

***ITROB 18. oktober 2015***

***Skrevet af: Vejleder:***

*Alex Andbæk Nielsen 201270408 Peter Ahrendt*

*Jacob Mose Hansen 201270410 Michael Alrøe*

*Mads Gad Krogsgaard 201270392*

*Morten Hoffmann Christensen 201270118*

Gruppe 7 – QRRobot.

# Abstract

This project involves the use of the Crustcrawler robot, OpenCV vision system and Matlab calculations to build a system able to detect a QR code with an embedded pattern, and building these patterns using Lego Dublo bricks automatically. To achieve this we used ROS (Robot Operating System) in Linux together with Python scripts to handle all the functionality. This report will seek to explain what and how the different parts of the project is realized. The actual test results are presented, and discussed in order to reflect on what we have learned throughout the course. Thus, showing the QRRobot’s capabilities in order to reflect the main goals of this project.

Indholdsfortegnelse

[Abstract 2](#_Toc432921674)

[Generel systembeskrivelse 4](#_Toc432921675)

[ROS 5](#_Toc432921676)

[Introduktion af ROS 5](#_Toc432921677)

[ROS design 5](#_Toc432921678)

[Unified Robot Description Format (URDF) 5](#_Toc432921679)

[Actionlib 5](#_Toc432921680)

[Metoder og resultater 6](#_Toc432921681)

[Den fysiske del af robotten 6](#_Toc432921682)

[Inverse Kinematic 7](#_Toc432921683)

[Mathcad 7](#_Toc432921684)

[Matlab Test 8](#_Toc432921685)

[Transformation og rotation mellem frames i Matlab 9](#_Toc432921686)

[Transformation og rotation mellem frames i Python 11](#_Toc432921687)

[Vision 12](#_Toc432921688)

[OpenCV 12](#_Toc432921689)

[QRTools 12](#_Toc432921690)

[Detektering af klodser 12](#_Toc432921691)

[Diskussion 15](#_Toc432921692)

[Yderligere diskussion 15](#_Toc432921693)

[Udfordringer og erfaringer 15](#_Toc432921694)

[Konklusion 16](#_Toc432921695)

[Bilag 17](#_Toc432921696)

[Installationsguide 17](#_Toc432921697)

[Invers Kinematic formler 17](#_Toc432921698)

[Motor kalibrering 17](#_Toc432921699)

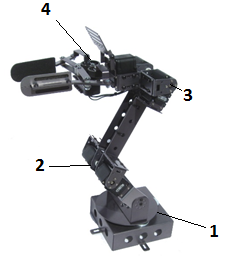
[QR koder 18](#_Toc432921700)

# Generel systembeskrivelse

I projektet ønskes det at robot systemet skal være i stand til at bygge nogle mønstre af Lego Dublo klodser, og hvor mønstrene der skal bygges vælges ud fra en scannet QR kode.

Måden hvorpå QRRobot er tiltænkt at virke, er ved at have et magasin af klodser som den kan tage fra, hvor den vha. vision systemet kan detektere en klods farve, og ud fra farven detekteres den givne klods’ center punkt. Dette center punkts koordinater udregnes så til en aktuel lokation som QRRobot bruger til at samle en klods op. Når en klods er samlet op, vil QRRobot begynde at bygge det scannede mønster, klods for klods, indtil det er færdigt.

Projektet beskæftiger sig kun med 2-D hvor klodserne er stillet i ét lag. Eventuelt vil der være mulighed for en opgradering af vision kameraet til et Kinect og opnå 3-D funktionalitet hvorved det formodes muligt at lave mønstre i flere lag. Et Kinect kamera giver den ekstra dimension der kan arbejdes med, så det ikke kun er i X-Y men også Z retningen og hermed opnå dybde mønstrene.



Figur - Crustcrawler

De anvendte aktuatorer som Crustcrawler AX-12A Smart Robotic Arm er komponeret af, er kort beskrevet her:

* 1st joint (Base Rotation) - Giver mulighed for armen at rotere om sig selv.
* 2nd joint (Dual Shoulder Rotation) og 3rd joint (Dual Forarm Rotation) anvendes til at hente og placere klodserne.
* 4th joint (Wrist Rotation) anvendes til at rotere armen under opsamling og placering af klodsen.
* Vision sensor – anvendes til detektering af klodser.

Der implementeres algoritmer til udregning, detektering og placering af klodser, som har til ansvar at styre QRRobot’s forskellige aktuarer ift. hvad den skal samle op og hvilket mønster der skal bygges.

Projektet har defineret nogle test scenarier som er den minimum funktionalitet som QRRobot skal være i stand til at udføre:

* Test 1
  + Få armen til at bevæge sig.
* Test 2
  + Få armen til at samle en klods op fra en prædefineret placering.
* Test 3
  + Lægge klodsen på en prædefineret placering.
* Test 4
  + Bygge et mønster ud fra aflæst QR-kode.

# ROS

## Introduktion af ROS

ROS er et open-source robot OS. Det er ikke et OS i traditionel forstand, men såkaldt meta OS, der fungerer som et kommunikationslag oven på det eksisterende OS.

