Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Informasjonsteknologi, kybernetikk og signalbehandling	Vårsemesteret, 2012 Åpen
Forfatter: Stein Tore Stegen	…… (signatur forfatter)
Fagansvarlig: Kjersti Engan Veileder(e): Kjersti Engan	
Tittel på masteroppgaven: Kamerabasert navigasjonssystem implementert i MATLAB for fjernstyring av mobil robot. Engelsk tittel: Camera-based navigation system implemented in MATLAB for remote steering of a mobile robot.	
Studiepoeng: 30	
Emneord: Terskling, binær morfologi, RGB fargemodell, objektdeteksjon, servomekanismer, modellering, P-regulator, posisjonering, reguleringssløyfer, modellfeil, Lego Mindstorm, MATLAB, blåtann	Sidetall: …… + vedlegg/annet: …… Stavanger, …….. dato/år

Forord

Denne rapporten omhandler utviklingen av programvare til implementasjon i en fysisk modell som kan anvendes i laboratoriesammenheng for fremtidige masterstudenter ved kybernetikklinjen ved UiS. Modellen demonstrerer praktisk bruk av bildebehandling og reguleringsteknikk, og er ment som en motivasjonskilde og trekkplaster for kybernetikklinjen.

For best mulig forståelse av materialet som skal presenteres anbefales noe forkunnskaper innen geometrisk matematikk, bildebehandling, reguleringsteknikk og praktisk bruk av MATLAB. Denne masteroppgaven ble påbegynt som et forprosjekt høsten 2011 og videreføres i denne rapporten. Gjennomgang av forprosjektrapporten vil derfor også være nyttig. Rapportens innhold forklares i detalj og er prøvd fremstilt på en enkel og forståelig måte.

Rapporten er skrevet i tekstformateringsprogrammet LaTex med teksteditingsverktøyet TexMaker. Alle simuleringer og beregninger er utført i matematikkprogrammet MATLAB. Videre er figurer laget i både MATLAB og Paint, forsiden ble editert i Word, mens Visio ble brukt til å lage blokkskjematiske figurer.

Per dags dato holder jeg på med siste semester av min mastergradutdannelse innen kybernetikk, og rapporten er skrevet som en masteroppgaven for UiS våren 2012.

Jeg må få takke min veileder Kjersti Engan for god hjelp og innspill underveis, Ståle Freyer for gode arbeidsforhold, praktisk hjelp og tilrettelegging på laben, samt min samboer for både lån av digitalkamera, og fantastisk støtte og motivasjon ved arbeid hjemmefra.

Sammendrag

Denne rapporten omhandler utvikling, konstruksjon og testing av en fysisk modell som både skal kunne brukes i laboratoriesammenheng for fremtidige kybernetikkstudenter ved Universitetet i Stavanger (UiS), og for å skape interesse rundt kybernetikklinjen ved skolemesser og presentasjoner. Praktisk løsning for modellen består av følgende komponenter: en mobil robot fra Lego Mindstorms NXT, to baller, et webkamera, og den fysiske konstruksjonen som definerer robotens arbeidsområde. Programvare som *styrer* roboten i arbeidsområdet er blitt utviklet, og testet i forhold til både funksjonalitet og presisjon ved implementering av praktisk oppgave i modellen.

Programvaren ble utviklet i, og implementeres fra MATrix LABoratory (MATLAB) på en ekstern datamaskin (PC). For kommunikasjon mellom MATLAB og Lego Mindstorm NXT, er det tidligere blitt utviklet en verktøykasse til MATLAB for dette formålet ved Universitetet i Aasden i Nederland [1]. Verktøykassen inneholder ferdiglagede MATLAB funksjoner for kommunikasjon med og kontroll av robotens sensorer og motorer. All kommunikasjon i modellen ble satt opp i forprosjektet som vist i appendiks A. Implementasjon av programvaren innebærer at roboten skal lokalisere sin egen posisjon og orientering i planet, samt ballenes posisjon, deretter posisjonere seg ved nærmeste ball, og slå denne mot den andre ballen. For å kunne utføre denne oppgaven ble design av roboten gjort med fokus på både gode manøvreringsegenskaper, og konstruksjon av en robust slagarm. Programvaren består i hovedsak av 2 hovedalgoritmer, en for objektgjenkjenning som beregninger nødvendige objektkoordinater, og en for posisjonering som fysisk styrer roboten til en ønsket posisjon. Disse algoritmene integrerer henholdsvis bildebehandling og reguleringsteknikk ved lokalisering og posisjonering av roboten. For å styre roboten med god presisjon er beregning av aktuelle objektkoordinater før og under programimplementeringen essensielt. Modellens kamera (som kobles til PC via en universell seriebuss (USB) kabel) innhenter nødvendig bildedata til MATLAB, som ved bildebehandling beregner nødvendige objektkoordinater. Denne algoritmen finner først bildets interesseområde (ROI), deretter segmenteres fargebildet inn i fargekomponentbilder som konverteres til binære bilder ved terskling. Binære morfologiske operasjoner fjerner støy og skiller ut de interessante objekter fra bildene med ulike teknikker og identifiserer objektets koordinat til slutt. Alle koordinater i systemet defineres ut fra bildeplanet etter utført

bildebehandling, og brukes som utgangspunkt for posisjoneringen. I bildeplanet fremstår alle objekter som en projeksjon av det opprinnelige objektet i scenen. Bildeprojeksjonen introduserer en feilmargin som resulterer i en misvisning av de beregnede koordinatene. Feilmarginen ble funnet som en funksjon av robotens posisjon i planet ved testing, og automatisk korreksjon for misvisningen er implementert i programvaren. Korreksjonen viste en dokumentert forbedret presisjon ved posisjonering av roboten. Flere faktorer spiller inn på programvarens stabilitet og presisjon. Varierende lysforhold gir utfordringer i forhold til riktig objektdeteksjon. Roboten posisjoneres ved bruk av to metoder: sving og drift rett fram. Disse metodene ble modellert og implementert i reguleringssløyfer, og kontrolleres med henholdsvis tilbakekobling og foroverkobling. Robotens svingprosess består av to lineære servomekanismer, og null statisk reguleringsavvik oppnås ved implementasjon av en Proporsjonal-regulator (P-regulator) i programvaren. Programvarens presisjon ble testet i forhold til posisjonering av roboten, samtidig som robotens treffsikkerhet ble dokumentert. Ulike treffpunkter for ball på slagarmen gir ulike ballbaner slik at robotens treffprosenten avhenger presis posisjonering. Dette kombinert med unøyaktighet i forhold til beregnede koordinater for posisjonering (selv med korreksjon for misvisning) gjør at treffsikkerheten faller raskt som en funksjon av avstanden mellom ballene. Rapportens testresultater tilsier at programvaren er funksjonell, men unøyaktigheter i modellen fører til en reduksjon i treffpresisjonen. Korreksjon for projeksjonsfeil i modellen gir alikevel en bedre treffprosent som følge av nøyaktigere robotposisjonering.

Forkortninger

2D	To dimensjoner
3D	Tre dimensjoner
BRDF	Bidirectional Reflectance Distribution Function (toveis fordelingsfunksjon for reflektert lys)
CD	Compact disc (digital lagringsenhet)
FOV	Field Of View (synsfelt)
IPT	Image Process Toolbox (verktøykasse for bildebehandling)
MATLAB	MATrix LABoratory
P,PI og PID	Proporsjonal Integrasjon Derivasjon
PC	Personal Computer (personlig datamaskin)
RGB	Rød Grønn Blå (fargemodell basert på nevnte primærfarger)
ROI	Region Of Interest (interesseområde)
UiS	Universitetet i Stavanger
USB	Universal Serial Bus (universell seriebuss)

Definisjoner og begrep

Binært bilde	Hvert bildepiksel representeres med 1 bit.
Blåtann	Protokoll for trådløs overføring av data via radiosignal.
Emittans	Utstrålingstetthet eller strålingseksitans, er forholdet mellom den energi som per tidsenhet sendes ut i form av stråling og arealet av strålekilden. Måles i watt/m2.
Gråskala- og itensitetsbilder	Hvert bildepiksels verdi representeres innenfor et definert intensitetsområde.
Modellen	Begrep som omfatter roboten, ballene, webkamera og den fysiske konstruksjonen.
Lego Mindstorms	Robotbyggesett m/programvare og hardware fra leketøyfabrikanten Lego.
NTX	Intelligent datamaskinstyrt brikke (Lego Mindstorm robotens hjerne).
Ortografisk projeksjon	Projeksjonslinjene danner rette vinkler med projeksjonsplanet.
Perspektiv projeksjon	Projeksjonslinjene danner en vinkel med projeksjonsplanet.
Planet	Beskrivelser robotens fysiske operasjonsområde
Programvare	Beskriver all programkode som er utviklet i MATLAB.
Prosjektet	Begrep for både modell og programvare som en enhet.
Scene	Det fysiske området som fotograferes med et kamera.
YUY2	Fargemodell basert på lysintensitet, fargetone og metning.

Innhold

Figurer

FIGURER

Tabeller

Kapittel 1

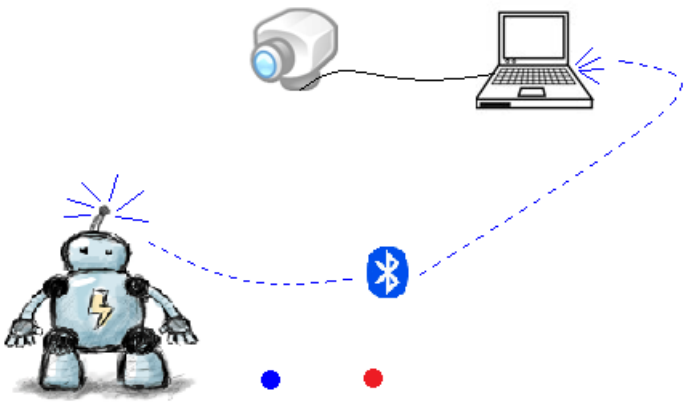
Innledning

Denne rapporten er skrevet som en masteroppgave ved og for Universitetet i Stavanger (UiS), våren 2012. Innhold og fagmomenter i oppgaven ble utformet i samarbeid med veileder etter framlegg av eget oppgaveønske. Oppgaven ble anerkjent som en faglig god og utfordrende masteroppgave med både praktiske og teoretiske momenter fra både reguleringsteknikk og bildebehandling. Oppgaven ble påbegynt som et forprosjekt høsten 2011, og arbeidet som fremstilles i denne rapporten er en videreføring av dette forprosjektet.

Hovedformålet med oppgaven er praktisk og teoretisk anvendelse av fagmomenter fra bildebehandling og reguleringsteknikk, der tilbakekobling, foroverkobling, terskling og morfologi er sentrale temaer som anvendes. Videre skal det ferdige produktet kunne brukes i en laboratoriesammenheng ved UiS for framtidige studenter.

Praktisk løsning for oppgaven ble konstruksjon av en fysisk modell og utvikling av relevant implementerbar programvare. Praktisk bruk av programvaren realiseres ved implementasjon av oppgaver i den fysiske modellen. Denne rapporten viser detaljert konstruksjon og implementasjon av utviklet programvare i modellen.

Figur 1.1 viser en skisse over modellen og hvordan kommunikasjonen er satt opp.



Figur 1.1: Modellskisse. Symbolene er hentet fra Internett for illustrasjons av modellens komponenter.

Modellen består av en robot, et webkamera, to baller, og konstruksjon som definerer modellens fysiske omfang (ikke illustrert i figuren). Datamaskinen (PC) er ikke definert som en del av modellen men brukes som et kontrollsenter for modellen. Programvare som skal implementeres i modellen finnes i PC som gjør nødvendige beregninger og kommuniserer kommandoer til roboten via en blåtann forbindelse.

Kameraet (som er tilkoblet en PC via en universell seriebuss (USB) kabel) tar bilder som brukes til å finne informasjon om robotens posisjon og orientering i planet. Kameraets bildedata bildebehandles i PC for å identifisere nødvendig informasjon til kontroll og *styring* av den mobile roboten. Roboten utfører oppgaver som defineres av PC operator. Roboten er konstruert som en mobil enhet som opererer på egenhånd innenfor et definert område.

For å realisere dette brukes bildebehandling til objektgjenkjenning, og reguleringsteknikk til posisjonering av roboten. Problemstillingen for oppgaven er utforming av gode robuste algoritmer som innen bildebehandlingen klarer å utheve og identifisere nødvendige objekter fra bildet som brukes videre ved posisjonering. Faktorer som kompliserer selve oppgaven er varierende lyssetting for kamera, og feilmarginer i kameramodellen.

• Kapittel 2, Teori: Dette kapittelet beskriver den teorien som er anvendt i rapporten, og danner grunnlaget for de implementerte metoder som brukes for å fremstille resultatene i rapporten. Herunder beskrives både prinsipper for bildebehandling og reguleringsteknikk, men også feilkilder og problemer som må tas høyde for omtales. Delkapitlene om terskling og morfologi er hentet fra forprosjektet [2].

• Kapittel 3, Programmering: Kapittelet gir en kort introduksjon til MATrix LABoratory (MATLAB), et programmeringsspråk for teknisk databehandling [3]. Dette programmet brukes til utvikling og implementasjon av programvaren som implementeres i modellen. For bedre forståelse gis det en liten innføring i programmet med hensyn på bildebehandling. Kommunikasjon med Lego Mindstorms intelligente databrikke som styrer roboten (NTX)diskuteres også.

• Kapittel 4, Implementering av modell: Dette kapittelet forklarer oppbygningen av hele konstruksjonen i detalj med begrunnelser for ulike valg som er tatt. Design av robot og hvordan roboten posisjon og orientering finnes beskrives. Noen eksempler viser trinnvis hvordan de implementerte algoritmene i programvaren fungerer.

• Kapittel 5, Reguleringsteknikk: Kapittelet beskriver systemidentifikasjonen av de prosesser som styres i reguleringssløyfer, og valg regulator.

• Kapittel 6, Resultat: Her presenteres alle resultater som dokumenterer utvikling av, eller test av systemets presisjon og funksjonalitet. Dette innebærer resultater i forbindelse med modellutvikling der mange parametre ble fastsatt, testing av de implementerte algoritmene og det ferdige produktet. Resultatene presenteres i både tabeller og plot for bedre visualisering.

• Kapittel 7, Konklusjon: Kapittelet oppsummerer hva som er blitt utviklet og hvilken grad oppgavens problemstilling er blitt løst. Styrker og svakheter i systemet presiseres og forslag til forbedringer diskuteres.

• Appendiks A, Fra forprosjekt: Her finnes nyttig informasjon om oppsett av systemkommunikasjonen som ble utført i forprosjektet.

• Appendiks B, M-filer: Dette appendikset gjengir alle m-filer som er utviklet, testet, og er implementert i programvaren. Resterende m-filer bruk ved testing og utvikling av systemet er lagt ved rapporten på en digital lagringsenhet (CD).

• Appendiks C, Prosjektbilder: Viser en rekke bilder av den konstruerte roboten fra ulike vinkler.

• Appendiks D, CD materiell: Alt materiell som er brukt til utvikling og testing av programvare er lagt ved som vedlegg på CD.

• Appendiks E, Datablad: Spesifikasjon av kameraet som anvendes i modellen.

Kapittel 2

Teori

Teorikapittelet presenterer de teoretiske prinsipper og metoder som er implementert i programvaren til *styring* av roboten i modellen. Objektidentifisering og posisjonering av roboten utføres med bildebehandling og reguleringsteknikk. Først skal det forklares hvordan disse teknikkene fungerer på et detaljert nivå. Deretter forklares RGB fargemodellens oppbygning, hvordan finne absolutt orientering i et koordinatsystem, hvordan et digitalt bilde dannes, og hvordan en projeksjonsfeil introduseres i bildeplanet ved fotografering av scenen.

2.1 Bildebehandling

Bildebehandling er metoder og teknikker som kan implementeres på et bilde for å utheve spesifikke karakteristikker eller ønskede egenskaper i bildet. Digital bildebehandling tilsvarer bildebehandling på digitale bilder der alle metoder implementeres i algoritmer. Teknikken bak terskling og binær morfologi forklares med enkle illustrasjoner for binære bilder i delkapittelet.

2.1.1 Terskling

Terskling er en teknikk som brukes innen bildesegmentering. Representasjon av et intensitetsbilde [1] som binært bilde[2] er en typisk anvendelse for bildesegmentering. Bakgrunnen for en slik segmentering er å fremheve det som er av interesse i bildet, slik at det blir enklere å analysere bildet videre. Dette kan være objekter i bildet, også kalt forgrunn. Tekstdokumenter som inneholder støy og uønskede

[1] Et intensitetsbilde kan være et såkalt gråskala bildet der hvert piksel i bildet kan ha 8 bits oppløsning for sin intensitet.

[2] Pikslene i bildet består kun av 0 og 1 som representerer henholdsvis hvit og svart

elementer der en ønsker å skille teksten fra bakgrunnen er et typisk eksempel for anvendelsesområde. Målet er å skille objektet best mulig ut ifra bakgrunnen uten å tape ønsket informasjon.[4]

Tersklingsteknikken brukes ved segmentering av gråskala bilde til binært bilde. Ideen bak teknikken er å separere de pikselintensiteter som representerer for- og bakgrunn i bildet hver for seg, ofte også kalt for klassene i bildet. Dette gjøres ved å definere en grenseverdi som bestemmer utfallet av hvert enkelt piksel i det binære bildet. Det er dette som kalles terskling. Det finnes to typer terskling, binær- og multiterskling [3]. Ved binær terskling deles bildet som segmenteres inn i to klasser, dette kan beskrives matematisk som vist i ligning 2.1. Ved multiterskling deles bildet inn i så mange klasser som ønskelig. Da finnes flere tersklingsgrenser for å separere de ulike klassene fra hverandre. Begge disse variantene for terskling kan så deles inn i global og lokal terskling. For binær terskling vil en global metode kun beregne en terskelverdi for hele bildet, mens en lokal tersklingsmetode beregner flere grenser avhengig av posisjonen i bildet.

$$g(x,y) = \begin{cases} 1 & \text{if} \quad f(x,y) \geq T \\ 0 & \quad\quad ellers \end{cases} \tag{2.1}$$

Funksjonene f og g representerer pikselintensiteten for henholdsvis gråskala og det binære bildet, ved pikselindeks x og y. Dersom pikselintensiteten i f(x,y) er over en definert terskelverdi T så blir utfall av piksel $g(x,y) = 1$, og 0 ellers. Dette gjøres for alle pikslene i gråskalabildet. Resultat blir et binært bilde der alle "0"ere og "1" ere representerer henholdsvis bakgrunn og objekt (eller motsatt) [4].
I MATLAB finnes terskelverdien til intensitetsbildet ved funksjonen *graythresh*. Funksjonen anvender Otsu's metode for å finne terskelverdien i bildet. Metoden tester alle mulige terskelverdier for bildet og velger den terskelverdien som minimerer den totale variansen for klassene [5].

2.1.2 Morfologi

Morfologi omhandler form og struktur, og begrepet brukes i mange ulike sammenhenger. Begrepet stammer fra biologien hvor det beskriver studie av form og struktur til dyr og planter. I morfologisk bildebehandling anvendes matematikk basert på set-teori, som et verktøy for å fremheve nyttige bildekomponenter som beskriver områdeform, eksempelvis grenser, skjelett eller konvekse hull. To fundamentale morfologiske set-teori operasjoner som anvendes er *dilasjon* og *erosjon*. Disse danner grunnlaget for de mer komplekse operasjonene *åpning* og *lukking*. I dette delkapittelet forklares de teoretiske prinsippene for disse operasjonene

[3] Multi er Engelsk for mange. Dvs. mange terskelgrenser, videre kun omtalt som multiterskling

i detalj. Morfologisk bildebehandling kan implementeres på både binær- og gråskalabilder, men for enkelthetsskyld anvendes binære bilder i eksemplene. Morfologi er hjørnesteinen blant de verktøy som brukes i ulike bildebehandlingteknikker med det formål å hente ut objekter fra bildet.

En morfologisk set-teori operasjon utføres på et bilde ved hjelp av et eksternt bildeobjekt som er med å forme resultatet. Det eksterne bildeobjektet kalles et strukturelement, der elementets form og størrelse velges ut fra kjennskap til originalbildet og hva som ønskes utført.

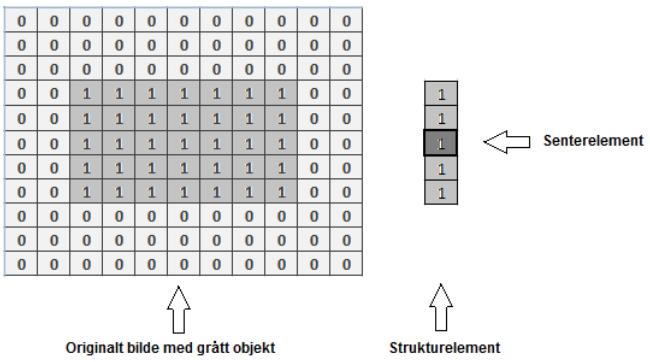
Operasjonene *dilasjon* og *erosjon* resulterer i at bildeobjektets størrelse henholdsvis økes og reduseres. Utforming og størrelse på strukturelementet som velges bestemmer i hvilken grad bildeobjektet vil endres. Eksempelvis, desto større strukturelement gir en desto større utvidelse av bildeobjektet, og vice versa.

Følgende matematisk notasjon brukes for operasjonene

- Dilasjon: $A \oplus B$
- Erosjon: $A \ominus B$

A og B representerer henholdsvis originalbildet og strukturelementet som brukes.

Et lite eksempel illustrerer enkelt bruken av operasjonene. I figur 2.1 vises et binært bilde med et strukturelement som anvendes til operasjonene. Same strukturelement skal brukes for både *dilasjon* og *erosjon*. Ved implementering av operasjonene itereres strukturelementet over hele bildet, og sammenligner i hver posisjon egne bitverdier mot bildets. I hver posisjon er det senterelementets nåværende posisjon i bildet som avgjør bildets resulterende bitverdi i denne posisjonen basert på gitte operasjonskriterier for henholdsvis *dilasjon* og *erosjon*.



Figur 2.1: Matematisk morfologi implementert ved set-teori. Strukturelementet itereres over hele bildet, sammenligner bitverdier og definerer resultatet av valgt operasjon.

Resultatene etter utført *dilasjon* og *erosjon* er vist i figur 2.3, der operasjonene implementeres etter følgende kriterier:

- Ved *dilasjon* defineres senterelementets posisjon som objekt i alle posisjoner hvor minst et av strukturelementets elementer overlapper objektet i bildet.

- Ved *erosjon* defineres strukturelementets senterelement som objekt kun i posisjoner hvor hele strukturelementet overlapper bildeobjektet.

0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0

(a) Resultatet etter morfologisk *dilasjon* av objekt i 2.1.

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

(b) Resultatet etter morfologisk *erosjon* av objekt i 2.1

Figur 2.2: Morfologisk bildebehandling ved *dilasjon* og *erosjon*. Resultatet etter *dilasjon* som vist i figur (a) bekrefter en utvidelse av det originale bildeobjektet, og vica versa resultat for *erosjon* operasjonen som vist i (b).

I praktisk morfologisk bildebehandling brukes *dilasjon* og *erosjon* om hverandre ulikt for å fremstille ønsket resultat. Et bilde kan gjennomgå en ubestemt antall *dilasjoner* og/eller *erosjoner* med samme eller ulike strukturelement før resultatet er tilfredsstillende. Ved kombinasjon av disse grunnleggende operasjonene dannes grunnlaget for morfologisk *åpning* og *lukking*.

Morfologisk *åpning* er først å utføre en *erosjon*, etterfølgt av en *utvidelse* med resultatet fra *erosjonen*. Same strukturelement brukes for begge delprosessene. En morfologisk *åpning* fjerner alle objekter i bildet som er mindre i størrelse enn strukturelementet, glatter objektkonjunkturer og fjerner tynne koblinger mellom to objekter.
Morfologisk *lukking* er utførelse av en *utvidelse* først, der resultatet etterbehandles med en *erosjon* med same strukturelement for begge operasjonene
Som for *åpning* vil *lukking* glatte ut konjunkturene i objekter, men ellers vil operasjonen slå sammen smale sprekker, fylle lange tynne viker og fylle hull i objekter som er mindre en strukturelementet. [4]

Følgende matematisk notasjon brukes for operasjonene

• Morfologisk *åpning*: $A \circ B = (A \ominus B) \oplus B$

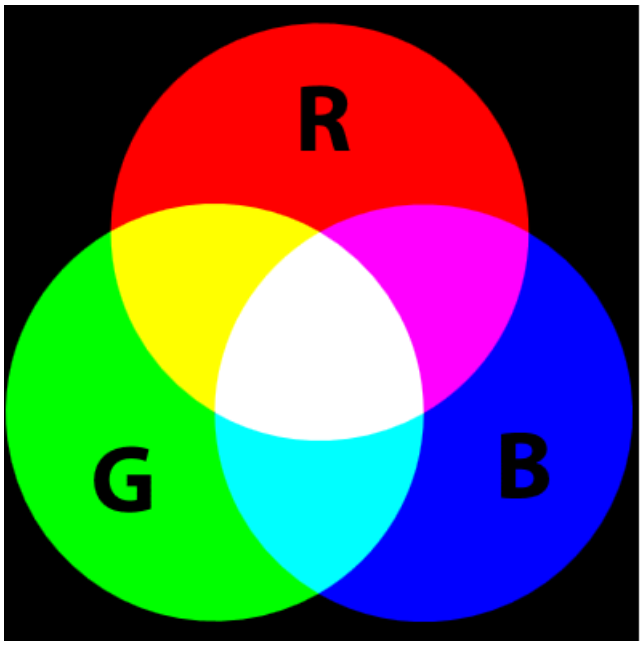
• Morfologisk *lukking*: $A \bullet B = (A \oplus B) \ominus B$

Resultatet etter utført morfologisk *åpning* og *lukking* er illustrert i figur 2.3.

Figur 2.3: Morfologisk lukking og åpning med et kvadratisk strukturelement (4x4 piksel). Originalt binært bilde i figur fra [6]

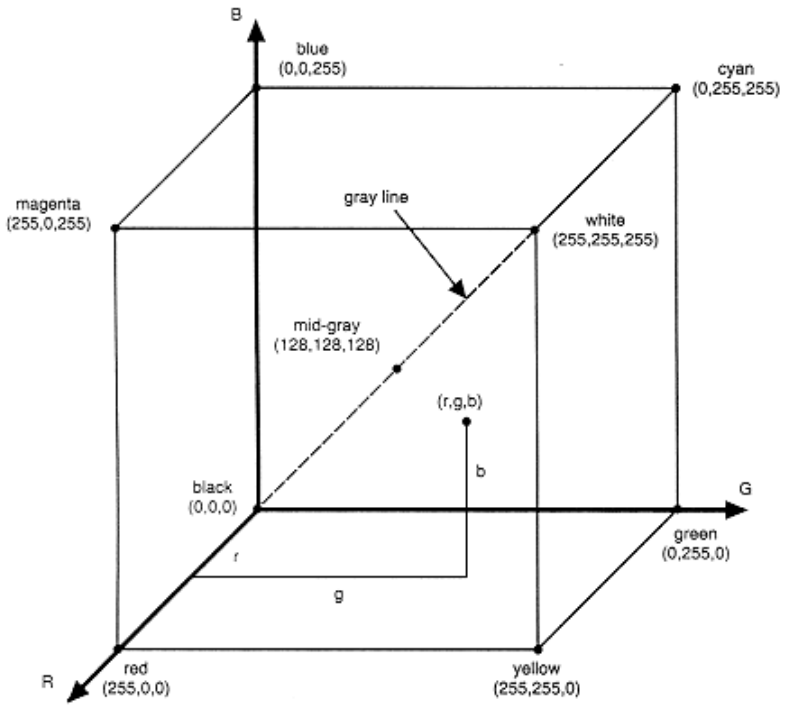
2.2 RGB fargemodellen

Farger implementeres digitalt i et bildepiksel ved anvendelse av en fargemodell. I fargeteorien defineres en fargemodell som en matematisk modell som beskriver hvordan fargene kan representeres. Modellene bruker ulike prinsipper i fremstillingen av de digitale fargene. En av disse modellene bruker primærfargene rød, grønn og blå (RGB) til representasjon av hele fargespekteret. RGB modellen beskriver farget lys som betraktes fra lyskilden, den kalles en additiv fargemodell da den adderer primærfargenes lys, bølgelengde for bølgelengde, og danner nye farger som vist i figur 2.4.



Figur 2.4: RGB modellen. Addisjon av primærfargene gir en ny farge. Hvitt blir en kombinasjon av alle tre primærfargene, mens svart er et resultat av fraværende lys. Figur fra [7].

Hele fargespekteret fra hvitt til svart kan dannes ved variasjon av primærfargenes intensitet. Hvor mange farger som kan dannes digitalt avhenger av primærfargeintensitetens oppløsning. Oftest representeres et fargepiksel med 24 bit, dvs. 8 bit per fargekomponent som igjen betyr at hver fargekomponents intensitet kan representeres med verdier mellom 0-255. Ved kombinasjon av alle primærfargenes pikselintensiteter kan 16 millioner ulike farger produseres. Høye RGB verdier betyr mer lys av hver farge som resulterer i en lys farge, og ergo lavere verdier betyr mindre lys og derfor en mørkere farge. Fargespekteret kan illustreres som vist i figur 2.5. Kubens ytterpunkter er markert med farge og respektive RGB verdier. Den heltrukne linje gjennom kuben fra svart til hvitt representerer gråskalaintensiteter.



Figur 2.5: RGB modellens fargespekter. Figur fra [8]

RGB fargemodellen brukes mye i datagrafikkpakker, programmering og websider [9].

2.3 Maskinsyn

Dette delkapittel er om ikke annet er definert inspirert av boken Computer vision: Principles [10].

