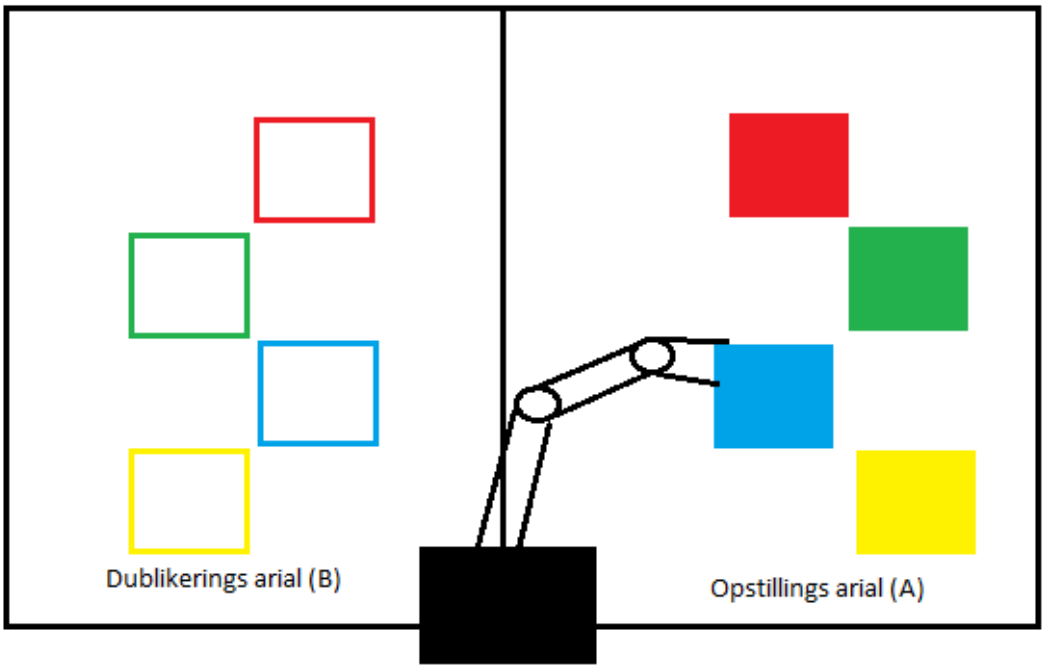
Indholdsfortegnelse

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.



