

Indholdsfortegnelse

Kapitel 1	Indledning	1
Kapitel 2	Opgavebeskrivelse	2
Kapitel 3	Systembeskrivelse	3
3.1	Hardware	3
3.2	Software	4
Kapitel 4	Robot Operating System (ROS)	5
4.1	Robot framework	5
4.2	Grundlæggende principper	5
Kapitel 5	Metode	8
5.1	Vision system	8
5.1.1	MATLAB - Color Thresholder	8
5.1.2	Vision med OpenCV-Python	10
Kapitel 6	Resultater	13
Kapitel 7	Diskussion	14
Kapitel 8	Konklusion	15
Appendiks A	Python Kode	16
A.1	Main	16
A.2	Vision Node	17
A.3	Vision funktioner	17
A.4	Inverse Robot	20
A.5	Strain Gauge	23
Appendiks B	MATLAB Vision	25
B.1	VisionSimulation.m	25
B.2	findBlue	26
B.3	findRed	26
B.4	findGreen	27
B.5	findYellow	27
Appendiks C	Vision principper	28
C.1	Masker	28
C.2	Morphologi	28
C.2.1	Erode	28
C.2.2	Dilate	28

Appendiks D Koordinatsystem konvertering**29**

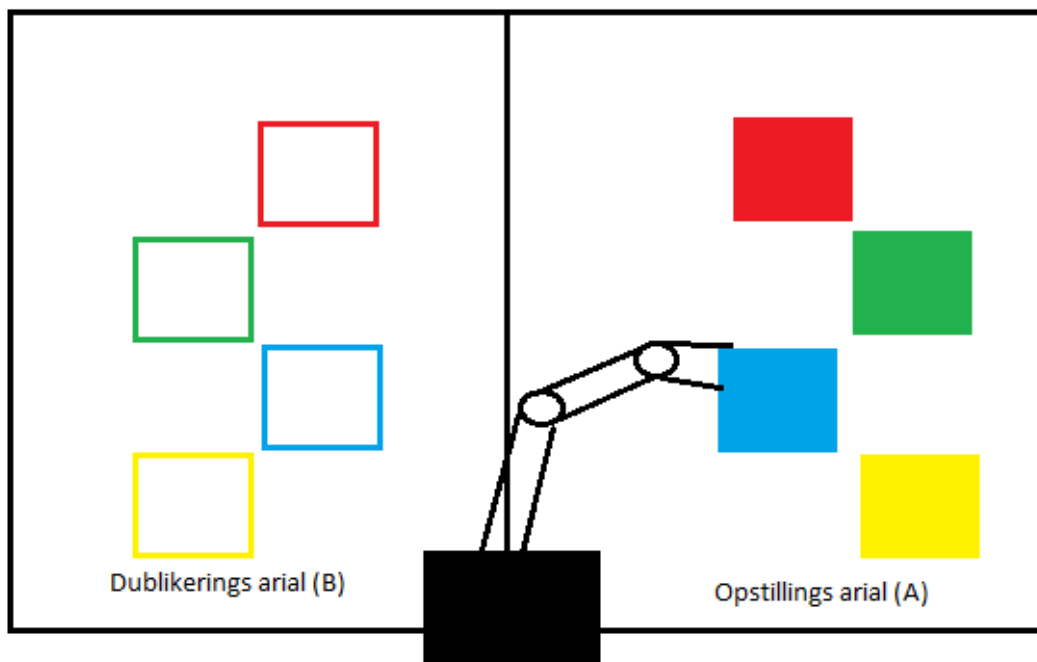
Indledning 1

I den industrielle 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 industri verden er det billigere at have en robot der kan arbejde fireogtyve timer i døgnet end et menneske, 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 virksomheder 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 separate noder, hvilket vil gøre at noget af CrustCrawlerens funktionalitet blev kørt asynkront.

Det blev bestemt af gruppen, at CrustCrawler skulle kunne flytte 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.