Design og bruk av intelligente datamaskinbaserte systemer er idag vanlig i industrien, spesielt innenfor industriell inspeksjon. Datamaskinens evne til å se ved hjelp av kamera baseres på bestemte grunnlegende prinsipp og teknikker. Men utvikling av et generelt flerbrukssystem som kan tolke en stadig varierende scene fremstår som en krevende og utfordrende oppgave.
Design av et praktisk fungerende datamaskinbasert system krever en total og grundig forståelse av alle aspekter som representeres i systemet, helt fra hvordan bildet dannes til fremstilling av det ferdige produktet. Typisk prosedyre for å hente ut nyttig informasjon om bildeobjekter fra sensordata kan være:

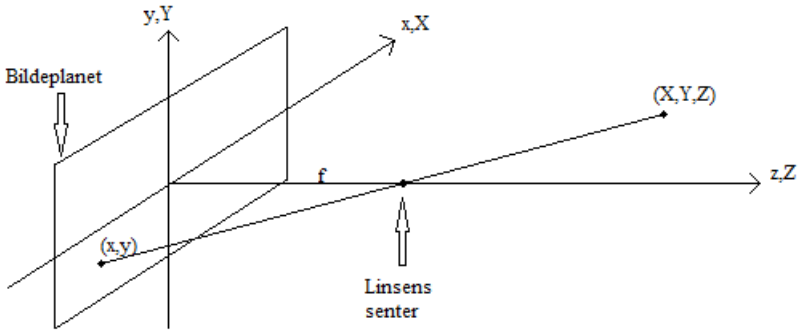
• Innhenting av bilde og forprosessering av bildet i forhold til ønsket interesseområde (ROI).
• Segmentering.
• Videre utvelgelse i forhold til ønskede objektegenskaper.
• Sammenligne egenskaper mot modeller.
• Anvende applikasjoner for gjenkjenning av objekter i scenen.

For hvert av disse punktene innvirker mange faktorer i forhold til valg av algoritmer og teknikker. Den som designer systemet bør ha kunnskap angående problemer og avveininger i forhold til den praktiske løsningen av systemet.

Et bilde dannes ved at en sensor registrerer stråling i form av lysstråler i en todimensjonal (2D) funksjon. Lysstyrke og intensiteter som gjenspeiles kan representere ulike fysiske faktorer, eksempelvis gjengir et kamera lyset som reflekteres fra objekter i scenen, mens et termisk bilde representerer objektenes temperatur i bildet. Ofte tas det flere bilder av scenen med ulike sensorer for en mer robust og pålitelig tolking av scenen. Valg av riktig system ved bildedannelsen spiller en viktig rolle innen design av datamaskinsynsystemer.

Intensistetsbilder dannet fra synlig lys er mest vanlig innen datamaskinsynsystemer. Den største utfordringen ved datamaskinsyn er realisering av en tredimensjonal (3D) tolkning basert på 2D bildet. Grunnlaget for å klare dette er tilfredsstillende kunnskap om kameramodellen som beskriver bildets opphav, der bildegeometri, projeksjonsprosessen og refleksjonsegenskapene til objekter i bildet er viktig for tolkning og analyse.

En slik kameramodell er gjengitt i figur 2.6



Figur 2.6: Sentrert kameramodell som beskriver forhold mellom scene og bilde. Verdenskoordinatsystemet er valgt slik at det samsvarer med bildeplanet, og der Z-aksen passerer gjennom kameralinsen. Linsens avstand til bildeplanet bestemmes av fokallengden f. (X,Y,Z) og (x,y) representerer henholdsvis koordinatene for scenen og bildeplanet. Figur fra [11]

Forholdet mellom et punkt i scenen (X,Y,Z) og bildeplanet (x,y) kan beskrives matematisk som

$$x = \frac{f \cdot X}{f - Z} \tag{2.2}$$

$$y = \frac{f \cdot Y}{f - Z} \tag{2.3}$$

Ligningene 2.2 og 2.3 beskriver projeksjonen for bildesystemet. Når fokallengden f er stor kan perspektiv projeksjon tilnærmes en ortografisk projeksjon der $x = X$ og $y = Y$. Som regel er ikke kameraet plassert i senter av verdenskoordinatsystemet men har ulik grad av frihet i forhold til forskyvning, skalering og rotasjon. I slike tilfeller er det ofte hensiktsmessig å bruke verdenskoordinater for scenen.

Ofte er eksakt lokalitet, orientering og fokallengde til kameraet ukjent. Ved å bruke noen kjente punkter i scene og deres gjenspeiling i bildeplanet kan en beregne parametrene som beskriver transformasjonen mellom kamera og verdenskoordinatsystemet. Forholdet mellom et punkt i verdenskoordinat og samsvarende punkt i bildeplanet for en kameramodell med vilkårlig plassering og orientering er gitt av [12]

$$a_{11}X + a_{12}Y + a_{13}Z - a_{41}xX - a_{42}xY - a_{43}xZ - a_{44}x + a_{14} = 0 \tag{2.4}$$

$$a_{21}X + a_{22}Y + a_{23}Z - a_{41}yX - a_{42}yY - a_{43}yZ - a_{44}y + a_{24} = 0 \tag{2.5}$$

De ukjente koeffisientene fra 2.4 og 2.5 kan beregnes ved kamerakalibrering når både bildeplan og verdenskoordinater er kjent for 6 punkter. Deretter kan et hvilket som helst punkt i scenen beregnes ved bruk av disse koeffisientene.

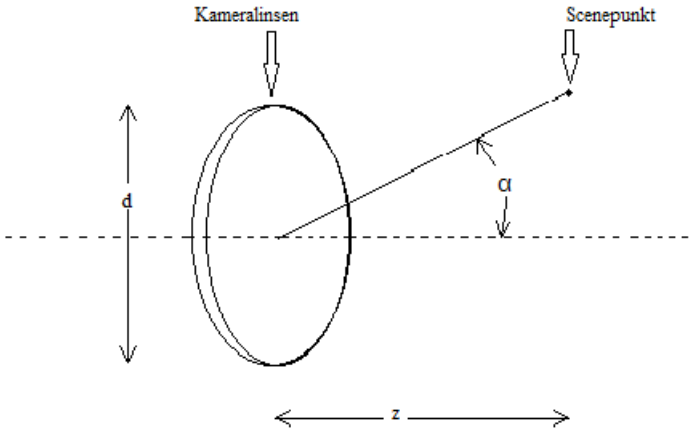
Intensitet og styrke som registreres i hvert enkelt punkt i bildeplanet bestemmes av bildegeometrien, belysning av scenen, refleksjonsegenskaper og overflateorientering til objektene i scenen og kameraets intensitetssensor.

Refleksjonsegenskapene til en objektoverflate beskrives ofte med en toveis fordelingsfunksjon for reflektert lys (BRDF), som gir et mål på stråleglansen i observatørretning fra bildeobjektet skapt av lysstrålingen fra en lyskilde i en gitt retning. Funksjonen beskriver da hvor sterkt en objektoverflate vil fremstå i bildeplanet, når observert fra en gitt retning og belyst fra en annen. Scenens belysning sammen med BDRF bestemmer scenens stråleglans i et gitt punkt. Sammenhengen mellom

bildets elektromagnetiske lysstråling E, scenens stråleglans L, kameralinsens diameteren d, linsens fokallengde f, og vinkelen α er gitt av ligning 2.6 [13]:

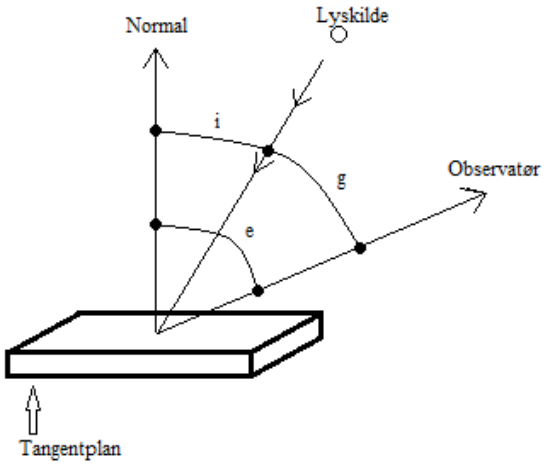
$$E = L\frac{\pi}{4}\left(\frac{d}{f}\right)^2 cos^4\alpha \tag{2.6}$$

Figur 2.7 viser disse faktorene i forhold til en kameralinse. Vinkelen α defineres som vinkelen mellom kameraets optiske akse og linjen mellom linsens senter og et gitt punkt i scenen. z definerer dybdeavstand til punktet.



Figur 2.7: Ståleglans i et gitt punkt. Figuren er inspirert fra boken "Robot Vision"[14].

For å avgjøre hvor mye lysstråling som reflekteres til en observatør fra et bestemt punkt i et bildeobjekt, må en ha god forståelse for den geometriske lyskoblingen mellom lyskilden, objektet og observatør. Objektoverflatens orientering spiller her en stor rolle. Mengde lys som reflekteres fra en objektoverflate avhenger av objektets mikrostruktur og fordelingen av innfallende lys. Ved konstruksjon av et tangentplan i et bestemt punkt i objektoverflaten ser en at innfallende lys kan komme fra alle retninger fordelt i halvkulen overfor tangentplanet. Det enkleste tilfellet blir å se på en enkelt lyskilde der geometrien til det reflekterte lys fra et punkt avhenger av 3 vinkler som vist i figur 2.8.



Figur 2.8: Refleksjon i et punkt i tangentplanet blir en funksjon av innfallende, emittans og fasevinkelen. Figur fra [10]

Innfallende vinkel i defineres som vinkelen mellom lysstråle og den lokale normalen i tangentplanet, e står for emittans og defineres som vinkelen mellom observert reflektert stråle av observatør. Lokal normal og fasevinkelen g defineres av vinkel mellom innfallende og reflektert lysstråle [15]. Refleksjonsfunksjonen gir et mål på hvor mye av det innfallende lyset som reflekteres tilbake i en gitt retning.

Ofte finnes det flere lyskilder som belyser det observerte objektet. Da integreres innfallende- og reflektert lys for alle synlige lyskilder for det observerte punkt. Summen av denne integrasjonen representerer det totale reflekterte lys i retning mot observatør.

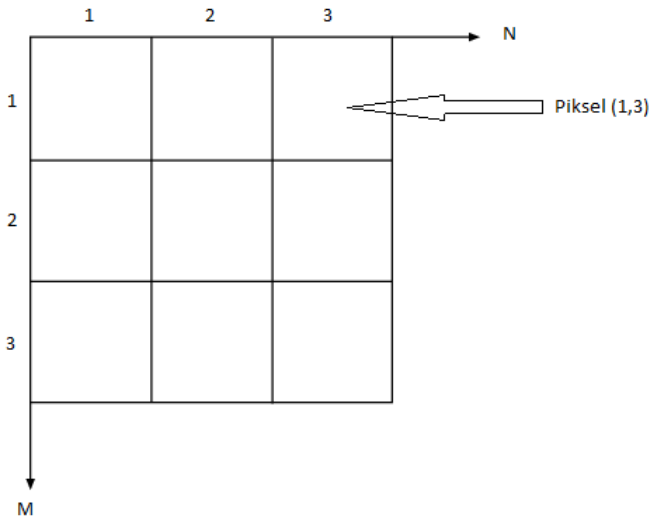
2.3.1 Digital bilderepresentasjon

Et bilde dannes når en scenes lysintensitet registreres ved hjelp av en eller annen form for lysfølsom komponent som lagrer bildeinformasjonen. Scenens lysintensitet er kontinuerlig, og derfor analog av natur, som de fleste signaler i den fysiske verden. Om bildet lagres analogt eller digitalt bestemmes av hvilken lysfølsom komponent som brukes. Analoge motiver som representeres digitalt, sies å ha blitt digitalisert.

Et analogt bilde kan defineres som en 2D funksjon $f(x, y)$, der x og y er plankoordinater, og amplituden til f i et hvilken som helst punktkoordinat (x, y) beskriver punktintensiteten. Bilder som dannes fra en eller ulike nyanser av en farge beskrives ofte som gråskalabilder. Et RGB fargebilde kan beskrives som tre gråskalabilder lagt ovenpå hverandre, der hver bildekomponent er representert

med nyanser av primærfargene rød, grønn og blå. Bildebehandlingsteknikker som er utviklet for gråskalabilder kan derfor anvendes for fargebilder ved å behandle hver fargekomponent individuelt.

Et analogt bilde kan beskrives med bildekoordinater og intensiteten i hvert koordinat. Disse bildeparametrene må digitaliseres for å fremstille motivet digitalt. Digitalisering av koordinater og amplitude kalles henholdsvis for punktprøving eller sampling, og kvantifisering. Bildet beskrives som digitalt når både bildekoordinater og amplitude representeres med endelige diskre enheter (ikke helt sant for kvantifiseringen). Etter digitalisering beskrives bildet i en 2D tallmatrise. Matrisens horisontale- og vertikale dimensjon (M x N), bestemmes av samplingsrate som anvendes på bildet i henholdsvis y og x retning. Koordinatverdiene eller indeksverdiene x og y er nå diskrete heltallsenheter, og beskriver pikselkoordinatsystemet som vist i figur 2.9.



Figur 2.9: Pikselkoordinatsystemet slik det er definert i IPT. Figur fra [4]

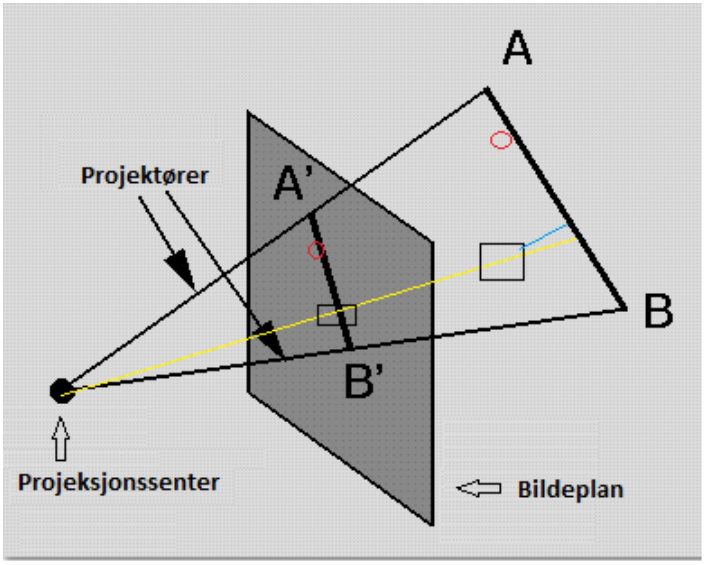
Koordinatsystemet fra 2.9 og definisjonen av intensitetsfunksjonen f, gjør at det digitale bildet kan representeres som en matrise

$$f(x,y) = \begin{pmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & \ddots & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{pmatrix}$$

Høyre side i ligningen ovenfor representerer det digitale bildet per definisjon. Hvert element i matrisen kalles for et bildeelement eller piksel.[4]

2.3.2 Feilmargin ved bildeprojeksjon

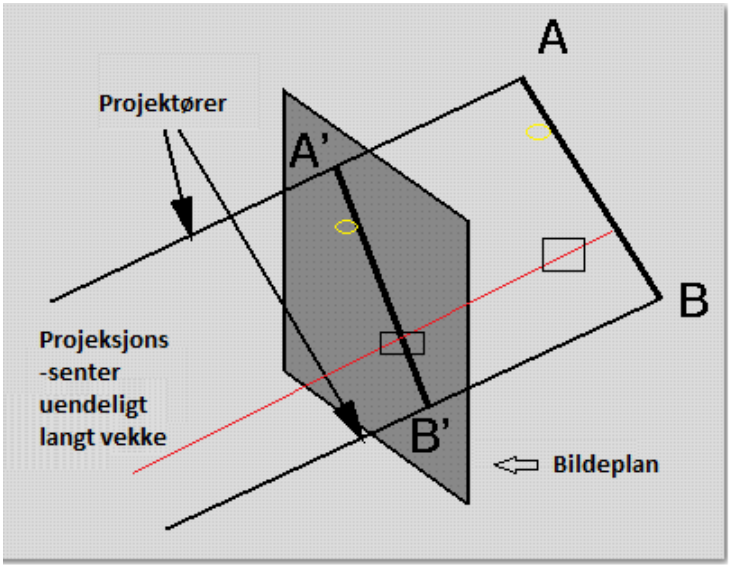
Scenen som fotograferes er vanligvis representert i 3D. Når et bilde dannes blir scenen transformert fra 3D til 2D i bildeplanet, og objektene i scenen gjenspeiles i bildeplanet som en projeksjon av den virkelige 3D scenen. I figur 2.10 brukes et eksempel der kameraet tenkes takmontert med linsen vendt vertikalt nedover. Da representerer linjen A til B en linje i rommets gulv, og linjen A' til B' er gjenspeiling av den same linjen i bildeplanet. Et par figurobjekter er representert i scenen og gjenspeiles i bildeplanet noe nedskalert. Det store firkantobjektet er posisjonert noe over planet som gir opphav til en projeksjonsfeil.



Figur 2.10: Perspektiv projeksjon. Figur fra [16]

Gul linje viser projeksjonslinjen fra kamerasenter til planet. Objektets midtpunkt befinner seg i planet der hvor den blå linjen krysser AB linjen, mens faktum er at objektets posisjon som gjengis i bildeplanet er der hvor den gule linjen krysser AB linjen. Dette resulterer i at objekter gjengis lengre fra hverandre i bildeplanet, riktignok nedskalert etter forholdet mellom bildeplanet og FOV. Feilmarginen blir avstanden mellom blå og gul linje i krysning av AB linjen. Sirkelobjektet i scenen ligger tilnærmet i planet slik at feilmarginen for dette objektet blir minimal.

På grunn av kameraets geometriske oppbygning vil større avstand fra kamera til scenen gi mindre projeksjonsfeil som vist i figur 2.11. Ved uendelig lang avstand fra kamera til planet passerer projeksjonslinjene for alle objektene i kassen vinkelrett gjennom bildeplanet slik at det ikke oppstår noen feilmargin.



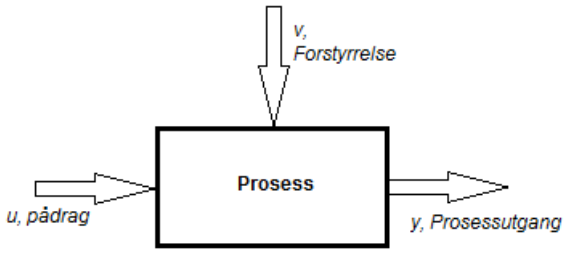
Figur 2.11: Ortogonal projeksjon. Figur fra [16]

Dette kameraplasseringen er ikke implementerbar i virkeligheten av praktiske årsaker. Ved økende avstand fra kamera til scenen vil både en konstruksjons omfang øke i tillegg til at det faktiske interessante objekter i scenen blir mindre i bildeplanet. I systemer hvor det er krav til god presisjon vil korreksjon av introdusert feilmargin være nødvendig.

2.4 Reguleringsteknikk

Presist formulert så innebærer reguleringsteknikk metoder og teknikker for automatisk styring av en fysisk prosess med det formål å bringe en bestemt prosessvariabel nærmest mulig en satt referanseverdi. Reguleringsteknikk kan brukes til ulike formål, som posisjonering av fartøy/kjøretøy, nivåstyring av prosesser/tanker, og har i industrien stor betydning for en rekke forhold som produktkvalitet, driftsøkonomi, sikkerhet, miljøvern, komfort og teknisk gjennomførbarhet.

Et eksempel på en generell prosess som skal reguleres er gjengitt i figur 2.12. Prosessen kan være mekanisk, elektrisk, termisk eller materiell.



Figur 2.12: Blokkdiagrambeskrivelse av en generell prosess. Figur fra [17]

• Prosessen er det fysiske systemet som skal kontrolleres ved reguleringsteknikk.

• Pådraget u, er den variabelen som styrer prosessens inngang for å oppnå ønsket resultat på prosessutgangen.

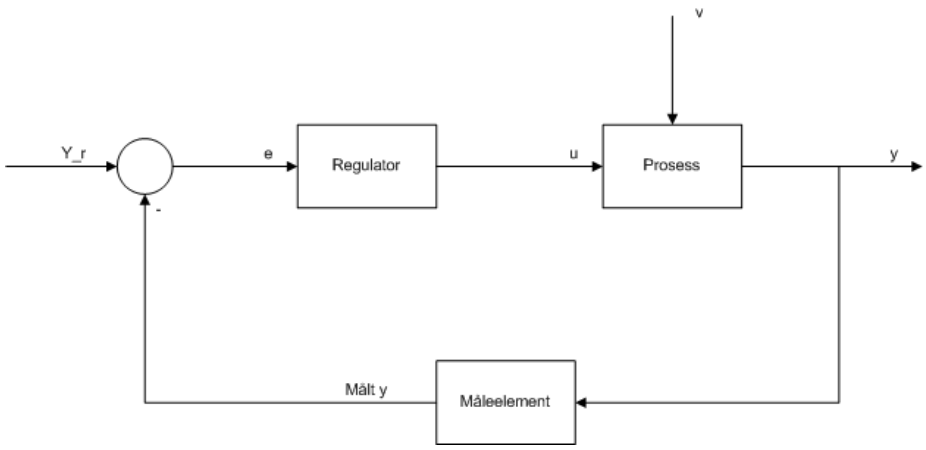
• Prosessutgangen y, representerer den prosessvariabelen som skal reguleres. Kalles ofte for prosessens ER-verdi.

• Forstyrrelsen v, er en ikke kontrollerbar inngangsvariabel som påvirker prosessutgangen.

Den ønskede verdien for prosessutgangen beskrives som prosessens referansepunkt. Punktet betegnes symbolsk ved y_r, og omtales med en rekke navn som settpunkt, SKAL-verdi, referanseverdi eller bare referanse. Avviket mellom SKAL- og ER-verdi kalles for reguleringsavviket og er definert som:

$$e = y_r - y \tag{2.7}$$

Reguleringsproblemet består i å finne det pådraget u, som gir ønsket verdi y_r på prosessutgangen, eventuelt. at reguleringsavviket ligger innenfor en akseptabel grense som defineres av den spesifikke prosessen. Ofte vil avviket være større i en innsvingningsfase (det vil si etter endring i referansepunkt eller en forstyrrelsevariabel) mens kravet til det statiske reguleringsavviket (stasjonære forhold der aller variabler er konstante) er ofte null. Prosessens referansepunkt definerer prosessens arbeidspunkt til enhver tid, og dersom null reguleringsavvik oppnås sies prosessen å være i det ønskede arbeidspunktet. Figur 2.13 viser hvordan reguleringsproblemet kan løses med en tilbakekoblingssløyfe. Prosessutgangen måles og tas i betraktning ved beregning av nytt prosesspådrag.



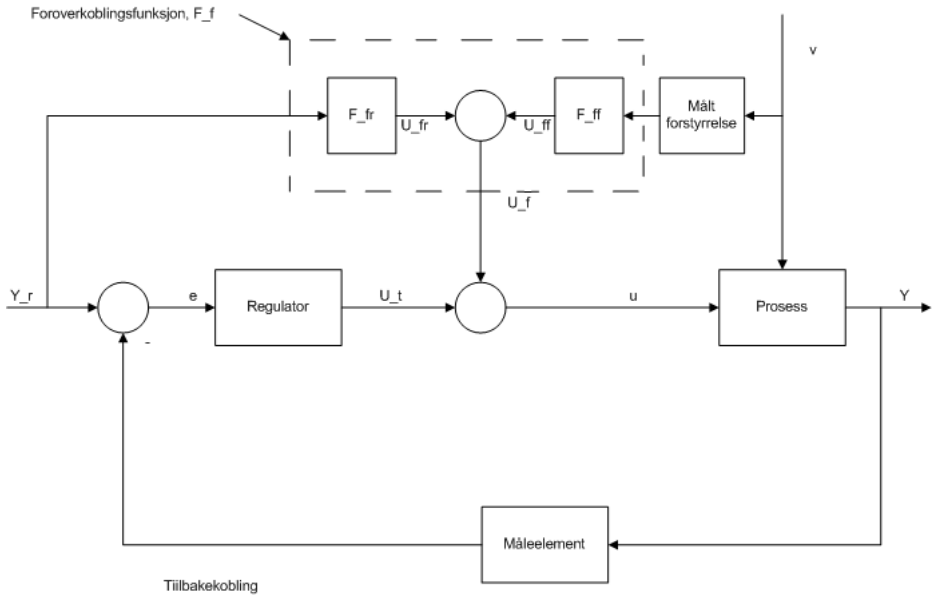
Figur 2.13: Reguleringssløyfe med tilbakekobling av en generell prosess. Figur fra [17]

Ved tilbakekobling beregnes pådraget som en kontinuerlig funksjon av reguleringsavviket. For å finne avviket krever metoden også at prosessutgangen må måles. Beregningen av prosessens pådrag skjer inne i regulatoren, som ofte realiseres i et skreddersydd datamaskinprogram. Regulering med tilbakekobling kalles ofte for avviksstyrt regulering, og sløyfen bestående av prosess, regulator og måleelement kalles ofte for reguleringssløyfen.

Stadige målinger av prosessutgangen og beregning av nytt pådrag gjør metoden godt egnet i forhold til det statiske reguleringsavviket. Når det statiske avviket blir null etter sprang i referansen, har reguleringssystemet perfekte følgeegenskaper, mens null avvik etter sprang i forstyrrelsen gir at systemet har perfekte statiske kompenseringsegenskaper.

En bakdel med metoden er at etter sprang i referansen eller forstyrrelse, skjer det ingen endring i pådraget før reguleringsavviket har fått tid til å bygge seg opp.

Dette kan kompenseres for ved bruk av en foroverkobling. En foroverkobling innebærer en direkte kobling fra både referansen og/eller forstyrrelsen til pådraget. Denne pådragsendringen er ikke avviksbasert men beregnes fra matematisk modell som beskrives det fysiske systemet. En perfekt foroverkobling vil gi null reguleringsavvik uansett hvilken type endring som oppstår i referanse eller forstyrrelse. I praksis får en ikke en perfekt foroverkobling da den matematiske modellen som beskriver systemet ofte inneholder antagelser og forenklinger som resulterer i en modellfeil. Derfor kombineres ofte foroverkobling med tilbakekobling som vist i figur 2.14.



Figur 2.14: Reguleringssystem med både foroverkobling og tilbakekobling. Figur fra [17]

Kombinasjon av disse teknikkene er optimalt i forhold til minimalt reguleringsavvik. [17]

2.4.1 Regulatorfunksjoner

Pådraget til prosessen i et reguleringssystem med tilbakekobling beregnes av regulatoren med bakgrunn i reguleringsavviket. Det kan nevnes noen ulike regulatortyper som alle har spesifikke egenskaper.

- Av/på-regulator
- Proporsjonal(P)-regulator
- Proporsjonal Integrasjon (PI)-regulator
- Proporsjonal Integrasjon Derivasjon (PID)-regulator

Av/på regulatoren er den enklest implementerbare, men den gir også den dårligste reguleringen da stående svingninger oppstår i reguleringssløyfen på prosessutgangen. Termostaten i en varmeovn er et typisk eksempel på en av/på- regulering. P- og PI-regulatoren er forenklede utgaver av PID-regulatoren, der PI- og PID-regulatoren er mest brukt i industrien. Valg av en tilstrekkelig god regulator avhenger av kriteriene som stilles til prosessen.

Det stilles ofte krav til det statiske reguleringsavviket ved sprang i referansen

eller forstyrrelsen. For å oppnå null statisk reguleringsavvik ved sprang i referansen, trengs en integrator i reguleringssløyfen. Noen prosesser inneholder en naturlig integrator, som innebærer at en P-regulator vil være tilstrekkelig for å oppnå null reguleringsavvvik ved et sprang i referansen. Men et sprang i forstyrrelsen kan i slike tilfeller gi et statisk reguleringsavvik. Bruk av PI- eller PID-regulator som har en integrator i regulatoren vil gi null statisk reguleringsavvik for sprang i både referanse og forstyrrelse. Dette gjelder for konstante referanser, dersom det er ønskelig å følge en mer avansert referanse (som varierer over tid, for eksempel en rampe eller sagtann) må en foroverkobling inkluderes i reguleringssløyfen.

Regulatoren beregner pådraget u som

$$u = u_o + u_e$$

der u_o er det nominelle pådraget, og u_e er pådragsbidraget beregnet fra reguleringsavviket i sløyfen. Nominelt pådrag beskriver det pådraget som skal til for å holde en prosess i eller nær arbeidspunktet når regulatoren står i manuell modus, og brukes ofte som utgangspunkt for pådraget ved overgang til automatisk modus for regulatoren. Det avviksbaserte leddet u_e kompenserer for endringer i referansen eller noen av prosessens forstyrrelser.

Beregningen av det avviksbaserte leddet u_e gjøres på forskjellige måter for de ulike regulatorfunksjonene som beskrevet i tabell 2.1 fra [17].

Tabell 2.1: Beregning av det avvikbaserte pådragsleddet u_e, for ulike regulatorfunksjoner

Reg. type	Symb.ledd	u_e
Av/på-reg.	u_e	A for $e >= 0$
		-A for $e < 0$
P-reg	u_p	$K_p \cdot e$
PI-reg.	$u_p + u_i$	$K_p \cdot e + \frac{K_p}{Ti} \int_0^t e d\tau$
PID-reg.	$u_p + u_i + u_d$	$K_p \cdot e + \frac{K_p}{Ti} \int_0^t e d\tau + K_p \cdot T_d \frac{de}{dt}$

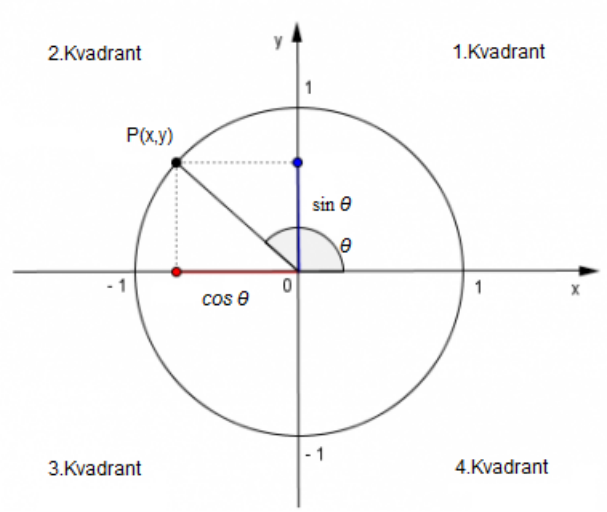
Av/på regulatoren kan brukes dersom kravet til en presis regulering ikke avgjørende for prosessen, og avviksbidragets størrelse avhenger av amplituden A. Et lite eksempel forklarer hvorfor stående svingninger oppstår på utgangen. Ved temperaturregulering vil et positivt reguleringsavvik gi nytt pådrag $u = u_o + A$, utgangen vil starte å øke helt til utgangens verdi blir større enn referansen. Da vil pådraget bli $u = u_o - A$ som fører til at utgangen synker igjen helt til den igjen passerer referanseverdien og same prosessen starter over igjen.

P-regulatoren er en proporsjonalregulator der tilbakekoblingsleddet u_p bestemmes av den konstante proporsjonalforsterkningen K_p. Med unntak av prosesser med integrator klarer ikke P-regulatoren å bringe det statiske avviket til null fordi u_p blir mindre når e blir mindre. Økning av K_p vil kunne redusere avviket, men det går på bekostning av reguleringsystemets stabilitet. Dersom systemet kan bli ustabilt vil en for stor K_p verdi gjøre reguleringssystemet ustabilt.

