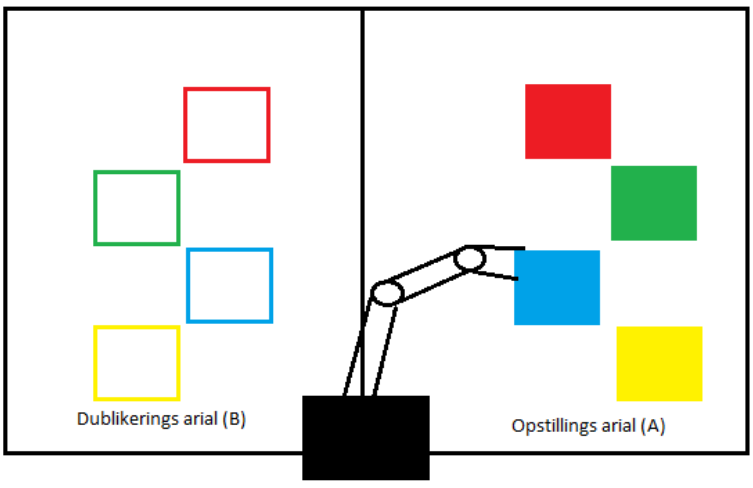
Indholdsfortegnelse

Indledning 1

I den indrustrielle verden i dag anvendes robotter i høj grad. Robotterne bruges til at lette arbejdsburden for den almindelige arbejder, og for at opnå en hurtigere arbejdsgang. Robotterne er simpelthen med til at fremme arbejdsprocessen, da virksomhederne er interesseret i at holde omkostningerne lave. For indrustri verden er det billigere at have en robot der kan arbejde fireogtyve timer i døgnet end et mennesker, som har krav på en pause en gang i mellem. Robotterne kan for eksempel bruges til, at overvåge andre systemer, og give feedback på de data som den har adgang til. Robotterne er ofte meget effektive og ikke mindst præcise i deres arbejde, og kan derfor spare virkesomheder for menneskelige fejl. Mennesker flytter sig mere og mere fra at skulle være i den fysiske del af en produktion, til at være i den kreative del, altså udviklingsledet. Robotterne overtager stille og roligt den fysiske del og er de kommet for at blive.

Opgavebeskrivelse 2

I faget ITROB1 er der blevet stillet en opgave om, at skrive et program, som skal kunne styre den mekaniske robotarm også kaldet CrustCrawler. Selve opgaven blev stillet meget fri, og derfor var det op til gruppen, at bestemme hvordan denne skulle løses. De eneste krav til opgaven var at der skulle oprettes en forbindelse mellem roboten og den tilhørende webcam. Dertil skulle der også oprettes to seperate noder, hvilket vil gøre at noget af CrustCrawlerens funktionalitet bliv kørt asynkront.
Det blev bestemt af gruppen, at CrustCrawler skulle kunne flytter klodser fra en position til den inverse position. For at kunne inkludere kameraet (webcam) bliver dette brugt til at finde klodsernes start position. Grundet klodserne har forskellige farver kan kameraet via nogle grænseværdier skelne de forskellige klodser fra hinanden, og derved finde deres positioner. CrustCrawleren får disse koordinater og samler klodsen op. Via inverse kinematik regning bliver den inverse position fundet. CrustCrawleren lægger klodsen på den ny fundne position og retunerer til udgangspunktet.