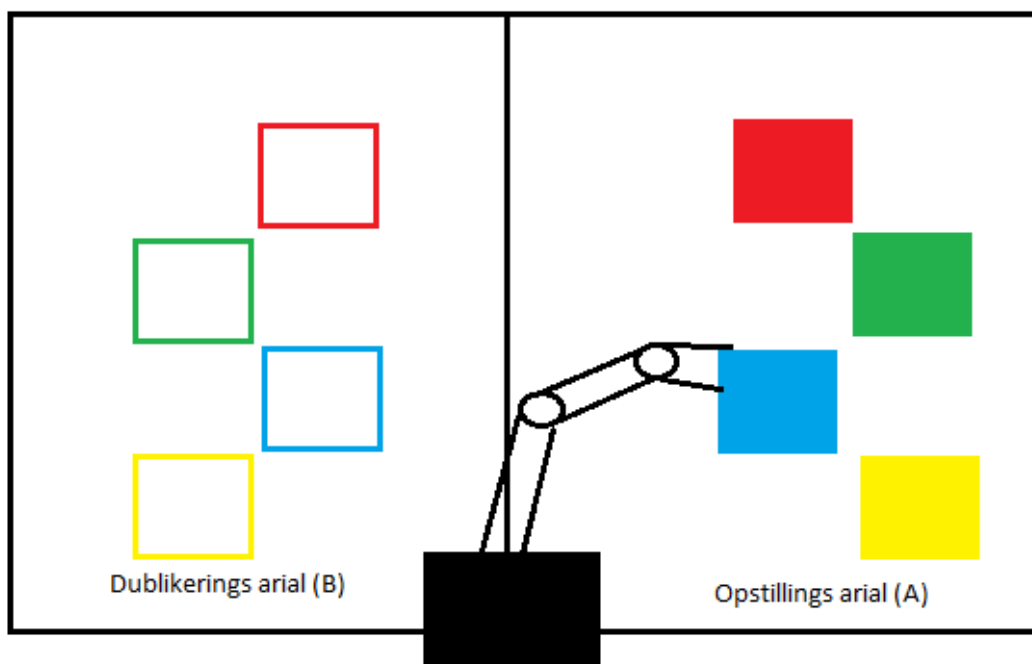


Figur 2.1. Robot Figur

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 hardcodede afslutningssignal. Brugerens hånd placeres, med håndfladen opad, i midten af PMD's arbejdsområde. PMD vil bevæge sig mod hånden. Ved kontakt mellem bruger og 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 udrække 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 microcontroller.

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 Serial blevet brugt, til at lave en publisher. sproget er adruinos subset af C.

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

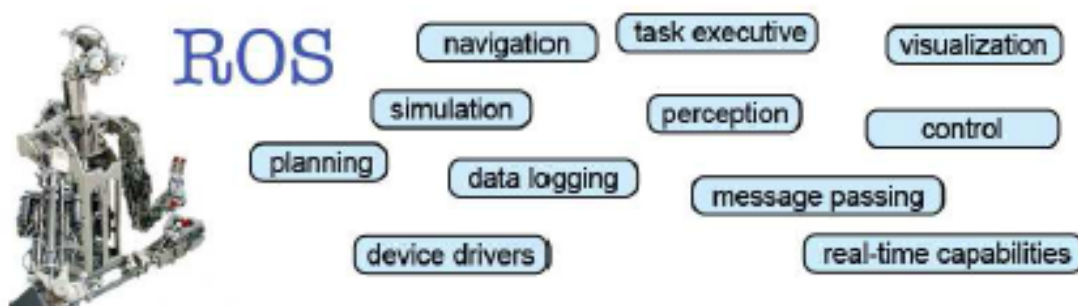
Robot Operating System (ROS) 4

4.1 Robot framework

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 frameworks, 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.

