

***ITROB 9. oktober 2015***

***Skrevet af: Vejleder:***

*Alex Andbæk Nielsen Peter Ahrendt*

*Jacob Mose Hansen Michael Alrøe*

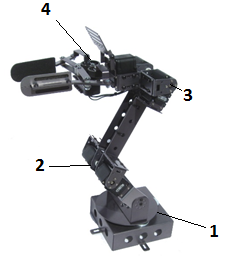
*Mads Gad Krogsgaard*

*Morten Hoffmann Christensen*

Gruppe 7 – Mr. QR-bot.

# 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.

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

* Platformsaktuatoren(1 – Base Rotation) der giver mulighed for armen at rotere om sig selv.
* Første led(2 – Dual Shoulder Rotation) og andet led(3 – Dual Forarm Rotation) anvendes til at opnå at hente og placere klodserne.
* Tredje 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

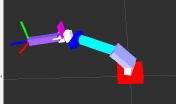
Ofte er kode der bliver udviklet til robotter viklet ind i middlewaren[[1]](#footnote-1) som gør det besværligt at udvinde funktonaliteten og genanvende den til anden brug. Derfor opfordrer ROS til at alle drivere og algoritmer bliver udviklet i selvstændige biblioteker der ikke er afhængige af ROS. 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.

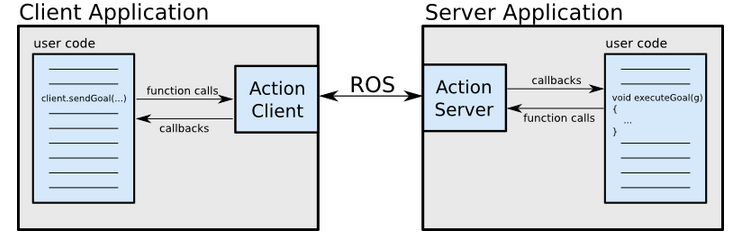
## ROS i vores projekt

**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.

**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

## Hvordan problemet løses

### Crustcrawler – Mads

### RVIZ – Morten

### Kamera til Crustcrawler algoritmer – Alex og Jacob

## ROS arkitektur design

## Kamera algoritmer

## Matlab simuleringer og beregninger

# Resultater

**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 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.**

# Diskussion

# Konklusion

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)