PI-regulatoren består av både proporsjonal- og integralledd. Integralleddets pådrag u_i beregnes som tidsintegralet av reguleringsavviket fra $t = 0$ og fram til nåtid for regulatoren, og resultatet blir at det statiske avviket blir null over tid. Integrasjonshastigheten bestemmes av integraltiden T_i. [17]

2.5 Absolutt orientering i et koordinatsystem

Retningen til en vektor i et koordinatsystem kan finnes med kjennskap til 2 koordinatpunkter på vektoren. For et koordinatsystem i planet som vist i figur 2.15, vil absolutt orientering for en vektor defineres i et vinkelområde mellom 0-360°. Beregning av vektorens vinkel kan gjøres med trigonometriske funksjoner.



Figur 2.15: Kartesisk koordinatsystem med enhetssirkelen. Figur fra [18]

Den trigonometriske tangensfunksjonen defineres med punktet P som

$$\tan\theta = \frac{\sin\theta = x}{\cos\theta = y} \tag{2.8}$$

Funksjonene kan beskrives periodisk som vist i tabell 2.2 fra [18].

Tabell 2.2: Oversikt definisjonsområde, verdi og inversfunksjon for de trigonometriske funksjonene $\sin, \cos og \tan$

f	Def.omr$_f$	Verdi$_f$	f^{-1}
$\sin x$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	[-1,1]	$\sin^{-1} x = \arcsin x$
$\cos x$	$[0, \pi]$	[-1,1]	$\cos^{-1} x = \arccos x$
$\tan x$	$(-\frac{\pi}{2}, \frac{\pi}{2})$	Reelle tall	$\tan^{-1} x = \arctan x$

Vektorens vinkel i koordinatsystemet finnes fra ligning 2.8 men tangensfunksjonen returnerer en vinkel i definisjonsområde som ikke nødvendigvis gjenspeiler faktisk vektorretning. Eventuell misvisningen av vektorens vinkel kan korrigeres for med kjennskap til hvilken kvadrant de ulike vektorpunktene ligger i. Et lite eksempel illustrerer denne misvisningen enkelt. Avlesning av θ i figur 2.15 gir $\theta = 135°$, mens beregnet vinkel fra formel gir:

$$\theta = \arcsin\left(\frac{\sin(135)}{\cos(135)}\right) = -45°$$

Altså den beregnede vinkelen fra formel stemmer ikke overens med den avleste. Et aksesystem deles inn i 4 ulike kvadranter som vist i figur 2.15, der de ulike kvadrantene representerer ulike vinkelområder. Det som skiller kvadrantene fra hverandre er ulike fortegn på x- og y komponenten som beskriver et punkt i planet. 1.kvadrant representer området i koordinatsystemet der θ ligger mellom 0-90°, videre definerer 2., 3., og 4. kvadrant henholdsvis vinkelområdene 90-180°, 180-270° og 270-360°. Dersom det er ønskelig å identifisere en retning for linjen mellom to punkter i planet i området 0-360° må kunnskap om hvilken kvadrant vinkelen befinner seg i utnyttes. Vinkler i 1. kvadrant finnes direkte, dersom θ ligger i i 2. eller 3. kvadrant adderes vinkelen men faktor π, og for 4. kvadrant adderes vinkelen med en faktor på 2π for riktig gjengivelse av absolutt retning i koordinatsystemet.

I prinsippet finnes her en vinkelretningen for linjen mellom to punkter i aksesystemet, nemlig punktet P og origo O. Prinsippet fungerer generelt og kan brukes til å finne vinkelen i planet mellom 2 vilkårlige punkter i et koordinatsystem. [18]

Kapittel 3

Programmering

All programvare som er implementert for modellen er utviklet og testet i matematikkprogrammet MATLAB. Dette kapittelet gir en kort introduksjon til hvordan MATLAB anvendes til digital bildebehandling, litt om ulike datatyper som finnes og hvordan fargebilder av type RGB fremstilles. Selve programvarekoden som er implementert er lagt ved i appendiks B.

3.1 Bildebehandling i MATLAB

MATLAB er godt egnet til anvendelse innen digital bildebehandling. Programmet gir tilgang til et omfattende utvalg av funksjoner for behandling av multidimensjonale tabeller. I den sammenheng representeres et bilde i MATLAB som en 2D tabell. Funksjonene er samlet i IPT, og som kombinert med MATLAB sin numeriske beregningskapasitet er med å øker MATLABs evne til løsning av digitale bildeproblemer. Selve utførelsen av bildebehandlingsprosessene implementeres med enkle kompakte kommandoer, og gjør MATLAB til både en ideell arbeidsplattform og et ideelt redskap ved løsning av bildebehandlingsproblemer [4]. Programmet egner seg spesielt godt som et prototype verktøy, forskningsverktøy og til undervisningsformål. Dersom et produkt skulle utvikles ville en annen plattform med større beregningshastighet vært å foretrekke.

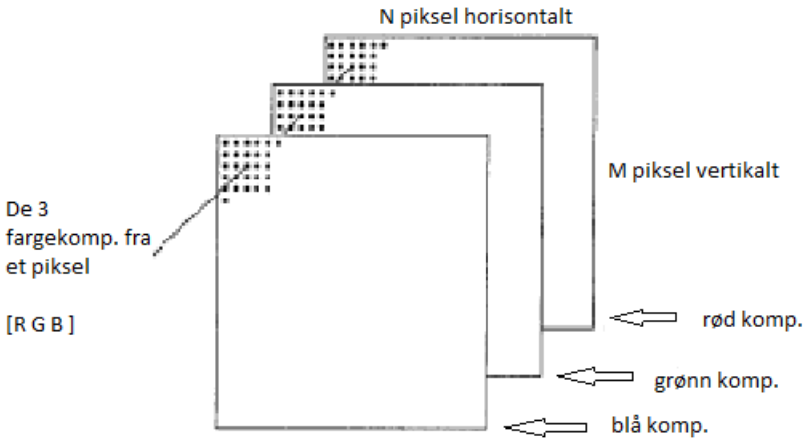
3.2 Datatyper

Pikselindeksene for et digitalt bilde i MATLAB representeres ved heltall og lagres som datatypen *integer*. Pikselintensiteten til hvert piksel kan representeres med ulike oppløsning, og lagres som relevant datatype i forhold til oppløsningen. Hvor mye minne som tas opp for å lagre pikselets intensitetsdata avhenger av

datatypen. De mest vanlige datatypene er *uint*8 og *logical* som henholdsvis krever 1 byte og 1 bit minne per pikselelement. Numeriske beregninger i MATLAB gjøres med datatype *double* for stor flyttallspresisjon, der hvert element representeres med 8 bytes. IPT støtter følgende fire bildetyper: Intensitet, binære, indeks og RGB bilder. Et intensitetsbilde skalerer datamatrisens verdier for å representere en intensitet. Eksempelvis vil elementer av datatype *uint*8 ha verdier mellom 0-255, mens *double* klassen representerer verdiene som flyttal mellom 0-1. Binære bilder består av en bildematrise med utelukkende 0 og 1 som verdier. [4]

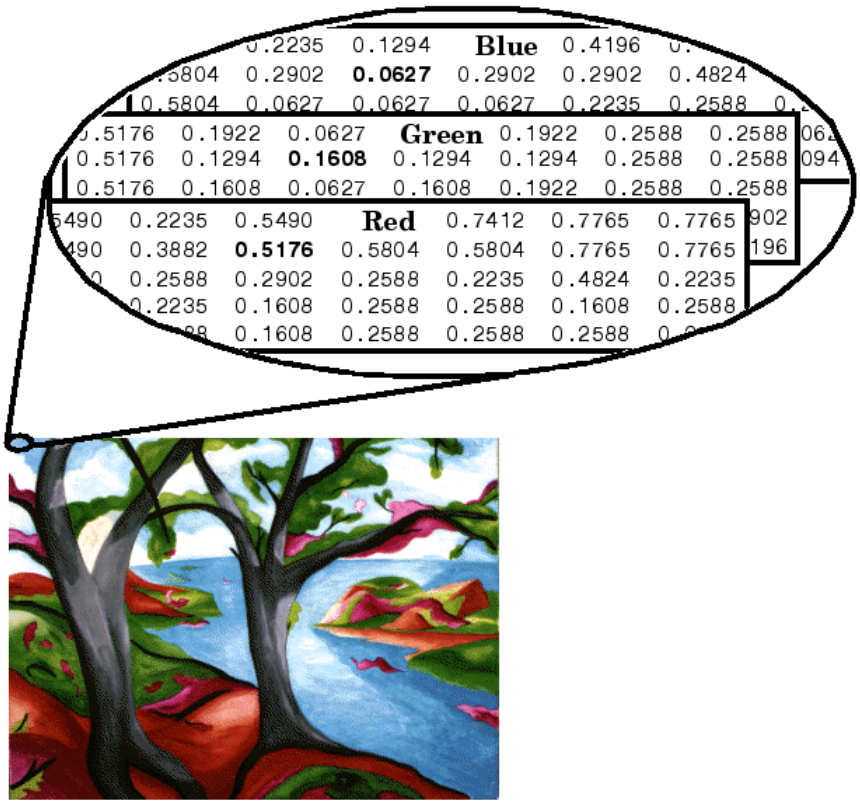
3.3 RGB fargemodellen

I MATLAB er et RGB bilde bygget opp av 3 lag, hvert bestående av komponentfargene rød, grønn og blå. Bildet representeres som en 3D tabell med dimensjonene $MxNx3$ som vist i figur 3.1.



Figur 3.1: Representasjon av et fargebilde i MATLAB. M og N utgjør bildets dimensjon i henholdsvis vertikal og horisontal retning, der tallet 3 beskriver de tre fargekomponentlagene. Figuren fra [19].

Resultatet blir et fargebilde når fargekomponentene gis som input til for eksempel en fargeskjerm. De tre bildene som danner RGB fargebildet kalles for henholdsvis det røde, grønne og blå komponentbildet. Pikselverdienes definisjonsområde i komponentbildene bestemmes av hvilken dataklasse pikseldataene lagres i. For et RGB bildet av klasse *double*, vil pikselverdiene skaleres og ligge mellom $0-1$ som vist i figur 3.2.



Figur 3.2: RGB bilde av klasse double. Figur fra [19].

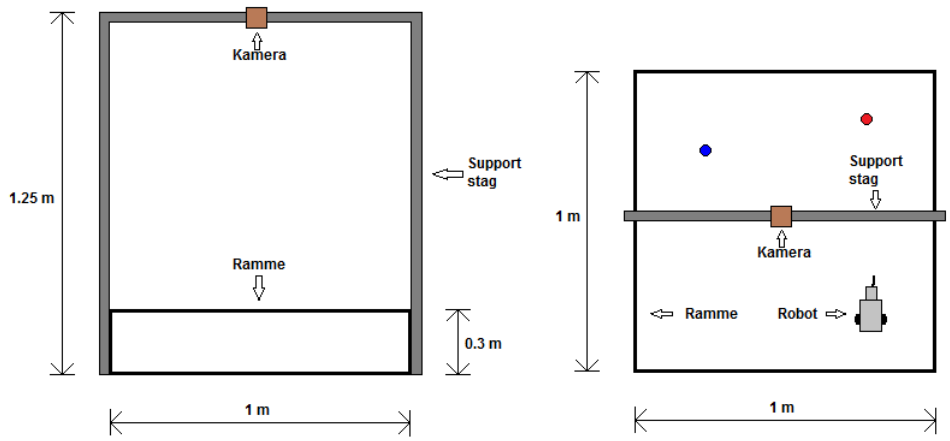
Kapittel 4

Implementering av modell

Modellen som den utviklede programvaren skal implementeres i består av en robot, to baller, et webkamera, og den fysiske konstruksjonen. Selve ideen til praktisk løsning av modell ble utformet og presentert i forprosjektet til masteroppgaven [2]. Dette kapittelet forklarer og begrunner alle valg og metoder for konstruksjonen, sammenstilling av praktisk løsning, design av robot, og hvordan robotens orientering og posisjon i planet defineres. Deretter beskrives de 2 hovedalgoritmene som brukes i programvaren til henholdsvis beregning av objektkoordinater og posisjonering av roboten i planet. Til slutt beregnes teoretisk misvisning av beregnede koordinater funnet fra bildeplanet.

4.1 Sammenstilling av modell

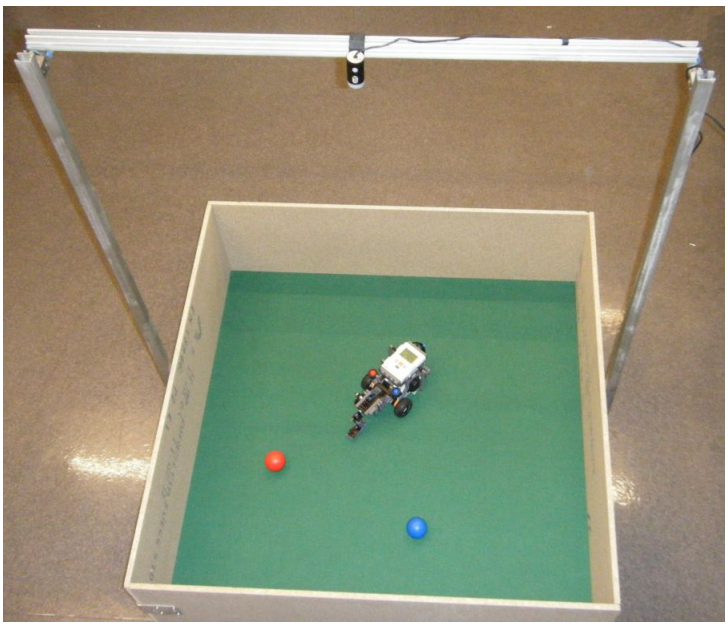
Design av en modellkonstruksjon ble utført for å imøtekomme kriteriene for oppgaven, dvs i forhold til implementasjon av fagmomenter fra masterstudiene. Roboten må ha stor nok plass til å kunne manøvrere fritt uten for mange begrensninger, samtidig som at robotens arbeidsområdet må begrenses til kameraets synsfelt. Av praktiske årsaker i forhold til modellens størrelse og kameraets synsfelt (FOV) ble robotens arbeidsområde begrenset til 1 m^2. Et større arbeidsområde resulterer i større kameraavstand til planet for full dekning av arbeidsområdet. I tillegg gir større kameraavstand større utfordringer i forhold til objektdeteksjon. Figur 4.1 viser modellkonstruksjonen med fysiske mål.



(a) Modellen sett fra siden (b) Modellen sett ovenfra

Figur 4.1: Figur (a) og (b) illustrerer modellens fysiske konstruksjon fra henholdsvis siden og ovenfra.

Fysiske avgrensingen av robotens arbeidsområde ble realisert med en ramme skrudd sammen av fire sponplater. Kamerahøyden ble fastsatt etter kriterier i forhold til kameraets FOV, der dekning av robotens arbeidsområdet er et minimumskrav. Etter testing ble kamerahøyden fastsatt til 1.25 m, og kameraet ble festet i modellen aluminiumsskinner. En finerplate ble trukket med en grønn filt som vist i figur 4.2 og brukes som underlag i kassen. Underlaget er jevnt og av et materiell som sørger for både passelig rullemotstand for ballene og tilstrekkelig friksjon mellom robotens hjul og underlaget.



Figur 4.2: Den praktiske løsningen for modellen.

4.2 Design av robot

Styring av roboten med god presisjon stiller krav til informasjon om posisjon og retning arbeidsområdet til enhver tid. Bildedata innhentes fra kameraet og bildebehandles for å hente ut nødvendig informasjon. I denne prosessen er det en fordel å identifisere spesielle egenskaper til de objekter som en søker informasjon fra. En metode som kan anvendes til enklere objektgjenkjenning av roboten er introduksjon av markører. En markør har en eller flere typiske egenskaper som skiller seg ut i forhold til resten av bildet, og som forenkler bildebehandlingsprosessen. Markørens egenskaper kan typisk være geometrisk form, farge eller mønster, men uansett hvilken egenskap som utnyttes, så brukes markøren som et referansepunkt i forhold til objektets posisjon.

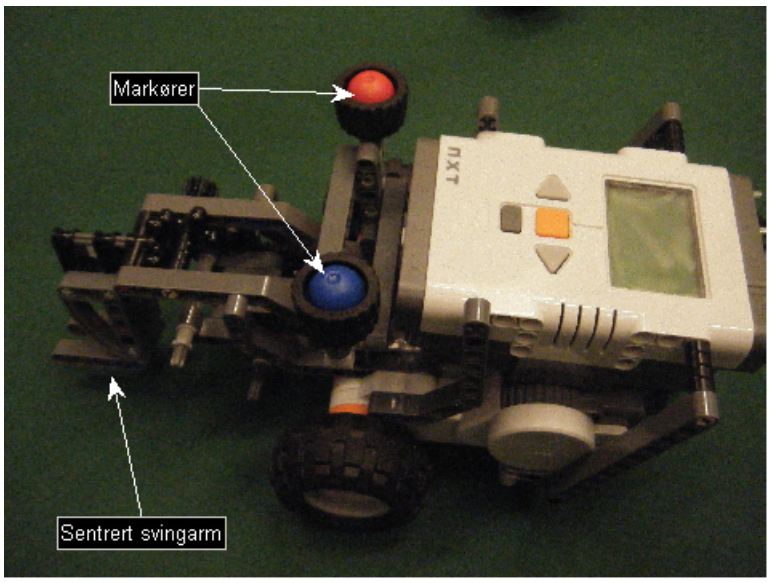
En objektposisjon kan identifisering fra en markør, men dersom dette objektets orientering (retning i planet) er av interesse vil bruk av 2 markører for objektet være fordelaktig [20]. De to markørenes posisjon identifiseres, og objektets orientering kan beregnes med kjennskap til markørenes plassering på roboten. Det er da viktig at de to ulike markørene kan skilles fra hverandre og identifiseres hver for seg under bildebehandlingen, selv under varierende lysforhold. Derfor vil god kontrast mellom markørene være en fordel. Primærfargene rød og blå valgt som markørfarger da disse representerer to ytterpunkter for sin respektive fargekomponent. Markørene skilles ut av bildet ved bildebehandlingsteknikker og objektkoordinatene finnes.

Bildebehandlingen som identifiserer objektkoordinatene fungerer teoretisk godt, men i praksis oppstår det ofte problemer med gjenskinn i kameraobjektene, som skyldes nærliggende lyskilder i scenen. Gjenskinn representerer en lys farge som i RGB fargemodellen gir høy intensitet for alle komponentene. Problem er at både lysere farger og gjenskinn vil gi flere objekter etter terskling som må adresseres med videre bildebehandling.

Algoritmer som kompenserer for gjenskinn og andre kompliserende faktorer er ofte dyre å utvikle, derfor er det mer vanlig i industrien å tilpasse forholdene rundt kameraet for et optimalt resultat. Enkle modifiseringer i virkeligheten kan forenkle bildebehandlingsalgoritmen mye, og det vil være fordelaktig i forhold til kostnader og databeregningstid. For å begrense problemet med gjenskinn ble modellens farger valgt bevisst for å gjøre bildebehandlingsalgoritmen mest mulig robust. Ved konstruksjon av roboten ble mørke legoklosser anvendt i størst mulig grad, videre ble både baller og robotens markører valgt i fargene rød og blå for å skille de fra hverandre. I tillegg skaper den grønne filten god kontrast mellom underlaget og objektene i modellen.

Gjenskinn i markørene kan gi problemer i forhold til riktig detektering. Problemet er at dersom en får mye gjenskinn i både rød og blå markør vil en få problemer med å skille de fra hverandre siden alle RGB verdiene vil være høye. Derfor er det ønskelig å finne en markør som er tilstrekkelig robust i forhold til gjenskinn for å minimere sjansen for en feildeteksjon av et av modellobjektene.

Noen ulike markørformer ble testet for å finne en optimale markører til roboten. Markørens fysiske utforming påvirker hvor mye gjenskinn som produseres. Dette er naturlig da en ulik utforming vil føre til ulik refleksjon av omliggende lyskilder. Lys som treffer runde objekter reflekteres i ulik grad på grunn av overflatens krumning, men for et flat objekt vil refleksjonen bli nokså konstant over hele flaten til markørobjektet. Roboten som ble designet og konstruert for bruk i modellen er vist i figur 4.3. I appendiks C vises flere bilde av roboten fra flere vinkler.

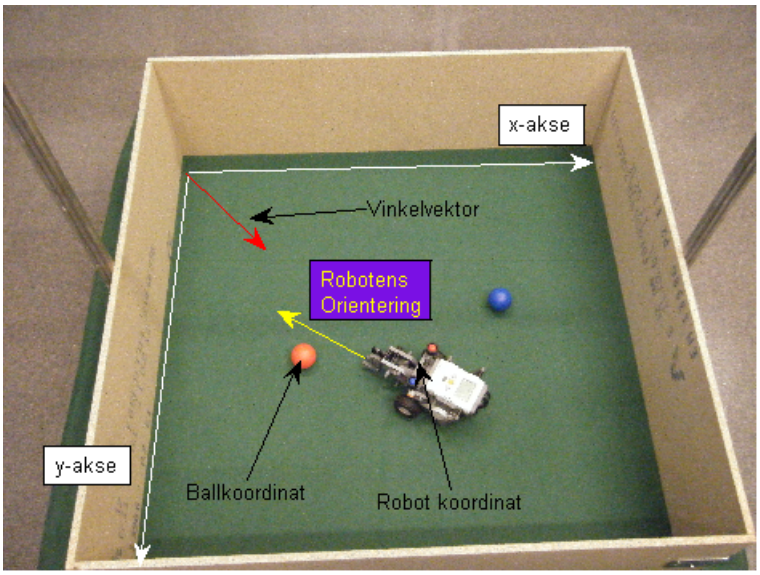


Figur 4.3: Modellens konstruerte robot.

Valg av markør ble gjort etter testing som beskrevet i kapittel 6.1.1. Svarte dekk ble montert rundt markørene for å skape større kontrast mot bakgrunnen for bedre bildesegmentering. Markørenes plassering på robotens ble valgt hensiktsmessig rett ovenfor hjulakslingen som roboten roterer rundt ved manøvrering. Dette forenkler de implementerte posisjoneringsalgoritmene og eliminerer en potensiell feilkilde. Robotens svingarm ble designet med tanke på lavt tyngdepunkt og praktisk tilnærming til ballen ved posisjonering. Resultatet ble en arm montert lavt og sentrert rett foran roboten i dens kjøreretning. Svingarmens rotasjon går perpendikulært på robotens kjøreretning, og må tas i betraktning i posisjoneringsalgoritmen.

4.3 Robotens posisjon og retning

Robotens posisjoneres ved aktiv bruk av til enhver tid beregnet koordinat og orientering. De beregnede dataene trenger et referanseppunkt. Derfor ble et koordinatsystem som vist i figur 4.4 definert i modellen. Det er koordinatene fra bildeplanet som anvendes ved beregning av parametre. Koordinatsystemet for modellen er definert likt som for koordinatene i bildeplanet brukt ved bildebehandlingen. Posisjon og retning for roboten defineres i forhold til dette koordinatsystemet, der posisjon og retning beskrives henholdsvis som koordinat og en retning i forhold til x-aksen. Koordinatsystemets x-akse defineres som retning 0°, der vinkelvektorens rotasjon med klokken samsvarer med positivt økende vinkel i koordinatsystemet. En retning med avvik fra x-asken defineres som en vinkelvektor med stigende verdi når den roteres med klokken.



Figur 4.4: Modellens aksesystem, defineres likt som pikselkoordinatsystemet i bildeplanet. Her har den røde vinkelvektoren orientering lik 45 ° mens robotens retning (gul pil) tilsvarer ca. 210 ° i koordinatsystemet.

Robotens posisjon finnes ved bildebehandling av bildedata fra kamera. Mer detaljert så finnes koordinatene til robotens markører (som vist i figur 4.3). Når markørobjektenes koordinater er kjent, og med kjennskap til deres plassering på roboten beregnes robotens x- og y koordinater som gjennomsnittet av markørkoordinatene [20]:

$$x = \left(\frac{x_r + x_b}{2} \right) \tag{4.1}$$

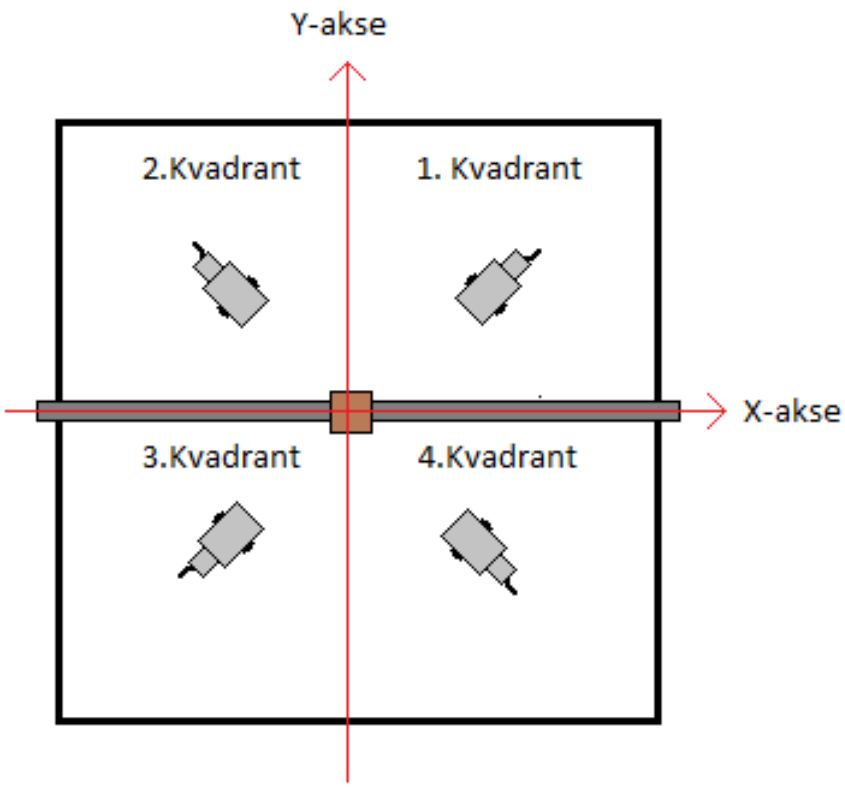
$$y = \left(\frac{y_r + y_b}{2} \right) \tag{4.2}$$

Beregnede koordinatpunkt blir som indikert i figur 4.4 midt i mellom markørene, og defineres som robotens koordinat. Robotens orientering i koordinatsystemet defineres som en vinkel mellom 0-360°, tilsvarende en full omdreining i planet. Når vinkelvektoren passerer 360° så starter den på 0° igjen. Markørkoordinatene brukes også til å beregne robotens orientering σ etter formel [20]:

$$\sigma = \arctan \left(\frac{y_r - y_b}{x_r - x_b} \right) \tag{4.3}$$

Vinkelen som finnes fra ligning 4.3 vil alltid ligge mellom $\pm$ 90°, og trenger ikke gjenspeile robotens absolutte orientering i modellen. I tillegg så beskriver vinkelen strengt tatt retningen for linjen mellom de to markørkoordinatene. Robotens retning ligger perpendikulært på denne linjen. Dette korrigeres for i den implementerte algoritmen. Kompensasjon for absolutt orientering kan illustreres med et eksempel.

Robotens retning i planet kan defineres for fire ulike scenarioer, en for hver kvadrant i koordinatsystemet som vist i figur 4.5.



Figur 4.5: Kvadrantene som definerer ulike retningsområder for roboten.

Koordinatsystemets er tegnet inn for å illustrere de ulike kvadrantene, og må ikke forveksles med det definerte koordinatsystemet i modellen. En robot med rød og blå markør er tegnet i hver sin kvadrant, der pilene indikerer kjøreretning. 1.kvadrant tilsvarer retning mellom 0-90°, 4.kvadrant -(0-90°), 3.kvadrant -(90-180)°og 2.kvadrant 90-180°. Det vil si dersom robotens faktiske retning befinner seg i 2. eller 3. kvadrant så vil beregnet vinkel fra ligning 4.3 gjengis som i henholdsvis. 4. og 1. kvadrant. Ved å overvåke fortegnet på teller og nevner i ligning 4.3 finnes den absolutte retningen og nødvendige kompensasjoner gjøres.

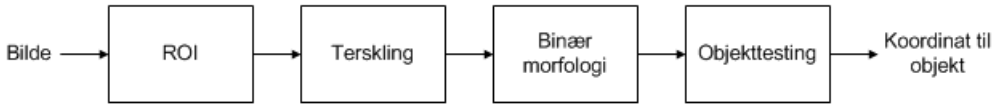
4.4 Implementerte algoritmer

Dette delkapittelet beskriver de algoritmer som er utviklet og implementert som programvare i modellen. Algoritmens forskjellige steg illustreres med først med et blokkskjema, og forklares deretter mer detaljert med tekst. Programkoden som implementerer pseudokodene i kapittelet er lagt ved i appendiks B.

4.4.1 Objektdeteksjon

Algoritmen som beskrives er implementert både for å finne ballenes posisjoner, og robotens posisjon og orientering i koordinatsystemet. I begge tilfeller brukes same algoritmen men med ulike parameterverdier funnet for å utheve det ønskede objektet. Algoritmen er tilpasset eget formål men er inspirert av artikkelen "A very low cost distrubuted localization and navigation system for a mobile robot"[20].

De forskjellige trinnene i objektdeteksjonsalgoritmen kan illustreres blokkskjematisk som vist i figur 4.6.



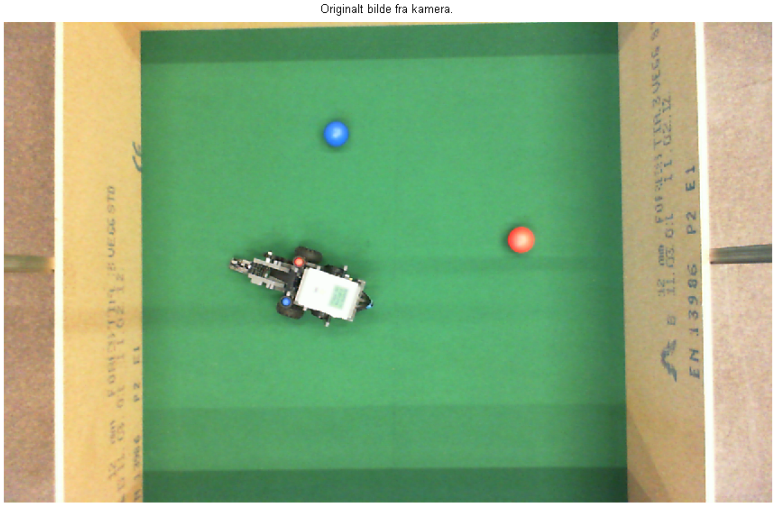
Figur 4.6: Skjematisk oppbygning av koordinatalgoritmen.

