4. brik på spillebrættet.*

Næste trin var herefter at fjerne en af de 3 oprindelige brikker. Det blev derfor beregnet, for hvilken af disse brikker det havde færrest konsekvenser, når den blev fjernet. For scenariet vist på fig: 2.10, blev trækket på fig: 2.10c altid valgt.



(a) *Simulering 1* **(b)** *Simulering 2* **(c)** *Simulering 3*

Figur 2.10: *Illustration af hvordan "den lille modstander" (X) fjerner en af de ekstra brikker.*

Årsagen til denne fejl var, at der ikke var en verifikations proces under "den lille modstanders" beregning af træk. Af denne grund blev trækket vist på fig: 2.10c anset som det eneste træk, der forhindrede den i at tabe.

Af denne grund var det derfor nødvendigt at tilføre følgende regel, hvilket simulerede verifikations processen:

Behold vinderposition:
Hvis der er 3 ens brikker i samme stribe, bør ingen af disse brikker flyttes.

2.5.4 Opsummering af udvidet strategi

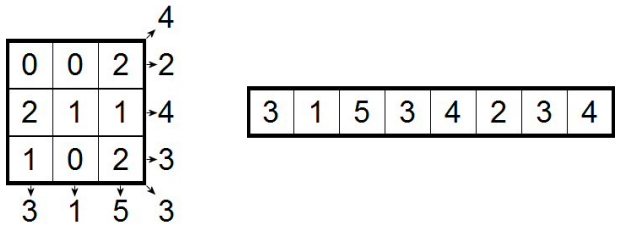
Med disse ekstra regler var strategien for "den lille modstander" komplet. Den endelige liste over regler for denne strategi, nævnt i deres prioriterede rækkefølge, er følgende:

1. Behold et vindende træk.
2. Lav et vindende træk.

3. Forhindrer modspiller i at lave et vindende træk.
4. Lav en fælde.
5. Undgå modspillerens dobbelt fælder.
6. Forhindrer modspiller i at lave en fælde.
7. Lav en trussel.
8. Forhindrer modspiller i at lave en trussel.
9. Placer brik i tomme striber.

2.6 Udfoldning

På grund af at strategiens regler bygger på indholdet af de enkelte striber, blev det valgt at basere beregningerne på et stribe format. Til dette formål blev der lavet en funktion, der konverterede spillebrættets matrice om til en stribe vektor, således at efterfølgende beregninger kunne blive baseret på stribernes indhold.



(a) *Matrice* **(b)** *Stribe vektor*

Figur 2.11: *Illustration af en matrice, samt stribe vektoren der laves ved udfoldning.*

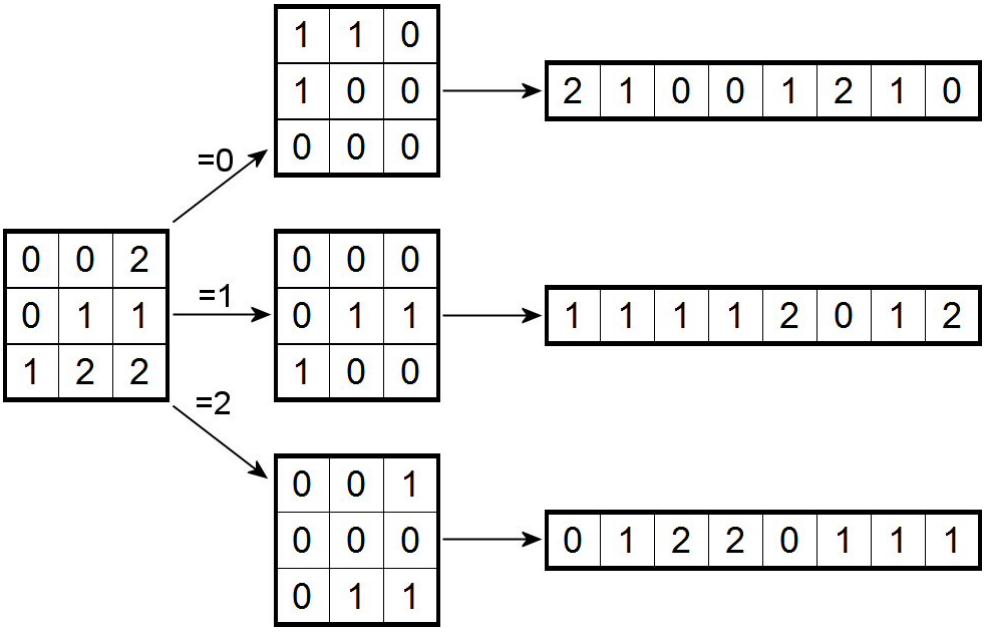
Denne funktion, som illustreret på fig: 2.11, summerer indholdet af hver af spillebrættes striber, hvorefter resultatet gemmes i en stribe vektor med 8 elementer. Denne funktion vil herefter blive henvist til som udfoldning af spillebrættet.

Selve udfoldnings processen, som vist på fig: 2.12 på næste side, starter med at spillebrættets matrice konverteres til 3 forskellige binære matricer, der beskriver indholdet af de enkelte felter. Når disse matricer herefter bliver udfoldet, dannes stribe vektorer der beskriver hvor mange felter, af en given tilstand, der er i de enkelte striber.

Ud fra disse stribe vektorer kan en række logiske operationer herefter blive udført, hvilket beregner om en given stribe opfylder en eller flere regler for strategien.

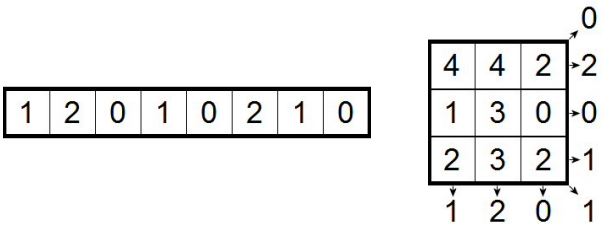
2.7 Sammenfoldning

For at konvertere informationen fra de enkelte striber tilbage til et matrice format, blev en ekstra funktion lavet.



Figur 2.12: *Illustration af hele udfoldnings processen for spillebrættet.*

Denne funktion, som illustreret på fig: 2.13, summerer indholdet af hver stribe, et givent felt er en del af, hvorefter resultatet gemmes i en 3×3 matrice. Denne funktion vil herefter blive henvist til som sammenfoldning af stribe vektorer.



(a) *Stribe vektor* **(b)** *Matrice*

Figur 2.13: *Illustration af en stribe vektor, samt matricen der laves ved sammenfoldning.*

Det skal her bemærkes, at udfoldning og sammenfoldnings funktionerne ikke er invers funktioner af hinanden. Det er dermed ikke muligt at gendanne den originale matrice for spillebrættet, når først den er blevet udfoldet.

Formålet med denne funktion er primært at konvertere prioriteten af de enkelte striber til prioriteten af de enkelte felter, således at et specifikt felt kan blive udvalgt af “den lille modstander” .

2.8 Prioritering

Det blev valgt at benytte et point system, frem for en prioriteringsliste til at bedømme de enkelte træk. Årsagen til dette valg var at det gjorde det lettere at redigere antallet af point for de enkelte træk, hvilket kunne benyttes til at ændre prioriteringen.

Formel 2.1 blev benyttet til at tildele hver stribe point, baseret på hvilke regler en given stribe opfylder. Denne formel bygger på en eksponentiel funktion, hvilket sikrer, at antallet af point for en given regel altid vil være større end summen af point fra de understående regler.

$$u = \sum_{n=1}^{n_{max}} \left(\mathrm{f}(n) \cdot 10^{(n_{max}-n)} \right) \tag{2.1}$$

Hvor:
u er antallet af point for den givne stribe stribe.
n_{max} er antallet af regler i strategien.
$\mathrm{f}(n)$ er en funktion der beregner om den givne stribe opfylder regel n.

2.9 Valg af træk

Når hver stribe var blevet tildelt point, blev sammenfoldning benyttet til at beregne antallet af point for de individuelle felter.

Herefter blev hvert felt tildelt et tilfældigt antal point mellem $[0;1]$, hvilket sikrede at et tilfældigt felt ville blive udvalgt, hvis 2 felter havde samme antal point.

Som sidste del af udvælgelsesprocessen blev antallet af point for felter der indeholdte en brik nulstillet, hvorefter feltet med det højeste antal point blev udvalgt som "den lille modstanders" træk.

2.10 Sværhedsgrad

På grund af at strategien findes i form af et regelsæt, kan dette benyttes til at styre sværhedsgraden af "den lille modstander" . Ved at ignorere enkelte regler ville "den lille modstander" lave fejl, når den valgte træk og dermed ikke spille perfekt.

En yderligere fordel, ved at regelsættet var i form af en prioriteringsliste, var at disse regler kunne blive ignoreret i modstat rækkefølge af deres prioritet. Dette sikrede at størrelsen af fejl, "den lille modstander" lavede var proportionelle med dens sværhedsgrad.

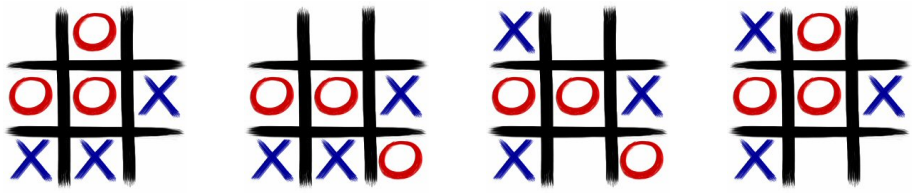
Udover at vælge en sværhedsgrad i starten af spillet, blev det også valgt gradvist at sænke "den lille modstanders" sværhedsgrad i løbet af et spil. På trods af at det fra et teknisk perspektiv er mere imponerende, når "den lille modstander" spiller perfekt, blev det vurderet, at det ville give et bedre indtryk, hvis modspilleren var i stand til at vinde.

Den automatiske reduktion af sværhedsgraden blev sat til at begynde, når alle brikker var placeret på spillebrættet. Herefter ignorerede "den lille modstander" en ekstra regel i dens strategi for hvert andet træk den udførte. Hastigheden af denne reduktion blev valgt ud fra, hvad der føltes som en tilpas længde af spillet, før "den lille modstander" blot flyttede brikker tilfældigt.

2.11 Variation

På trods af at det var ønsket, at få "den lille modstander" til at spille perfekt, blev det fundet, at dette resulterede i en række kedelige spil.

Dette skyldes det anvendte pointsystem, hvilket danne flere lokale maksima punkter. Disse punkter resulterede i at der flere scenarier, hvor det bedste træk for "den lille modstander" var at gå tilbage til en tidligere tilstand af spillebrættet. Dette resulterede i at "den lille modstander" gentog de samme 2 træk, indtil dens sværhedsgrad var tilpas lav. Et af disse scenarier kan ses på fig: 2.14.



Figur 2.14: *Eksempel på et scenarie hvor "den lille modstander" gentager en række træk.*

Det blev af denne grund valgt at tildele ekstra point til træk, "den lille modstander" ikke valgte. Disse point blev gemt i "den lille modstanders" hukommelse og ville dermed langsomt akkumulere. Når "den lille modstander" valgte et givent træk, blev alle ekstra point for dette nulstillet.

Fordelen var at disse ekstra point dermed kunne få et givent træk, der ikke anses som værende optimalt, til at have førsteprioritet, hvilket dermed brød "den lille modstanders" bevægelsesmønster.

En ulempe ved disse ekstra point var, at de i teorien kunne få "den lille modstander" til at bryde alle regler, så længe den var forhindret i at lave et givent træk i løbet af et tilstrækkeligt antal ture.

På grund af den automatiske reduktion af "den lille modstanders" sværhedsgrad, blev det valgt ikke at sætte en øvre grænse for denne tvungne variation. Dette skyldes at spillets varighed var for kort, på grund af den automatiske reduktion af sværhedsgraden, til at dette kunne resultere i grove fejl.

Herudover blev det også valgt at benytte et tilfældigt felt, hvis "den lille modstander" var den første til at placere en brik på spillebrættet. Dette valg blev taget på baggrund af, at "den lille modstander" kunne undgå at tabe, uanset hvor den placerede den første brik. Af denne grund blev et mere varieret spil igen prioriteret over en strategi der gav større chancer for at vinde.

2.12 Opsummering

Trods anvendelsen af mere utraditionelle metoder til at beregne de enkelte træk i kryds og bolle, var det muligt at få "den lille modstander" til at spille perfekt.

Denne metode gav en række fordele, eftersom de enkelte regler kunne benyttes til at justere "den lille modstanders" sværhedsgrad.

En ulempe ved denne metode var at flere spil blev anset som værende kedelige, på grund af de lokale maksima punkter. Der blev af denne grund lavet en række tilføjelser til "den lille modstanders" strategi,

således at den ikke gentog de samme træk for ofte. Dette reducerede "den lille modstanders" chancer for at vinde spillet, men skabte et mere varieret spil, der kunne holde modspillerens interesse.

Disse tilføjelser gjorde, at "den lille modstanders" strategi herefter kan anses som værende ustabil. På grund af denne ustabilitet er proportionel med varigheden af et givent spil, blev det valgt ikke at forbedre strategien. Årsagen til dette valg skyldes, at den automatiske reduktion af sværhedsgraden gør, at hvert spil er for kortvarigt til, at denne ustabilitet giver problemer for "den lille modstander" .

Af denne grund kan "den lille modstander" ikke længere anses som værende i stand til at spille perfekt. Dog anses dette mere varierede spil at tjene den oprindelige vision af "den lille modstander" bedre, end hvis den havde spillet perfekt.

Kapitel 3

Robot interface

Et Matlab interface med PolyScope var allerede lavet af Fereshteh Aalamifar, hvilket kunne benyttes til at styre robotten. [Fereshteh, Aalamifar]

Dette interface sørgede for at oprette kommunikation imellem begge programmer, samt gjorde brugeren i stand til at benytte følgende funktioner:

- Læse robottens tool point position og orientering.
- Flytte robottens tool point, via lineære bevægelse og fast hastighed, til den ønskede position og orientering.
- Aktivere og deaktivere "kooperativ" tilstand for robotten.

Med disse funktioner var den eneste manglende funktionalitet, at kunne styre griberen der var påmonteret robotten. Tilføjelsen af denne funktion vil blive beskrevet i afsnit 3.3.1 på side 29.