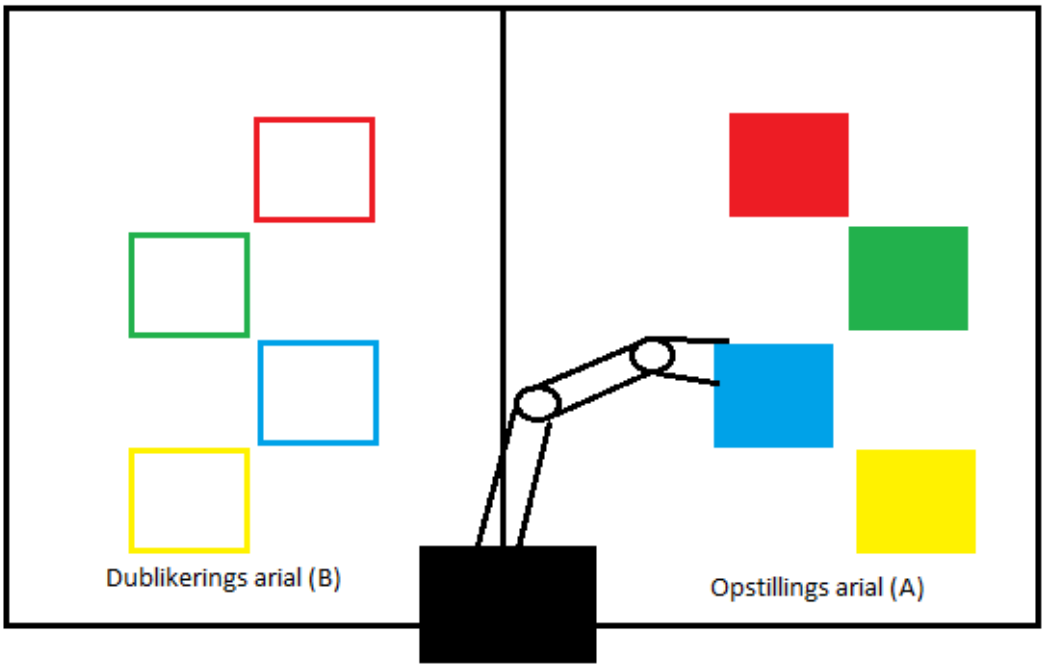
Figur 2.1. Robot Figur

Systembeskrivelse 3

Nedenstående beskriver et system, Pattern Mirroring Device, herefter benævnt PMD, bestående af en robotarm, der anvendes til at konstruere et spejlbillede af et eksisterende mønster bestående af farvede klodser.

Før spejling klargøres et mønster bestående af klodser med klare, ensartede, farver på det dertilhørende opstillingsareal (A). PMD vil derefter påbegynde duplikeringen af det fremlagte mønster, blot spejlvendt, på det tilstødende duplikeringsareal (B).

Ved færdig spejling vil PMD bevæge sig til en neutral position og påbegynde det hard-codede afslutningssignal. Brugerens hånd placeres, med håndfladen opad, i midten af PMD's arbejdsområde. PMD vil bevæge sig mod hånden. Ved kontakt mellem bruger of PMD signalerer korrekt spejlning. PMD returnerer herefter til udgangspositionen.



Figur 3.1. Systemillustration

3.1 Hardware

Systemet indeholder vision recognition i form af et fast monteret kamera, der anvendes til at analysere det eksisterende mønster og til at udrssegne de påkrævede koordinater, som

PMD skal navigere til.

Udover dette anvendes en strain gauger til at detektere, hvorvidt et objekt er holdt fast af robotten ved at måle det tryk, som denne udøver på den givne overflade. strain gauge er tilsluttet en arduino microcontoller.

Robottens eksisterende aktuator anvendes til at manipulere de påkrævede objekter.

3.2 Software

Strain gauge bliver implementeret i en arduino microcontroller. For at denne kan kommunikere med resten af system er biblioteket ROS Seriel blevet brugt, til at lave en publisher. sproget er adruinos subset af C.

Programmet til at controllere robotten er skrevet i python hvor ROS PY biblioteket er blevet brugt. Robotten bliver styret fra en virtuelmaskine i et linux miljø.