Der anvendes i dette projekt, en portering af ROS der hedder R*ospy* skrevet i python. Designet af ROS favoriserer udviklingstiden af softwaren, over selve ydelsen i realtid. Derfor kan man hurtigere komme i gang med at udvikle prototyper til sin robot.[[1]](#footnote-1)

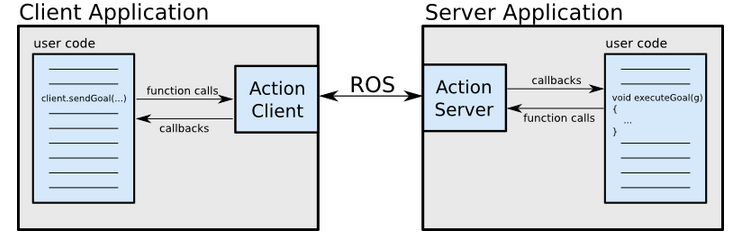
## ROS design

Overodnet set er der nogen hovedbegreber i ROS som vil blive gennemgået i dette afsnit. Derudover vil ROS specifikke biblioteker, og formater også kort blive berørt.  
  
Grundkoncepterne i ROS er noder, beskeder, services og emner. Noder er processer, og er at betragte som standalone applikationer der udfører en bestemt funktion. Noder er modulære af natur og kan kommunikere med hinanden vha. beskeder. Beskeder er baseret på IDL formatet. Beskeder bliver publiceret til et emne som andre noder der abonnerer på dette emne kan modtage. Services anvendes til synkron kommunikation, hvilket vil sige at der sendes en besked, og modtages et svar, en såkaldt client/server arkitektur.

### Unified Robot Description Format**[[2]](#footnote-2)** (URDF)

URDF filen gør det muligt for ROS at modtage informationer om robottens konfiguration og i dette tilfælde er det dens joints. URDF’s beskrivelse af robotten er forbundet til den reelle joint kontroller via navnet på joints. Disse er specificeret i joints.yaml som indeholder konfigurationerne for hver enkelt joint på robotten.

Actionlib  
Actionlib anvendes overordnet på samme måde som ROS services, altså en client/server lignende opførsel. Forskellen ligger i at actionlib er designet til at håndtere processer der har en længere udførselstid. Derfor er der også mulighed for at annullere kommandoer i tilfælde af at den tager for lang tid at udføre.



En klient og server applikation kommunikerer ved hjælp af en protokol, der er bygget oven på ROS beskeder. Klient og server har et simpelt API til opgave forespørgsel på både klient og server siden.

# Metoder og resultater

Dette afsnit beskriver hvilke metoder der er blevet brugt til at realisere projektet, samt de opnåede resultater.

## Den fysiske del af robotten

En definition af Crustcrawlerens fysiske parametre er essentielt for at den opfører sig som forventet og hermed opnår en ønsket position.

Disse referer til dem på Figur 2 og er følgende:

d1 = 16.5 cm (height position of 2nd joint from base)

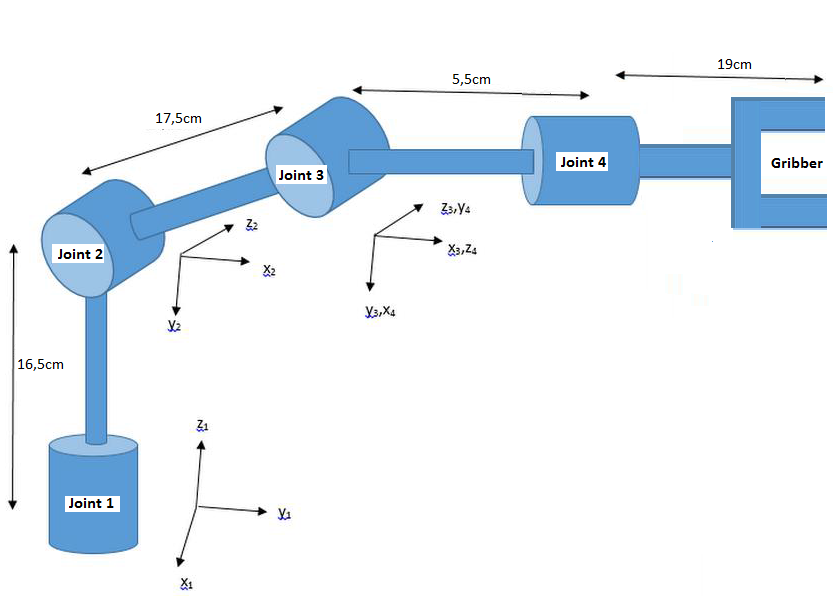
a1 = 0.0 cm (distance along "y-axis" to 2nd joint)

a2 = 17.5 cm (distance between 2nd and 3rd joints)

d4 = 24.5 cm (distance from 3rd joint to gripper center - all inclusive, ie. also 4th joint)

Hvor “*d4*” består af: de 5,5 cm som er rotationsledet + 19 cm som er gribberen = 24,5 cm.

Disse fire værdier giver en totallængde på 58,5 cm som er robottens yderpunkt for at gribe objekter ift. origo.



Figur - Crustcrawler dimensioner

Crustcrawleren består af fire Joints og en Gribber

Joint 1: Én motor der står for rotationen af hele robotten

Joint 2: To motorer der samarbejder om at vinkle ledet fra Joint 2 til 3.