Flere af funktionerne i dette Matlab interface blev anset for kun at opfylde minimumskravene for styringen af robotten, og derfor blev det valgt at udvide dette interface. Dette blev gjort ud fra et ønske om at kunne benytte flere af robottens funktioner samt gøre de eksisterne funktioner mere fleksible.

Trods brugen af dette udvidet interface i projektet, vil basis interfacet blive benyttet til at gennemgå koden for dette interface. Dette valg skyldes, at flere funktioner i det udvidede interface genanvender de samme linjer kode flere gange, hvorfor basis vil være lettere at overskue.

Den fulde kode af begge versioner af PolyScope programmet er at finde i bilag A, mens en kort beskrivelse af hvilke funktionaliteter, der blev tilføjet vil være at finde i afsnit 3.4 på side 30.

3.1 PolyScope

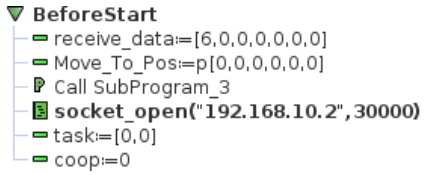
I PolyScope er det muligt at benytte flere tråde, hvilket betyder at programmet er i stand til at køre flere programmer parallelt. I dette program er der gjort brug af 2 tråde.

Den første tråd benyttes til at styre kommunikationen med Matlab, hvorfor dens funktioner benyttes til at ændre variabler samt at sende og modtage beskeder.

Den anden tråd benyttes derimod til at styre selve robottens bevægelser, hvorfor denne funktion hovedsageligt kun læser variabler.

3.1.1 Program start

De 2 tråde der benyttes gør brug af de samme variabler, hvorfor det er nødvendigt at sikre, at disse variabler r defineret. Til dette formål benytter PolyScope et program kaldet *BeforeStart*, hvilket køres før de 2 tråde bliver startet.

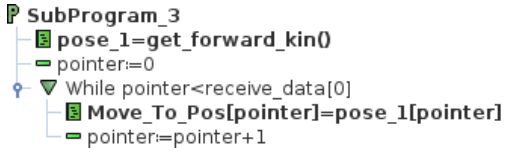


Figur 3.1: *BeforeStart funktionen i basis interfacet til PolyScope*

På fig: 3.1 kan dette dette program ses, hvilket benyttes til at definere variablerne *receive_data*, *Move_To_Pos*, *task* og *coop*.

Herudover forsøger dette program at skabe forbindelse imellem Matlab og Polyscope. Det skal her bemærkes, at PolyScope automatisk vil gå i timeout, hvis det ikke lykkes at skabe kontakt efter 2 sekunder. Af denne grund er det vigtigt at Matlab delen af dette interface startes før PolyScope programmet. [URScript Manual]

Yderligere anvender dette program et underprogram, kaldet *SubProgram_3*. På fig: 3.2 kan det ses at dette underprogram først gemmer positionen og orienteringen af robottens tool point i variablen *pose_1*. Herefter overskrives variablen *Move_To_Pos* med samme information.



Figur 3.2: *SubProgram_3 funktionen i basis interfacet til PolyScope*

Dette underprogram sætter dermed den ønskede position for robotten til at være dens nuværende position. Dette sikrer, at robotten ikke laver nogen uhensigtsmæssige bevægelser under start af programmet.

Det vides ikke, hvorfor det er valgt at benytte denne omvej for at omskrive *Move_To_Pos*.

3.1.2 Første tråd

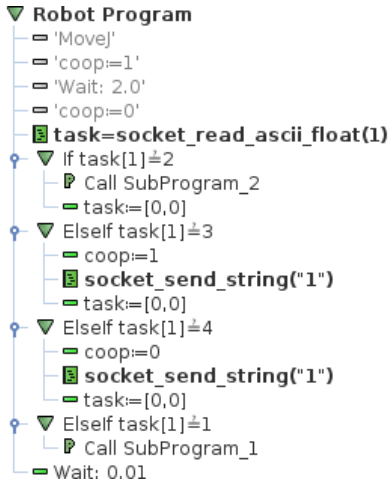
Den første tråd i programmet, kaldet *Robot Program* har til opgave at modtage beskeder fra Matlab. Disse beskeder bliver herefter læst, og de ønskede funktioner bliver benyttet.

På fig: 3.3 på næste side kan det ses at modtagne beskeder bliver gemt i variablen *task*, hvorefter en række *if then* operationer benyttes til at identificere, hvilken funktion den sendte besked ønsker at benytte.

Det skal her bemærkes at PolyScope benytter 0 som det første element i dens variabler. Yderligere skal det bemærkes, at beskeder bliver tilført et tal, først i beskeden, der siger hvor mange elementer en besked består af. Af denne grund vil beskeden (3) blive indlæst i PolyScope som værende (1,3).

Dette er grunden til, at variablen *task* består af 2 elementer, hvorimod kun det ene bliver benyttet i programmet.

Ligesom ved oprettelsen af forbindelse imellem Matlab og PolyScope, går modtagelsen af beskeder også i timeout efter 2 sekunder. I dette tilfælde vil beskeden blive gemt som (0,NaN). Det første element vil dermed signalere, at denne besked ikke har noget indhold, mens det andet element ikke har nogen tildelt



Figur 3.3: *Robot Program tråden i basis interfacet til PolyScope*

værdi. [URScript Manual]

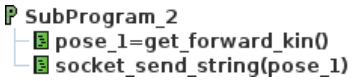
Som det kan ses på fig: 3.3 er der 4 funktioner, der kan benyttes, hvoraf 2 af disse benytter deres eget underprogram. De 2 resterende funktioner benyttes til henholdsvis at slå robottens "kooperative" tilstand til eller fra.

På grund af at denne tråd ikke styrer robotten, ændrer den kun værdien for variablen *coop*. Dette signalerer til den anden tråd, hvilken type bevægelser der ønskes at blive benyttet.

Efter variablen er blevet ændret sendes et kvitteringssignal til Matlab, hvorefter variablen *task* nulstilles.

3.1.3 Læs nuværende position

SubProgram_2 er underprogrammet, der sender robottens nuværende position og orientering. Som det kan ses ud fra fig: 3.4 er dette et simpelt program, hvilket først gemmer den ønskede information i variablen *pose_1*, hvorefter denne sendes til Matlab.



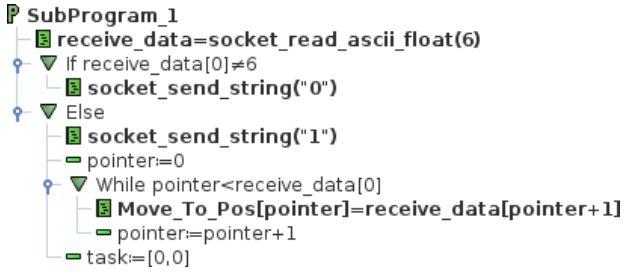
Figur 3.4: *SubProgram_2 funktionen i basis interfacet til PolyScope*

Som det kan ses benytter dette underprogram den sendte information som dets kvitteringssignal.

3.1.4 Send ønsket position

SubProgram_1 er underprogrammet, der opdaterer hvilken position og orientering det ønskes, at robotten skal opnå. På grund af at den første besked fra Matlab kun fortalte PolyScope, hvilken funktion det var ønsket at benytte, kræver PolyScope koordinaterne for den ønskede position og orientering.

Af denne grund starter dette underprogram, som vist på 3.5 på den følgende side, med at afvente besked fra Matlab, hvilket bliver gemt i variablen *receive_data*.



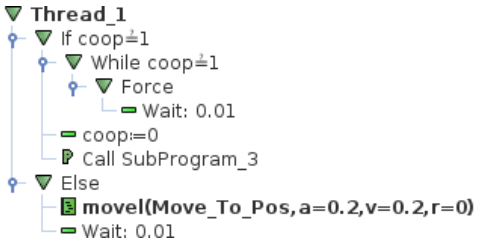
Figur 3.5: *SubProgram_1 funktionen i basis interfacet til PolyScope*

Beskedens længde verificeres herefter ved at checke det første element i beskeden, hvilket indeholder en værdi for beskedens længde. Hvis beskeden har den korrekte længde sendes et kvitteringssignal, hvorimod et fejlsignal vil blive sendt hvis dette ikke er tilfældet.

Hvis det rigtige antal koordinater blev modtaget, vil variablen *Move_To_Pos* blive opdateret, hvorefter variablen *task* nulstilles.

3.1.5 Anden tråd

Koden for den anden tråd i PolyScope programmet, kaldet *Thread_1*, kan ses på fig: 3.6.



Figur 3.6: *Thread_1 tråden i basis interfacet til PolyScope*

Hvis den "kooperativ" ikke benyttes, og variablen *coop* dermed er 0, vil denne tråd forsøge at flytte robotten til den ønskede position og orientering, som specificeret i variablen *Move_To_Pos*.

Dette viser nogle af begrænsningerne for basis interfacet, eftersom dette kun gør brug af lineære bevægelser med en ønsket hastighed og acceleration på henholdsvis 0.2 $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$ og 0.2 $\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]$.

"Kooperativ" tilstand

Hvad angår koden for robottens såkaldte "kooperative" tilstand, kan det ses på fig: 3.6 at den funktion der benyttes kaldes *Force*. Når denne funktion benyttes, forsøger robotten at holde en konstant kraftpåvirkning i de ønskede akser. Herudover kan det også specificeres, i hvilke akser robotten skal forsøge at undvige eksterne kraftpåvirkninger.

Denne funktion kan dermed imitere robottens *teach mode* ved at specificere at robottens kraftpåvirkninger i alle akser skal være 0 [N], samtidig med, at den skal undvige eksterne kraftpåvirkninger.

Det er dermed muligt at tage fat om robotten og flytte den manuelt, uden brug af dens panel og dermed *teach mode*. Trods det er muligt at flytte robotten manuelt i denne tilstand, kræver dette langt større

kraftpåvirkninger end hvis *teach mode* benyttes.

En yderligere problemstilling ved denne tilstand er, at robotten har en tendens til at gå i selvsving, hvilket ofte resulterer i, at robotten går i nødstop, når leddene roterer uden for deres arbejdsområde.

Denne problemstilling er ikke blevet undersøgt nærmere, eftersom denne tilstand blev anset for at være for ustabil til at benytte i projektet, og det kan derfor ikke anbefales at benytte denne tilstand.

3.2 Matlab

Til at kommunikere med PolyScope programmet benytter Matlab interfacet flere funktioner. Det er her valgt ikke at gennemgå alle funktionerne, idet flere af de samme elementer indgår i disse funktioner. Det er derfor valgt kun at fokusere på funktionen til at starte kommunikationen imellem Matlab og PolyScope samt funktionerne til at sende og modtage koordinater for robottens tool point.

3.2.1 Start af server

For at starte kommunikationen med robotten, benyttes koden som vist på fig: 3.7.

```
% Connect to robot
Robot_IP = '192.168.10.1';
Socket_conn = tcpip(Robot_IP,30000,'NetworkRole','server');
fclose(Socket_conn);
disp('Press Play on Robot...')
fopen(Socket_conn);
disp('Connected!');
```

Figur 3.7: *Koden til at få Matlab til at kommunikere med UR5 robotten.*

I denne kode opsættes Matlab til at agere som server for kommunikationen imellem programmerne.

Herefter giver Matlab besked til brugeren om, at Matlab er klar til, at PolyScope interfacet bliver startet. I modsætning til PolyScope, vil Matlab ikke gå i timeout.

Når PolyScope interfacet er tilsluttet, får brugeren en besked der bekræfter dette.

3.2.2 Send besked til UR5

For at flytte robottens tool point, benyttes funktionen *moverobot*.

Denne funktion starter med at verificere antallet af inputs brugeren benytter, som vist på fig: 3.8 på den følgende side.

Det antages at denne verifikation skyldes, at robotten kræver både en translations og en rotations vektor for en given position. Hvis brugeren derfor benytter disse vektorer separat, er det derfor nødvendigt at samle dem til en vektor med 6 elementer.

Denne vektor bliver herefter formateret til formatet PolyScope, som vist på fig: 3.9 på næste side. Dette format er kommaseparerede tal omsluttet med parenteser til at signalere start og stop af beskeden.

```
% Goal_Pose should be in mm and Orientation the rotation vector
function P_new = moverobot(t,Goal_Pose,Orientation)
if nargin == 1
    error('error; not enough input arguments')
elseif nargin == 2
    P = Goal_Pose;
elseif nargin == 3
    P = [Goal_Pose,Orientation];
end
```

Figur 3.8: *Indledende verifikation for moverobot funktionen i Matlab*

Herudover bliver translations vektoren konverteret fra meter til millimeter.

```
P(1:3) = P(1:3) * 0.001; % Converting to meter
P_char = ['(',num2str(P(1)),',',...
    num2str(P(2)),',',...
    num2str(P(3)),',',...
    num2str(P(4)),',',...
    num2str(P(5)),',',...
    num2str(P(6)),...
    ')'];
```

Figur 3.9: *Formatering af besked for moverobot funktionen i Matlab.*

Med beskeden i det rigtige format kan Matlab sende den til PolyScope, hvilket gøres som vist på fig: 3.10.

```
success = '0';
while strcmp(success,'0')
    fprintf(t,'(1)'); % task = 1 : moving task
    pause(0.01);% Tune this to meet your system
    fprintf(t,P_char);
    while t.BytesAvailable==0
        %t.BytesAvailable
    end
    success  = readrobotMsg(t);
end
if ~strcmp(success,'1')
    error('error sending robot pose')
end
```

Figur 3.10: *Sending af besked for moverobot funktionen i Matlab.*

På grund af opbygningen af PolyScope interfacet kræver dette 2 beskeder når det ønskes at flytte robottens tool point. Den første besked Matlab sender, er derfor et ønske om at benytte funktion nr. 1 i PolyScope interfacet, hvilket er funktionen der ændrer den ønskede position og orientering af robottens tool point.

Herefter er der indsat en kort pause for at sikre, at PolyScope er klar til at modtage de ønskede koordinater, hvorefter disse bliver sendt.

Når beskeden er sendt afventer Matlab funktionen et kvitteringssignal fra PolyScope. Igen skal der gøres opmærksom på, at basis interfacet kun sender kvitteringssignaler for modtagne beskeder.

3.2.3 Læs besked fra UR5

For at læse robottens koordinater benyttes funktionen *readrobotpose*.