Robot Operating System (ROS) 4

4.1 Robot framework

4.1.1 Introduction

Metode 5

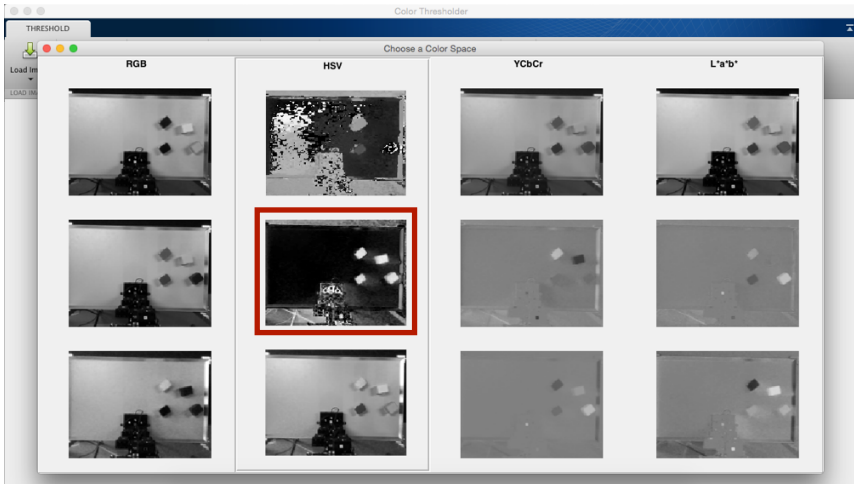
5.1 Vision system

Vision systemet i dette projekt har til formål at finde positionen af hver klods, der skal flyttes af CrustCrawleren. Dette gøre ved brug af billeder taget med et webcam monteret ca. 1 m over det bord, CrustCrawleren er monteret på. Dette gøres ved at oprette en maske, der fjerner alt på billederne undtagen klodserne der skal identificeres. Masken oprettes ved at lede efter farver inden for fastsatte grænseværdier og fjerne farver der ikke ligger inden for disse. MATLAB applikationen Color Thresholder bruges til at bestemme hvilket colorspace der skal arbejdet i og hvilke grænseværdier der identificerer klodserne. Til udvikling af vision noden i Python bruges funktioner fra OpenCV-Python, som indeholder algoritmer relateret til computer vision.

5.1.1 MATLAB - Color Thresholder

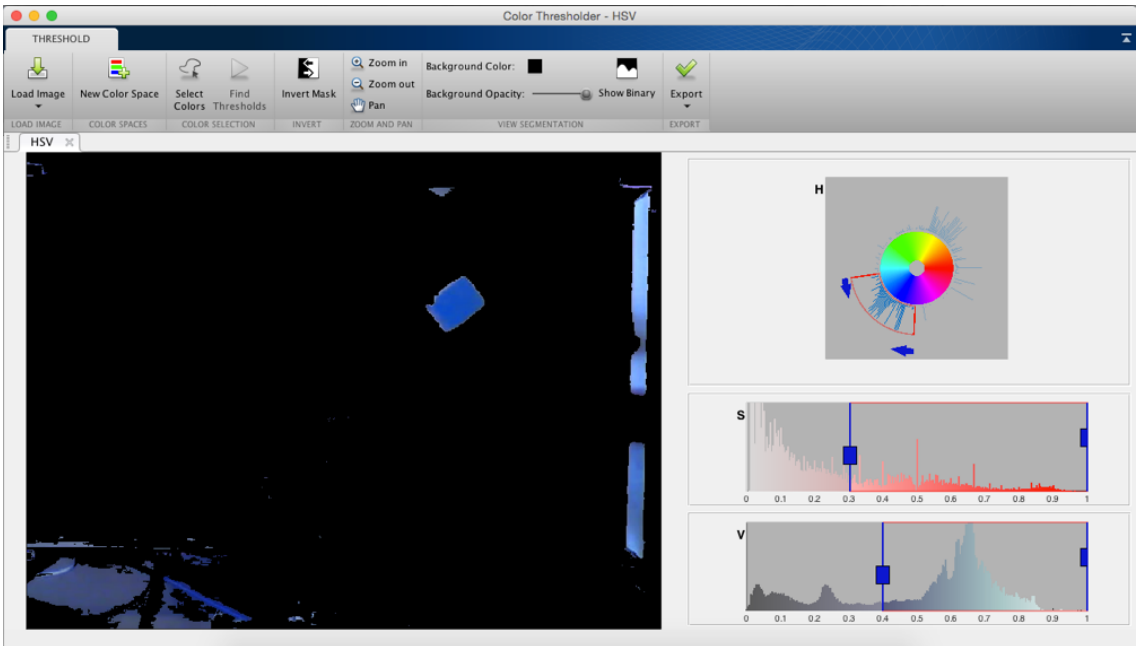
MATLAB's applikation Color Thresholder er blevet brugt til at bestemme, hvilket color space og hvilke grænseværdier, der er mest optimale til at identificere de farvede klodser med webcamet.

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, da bordpladen skal fjernes med den oprettede maske. På figur 5.1 ses det, at der i color spacet HSV (Hue, Saturation, Value) 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. Den røde firkant viser, hvor der er stor kontrast mellem klodser og bordplade.

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 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.