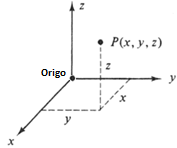
Joint 3: To motorer der samarbejder om at vinkle ledet Joint 3 til 4.

Joint 4: Én motor der står for rotationen af robottens ”Gribber”.

Griber: Én motor der står for robottens gribe funktion ved at styre gribeværktøjet.

Joint 2 og Joint 3 skal kalibreres så de 2x2 motorer arbejder sammen, metoden for dette kan findes i bilag.

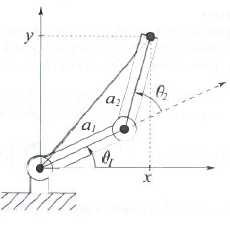
### Inverse Kinematic



Figur - Koordinatsystem

Inverse kinematic anvendes til at udregne et sæt vinkler for robotten Joints for punktet P, se Figur 3, i et koordinatsystem med en x, y og z akse, hvor punktet P repræsentere robottens værktøj (det yderste på robotten).

Ved et givent punkt P ønskes det at finde de forskellige værdier for θ1 og θ2 på Figur 4. Fordi vores robot ser ud som vist i Figur 2 introduceres yderligere formler[[3]](#footnote-3), som tager højde for robottens højde placering og det at kunne roterer om sig selv ift. Figur 4 der tog udgangspunkt alene i x/y plan og original formlerne.



Figur – Liggende Arm – 2 Dimensioner

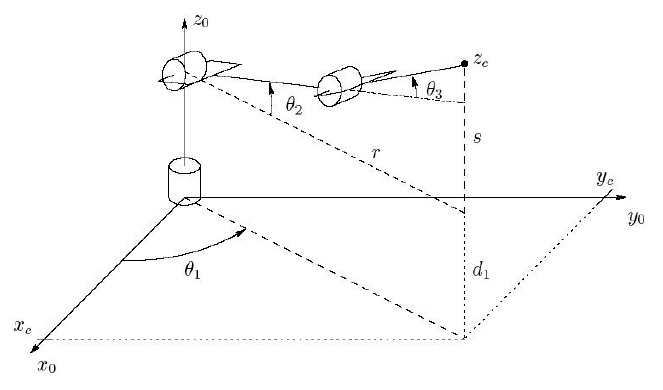
Ved at udvide formlerne ændres vores robot karakteristik sig fra den viste på Figur 4 til den viste på Figur 5 som udvider robotten med en z koordinat og en θ3.

Alle tidligere definitioner af θ erstattes nu af dem vist på Figur 5:

Θ1 (Joint1) = Rotationen omkring Z aksen (Se Figur 2)

Θ2 (Joint2) = Bevægelse (Rotation omkring Z aksen, se Figur 2)

Θ3 (Joint3) = Bevægelse (Rotation omkring Z aksen, se Figur 2)

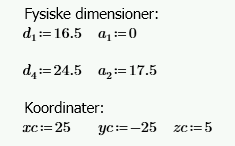


Figur - Vores Crustcrawler - 3 Dimensioner

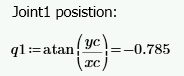
### Mathcad

Vi ønsker at bevise at vores Crustcrawler er bygget korrekt op og at formlerne vi anvender er korrekte derfor tester vi først teoretisk:

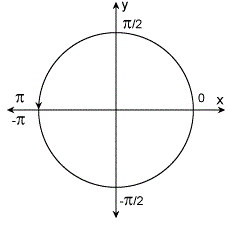
Vi angiver robottens fysiske dimensioner samt det koordinatsæt vi ønsker at robotten skal gå hen til:



Dette giver os følgende:

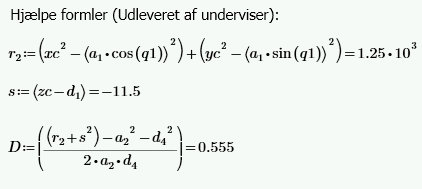


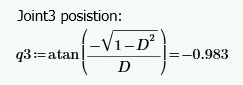
q1 returnere hvilken kvadrant der arbejdes, i dette tilfælde den fjerde kvadrant, se evt. Figur 6.