1. Innhenting av bilde: Bildet som skal analyseres innhentes av modellens webkamera og returneres til workspace i MATLAB. Kameraet returnerer bildestørrelse på 1280x800 piksler, i et fargeformat som baseres på lysintensitet, og fargens tone og metning (YUY2). Konvergerer fargeformatet til RGB, for enklere segmentering av rød og blå markør.
2. Klargjør bildet før terskling: Bildets ROI defineres, og fargekomponentbildene fra rød- og blå komponent trekkes utfra fargebildet. De neste trinnene i algoritmen gjøres hver for seg for de to segmenterte fargekomponentbildene.
3. Terksling: Fargekomponentbildene konverteres til binære bilder ved tersklingsteknikk. Det brukes en høy terskelverdi ved terskling for å skille ut de piksler med høy intensitet i fargekomponentbildene. Etter tersklingen vil da både ball- og markørobjekter tydelig vises i bildeplanet.
4. Binær morfologi: Bruker morfologisk *åpning* for å fjerne støy og objekter mindre enn målobjektet. Fyller eventuelle hull i målobjektene, og merker gjenværende objekter med *regionprops* funksjonen. Målobjektet blir definert som det objektet med minst areal.
5. Tester objekt: Gjenskinn genererer uønskede objekter som kan gi feildeteksjon av målobjektet. Derfor testes målobjektet mot kjent modellinformasjon før målobjektet godkjennes.

Et praktisk eksempel illustrerer hvordan både ballene og robotens koordinat finnes fra algoritmen punkt for punkt.

1. Innhenting av bilde: Figur 4.7 viser bildet som skal analyseres for å finne objektenes koordinater.

   Det originale bildet inneholder endel unødvendig informasjon som kan fjernes for å forenkle videre bildebehandling. ROI defineres som robotens arbeidsområde (det grønne underlaget), og trekkes ut fra originalbildet.



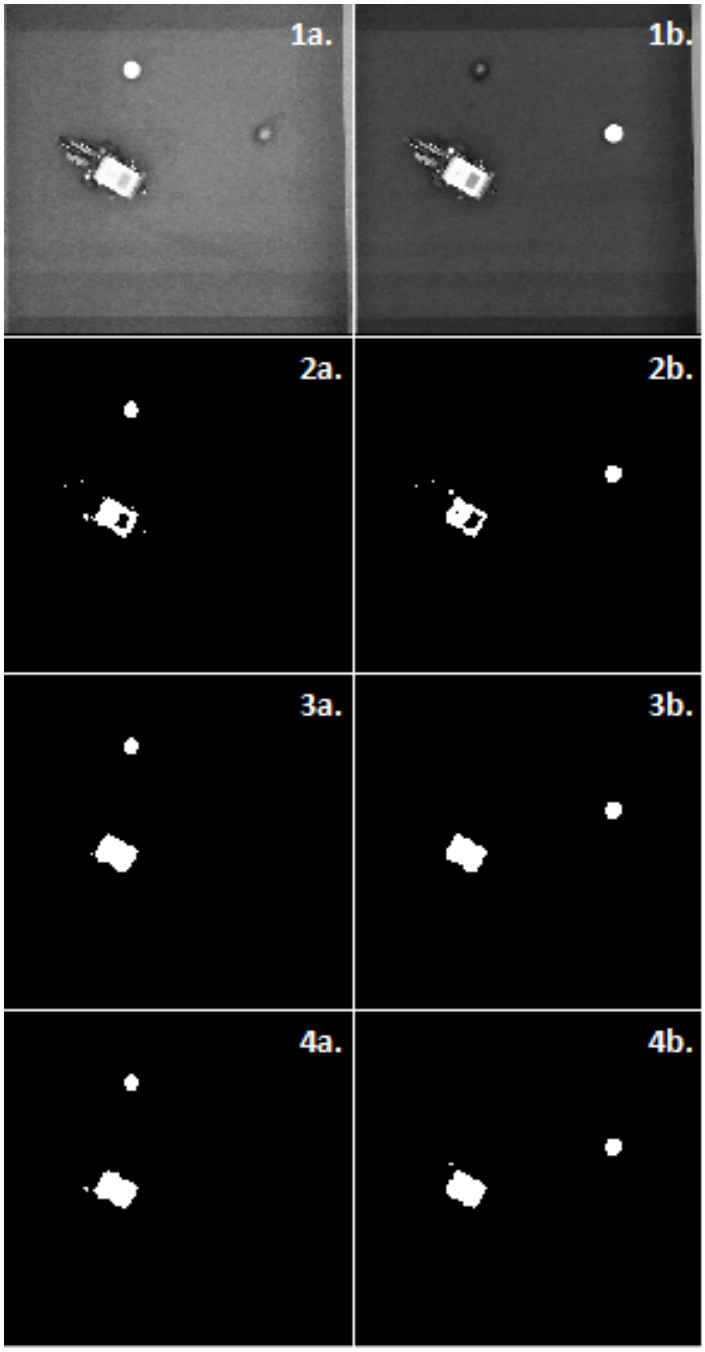
Figur 4.7: Bildet som skal analyseres for å finne objektenes koordinater.

   Resultat etter hvert steg i videre bildebehandling er vist i figur 4.8. Her tilhører a- og b-bildene henholdsvis bildebehandling utført på blått og rødt komponentbilde.

2. Klargjør bildet før terskling: ROI og fargekomponentbildene trekkes ut av originalbildet som vist i fra figur 4.8 (delfigur 1a og b). Komponentbildene: Originalbildet er nå delt inn et rødt og et blått komponentbilde, samtidig som at unødvendig informasjon er blitt fjernet. Komponentbildene er indeksbilder, der pikselintensitet er avhenging av hvilken pikselverdi den respektive fargekomponenten hadde i fargebildet. Det vil si at en høy pikselverdi for det røde komponent i fargebildet gir høy pikselintensitet for samme piksel i det røde komponentbildet. Den røde ballen i originalbildet gir høy pikselverdi for den røde komponenten, og lavere verdier for den blå og grønne komponenten. Ballen gjengis i det røde komponentbildet med høy intensitet (hvit farge). Det kan legges merke til at robotens hvite NTX også gjengis med høy intensitet for begge fargekomponentene. Dette skyldes at den hvite roboten i fargebildet inneholder høy intensitet fra alle RGB komponentene.

3. Terskling: De binære bildene etter tersklingen som vist i figur 4.8 (delfigur 2a og b) viser at både baller og robotens markører skilles tydelig ut. Det

finnes fortsatt endel ekstra uønskede objekter i det binære bildet, dette kan i stor grad skyldes gjenskinn.



Figur 4.8: Objektdeteksjon av robot og baller

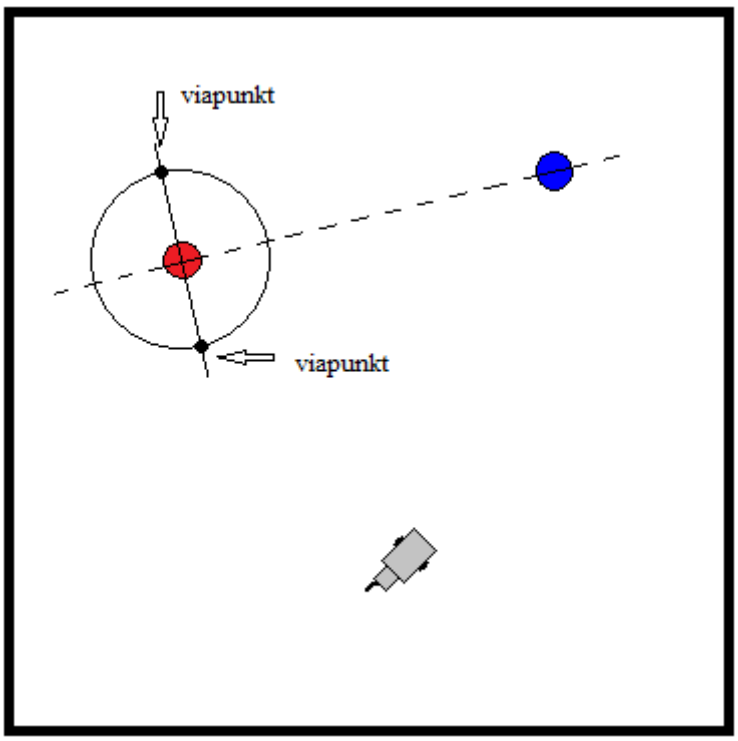
4. Binær morfologi: utføres med ulike parametre for identifiseringen av ball- og markørobjektene. Etter binær morfologi finnes ballobjektene fra figur 4.8 (delfigur 3a og b), mens markørobjektene finnes fra delfigur 4a og b. I figur 4.8 (delfigur 3a og b) er alle objekter mindre enn ballobjektet blitt

fjernet og objekthuller er fylt igjen. Strukturelementets størrelse velges i forhold til ballobjektets størrelse slik at alle mindre objekter blir fjernet. Dette fører til at kun ball- og robotobjektet gjenstår i bildeplanet. Videre brukes *regionprops* funksjonen i MATLAB for å innhente informasjon fra gjenværende objekter i forhold til objektenes areal og senterkoordinat i bildeplanet. Ballobjektet defineres som det objektet med minst arealet. I figur 4.8 (delfigur 4a og b) er alle objekter mindre enn markørobjektene fjernes og objekthull er fylt igjen. Nå er strukturelementet størrelse valgt slik at alle objekter mindre enn robotens markørobjektene fjernes. Dette fører til at kun markør-, ball- og robotobjektet gjenstår i bildeplanet. Som for ballobjektet defineres markørobjektet som minste objektet i det binære bildet. Markørobjektets pikselstørrelse i bildeplanet variere i forhold til posisjon i planet. For å ikke fjerne markørobjektet når binær morfologi utføres settes strukturelementets størrelse med litt sikkerhetsmargin. Dette kan resultere i en feildeteksjon av markørobjektet dersom gjenskinn skaper et objekt etter terskling som er større i pikselstørrelse enn grensen for objektfjerning, og samtidig mindre enn markørobjektets areal.

5. Test av identifisert markørobjekt: De to identifiserte markørobjektene testes mot gitte kriterier for å eliminere en eventuell feilidentifisering. Kriteriene bygger på kjennskap til den fysiske modellen, som for eksempel avstanden mellom de to identifiserte markørene. Dersom testen ikke godkjennes så tester algoritmen alle identifiserte objekter i bildeplanet mot hverandre til alle modellkriteriene er oppfylt.

4.4.2 Posisjonering og utførelse av slag

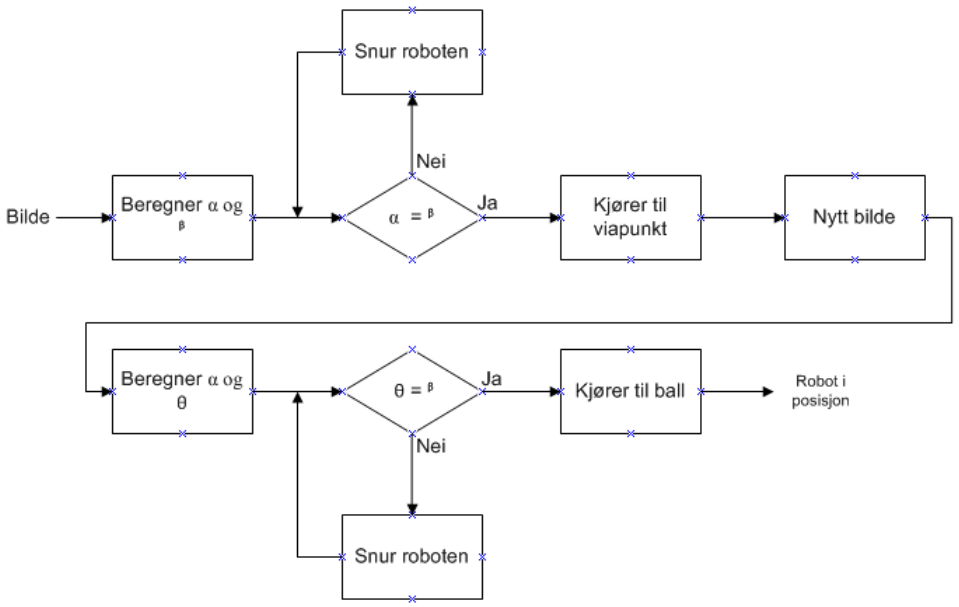
Den implementerte algoritmen som posisjonerer roboten i planet beskrives i dette delkapittelet trinn for trinn. Figur 4.9 viser et tilfeldig scenario der roboten skal posisjoneres med den implementerte algoritmen.



Figur 4.9: Scenario for posisjonering av roboten. Viapunktene brukes aktivt som hjelpepunkter ved posisjoneringen.

Algoritmens mål er å posisjonere roboten slik at den kan utføre et golfslag av den ene ballen med formål om å treffe den andre. Algoritmen fungerer for tilfeldig plassering av ballene og roboten innenfor robotens arbeidsområdet, men med noen begrensninger i forhold til fysisk gjennomførbarhet. For eksempel dersom en balls posisjon ligger helt i et hjørne av modellen vil et slag ikke være fysisk mulig, og en melding genereres til bruker av modellen. Koden er testet for ulike scenarioer som beskrevet i kapittel 6.

Posisjoneringsalgoritmens prinsipp kan enkelt illustreres i et blokkskjema som vist i figur 4.10.



Figur 4.10: Skjematisk oppbygning av posisjoneringsalgoritmen.

1. Bilde av scenario beskrevet i figur 4.9 innhentes fra webkamera.

2. Koordinater for begge ballene og robotens koordinat og orientering i modellen defineres i objektgjenkjenningsalgoritmen. Robotens orientering beskrives med vinkelen α.

3. Avgjør hvilken ball som ligger nærmest, deretter defineres de to punktene som roboten kan posisjonere seg i for å få en størst sjanse for å treffe den andre ballen. På grunn av den praktiske utforming av robotens svingarm ligger disse punktene på en linje perpendikulært fra nærmeste ball på linjen mellom de to ballene som vist i figur 4.9. Det punktet som ligger nærmest roboten defineres som viapunktet. Avstand fra viapunktet til ballen er satt utfra robotens fysiske størrelse ved rotasjon for at roboten skal kunne rotere rundt i dette punktet uten å røre ballen.

4. Retning mellom robotens senterkoordinat og det valgte viapunktet finnes, og beskrives med vinkelen β

5. Roboten roteres helt til $\alpha = \beta$. Rotasjon av roboten gjøres med både foroverkobling og tilbakekobling. Først gjøres en grovjustering av orientering der pådraget beregnes ut fra kjennskap til modellen. Siste finjustering gjøres med tilbakekobling helt til reguleringsavviket er innenfor satt kriterie.

6. Robotens avstand til viapunkt i figur 4.9 beregnes. Videre kjører roboten til viapunktet med pådrag beregnet fra kjennskap til modellen.

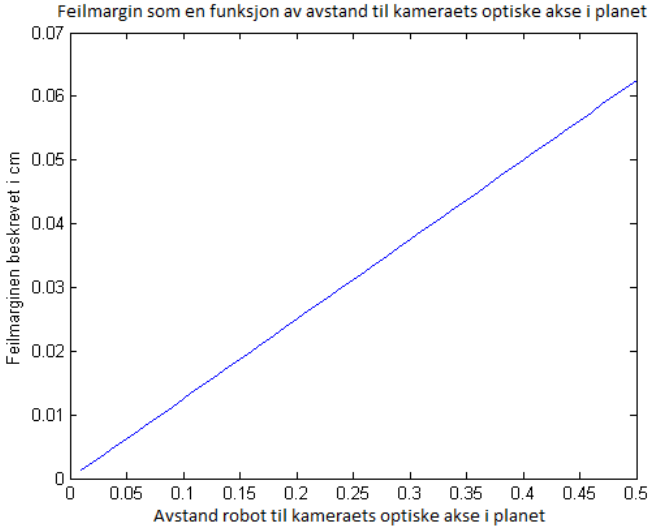
7. I viapunkt innhentes nytt bilde, robotens orientering α finnes på nytt.

Retning mellom robotkoordinat og nærmeste ballen defineres ved vinkelen γ.

8. Roboten roteres mot ball med foroverkobling og tilbakekobling helt til $\alpha = \gamma$.
9. Slagets retning finnes og svingarm heves til slagposisjon i henhold til riktig slagretning.
10. Robotens avstand til nærmeste ball beregnes, deretter posisjoneres roboten i slagposisjon inntil ballen med foroverkobling.
11. Slag utføres og nytt bilde tas for å sjekke om den andre ballen ble truffet.
12. Dersom ballen ble truffet avsluttes algoritmen, ellers starter den fra toppen igjen frem til et balltreff oppnås.

4.5 Modellfeil

Ved beregning av koordinater og avstander i modellen introduseres en feilmargin som forklart i kapittel 2.3.2. Robotens markører er montert 10 cm over planet og representerer den største kilden til feilmargin i modellen. Feilmarginen som oppstår ved bildeprojeksjon er proporsjonal med robotens avstand til kameraets optiske aksen. Figur 4.11 illustrerer teoretisk beregnet feilmarginen med bakgrunn i den fysiske modellen, som en funksjon av avstand til kameraets optiske akse.



Figur 4.11: Lineær sammenheng mellom robotens posisjon i planet og introdusert projeksjonsfeil

Dersom roboten er plassert i ytterkant av kassen blir feilmarginen på over 6 cm. En så stor feilmargin fører til dårlig presisjon ved posisjonering av roboten. Derfor må feilkilden kompenseres for i operasjonsområdet for å oppnå bedre presisjon.

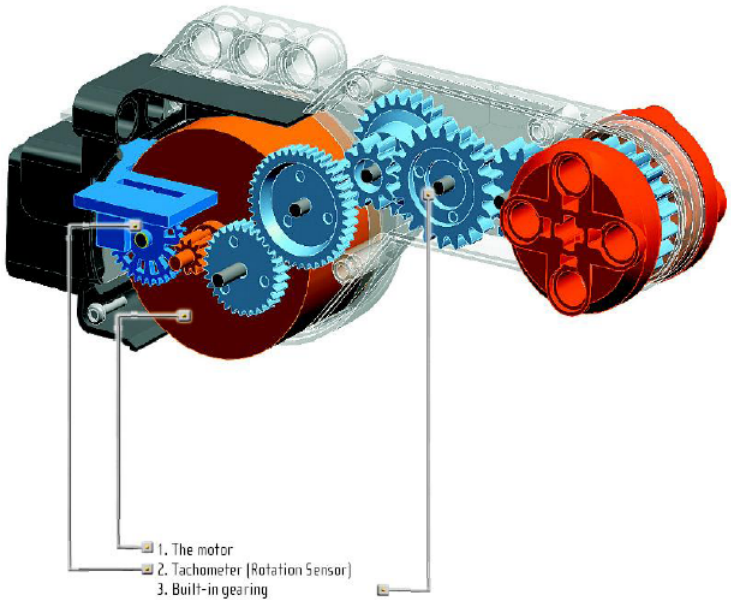
Kapittel 5

Reguleringsteknikk

Reguleringsteknikk brukes i prosjektet som en delløsning for problemstillingen. Dette kapittelet beskriver modellering av de prosessene som inngår i modellen, og deretter reguleringssløyfene som *styrer* disse. En prosess som modelleres en hvordan roboten roteres rundt sin egen akse i planet. Modellering gjøres for å skaffe nødvendig informasjon om prosessen som brukes ved implementering av prosessen i en reguleringssløyfe. Først i kapittelet gjøres modellering og deretter defineres reguleringsløyfen og valg av regulator.

5.1 Modellering av svingprosess

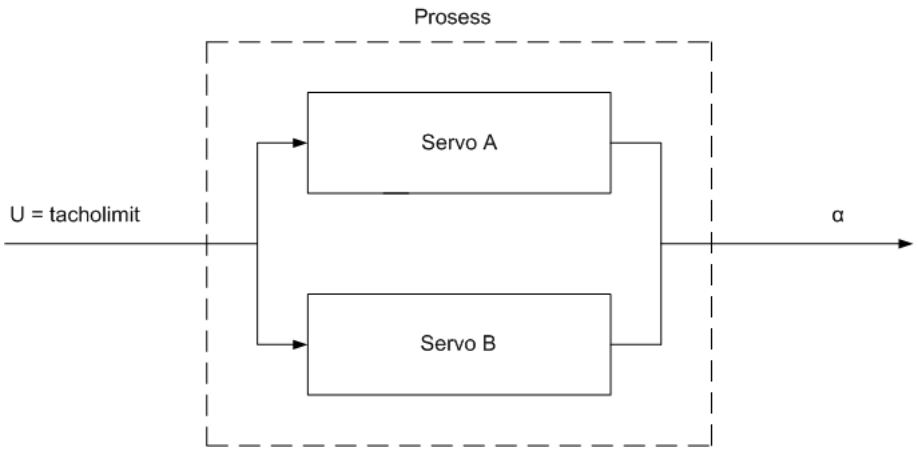
Robotens svingprosess består av to servomoter som kjøres samtidig i motsatt retning. Begge servoene er koblet direkte til hvert sitt hjul på roboten og driver disse rundt. Svingprosessen kontrolleres i en reguleringssløyfe med parametre funnet fra modelleringen. Figur 5.1 viser en skisse av servoen som brukes i roboten, som er konstruert med innebygget rotasjonssensor og regulator [6].



Figur 5.1: Servomotorens oppbygning. Figur fra [6]

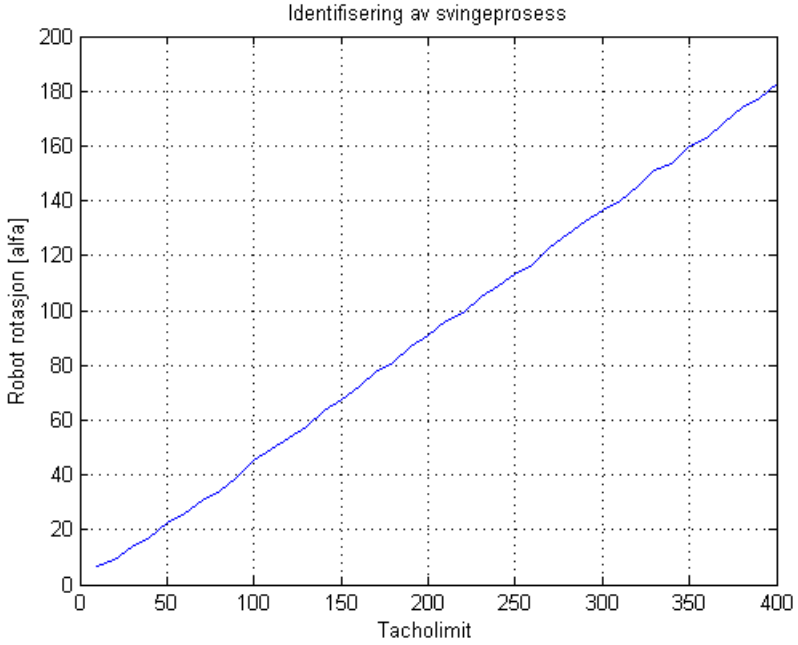
Med disse innebyggede komponentene kan servoens rotasjon kontrolleres med stor presisjon. Kontroll av servoen konfigureres i programmeringskoden ved hjelp av ulike parametre. Parameteren *tacholimit* (SKAL-verdi) definerer hvor mange grader servoen skal rotere. Rotasjonssensoren registrer hvor mange grader servoen har rotert, og lagrer data i parameteren *tachometer* (ER-verdi). Servoens innebygde regulator kontrollerer rotasjonen av servoen helt til reguleringaavviket er null.

Prosessen som kontrollerer robotens rotasjon kan er vist i figur 5.2



Figur 5.2: Blokkskjema av robotens svingprosess. Vinkelen α beskriver robotens rotasjon rundt sin egen akse etter et gitt pådrag U.

Prosesspådraget U sendes til begge servoer som er konfigurert til rotasjon i hver sin retning. For å implementere denne prosessen i et reguleringssystem må en finne forholdet mellom prosessens inn- og utgang. Prosessens arbeidsområdet er mellom 0° og 180°, som skyldes at roboten kan rotere i begge retninger og er implementert til å alltid søke korteste vei til målet. I dette arbeidsområdet ble prosessens sprangrespons plotet for å identifisere prosessen. Sprangets amplitude($tacholimit$) ble variert og testet i intervallet 10-400, og responsen ble målt i hvor mange grader roboten roterte for hvert sprang i pådraget. Modellens kamera ble brukt til beregning av responsen med allerede implementert kode. Figur 5.3 viser plot at testresultatene.

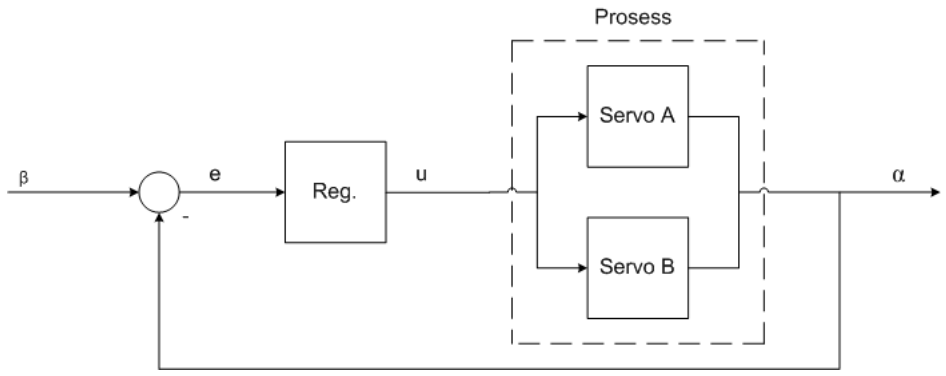


Figur 5.3: Modellering av robotens svingprosess.

Figur viser at prosessen er lineær i hele arbeidsområdet, og der prosessforsterkningen K kan finnes direkte som grafens stigningstall. Beregninger med testdata gir en prosessforsterkning $K = 0.45$. De små variasjonene i linjen skyldes unøyaktigheter i systemet, og der standardavviket fra den lineære linjen for prosessen ble beregnet til $\sigma = 1.2371$.

5.2 Reguleringssløyfe

Svingprosessen fra forrige delkapittel kontrolleres ved reguleringteknikk i en reguleringssløyfe med tilbakekobling. som vist i figur 5.4. Regulatoren beregner prosesspådraget med bakgrunn i reguleringsavviket.



Figur 5.4: Reguleringsløyfen som *styrer* robotens svingprosess. Vinkelen β representerer reguleringssystemets referanseverdi (SKAL-verdi), α den målte faktiske vinkelen (ER-verdi), e reguleringsavviket, og u prosesspådraget som beregnes av regulatoren.

5.2.1 Valg av regulator

For valg av type regulator til prosessen er det statiske reguleringsavviket en avgjørende faktor. Det optimale for prosessen i reguleringssløyfen er null statisk avvik som vil resultere i en mer presis posisjonering av roboten. Rotasjon av roboten gjøres ved to servomekanismer som hver tilfører egne integratorer til prosessen. For slike prosesser kan en P-regulator brukes for tilfredsstillende resultater, spesielt for posisjonsregulering vil de statiske følgeegenskapene bli null. Et reguleringsavvik kan fortsatt oppstå dersom det finnes en konstant forstyrrelse i prosessen, og i slike tilfeller vil en PI-regulator være å foretrekke [17]. For robotens svingprosess ble det valgt å bruke en P-regulator i reguleringssløyfen.

Kapittel 6

Resultat

Dette kapittelet viser alle resultater i prosjektet i forbindelse med utvikling av fysisk modell og realisering av tilhørende programvaren. Resultatene ligger til grunn for mange valg som har blitt gjort, og dokumenterer programvarens funksjonalitet og grad av presisjon. Først presenteres resultater fra utvikling av den fysiske modellen, deretter testes den implementerte objektgjenkjenningsalgoritmens stabilitet og klassifiseringsevne. Beregnet robotkoordinat inkluderer en misvisning som skyldes bildeprojeksjon. Denne misvisningen finnes her som en funksjon av robotens posisjon i planet slik at en korreksjon kan implementeres i koden. Til slutt demonstreres hele programvarens presisjon og nøyaktighet ved implementasjon i modellen.

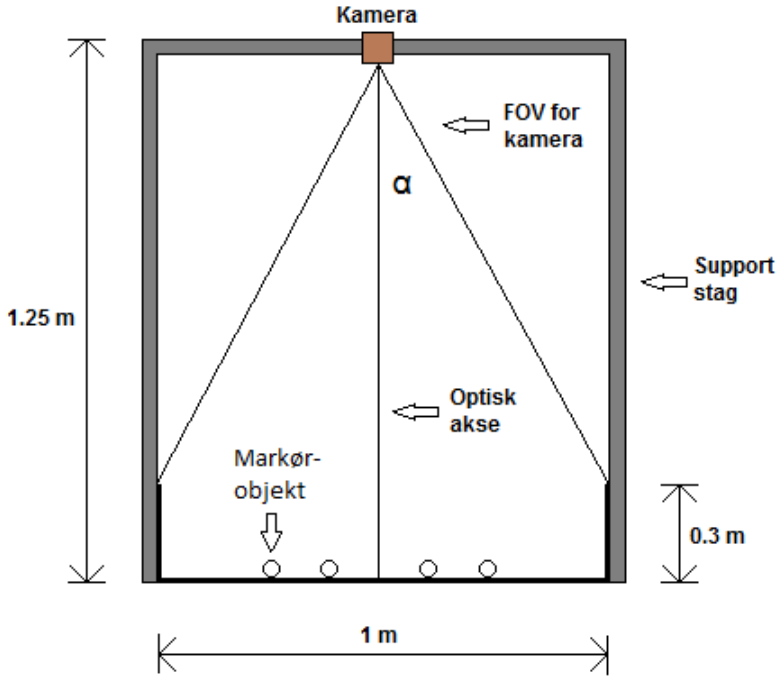
6.1 Modellutvikling

Mange valg ble gjort i forbindelse med modellkonstruksjonen, spesielt for roboten. Først i delkapittelet utdypes valg av type markører som brukes til objektgjenkjenning av roboten ved bildebehandling. Videre begrunnes de parametre som er implementert i programvaren med bakgrunn fra den fysiske modellen. Til slutt beskrives test av robotens svingradius for ulike svingmetoder.

