

***ITROB 16. oktober 2015***

***Skrevet af: Vejleder:***

*Alex Andbæk Nielsen Peter Ahrendt*

*Jacob Mose Hansen Michael Alrøe*

*Mads Gad Krogsgaard*

*Morten Hoffmann Christensen*

Gruppe 7 – Mr. QR-bot.

# Abstract

Indholdsfortegnelse

[Abstract 1](#_Toc432763199)

[Indledning 3](#_Toc432763200)

[Generel systembeskrivelse 3](#_Toc432763201)

[Beskrivelse af ROS 3](#_Toc432763202)

[Introduktion af ROS 3](#_Toc432763203)

[Hvordan virker ROS 4](#_Toc432763204)

[ROS i vores projekt 5](#_Toc432763205)

[Metoder og resultater 6](#_Toc432763206)

[Robottens dimensioner 6](#_Toc432763207)

[Inverse Kinematic 7](#_Toc432763208)

[Transformation og rotation mellem frames i Matlab 9](#_Toc432763209)

[Transformation og rotation mellem frames i Python 9](#_Toc432763210)

[Vision 9](#_Toc432763211)

[OpenCV 9](#_Toc432763212)

[QRTools 9](#_Toc432763213)

[Detektering af klodser 9](#_Toc432763214)

[Diskussion 9](#_Toc432763215)

[Konklusion 9](#_Toc432763216)

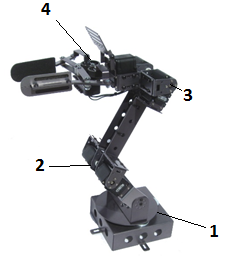
[Bilag 9](#_Toc432763217)

# Indledning

# Generel systembeskrivelse

Vi ønsker at bygge et mønster ud fra brugerens valg, valget kunne tages ud fra at fortælle robotten via en QR-kode eller en GUI på computer hvor man herfra kan vælge det mønster man gerne vil have den til at bygge.

Vi starter med at give robotarmen et magasin som den kan tage nogle klodser fra. Robotten detektere en klods i magasinet ud fra vision systemet hvor den lokalisere en klods og samler den op. Disse klodser skal så stilles efter det givne mønster. I starten beskæftiger vi os med 2-D hvor vi stiller klodser i ét lag. Eventuelt vil der være mulighed for en opgradering af vores vision kamera til et Kinect og hermed opnå 3-D funktionalitet hvorved det formodes muligt at lave mønstre i flere lag. Et Kinect kamera vil give os en ekstra dimension som giver os den fordel at vi kan arbejde ikke kun i X-Y men også Z retningen og hermed opnå dybde i vores mønstre. Dvs. at vi nu anvender længde bredde og højde målinger for at estimere placering af klodser.



Figur 1 - Crustcrawler

Vi anvender de aktuatorer der er i AX-12A Smart Robotic Arm som er på robotarmen.

* Første led(1 – Base Rotation) der giver mulighed for armen at rotere om sig selv.
* Andent led(2 – Dual Shoulder Rotation) og tredje led(3 – Dual Forarm Rotation) anvendes til at opnå at hente og placere klodserne.
* Fjerde led(4 – Wrist Rotation) anvendes til at rotere armen under opsamling af klodsen og placering.
* Vision sensor og evt. Kinect.
* Algoritme til at aflæse QR-kode.
* Vi vil anvende algoritmer til at samle klodserne op med som er placeret på prædefineret placeringer. Disse skal udregne hvordan armen skal bevæges for at samle klodsen op.
* Vi skal have nogle algoritmer til at udregne hvordan robotarmen skal opnå at placere klodsen på den givne placering ifht. det mønster vi giver robotten. (både placeringen fra sig X/Y/(Z) men også vinklen på klodsen (om den skal drejes))

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.

# Beskrivelse af ROS

## Introduktion af ROS

Robot Operating System (ROS) er et open-source robot operations system (OS). ROS er ikke et traditionelt OS, det er en struktureret kommunikations lag der ligger oven på det OS der anvendes.

ROS er et værktøj der skaber nogle basale rammer for software udviklingen af en robot. Vi anvender en udvikling af ROS der hedder *rospy* og er principielt et bibliotek af ROS der er implementeret i Python. Rospy er et Applikations Programmerings Interface (API) og det muliggøre programmører at arbejde med Python til hurtigere at udvikle deres robotter designet med ROS. Designet af ROS favorisere udviklingen af softwaren over selve ydelsen i realtid på robotten. Derfor kan man hurtigere komme i gang med at udvikle prototyper til sin robot.