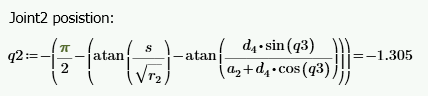


Figur – Kvadrant der arbejdes i



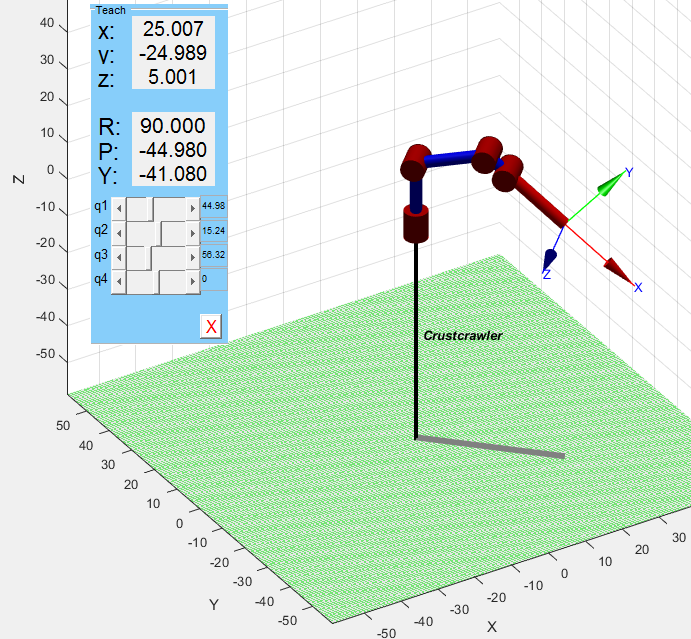






De værdier der bliver regnet for q1, q2 og q3 er vinklerne til de respektive joints i enheden radianer.

Grunden til at vi invertere q2 og trækker pi/2 fra er for at opnå at den ønskede x/y akse position ift. origo vist i Figur 6 opnås. Havde vi ikke gjort det havde joint 3 ikke opført sig som ønsket, et eksempel på dette kan findes i bilag.



Figur - Matlab Crustcrawler

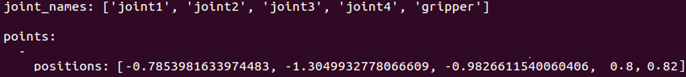
### Matlab Test

For at verificere teorien, opstiller vi et eksempel i Matlab. Her har vi ikke inverteret q2 og trukket pi/2 fra. Derfor fremstår q2 anderledes end teorien, mere om det i diskussionen, men som det ses på Figur 7 opnås den ønskede position af punktet P.

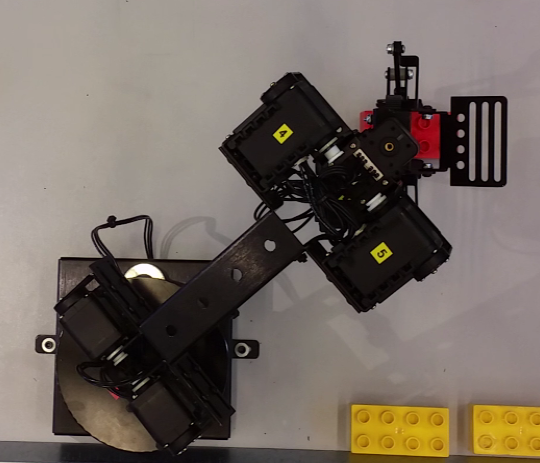
|  |  |  |
| --- | --- | --- |
|  | Radianer | -0,785 |
| q1 | Grader = | **-44,9772** |
|  | Radianer | 0,266 |
| q2 | Grader = | **15,24068** |
|  | Radianer | -0,983 |
| q3 | Grader = | **-56,3218** |

Funktions test

Her et udklip af de værdier der bliver sendt til Crustcrawleren med X = 25, Y = -25 og Z = 5:



Som det kan ses ovenover stemmer værdierne for joint1(q1), joint2(q2) og joint3(q3) overens.

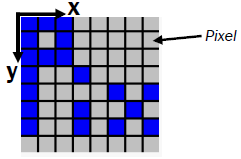


Figur - Testbillede af Crustcrawler

Her et udklip, Figur 8, fra Crustcrawlerens position hvor det ses at dens position passer overens med det forventede ift. det viste x/y frame på Figur 10 som er det frame robotten arbejder i.

## Transformation og rotation mellem frames i Matlab

Ved hjælp af det påmonterede IP kamera og den udviklede vision algoritme, er det muligt at detektere hvilke objekter der er synlige på bordpladen. Når vision algoritmen har detekteret en klods, samt dens farve, returneres center punktet for objektet i form af pixelkoordinater. Da billedets koordinatsystem hverken i enhed eller orientering, stemmer overens med robottens koordinatsystem, er det nødvendigt at lave en transformation mellem de to ”frames”.



Figur - Pixel eksempel

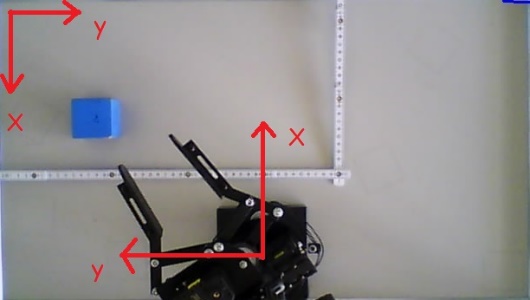
Da objektets position leveres i pixelkoordinater, omregnes pixel til centimeter. Ved hjælp af formlen

   
Formlen giver en god tilnærmelse for objektets resolution i cm. Ved at måle bordets fysiske længde og højde inden for aluminiumskanten, samt de tilsvarende længder i pixel på billedet, gav det følgende resultater:

Længde opløsning: 

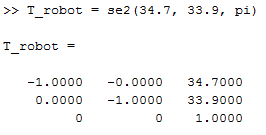
Højde opløsning: 

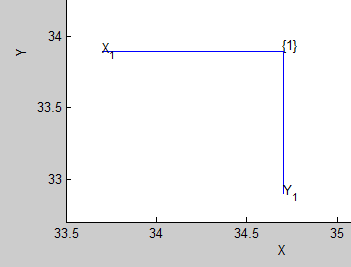
For kun at detektere klodser på bordpladen, beskæres billedet til kun at dække bordpladen. Dette har også den fordel at øverste venstre hjørne på billedet svarer til øverste venstre hjørne på bordpladen set fra kameraets position. Da kameraets ”frame” er defineret som øverste venstre hjørne er det muligt at måle robottens fysiske position ud fra hjørnet. Robottens position måles til samt en rotation på 180o i forhold til kameraets ”frame”.