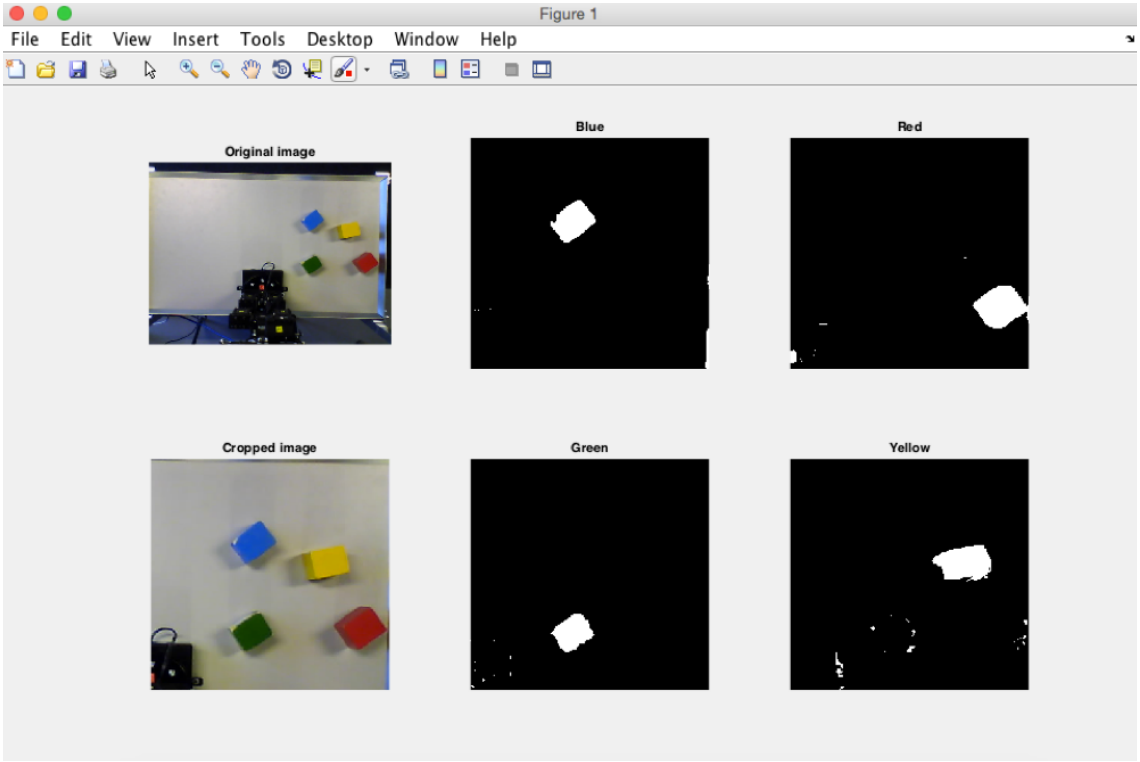
For hver farve blev følgende grænseværdier identificeret (se bilag A på side 14 for funktioner til identifikation af hver farve):

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 nogle områder uden for bordpladen ikke forsvinder helt ved denne segmentering, da de har farver, der ligger tæt på 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 CrustCrawleren, 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 14 for script og funktioner brugt i analysen. Det ses også på figur 5.3, at det vil være nødvendigt at implementere en grænseværdi for hvor store områder, der skal identificeres som klodser, da der er små områder, der ikke forsvinder helt ved segmenteringen.



Figur 5.3. Resultat af MATLAB analysen

5.1.2 Vision med OpenCV-Python

Vision noden er udviklet i Python med det formål at identificere klodsernes placering. Dette gøres ved at finde centrum af de klodser, der identificeres ved brug af en oprettet maske. Den identificerer klodserne én farve af gangen og samler til sidst centrum for alle fundne klodser i et array.

I vision noden findes 6 funktioner; `find_brick_centers`, `get_from_webcam`, `extract_single_color_range`, `threshold_image`, `contours` og `get_centers`.

`find_brick_centers` er main funktionen til at finde centrum af klodserne, se bilag B på side 17 for fuld Python kode for noden. Funktionen starter med at definerer grænseværdierne for farverne. OpenCV-Python's range i HSV color spacet er ikke det samme som MATLAB's, hvorfor værdierne fundet i MATLAB skal konverteres:

H	0-179
S	0-255
V	0-255

Tabel 5.2. Range for kanalerne H, S og V i OpenCV-Python

Farve	H	S	V
Blå	92-141	76-255	102-255
Rød	161-14	76-255	0-255
Grøn	36-74	76-255	0-255
Gul	21-38	76-255	102-255

Tabel 5.3. Grænseværdier konverteret fra MATLAB range til OpenCV-Python range og afrundet til heltal.