Som vist på fig: 3.11, starter denne funktion med at slette alle beskeder fra PolyScope, der ikke er blevet læst.

```
% t is the Socket Connection handle
% P is translation in mm and rotation vector
function P = readrobotpose(t)

if t.BytesAvailable>0
    fscanf(t,'%c',t.BytesAvailable);
end
```

Figur 3.11: *Slet af tidligere beskeder for readrobotpose funktionen i Matlab.*

Det antages at disse ulæste beskeder slettes for at simplificere opgaven med at modtage koordinaterne for robottens tool point, trods risikoen for at vigtige beskeder bliver slettet.

Herefter starter den egentlige opgave med at få koordinaterne fra robottens tool point, som vist på fig: 3.11. For at få tilsendt koordinaterne sender Matlab først en besked til PolyScope om at benytte funktion nr. 2, hvilket sender de ønskede koordinater tilbage til Matlab.

Formatet PolyScope benytter dets koordinater, som vist i formel 3.1, til at verificere den modtagne besked. Dette gøres ved at checke, at det første og sidste tegn i beskeden henholdsvis er p og].

$$\mathrm{p}\,[t_x, t_y, t_z, r_x, r_y, r_z] \tag{3.1}$$

$$\mathrm{p}[t_x, t_y, t_z, r_x, r_y, r_z, \tag{3.2}$$

Efter denne besked er verificeret, bliver den formateret til det format, som Matlab benytter. Som vist på fig: 3.12, starter denne formatering ved at ændre det sidste tegn i beskeden til et komma. Dette ændrer beskedens format fra formatet benyttet i formel 3.1 til formatet benyttet i formel 3.2.

```
rec(end) = ',';
Curr_c = 2;
for i = 1 : 6
    C = [];
    Done = 0;
    while(Done == 0)
        Curr_c = Curr_c + 1;
        if strcmp(rec(Curr_c) , ',')
            Done = 1;
        else
            C = [C,rec(Curr_c)];
        end
    end
    P(i) = str2double(C);
end
```

Figur 3.12: *Formatering af besked for readrobotpose funktionen i Matlab.*

Denne ændring gør, at alle tal i beskeden vil blive efterfulgt af et komma, hvormed det er muligt at separere de 6 forskellige koordinater i beskeden. Efter denne separation konverteres beskeden fra en tekststreng til et tal, hvilket giver en vektor med 6 elementer.

```
for i = 1 : length(P)
    if isnan(P(i))
        error('robotpose read error (Nan)')
    end
end
P(1:3) = P(1:3)*1000; % converting to mm
end
```

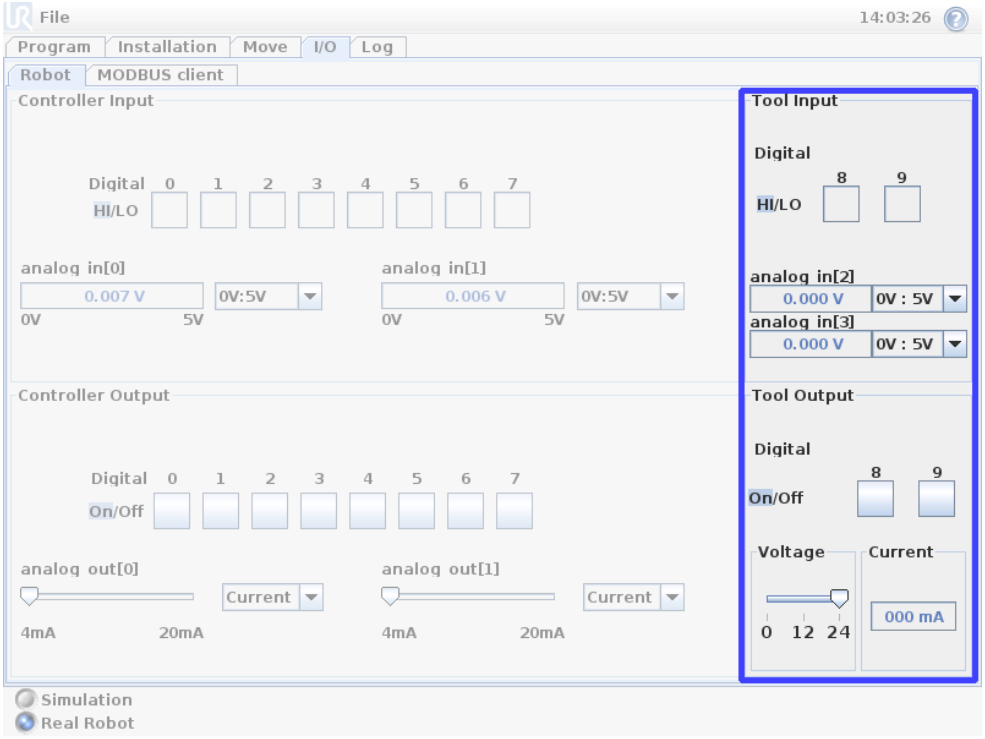
Figur 3.13: *Efterbehandling af besked for readrobotpose funktionen i Matlab.*

Som vist på fig: 3.13 verificeres det herefter, at alle elementerne er tal, hvorefter translationsvektoren konverteres fra meter til millimeter.

3.3 Styring af griber

Til at flytte brikkerne på spillebrættet benyttes en RG2 griber, lavet af On Robot, som er designet til at blive påmonteret UR robotter.

Som det kan ses ud fra fig: 3.14 gør griberen brug af 2 udgående og 2 indgående digitale porte samt 2 indgående analog porte.



Figur 3.14: *Interface for griber.*

De udgående digitale porte benyttes til at styre griberens position og styrke, afhængig af hvilken af de 2 mulige konfigurationer der benyttes.

I den første konfiguration, kaldet *1G2P2F*, har griberen 2 forskellige positioner og 2 mulige gribe styrker. I denne konfiguration benyttes den første port til at skifte imellem de mulige positioner, mens den anden port benyttes til at skifte gribe styrken.

For den anden konfiguration, kaldet *1G4P1F*, benytter griberen derimod 4 forskellige positioner der alle benytter samme gribe styrke. I denne konfiguration skifter den første port imellem 2 positioner, mens den anden port skifter, hvilket sæt den første port benytter.

Til griberen medfulgte en USB med et PolyScope program der kunne benyttes til at vælge, hvilken konfiguration der skulle benyttes samt den ønskede gribe styrke og afstand imellem fingrene for hver position.

De analoge porte på griberen benyttes til at måle afstanden imellem dens fingre. Dette gøres ud fra formel 3.3. [RG2 User Manual]

$$x = \frac{v_{cur}}{v_{max}} \cdot 110 \text{ [mm]} \tag{3.3}$$

Hvor:
x er afstanden imellem griberens fingre.
v_{cur} er den målte spænding.
v_{max} er den maksimale spænding.

Den maksimale spænding kan beregnes med formlerne 3.4 og 3.5, hvilke benyttes for henholdsvis spændingsintervallet imellem $0\text{ [V]} - 5\text{ [V]}$ og $0\text{ [V]} - 10\text{ [V]}$. [RG2 User Manual]

$$3.7\text{ [V]} = 5.0\text{ [V]} \cdot \frac{29\text{ [k}\Omega\text{]}}{10\text{ [k}\Omega\text{]} + 29\text{ [k}\Omega\text{]}} \tag{3.4}$$

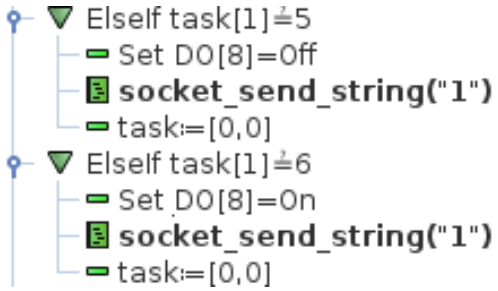
$$3.0\text{ [V]} = 5.0\text{ [V]} \cdot \frac{15\text{ [k}\Omega\text{]}}{10\text{ [k}\Omega\text{]} + 15\text{ [k}\Omega\text{]}} \tag{3.5}$$

Det antages at de 5 [V] der benyttes i begge formler er på grund af griberens maksimale output. Valget af spændingsinterval er der med baseret på hvilket input der forventes.

3.3.1 Styring af griber

For "den lille modstander" blev det valgt at benytte konfigurationen *1G4P1F* hvilket gav griberen 4 positioner og en gribe styrke. De første 2 positioner blev tilpasset brikkerne, mens de resterende 2 blev tilpasset kalibrerings dornen.

Givet det kun var nødvendigt at benytte 2 af positionerne, når "den lille modstander" spillede kryds og bolle, var det tilstrækkeligt kun at ændre værdien for en af de digitale udgangsporte. Til dette formål kunne samme metode bruges, som basis interfacet benyttede til at ændre robottens "kooperative" tilstand. Koden, vist på fig: 3.3 på side 23, blev af denne grund kopieret, variablen *coop* blev erstattet af den ønskede digitale udgang, hvilket resulterede i koden som vist på fig: 3.15 på den følgende side.



Figur 3.15: *Efterbehandling af besked for readrobotpose funktionen i Matlab.*

3.4 Udvidet interface

Ved at tilføre styring af griberen til basis interfacet havde dette tilstrækkelig funktionalitet for "den lille modstander" . Dog opfyldte dette kun minimumskravene, hvorfor det var nødvendigt at improvisere løsninger for enkelte problemstillinger.

En af de større mangler ved basis interfacet var, at det kun havde kvitterings signaler for modtagne beskeder. Af denne grund var det nødvendigt at benytte funktionen til at hente koordinaterne for robottens tool point for derefter at sammenligne disse med de ønskede koordinater.

Ligeledes var der intet kvitterings signal for griberen, hvorfor en pause ville være nødvendig for at sikre, at griberen var lukket.

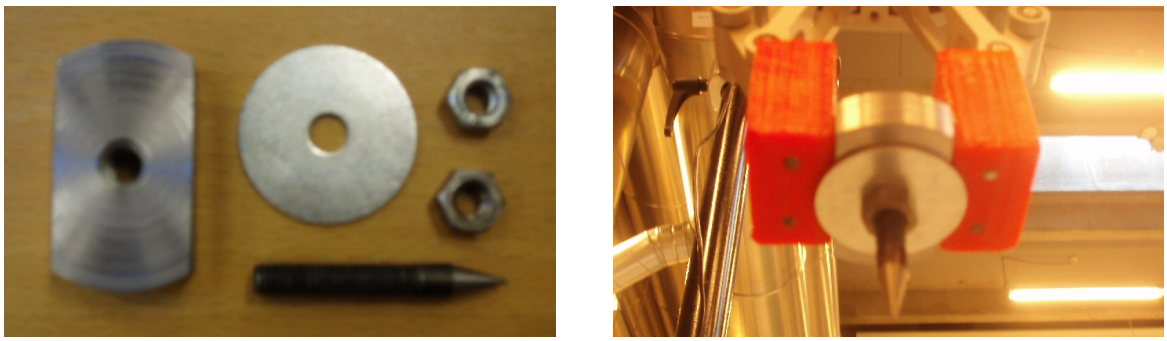
På grund af disse begrænsninger blev det valgt at udvide basis interfacet ved at tilføre flere funktioner samt gøre eksisterende funktioner mere generelt gældende. Dette resulterede i følgende ændringer:

- Funktion til at læse digitale indgange.
- Funktion til at læse analoge indgange.
- Funktion til at ændre digitale udgange.
- Kvitterings signal for når robottens tool point opnår den ønskede position.
- Funktion til at ændre robottens hastighed og acceleration.
- Funktion til at skifte imellem lineære og led bevægelser.
- Funktion til at ændre gribe bredde og gribe styrke for RG2 griber.
- Måling af tool pointets kraftpåvirkninger til sikkerhedsstop.

Kapitel 4

Kalibrering af robot

Med interfacet til robotten kunne den herefter kalibreres til spillebrættet. Til dette formål blev en dorn konstrueret, som vist på fig: 4.1, der kunne blive monteret i robottens griber.



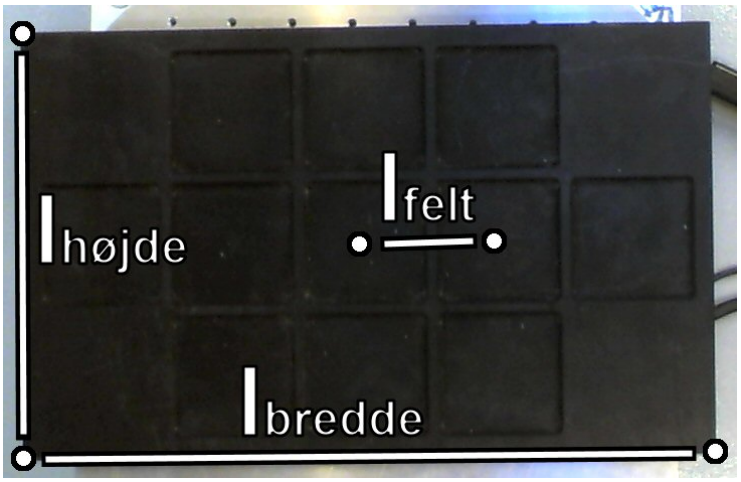
(a) *Adskilt* **(b)** *Monteret*

Figur 4.1: *Billeder af kalibrerings dorn.*

Spændskiven blev tilføjet for at reducere risikoen for, at dornens placering i griberen ikke ændrede sig, når den kom i kontakt med spillebrættet.

4.1 Position

For kalibrering af *robot space* til *board space* blev spillebrættets 4 hjørnepunkter benyttet, hvilket gjorde det muligt at finde et givent punkt på spillebrættet ved brug af interpolering.



Figur 4.2: *Målte længder på spillebrættet der benyttes til interpolering.*

Da denne interpolering også ville blive benyttet til at kalibrere *image plane* til *board space*, blev det valgt at udtrykke koordinaterne af spillebrættet ud fra vektorerne s og t. Feltkoordinaterne for spillebrættet kunne dermed blive udtrykt ud fra formel 4.1 og 4.2. De målte længder der er benyttet i disse formler kan ses på fig: 4.2.

$$s = \frac{l_{\text{felt}} \cdot (i-2) + \left(\frac{l_{\text{højde}}}{2}\right)}{l_{\text{højde}}} \tag{4.1}$$

$$t = \frac{l_{\text{felt}} \cdot (j-2) + \left(\frac{l_{\text{bredde}}}{2}\right)}{l_{\text{bredde}}} \tag{4.2}$$

Hvor:
s og t er ortogonale vektorer.
i og j er heltal der mellem $[1;3]$.