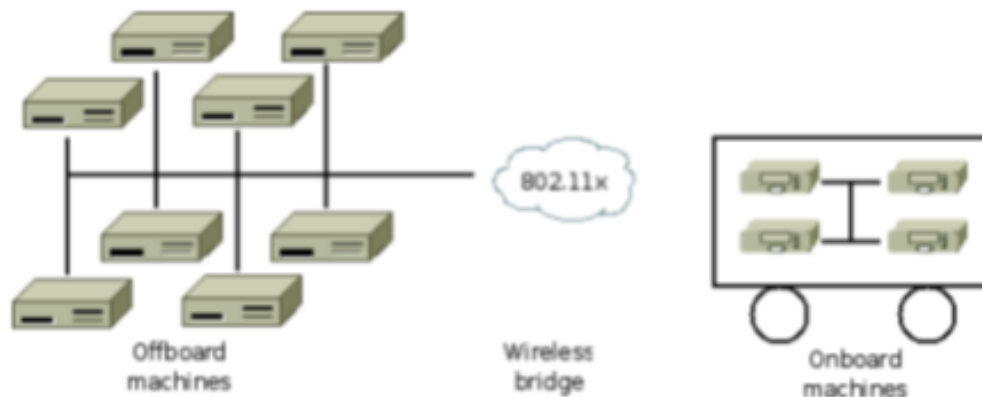


Figur 4.1. ROS design kriterier

Frameworket er designet efter nedenstående kriterier.

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

Peer to peer (P2P): Et system som benytter sig af ROS bruger en række processer på et bredt antal af forskellige host, som er forbundet runtime i en peer-to-peer topologi. De processer som bruges består i form af noder, hvor det kan opdelt således at én node udføre én handling. På store robotter, som bruges i industrien, er der typisk onboard maskiner som er forbundet igennem ethernet. Dette netværk er "bridget" (så det har mulighed for at kommunikere) gennem trådløs LAN til en offboard maskine der anvender et vision system eller voice recognition. ROS P2P har ikke en central server, og der kan det undgås at on- og offboard maskiner vil medføre stor data trafik, og dermed gøre den trådløse forbindelse langsom pga. af de informationer der bliver pushet og lageret i subnettet. På nedenstående figur ses hvordan P2P visuelt fungerer.



Figur 4.2. ROS netværks konfiguration

Tool-based: For at kunne håndtere kompleksiteten af ROS er der valgt at bruge et micro-kernel design. Designet er bygget således, at et stort nummer af mindre værktøjselementer er brugt til at bygge ROS komponenterne. Dette design er blevet valgt frem for at bruge en monolitisk udvikling og runtime miljø. Disse små værktøjer kan alt fra at navigere i source koden til at sætte konfigurations paramenterne osv.

Multi-lingual: Multi-lingual gør at ROS er et programmeringssprog, som supporterer 4 forskellige sprog. Disse sprog er C++, Python, Octave, LISP. Grundet at ROS supporterer disse forskellige programmeringssprog, gør at ROS i sig selv er let anvendeligt for de fleste programmører, og derfor taler det til et bedre publikum. For at supportere cross-language, anvender ROS en simpel sprog neutralt interface (IDL) for at beskrive beskeder, der sendes igennem modulerne. IDL anvender en kort tekst fil til at beskrive felterne af hver enkelt besked og tillader en sammensætning af beskeder. Dette ses på figuren nedenfor.


```
Header header
Point32[] pts
ChannelFloat32[] chan
```

Figur 4.3. IDE message file

Det endelige resultat kan beskrives som at ROS udgør et programmeringssprog, hvor der er mulighed for at blande de forskellige programmeringssprog på kryds og tværs som der ønskes.

Thin: Mange af de robotsoftwareprojekter består af mange forskellige drivers og algoritmer, som kan bruges uden for det egentlige projekt. Derfor er ROS designet til at være så "tyndt" som overhovedet muligt. Dette skyldes, at der ikke skal være for mange begrænsninger, som kan udelukke andre robot frameworks til at arbejde sammen med ROS. ROS arbejder derfor let med andre robotframeworks. Når der importeres ROS i ens projekt anvender ROS nogle forskellige imports af enkeltstående biblioteker, som har en minimal afhængighed til ROS. ROS genbruger kode fra de forskellige open-source projekter, som for eksempel vision algoritmen fra OpenCV.

Free and Open-Source: ROS er fri tilgængelig og kan benyttes af alle der ønsker det. Dog har ROS alligevel nogle værktøjer, som er lukket for den almen bruger, og kræver en lincens for at få adgang til.