## Hvordan virker ROS

ROS er designet ud fra fem mål som beskriver hvordan ROS virker, disse er:

* Peer-to-peer

Forbindelsesmetoden der anvendes er Peer-to-peer (ptp) og dette sikre at de forskellige processor der potentielt set kan kører på forskellige værter, forbundet i realtid i et ptp netværk, stadig kan opretholde kommunikation selvom nettet er trafikeret. Dette kræver en form for opslags mekanisme der tillader at forskellige processor kan finde hinanden i realtid, disse er kaldt *name service* som er yderligere beskrevet i punktet ”Tools-based”.

* Multi-lingual

Når der arbejdes med kode har man forskellige præferencer for programmerings sprog. Derfor er ROS designet i et neutralt sprog og dette kan håndtere fire forskellige sprog: C++, Python, Octave og LISP. Derfor er ROS et meddelelses lag. Derfor er ROS’ kode generator eksisterende for hvert sprog og denne håndtere konverteringen af sproget så programudvikleren spare en masse tid. Det er også muligt at anvende flere sprog af gangen.

* Tools-based

ROS er designet til at køre bedst på flere mindre kerner der hver i sær håndtere en lille del af ROS’ værktøjer. Dette er valgt frem for en stor kerne der skulle håndtere alle opgaverne samlet. Disse små dele af værtkøjer kunne fx håndtere kildekoden, læse og skrive konfigurations parametre, visualisere ptp netværket osv. Dette er gjort fordi ROS tror på at tabet i effektivitet er stører end gevinsten ved stabilitet og at have et mindre komplekst system.

* Thin

Ofte er kode der bliver udviklet til robotter viklet ind i middlewaren[[1]](#footnote-1) som gør det besværligt at udvinde funktonaliteten og genanvende den til anden brug. Derfor opfordrer ROS til at alle drivere og algoritmer bliver udviklet i selvstændige biblioteker der ikke er afhængige af ROS. ROS’ funktion *CMake* gør det tilnærmelsesvis nemt at anvende denne ideologi. Som en ekstra fordel er enhedstest ofte nemmere når koden er sat i biblioteker. Derfor genanvender ROS også kode fra andre open-source projekter for at sikre at der automatisk bliver opdateret.

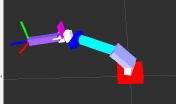
* Free and Open-Source

ROS er open-source og derfor kan alle bidrage til udviklingen og alle har mulighed for at modificere ROS til at omfavne det projekt man selv arbejder med.

## ROS i vores projekt

**Unified Robot Description Format[[2]](#footnote-2) (URDF)** anvendes til at beskrive en robot.

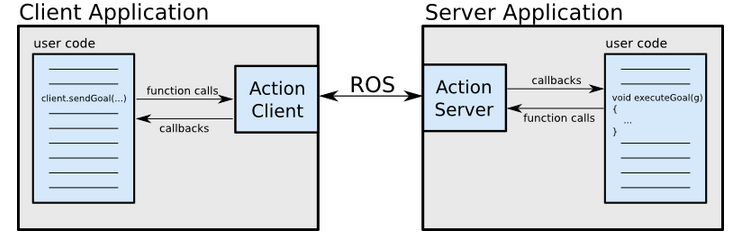
URDF filen gør det muligt for ROS at modtage informationer om robottens konfiguration og i dette tilfælde er det dens joints. URDF’s beskrivelse af robotten er forbundet til den reelle joint kontroller via navnet på jointsne. Disse er specificeret i joints.yaml som indeholder konfigurationerne for hver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



Figur 2 - RVIZ figure

**Ross visualization[[3]](#footnote-3) (RVIZ)** er et værktøj til at visualiserer den robot der arbejdes med. Det kan anvendes til at verificere den opstilling man har i virkeligheden med den som man får genereret i programmerne. Det er muligt at navigere robotten vha. musen og man kan heraf få et indtryk i robottens muligheder og begrænsninger.

**Actionlib** anvendes som et interface til … at administrere at forespørgsler kommer igennem når andre handlinger optager alle resourcer.



ActionClient og ActionServer kommunikere via en ”ROS Action Protocol”, som er bygget ovenpå ROS beskeder. Client og Server giver så en simpel API for brugere at forespørge om opgaver, på klientsiden, eller udføre opgave, på serversiden, vha. funktions kald og callbacks.