Disse koordinater kan herefter benyttes til at interpolere koordinater fra *robot space* eller *image plane* til *board space* ved brug af formel 4.3.

$$p(s,t) = (1-s)\cdot(1-t)\cdot p_1 + (1-s)\cdot t\cdot p_2 + s\cdot(1-t)\cdot p_3 + s\cdot t\cdot p_4 \tag{4.3}$$

Hvor:
$p(s,t)$ er koordinaterne for det ønskede punkt.
$p_{1\ldots4}$ er koordinater for de 4 hjørnepunkter.
s og t er tal mellem $[0;1]$.

Ud fra spillebrættets udformning kunne formel 4.2 ændres til formel 4.4, således, at koordinaterne for spillebrættes ekstra felter samt depoter også kunne identificeres.

$$t = \frac{l_{\text{felt}} \cdot (j-3) + \left(\frac{l_{\text{bredde}}}{2}\right)}{l_{\text{bredde}}} \tag{4.4}$$

Hvor:
j er heltal mellem $[1;5]$.

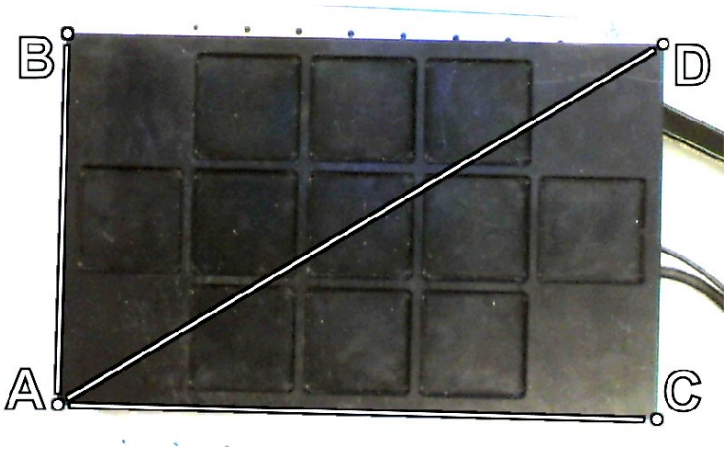
Formålet med disse ekstra felter og depoter var at simplificere processen med at identificere brikker uden for spillebrættet, ved at have dem i faste positioner. For de 4 depoter, nær hjørnerne af spillebrættet, var det nødvendigt at have et ekstra koordinat der tog højde for at disse depoter ikke var udfræset.

På fig: 4.3 på næste side er koordinaterne for centeret, samt hjørnerne af, spillebrættets felter og depoter vist på *image plane*.

Det skal her bemærkes at interpolering benyttet i formel 4.3 af afhængig af rækkefølgen af de benyttede kalibreringspunkter, hvorfor det er nødvendigt at denne er korrekt.

For at sortere hjørnepunkterne i "den lille modstander" , gøres der brug af at spillebrættet er udformet som et rektangel i både *robot space* og *image plane*. Dette gøres ved sortere hjørnepunkterne ud fra deres afstand til et fast hjørnepunkt, hvilket anvendes som nulpunkt for disse beregninger. Afstanden imellem det udvalgte hjørnepunkt og nulpunktet vil dermed være 0, mens de resterende hjørnepunkter vil have forskellige afstande til dette nulpunkt. En illustration af denne sorteringsmetode kan ses på fig: 4.4 på modstående side.

Figur 4.3: *Koordinater for felter og depoter på spillebrættet.*



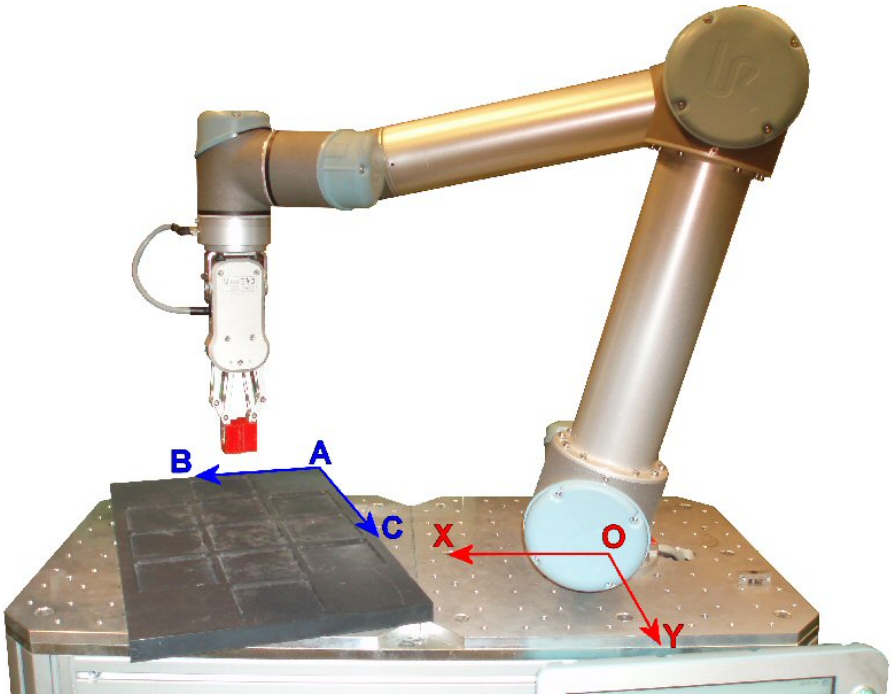
Figur 4.4: *Rækkefølgen af hjørnepunkter, hvis A vælges som nulpunkt*

Denne sorteringsmetode reducerer dermed risikoen for fejl ved interpoleringen, eftersom den kun er afhængig af det valgte nulpunkt, men i sig selv ikke er tilstrækkelig.

Det er af denne grund op til brugeren at sørger for at samme hjørne bliver anvendt som nulpunkt i både *robot space* og *image plane*.

4.2 Orientering

For at få robotten til at gribe brikkerne korrekt, var det nødvendigt at tage højde for orienteringen af *board space* i forhold til *robot space*, hvilket er illustreret på fig: 4.5.



Figur 4.5: *Skitsering af orientering af board space i forhold til robot space.*

Til dette formål kunne 2 hjørnepunkter, fra samme kant på spillebrættet, benyttes til at beregne en retningsvektor. En anden retningsvektor kunne herefter opstilles, langs en af robottens akser, hvorefter vinklen imellem disse kunne beregnes ud fra formel 4.5.

$$\theta = \cos^{-1}\left(\frac{\underline{AB} \bullet \underline{OX}}{\|\underline{AB}\| \cdot \|\underline{OX}\|}\right) \tag{4.5}$$

$$\widehat{n} = \underline{AB} \times \underline{OX} \tag{4.6}$$

Herefter kunne krydsproduktet af disse vektorer, som vist i formel 4.6, benyttes til at beregne deres normalakse.

Normalaksen og vinklen kan herefter benyttes til at beregne rotationsvektoren imellem *board space* og *robot space*. For at beregne rotationsvektoren ved brug af *axis-angle* notation benyttes formel 4.7, hvorimod Eulor vektorer benytter notationen beregnet i formel 4.8.

$$\begin{aligned}\underline{r} &= \begin{bmatrix}\widehat{\underline{n}} & \theta\end{bmatrix} \\ &= \begin{bmatrix}\widehat{n}_x & \widehat{n}_y & \widehat{n}_z & \theta\end{bmatrix} \qquad (4.7)\\ \underline{\omega} &= \widehat{\underline{n}} \cdot \theta \\ &= \begin{bmatrix}\widehat{n}_x \cdot \theta & \widehat{n}_y \cdot \theta & \widehat{n}_z \cdot \theta\end{bmatrix} \qquad (4.8)\end{aligned}$$

Grunden til at disse 2 notationer benyttes, skyldes at Matlab gør brug af *axis-angle* notation, hvorimod PolyScope benytter Euler vektorer. For at omregne til Euler vektorer, kan de enkelte elementer fra formel 4.7 isoleres og benyttes i formel 4.8.

For derimod at omregne fra Euler vektorer tilbage til *axis-angle* notation, er det nødvendigt at benytte formel 4.9 og 4.10.

$$\begin{aligned}\theta &= \|\underline{\omega}\| \\ &= \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2} \qquad (4.9)\\ \widehat{\underline{n}} &= \begin{bmatrix}(\omega_x/\theta) & (\omega_y/\theta) & (\omega_z/\theta)\end{bmatrix} \qquad (4.10)\end{aligned}$$

På grund af at robottens tool point allerede er roteret i forhold til robottens koordinatsystem, er det nødvendigt at kombinere dens eksisterende rotationsvektor med den ønskede ændring. Til dette formål benyttes Rodrigues formel, 4.11, at konvertere begge vektorer til rotations matricer. [Szeliski, Richard,2010]

$$\underline{\underline{R}} = \underline{\underline{I}} + \sin(\theta) \cdot \underline{\underline{[\widehat{n}]_X}} + (1 - \cos(\theta)) \cdot \underline{\underline{[\widehat{n}]_X}}^2 \qquad (4.11)$$

$$\underline{\underline{[\widehat{n}]_X}} = \begin{bmatrix} 1 & -\widehat{n}_z & \widehat{n}_y \\ \widehat{n}_z & 1 & -\widehat{n}_x \\ -\widehat{n}_y & \widehat{n}_x & 1 \end{bmatrix} \qquad (4.12)$$

Efter at rotations matricerne er beregnet kan vinkelændringen tilføres ved at multiplicere matricerne, ved brug af formel 4.13, hvorefter den endelige matrice kan konverteres tilbage til vektorformat ved brug af formel 4.14 og 4.15.

$$\underline{\underline{R_{\text{efter}}}} = \underline{\underline{R_{\text{før}}}} \cdot \underline{\underline{R_{\text{ændring}}}} \qquad (4.13)$$

$$\theta = \cos^{-1}\left(\frac{\text{trace}\left(\underline{\underline{R}}\right) - 1}{2}\right) \qquad (4.14)$$

$$\widehat{\underline{n}} = \frac{1}{2 \cdot \sin(\theta)} \cdot \begin{bmatrix} R(3,2) - R(2,3) \\ R(1,3) - R(3,1) \\ R(2,1) - R(1,2) \end{bmatrix} \qquad (4.15)$$

$$\text{trace}\left(\underline{\underline{R}}\right) = R(1,1) + R(2,2) + R(3,3) \qquad (4.16)$$

Rodrigues formler er indbygget i Matlab i form af funktionerne *vrrotvec2mat* og *vrrotmat2vec*.

Med den beregnede orientering af robottens tool point i forhold til spillebrættet, var "den lille modstander" i stand til at gribe de krydsformede brikker i felterne. For de cirkelformede brikker, eller brikker uden for felterne, var det nødvendigt at beregne den givne briks orientering.

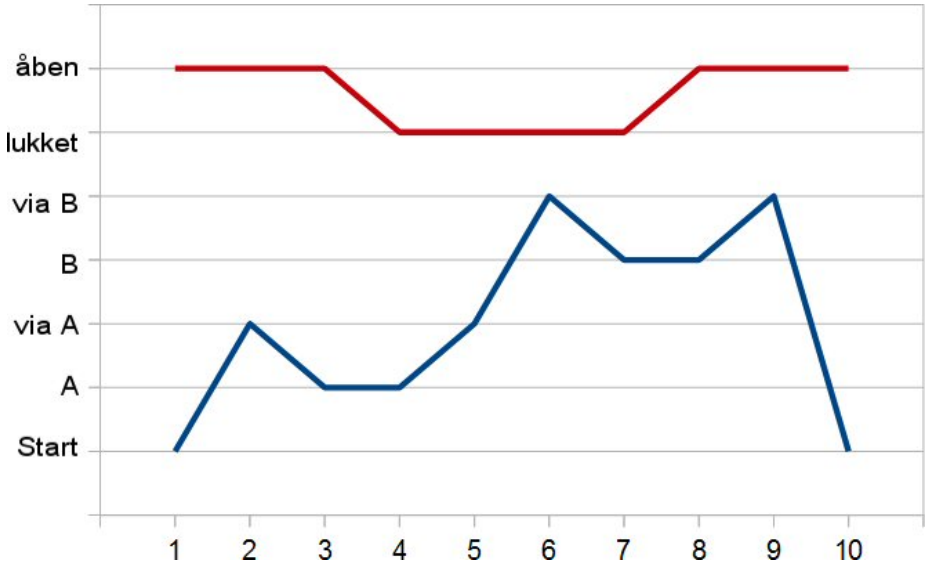
4.3 Program

Bevægelsesmønsteret "den lille modstander" benytter, når den spiller kryds og bolle, er relativt simpelt.

Robotten starter i en position væk fra spillebrættet, hvilket giver kameraet frit syn til dette. Når det er "den lille modstanders" tur til at lave et træk, og dette er beregnet, flyttes robotten hen til det første felt hvor en brik gribes. Herefter flyttes robotten til det andet feltet, hvor griberen løslader brikken. Endeligt afsluttes programmet ved at robotten flyttes tilbage til dens startposition.

Når robotten flyttes til eller fra et givent felt, går den igennem et via punkt i en fast højde over selve feltet. Dette via punkt er tilføjet for at sikre at robotten ikke støder på spillebrættet eller andre brikker, når den bliver flyttet imellem felterne.

Et sekvens skema over robottens bevægelser kan dermed opstilles, som vist på fig: 4.6, hvor punkt A og B er felterne hvor en brik henholdsvis ønskes at blive taget fra og placeret i.



Figur 4.6: *Sekvens skema for robottens bevægelser.*

Under implementering af denne rute blev det fundet at "den lille modstander" tog billeder af spillebrættet før robotten havde nået sin startposition. Årsagen til dette problem skyldes at Matlab valgte at tage billeder af spillebrættet, før robotten havde nået sin startposition og givet kvitteringsignal for dette. Dette problem blev løst ved at indsætte en pause på 1 [ms] efter kvitteringsignalet blev modtaget, hvilket sikrede at Matlab ventede på at robotten havde nået sin startposition.

Kapitel 5

Vision

I dette kapitel vil processen med at identificere spillebrættet, samt brikkerne, i billedet. Denne proces følger den klassiske model.

Under denne proces bliver et billede taget. Billedet bliver herefter forbehandlet, hvilket involvere korrigering af forvrængning samt reduktion af støj. Herefter vil features i billedet blive fundet, hvilket kan benyttes til at segmentere de ønskede objekter fra resten af billedet.