Derudover skal H grænseværdierne for den røde klods deles op i to intervaller; 0-14 og 161-179.

Efter at have defineret grænseværdierne kaldes metoden `get_from_webcam`, hvilken indlæser et billede fra webcamet og beskærer det så det kun er det relevante område der vises på billedet. Funktionen returnerer det beskårede billede.

Billedet fra `get_from_webcam` konverteres fra color spacet BGR til HSV ved brug af OpenCV-Python funktionen `cv2.cvtCOLOR`. Efter konverteringen kaldes funktionerne `extract_single_color_range`, `threshold_image`, `contours` og `get_centers` på billedet for hver farve.

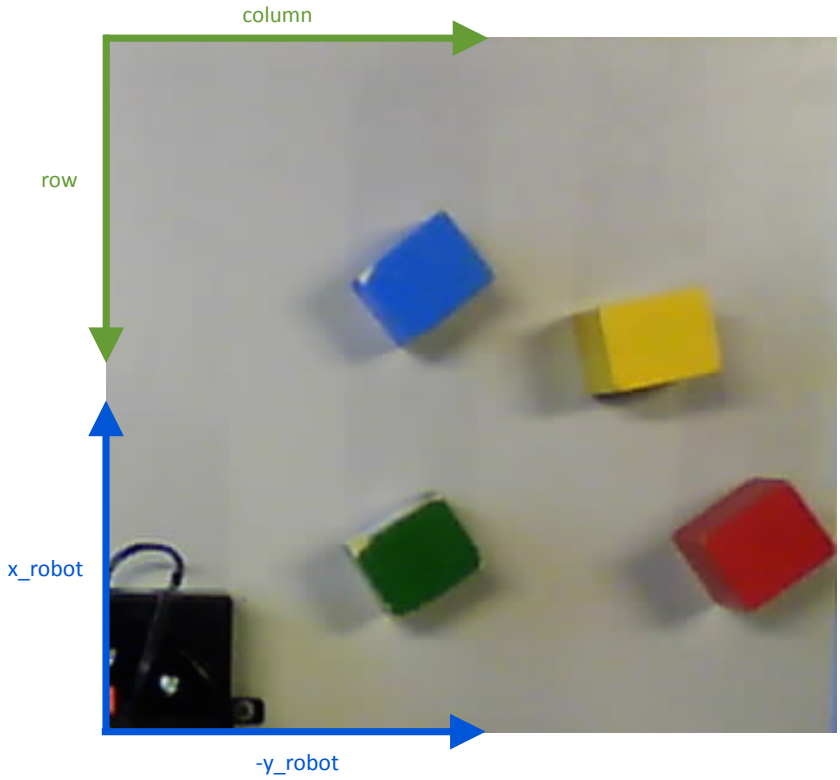
`extract_single_color_range` opretter en maske fjerner alle farver på billedet som ikke ligger i den specificerende grænseværdi, se bilag C afsnit C.1 på side 20 for princippet bag en maske. Maksen oprettes ved brug af OpenCV-Pyhton funktionen `cv2.inRange`. Funktionen returnerer billedet, hvor kun farverne inden for den specificerede grænseværdi er synlige.

`threshold_image` thresholder billedet med OpenCV funktionen `cv2.threshold` og udfører morfologiske operationer på det. Den udfører en *dialate* med OpenCV funktionen `cv2.dilate` for at lukke små huller og en close, med OpenCV funktionen `cv2.morphologyEx`, for at lukke evt. brudte kanter.

`contours` konverterer det returnerede billede fra `threshold_image` til et gråskala-billede ved brug af OpenCV-Python funktionen `cv2.cvtColor`. Derefter identificerer den konturer på billedet ved brug af OpenCV-Python funktionen `cv2.findContours`. Funktionen returnerer disse konturer.