Metode 5

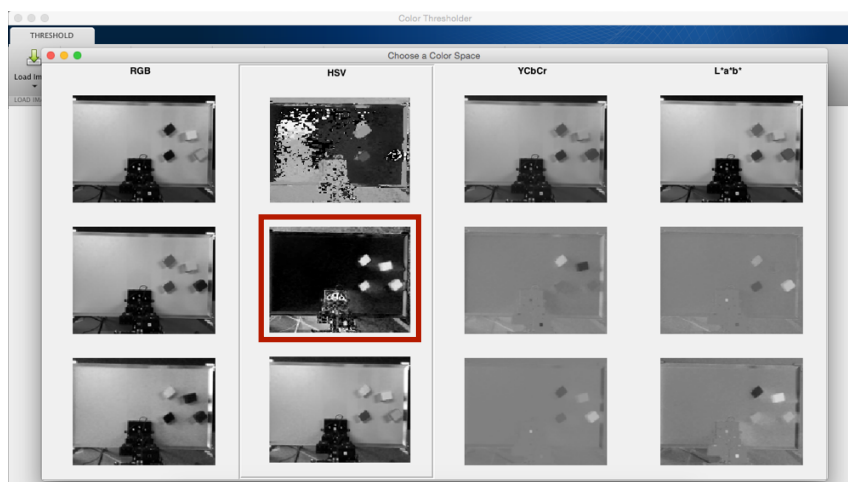
5.1 Vision system

Vision systemet i dette projekt har til formål at finde positionen af hver klods, der skal flyttes af CrustCrawleren. Dette gøres 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 arbejdes 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

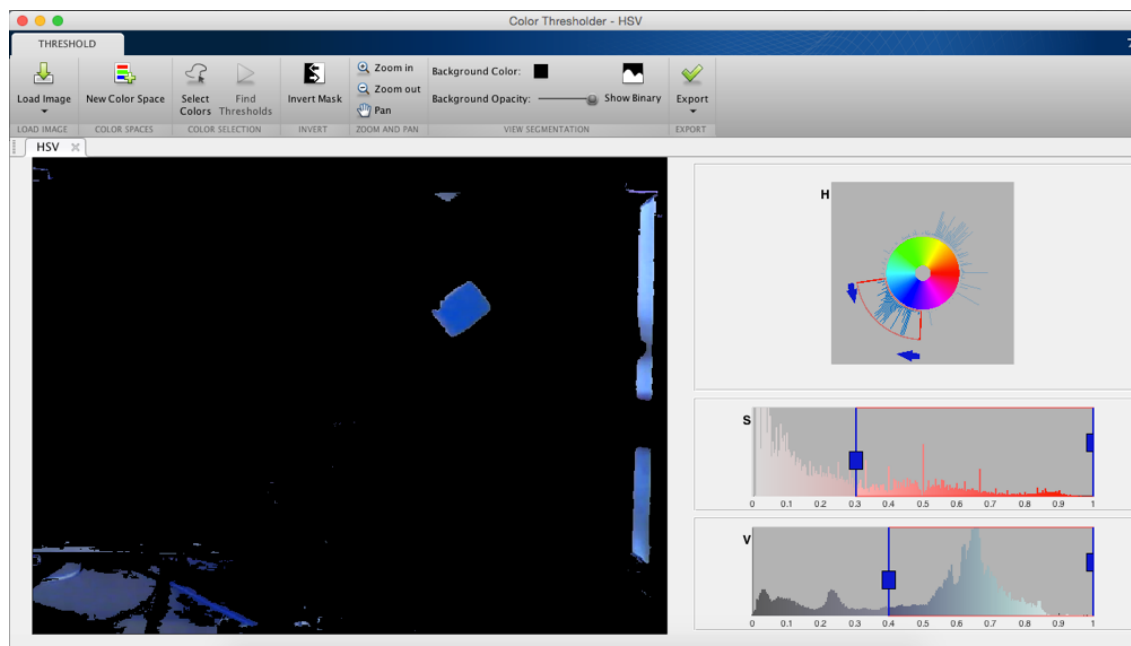
MATLAB's applikation Color Thresholder er blevet brugt til at bestemme, hvilket color space og hvilke grænseværdier, der er mest optimale til at identificere de farvede klodser med webcams.

I Color Thresholder indlæses et billede, taget med webcams, 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.