5.1 Kamera kalibrering

Første trin i billedbehandling er at sikre at det anvendte kamera er kalibreret korrekt. En af årsagerne til forvrængede billeder kan være linsen, som kameraet benytter. Et klassisk eksempel af billedforvrængning på grund af kameralinsen, er billeder taget med en såkaldt fiskeøje linse, som vist på fig: 5.1.

Figur 5.1: *Forvrænget billede taget med fiskeøjelinse.*

Problemet med denne type forvrængning er at rette linjer i scenen ikke vil være rette i billedet. Dette komplicerer dermed beregninger der er afhængige af rette linjer i scenen. Dette er særligt gældende for "den lille modstander" , givet nogle af spillebrættets tydeligste kendetegn er dets rektangulære form og kvadratiske felter.

Det er dermed nødvendigt at kalibrere kameraet før billedbehandlingen kan begynde, for at minimere forskelle imellem scene og billeder.

Til at kalibrere kameraet benyttes en app der er indbygget i Matlab, som beregner de interne og eksterne parametre for kameraet. [MathWorks]

De interne parametre benyttes til at korrigere forvrængning af billedet, hvorfor disse kan benyttes for at transformere fig: 5.1 til fig: 5.2 på den følgende side.

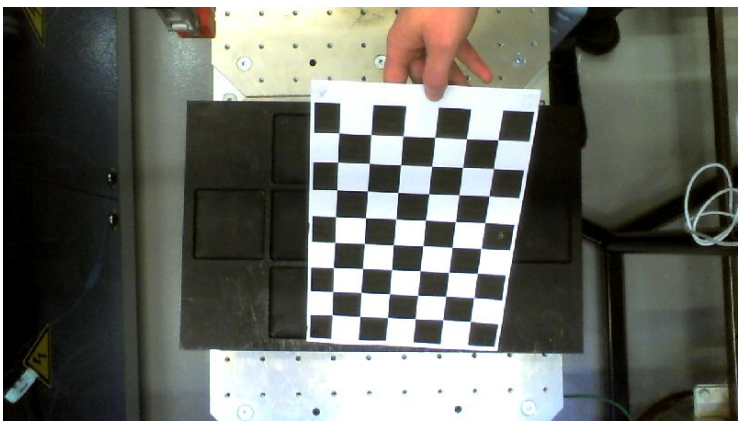
De eksterne parametre benyttes til at identificere kameraets placering i rummet, hvilket dermed kan benyttes til at transformere koordinater i *image plane* til *world space*.

Figur 5.2: *Korrigeret billede taget med fiskeøje linse.*

Til denne kalibrering kræver Matlab imellem $10 - 20$ billeder af et skakbræt med forskellig position og orientering i scenen. Yderligere er det vigtigt at have billeder hvor skakbrættet er placeret i det plan der ønskes anvendt således, at de eksterne parametre kan blive beregnet for dette.

Brugen af et skakbræt til denne kalibrering skyldes at hjørnerne af felterne er lette at identificere automatisk på grund af deres kontrastforskel. Efter hjørnerne er identificeret kan de rette linjer i skakbrættet beregnes, hvilke kan benyttes til at beregne skakbrættets orientering i rummet. For beregningerne af skakbrættets placering i *world space* kræver Matlab størrelsen af dets felter.

For at minimere riskoen for fejl under denne kalibrering, blev det anvendte skakbræt påhæftet et tykt lag pap, som vist på fig: 5.3.

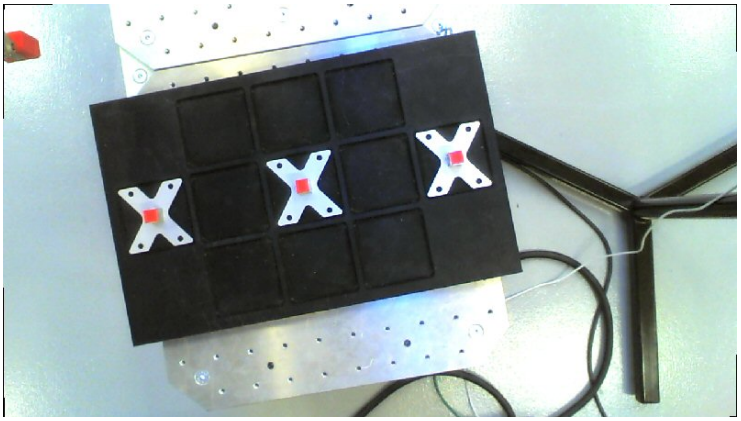


Figur 5.3: *Skakbrættet der blev benyttet til at kalibrere kamera parametrene.*

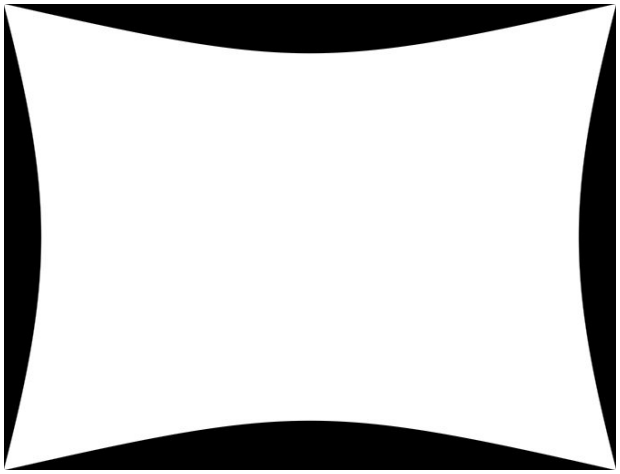
Efter de interne parametre var beregnet kunne disse blive benyttet til at korrigere billederne fra kameraet, "den lille modstander" benytter. Som det kan ses på fig: 5.4 på modstående side har de korrigerede billeder sorte pixels langs kanten. Dette er mere tydeligt på fig: 5.2, der krævede en mere ekstrem korrigering.

For at sikre disse sorte pixels ikke gav fejl under billedbehandlings processen, blev der lavet en maske til denne korrigering. For fig: 5.2 ville denne maske se ud som fig: 5.5 på næste side.

En maske er i form af et binært billede. Når denne anvendes bliver alle pixels i det sorte område af masken



Figur 5.4: *Korrigeret billede af scene.*



Figur 5.5: *Eksempel på maske til at fjerne tomme pixels.*

nulstillet, hvorimod pixels i det hvide område vil blive bevaret. Grunden til at denne maske er nødvendig, skyldes at de sorte pixels langs kanten kan have indflydelse på senere billedbehandlings processer. Ved dermed at benytte masken efter en given billedbehandlingsprocess, vil eventuelle fejl, i områderne uden for masken, blive slettet.

For "den lille modstander" blev de eksterne parametre ikke benyttet. Grunden til dette skyldes at disse parametre er afhængige af kameraets position i *world space*. Det blev derfor valgt at benytte koordinaterne, robotten gjorde brug af, til at kalibrere *robot space* til *board space*.

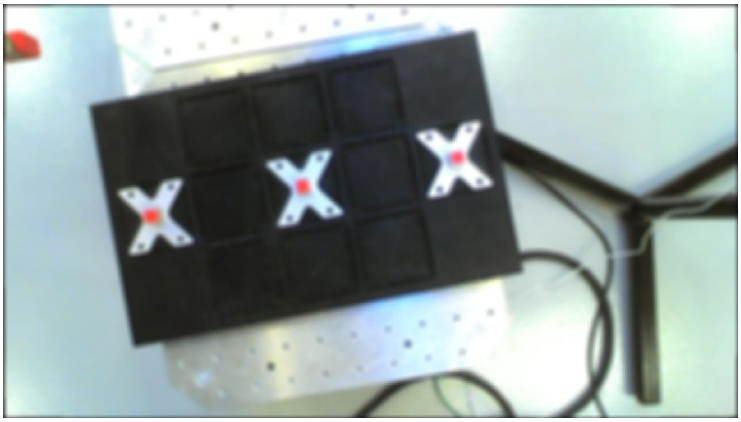
Ved at benytte disse koordinater var det krævet at de samme kalibreringspunkter blev identificeret i billedet. Med disse kalibreringspunkter kunne interpolering herefter benyttes til at beregne koordinater i *image plane* til *board space*.

I tilfælde af at kameraets position blev ændret i forhold til spillebrættet, vil det dermed kun være nødvendigt at identificere spillebrættets hjørnepunkter, for at sikre en korrekt transformation af koordinater i *image plane* til *robot space*.

5.2 Støj filtre

Selv efter kameraet er blevet kalibreret, således at dets billeder ikke længere er forvrængede, kræves det at billederne bliver forbehandlet før den egentlige billedbehandling. Dette skyldes at billederne kan indeholde støj, enten på grund af fejl i kameraets sensor eller lysforholdene i scenen.

Denne støj er ofte højfrekvent, hvorfor et lavpasfilter er nødvendigt. Til dette formål kan et Gaussian filter benyttes. Her skal gøres opmærksomt på, at selvom højfrekvent støj vil blive fjernet, slører Gaussian filteret også billedet. Et eksempel på dette kan ses på fig: 5.6



Figur 5.6: *Slørret billede på grund af for kraftigt Gaussian filter.*

Et filter er i form af en matrice med et ulige antal rækker og søjler, hvis størrelse og værdier afgør dets funktion. Eksemplet vist i formel 5.1 på næste side er et Gaussian 5×5 filter, med $\sigma = 1$. Det er her underforstået at værdierne i denne matrice er en vægtning, hvorfor den reelle værdi af et element kan findes ved at dividere det med summen af matricen.

$$\frac{1}{272}\cdot\begin{bmatrix}1&4&7&4&1\\4&16&26&16&4\\7&26&41&26&7\\4&16&26&16&4\\1&4&7&4&1\end{bmatrix}=\begin{bmatrix}0.0037&0.0147&0.0257&0.0147&0.0037\\0.0147&0.0588&0.0956&0.0588&0.0147\\0.0257&0.0956&0.1507&0.0956&0.0257\\0.0147&0.0588&0.0956&0.0588&0.0147\\0.0037&0.0147&0.0257&0.0147&0.0037\end{bmatrix} \tag{5.1}$$

Måden et filter benyttes er ved at lægge det over en sektion af billedet, der skal filtreres. Herefter multipliceres hvert element i filteret med den underliggende pixel værdi i billedsektionen. Summen af disse multiplikationer gemmes i centeret af en tilsvarende sektion i det resulterende billede.

Et eksempel på denne operation kan ses i formel 5.2, hvor et 3×3 median filter bliver benyttet på et 4×4 billedsektion.

$$\begin{bmatrix}4&4&3&2\\4&3&3&0\\3&3&0&0\\2&0&0&0\end{bmatrix}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}=\begin{bmatrix}-&-&-&-\\-&3&2&-\\-&2&1&-\\-&-&-&-\end{bmatrix} \tag{5.2}$$

Som det ses bliver pixels langs kanten i det resulterende billedsektion ikke tildelt nogen pixel værdi. Dette skyldes at filteret benytter en 3×3 matrice, mens dets resultat kun er 1 pixel, hvorfor det ikke er muligt at centrere filteret over pixels langs kanten.

Det er ukendt hvordan Matlab er i stand til at beregne værdierne for pixels langs kanten af billeder, hvilket dermed bevarer billedets størrelse. Fordelen ved dette er dermed at der ikke skal tages højde for tab af pixels når resultaterne fra flere filtre kombineres.

5.3 Gråtone billeder

Farvebilleder består af 3 lag, et for hver af de 3 farvekanaler. For at benytte et filter, eller anden form for beregninger, på et farvebillede, er det derfor nødvendigt at gentage disse beregninger for hver farve.

Af denne grund benytter billedbehandling ofte gråtone billeder, eftersom disse kun benytter en enkelt farvekanal og dermed reducerer antallet af beregninger. For at konvertere et farvebillede til et gråtone billede benyttes formel: 5.3, hvilket bevarer den opfattede lysintensitet.

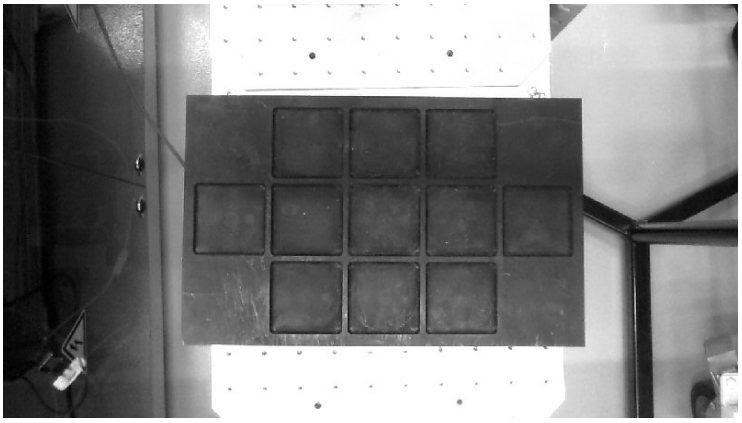
$$im_{gray} = 0.2989\cdot im_r + 0.5870\cdot im_g + 0.1140\cdot im_b \tag{5.3}$$

5.4 Threshholding

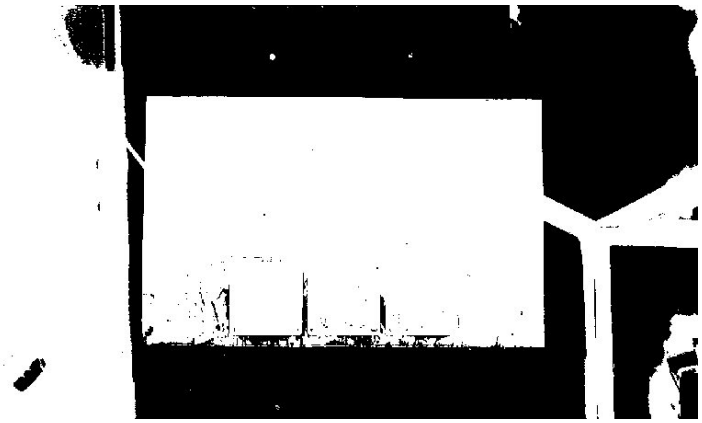
Efter billedet er blevet forbehandlet kan den egentlige billedbehandlingsopgave begynde.

Som første trin i denne proces, blev threshholding benyttet. Grunden til at denne metode blev anvendt, skyldes, at et af spillebrættets kendetegn er, at det er meget mørkere end bordet det er placeret på.