Figur 2.1. Robot Figur

Systembeskrivelse 3

3.1 System setup

3.2 Hardware

3.3 Software

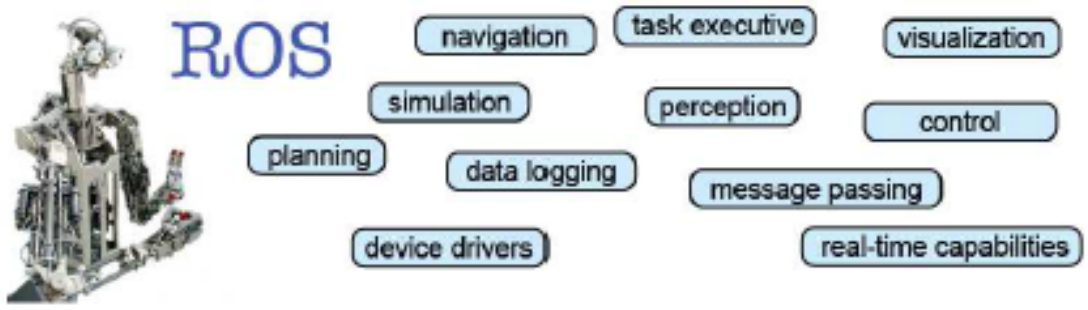
Robot Operating System (ROS) 4

4.1 Robot framework

Ros står for Robot Operatin System, som er et open-source system. Det kan være svært at skrive software programmer for robotter, da udviklingen af robotter er stigende og de kommer i forskellige former og størrelser. De forskellige robotter kan have varierende hardware, hvilket resulterer i at genbrug af software ikke altid er en mulighed. Derfor har robotforsker/ingeniør igennem tiden udviklet mange forskellige framworks, som kan håndtere de enkelte robotter. Da dette er ekstremt uhensigtsmæssig og kan føre til at skulle omskrive kode igen og igen, har robotingeniør udviklet et framework, som kan håndtere disse udfordringer inden for robot området. ROS frameworket blev udviklet i 2007, og er et samlet produkt af kompromiser og prioriteringer, som blev valgt i desginfasen. Selvom ROS framewroket blev udviklet som en samlet løsning, så har frameworket dets begrænsninger og derfor er det måske ikke det bedste framework til udvikling af software for robotter. Dokumentation " ROS: an open-source Robot Operating System" mener, at der ikke nødvendigvis findes et framework, som er det bedste for hver enkelt robot. ROS bliver hele tiden forbedret, samtidig med at der kommer flere robotter til, og derfor er det et aldrig færdig framework.

4.2 Grundlæggende principper

Filosofien for ROS og de mest almindelige funktioner er vist på figuren nedenfor.



Figur 4.1. ROS design kriterier

Frameworket er designet efter nedenstående kriterier.

- Peer-to-peer
- Tools-based
- Multi-lingual
- Thin
- Free and Open-Source...

Metode 5

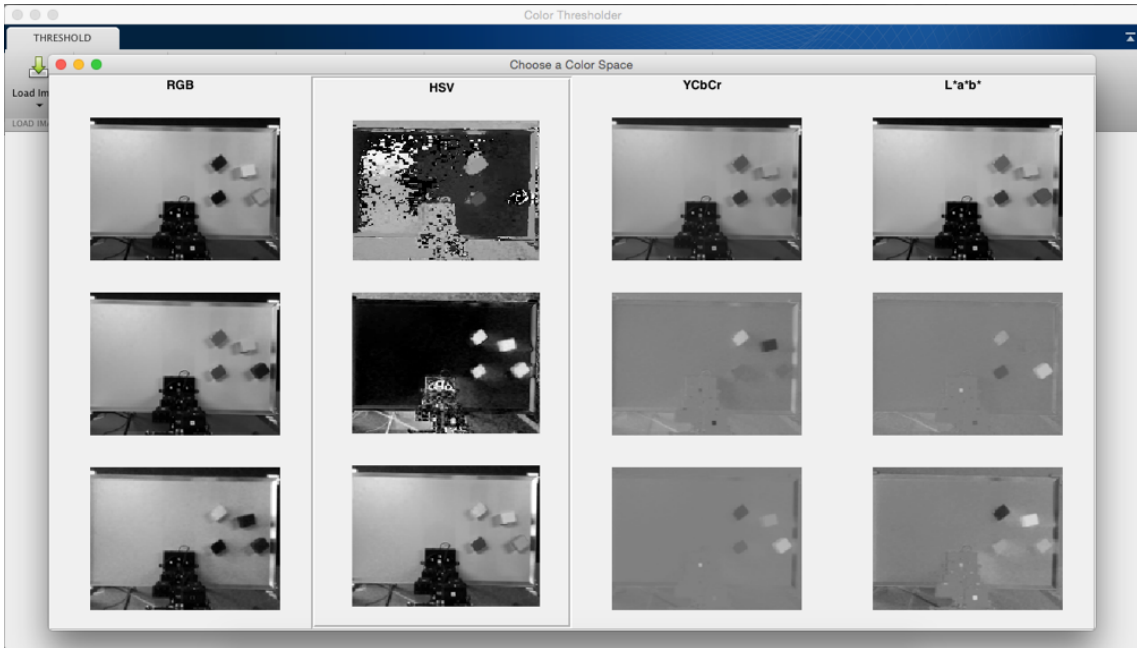
5.1 Vision system

Overordnet beskrivelse af vision systemet.