`get_centers` finder for hver kontur en firkantet repræsentation af konturen ved brug af OpenCV-Python funktionerne `cv2.arcLength` og `cv2.approxPolyDP` og finder derefter arealet af disse ved brug af OpenCV-Python funktionen `cv2.contourArea`. Hvis dette areal er større end 500 pixels ses den som en klods og et vægtet gennemsnit af områdets pixels (momentet) findes ved brug af OpenCV-funktionen cv2.moments. Centrum koordinaterne (række og kolonne i billedet) for momentet findes og returneres.

Da koordinatsættet, returneret fra `get_centers`, er i pixels, en række og en kolonne i billedet, skal det konverteres til koordinater i CrustCrawlerens koordinatsystem. Se figur 5.4 på den følgende side for en illustration af de to koordinatsystemer.



Figur 5.4. CrustCrawlerens koordinatsystem i forhold til række og kolonne koordinatsystemet for billedet.

For at identificere hvor mange pixels der er pr. enhed i CrustCrawlerens koordinatsystem udføres en test hvor klodser sættes i et kendte punkter i CrustCrawlerens koordinatsystem. Derefter tages et billede med webcamet og klodsernes koordinater i pixels identificeres via. vision algoritmerne. Se bilag D på side 21 for den fulde test. Testen viste at der var 9 pixels pr. enhed i CrustCrawlerens koordinatsystem. "x_robot" aksen og "row" aksen vender hver sin vej og har origo i hver sit hjørne af billedet. Derfor trækkes pixel koordinatets rækkeværdi fra højden af billedet i pixels før det divideres med de 9 pixels pr. enhed for at vende aksen om. Dermed returnerer `find_brick_centers` centrum i CrustCrawlerens koordinatsystem for alle klodser identificeret af vision aloritmerne.

Resultater 6

7 Diskussion

8 Konklusion

MATLAB Vision A

A.1 VisionSimulation.m

```
%% Vision simulation in Matlab
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');
```

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*179
channel1Max = 0.790; %Conversion to openCV range channel1Max*179

% 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;
```

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*179
channel1Max = 0.080; %Conversion to openCV range channel1Max*179

% 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;
```

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*179
channel1Max = 0.415; %Conversion to openCV range channel1Max*179

% 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;
```

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*179
channel1Max = 0.210; %Conversion to openCV range channel1Max*179

% 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;
```

Vision Node B

```
#!/usr/bin/env python
import cv2
import urllib
import numpy as np
import math

def get_from_webcam():
    """
    Fetches an image from the webcam
    """
    print "try fetch from webcam..."
    stream=urllib.urlopen('http://192.168.0.20/image/jpeg.cgi')
    bytes=''
    bytes+=stream.read(64500)
    a = bytes.find('\xff\xd8')
    b = bytes.find('\xff\xd9')

    if a != -1 and b != -1:
        jpg = bytes[a:b+2]
        bytes= bytes[b+2:]
        i = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8),cv2.CV_LOAD_IMAGE_COLOR)
        i_crop = i[55:350, 300:610]
        return i_crop
    else:
        print "did not receive image, try increasing the buffer size in line 13:"

def extract_single_color_range(image,hsv,lower,upper):
    """
    Calculates a mask for which all pixels within the specified range is set to 1
    the ands this mask with the provided image such that color information is
    still present, but only for the specified range
    """
    if len(lower) == 2 and len(upper) == 2:
        mask0 = cv2.inRange(hsv, lower[0], upper[0])
        mask1 = cv2.inRange(hsv, lower[1], upper[1])
        mask = mask0+mask1
    else:
        mask = cv2.inRange(hsv, lower, upper)
    res = cv2.bitwise_and(image,image, mask= mask)
    return res

def threshold_image(image):
    """
    Thresholds the image within the desired range and then dilates with a 3x3 matrix
    such that small holes are filled. Afterwards the 'blobs' are closed using a
    combination of dilate and erode
    """
    ret,th1 = cv2.threshold(image,50,255,cv2.THRESH_BINARY)
    resdi = cv2.dilate(th1,np.ones((3,3),np.uint8))
    closing = cv2.morphologyEx(resdi, cv2.MORPH_CLOSE,np.ones((5,5),np.uint8))

    return closing