Dynamixel\_controller tildeler Crustcrawleren med en ActionServer.

Der forespørges om en ActionClient til at kommunikere med Crustcrawleren.

Joint Trajectory Action Controlleren eksekvere joint-space banerene for et sæt af joints. Den ønskede position og hastighed for hver joint til et givent punkt i forhold til tiden styres her. Dette gør den ved at interpolere punkterne i banen med splines.

**Nodes**

# Metoder og resultater

”Beskriv hvad dette afsnit indeholder”

## Robottens dimensioner

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 i Figur 3 og er følgende:

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

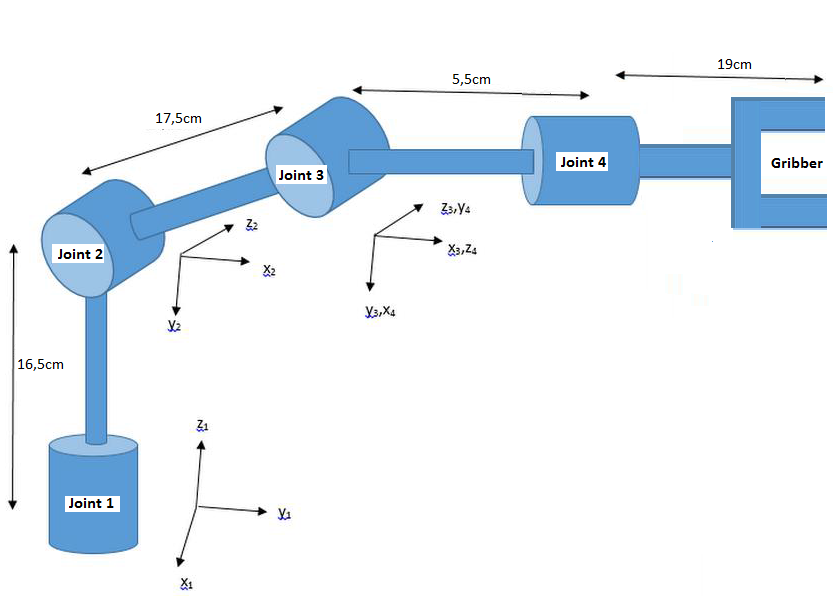
a1 = 0.0 cm (distance along "y-axis" to 2nd joint)

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 3 - 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.

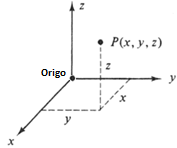
Joint 3: To motorer der samarbejder om at vinkle ledet Joint 3 til 4.

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

Gribber: Én motor der står for robottens gribe funktion sammen med gribeværktøjet.

## Inverse Kinematic

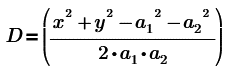
Inverse kinematic anvendes til at udregne et sæt vinkler for robotten Joints for punktet P, se Figur 4, i et koordinatsystem med en x, y og z akse, hvor punktet P repræsentere robottens værktøj (det yderste på robotten).

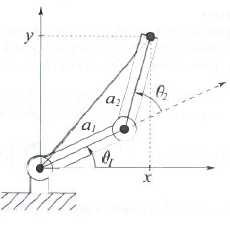


Figur 4 - Koordinatsystem

Ved et givent punkt P ønskes det at finde de forskellige værdier for a1, a2,θ1 og θ2.

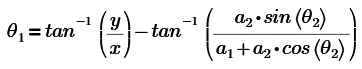
Dette gøres vha. formlerne vist herunder[[4]](#footnote-4) for armen vist på Figur 5:





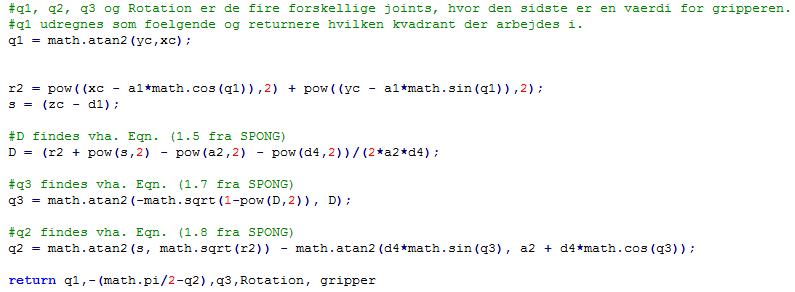
Figur 5 - Arm

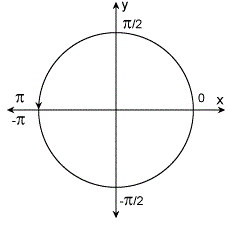




Fordi vores robot ser ud som vist i Figur 3 introduceres følgende formler[[5]](#footnote-5), som tager højde for robottens højde placering og det at kunne roterer om sig selv ift. Figur 5:

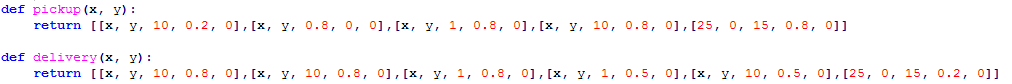
q1 udregnes som følgende og returnere hvilken kvadrant der arbejdes i.





Figur 6 – Kvadrant der arbejdes i

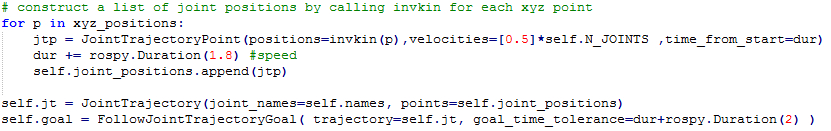
Grunden til at vi invertere q2 og trækker pi/2 fra er for at opnå at den ønskede x/y/z akse position ift. origo vist i Figur 3 opnås. De værdier der returneres er vinklerne til de respektive joints i enheden radianer.



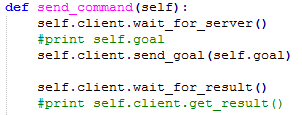
Denne kode bliver kaldt i forbindelse med når koordinater skal tilgås.

Først bliver koordinaterne defineret hvor x, y kommer fra vision delen beskrevet i og z og gripper bestemmer denne del af koden, derfor er disse statiske. Rotation er bestemt af vinklen klodsen skal placeres i, som kommer fra angle I main koden.

Koden der anvender funktionerne er:

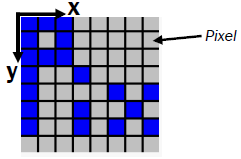


Denne kalder, for hvert koordinatsæt, en inverse kinematic beregning (beskrevet tidligere) af punkterne for så at sende det til Crustcrawleren:



## 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 7 - Pixel eksempel

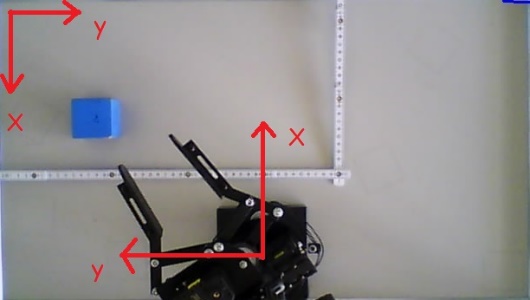
Da objektets position leveres i pixelkoordinater, omregnes pixel til centimeter. Ved hjælp af formlen

   
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: 

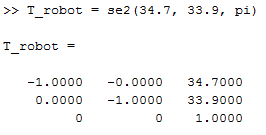
Højde opløsning: 

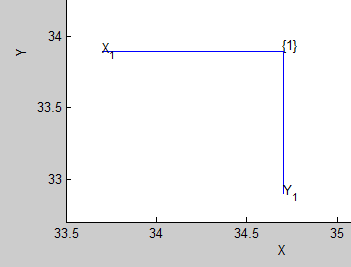
For kun at detektere klodser på bordpladen, beskæres billedet til kun at dække bordpladen. Dette har også den fordel at øverste venstre hjørne på billedet svarer til øverste venstre hjørne på bordpladen set fra kameraets position. Da kameraets ”frame” er defineret som øverste venstre hjørne er det muligt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



Figur 8 - 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.



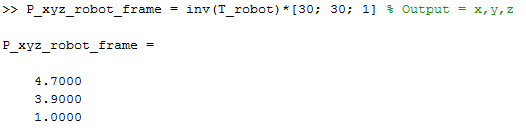


Figur 9 - Resultat fra Matlab

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:





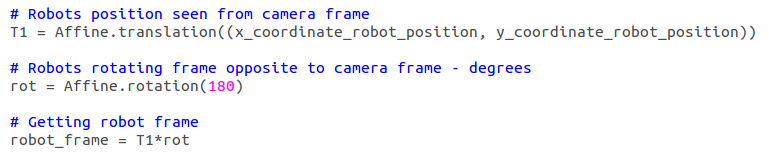
Da robottens position er ses det tydeligt at det beregnede punkt er korrekt.