Figur - Frames

Ved hjælp af Matlab funktionen ”*se2*” i Robotic toolbox udviklet af Peter Corke, er det muligt at få returneret den homogene transformation som repræsentere translation (34.7, 33,9) samt rotationen på 180o.



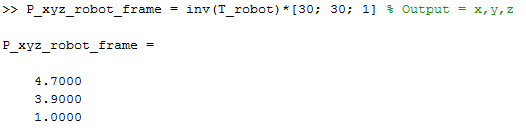


Figur - Resultat fra Matlab

Som det ses på plottet stemmer transformationen overens med de to frames position på billedet.

For at finde en position opgivet ud fra kameraets frame, er det nødvendigt at tage den inverse af ovenstående transformation for at finde klodsens position ud fra robotarmen. Det vil sige at punktet set fra robottens frame er:





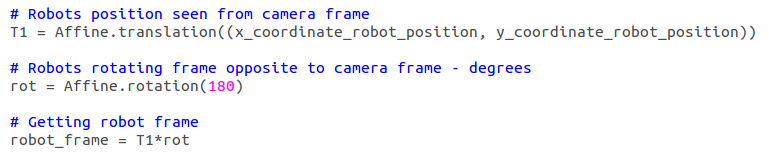
Da robottens position er ses det tydeligt at det beregnede punkt er korrekt.

## Transformation og rotation mellem frames i Python

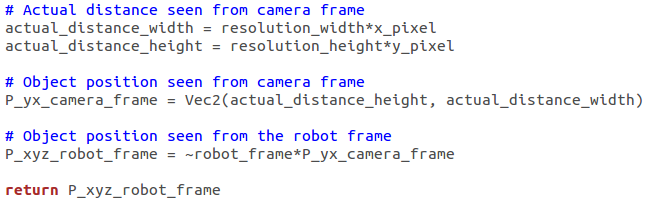
For at kunne bruge udregningerne lavet i Matlab i det samlede projekt skulle vi have konverteret disse rotations transformationer til python kode. Da det ikke var muligt at mappe direkte fra matlab til python pga. den brugte ”se2” funktion fra robotics toolbox, fandt vi et 2D geometri bibliotek til python ved navn ”Planar”. Vha. dette bibliotek, kunne vi lave vores lineær transformation i 2D og derefter rotere vores transformerede frame så det passer til QRBotten.

Python koden herunder viser hvordan vi benytter Planar bibliotekets translation funktion til at flytte koordinat systemet så det passer til robottens placering, og rotation funktionen som roterer dette koordinat system så vender rigtigt ift. kamera framet. Variablerne x\_coordinate\_robot\_position og y\_coordinate\_robot\_position er robottens verdens koordinater set fra kamera framet.





Måden hvorpå vi så udregner et objekts position ud fra robotten frame, er ved først at finde de aktuelle distancer set fra kamera framet. Dette gøres ved at multiplicere længde og højde opløsningen med de x og y koordinater, angivet i pixels, som modtages fra vision delen. Den aktuelle højde og længde udgør derfor et objekts position set fra kamera framet. Der gøres her brug af en funktion fra Planar biblioteket ”Vec2” laver vores aktuelle distancer om til en vector. For så at finde et objekts position set fra robotten frame er ved at tage den inverse robot frame og multiplicerer med vores vector som derved retuneres.



## Vision

### OpenCV

OpenCV er et open source projekt, oprindeligt udviklet af Intel, til objekt detektering, ansigtsgenkendelse, motion tracking og en lang række andre ting. [[4]](#footnote-4) I dette projekt er det blevet brugt til at detektere duplo klodser ved hjælp af et kamera.

### QRTools

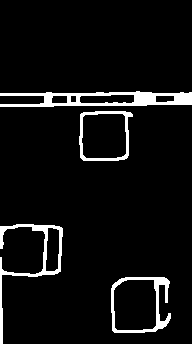
Til at oprette, og aflæse QR koder er QRTools[[5]](#footnote-5) til python blevet anvendt. QRtools er et bibliotek der er sammensat af projekterne qrencode[[6]](#footnote-6) som anvendes til at oprette QR mønstre, og zbar[[7]](#footnote-7) som bruges til at aflæse koderne med. QRTools er valgt fordi det er let at bruge, og anvendelsesmulighederne er mange. Et problem med det var dog at det ikke umiddelbart er kompatibelt med ip kameraer, som er den type kamera der er anvendt i dette projekt. Dette er blevet løst ved kontinuerligt at gemme et billede lokalt ved hjælp af OpenCV, og så scanne dette billede med QRTools. I QR koden er der gemt en tekststreng som, i koden, bliver anvendt til at vælge hvilket mønster der skal bygges. Et problem opstod da vi skulle dekode QR koden tilbage til en tekststreng, fordi strengen som QRTools returnererede var formateret i udefineret format. Problemet var at man ikke kunne sammenligne den streng med en almindelig streng i python. Løsningen blev at sætte default encoding i python scriptet til UTF-8, og derefter fungererede scriptet efter hensigten.