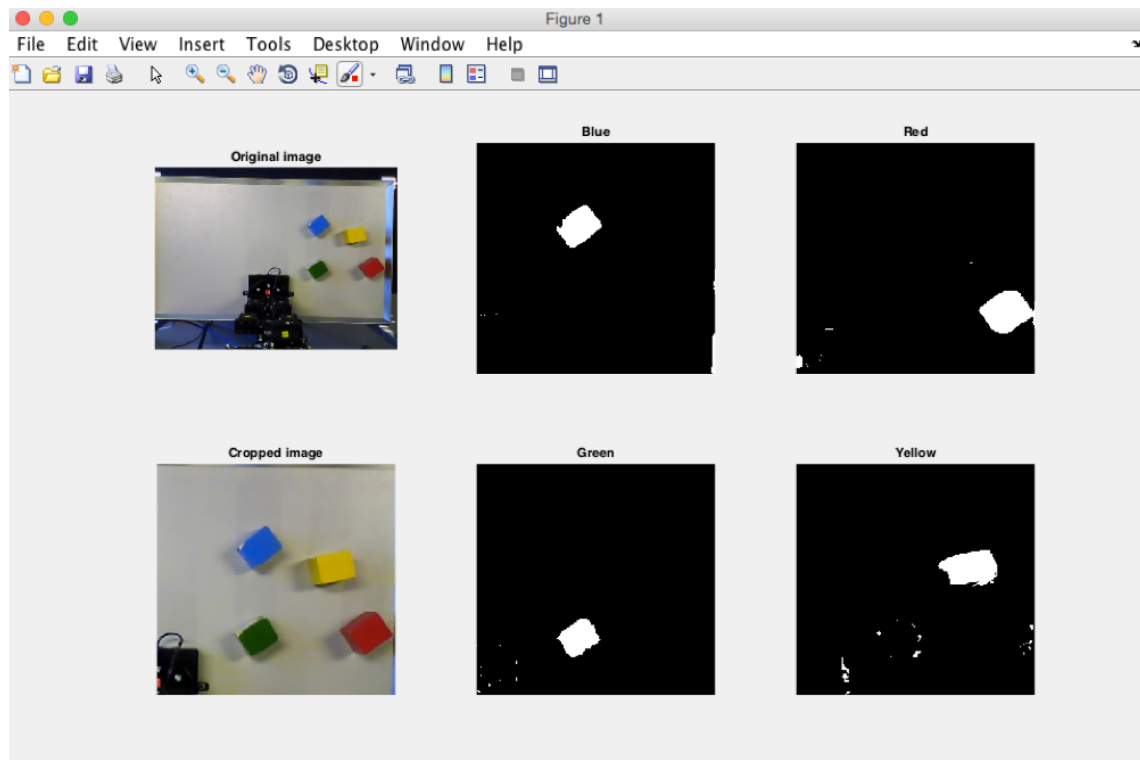
For hver farve blev følgende grænseværdier identificeret (se bilag B på side 25 for funktioner til identifikation af hver farve):

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

Tablet 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 B på side 25 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 ?? på side ?? 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 webcameraet og beskærer det så det kun er det relevante område der vises på billedet. Funktionen returnerer det beskærede billede.

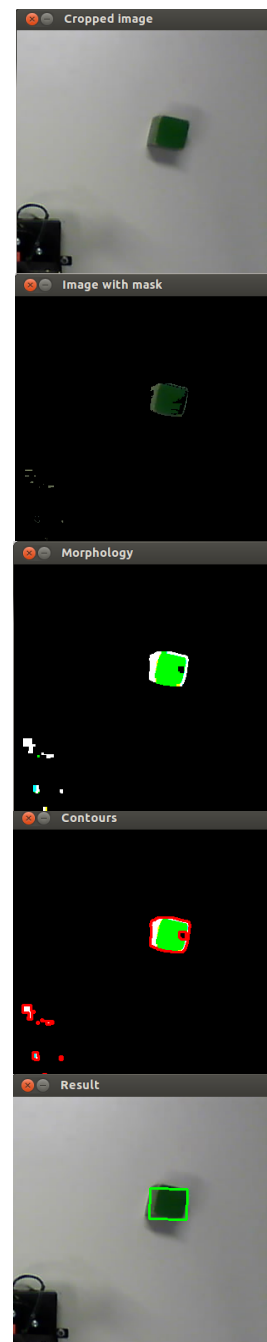
Billedet fra `get_from_webcam` konverteres fra color spacet BGR til HSV ved brug af OpenCV-Python funktionen `cv2.cvtColor`. Efter konverteringen kaldes funktionerne `extract_single_color_range`, `threshold_image`, `contours` og `get_centers` på billedet for hver farve.