6.1.1 Valg av markør

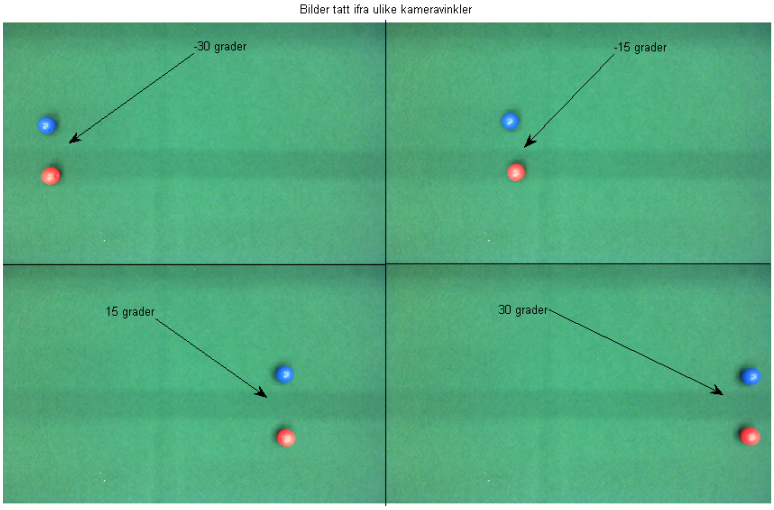
Objektgjenkjenning av roboten gjøres ved identifisering av dens markører i bildeplanet. Derfor vil en "god" markør være essensielt for et hele prosjektets funksjonalitet. Kravene til en "god" markør vil være at den er identifiserbar med stor nøyaktighet og stabilitet under ulike forhold, og god uavhengighet i forhold til gjenskinn. Markørens fysiske utforming og overflate vil ha innvirkning

på hvor mye gjenskinn som reflekteres tilbake fra objektet. Noen ulike objekter ble testet for å finne en markør som tilfredsstiller disse kriteriene. Markøren ble fotografert og analysert fra ulike kameravinkler som angitt i figur 6.1.



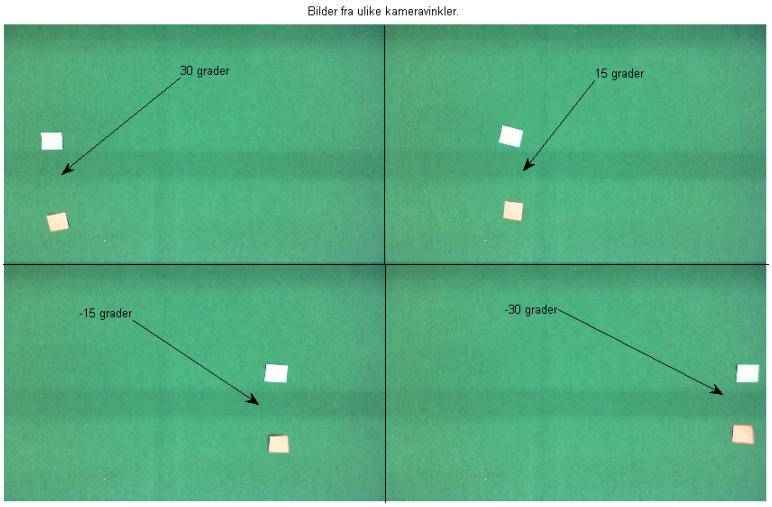
Figur 6.1: Testposisjoner for test av markørgjenskinn. Figuren viser modellen sett fra siden. Vinkelen α representerer vinkelen mellom kameraets optiske akse, og en tenkt linje mellom kameraet og de ulike markørposisjonene. Vinkelen α for markørenes plassering i testen var henholdsvis -30,-15,15 og 30 °.

En kuleformet og et flatt markørobjekt ble testet i modellen der resultatene er gjengitt i henholdsvis figur 6.2 og 6.3.



Figur 6.2: Gjenskinnstest for kuleformet markørobjekt.

Gjenskinnet i markøren kommer tydelig fram ved alle kameravinkler, men markørenes krumning gjør at gjenskinnet kun utgjør en liten del av markøren. Dvs. at markørenes område som ikke er påvirket av gjenskinnet gjengir sin farge. Det resulterer i at rød og blå markør i dette tilfellet enkelt kan skilles fra hverandre i et RGB komponentbilde.



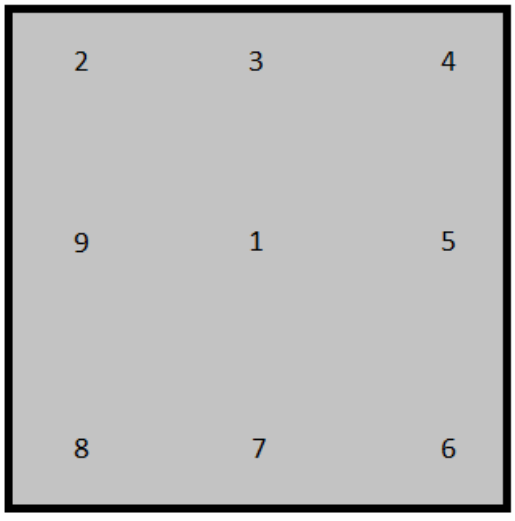
Figur 6.3: Gjenskinnstest for flatt markørobjekt.

For disse markørobjektene er det mye vanskeligere å skille gjenskinn fra markørens farge. I dette tilfellet vil høye RGB verdier på grunn av gjenskinn gjøre det vanskelig å skille rød og blå markør. De to resultatene kan ikke sammenlignes direkte fordi de to markørobjektenes overflate ikke er identisk, men resultatene gir en god indikasjon på best robusthet mot gjenskinn for det runde markørobjektet.

Den runde markøren gir et mer stabilt skille i farge for alle de ulike kameravinklene. Dette er naturlig siden kameravinkelen inn på objektet har liten betydning for mengde gjenskinn reflektert fra objektet. Lys som treffer den runde overflaten reflekteres tilbake i ulik grad i forhold til geometrien mellom lys og hvert punkt på markørobjektet. Resultatet blir at det vil alltid eksistere et område med gjenskinn så lenge en lyskilde er tilgjengelig. Ergo, derfor finnes det det også alltid et område med lite gjenskinn, og som dermed reflekterer objektets farge. Ulike kameravinkler for det flate markørobjektet viste dårligere robusthet mot gjenskinn, og på grunnlag av testen ble valget til bruk av runde markørobjekter enkelt.

6.1.2 Algoritmeparametre

Endel parametre er implementert i algoritmene som kontrollerer robotens posisjonering i modellen. Disse parametrene baseres på modellens fysiske utformingen, og brukes som både kriterier og grenseverdier i den implementerte programvaren. De fleste parametrene beskriver fysiske egenskaper for et objekt som gjenspeiles i bildeplanet. Parametrene varierer i forhold til de observerte objektenes posisjon i planet som en følge av endrede lys- og kamerabetingelser. Parametrenes variasjon ble funnet ved testing, og brukes til å definere variabelverdiene som er implementert i programvaren. Mye av testingen i dette delkapittelet og senere i rapporten gjøres i spesifikke posisjoner i planet som definert fra figur 6.4.



Figur 6.4: Testposisjoner for modellen.

1. **Pikselverdi for markørene.**

   I objektgjenkjenningsalgoritmen brukes binær morfologi for å fjerne mindre

objekter i bildet. Grensen for hvor store objekter som fjernes bestemmes av markørens størrelse i bildeplanet. Det er her ønskelig å finne en pikselgrense som vil fjerne alle objekter mindre enn markøren. Beregningen av antall piksler markørobjektet representere i bildeplanet ble testet i forhold til posisjoner i figur 6.4 på side 49. I hver posisjon ble det tatt 10 bilder som danner grunnlag for resultatene vist i tabell 6.1.

Tabell 6.1: Antall piksler for robotmarkører.

Pos.	$\overline{R}$	R_{sd}	R_{min}	$\overline{B}$	B_{sd}	B_{min}
1	111.20	0.63	110	87.80	1.75	85
2	114.10	0.74	113	93.20	2.15	89
3	120.10	1.20	118	94.70	1.89	91
4	119.80	1.87	117	97.80	1.32	96
5	115.90	1.66	113	91.30	2.06	88
6	122.20	1.55	121	99.20	3.01	94
7	123.10	0.99	122	95.90	2.73	91
8	124.20	1.43	114	91.00	1.63	88
9	113.50	0.71	112	85.20	2.37	80

Resultatene viser at pikselstørrelsen til robotmarkørene er variable i forhold til posisjon i planet. Den Røde markøren skilles noe bedre ut ved segmentering og gir en høyere pikselverdi i forhold til den blå markøren. Utfra disse resultatene defineres grenseverdiene som anvendes ved binær morfologi i objektdeteksjonsalgoritmen. For å unngå fjerning av selve markørobjektene ble grensene fastsatt med sikkerhetsmargin til henholdsvis 105 og 75 piksler for den røde og blå markøren.

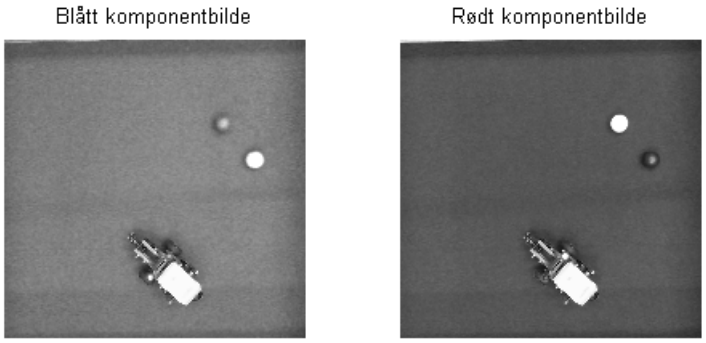
2. **Pikselverdi for rød og blå ball.** Binær morfologi brukes i objektgjenkjenningsalgoritmen til å fjerne uinteressante objekter i det binære bildet. Hvor store objekter som skal fjernes avhenger av ballobjektets størrelse i bildeplanet, altså same prinsipp som ved identifisering av markørene. Antall piksler som ballobjektene utgjør i bildeplanet ble testet på lik måte som for markørobjektene i henhold til posisjonene i figur 6.4 på side 49. I hver posisjon ble 10 bilder tatt for beregningene av resultatene som vist i tabell 6.2.

Tabell 6.2: Test av antall piksel for rød og blå ball.

Pos.	$\overline{R}$	R_{sd}	R_{min}	$\overline{B}$	B_{sd}	B_{min}
1	1306	3.4059	1300	1204	16.2494	1165
2	1446	3.5103	1441	1256	8.7914	1240
3	1378	2.7809	1370	1270	7.9645	1259
4	1432	2.5144	1428	1195	7.9057	1182
5	1374	2.2509	1372	1137	4.8028	1128
6	1418	2.1731	1416	1090	7.3060	1080
7	1273	2.6013	1267	1164	10.7062	1152
8	1345	2.2136	1341	1129	16.0281	1113
9	1415	2.1318	1413	1199	10.2897	1174

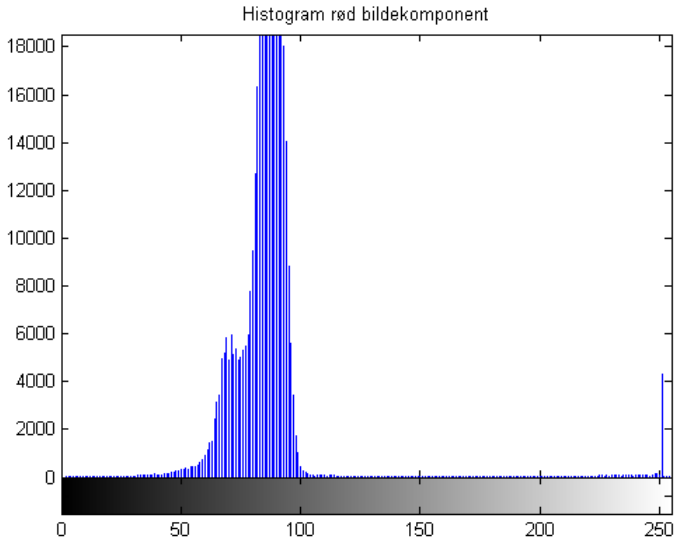
Resultatene fra tabell 6.2 viser at ballenes pikselstørrelse varierer med posisjon i modellen, og at den røde ballen genererer generelt sett et høyere antall piksler i det binære bildet. Pikselverdiene for ballobjektene er mye større enn støyobjektene som skal fjernes. Grensen for objektfjerning med binær morfologi kan settes endel lavere enn målte verdier for ballene. Da fjernes små støyobjekter fra bildeplanet samtidig som at ballobjektet beholdes. Grensen ble satt med god sikkerhetsmargin til 500 piksler for begge ballene.

3. **Terskelverdi ved terskling.** Det røde og blå fargekomponentbildet konverteres til binære bilder ved bruk av tersklingsteknikk. Valg av terskelverdi er avgjørende for hva som representeres som objekter i det binære bildet etter terskling. En optimal terskelverdi blir et kompromiss mellom fjerning av mest mulig uønsket informasjon, samtidig som at interessante objekter påvirkes i mindre grad. For programvaren ble en funksjonell terskelverdi funnet ved testing med utgangspunkt i figur 6.5. I algoritmen blir det røde og blå komponentbildet tersklet hver for seg og bildebehandlet som to separate bilder.



Figur 6.5: Utgangspunkt for terskling: Figurene viser komponentbildene som skal konverteres til binære bilder ved terskling. Det kan legges merke til at rødt komponentbilde representeres den røde ballen med en høy intensitet, tilsvarende for blå ball i blått komponentbilde, og samtidig gir også store deler av roboten høy intensitet.

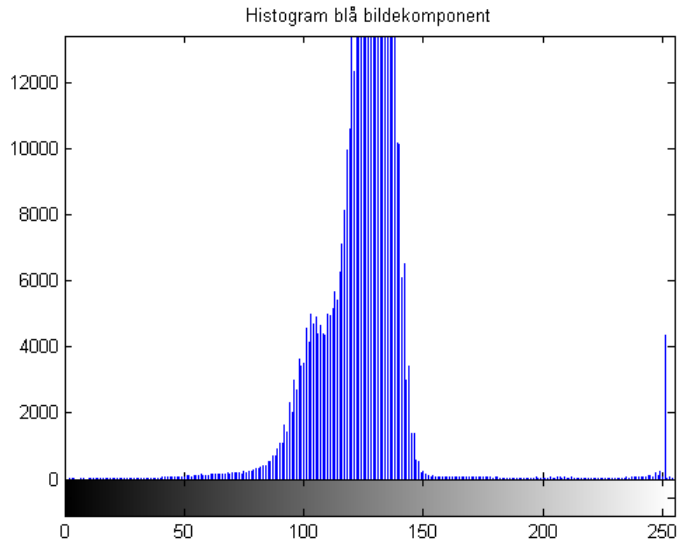
Objekter som ønskes segmentert ut fra komponentbildene er de to ballene og robotens markører. I tersklingsprosessen sammenlignes pikselintensiteten i hvert piksel med den definerte terskelverdien for å avgjør utfall av pikselverdiene i det binære bildet. En funksjonell terskelverdi finnes på grunnlag av pikselintensiten i både interesseobjekter og bildet generelt. Denne informasjonen kan finnes fra bildets histogram, som gjengir fordeling av bildets pikselintensiter. Figur 6.6 og 6.7 viser histogrammene for de to fargekomponentbildene.



Figur 6.6: Histogram for det røde komponentbildet.

X- og Y-aksen representerer henholdsvis pikselintensitet og antall piksler. Den høye toppen mellom pikselintensitet 50-100 representerer mesteparten av informasjonen i bildet, i dette tilfellet bakgrunnen. De høye verdier på

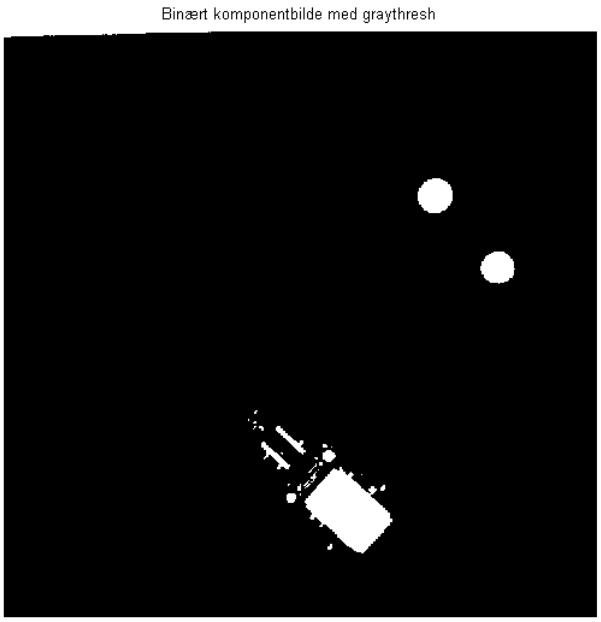
skalaen tilsvarer objektene med høyest intensitet i figur 6.5, blant annet den røde ballen.



Figur 6.7: Histogram for det blå komponentbildet.

De to histogrammene er nokså like, men det kan legges merke til at bakgrunnens pikselintensitet er noe høyere for det blå komponentbildet. Ergo kontrasten mellom bakgrunn og interessante objekter er større i det røde komponentbildet. Større kontrast gir en enklere segmentering av objektene og forklarer hvorfor rød farge gir flere piksler i det binære bildet kontra blå farge. I begge histogrammene må en høy terskelverdi benyttes for å fjerne den store toppen med unyttig bildeinformasjon. MATLAB har en egen funksjon som beregner terskelverdien automatisk med Otsu's metode [5]. For å finne en optimal terskelverdi ble terskling utført med MATLAB sin autofunksjon og med manuelt satte verdier.

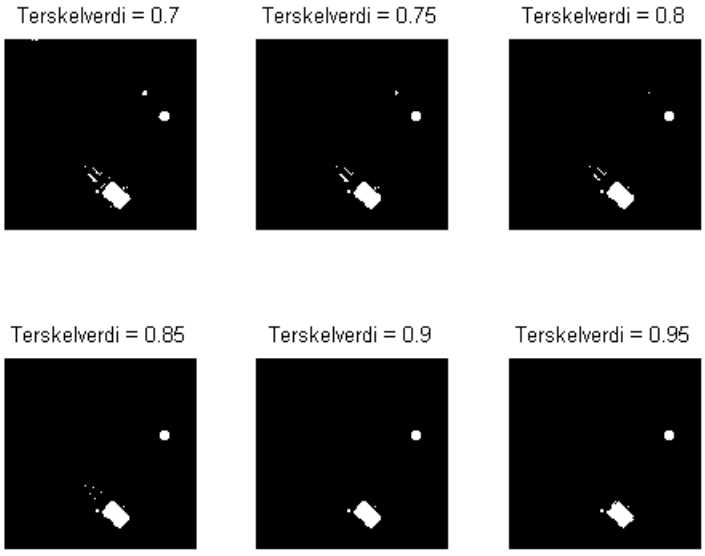
Figur 6.8 viser terskling av komponentbildene med terskelverdi funnet fra MATLAB sin *graythresh* funksjon.



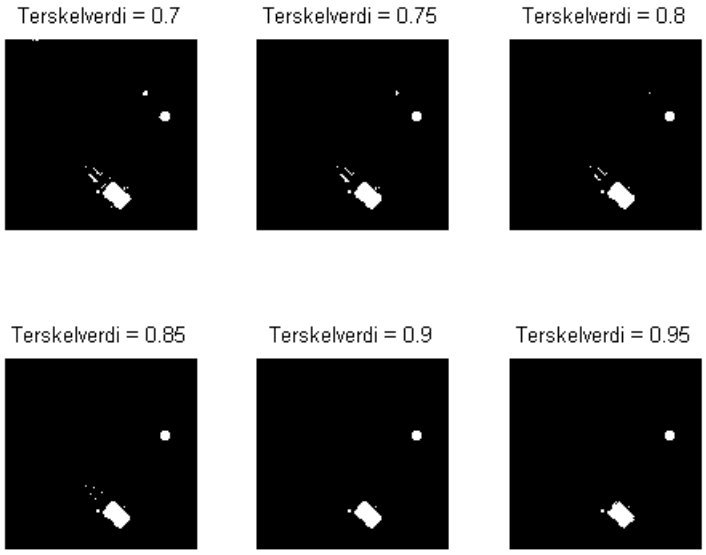
Figur 6.8: Terskelverdi funnet med *graythresh*.

Ball- og markørobjektene gjengis godt i tillegg til endel unyttig informasjon. I figuren er de binære bildene funnet fra rødt- og blått komponentbilde lagt ovenpå hverandre for å fremstille alle objektene i same bildet.

Manuelle terskelverdier ble testet som vist i figur 6.9 og 6.10.



Figur 6.9: Manuell terskelverdi for rødt komponentbilde.



Figur 6.10: Manuell terskelverdi for blått komponentbilde.

Samtidig er de ønskede objektene redusert noe i størrelse men det har ingen markant betydning for objektets senterkoordinat.

For begge komponentbildene gir høyere terskelverdi mindre informasjon i den binære bilderepresentasjonen, og vica versa for lavere terskelverdi. Ved terskelverdi 0.9 gjenstår omtrent bare ball, markør og robotobjektet. Enda høyere terskelverdi detekterer de same objektene men reduserer i tillegg objektets omfang. Terskelverdi lik 0.9 ble valgt som et kompromiss for fjerning nyttig- og unyttig informasjon i bildeplanet. Ved denne terskelgrensen oppnås tilfredsstillende segmentering av interessante objekter uten merkbar objektreduksjon.

4. **Pikselavstand mellom markørene** Avstanden mellom robotens markører brukes i algoritmen for å kvalitetssikre at de riktige markørobjektene er identifisert. Avstanden er konstant i virkeligheten, men i bildeplanet vil beregnet pikselavstand variere i forhold til robotens posisjon. Denne variasjonen ble dokumentert ved testing der 10 bilder ble tatt i hver posisjon i henhold til figur 6.4 på side 49 for beregning av pikselavstand. Tabell 6.3 gjengir resultatene av testen.

Tabell 6.3: Beregnet pikselavstand mellom robotens markører.

Pos.	Min.avst.	Maks.avst.	Diff.min.maks.
1	70.23	70.52	0.25
2	68.53	68.79	0.26
3	68.74	69.09	0.35
4	69.25	69.37	0.12
5	70.43	70.58	0.15
6	71.86	72.20	0.34
7	71.60	71.95	0.35
8	71.38	71.59	0.21
9	70.13	70.34	0.21

Denne pikselavstanden brukes som et kriterie for test av de identifiserte markørobjektene i objektgjenkjenningsalgoritmen. En nedre- og øvre grense i algoritmen settes med bakgrunn i testresultatene inkludert en sikkerhetsmargin til henholdsvis 67- og 73 piksler.

5. **Skalering pikselavstand til virkelig avstand i cm.** Den implementert programvaren bruker endel parametre som stammer fra fysiske avstander i modellen. Programvaren bruker informasjon fra bildeplanet, og derfor må forholdet til virkelig fysisk størrelse finnes.

   Dette ble gjort med en enkel test der avstand mellom to punkter på roboten ble fysisk målt til 10 cm. I neste trinn ble pikselavstanden til punktene i bildeplanet avlest til 80 piksler. Målingene i både x- og y-retning gav likt resultat som gir et skaleringsforhold som

$$S = \frac{80}{10} = 8$$

   Hver cm i planet representeres med 8 piksler i bildeplanet, og ergo kan forholdet uttrykkes generelt som

$$ant.piksler = 8 \cdot ant.cm$$

6.1.3 Svingradius

God fleksibilitet og presisjon ved manøvrering og posisjonering av roboten er viktig for å oppnå en best mulig funksjonell modell. I denne sammenheng er robotens svingradius en begrensende faktor som ble testet og dokumentert. Roboten svinges ved bruk av to servomotorer, slik at utførelse av en sving kan implementeres med ulike metoder. Valg av metode som skal brukes ble bestemt utfra både svingpresisjon og svingradius. Følgende metoder som beskrevet under ble testet.

1. Ethjulsdrift: Framoverdrift på høyre hjul mens det venstre står i ro.
2. Tohjulsdrift: Samtidig drift av begge hjulene i henholdsvis motsatt retning.
3. Periodevis drift: Først 45° sving ved fremoverdrift på høyre hjul. Sekvensen fullføres med deretter like lang bakoverdrift på venstre hjul.

Bruk av ovennevnte metoder gjør at roboten roteres rundt et punkt i planet, enten det er det ene hjulet eller senter av hjulakslingen. Presisjonen til metodene kontrolleres av dette punktet, ved målinger før- og etter at metoden implementeres. Differansen mellom de to målingene blir et mål på metodens svingpresisjon, og omtales videre som forskyvningen. Test av hver metode ble utført 10 ganger, og danner grunnlaget for beregning av både standardavviket til forskyvningen og svingradiusen.

Metode 1:
For denne metoden roterer roboten rundt aksesenter i det venstre hjulet. Før testing ble den blå markør ble plassert rett ovenfor venstre hjul for måling av forskyvning i forbindelse med. rotasjonen. Tabell 6.4 viser resultatene fra testen.

Tabell 6.4: Måleresultat for enhjulsdrift.

Test nr.	Ref.pkt.[x y]	Nytt pkt.[x y]	Forskyvning [piksler]
1	385.70 374.05	382.87 370.94	4.21
2	382.79 371.00	380.80 374.87	4.35
3	384.41 377.89	386.73 375.31	3.47
4	386.87 375.29	384.00 372.93	3.71
5	383.92 373.00	381.13 376.90	4.79
6	381.01 376.78	383.97 379.14	3.79
7	384.01 379.22	386.01 376.68	3.23
8	382.98 373.64	380.34 377.29	4.50
9	379.61 377.03	382.66 379.24	3.77
10	382.58 379.29	384.73 376.33	3.66

Beregnet koordinatpunkt før og etter rotasjon beskrives i tabellen som henholdsvis referanse- og nytt punkt. Svingradiusen for metoden beregnes som hjulakslingens radius ± gjennomsnittlig forskyvning i cm. Teoretisk sett gir drift på et hjul svingradius lik halvparten av avstanden mellom stillestående hjul og roterende hjul, dersom det statiske hjulet blir stående helt i ro. Forskyvningens retning i planet vil påvirke svingradiusen som defineres som et intervall basert på testdataene. Svingradiusen for metoden kan settes opp som: Hjulakslingens halve lengde ± gjennomsnittlig forskyving [cm] ± forskyvningens standardavvik σ. Standardavviket beskriver avviket fra gjennomsnittet til dataene og beregnes fra formel 6.1 [18].

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{6.1}$$

Hjulakslingens halve lengde ble målt til 6 cm, gjennomsnittlig forskyving beregnet til 0.49 cm, og σ beregnet til 0.06 cm.

Det resulterer i et svingradiusintervall mellom 5.45 - 6.55 cm.

Metode 2 og 3:

Roboten svinger rundt sin egen akse, derfor defineres kontrollpunktet i same punkt som robotens koordinatpunkt. Svingradiusen blir teoretisk lik null ved rotasjon rundt egen akse. I praksis vil unøyaktigheter gi utslag som testresultatene fra tabell 6.5 viser.

Tabell 6.5: Testresultat for tohjulsdrift og periodevis drift.

Test nr.	Forskyvning metode 2 [piksler]	Forskyvning metode 3 [piksler]
1	5.87	28.54
2	4.59	29.41
3	4.54	29.10
4	3.71	29.08
5	4.88	29.92
6	4.74	28.74
7	2.11	28.38
8	4.49	29.48
9	5.45	29.02
10	5.16	28.13
σ	1.04	0.54
$\overline{x}$	4.55	28.98

Svingradiusintervallet for disse metodene blir gjennomsnittlig forskyvning $\pm\ \sigma$, som gir følgende intervaller for testene

Test 2: Svingradius 0.43 - 0.69 cm
Test 3: Svingradius 3.55 - 3.68 cm

Konklusjon for svingmetodene: Av de tre ulike metodene for rotasjon av roboten gir samtidig drift på begge hjul med motsatt rotasjonsretning minst svingradius

for roboten, og implementeres i modellen.

6.2 Objektgjenkjenning

Ballenes og robotens koordinater i modellen finnes fra bildebehandling, ved identifiseres av objektene i bildeplanet. En egen algoritme ble utviklet til å håndtere denne oppgaven. Varierende lysforhold og gjenskinn fra objektene innvirker på disse faktorene i modellen for å sikre god stabilitet i modellen. Først i dette delkapittelet testes algoritmen for to ulike ballscenarioer, og deretter testes den i forhold til å finne robotens koordinater for flere scenarioer.

6.2.1 Ballkoordinat

Objektgjenkjenningsalgoritmen ble testet for identifisering av ballkoordinatene for scenarioer som beskrevet nedenfor. Formålet med testen var å verifisere og dokumentere algoritmens pålitelighet i forhold til riktig objektdeteksjon.

• Testmetode 1: Roboten er stasjonær, mens ballen ble plassert i ulike posisjoner som vist i figur 6.4 på side 49.
• Testmetode 2: Ballen er stasjonær, men med en dynamisk robot som endrer både posisjon og orientering i planet. Hensikten med denne testen er å framprovosere gjenskinn i robotstrukturen. Testen gir en god indikasjon om algoritmen er stabil nok for deteksjon av ballenes koordinater under varierende forhold.

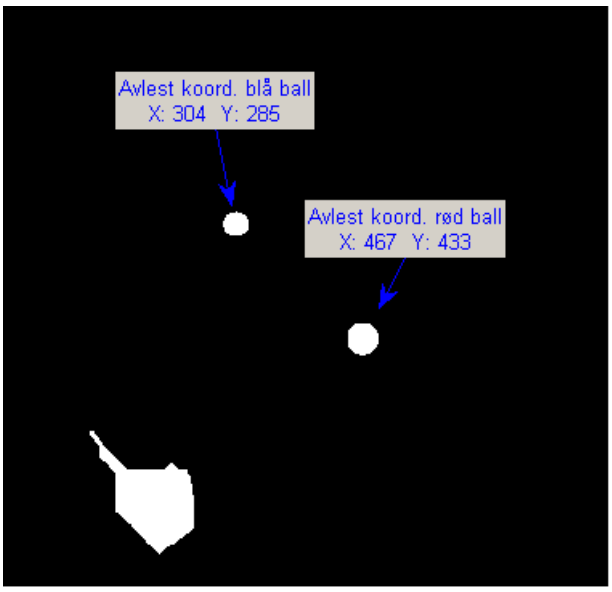
Test 1:
I hver testposisjon ble ballens koordinat i planet beregnet av objektgjenkjenningsalgoritmen, og sammenlignet med direkte avleste koordinater i MATLAB. Resultatene er gjengitt i tabell 6.6.

Tabell 6.6: Testresultater for test 1 av ballkoordinatalgoritmen.