### Detektering af klodser

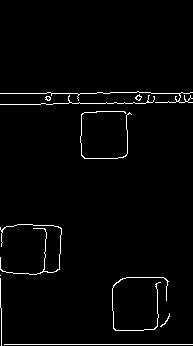
Til at detektere klodserne anvendes kameraet som er monteret ovenover crustcrawler robotten.   
For at klods detektionen kan fungere så optimalt som muligt, har det været nødvendigt at omskrive den udleverede kode[[8]](#footnote-8) til en mere stabil løsning. Størstedelen af genkendelseskoden er indeholdt i funktionen detect brick. Denne funktion tager billedet der skal undersøges, samt øvre og nedre grænseværdier for farven på klodsen angivet i BGR værdier som parametre. Koden for denne funktion vil blive gennemgået i detaljer her.



Først beskæres billedet så unødig genskind fra kanter af bordet fjernes. Dette gøres også for at få arbejdsområdet for robotten, så andet unødigt støj ligeledes bortsorteres.   
Efter dette anvendes medianBlur til at sløre billedet en smule, for at udjævne billedet, for derved at eliminere pixelfejl og deslignende. Derefter laves det om til gråtoner, da det er nemmere at detektere kanter på et gråtone billede. Canny edge detection anvendes for at trække kanterne i billedet længere frem inden threshold funktionen transformerer billedet om til et binært billede[[9]](#footnote-9) således at kanterne bliver hvide, og resten er sort. Efter dette anvendes dilation til at forstørre kanterne så det bliver nemmere at finde konturerne i billedet efterfølgende. Næstefter anvendes morphologyEx til yderligere at tydeliggøre kanterne med, og til sidst anvendes findContours til at finde konturerne i billedet.  
I for løkken traverseres alle fundne konturer igennem, og der approksimeres en konturform og størrelse samt arealet af konturen. Der valideres så på om arealet holder sig inde for nogen grænseværdier, og hvis dette er tilfældet detekteres centerpunktet for konturen. BoundingRect bliver brugt til at finde den, for klodsen, omkringliggende firkantkontur, og billedet beskæres for at sortere andet fra inden farven detekteres. Det beskårede billede bruges så til at finde en middelværdi for farven på klodsen. Den fundne farve bliver så indsat i et numpy array, og der verificeres at arrayet ikke består kun af 0 værdier. Såfremt arrayet består af 0 værdier betyder det at ingen farve er funden, og den bliver derfor sorteret fra. Er der fundet en farve gemmes denne farve i en tuple sammen med centerkoordinatet, og denne tuple returneres fra funktionen.



Figur - resultat efter morphologyEx, og thresholding



Figur - resultat efter canny edge detection og beskæring af billedet

# Diskussion

**Test 1 – Få armen til at bevæge sig.**

I forbindelse med de laboratorie øvelser vi har haft, har vi beskæftiget os med at få Joint 1-4 til at bevæge sig. At tilføje rotationsledet (joint 4) og gribberen sammen med joint 1, 2 og 3 var den svære del, da denne ikke var udleveret til den inverse kinematic, kontra den første del. Det lykkedes dog for os at bevæge alle led på robotten. Dette blev gjort ved at udvide den eksisterende kode til joint 1-3 for inverse kinematic som i stedet for kun at fungere med tre parametre [X,Y,Z] blev udvidet til at fungere med fem parametre [X,Y,Z,Gribber,Rotation].

**Test 2 – Få armen til at samle en klods op fra en prædefineret placering.**

Når der var styr på at få leddene til at bevæge sig, fokuserede vi på at få robotten til at gå til prædefineret placeringer. Grundet robotten havde flere præcisions variationer blev der kompenseret for disse, da variationerne var gentagelige. F.eks. var der behov for at ændre på værdien for q2 da det ved trial-and-error tests blev verificeret at robotten arbejdet omkring et forkert origo. Efter dette kunne vi måle et punkt ud på arbejdsbordet, indtaste værdierne til robotten, hvorefter den vil gå hen til denne position for at samle en evt. klods op. Det viste sig dog at hvis vi skiftede robot kunne vi få f.eks. overload men dette blev fixet ved at kalibrer motorerne i jointsne på ny.

**Test 3 – Lægge klodsen på en prædefineret placering.**

Efter at robotten succesfuldt fik samlet en klods op, var det simpelt at få den placeret i en prædefineret placering med en tolerance på få mm. At lægge adskillige klodser tæt på hinanden, tog nogle forsøg før det lykkedes.

**Test 4 – Bygge et mønster ud fra aflæst QR-kode.**

Implementeringen af QR-koden var relativt let da biblioteket var nemt at anvende, der var lidt problemer med IP-kameraet men det blev løst og dette fungere således også. Mønsteret var prædefineret men klodsernes opsamlings position var ukendt for QR-koden. Dette skulle vision delen håndtere og dette lykkedes relativt godt at få til at fungerer.