5.1.1 MATLAB - Color Thresholder

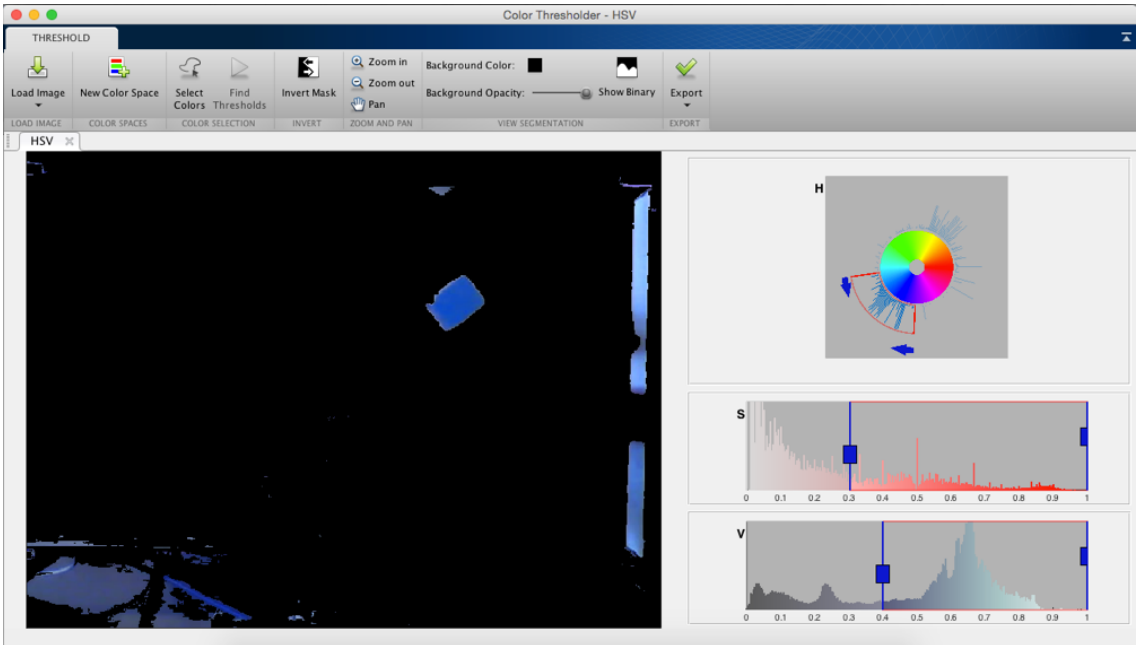
MATLAB's applikation Color Thresholder er blevet brugt til at bestemme, hvilket color space og hvilke farvegrænseværdier, der er mest optimale til at identificere de farvede klodser med webcamet.

I Color Thresholder indlæses et billede, taget med webcamet, hvorpå en klods af hver farve er repræsenteret. Valget af color space baseres på hvor klodserne adskiller sig mest fra bordpladen. På figur 5.1 ses det, at der i color spacet HSV er størst kontrast mellem klodser og bordplade. Det vælges derfor at arbejde i color spacet HSV.



Figur 5.1. MATLAB's applikation Color Thresholder til identifikation af optimalt color space.

Color Thresholder bruges derefter til at finde grænseværdierne til hhv. H, S og V kanalerne, til identifikation af de forskelligt farvede klodser. Figur 5.2 på næste side er et eksempel på grænseværdierne til identifikation af den blå klods.