	Koordinater for rød ball		Koordinater for blå ball	
Pos.	Avlest pikselpos. [x y]	Beregnet pos. [x y]	Avlest pikselpos. [x y]	Beregnet pos. [x y]
1	390 370	390.22 370.14	382 370	382.14 370.94
2	76 83	76.16 83.51	99 108	99.67 108.28
3	353 75	353.88 75.11	365 113	365.69 113.53
4	704 70	704.13 70.19	690 127	690.37 127.08
5	681 359	681.03 359.35	713 390	713.93 390.44
6	717 662	717.79 662.51	716 684	716.82 684.49
7	374 679	374.37 679.64	355 691	355.97 691.62
8	91 688	91.35 688.89	94 676	94.65 676.94
9	71 402	71.15 402.77	79 394	79.74 394.94

Sammenligning av avleste- og beregnede pikselverdier forteller at algoritmen identifiserte ballkoordinatene riktig i alle posisjoner for begge ballene.

Test 2: Figur 6.11 viser tilfeldig plassering av ballene for testen. Objektgjenkjenningsalgoritmen ble så testet i forhold til ulike posisjoner og orientering for roboten for å dokumentere eventuell innvirkning på beregnede ballkoordinater. Resultatene er presentert i tabell 6.7.



Figur 6.11: Plassering av den røde og blå ballen. Det store objektet representerer en gjenspeiling av roboten i bildeplanet.

Tabell 6.7: Testresultater for test 2 av ballkoordinatalgoritmen.

Robot pos.	Robot Orientering [°]	Beregnet pos. rød [x y]	Beregnet pos. blå [x y]
2	0	468.38 435.90	302.07 285.06
	90	467.43 434.88	302.87 285.65
	180	467.54 433.97	304.08 285.50
	270	467.52 433.99	303.82 285.31
3	0	467.50 434.00	304.53 284.70
	90	467.50 434.00	304.65 286.13
	180	467.50 434.00	304.61 286.28
	270	467.49 433.98	360.45 192.60
4	0	467.49 433.98	304.75 286.42
	90	467.43 433.99	304.91 286.45
	180	467.45 433.96	304.77 286.56
	270	467.47 433.99	304.80 286.61
5	0	467.54 433.97	304.78 286.64
	90	467.44 433.98	304.66 286.45
	180	467.51 433.94	304.76 286.54
	270	467.47 433.96	304.79 286.34
6	0	467.43 433.98	304.75 286.42
	90	467.45 433.96	304.71 286.36
	180	467.47 433.96	304.70 286.33
	270	467.43 433.96	304.62 286.44
7	0	467.46 433.96	304.68 286.22
	90	467.42 434.05	304.70 286.13
	180	467.46 433.96	304.71 286.37
	270	467.46 433.95	304.79 286.36
8	0	467.46 433.96	304.64 286.45
	90	467.44 433.98	304.78 286.27
	180	467.43 433.96	304.73 286.30
	270	467.46 433.96	304.84 286.31
9	0	467.49 433.97	304.85 286.30
	90	467.48 433.97	304.77 286.30
	180	467.46 433.96	303.95 287.25
	270	467.48 433.98	304.80 286.42

Algoritmen beregner ballenes koordinater 100 % riktig for de utvalgte robotkriterier. Med bakgrunn i testene blir konklusjonen at algoritmen beregninger riktige ballkoordinater, uavhengig av robotens posisjon og orientering i planet.

6.2.2 Robotens koordinat og retning

Robotens koordinat og orientering finnes ved identifisering av markører som beskrevet i delkapittel 4.3. Kort oppsummert så identifiseres markørenes koordinat i bildeplanet, og brukes til beregning av robotens posisjon og orientering i planet.

Den implementerte algoritmen ble testet for ulike scenarioer i forhold til robotens plassering og orientering i planet. I hver posisjon i henhold til figur 6.4 på side 49 ble ulike robotorienteringer testet for å simulere virkeligheten bedre. For testing ble roboten plassert manuelt i forhold til orientering og posisjon som indikert i tabell 6.8. Avlest koordinat i hver posisjon, ble definert som det avleste koordinat i bildeplanet ved orientering 0°, og ble videre brukt som referansekoordinat for de resterende orienteringer. I hver testplassering ble robotens koordinat og orientering beregnet fra algoritmen for å verifisere dens pålitelighet.

Tabell 6.8: Testresultat for identifisering av robot posisjon og orientering i planet.

Pos./avlest koord [x y].	**Orient[°].**	**Beregnet koord [x y].**	**Beregnet orient[°].**
1 [379 334]	0	378.95 334.66	358.64
	45	380.01 341.32	41.96
	90	363.10 351.40	89.03
	135	371.06 352.40	133.68
	180	367.40 364.49	179.47
	225	375.13 361.81	216.76
	270	366.43 368.32	269.09
	315	372.62 372.84	321.31
2 [171 149]	0	171.38 149.21	359.80
	45	168.69 149.78	42.94
	90	167.91 150.04	89.56
	135	167.77 148.50	134.18
	180	168.96 147.23	178.43
	225	170.65 146.69	222.39
	270	171.14 146.78	270.46
	315	172.11 147.54	315.79
3 [395 123]	0	395.75 123.46	0.26
	45	396.29 122.91	45.57
	90	395.10 121.54	91.38
	135	394.15 119.76	135.73
	180	394.05 118.74	179.24
	225	394.68 117.95	223.93
	270	396.40 117.84	267.42
	315	397.99 118.71	312.12
4 [662 102]	0	662.68 102.03	359.54

Fortsettes på neste side

Tabell 6.8 – *Fortsettelse fra forrige side*

Pos./avlest koord [x y].	**Orient[°].**	**Beregnet koord [x y].**	**Beregnet orient[°].**
	45	662.23 102.59	46.16
	90	659.90 101.02	93.37
	135	658.56 99.33	137.13
	180	659.04 97.22	181.48
	225	660.45 94.99	226.00
	270	662.87 94.85	269.25
	315	664.74 95.49	312.79
	0	680.85 390.58	358.81
	45	680.06 391.13	45.49
	90	644.10 376.91	88.29
	135	642.76 373.38	136.39
5 [680 390]	180	642.14 373.21	178.58
	225	642.18 371.79	224.28
	270	642.84 370.90	267.84
	315	643.36 372.22	312.12
	0	689.54 660.88	358.41
	45	688.05 662.06	42.24
	90	683.62 660.62	88.11
	135	681.50 657.27	133.05
6 [689 660]	180	682.47 655.73	177.82
	225	666.33 631.11	225.46
	270	681.55 627.25	268.97
	315	691.08 623.97	321.13
	0	375.70 660.76	359.15
	45	374.51 662.40	47.95
	90	373.15 663.35	90.14
	135	372.10 662.31	136.33
7 [375 660]	180	372.45 660.72	181.73
	225	374.35 659.68	226.05
	270	376.03 659.11	271.34
	315	377.50 660.03	316.31
	0	121.82 682.29	358.89
	45	118.24 683.51	43.05
	90	116.20 683.67	87.59
	135	114.83 682.19	132.63
8 [121 682]	180	115.27 680.17	178.86
	225	117.10 679.12	124.77
	270	119.23 678.94	271.24
	315	120.50 681.16	317.30
	0	119.57 389.26	1.01
	45	115.36 391.24	44.84
	90	112.94 390.49	89.41
	135	111.26 388.88	134.29
9 [119 389]			

Fortsettes på neste side

Tabell 6.8 – *Fortsettelse fra forrige side*

Pos./avlest koord [x y].	**Orient[°].**	**Beregnet koord [x y].**	**Beregnet orient[°].**
	180	110.49 386.77	179.08
	225	111.67 384.40	224.38
	270	114.19 384.81	268.15
	315	115.55 386.22	314.62

Resultatene i tabellen viser noe avvik mellom beregnede- og avleste verdier. Det skyldes i hovedsak unøyaktighet ved manuell flytting av roboten til en ny posisjon. Bortsett fra de små avvikene beregner algoritmen riktig robotinformasjon i alle tilfeller. På grunnlag av disse resultatene ble objektgjenkjenningsalgoritmen implementert i programvaren.

6.2.3 Koordinatstabilitet

Varierende lysforhold for modellen innvirker på bildet som danner grunnlaget for beregning av objektkoordinater. For å dokumentere denne variasjonen ble det ble en relevant test iverksatt. 10 bilder av same scenen for å beregne standardavviket i de beregnede koordinatene. Tabell 6.9 viser resultatene fra denne testen.

Tabell 6.9: Testresultat fra koordinatstabilitet.

Objekt	Std.avvik x-koord.[piksler]	Std.avvik y-koord.[piksler]
Rød ball	0.83	0.02
Blå ball	0.84	0.05
Robot	0.01	0.03

Resultatene fra tabellen bekrefter noe variasjon i beregnet koordinat mellom bildene. Variasjonen er relativt liten. Det største standardavviket finnes for den blå ballens x-koordinat, og tilsvarer her 0.1 cm som ikke vil gi store utslag i prosjektets presisjon.

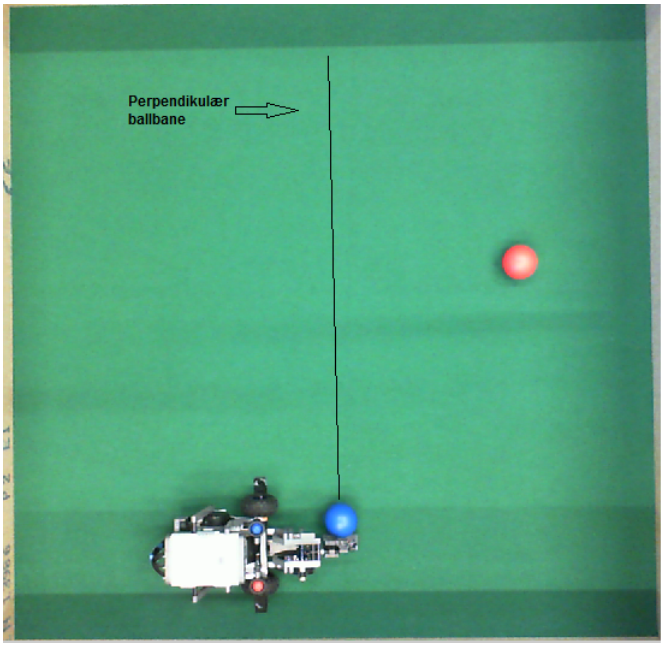
6.3 Feilmarginer

Robotens presisjon og nøyaktighet i forhold til posisjonering og treffsikkerhet påvirkes av projeksjonsfeil og unøyaktigheter i modellen. Først i dette delkapittelet

beskrives test av svingarmens presisjon i forhold til ballbane, og deretter finnes en funksjon som beskriver misvisningen som oppstår fra bildeprojeksjon i modellen.

6.3.1 Svingarm

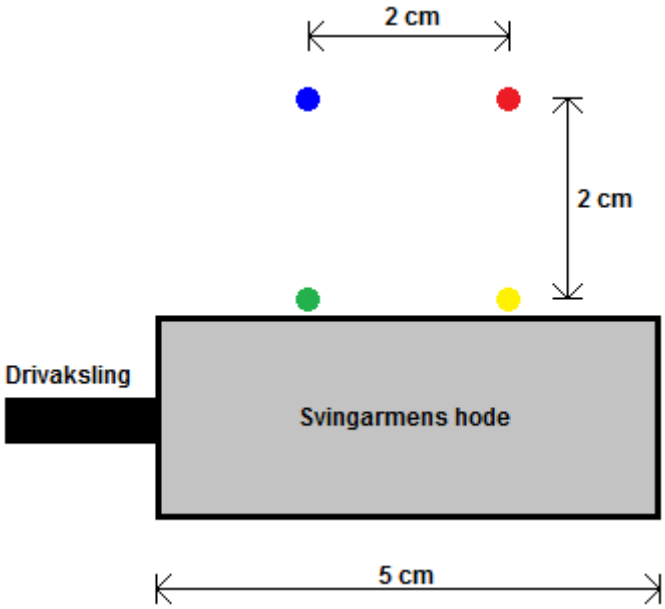
Den praktiske løsningen av robotens svingarmen representerer et usikkerhetsmoment i forhold til presisjon og nøyaktighet. Etter en fysisk inspeksjon av svingarmen ble det påvist dødgang i akslingen til svingarmen, både i vertikal, horisontal og rotasjonsretning. Akslingens dødgang i horisontalretning endrer svingarmens inngangvinkel til ballen og kan gi utslag i ballbanen. Dødgangen i horisontal retning ble før praktisk testing målt til 2 mm. Dødgang i akslingen skyldes både girsystemet, og unøyaktigheter i den fysiske konstruksjonen. I tillegg er den er laget av bøyelig plast, slik at sentrifugalkreftene ved et slag kan føre til en vridning i svingarmen som resulterer i dårligere presisjonen. Med bakgrunn i ovennevnte opplysninger ble nødvendig testing gjort med utgangspunkt i scenario fra figur 6.12.



Figur 6.12: Utgangspunkt for presisjonstesten. Bildet er tatt av modellens kamera. Objektgjenkjenningsalgoritmen beregnet robotens orientering til ca. 359° i planet.

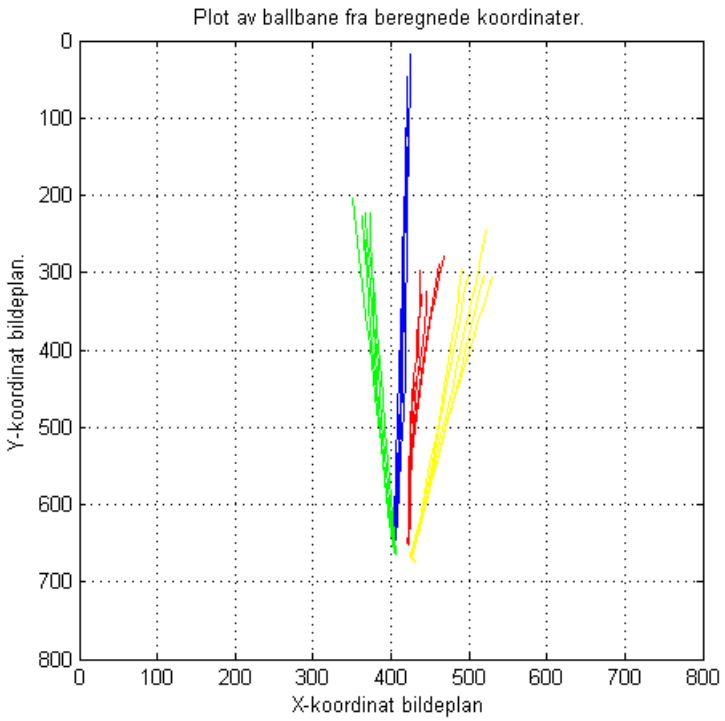
Den teoretiske ballbanen fra figuren ligger perpendikulært på robotens orientering, og definerer referansepunktet for svingarmens presisjon og nøyaktighet i testen. Robotens posisjonering i planet skjer med en viss feilmargin. Denne modellfeilen fører til at robotens oppstilling ved ballen kan variere for hver gang posisjonering

gjøres. Derfor ble ulike balltreffpunkter på svingarmen som vist i figur 6.13 testet for å dokumentere ulike treffpunkters innvirkning på ballbanen.



Figur 6.13: Svingarmens treffpunkter for testen. De fargede punktene i figuren er definert som ballens senterpunkt ved ballplassering.

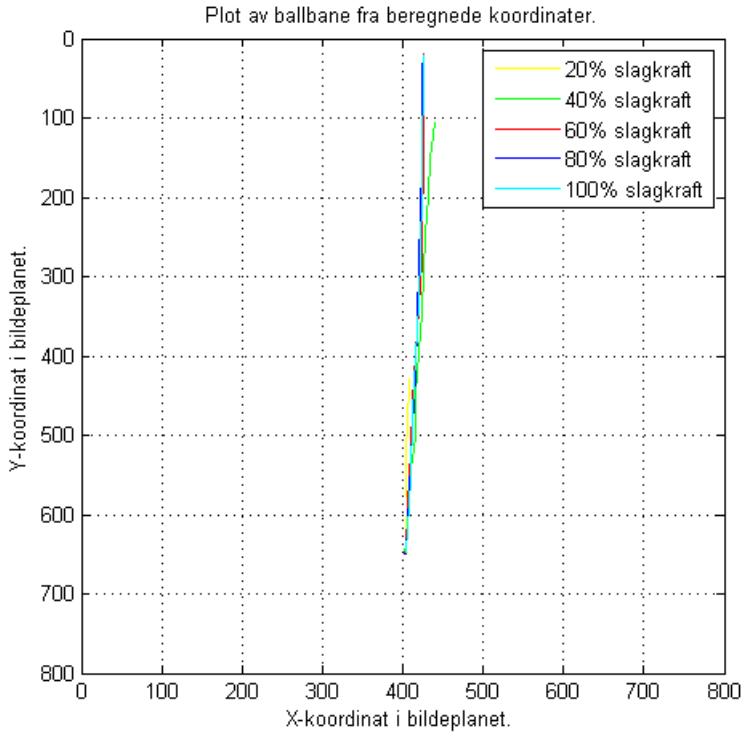
I hvert treffpunkt ble 5 slag utført fra same utgangspunkt som vist i figur 6.12. Modellens kamera ble satt opp til videologging av ballbanen i planet, og servoens slagkraft ble satt til 60 %. Kameraet ble satt opp med automatisk trigger i algoritmen for å starte akkurat i det slaget gikk. Avstanden mellom hvert kamerabilde ved logging ble satt til 50 ms, som resulterte i at 13 bilder fra logging ble hentet ut for å dekke hele hendelsesforløpet. For hvert bilde ble ballkoordinat beregnet og plotet som vist i figur 6.14.



Figur 6.14: Ballbaner for ulike treffpunkter på svingarmen. Fargene representer ballbaner i henhold til treffpunkter som definert i figur 6.13. Ulik lengde på plottene skyldes ulik kraftoverføring til ballen for treffpunktene.

Resultatene viser høy presisjon for hvert treffpunkt men med dårlig nøyaktighet, med unntak av blått treffpunkt der også nøyaktigheten er bedre. Dette danner grunnlaget for å anta at det finnes et ideelt treffpunkt mellom grønt- og blått treffpunkt, som vil gi en ballbane perpendikulært på robotens orientering i planet. Ballplassering i forhold til svingarmens hode har betydning for ballbanen, som betyr at nøyaktig posisjonering er nødvendig for å oppnå god treffsikkerhet for roboten.

Varierende slagkraft ble også testet for å se om økende sentrifugalkrefter i svingarmen ville gi utslag i ballbanen. Ulik slagkraft ble testet for blått treffpunkt, og ballbanene ble logget og plotet som vist i figur 6.15.



Figur 6.15: Ballbaner for ulik slagkraft fra same treffpunkt.

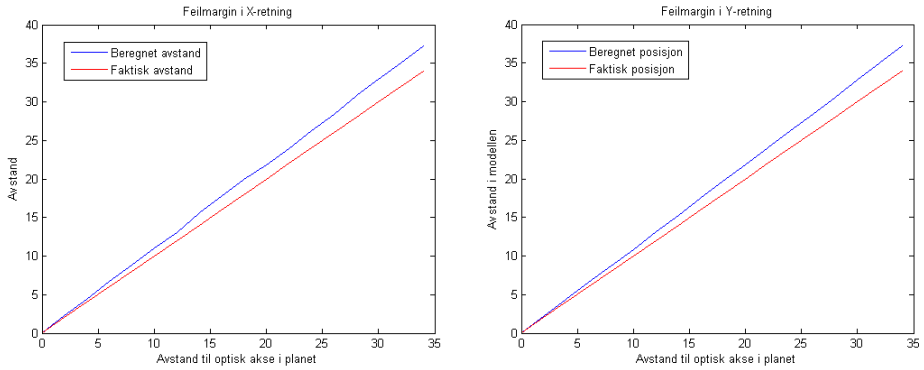
Figuren viser at ulik slagkraft ikke har innvirkning på ballbanens retning i planet, og at presisjonen er god. Med bakgrunn i disse testene kan det konkluderes med at svingarmens konstruksjon gir opphav til en feilmargin i forhold til slagpresisjon, og at robotens plassering i forhold til ballen har stor betydning for ballbanen. Ellers hadde slagkraften ingen betydning for ballens retning.

Hvert treffpunkt på svingarmen vurdert for seg selv tilsier en god presisjon men med dårlig nøyaktighet for ballbanene. En samlet vurdering av resultatene for svingarmen som en enhet gir tilsier dårlig presisjon med god nøyaktighet.

6.3.2 Modellfeil ved projeksjon

Ved projeksjon av scenens objekter inn i bildeplan introduseres en misvisning i modellen. Objektene høyde i scenen vil påvirke misvisningen av objektene i bildeplanet. For å korrigere for misvisningen ble en test utført med det formål å utvikle en generell korreksjonsfunksjon for modellen. I teorien vil en robotplassering rett under kameraet (i kameraets optiske akse) gi null misvisning, og videre vil misvisningen øke lineært med robotens forflytning vekk fra den optiske aksen. Med utgangspunkt i kameraets optiske akse ble roboten flyttet i både x- og y-retning med 2 cm intervaller. I hver posisjon ble robotens beregnede koordinat

funnet fra algoritmen og sammenlignet mot fysisk målt avstand. Figur 6.16 gjengir resultatene fra testet og viser den lineære misvisningen i modellen.



(a) Avvik i x-retning (b) Avvik i y-retning

Figur 6.16: Modellens misvisning i planets x- og y-retning

Misvisningen er tilnærmet lik i henholdsvis x- og y-retning, og er tydelig lineært økende med økende avstand fra den optiske aksen. I praksis betyr det at beregnede koordinater gjengis lenger bort fra kameraets optiske akse en robotens faktiske posisjon i planet.

Misvisningen kan kompenseres for direkte i en funksjon som beskriver forholdet mellom de to grafene i eksempelvis figur 6.16(b). Funksjonen finnes med utgangspunkt i ligningen for grafene.

- Beregnet koordinat (blå linje): $y_1 = 2.071 \cdot x_1$
- Virkelig koordiant (rød linje): $y_2 = 1.889 \cdot x_2$

X-verdiene for grafene er like slik at $x_1 = x_2$ gir

$$\frac{y_1}{2.071} = \frac{y_2}{1.889}$$

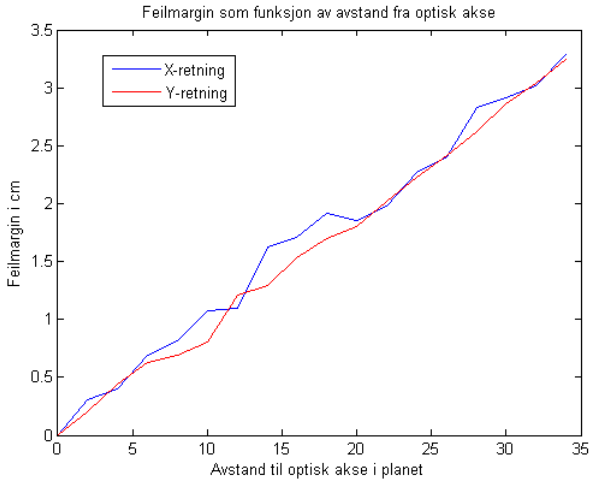
Ved å løse ligningen for y_2 som gir virkelig koordinat basert på det beregnede fra algoritmen, kan korreksjon for misvisningen implementeres direkte som

$$y_2 = 0.912 \cdot y_1$$

Funksjonen vil korrigere for forskjellen mellom grafene men kan ikke brukes i modellen på grunn av måten bildeplanets aksesystem er definert på. Som

figur 4.5 viser vil en robotposisjon i de ulike kvadrantene kreve at misvisningen korrigeres med riktig fortegn henholdsvis i x- og y-retning.

Misvisningen mellom grafene som representerer feilmarginen fra projeksjonen ble plottet som vist i figur 6.17.



Figur 6.17: Feilmargin fra projeksjon

Sammenligning av faktisk feilmargin med den teoretisk beregnede i kapittel 4.5 viser at den faktiske feilmargin blir noe mindre enn beregnet. Feilmarginen settes opp som en funksjon av avstanden til den optiske aksen som

$$F = 0.096 \cdot A \tag{6.2}$$

der F er beregnet feilmargin og A representer avstand fra det aktuelle koordinat til den optiske aksen.

Den implementerte korreksjon for feilmarginen i modellen kan oppsummeres som følger:

1. Avstand fra det beregnede koordinat til den optiske akse finnes i henholdsvis x- og y-retning.
2. Feilmargin i henholdsvis x- og y-retning beregnes fra ligning 6.2.
3. Korreksjon av misvisningen gjøres videre med kjennskap til hvilken kvadrant det beregnede koordinat tilhører.

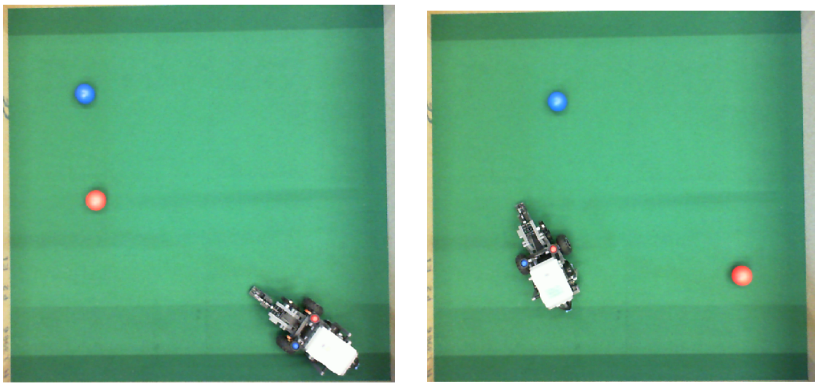
6.4 Posisjonering

Ved posisjonering av roboten i modellen tas all implementert programvare i bruk. Robotens posisjonering og treffsikkerhet brukes som et mål på hele prosjektets presisjon. Robotens treffsikkerhet ble testet som en funksjon av ballavstand, og til slutt ble presisjon ved posisjonering av roboten dokumentert.

Et par videoer av robotposisjonering er lagt ved i appendiks D. En av videoene kan også sees på denne YouTube linken:
"http://www.youtube.com/watch?v=1KCSE51zWbw"

6.4.1 Treffsikkerhet

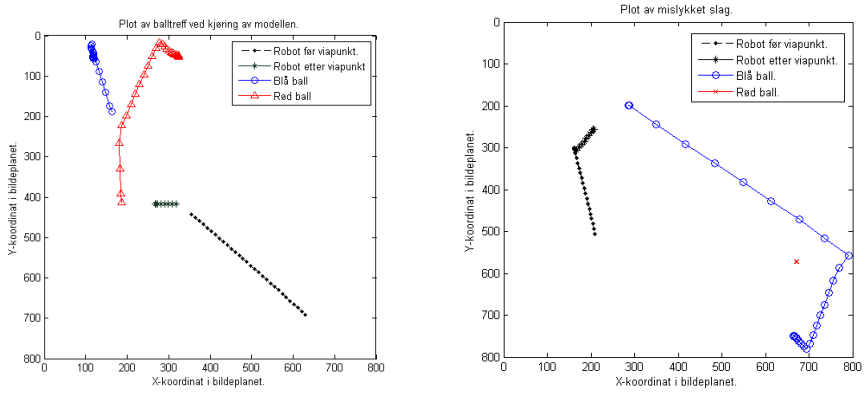
Robotens presisjon i forhold til å treffe målet sitt ble plotet for to ulike ballavstander som vist i figur 6.18.



(a) Scenario med liten ballavstand. (b) Scenario med lang ballavstand

Figur 6.18: 2 scenarioer for test av slagpresisjon.

Programvaren ble implementert for begge scenarioene og resultatet plotet som vist i figur 6.19. Resultatene ble bom og treff for henholdsvis lang og kort avstand mellom de to ballene.



(a) Roboten treffer målet for liten ballavstand.

(b) Roboten bommer på målet sitt ved lang ballavstand

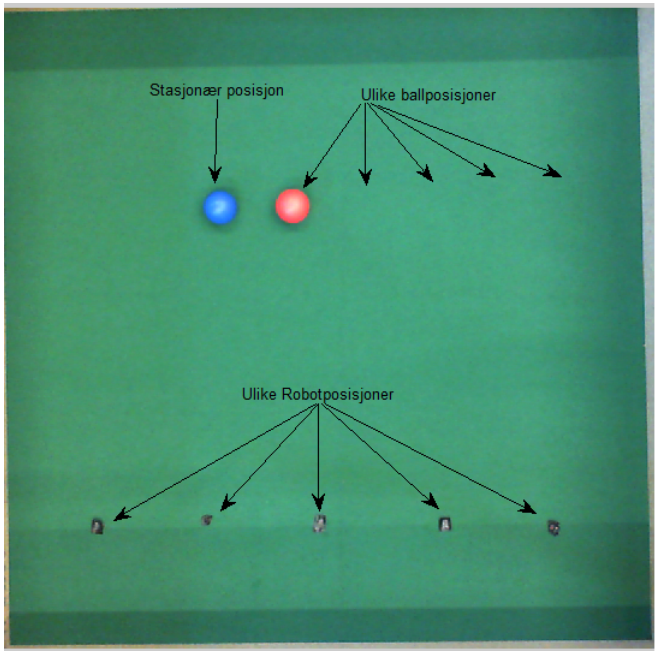
Figur 6.19: Plot av posisjonering og slag.

Denne unøyaktigheten danner grunnlaget for grundigere testing av modellen for å avdekke faktorer som resulterer i dårlig presisjon.

6.4.2 Robotens treffrate som funksjon av avstand

Treffsikkerheten ble testet som en funksjon av avstanden mellom ballene. Dette ble gjort både med og uten koordinatkorreksjonen aktivert for å dokumentere forskjellen i suksessraten.

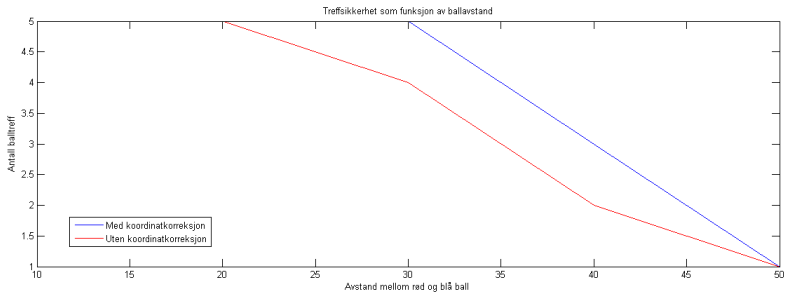
Roboten ble testet fra fem ulike posisjoner i planet som vist i figur 6.20.



Figur 6.20: Posisjoner for test av treffsikkerhet.

I hver punkt ble det utført fem tester for ulike avstander mellom ballen for å dokumentere robotens treffsikkerhet.