## Yderligere diskussion

### Udfordringer og erfaringer

Igennem projektet har der været et par udfordringer som har haft en stejl kurve at komme igennem, men også givet os den erfaring der skal bruges for løse de diverse udfordringer. Dette diskuteres kort i følgende afsnit:

* **Motor kalibrering**

Der er under brug af robotten ofte sket overload af nogle af robottens motorer. Dette gør at vi har været nødt til at genstarte de eventuelle tests der har kørt når overload er sket. For at ”resette” robotten og fjerne overload fejlen bliver man nødt til at tage strømmen helt fra robotten og starte forfra. Måden hvorpå vi kan være på forkant for overloading, er ved at kalibrer motorerne så begge motorer i de forskellige joints er synkroniseret og arbejder sammen i stedet for det kun er den ene der tager det meste af slæbet.

* **Kamera placering**  
  Selve placeringen af kameraet på den fastmonterede stålramme har ikke været ens på alle robotter hvilket har medført at vores tests af robotten har været inkonsistent ift. hvilken robot vi har kunne bruge. Dette gjorde at vi har prøvet at tilpasse vores udregninger og kode så det passer til den individuelle robot vi har siddet med. Det har været en stor udfordring at komme uden om dette problem da vi en dag lavede perfekte tests og så dagen efter fejlede det hele. Vi har her erfaret at vi er nødt til at tjekke op på kameraets reelle placering når vi har fat i en ny robot, så origo ud fra kamera framet passer til de parametre som vi køre vores robot med.
* **Lyssætning**

Når farven på klodsen skal detekteres, er problemet at lysforholdende svinger utroligt meget. Selv vejret udenfor kan have indflydelse på om en farve kan detekteres korrekt. Den primære årsag til dette skal findes i at de interval værdier der er blevet anvendt i dette projekt ikke har været præcise nok. Årsagen er at intervallerne for de enkelte farve overlapper hinanden, hvilket besværliggør detektionen. En løsning på dette kunne være at eliminere value værdierne i hsv billedet inden inRange kaldes. En anden problematik er det anvendte kameras dårlige billedkvalitet, og farve gengivelse.

# Konklusion

QRRobot projektet har budt på en række udfordringer undervejs, hvor specielt detektering af klodsers farver har vist sig upålidelig. Den umiddelbare konklusion er, at detekteringen er alt for følsom over for udsving i lysforholdene, samt at et almindeligt kamera ikke er den optimale løsning.

I starten af projektet har nøjagtigheden af en klods placering haft for høj prioritering. Dette kom bl.a. til udtryk ved, at der blev brugt for meget energi på undersøge forskydningseffekten på klodser placeret skråt for kameraet, samt forsøge kun at åbne griberen mindst muligt for at gribe en klods. Problemet består i at der introduceres en række fejkilder som gør at tolerancen for griberen er nødt til at være større. Af fejlkilder kan nævnes: center position på klods via vision, afstand mellem kamera frame og robot frame samt vinklen mellem disse og ikke mindst modellen af selve robotten.

Overordnet set opfylder QRRobot de opstillede krav: aflæse QR, opsamle klods vha. vision og placere klodserne efter mønstre fra QR kode. Dog har farvegenkendelsen vist sig at være mangelfuld.

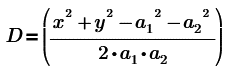
# Bilag

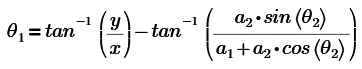
## Installationsguide

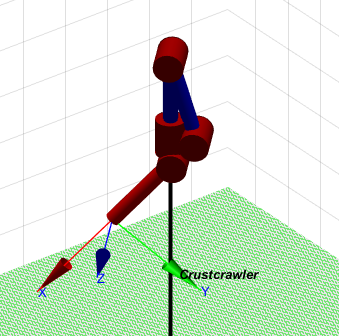
For at kunne køre applikationen skal QRTools installeres. Der er i zip filen vedlagt en debian pakke som skal installeres for at kunne køre projektet.  
I nodes mappen ligger selve projekt filerne, og i launch mappen ligger base.launch, og meta.launch. Disse er stort set identisk med dem fra undervisningen, med små ændringer til forskel. Conf mappen indeholder joints.yaml, og denne er anvendt til at kalibrere robotarmen. Der er til projektet ikke oprettet nogen ny package i ROS, derimod er au\_crustcrawler\_base blevet anvendt.