Threshholding benyttes til at konvertere et billede til binært format, hvilket er sort/hvid repræsentation. Ved brug af threshholding vælges først en grænseværdi, hvorefter hver pixel i billedet med en værdi under denne grænse blive hvide, mens de resterende bliver sorte. Et eksempel på dette kan ses på fig: 5.7.



(a) *Gråtone billede.*



(b) *Binært billede.*

Figur 5.7: *Billede før og efter threshholding blev benyttet.*

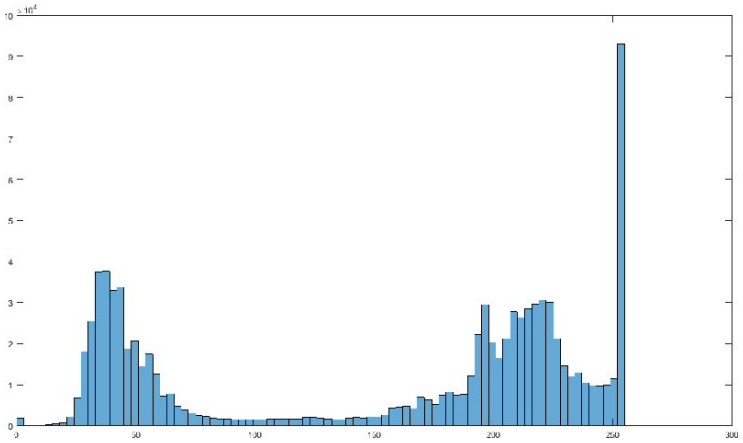
5.4.1 Otsu threshholding

Denne grænse for threshholding operationen kan beregnes ud fra en metode, kaldet Otsu threshholding. [Radke, Rich]

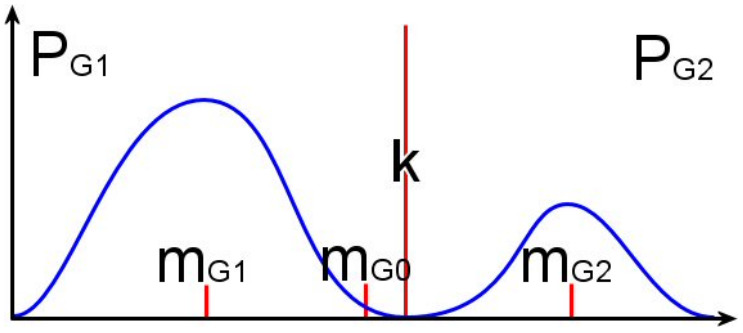
Denne metode tager udgangspunkt i histogrammet for et givent billede. Dette histogram viser fordelingen af pixel værdier for et givent billede, som vist på fig: 5.8 på næste side.

Under ideelle forhold vil dette histogram bestå af 2 områder med normaltfordelte pixel værdier. Det ene område vil være pixel værdier for baggrunden, mens det andet vil være pixel værdier for objektet, der ønskes at blive segmenteret. Under disse ideelle forhold, som vist på fig: 5.9 på modstående side, vil et threshhold blive sat imellem de 2 områder og dermed segmentere objektet fra dets baggrund.

Denne ideelle model er hvad Otsu threshholding benytter til at beregne et threshhold for et givent billede. Dette threshhold vil opdele alle pixels i billedet i 2 grupper, givet ved formel 5.4 og 5.5 på side 44. Givet disse beregninger er sandsynlighedsberegninger, skal summen af disse være 1, som beskrevet ved



Figur 5.8: *Eksempel på histogram over pixel værdier.*



Figur 5.9: *Ideel model af Otsu threshholding*

formel 5.6

$$P_{G1} = \sum_{i=0}^{k} P_i \tag{5.4}$$

$$P_{G2} = \sum_{i=k+1}^{255} P_i \tag{5.5}$$

$$1 = P_{G1} + P_{G2} \tag{5.6}$$

Hvor:
P_{G1} er gruppen for pixel værdier under threshhold.
P_{G2} er gruppen for pixel værdier over threshhold.
P_i er sandsynligheden for $I(x,y) = i$.
i er en pixelværdi i intervallet $[0;255]$.

Ved at kende grupperingen af pixelværdier over og under threshhold, kan middelværdien for disse beregnes med henholdsvis formel 5.7 og 5.8. Ligeledes kan middelværdien for hele billedet beregnes med formel 5.9.

$$m_{G1} = \frac{\sum_{i=0}^{k} i \cdot P_i}{P_{G1}} \tag{5.7}$$

$$m_{G2} = \frac{\sum_{i=k+1}^{255} i \cdot P_i}{P_{G2}} \tag{5.8}$$

$$m_{G0} = \sum_{i=0}^{255} i \cdot P_i \tag{5.9}$$

Hvor:
m_{G1} er middelværdien for pixel under threshhold.
m_{G2} er middelværdien for pixel over threshhold.
m_{G0} er middelværdien for billedet.

Formel 5.10 opstilles ud fra ønsket om at maksimere afstanden imellem de 2 pixel grupper.

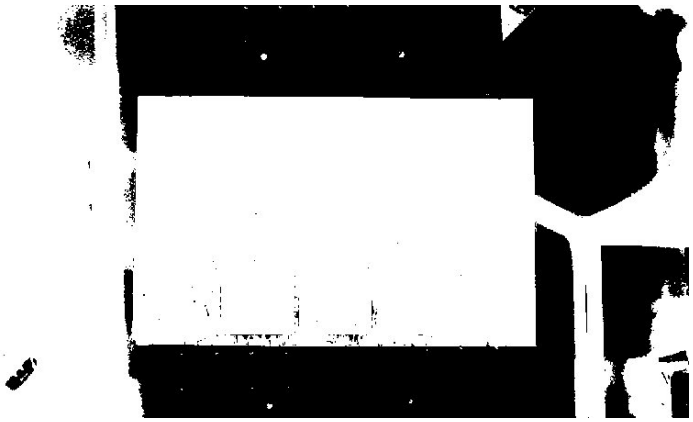
$$\sigma_k^2 = P_{G1}\left(m_{G1} - m_{G0}\right)^2 + P_{G2}\left(m_{G2} - m_{G0}\right)^2 \tag{5.10}$$

Hvor:
σ_k^2 er spredningen imellem de 2 grupperinger, for threshhold værdien k.

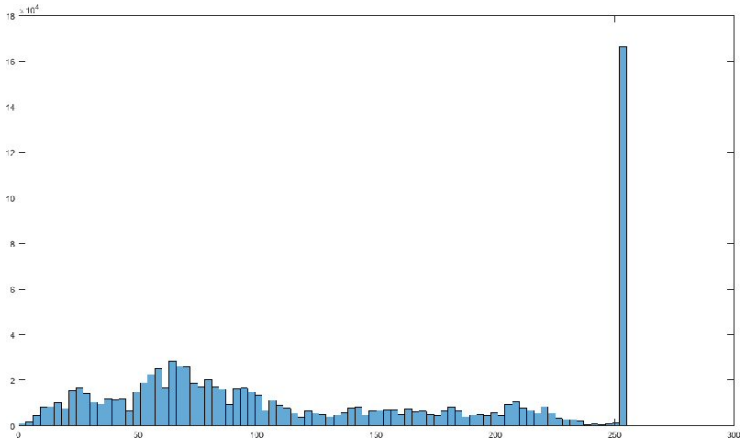
På grund af at der kun er 256 forskellige pixel værdier i et gråtone billede, kan den maksimale værdi for formel 5.10 beregnes ved at gennemgå alle mulige værdier af k.

Brugen af Otsu threshholding viste sig ikke altid at være ideel. På fig: 5.10 på næste side kan det ses at Otsu threshholding ikke gav en god grænseværdi til at segmentere objekterne i billedet.

Årsagen til denne fejl kan observeres ved at se på billedets histogram, som vist på fig: 5.11 på modstående side, hvilket har et højt antal pixel med den maksimale værdi.



Figur 5.10: *Binært billede fra Otsu threshholding.*



Figur 5.11: *Eksempel på histogram der ikke følger den ideelle model.*

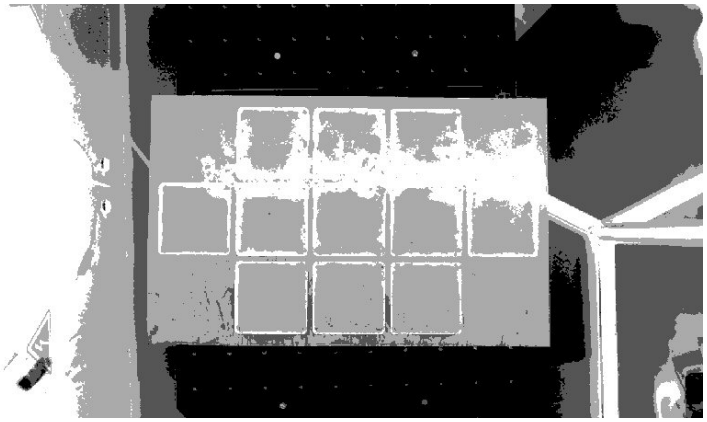
Denne bratte stigning skyldes bordet hvorpå spillebrættet er placeret. Dette bord er blankt, hvorfor dets pixel værdi overstiger den maksimale lysintensitet billedet tillader, hvilket gør at disse pixels bliver tildelt den maksimale værdi. På grund af at alle pixels på bordet har samme værdi, anser Otsu at dette område har en meget lav spredning. Dette resultere i at Otsu placerer dets threshhold tættere på denne top, og dermed tillader flere lyse pixel i det binære billede.

For at overkomme denne problemstilling benyttes en mere generel formel for Otsu threshholding, hvilket kan ses på fig: 5.11.

$$\sigma_k^2 = \sum_{n=1}^{N} P_n \left(m_n - m_{G0}\right)^2 \tag{5.11}$$

Hvor:
N er antallet af grupperinger for pixel værdierne.

For "den lille modstander" blev det valgt at beregne 3 threshhold, hvoraf kun nr. 2 blev benyttet. Resultatet for hvilke pixel, de enkelte threshhold ville have bevaret, i billedet kan ses på fig: 5.12.



Figur 5.12: *Billede lavet ud fra 3 Otsu threshhold.*

5.5 Edge detection

En ulempe ved thresholding er at det benytter billedets histogram, og dermed de globale pixel værdier, til at segmentere billedet.

For dermed at benytte lokale pixel til at segmentere billedet, blev det valgt at benytte edge detection. Den anvendte form for edge detection kaldes Canny edge detection, hvilket er en udvidet version af Sobel edge detection.

Af denne grund starter både Canny og Sobel edge detection med at benytte et Sobel filter på gråtone billedet. Som det kan ses ud fra formel 5.12 på næste side har dette filter negativ vægtning for nogle af dets elementer.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{5.12}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{5.13}$$

Disse elementer gør at Sobel filteret beregner forskellen imellem pixel værdier. Det skal her bemærkes at denne forskel kan være negativ, hvorfor resultatet af dette filter ikke bør gemmes i et billedformat.

Som det kan ses ud fra formel 5.12, beregner dette Sobel filter kun forskellen imellem pixel værdier langs en akse i billedet. Af denne grund benyttes et andet Sobel filter, der er drejet 90°, som vist i formel 5.13.

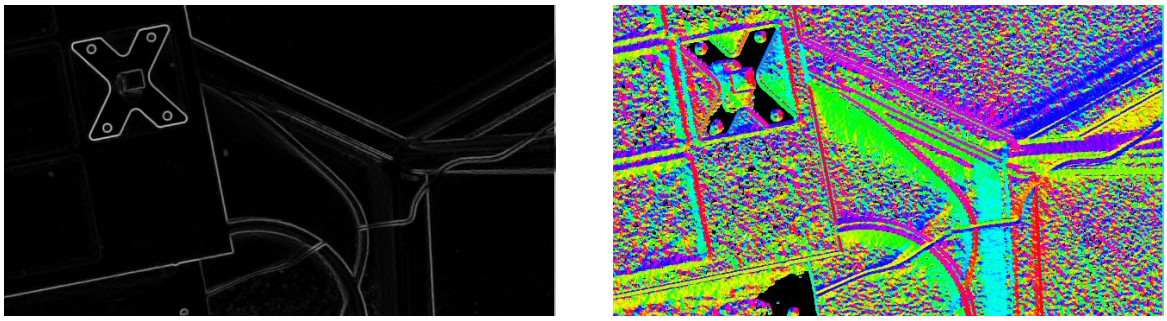
Disse 2 filtre benyttes i parallel, hvilket resulterer i 2 forskellige billeder, der hver beskriver forskellen i pixel værdier langs deres givne akse.

Størrelsen af pixel værdierne i disse billeder beskriver hvor tydelig en kant der er i det givne område, mens fortegnet beskriver orienteringen langs aksen. Det er dermed muligt at kombinere resultaterne fra begge Sobel filtre, hvilket giver en samlet vektor for hver pixel i *image plane*.

$$\|I\| = \sqrt{I_x^2 + I_y^2} \tag{5.14}$$

$$I_\theta = \text{atan}^{-1}\left(\frac{I_y}{I_x}\right) \tag{5.15}$$

Denne vektor kan herefter benyttes til at beregne størrelsen af en kant, samt dens orientering ved brug af henholdsvis formel 5.14 og 5.15. En visualisering af størrelsen og orientering af kanter vektoren kan ses på fig: 5.13.



(a) *Størrelse* **(b)** *Orientering*

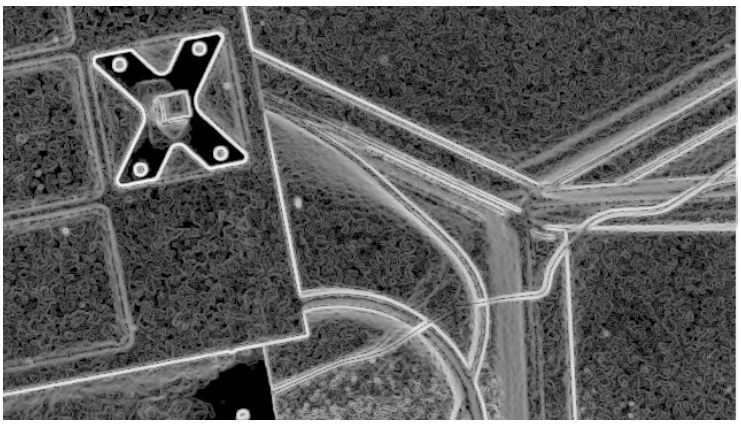
Figur 5.13: *Størrelse og orientering af kanter*

Billedet af orienteringen af kanter indeholder meget støj, da selv den mindste variation i billedet danner en kant og dermed bliver tildelt en orientering. Her skal det bemærkes at enkelte pixel er sorte i billedet, hvilket betyder at de ikke har en orientering. Disse pixel er hovedsageligt fra bordet, hvorpå spillebrættet

er placeret, og har den maksimale pixel værdi i det oprindelige billede. Af denne grund er der ingen variation omkring disse pixels, hvorfor disse ikke har en orientering.