Figur 5.2. Identifikation af blå klods i MATLAB's applikation Color Thresholder i color space HSV

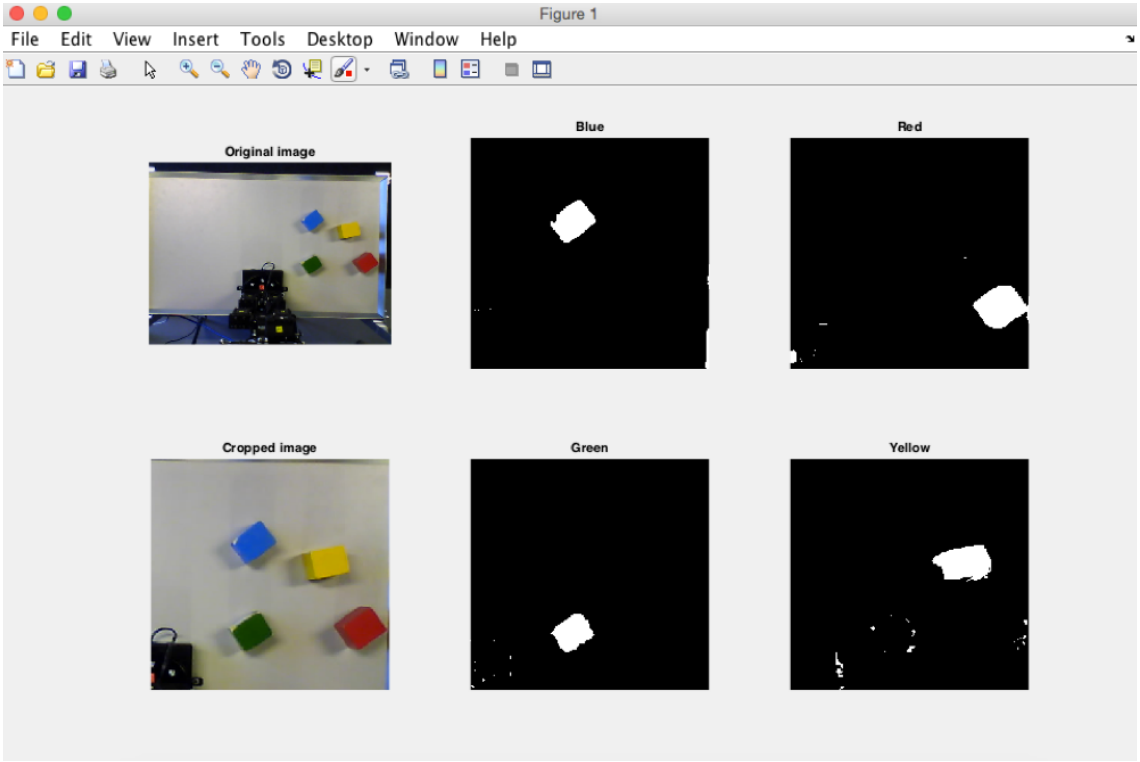
For hver farve blev følgende grænseværdier identificeret (se bilag A på side 12 for funktioner til identifikation af hver farve):

Farve	H	S	V
Blå	0.515-0.790	0.300-1.000	0.400-1.000
Rød	0.900-0.080	0.300-1.000	0.000-1.000
Grøn	0.200-0.415	0.300-1.000	0.000-1.000
Gul	0.115-0.210	0.300-1.000	0.400-1.000

Tabel 5.1. Grænseværdier til identifikation af de fire farver, blå, rød, grøn og gul. MATLAB range 0-1.

Det ses på figur 5.2, at nogen områder uden for bordpladen ikke forsvinder helt ved denne segmentering, da de har farver, der ligger tæt på om den blå klods. Dette ville løses ved at beskære billedet således, at kun bordpladen vises på billedet. Derudover skal der kun identificeres klodser på højre side af billedet, hvorfor billedet beskæres yderligere, så kun det interessante område er vises.

Figur 5.3 på den følgende side viser resultatet på MATLAB analysen, som beskærer billedet og identificerer klodser i de fire farver, blå, rød, grøn og gul. Se bilag A på side 12 for script og funktioner brugt i analysen. Det ses på figur 5.3 på den følgende side, at det vil være nødvendigt at implementerer en grænseværdi for størrelsen af klodserne, da der er små områder der ikke er forsvundet helt ved segmenteringen.