## Transformation og rotation mellem frames i Python

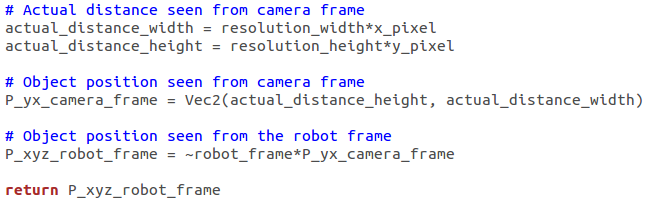
For at kunne bruge udregningerne lavet i Matlab i det samlede projekt skulle vi have konverteret disse rotations transformationer til python kode. Da det ikke var muligt at mappe direkte fra matlab til python pga. den brugte ”se2” funktion fra robotics toolbox, fandt vi et 2D geometri bibliotek til python 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. [[6]](#footnote-6) I dette projekt er det blevet brugt til at detektere duplo klodser ved hjælp af et kamera.

### QRTools

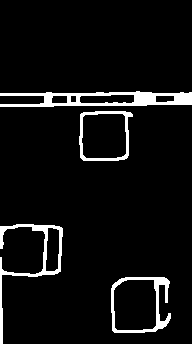
Til at oprette, og aflæse QR koder er QRTools[[7]](#footnote-7) til python blevet anvendt. QRtools er et bibliotek der er sammensat af projekterne qrencode[[8]](#footnote-8) som anvendes til at oprette QR mønstre, og zbar[[9]](#footnote-9) som bruges til at aflæse koderne med. QRTools er valgt fordi 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

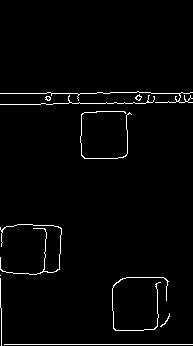
Til at detektere klodserne anvendes kameraet som er monteret ovenover crustcrawler robotten.   
For at klods detektionen kan fungere så optimalt som muligt, har det været nødvendigt at omskrive den udleverede kode[[10]](#footnote-10) til en mere stabil løsning. Størstedelen af genkendelseskoden er indeholdt 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. 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[[11]](#footnote-11) 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 10 - resultat efter morphologyEx, og thresholding



Figur 11 - resultat efter canny edge detection og beskæring af billedet

# Diskussion

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

Efter en masse forskellige forsøg med at få diverse led til at bevæge sig, lykkedes det at få alle led til at bevæge sig. At tilføje rotationsledet (joint 4) og gribberen foruden joint 1, 2 og 3 var den svære del, da denne ikke var udleveret, kontra den første del. Dermed lykkedes det os at bevæge alle led i armen.

**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. 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 klods op.

**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 og ovenpå hinanden tog nogle forsøg før det lykkedes.

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

Diskuter om ”lysforhold, fyssiske paramtre der vairer, kode crap i python, forskellige hacks i python.”

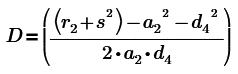
# Konklusion

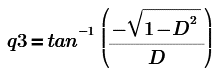
# Bilag

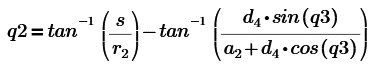












1. <https://da.wikipedia.org/wiki/Middleware> [↑](#footnote-ref-1)
2. <http://wiki.ros.org/urdf> [↑](#footnote-ref-2)
3. <http://wiki.ros.org/rviz> [↑](#footnote-ref-3)
4. Formlerne er fra SPONG P22-23 [↑](#footnote-ref-4)
5. Formlerne for r2, s, D, q2 og q3 er i bilag. [↑](#footnote-ref-5)
6. https://en.wikipedia.org/wiki/OpenCV [↑](#footnote-ref-6)
7. https://launchpad.net/qr-tools [↑](#footnote-ref-7)
8. https://github.com/Arachnid/pyqrencode/tree/master [↑](#footnote-ref-8)
9. http://zbar.sourceforge.net/ [↑](#footnote-ref-9)
10. Se au\_opencv\_example.py [↑](#footnote-ref-10)
11. Et binært billede er et billede hvor en pixel kun kan antage to værdier, typisk sort/hvid [↑](#footnote-ref-11)