## Invers Kinematic formler

Originalformlerne som de findes i SPONG P22-23





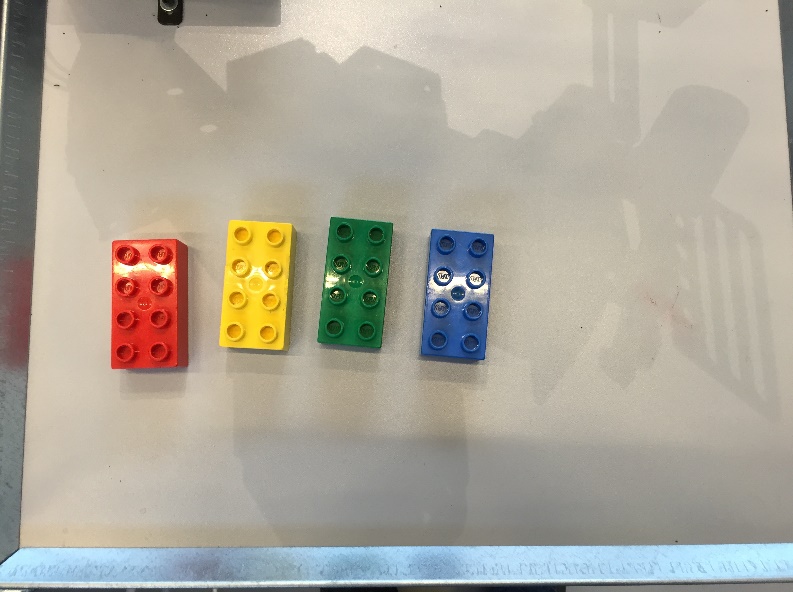


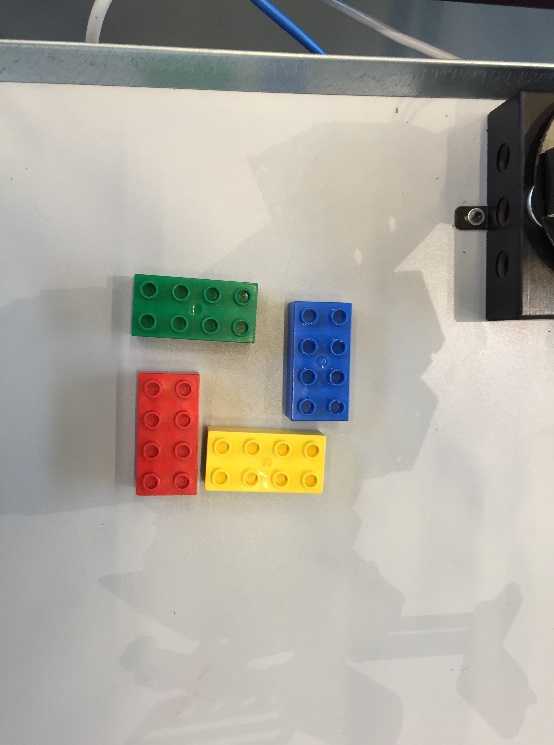
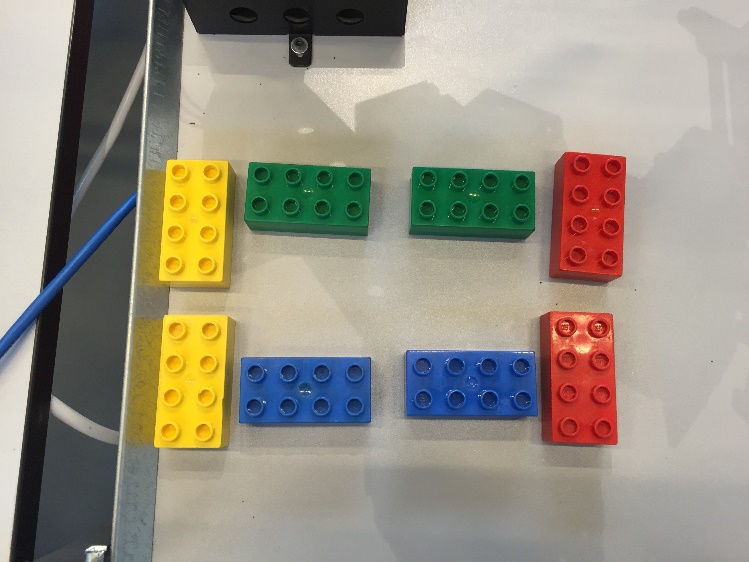
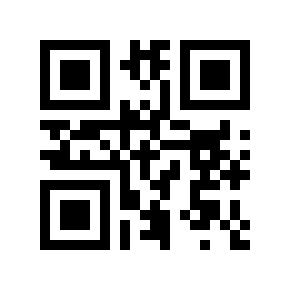
Eksempel hvis q2 ikke havde været kompenseret 🡪

## Motor kalibrering



## QR koder

Test pattern  


Box pattern  
  
  
Rectangle pattern  


1. Icraoss09-ROS.pdf [↑](#footnote-ref-1)
2. <http://wiki.ros.org/urdf> [↑](#footnote-ref-2)
3. De originale formler fra SPONG P22-23 er i bilag og disse er baseret på Figur 4. [↑](#footnote-ref-3)
4. https://en.wikipedia.org/wiki/OpenCV [↑](#footnote-ref-4)
5. https://launchpad.net/qr-tools [↑](#footnote-ref-5)
6. https://github.com/Arachnid/pyqrencode/tree/master [↑](#footnote-ref-6)
7. http://zbar.sourceforge.net/ [↑](#footnote-ref-7)
8. Se au\_opencv\_example.py [↑](#footnote-ref-8)
9. Et binært billede er et billede hvor en pixel kun kan antage to værdier, typisk sort/hvid [↑](#footnote-ref-9)