```

```
54  def contours(image):
55      """
56      Extract the contours of the image by first converting it to grayscale and then
57      call findContours
58      """
59      imgray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
60      contours, hierarchy = cv2.findContours(imgray,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
61
62      return contours
63
64  def get_centers(contours):
65      """
66      For each contour in contours
67          approximate the contours such that small variations are removed
68          calulate the area of the contour
69          if the area is within the desired range we append the box points to the
70          bricks.
71      """
72      centers = []
73      for cnt in contours:
74          epsilon = 0.1*cv2.arcLength(cnt,True)
75          approx = cv2.approxPolyDP(cnt,epsilon,True)
76          area = cv2.contourArea(approx)
77
78          if area > 500:
79              moments = cv2.moments(cnt)
80              centers.append((int(moments['m10']/moments['m00']),
                    int(moments['m01']/moments['m00'])))
81
82      return centers
83
84
85  def find_brick_centers():
86      lower_blue = np.array([92,76,103])
87      upper_blue = np.array([141,255,255])
88
89      lower_green = np.array([36,76,0])
90      upper_green = np.array([74,255,255])
91
92      lower_yellow = np.array([21,76,103])
93      upper_yellow = np.array([38,255,255])
94
95      lower_red = np.array([np.array([0,76,103]),np.array([161,76,103])])
96      upper_red = np.array([np.array([14,255,255]),np.array([179,255,255])])
97
98      image = get_from_webcam()
99      hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
100
101     single_color_img_blue = extract_single_color_range(image,hsv,lower_blue,upper_blue)
102     single_color_img_green =
            extract_single_color_range(image,hsv,lower_green,upper_green)
103     single_color_img_yellow =
            extract_single_color_range(image,hsv,lower_yellow,upper_yellow)
104     single_color_img_red = extract_single_color_range(image,hsv,lower_red,upper_red)
105
106     single_channel_blue = threshold_image(single_color_img_blue)
107     single_channel_green = threshold_image(single_color_img_green)
108     single_channel_yellow = threshold_image(single_color_img_yellow)
109     single_channel_red = threshold_image(single_color_img_red)
110
111     cont_blue = contours(single_channel_blue)
112     cont_green = contours(single_channel_green)
113     cont_yellow = contours(single_channel_yellow)
114     cont_red = contours(single_channel_red)
115
116     centers_blue = get_centers(cont_blue)
```

```
117     centers_green = get_centers(cont_green)
118     centers_yellow = get_centers(cont_yellow)
119     centers_red = get_centers(cont_red)
120 
121     centers_cm = []
122     for c in centers_blue:
123        x_koor = float((295.0-c[1])/9.0)
124        y_koor = float(-(c[0])/9.0)
125        centers_cm.append(x_koor)
126        centers_cm.append(y_koor)
127 
128     for c in centers_green:
129        x_koor = float((295.0-c[1])/9.0)
130        y_koor = float(-(c[0])/9.0)
131        centers_cm.append(x_koor)
132        centers_cm.append(y_koor)
133 
134     for c in centers_yellow:
135        x_koor = float((295.0-c[1])/9.0)
136        y_koor = float(-(c[0])/9.0)
137        centers_cm.append(x_koor)
138        centers_cm.append(y_koor)
139 
140     for c in centers_red:
141        x_koor = float((295.0-c[1])/9.0)
142        y_koor = float(-(c[0])/9.0)
143        centers_cm.append(x_koor)
144        centers_cm.append(y_koor)
145 
146     return centers_cm
```

Vision principper C

C.1 Masker

Koordinatsystem konvertering D

(10,-10)(10.5,-11) (10,-15)(10.4,-15.6) (10,-20)(10,-20.7) (15,-15)(15.5,-15.8) (15,-10)(15.2,-10.6) (15,-20)(15.5,-20.7) (20,-20)(20.1,-20.7) (20,-10)(20.6,-10.3) (20,-15)(20.4,-15.4)