Figur 5.3. Resultat af MATLAB analysen

5.1.2 OpenCV vision

Resultater 6

7 Diskussion

Konklusion 8

MATLAB Vision A

A.1 VisionSimulation.m

```
image = imread('image3.jpg'); % Read image

subplot(2,3,1)
imshow(image);
title('Original image');

imageCrop = imcrop(image,[300 45 310 300]); % Crop image

subplot(2,3,4)
imshow(imageCrop);
title('Cropped image');

imageBlue = findBlue(imageCrop); % findBlue function created by Color
 Thresholder App
subplot(2,3,2)
imshow(imageBlue);
title('Blue');

imageRed = findRed(imageCrop); % findRed function created by Color
 Thresholder App
subplot(2,3,3)
imshow(imageRed);
title('Red');

imageGreen = findGreen(imageCrop); % findGreen function created by
 Color Thresholder App
subplot(2,3,5)
imshow(imageGreen);
title('Green');

imageYellow = findYellow(imageCrop); % findYellow function created by
 Color Thresholder App
subplot(2,3,6)
imshow(imageYellow);
title('Yellow');
```

Figur A.1. MATLAB kode for scriptet `VisionSimulation`.

A.2 findBlue

```
function [BW,maskedRGBImage] = findBlue(RGB)
% Auto-generated by colorThresholder app on 05-Oct-2016
%------------------------------------------------------
% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.515; %Conversion to openCV range channel1Min*180
channel1Max = 0.790; %Conversion to openCV range channel1Max*180

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.300; %Conversion to openCV range channel2Min*255
channel2Max = 1.000; %Conversion to openCV range channel2Max*255

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.400; %Conversion to openCV range channel3Min*255
channel3Max = 1.000; %Conversion to openCV range channel3Min*255

% Create mask based on chosen histogram thresholds
BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
```

Figur A.2. MATLAB kode for funktionen `findBlue`.

A.3 findRed

```
function [BW,maskedRGBImage] = findRed(RGB)
% Auto-generated by colorThresholder app on 05-Oct-2016
%------------------------------------------------------
% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.900; %Conversion to openCV range channel1Min*180
channel1Max = 0.080; %Conversion to openCV range channel1Max*180

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.300; %Conversion to openCV range channel2Min*255
channel2Max = 1.000; %Conversion to openCV range channel2Max*255

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.0; %Conversion to openCV range channel3Min*255
channel3Max = 1.000; %Conversion to openCV range channel3Min*255

% Create mask based on chosen histogram thresholds
BW = ( (I(:,:,1) >= channel1Min) | (I(:,:,1) <= channel1Max) ) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
```

Figur A.3. MATLAB kode for funktionen `findRed`.

A.4 findGreen

```
function [BW,maskedRGBImage] = findGreen(RGB)
% Auto-generated by colorThresholder app on 05-Oct-2016
%------------------------------------------------------
% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.200; %Conversion to openCV range channel1Min*180
channel1Max = 0.415; %Conversion to openCV range channel1Max*180

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.300; %Conversion to openCV range channel2Min*255
channel2Max = 1.000; %Conversion to openCV range channel2Max*255

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.000; %Conversion to openCV range channel3Min*255
channel3Max = 1.000; %Conversion to openCV range channel3Min*255

% Create mask based on chosen histogram thresholds
BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
```

Figur A.4. MATLAB kode for funktionen `findGreen`.

A.5 findYellow

```
function [BW,maskedRGBImage] = findYellow(RGB)
% Auto-generated by colorThresholder app on 05-Oct-2016
%------------------------------------------------------
% Convert RGB image to chosen color space
I = rgb2hsv(RGB);

% Define thresholds for channel 1 based on histogram settings
channel1Min = 0.115; %Conversion to openCV range channel1Min*180
channel1Max = 0.210; %Conversion to openCV range channel1Max*180

% Define thresholds for channel 2 based on histogram settings
channel2Min = 0.300; %Conversion to openCV range channel2Min*255
channel2Max = 1.000; %Conversion to openCV range channel2Max*255

% Define thresholds for channel 3 based on histogram settings
channel3Min = 0.403; %Conversion to openCV range channel3Min*255
channel3Max = 1.000; %Conversion to openCV range channel3Min*255

% Create mask based on chosen histogram thresholds
BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
    (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
    (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);

% Initialize output masked image based on input image.
maskedRGBImage = RGB;

% Set background pixels where BW is false to zero.
maskedRGBImage(repmat(~BW,[1 1 3])) = 0;
```

Figur A.5. MATLAB kode for funktionen `findYellow`.