Billedet af størrelsen af kanter kan ved første øjekast se meget ordentligt ud. Dette skyldes at gråtone billeder benytter kun 256 forskellige gråtoner, hvorimod værdien for en kant kan have langt større variation på grund af vægtningen af Sobel filtret.

Det er af denne grund nødvendigt at skalere kant værdierne, hvis disse ønskes at blive vist, hvilket skjuler støjen i områderne uden tydelige kanter. For bedre at illustrere indholdet i billedet kan der benyttes en logaritmisk skalering, som vist på fig: 5.14.



Figur 5.14: *Logaritmisk skallering af størrelsen af kanter.*

Efter at kanternes størrelse og orientering var fundet, kunne disse bruges til at lave et binært billede over linjerne i scenen. På dette billede ønskes det kun at bevare betydningsfulde linjer, samt at reducere deres bredde til 1 pixel.

Som udgangspunkt benyttes billedet over kanternes størrelse, hvilket er sløret på grund af det tidligere anvendte Gaussian filter. For at reducere bredden af linjerne i dette billede, anvendes billedet med kanternes orientering.

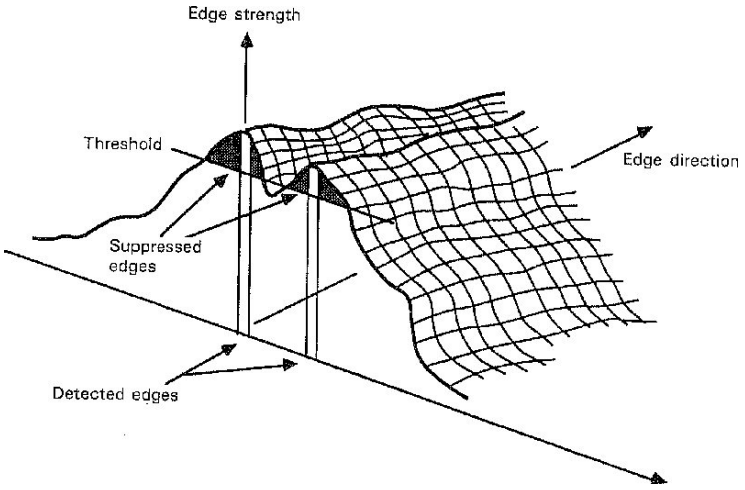
Ved at kende orienteringen af kanterne, som går på tværs af de ønskede linjer, kan der benyttes non maxima suppression til at frasortere kanter der ikke ligger i midten af en given linje, som vist på fig: 5.15 på næste side.

For Sobel edge detection ville næste trin være at identificere et threshhold for at segmentere de linjer med betydning fra dem, der er dannet på grund af støj.

Problemet med at vælge dette threshhold skyldes at der ønskes at bruge en lav værdi, hvilket reducerer risikoen for at frasortere ønskede linjer. Derimod ønskes det også at benytte et højt threshhold for at sikre at det binære billede ikke ville indeholde støj.

Af denne grund benytter Canny edge detection 2 threshhold med forskellige værdier. Linjer der har en værdi under begge threshhold vil automatisk blive anset som værende støj. Hvorimod linjer hvis værdi er over begge threshhold anses for at være stærke, og vil dermed automatisk blive bevaret.

Linjer der ligger imellem begge threshholds kaldes svage, hvorfor disse kun vil blive bevaret hvis de er i kontakt med stærke linjer.



Figur 5.15: *Non maxima supression. [Vernon, David,1991]*

Ved dermed at benytte 2 threshholds kan det ene sættes højt for at fjerne støj, hvorimod det andet kan sættes lavt og dermed sikre at svage linjer bliver bevaret.

5.6 Hough transformation

De linjer, der blev fundet ved brug af edge detection, kunne herefter benyttes til at identificere spillebrættet, eftersom et af dets kendetegn er dets rette linjer.

Til dette formål benyttes Hough transformation, hvilket er en metode til at identificere simpel geometri i et binært billede. I dette tilfælde ønskes det at identificere rette linjer i billedet, hvorfor formel 5.16 anvendes. [Sonka, Milan,2008]

$$y = a \cdot x + b \tag{5.16}$$

For at begrænse antallet af beregninger, gør Hough transformation brug af et diskret område for de anvendte parametre. Denne diskretisering er et problem for standardformlen for en ret linje, da værdien a vil være ∞ for lodrette linjer i billedet.

Af denne grund gør Hough transformation brug af polære koordinater til at beskrive rette linjer, hvilket gøres ud fra formel 5.17.

$$s = x \cdot \cos(\theta) + y \cdot \cos(\theta) \tag{5.17}$$

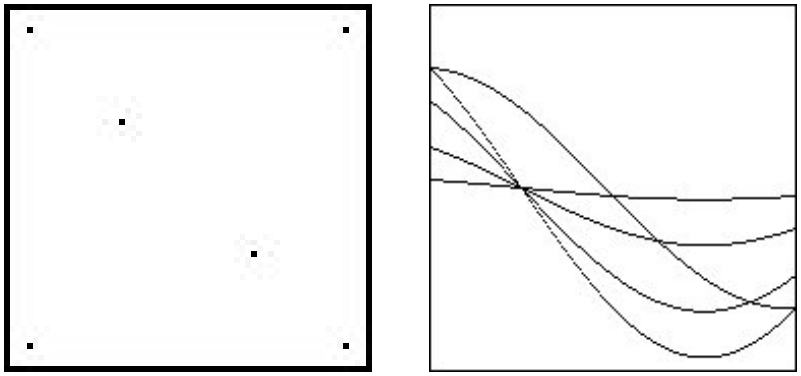
Hvor:
s er den ortogonale afstand imellem linjen og nulpunktet.
θ er vinklen for linjen.

For diskretiseringen benyttes hele grader for θ, med afgrænsningen $[-90°; 90°]$.

For afstanden *s* blev nulpunktet placeret i hjørnet af billedet, hvilket gav en minimums afstand på 0. Den maksimale afstand ville dermed være afstanden til det fjerneste punkt i billedet, hvorfor Pythagoras formel kan benyttes til at beregne dette.

For at identificere rette linjer i billedet blev et givent sæt parametre anvendt, hvorefter det blev målt hvor mange pixels der fulgte denne rette linje. Dette antal pixels blev gemt, hvorefter et nyt sæt parametre blev testet.

Som det kan ses på fig: 5.16 bliver en sinus kurve dannet for hver pixel i grafen for linjeparametrene. Herudover kan det observeres at disse kurver krydser hvis samme parametre kan benyttes til at tegne en ret linje imellem 2 forskellige pixel.



(a) *Sort hvidt billede* **(b)** *Hough parametre*

Figur 5.16: *Sort hvidt billede med 6 punkter og dets Hough linje parametre.*

Det er dermed muligt at identificere hvilke parametre der bedst beskriver en ret linje i billedet ud fra hvor mange pixels der ligger på den rette linje.

5.6.1 Beregnede linjer

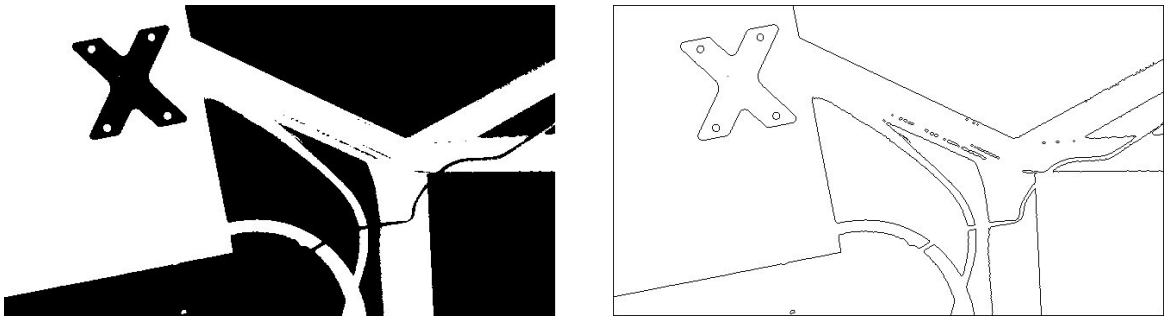
Udover at benytte Hough transformation på billedet fra edge detection blev det også valgt at anvende billedet fra farve threshholding. Dette billede gennemgik først en blob transformation før Hough transformation blev anvendt på dette.

Under denne blob transformation blev alle hvide pixel slettet, hvis alle dets nabo pixel også var hvide. Resultatet af denne transformation var at kun omridset af de enkelte blobs blev bevaret i billedet. En illustration af denne transformation kan ses på fig: 5.17 på næste side.

Grunden til dette valg skyldes at billedet fra edge detection bevarede flere linjer i billedet, men ofte var det nødvendigt at identificere op til 20 linjer før kanterne af spillebrættet blev identificeret.

I modsætning til dette var ikke alle af spillebrættets kanter synlige på billederne fra farve threshholding. Derimod var det kun nødvendigt at benytte et relativt lavt antal linjer for at sikre at alle kanter i billedet blev identificeret.

Ved at udvælge linjer fra farve threshholding var der af denne grund større sandsynlighed for at udvælge en linje langs spillebrættets kant. Hvorefter linjer fra edge detection kunne blive tilføjet for at sikre at det var muligt at identificere de 4 linjer der beskrev spillebrættets kant.



(a) *Sort hvidt billede* **(b)** *Omridset*

Figur 5.17: *Sort hvidt billede og dets omrids.*

5.7 Identifikation af spillebræt

Efter de rette linjer i scenen var identificeret, var næste opgave at identificere hvilke af disse 4 linjer der tilhørte spillebrættets kant.

På grund af at nogle linjer var baseret på Canny edge detection, betød dette at der var linjer der gik igennem midten af spillebrættet. For at frasortere disse blev det valgt at tælle hvor mange hvide pixels der lå langs hver linje, baseret på det binære billede af scenen.

Herefter blev dette binære billede eroderet, hvorefter antallet af hvide pixels omkring linjen igen blev talt.

Linjer der lå langs kanten af spillebrættet ville dermed have en stor variation af hvor mange pixels der blev talt, mens linjer der gik igennem spillebrættet ville have en meget lille variation.

Herefter blev også linjer der ikke lå nær et tilstrækkeligt antal hvide pixel også frasorteret.

Yderligere blev det sikret at linjer, hvis kanter havde samme orientering, ikke kunne blive benyttet sammen.

5.7.1 Søgeteknik

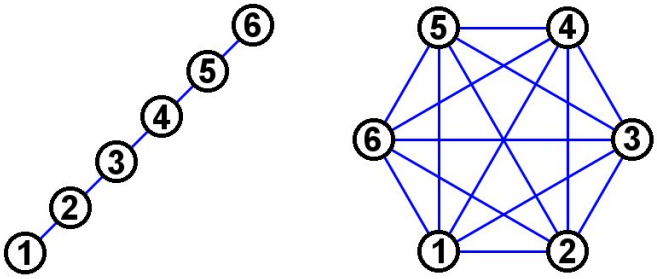
Med enkelte linjer frasorteret var næste trin at identificere hvilke firkanter der kunne blive dannet ud fra linjerne de resterende linjer.

Til dette formål blev det forsøgt at benytte grafteori, hvor krydspunkter imellem 2 linjer var knudepunkterne og selve linjerne var kanter. Her skal det bemærkes at en knude på en given linje ville have en kant til alle andre knuder på samme linje, som vist på fig: 5.18 på den følgende side.

Det blev dog erfaret at grafteori ikke var velegnet til denne opgave.

Det første problem ved denne metode var at en firkant ville være i form af en løkke med 4 knuder, hvilket kræver opmærksomhed når søgekriterierne specificeres.

Den anden problemstilling var at hver knude havde kant til alle punkter på samme linje, hvilket gjorde at grafen voksede eksponentielt efter hver besøgt knude. Hvis hver linje har 5 knuder vil første iteration



(a) *Punkter på linje* **(b)** *Graf af punkter på linje*

Figur 5.18: *Punkter på en linje, samt deres graf repræsentation.*

resultere i 5 endepunkter, mens anden iteration ville resultere i 25 og tredje iteration i 125. Flere af disse endepunkter vil være ens, da disse kan tilhøre forskellige løkker.

På grund af at det var ønsket at identificere en løkke kunne *width first* søgning ikke benyttes, eftersom startpunktet ville være afskåret efter første søgning. *Depth first* søgning var i stand til at løse opgaven, ved at begrænse dybden til 3 knuder, men var enormt tidskrævende på grund af grafens størrelse.

Af denne grund blev det valgt at konstruere en søgeteknik for denne problemstilling.

Denne søgeteknik benyttede ligeledes knudepunkterne fra grafteori, men benyttede dem til at opstille en matrice over hvilke linjer der krydsede. Denne binære matrice benyttede linje numrene som række og søjle index. Hvis 2 linjer krydsede ville deres index være 1, mens det ville være 0 for linjer der ikke krydsede.

På grund af en firkants geometri ville denne bestå af 4 knudepunkter og 4 linjer. Hvis 2 af disse knudepunkter blev udvalgt, hvorefter linjerne der gik igennem dem blev noteret, ville kun modstående hjørnepunkter kunne benyttes til at identificere alle 4 linjer.

Det kunne dermed verificeres om 2 tilfældigt udvalgte knuder var en del af samme firkant ved at verificere at deres linjer krydsede hinanden. Her skal det bemærkes at denne verifikationsproces kun identificerede modstående knuder i firkanten, hvorfor alle knudepar blev testet.