Figur 6.21 viser plot av treffsikkerheten som en funksjon av ballavstanden både med og uten korreksjon for objektkoordinat.

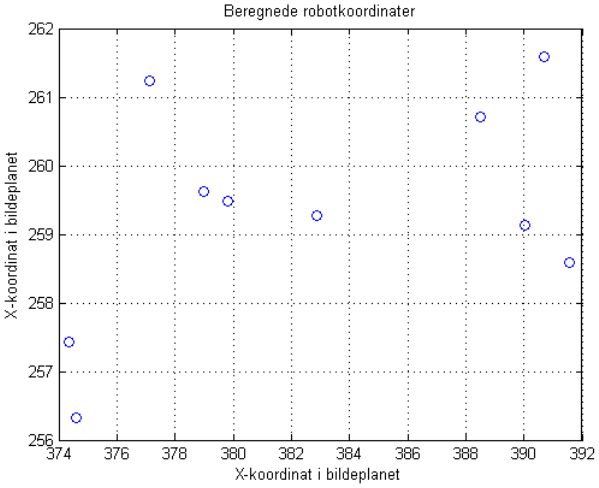


Figur 6.21: Treffsikkerhet som en funksjon av ballavstand både med og uten korreksjon for feilmargin i modellen.

6.4.3 Presisjon ved balloppstilling

For god presisjon av i forhold til ballbane ved slag er en stabil, riktig og presis oppstilling i forhold til ballen avgjørende for ballbanen som forklart i delkapittel 6.3.1. I dette delkapittelet testes og dokumenteres robotens presisjon ved posisjonering

i forhold til ballen. Ballene plasseres tilfeldig i planet, der posisjonere deres er stasjonære for testen. Roboten plasseres i ti ulike tilfeldige posisjoner i planet, og i hver posisjon skal roboten posisjonere seg i forhold til nærmeste ball. Når roboten er ferdig posisjonert i balloppstilling avleses roboten beregnede koordinat og plotes som vist i figur 6.22. Plotene fra testen verifiseres at robotens fysiske posisjon ved posisjoneringen varierer i planet.



Figur 6.22: Robotplassering ved ball for ulike utgangspunkt.

X- og Y-koordinatenes standardavviket for resultatene ble henholdsvis 6.82 og 1.83 i pikselavstand, som tilsvarer 0.85 og 0.22 cm. Robotens posisjonvariasjon ved oppstilling til ballen er relativt liten men denne unøyaktigheten gir store utslag i ballbanen som vist i delkapittel 6.3.1.

Kapittel 7

Konklusjon

Utviklet programvare som *styrer* roboten fungerer som et identifiserings- og navigasjonssystem for roboten. Navigasjonssystemets funksjonalitet er tilfredsstillende, men unøyaktigheter i modellen og introdusert misvisning fra projeksjon i bildeplanet reduserer programvarens presisjon og nøyaktighet ved posisjonering av roboten i planet.

Hovedalgoritmene i programvaren ble testet i forhold til stabilitet,presisjon og nøyaktighet.

Test av ulike robot og ballplasseringer gav ingen feildeteksjon av noen objekter for objektgjenkjenningsalgoritmen. Algoritmens stabilitet er god men robotens beregnede koordinat inneholder en misvisning. Med andre ord så gjengis det beregnede koordinat med god presisjon og dårlig nøyaktighet. Korreksjon for misvisning ble funnet og implementert for bedre nøyaktighet ved posisjonering. Testing viser at selv med korreksjon for misvisning så oppstår det noe avvik ved gjentatt posisjonering til same punkt fra ulike startposisjoner. Dette resulterer i store utslag for ballbanen ved slag, og trekker ned robotens treffsikkerhet. Posisjoneringsalgoritmen posisjonerer roboten med god nøyaktighet, men med dårlig presisjon. Ballbanen etter utført slag ble vist avhengig av svingarmens treffpunkt på ballen. Derfor vil små variasjoner i posisjonering ved ball gi store utslag i forhold til robotens treffsikkerheten som en følge av dette. Treffsikkerheten for roboten reduseres raskt som en funksjon av ballavstand, imidlertid vises en forbedring av treffsikkerheten ved korreksjon for misvisningen i modellen.

Modellen kan videreutvikles i forhold til en mer presis svingarm, og en mer avansert posisjoneringsalgoritme for enda bedre presisjon og nøyaktighet i systemet. For lange ballavstander kan det også vurderes om roboten skal bruke to kalkulerte slag for å treffe målballen. For en slik metode må første slagets styrke tas i betraktning for å plassere slagballen nærmest mulig målballen. Deretter vil det andre slaget bli en enkel prosedyre for roboten.

Bibliografi

[1] www.mindstorms.rwth-aachen.de

[2] Stein Tore Stegen, *LegoRobot*, 2011.

[3] www.mathworks.com

[4] Gonzales,Woods,Eddins, *Digital Image Processing using MATLAB*, 2004.

[5] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms,"*IEEE Transactions on Systems, Man, & Cybernetics*, 9(1), 1979, pp.62-66"

[6] www.mindstorms.lego.com

[7] "http://en.wikipedia.org/wiki/File:AdditiveColor.svg", 2007-04-06

[8] H. E. Burdick, McGraw-Hill, *Digital Imaging: Theory and Applications*, 1997

[9] Charles A. Poynton, Morgan Kaufmann. *Digital Video and HDTV: Algorithms and Interfaces.* 2003.

[10] Rangachar Kasturi, Ramesh C. Jain *Computer vision: Principles*, 1991.

[11] Gonzales, Wintz, *Digital Image Processing*, 1977.

[12] Gonzales, Wintz, *Digital Image Processing, second edition*, 1987.

[13] B.K.P.Horn, R.W.Sjoberg, "Calculating the Reflectance Map"*Applied Optics*, 18(11), 1979, pp.1770-1779"

[14] B.K.P.Horn *Robot Vision*, McGraw-Hill, New York, N.Y., 1986.

[15] B.K.P.Horn, Obtaining shape from shading information, in: Winston, P. H., (Ed), *The Psychology og Machine Vision*, McGraw-Hill, New York, N.Y., 1975, pp.115-155

[16] "http://prosjekt.ffi.no/unik-4660/lectures04/chapters/Introduction.html#Projections"

[17] Finn Haugen, *Praktisk reguleringsteknikk, 2. utg.*, Trondheim, 2003.

[18] Lorentzen, Hole, Lindstrøm *Kalkulus med en og flere variabler*, 2003

[19] www.mathworks.se/help/techdoc/

[20] A very low cost distrubuted localization and navigation system for a mobile robot.

Vedleggsliste

Tillegg A

Fra forprosjekt

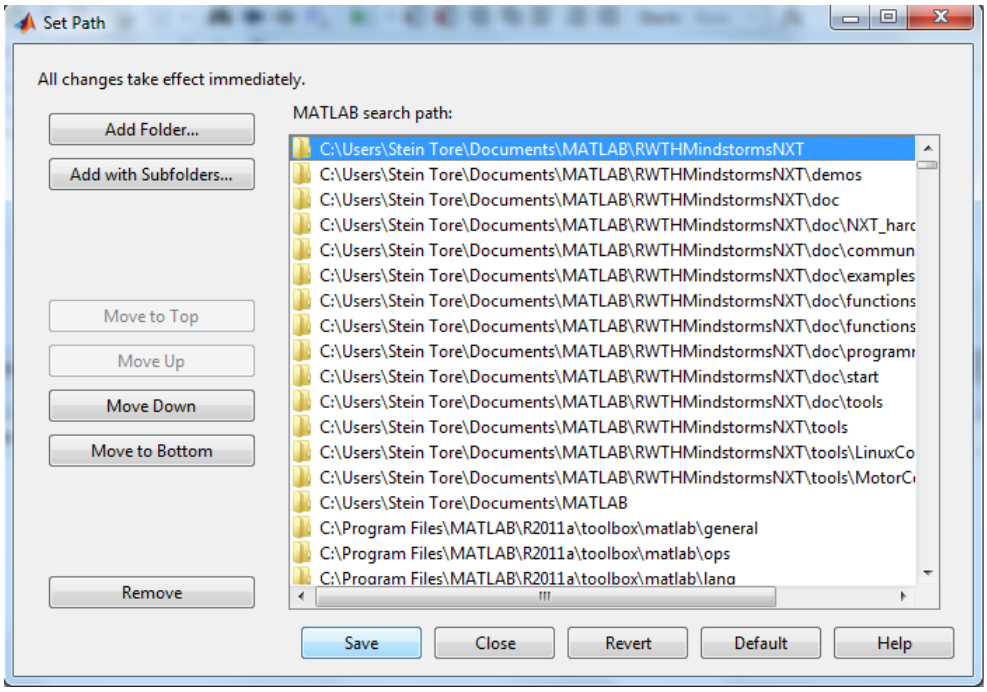
A.1 Oppsett av kommunikasjon

Dette delkapittelet forklarer i detalj oppsett av kommunikasjonen mellom webkamera, MATLAB og NXT. Instruksene er et utdrag fra den mer omfattende installasjonsmanualen til Lego Mindstorm NXT [1], med unntak av kapittel A.1.2 som er basert på eksempel fra MATLAB sin *image aquisition toolbox*. Prosedyre for oppsett avhenger av maskinvare og software, derfor anbefales installasjonsmanualen til Lego Mindstorms NXT ved problemer. Instruksene som beskrives her tilfredsstiller oppsett av kommunikasjon for modellen i oppgaven.

A.1.1 Nedlasting av verktøykasse til MATLAB

Lego mindstorm sin MATLAB verktøykasse ligger på nettet som åpen kilde, og ble lastet ned fra [1]. For nedlasting fra nettsiden, klikk på "download" pilen oppe i midten og velg deretter fra listen versjon 4.04. Denne versjonen ble valgt på grunn av god stabilitet. For installasjon følg denne prosedyren:

1. Pakk Zip-filene ut i en mappe, kall denne RWTHMindstormsNXT.
2. Oppdatere søkestien i MATLAB til å inkludere den nye mappen. I MATLAB, trykk på "file" og deretter "set path", figur A.1 kommer opp på skjermen. Press "add with subfolder", lokaliser stien til den nye mappen RWTHMindstormsNXT, klikk "ok" og deretter "save" for å lagre den nye stien.



Figur A.1: Valg av MATLAB sin søkesti.

A.1.2 Initialisering av webcamera

Kameraet tilkobles PC via USB kabel. Før kameraet kan tas i bruk i MATLAB må det initialiseres. Følgende eksempel for initialisering av enhet ble brukt:

1. I MATLAB, klikk på "help", og deretter på "produkt help" i rullegardinen.
2. Under "Contents", klikk på "Image Acquisition Toolbox", deretter "Getting started" og til slutt "Basic Image Acquisition Procedure".
3. Konfigurer kameraet for bruk i MATLAB med utførelse av de sju stegene i prosedyren. Kameraet er nå klart til anvendelse av både bilde og video i MATLAB.

Figur øverst på neste side viser kommandoer som ble brukt for blant annet initialisering av kameraet.

```
1
2 vid = videoinput('winvideo',2,'YUY2_640x360');
3 set(vid,'ReturnedColorSpace','rgb');
4 start(vid);
5 I=getsnapshot(vid);
6 stop(vid);
7 delete(vid)
8 clear all;
```

Initialisering av kamera skjer i linje 2 der et video objekt (vid) opprettes i MATLAB. Linje 3 setter hvilket fargekart som skal returneres fra kameraet. Kameraet støtter vanligvis YUY2 fargekart mens RGB er det ønskelige ved bildebehandling. Linje 4-8 starter video objektet, returnerer et enkeltbilde og rydder opp etterpå.

A.1.3 Oppdatere NXT/motorkontroll

Firmware til Lego Mindstorms NXT må være av versjon 1.26 eller høyere (anbefalt er 1.29). Oppdater firmware fra http://mindstorms.lego.com/en-us/support/files/Firmware.aspx dersom nødvendig [1] Servomotorene kontrolleres av et motorkontrollprogram i NXT'en. Toolbox versjoner til MATLAB nyere enn 2.01 kommer med dette programmet. Programmet mottar løpende "live" kommandoer fra MATLAB, og er operativt mens MATLAB utfører beregninger. Ved mottagelse av en kommando tar kontrollprogrammet over ansvaret for styring av motorene, mens MATLAB fortsetter på nye beregninger. MotorControll 2.2 er siste versjon og leveres med verktøykasse versjon 4.04.

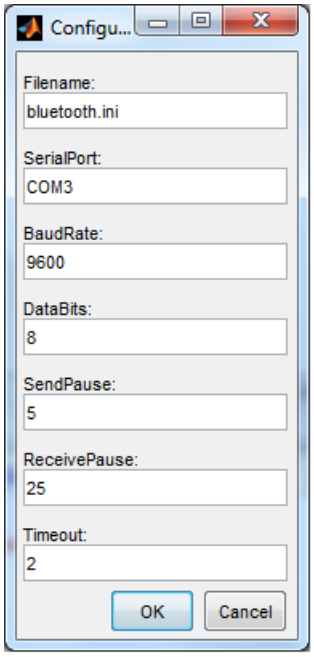
For å laste motorkontrollprogrammet over til NXT'en brukes NeXTTools, et program som lar bruker kommunisere med NXT brikken direkte. Lego Mindstorm NXT driver(fantom) må installeres før bruk av NeXTTools. Programmet støtter funksjoner som nedlasting av NXT firmware og ned- og opplasting av filer til NXT.
Motorkontrollprogrammet ble overført til NXT slik:

1. Last ned og installer driver for NeXTTool programmet fra http://mindstorms.lego.com/en-us/support/files/Driver.aspx
2. Last ned NeXTTool.exe fra http://bricxcc.sourceforge.net/utilities.html og lagre programmet i mappen /tools/MotorControl (undermappe i toolbox).
3. Bruk NeXTTool til å laste ned MotorControl.rxe til NXT. I windows, start TransferMotorControlBinaryToNXT.bat og følg instrukser på skjermen.

A.1.4 Kommunikasjon med blåtann

Prosedyre for oppsett av blåtann forbindelsen mellom NXT og MATLAB for første gang:

1. I MATLAB Command Window, kall opp metoden COM_MakeBTConfigFile
2. Klikk ja i første vindu som popper opp, og lagre konfigfilen i egnet mappe.
3. Nå bør figur A.2 vises på skjermen.



Figur A.2: Konfigurering av initialiseringsfilen til blåtann kommunikasjonen i modellen. I vinduet settes parametre for konfigurering av blåtann forbindelsen. Den viktigste parameter "seriel port". Denne må stemme overens med port som blåtann driveren bruker.

4. Slå på NXT og sørg for at Bluetooth funksjonen er slått på.
5. Søk etter bluetooth enheter fra PC og legg til NXT.
6. I oppkoblingsprosessen mellom PC og NXT må enhetene bekrefte hverandres identitet med en passordnøkkel. Et vindu med passordnøkkel skal dukke opp på skjermen. Bekreft oppkoblingen ved å taste inn identisk passord i både PC og NXT.
7. I NXT display, sjekk at blåtann ikonet oppe i venstre hjørne forandres fra B< til B< >. Om det ikke går, naviger deg fram i NXT menyen til blåtann, search, select your computer og connect. Symbolet skal endre

status, visst ikke så er kanskje ikke blåtann hardwaren kompatibel eller det trengs en annen driver. Sjekk i tilfelle http://www.mindstorms.rwth-aachen.de/trac/wiki/BluetoothAdapter for mer informasjon om drivere og blåtann hardware.

8. Dersom symbolet i NXT nå viser B< > så er kontakten mellom enheten opprettet, og kommunikasjonen initialiseres med følgende kommandoer i MATLAB.

```
1 COM_CloseNXT('all');
2 h=COM_OpenNXT('bluetooth.ini');
3 COM_SetDefaultNXT(h);
```

9. Dersom punkt 8 fullføres uten feilmeldinger så er forbindelsen initialisert, og NXT'en er mottagelig for instrukser i MATLAB via bluetooth.

A.1.5 USB forbindelse

Kommunikasjonen mellom PC og NXT skal skje via blåtann, men det er i denne sammenheng verdt å nevne at NXT'en også kan kommunisere via USB kabel. USB forbindelsen ble testet mellom NXT'en og PC med 64-bit operativsystem. Dette viste seg å ikke fungere som det skulle da NXT driveren (fantom) for USB forbindelsen ikke støtter 64-bit Windows system [1]. Videre søk etter alternative drivere ble ikke prioritert da kommunikasjonen i skal gå via blåtann.

For PC med 32-bit Windows system opprettes kommunikasjon med følgende prosedyre.

1. NXT Fantom driver må være installert. Kan lastes ned fra http://mindstorms.lego.com/en-us/support/files/Driver.aspx.
2. Koble USB kabel til NXT og PC.
3. I MATLAB i Command Window, skriv COM_OpenNXT. Dersom ingen feilmeldinger oppstår så er forbindelsen opprettet.

Tillegg B

M-filer

Liste over m-filer som brukes ved implementasjon av systemet.

```
%Main.
 clear all
 close all

%Kjører initialiseringsfil ved 1.gangs implementering etter oppstart.
%Oppretter BT forbindelse med robot og initialiserer webkamera.
 Init();
 slagnr = 1;
 status = 0;

 while(status == 0)

%Henter bilde fra kamera.
 I = getsnapshot(vid);

%Beregner koordinat og retning for baller og robot.
 [blue,red,senterROI,start_x,start_y,slutt_x,slutt_y] = BallKoord(I);

%Alternativt: Sjekker om ballene ligger i et tilgjengelig område for roboten.
%[blue,red] = Ballsjekk(blue,red,start_x,start_y,slutt_x,slutt_y,vid);

%Avslutter programmet dersom baller eller robot ikke er detektert av koden.
 if(blue(1) == 0 || red(1) == 0)
     return;
 end

 [robotkoord,bluemarkkoord,redmarkkoord] = RobotKord(I);
 [alfa,nevner] = RobotRetning(bluemarkkoord,redmarkkoord);

%Korrigerer robotkoordinat ift. feilmargin ved projeksjon inn i bildeplanet.
 [robotkoord] = KorrKoord(robotkoord,senterROI);

%Beregner viapunkt (maalkoord) som roboten posisjonerer seg i først.
 [ballkoord,maalkoord,treffkoord,fargetest] = DestPunkt(blue,red,robotkoord);

%Finner vinkel mellom robot og viapunkt, snur robot og kjører til viapunkt.
 betta = RetnRobotTilBall(maalkoord,robotkoord);
 NXT_PlaySoundFile('MS__voice_Track', 0);
 pause(1)
 SnurRobotMotBall(alfa,betta,nevner,maalkoord,vid,senterROI);
 Robdata = KjorTilPunkt(maalkoord,robotkoord,vid);

%I viapunkt,tar nytt bilde, beregner ny robot pos. og retning.
 I = getsnapshot(vid);
 [robotkoord,bluemarkkoord,redmarkkoord] = RobotKord(I);
 [alfa,nevner] = RobotRetning(bluemarkkoord,redmarkkoord);
 [robotkoord] = KorrKoord(robotkoord,senterROI);

%Beregner ny retning robot til ball, snur mot ball.
 betta = RetnRobotTilBall(ballkoord,robotkoord);
 alfa = SnurRobotMotBall(alfa,betta,nevner,ballkoord,vid,senterROI);

%Hever svingarm bakover til slagposisjon og kjører til ballposisjon.
 retning = HevArm(blue,red,alfa,ballkoord);
```

```
 Robdata2 = KjorTilBall(ballkoord,robotkoord,vid);

 %Slår ballen.
 Balldata = SlaBall(retning,vid);
 pause(1)

%Sjekker om ballen traff målet sitt.
 [slagnr,status] = TreffSjekk(treffkoord,fargetest,vid,slagnr);

%Fjerner gamle bilder fra minnet.
 flushdata(vid)

 end

%Lager plot av loggede bildedata.
%PlotPosisjonering(Robdata,Robdata2,Balldata)
```

```
%Initialiseringfil.
%Kommandoer i MATLAB som oppretter kommunikasjonen mellom PC og NXT via
%bluetooth. Bluetooth.ini filen må være initialisert på forhånd. Se
%kapittel 2.5.4 i forprosjektrapporten for detaljer.
 COM_CloseNXT('all');
 close all
 clear all
 h = COM_OpenNXT('bluetooth.ini');
 COM_SetDefaultNXT(h);

%Initialserer kamera.
 vid = videoinput('winvideo',2,'YUY2_1280x800');
 set(vid,'ReturnedColorSpace','rgb');
 triggerconfig(vid, 'manual')
 set(vid,'FramesPerTrigger',50)
 pause(3)
```

```
%Metode som finner ballenes koordinater i bildeplanet.

function
[kord_blue_ball,kord_red_ball,senterROI,start_x,start_y,slutt_x,slutt_y] =
BallKoord(I)

%Henter ROI automatisk fra funksjon.
 [red,blue,senterROI,start_x,start_y,slutt_x,slutt_y] = ROI(I);

%Binære bilder fra fargekomponentbildene.
 bw_blue = im2bw(blue,0.9);
 bw_red = im2bw(red,0.9);

%Bruker morfologi(opening) for å fjerne små objekter (mindre enn 500
%piksler) og støy, fyller igjen små huller i roboten.
 bw_blue = bwareaopen(bw_blue,500);
 bw_red = bwareaopen(bw_red,500);
 bw_red = imfill(bw_red,'holes');
 bw_blue = imfill(bw_blue,'holes');

%Lokaliserer ballenes posisjon med bruk av regionprops, sorterer objektene
%etter areal og velger info om den minste som skulle tilsi en ball.
 s_blue = regionprops(bw_blue,'centroid','area');
 s_red = regionprops(bw_red,'centroid','area');
 AntObjBlue = bwlabel(bw_blue);
 AntObjRed = bwlabel(bw_red);

%Sjekker at det finnes noen objekter, finner evt. ballobjektene som skulle
%tilsvare de minste objektene og deretter objektenes koordinater.
 if(AntObjBlue <= 1 || AntObjRed <= 1)

    disp('Tilstrekkelig antall objekter ble ikke detektert. Sjekk at romlys
er påslått, og at begge ballene plus roboten er tilgjengelig og prøv
igjen.');
  %Sender udefinerte koordinater tilbake til metodekallet.
    kord_blue_ball = [0,0];
    kord_red_ball = [0,0];
    return
 end

  minval_b = s_blue(1).Area;
  minval_r = s_red(1).Area;

for i = 1:AntObjBlue;
    if(s_blue(i).Area <= minval_b)
        label_b = i;
        minval_b = s_blue(i).Area;
    end

end

for i = 1:AntObjRed;
    if(s_red(i).Area <= minval_r)
        label_r = i;
        minval_r = s_red(i).Area;
```

```
    end

end

 kord_blue_ball = s_blue(label_b).Centroid;
 kord_red_ball = s_red(label_r).Centroid;

end
```

```
%Lager metode som trekker ut ROI automatisk fra originalbildet.

function [red,blue,green,senterROI,start_x,start_y,slutt_x,slutt_y] = ROI(I)
%Henter ut fargekomponentene i bildet.
 red = I(1:end,1:end,1);
 blue = I(1:end,1:end,3);
 green = I(1:end,1:end,2);
%Lager masken som brukes til definisjon av ROI.
 BW = roicolor(red,40,100);
 BW = imfill(BW,'holes');
 BW = bwareaopen(BW,10000);

%Henter info om avgrensing til masken med regionprops.
 s = regionprops(BW,'BoundingBox');
 start_x = s.BoundingBox(1);
 start_y = s.BoundingBox(uint8(2));
 lengde_x = s.BoundingBox(uint8(3));
 lengde_y = s.BoundingBox(uint8(4));
 slutt_y = start_y + lengde_y;
 slutt_x = start_x + lengde_x;

%Runder av til heltall for indeksbruk.
 start_x = round(start_x);
 start_y = round(start_y);
 slutt_x = round(slutt_x);
 slutt_y = round(slutt_y);

  if(slutt_y > 800)
     slutt_y = 800;
  end
  if(slutt_x > 1280)
     slutt_x = 1280;
  end

 red = red(start_y : slutt_y, start_x : slutt_x);
 blue = blue(start_y : slutt_y, start_x: slutt_x);
 green = green(start_y : slutt_y, start_x: slutt_x);

 %Definerer senterpunkt i bildeplanet.
 senterROI(1) = (slutt_x - start_x) / 2;
 senterROI(2) = (slutt_y - start_y) / 2;
end
```

```
%Metode som sjekker om ballen ligger i område der det er fysisk mulig for
%roboten å utføre et slag.
function[blue,red] = Ballsjekk(blue, red,start_x,start_y,slutt_x,slutt_y,vid)

nedre_x = 0;
ovre_x = slutt_x - start_x;
nedre_y = 0;
ovre_y = slutt_y - start_y;
OB_nedre_x = nedre_x + 100;
OB_ovre_x = ovre_x - 100;
OB_nedre_y = nedre_y + 100;
OB_ovre_y = ovre_y - 100;

while(blue(1) < OB_nedre_x || blue(1) > OB_ovre_x || blue(2) < OB_nedre_y ||
blue(2) > OB_ovre_y)

    input('Blå ball utenfor rekkevidde for slag. Plasser ball innenfor
stiplet linje og trykk enter.','s');
    I = getsnapshot(vid);
    [blue,red] = BallKoord(I);

end

while(red(1) < OB_nedre_x || red(1) > OB_ovre_x || red(2) < OB_nedre_y ||
red(2) > OB_ovre_y)

    input('Rød ball utenfor rekkevidde for slag. Plasser ball innenfor
stiplet linje og trykk enter.','s');
    I = getsnapshot(vid);
    [blue,red] = BallKoord(I);

end
%Begge baller er godkjennt.
    disp('Begge baller innenfor rekkevidde. Oppdraget fortsetter.')
end
```

```
%Henter ut robotens koordinat og retning i koordinatsystemet for bruk ved
%lokalisering av ball.

function [robotkoord,kord_blue,kord_red] = RobotKord(I)

%Henter ut ROI automatisk fra maskefunksjon.
 [red,blue] = ROI(I);

%Finner binært bilde av hhv. rød og blå RGB komponent.
 bw_blue = im2bw(blue,0.9);
 bw_red = im2bw(red,0.95);

%Bruker morfologi(opening) for å fjerne små objekter (mindre enn 85/90
%piksler) og støy, fyller igjen små huller i roboten.
 bw_blue = bwareaopen(bw_blue,75);
 bw_red = bwareaopen(bw_red,105);
 bw_red = imfill(bw_red,'holes');
 bw_blue = imfill(bw_blue,'holes');

%Lokaliserer kordinat til robotens markører med bruk av regionprops, henter
%info om areal og midtkoordinat til objektene.
 s_blue = regionprops(bw_blue,'centroid','area');
 s_red = regionprops(bw_red,'centroid','area');
 AntObjBlue = bwlabel(bw_blue);
 AntObjRed = bwlabel(bw_red);

%Søker gjennom alle objektene og beholder den med minst areal som skulle
%forhåpentligvis tilsvare en rød og blå markør og henter ut deres
koordinater.
 minvalue_b = s_blue(1).Area;
 minvalue_r = s_red(1).Area;
for i = 1:AntObjBlue;
    if(s_blue(i).Area <= minvalue_b)
        labelb = i;
        minvalue_b = s_blue(i).Area;
    end

end
for j = 1:AntObjRed;
    if(s_red(j).Area <= minvalue_r)
        labelr = j;
        minvalue_r = s_red(j).Area;
    end

end
%Henter koordinatene til minste objekt for rød og blå.
 kord_blue = s_blue(labelb).Centroid;
 kord_red = s_red(labelr).Centroid;

%Grenser for avstand mellom markører i piksler.
 nedre_grense = 67;
 ovre_grense = 73;

%Henter avstanden mellom de to koordinatene.
 avstand = dist_betw_pixels(kord_blue,kord_red);
```

```
 if (avstand < nedre_grense || avstand > ovre_grense)
     for j2 = 1:AntObjRed;
         for i2 = 1:AntObjBlue;
             kord_blueint = s_blue(i2).Centroid;
             kord_redint = s_red(j2).Centroid;
             avstand = dist_betw_pixels(kord_blueint,kord_redint);
              if ((avstand > nedre_grense && avstand < ovre_grense) &&
(s_blue(i2).Area < 150 && s_red(j2).Area < 150))
                 kord_blue = s_blue(i2).Centroid;
                 kord_red = s_red(j2).Centroid;
              end
         end
      end
  end

%Beregner x og y koordinat for robotens senterpunkt.
 Xrobot_centerkord = (kord_red(1) + kord_blue(1)) / 2;
 Yrobot_centerkord = (kord_red(2)+ kord_blue(2)) / 2;
 robotkoord = [Xrobot_centerkord Yrobot_centerkord];
 end
```

```
%Beregner robotens retning i systemet utfra beregnede markørkoordinater.
function [alfa,nevner] = RobotRetning(kord_blue,kord_red)
%Henter ut x og y kordinater for markørpunktene beregnet i RobotKord.
 Xkord_red = kord_red(1);
 Ykord_red = kord_red(2);
 Xkord_blue = kord_blue(1);
 Ykord_blue = kord_blue(2);

%Beregner robotens vinkel alfa.
 teller = Ykord_red - Ykord_blue;
 nevner = Xkord_red - Xkord_blue;
 alfa = atand(teller / nevner);

%Korrigerer for at vinkel beregnet fra markørpunktene er 90 grader
%forskjøvet i forhold til robotens kjøreretning.
 alfa = alfa - 90;

%Korrigerer i forhold til hvilken kvadrant retning roboten befinner seg i.
 if( nevner < 0)
     % 3. og 4. kvadrant.
      alfa = alfa + 180;
 else
     % 1. og 2. kvadrant.
      alfa = alfa + 360;
 end
end
```

```
%Metode som skal korrigere for projeksjonsavviket.
function [robotkoord] = KorrKoord(robotkoord,senterROI)

%Beregner avstand fra senter til beregnet robotkoord. i hhv. x- og
%y-retning.
 x_avst = abs(senterROI(1) - robotkoord(1));
 y_avst = abs(senterROI(2) - robotkoord(2));

% Tall etter utledning av feilmargin.
 x_avvik = 0.096 * x_avst;
 y_avvik = 0.096 * y_avst;

%Korrigerer for avvik etter hvilken kvadrant markørkoordinatet finnes.
%1. kvadrant.
 if(robotkoord(1) > senterROI(1) && robotkoord(2) <= senterROI(2))
  robotkoord(1) = robotkoord(1) - x_avvik;
  robotkoord(2) = robotkoord(2) + y_avvik;
 end
%2. kvadrant.
 if(robotkoord(1) <= senterROI(1) && robotkoord(2) < senterROI(2))
  robotkoord(1) = robotkoord(1) + x_avvik;
  robotkoord(2) = robotkoord(2) + y_avvik;
 end
%3. kvadrant.
 if(robotkoord(1) < senterROI(1) && robotkoord(2) >= senterROI(2))
  robotkoord(1) = robotkoord(1) + x_avvik;
  robotkoord(2) = robotkoord(2) - y_avvik;
 end
%4. kvadrant.
 if(robotkoord(1) >= senterROI(1) && robotkoord(2) > senterROI(2))
  robotkoord(1) = robotkoord(1) - x_avvik;
  robotkoord(2) = robotkoord(2) - y_avvik;
 end
end
```

```
%Rutine som beregner viapunkt som robot skal posisjonere seg i.

function [ballkoord,maalkoord,treffkoord,fargetest] =
DestPunkt(blue,red,robotkoord)

%Beregner hvilken ball som ligger nærmest roboten og som skal slås.
 dist_blue = dist_betw_pixels(blue,robotkoord);
 dist_red = dist_betw_pixels(red,robotkoord);

 if(dist_blue < dist_red)
    ballkoord = blue;
    treffkoord = red;
    farge = 1;
   %Bruker boolsk variabel for å merke hvilken farge treffkoord tilhører.
    fargetest = logical(farge);
 else
    ballkoord = red;
    treffkoord = blue;
    farge = 0;
    fargetest = logical(farge);
 end

%Beregner vinkel mellom ballene og korrigerer for hvilken kvadrant en
%befinner seg i.
 teller = blue(2) - red(2);
 nevner = blue(1) - red(1);

%Tilfellet for 4. kvadrant.
 delta = atand(teller / nevner);

%Sjekker om retning mellom ballene ligger i 1., 2. eller 3. kvadrant.
%Korigerer eventuelt.
  if(nevner > 0 && teller < 0)
       %1.kvadrant.
        delta = delta + 360;
  else if(nevner < 0)
       %2. og 3. kvadrant.
        delta = delta + 180;
       end
  end

%Har funnet vinkelen mellom ballene [0 360], beregner videre 2 punkter
roboten kan
%treffe ballen ifra. Definerer ny vinkel gamma [0 90]som brukes ved beregning
av
%de 2 punktene.
     gamma = delta;
  if( gamma > 90 && gamma >= 270)
     gamma = abs(gamma - 180);
  end
  if( gamma > 270)
     gamma = abs(gamma - 360);
  end

%Definerer radius fra ball til viapunktet.
```

```
 radius = 145;

%Beregner x og y komponenter med gamma vinkel.
 x = sind(gamma) * radius;
 y = cosd(gamma) * radius;

%Beregner koordinater for de to punktene.
 p1 = ballkoord + [-x y];
 p2 = ballkoord + [x -y];

%Finner det punkt med kortest avstand til roboten.
   avstand = [dist_betw_pixels(p1,robotkoord)
dist_betw_pixels(p2,robotkoord)];
  if(avstand(1) < avstand(2))
     maalkoord = p1;
  else
     maalkoord = p2;
  end
end
```

```
%Beregner vinkel betta mellom ball og robot som brukes for å snu roboten i
%kjøreretning mot ballen. Bruker ballkordinat og robotens senterkoordinat.

function [betta] = RetnRobotTilBall(kord_maalpunkt,robotkoord)
 teller2 = kord_maalpunkt(2) - robotkoord(2);
 nevner2 = kord_maalpunkt(1) - robotkoord(1);

%Tilfellet for 4. kvadrant.
 betta = atand(teller2 / nevner2);

%Sjekker om retning mellom ball og robot ligger i1., 2. eller 3. kvadrant.
%Korigerer eventuelt.
  if(nevner2 > 0 && teller2 < 0)
       %1.kvadrant.
        betta = betta + 360;
  else if(nevner2 < 0)
       %2. og 3. kvadrant.
        betta = betta + 180;
       end
  end
end
```

```
%Rutine som snur robot til retning mot viapunkt.
function [alfa] =
SnurRobotMotBall(alfa,betta,nevner,sluttkoord,vid,senterROI)

%Beregner tachometerverdien som bestemmer servoens grad av rotasjon.
  diff_vinkler = abs(alfa - betta); %Reg.avvik

  if(diff_vinkler > 180)
    diff_vinkler = abs(diff_vinkler - 360);
  end

 tachom = round(diff_vinkler * 2.15); %Skaleringsforholdet mellom tachoverdi
og antall
                                      %grader rotasjon. Avrunder verdi til
nærmeste integer.

%Grovjusterer først retning i forhold til vinkel.
%Dersom 3. eller 4. kvadrant, nevner < 0.
 if(nevner < 0)
    if(betta <( alfa + 180) && betta > alfa)
         %Svinger til høyre.
          mSvingH = NXTMotor('A','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tachom);
          mSvingV =
NXTMotor('B','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
om);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    else %Svinger venstre.
          mSvingH =
NXTMotor('A','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
om);
          mSvingV = NXTMotor('B','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tachom);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    end
 end

 %Dersom 1. eller 2. kvadrant.
 if(nevner > 0)
    if( betta > (alfa - 180) && betta < alfa)
         %Svinger venstre.
          mSvingH =
NXTMotor('A','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
om);
          mSvingV = NXTMotor('B','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tachom);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
```

```
    else %Svinger høyre.
          mSvingH = NXTMotor('A','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tachom);
          mSvingV =
NXTMotor('B','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
om);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    end
 end

%Tar nytt bilde og beregner ny robotretning og retning til ball.
  I = getsnapshot(vid);
  [robotkoord,bluemarkkoord,redmarkkoord] = RobotKord(I);
  [robotkoord] = KorrKoord(robotkoord,senterROI);
  [alfa,nevner] = RobotRetning(bluemarkkoord,redmarkkoord);
  betta = RetnRobotTilBall(sluttkoord,robotkoord);

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% P-regulator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Finjusterer det siste avviket med en p-reg tbk.
while(abs(alfa - betta) > 1)

%Definerer reguleringsavviket som forskjellen mellom vinklene.
    e = abs(alfa - betta);

%Avgjør retning
    tacholimit = round(e * 2.15);

  if(nevner < 0)
    if( betta <( alfa + 180) && betta > alfa)
         %Svinger til høyre.
          mSvingH = NXTMotor('A','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tacholimit);
          mSvingV =
NXTMotor('B','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
olimit);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    else %Svinger venstre.
          mSvingH =
NXTMotor('A','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
olimit);
          mSvingV = NXTMotor('B','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tacholimit);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    end
  end
```

```
 %Dersom 1. eller 2. kvadrant.
  if(nevner > 0)
    if( betta > (alfa - 180) && betta < alfa)
         %Svinger venstre.
          mSvingH =
NXTMotor('A','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
olimit);
          mSvingV = NXTMotor('B','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tacholimit);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    else %Svinger høyre.
          mSvingH = NXTMotor('A','Power',-
40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tacholimit);
          mSvingV =
NXTMotor('B','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tach
olimit);
          mSvingH.SendToNXT();
          mSvingV.SendToNXT();
          mSvingH.WaitFor();
          mSvingV.WaitFor();
    end
  end

%Tar nytt bilde og beregner ny robotretning og retning til ball.
  I = getsnapshot(vid);
  [robotkoord,bluemarkkoord,redmarkkoord] = RobotKord(I);
  [robotkoord] = KorrKoord(robotkoord,senterROI);
  [alfa,nevner] = RobotRetning(bluemarkkoord,redmarkkoord);
  betta = RetnRobotTilBall(sluttkoord,robotkoord);

 end
end
```

```
%Rutine som kjører til viapunkt.
function [Robdata] = KjorTilPunkt(punktkoord,robotkoord,vid)

%Beregner avstand mellom robot og ball i pikselverdi.
 avstand = dist_betw_pixels(punktkoord,robotkoord);

%Beregner input til servo i forhold til avstand.
tacholimit = round(avstand * 2.32);

%Kjører frem.
 mHjul =
NXTMotor('AB','Power',40,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tac
holimit);
 mHjul.SendToNXT();
%For videologging.
%Robdata = Bildelogging(vid, 30);
 Robdata=1;
 mHjul.WaitFor();

 NXT_PlaySoundFile('MS__voice_Objec', 0);
 pause(0.3)
 end
```

```
%Hever svingarm til slagposisjon før roboten posisjonerer i slagposisjon helt
inntil
%ballen.

function[retning] = HevArm(blue,red,alfa,ballkoord)

%Gjelder dersom blå ball er def som ballkoord. Inverter for rød ball.
%Dersom roboten tar oppstilling ved ball med retning 3. eller 4. kvadrant.
 if(alfa < 180)
   if (red(1) < blue(1))
       power = -30;
   else
       power = 30;
   end
 end

%Dersom roboten tar oppstilling ved ball med retning 1. eller 2. kvadrant.
 if(alfa >= 180)
   if (red(1) > blue(1))
       power = -30;
   else
       power = 30;
   end
 end

%Avgjør om ballkoord stammer fra rød eller blå ball.Inverterer for rød ball.
 if(ballkoord == red)
    power = -power;
 end

%Hever køllen 45 grader.
 tacholimit = 40;

%Konstruerer motorobjektet mArm og setter egenskaper.
 mArm =
NXTMotor('C','Power',power,'SpeedRegulation',0,'TachoLimit',tacholimit);
 mArm.ResetPosition;
 mArm.SendToNXT();
 mArm.WaitFor();

%Sender retning tilbake til funksjonskallet.
 retning = power;
 end
```

```
%Plasserer robot inntil ball i slagposisjon.
function [Robdata2] = KjorTilBall(ballkoord,robotkoord,vid)

%Beregner avstand mellom robot og ball i pikselverdi.
 avstand = dist_betw_pixels(ballkoord,robotkoord);

%Offset mellom robotkoordinat og robotens fremste punkt (svingarmen).
 offset = 90;
 avstand = avstand - offset;

%Beregner input til servo i forhold til avstand.
 tacholimit = round(avstand * 2.32);

%Kjører frem.
 mHjul =
NXTMotor('AB','Power',30,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit',tac
holimit);
 mHjul.SendToNXT();
%For videologging.
 %Robdata2 = Bildelogging(vid, 30);
 Robdata2 = 1;
 mHjul.WaitFor();
 end
```

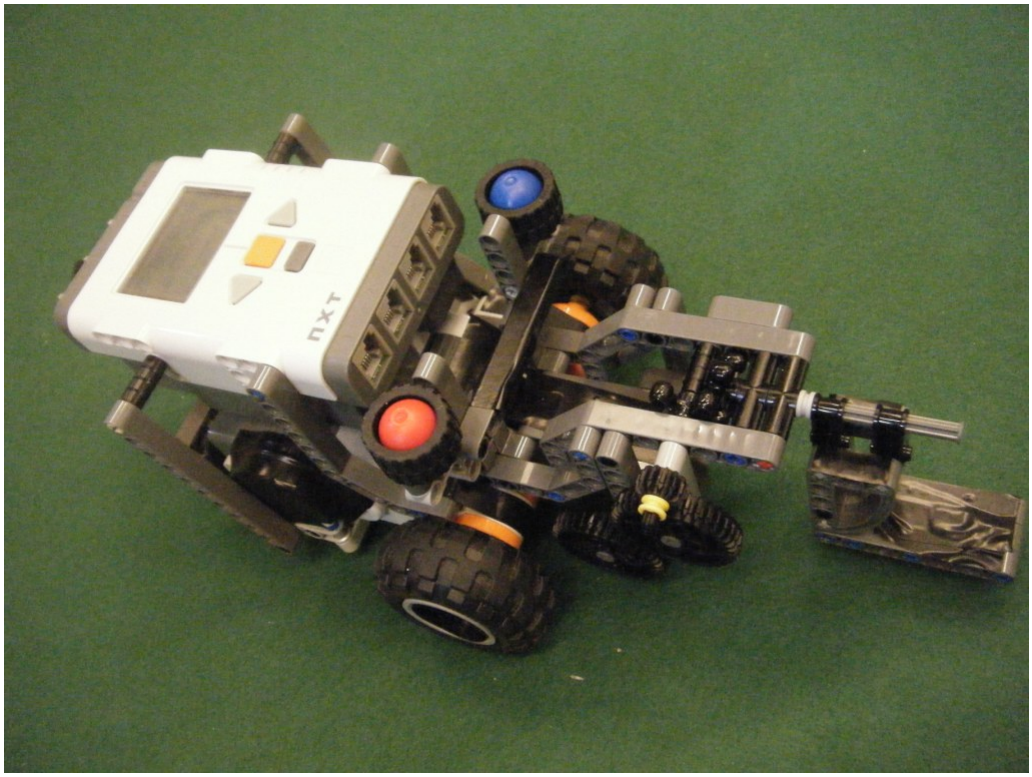
```
%Slår ballen.
 function[Balldata] = SlaBall(retning,vid)

 %Setter styrke på slaget og bestemmer i hvilken retning det skal utføres.
  if(retning > 0)
     power = 80;
  else
     power = -80;
  end

  NXT_PlaySoundFile('MS__voice_Shoot', 0);
  pause(0.5)
  mArm =
NXTMotor('C','Power',power,'SmoothStart',1,'SpeedRegulation',0,'TachoLimit'
,160);
  mArm.SendToNXT();
 %For videologging.
 %Balldata = Bildelogging(vid, 50);
  Balldata = 1;
  mArm.WaitFor();
  data = mArm.ReadFromNXT;
  posarm = data.Position;

  if(posarm > 0)
      kraft = -20;
  else
      kraft = 20;
  end

 %Senker arm tilbake til startposisjon.
  mArm =
NXTMotor('C','Power',kraft,'SmoothStart',1,'SpeedRegulation',1,'TachoLimit'
,abs(posarm));
  mArm.SendToNXT();
  mArm.WaitFor();
  mArm.ResetPosition;
end
```

```
%Metode som sjekker om roboten traff ballobjektet sitt med golfslaget.
%Metoden mottar koordinat for målball.
function[slagnr,suksess] = TreffSjekk (treffkoord,fargetest,vid,slagnr)

%Nytt bilde taes etter slaget for å sjekke om ballen traff målet sitt.
 I = getsnapshot(vid);

%Henter nye koordinater for ballene etter at slag.
[kord_blue_ball,kord_red_ball] = BallKoord(I);

%Tester hvilken ball en vil ha nytt koordinat for. Høy fargetest betyr at
%rød ball var den som skulle treffes (målball).
if (fargetest == 1)
    nyttreffkoord = kord_red_ball;
else nyttreffkoord = kord_blue_ball;
end

avstand = dist_betw_pixels( treffkoord,nyttreffkoord );

if(avstand < 8)
    disp('Mislykket slag, ball ble ikke truffet')
    suksess = 0;
    NXT_PlaySoundFile('MS__voice_Insan', 0);
    slagnr = slagnr + 1;
    pause(1)
else  sprintf('Hole in %d, ball truffet',slagnr)
    suksess = 1;
    NXT_PlaySoundFile('MS__voice_Got I', 0);

 end
end
```

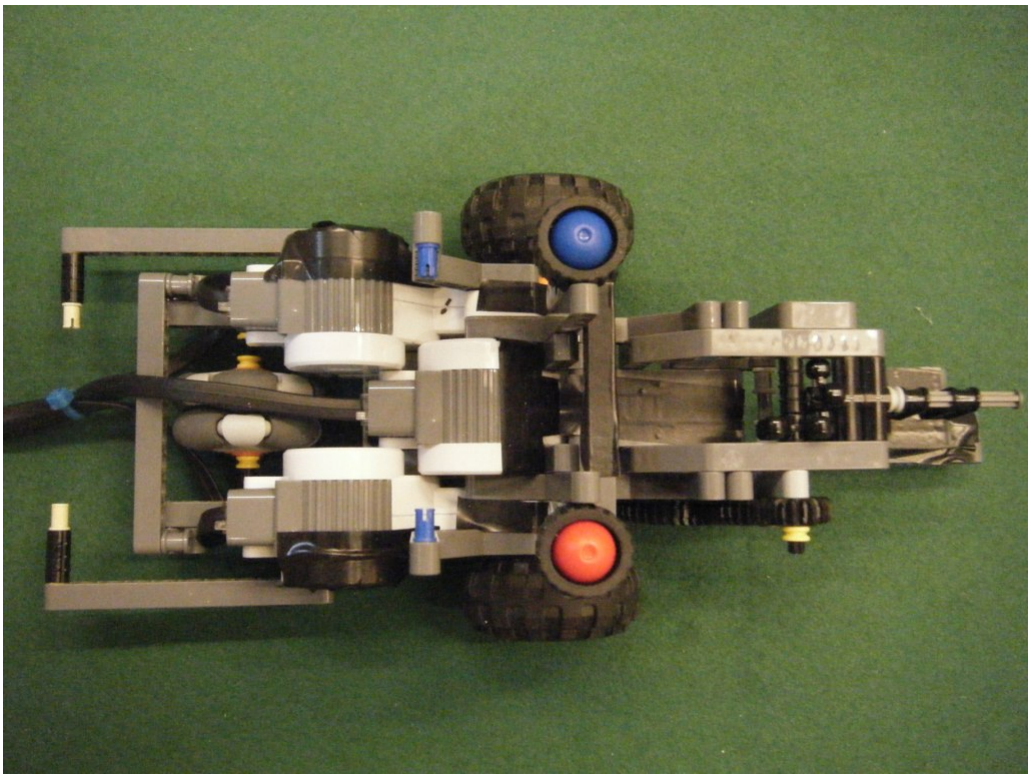
Tillegg C

Prosjektbilder

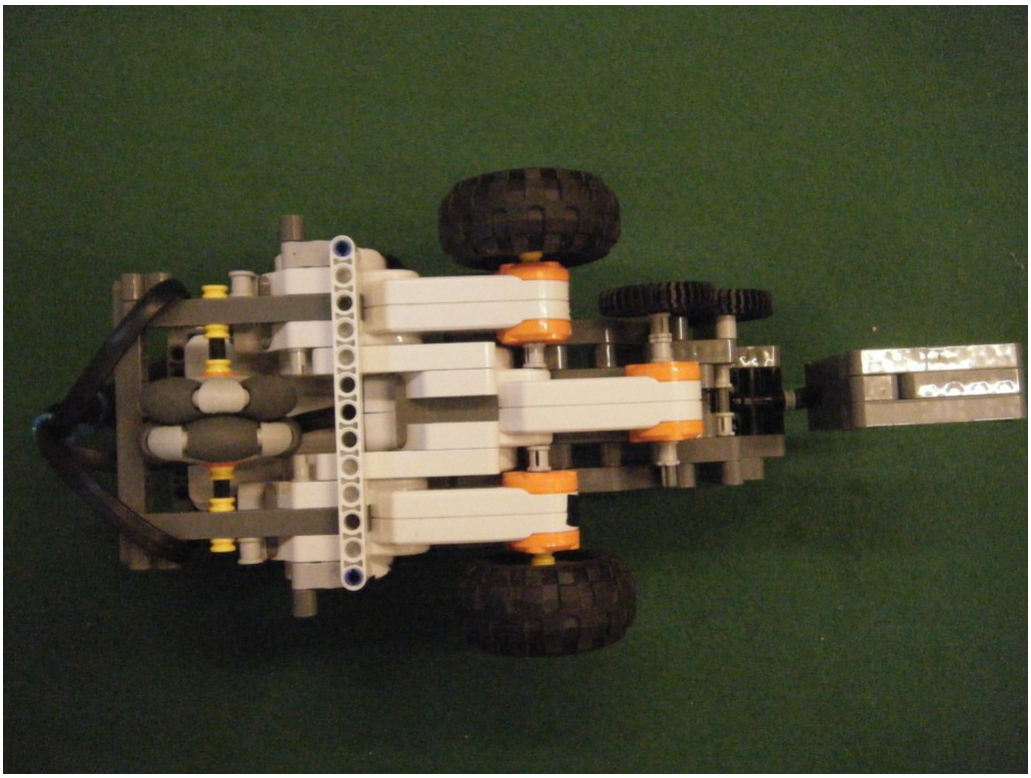
Bilder av den konstruerte roboten:



Figur C.1: Roboten sett skrått forfra.

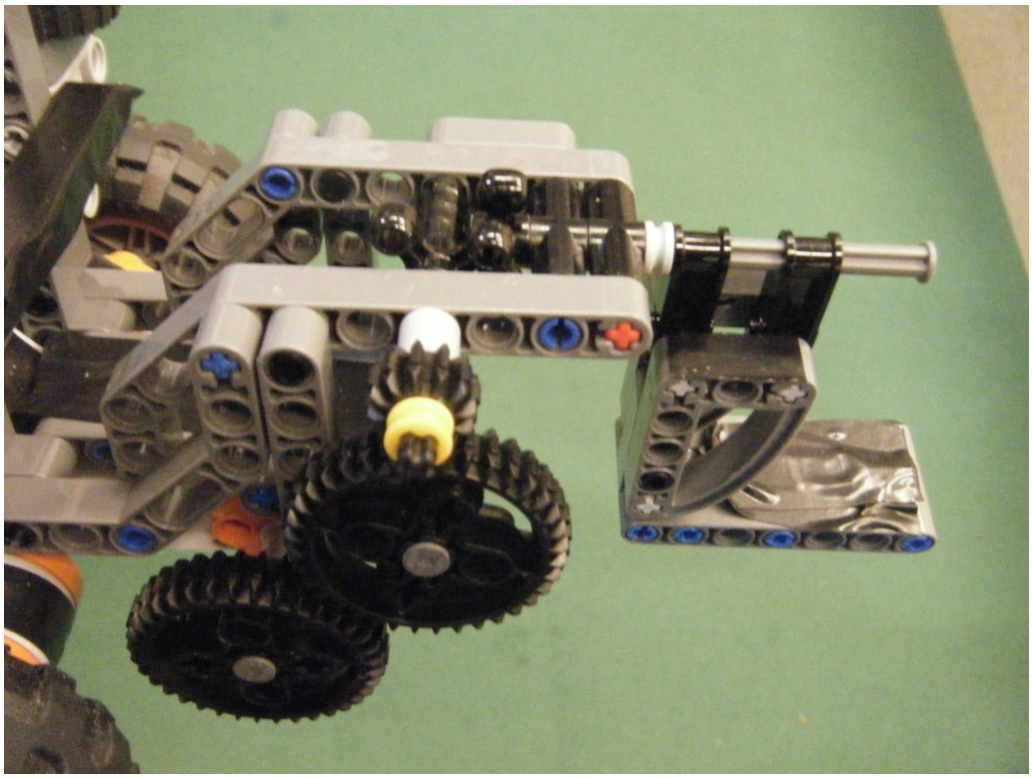


(a) Bilde av roboten ovenfra med NXT modulen demontert

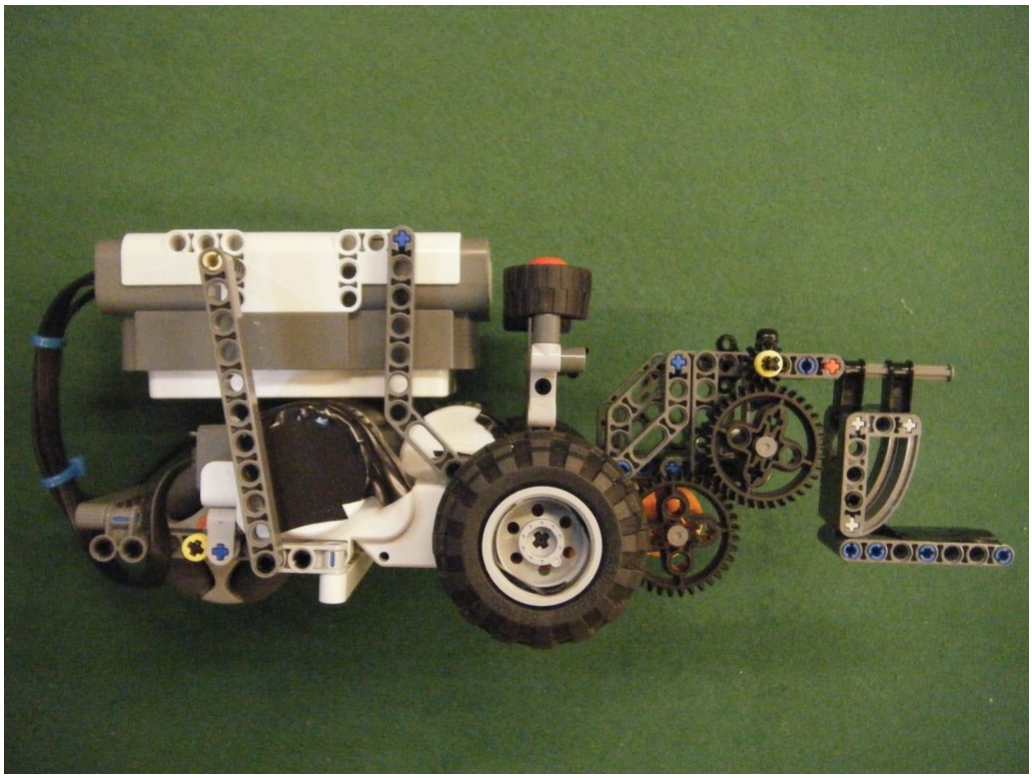


(b) Bilde av roboten sett fra undersiden

Figur C.2: Bilder av roboten



(a) Bilde av robotens girsystem



(b) Bilde av roboten sett fra siden

Figur C.3: Bilder av roboten

Tillegg D

CD materiell

Dette appendikset finnes på vedlagt CD bakerst i rapporten.

Tillegg E

Datablad

Microsoft® LifeCam Cinema™

Version Information	
Product Name	Microsoft® LifeCam Cinema™
Product Version	Microsoft LifeCam Cinema
Webcam Version	Microsoft LifeCam Cinema
Product Dimensions	
Webcam Length	2.20 inches (55.9 millimeters)
Webcam Width	1.81 inches (46.0 millimeters)
Webcam Depth/Height	1.58 inches (40.0 millimeters)
Webcam Weight	3.36 ounces (95.3 grams)
Webcam Cable Length	72.0 inches +6/-0 inches (1829 millimeters +152/-0 millimeters)
Compatibility and Localization	
Interface	Compatible with USB 2.0 High Speed specification
Operating Systems	Microsoft Windows® 7, Windows Vista®, and Windows XP Service Pack 2 or higher (excluding Windows XP 64-bit)
Top-line System Requirements	Requires a PC that meets the requirements for and has installed one of these operating systems: • Microsoft Windows 7, Windows Vista, or Windows XP Service Pack 2 or higher (excluding Windows XP 64-bit) • Intel Dual-Core 1.6 GHz (Intel Dual-Core 3.0 GHz recommended) • 1 GB of RAM (2 GB of RAM recommended) • 1.5 GB hard drive space • Windows-compatible speakers or headphones • USB 2.0 • Broadband internet access required, access fees may apply • CD-ROM • Microsoft LifeCam software version 3.0 Internet functions (post to Windows Live™ Spaces, sending e-mail, video calls), also require: Internet Explorer 6/7/8 browser software required for installation; 25 MB hard drive space typically required (users can maintain other default Web browsers after installation) Video call resolution dependent on messaging client. Basic Video & Audio Functionality with Windows Live Messenger, Microsoft Office Communicator, Yahoo! Messenger, AOL Instant Messenger, and Skype. Windows Live Movie Maker is only available for Windows 7 and Windows Vista users.
Compatibility Logos	• Compatible with Microsoft Windows 7 • Certified USB Logo
Software Localization	Microsoft LifeCam software version 3.0 may be installed in Simplified Chinese, Traditional Chinese, English, French, German, Italian, Japanese, Korean, Brazilian Portuguese, Iberian Portuguese, Russian, or Spanish. If available, standard setup will install the software in the default OS language. Otherwise, the English language version will be installed.
Windows Live™ Integration Features	
Video Conversation Feature	Windows Live call button delivers one touch access to video conversation
Call Button Life	10,000 actuations
Webcam Controls & Effects	LifeCam Dashboard provides access to animated video effects and webcam controls
Windows Live Integration Features	Windows Live Photo Gallery integration - Take a photo with LifeCam Software, then with one click open Photo Gallery to edit, tag and share it online Windows Live Movie Maker integration - Record a video with LifeCam Software and start a movie project on Movie Maker with just one click to then upload it to your favorite networking site
Imaging Features	
Sensor	CMOS sensor technology
Resolution	• Motion Video: 1280 x 720 pixels video • Still Image: Up to 5 megapixel (2880x1620 pixels, interpolated) photos*
Imaging Rate	Up to 30 frames per second
Field of View	73° diagonal field of view
Imaging Features	• Digital pan, tilt, and zoom • Auto focus, range from 6˝ to infinity • Automatic image adjustment with manual override
Audio Features	
Audio Features	Integrated uni-directional digital microphone
Frequency Response	200Hz to 8000Hz, +/- 4dB
Product Feature Performance	
Mounting Features	Desktop, Laptop, LCD, and CRT universal attachment base
Storage Temperature & Humidity	-40 °F (-40 °C) to 140 °F (60 °C) at <5% to 65% relative humidity (non-condensing)
Operating Temperature & Humidity	32° F (0° C) to 104° F (40° C) at <5% to 80% relative humidity (non-condensing)
Certification Information	
Country of Manufacture	People's Republic of China (PRC)
ISO 9001 Qualified Manufacturer	Yes
ISO 14001 Qualified Manufacturer	Yes
Restriction on Hazardous Substances	This device complies with all applicable worldwide regulations and restrictions including, but not limited to: EU directive 2002/95/EC on the Restriction of the Use of Certain Hazardous Substances in Electrical and Electronic Equipment and EU Registration Evaluation and Authorization of Chemicals (REACH) regulation regarding Substances of Very High Concern.
FCC ID	This device complies with Part 15 of the FCC Rules and Industry Canada ICES-003. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation. Tested to comply with FCC standards. For home and office use. Model number: 1393, LifeCam Cinema.
Agency and Regulatory Marks	• ACMA Declaration of Conformity (Australia and New Zealand) • ICES-003 report on file (Canada) • EIP Pollution Control Mark, EPUP (China) • CE Declaration of Conformity, Safety and EMC (European Union) • WEEE (European Union) • VCCI Certificate (Japan) • CITC Letter (Kingdom of Saudi Arabia) • KCC Certificate (Korea) • GOST Certificate (Russia) • FCC Declaration of Conformity (USA) • UL and cUL Listed Accessory (USA and Canada) • CB Scheme Certificate (International)
Windows Hardware Quality Labs (WHQL)	ID: 1451784 Microsoft Windows 7

* One megapixel = 1,000,000 pixels. Lower resolution available when sending video via instant messaging.

Results stated herein are based on internal Microsoft testing. Individual results and performance may vary. Any device images shown are not actual size. This document is provided for informational purposes only and is subject to change without notice. Microsoft makes no warranty, express or implied, with this document or the information contained herein. Review any public use or publications of any data herein with your local legal counsel.