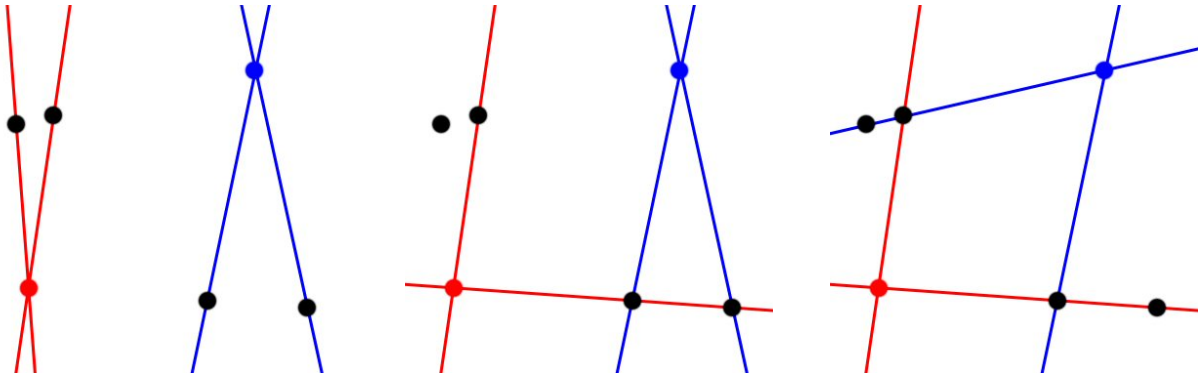
Til at verificere om linjer krydsede blev den tidligere nævnte matrice anvendt. Under denne verifikation var det nødvendigt at checke begge de mulige linjekombinationer, eftersom det var muligt for linjen fra en knude at krydse begge linjer fra den anden knude uden at den anden linje havde et krydspunkt. De mulige resultater af denne verifikationsproces kan ses på fig: 5.19 på næste side.

På grund af at denne søgeteknik benyttede modstående hjørnepunkter, ville 2 forskellige knudepar identificere samme firkant. Af denne grund blev listen af firkanter sorteret baseret på dets linjenumre, hvormed dubletter kunne blive slettet.

5.7.2 Identifikations proces

Med en liste over mulige firkanter i billedet, kunne information fra de enkelte linjer blive benyttet på firkanten som en helhed. Information såsom orienteringen af kanter langs de enkelte linjer kunne herefter benyttes til at frasortere firkanter hvor flere kanter havde samme orientering.

Mindre specifik information, såsom antallet af pixel omkring knudepunkterne, hvilket kunne benyttes til



(a) *Ingen krydspunkter* **(b)** *Krydspunkter på 1 linje* **(c)** *Krydspunkter på begge linjer*

Figur 5.19: *De 3 mulige scenarier for identifikationen af firkanter.*

at beskrive hjørner, kunne benyttes til at prioritere de enkelte firkanter.

Ud fra alle knudepunkterne i de givne firkanter, kunne deres areal beregnes ud fra formel 5.18, hvilket kan benyttes til at beregne arealet af en given polygon. [Math Open Reference]

$$A = \left| \frac{(x_1 \cdot y_2 - x_2 \cdot y_1) + (x_2 \cdot y_3 - x_3 \cdot y_2) \ldots + (x_n \cdot y_1 - x_1 \cdot y_n)}{2} \right| \tag{5.18}$$

Hvor:
A er arealet for den given polygon.
n er antallet af hjørner i den givne polygon.

Denne formel kræver at hjørnepunkterne blive benyttet enten i en rækkefølge, der følger urets retning eller er modsat denne, hvorfor den tidligere benyttede søgeteknik var gavnlig. På grund af at denne søgeteknik gjorde brug af modstående knudepunkter, kunne disse anvendes som første og tredje punkt i formlen og dermed garantere at rækkefølgen ville være korrekt.

Som sidste led i identifikationsprocessen blev de forskellige firkanter sorteret efter deres størrelse. Denne størrelse var vægtet ud fra mindre specifik information om firkanten, hvilke ikke kunne benyttes direkte til at godkende eller frasortere enkelte firkanter.

5.8 Identifikation af brikker

Ved at kende hjørnerne af spillebrættet i *image plane* kunne beregningerne i afsnit 4.1 på side 31 blive benyttet til at finde hjørnerne for hvert felt på spillebrættet.

Disse hjørner kunne herefter benyttes til at segmentere felterne, hvorefter deres indhold kunne blive analyseret. På grund af kontrasten imellem brikkerne og spillebrættet, var det tilstrækkeligt at benytte blob detection.

Som tidligere beskrevet har hvert felt på spillebrættet 3 mulige tilstande. Enten er det tomt eller også indeholder feltet en brik der er udformet som et kryds eller en cirkel.

Det kunne hurtigt identificeres om et givent felt var tomt, eftersom alle dets pixel i dette tilfælde ville være hvide. Her skal det bemærkes at det binære der benyttes oprindelig var beregnet til at identificere spillebrættet, hvorfor farverne er inverteret i forhold til hvad der forventes.

(a) *Kryds* **(b)** *Tom* **(c)** *Cirkel*

Figur 5.20: *Sort hvidt billede af hver af de 3 mulige tilstande for et givent felt på spillebrættet.*

For at skelne imellem de 2 forskellige brik typer, blev en morfologisk transformation benyttet. Denne transformation fjernede alle hvide pixels der havde kontakt med kanten af det segmenterede billede. Efter denne transformation var det let at skelne imellem brik typerne ud fra antallet af hvide pixel, eftersom de cirkel formede brikker efterlod 3 store blobs.

(a) *Kryds* **(b)** *Tom* **(c)** *Cirkel*

Figur 5.21: *De 3 mulige tilstande for et givent felt på spillebrættet, efter hvide pixel langs kanten er fjernet.*

Fordelen ved denne identifikations proces var at den var uafhængig af brikkernes orientering. Herudover blev den gjort uafhængig af skalering, ved at tælle antallet af hvide pixel som procentdel af det givne felt.

Procentdelen af hvide pixels i et givent pixel, både før og efter transformationen, kunne dermed benyttes til at beregne sandsynligheden for hver af de 3 tilstande, hvorefter den mest sandsynlige blev valgt.

Kapitel 6

Samling af system

Som sidste del i udviklingen af "den lille modstander" var det nødvendigt at få de individuelle systemer til at arbejde sammen.

6.1 Identifikation af tur

Til at identificere hvornår det var "den lille modstanders" tur, blev det valgt at tage billeder af spillebrættet med et interval på 1 [s].

Ud fra denne proces kunne det herefter beregnes hvilke ændringer der var nødvendige for at få spillebrættet i "den lille modstanders" hukommelse til at matche spillebrættet på billedet. Det kunne ud fra dette beregnes hvor mange træk der var nødvendige for at lave de observerede ændringer.

Hvis disse ændringer stemte overens med at modspilleren havde lavet et træk, ville "den lille modstander" beregne og udføre sit næste træk.

6.2 Afslutning af spil

Hvis "den lille modstander" identificerede 3 ens brikker i en stribe, ville "den lille modstander" benytte en af 2 tilgængelige bevægelses mønstre.

Hvis "den lille modstander" ikke benyttede brikkerne, ville den lukke griberen og sænke den. Denne bevægelse var beregnet til at signalere at "den lille modstander" var trist over at have tabt.

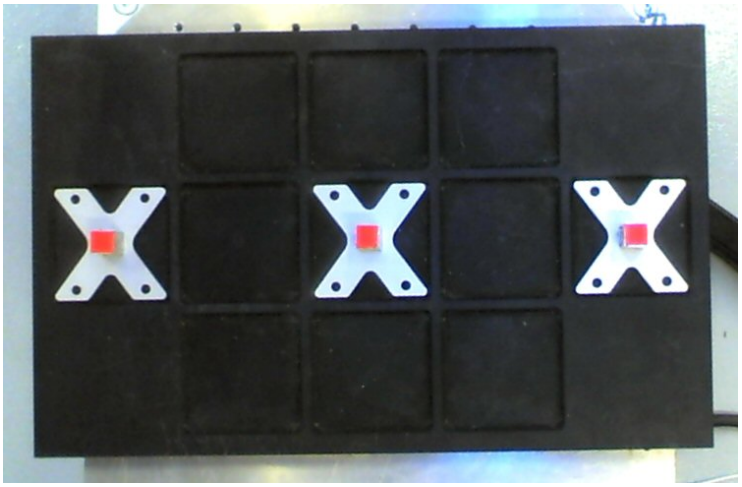
Hvis "den lille modstander" derimod benyttede de 3 brikker i samme stribe, ville den bevæge griberen frem og tilbage over striben for at signalere at den havde vundet.

Kapitel 7

Konklusion

Det lykkedes igennem projektet at få "den lille modstander" til at opfylde alle de ønskede minimumskrav, hvilket gjorde den i stand til at spille kryds og bolle. Dette indebar at UR robotten blev styret via matlab, og at den kunne identificere brik typerne på spillebrættet og beregne et træk ud fra disse, hvorefter dette blev udført.

Dog skete dette ikke på det ønskede niveau, eftersom "den lille modstander" er begrænset til kun at benytte de krydsformede brikker, samt at disse skal være i de specifikke positioner som vist på fig: 7.1.



Figur 7.1: *Den krævede startposition af brikker.*

Trods denne begrænsning var det muligt at fremvise "den lille modstander" for en konference der blev holdt af IDA, hvor den var i stand til at spille kryds og bolle med gæsterne. Denne interaktion skete, efter ønsket, uden input fra gæsterne om hvornår det var "den lille modstanders" tur.

For styring af UR robotter via Matlab blev kontrolleren ikke færdiggjort, hvilket mangler en brugerflade for at dette opnåede det ønskede niveau. Selve kommandoerne for denne kontroller er færdiggjorte, og kan dermed anvendes i programmer til at styre flere af UR robotterns funktioner.

Blandt de øvre kravspecifikationer blev enkelte kun delvist opfyldt. Det var muligt at gøre UR robotten i stand til at korrigere for når den blev skubbet, men dette er ikke blevet implementeret i programmet. Ligeledes var det muligt at identificere spillebrættet automatisk. Dog blev det valgt ikke at gøre brug af dette uden input fra brugeren, på grund af fejlraten af denne identifikationsprocess.

Kapitel 8

Perspektivering

For bedre at opfylde kravende for “den lille modstander” , burde opgaven med at identificere spillebrættet være simplificeret.

Denne opgave voksede eftersom “den lille modstander” ikke havde en fast position i værkstedet. Dette betød at baggrunden var stærkt varierende og det var dermed besværligt at finde en metode der var i stand til pålideligt at identificere spillebrættet.

Dette problem kunne have været reduceret ved at tilføre markeringstape langs kanten af spillebrættet, hvilket ville have givet en mere tydelig kant. Andre metoder kunne have gjort brug af en fast baggrund, eller mærkater der ville være lettere at identificere.

Valget om at forsøge at løse problemstillingen, uden at ændre på opstillingen, reducerede dermed ressourcerne til de resterende opgaver i projektet.

Yderligere blev det også valgt i løbet af projektet at udvide den anvendte kode til at styre UR robotten. Dette valg blev hovedsageligt baseret på at gøre det lettere for senere grupper at arbejde med robotten, eftersom de tilføjede features var langt over hvad der var krævet af dette projekt.

Litteratur

[Fereshteh, Aalamifar] Aalamifar Fereshteh. *UR5 Control Using Matlab*, 2015.
URL: https://se.mathworks.com/matlabcentral/fileexchange/50655-ur5-control-using-matlab.

[Math Open Reference] Math Open Reference. *Area of a polygon (Coordinate Geometry)*, 2011.
URL: http://www.mathopenref.com/coordpolygonarea.html.

[MathWorks] MathWorks. *Camera Calibration with MATLAB*, 2013.
URL: https://se.mathworks.com/videos/camera-calibration-with-matlab-81233.html.

[Radke, Rich] Rich Radke. *DIP Lecture 12: Thresholding*, 2015.
URL: https://www.youtube.com/watch?v=ojapO75FV38.

[RG2 User Manual] On Robot. *RG2 - User Manual*, 2015.
URL: http://www.onrobot.dk/onewebmedia/RG2%20User%20Manual%20V1.1.pdf.

[Sonka, Milan,2008] Vaclav og Boyle Roger Sonka, Milan og Hlavac. *Image Processing, Analysis, and Machine Vision*. Thomson Learning, 2008, 3 edition.
ISBN: 0-495-24438-4. 829 pp.

[Szeliski, Richard,2010] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer London, 2010, 1 edition.
ISBN: 9781848829343. 957 pp.

[UR5 User Manual] Universal Robots. *UR5 - User Manual*, 2014.
URL: https://www.universal-robots.com/media/8704/ur5_user_manual_gb.pdf.

[URScript Manual] Universal Robots. *The URScript Programming Languagel*, 2012.
URL: http://www.sysaxes.com/wp-content/uploads/2013/12/scriptmanual_en.pdf.

[Vernon, David,1991] David Vernon. *Machine Vision: Automated Visual Inspection and Robot Vision*. Prentice Hall, 1991, 1 edition.
ISBN: 0-13-543398-3. 260 pp.

[Wikipedia] Wikipedia the free encyclopedia. *Tic-tac-toe*, 2017.
URL: https://en.wikipedia.org/wiki/Tic-tac-toe.

[Wildberger, N. J.] N. J. Wildberger. *ElemMath 13 (K-6) Explained: Logical reasoning*, 2012.
URL: https://www.youtube.com/watch?v=_pJI5FJVbfQ.

[Youtube Klip 1] SheepWillbeSheared. *Automate 2015 - Robotics Demonstrations, Part I*, 2015.
URL: https://www.youtube.com/watch?v=f1eoL4AxEe0.

[Youtube Klip 2] SheepWillbeSheared. *Automate 2015 - Robotics Demonstrations, Part II*, 2015.
URL: https://www.youtube.com/watch?v=n1DOnbM-caw.

[Youtube Klip 3] Inc. R.R. Floody Company. *20150520 185313*, 2015.
URL: https://www.youtube.com/watch?v=2RVtKNBsfpY.

[Youtube Klip 4] jc102885. *Robot can't play tic tac toe*, 2015.
URL: https://www.youtube.com/watch?v=FTaVJ-x3Rt0.

[Youtube Klip 5] Michael Overstreet. *Baxter the Robot playing Tic-tac-toe Game 4*, 2015.
URL: https://www.youtube.com/watch?v=3hHMVEch0cg.

Bilag A

PolyScope kode

(a) *Adskilt* **(b)** *Monteret*

Figur A.1: *PolyScope kode for basis Matlab pakken.*

Figur A.2: *PolyScope kode for det udvidede program.*