`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 28 for princippet bag en maske. Maksen oprettes ved brug af OpenCV-Python 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 *dilate* 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 gråskalabillede 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.

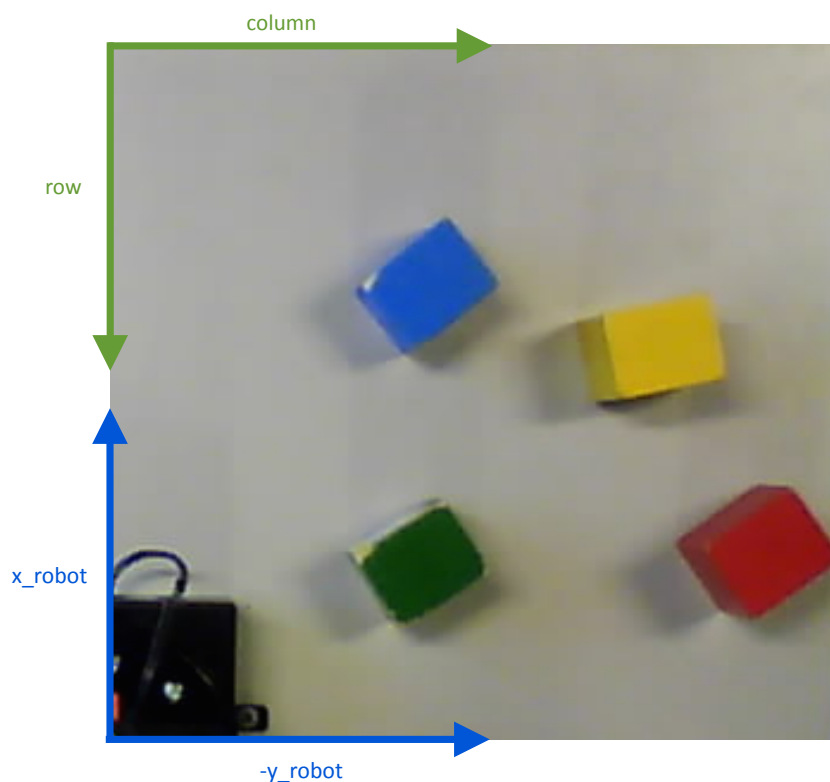
`get_centers` finder for hver kontur en firkantet repræsentation af konturen ved brug af OpenCV-Python funktionerne `cv2.arcLength` og `cv2.approxPolyDP` og finder derefter arealet af disse ved brug af OpenCV-Python funktionen `cv2.contourArea`. Hvis dette areal er større end 500 pixels



Figur 5.4. Illustration af vision algoritmen

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 5.5 for en illustration af de to koordinatsystemer.



Figur 5.5. 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 29 for den fulde test. Testen viste at der var 9 pixels pr. enhed i CrustCrawlerens koordinatsystem. "x_robot" akse og "row" akse vender hver sin vej og har origo i hver sit hjørne af billedet. Derfor trækkes pixel koordinatets række værdi fra højden af billedet i pixels før det divideres med de 9 pixels pr. enhed for at vende akse om. Dermed returnerer `find_brick_centers` centrum i CrustCrawlerens koordinatsystem for alle klodser identificeret af vision algoritmerne.

Resultater 6

Diskussion 7

Konklusion 8

Python Kode A

A.1 Main

```
1  #!/usr/bin/env python
2
3  import rospy
4  import actionlib
5  from control_msgs.msg import FollowJointTrajectoryAction
6  from control_msgs.msg import FollowJointTrajectoryFeedback
7  from control_msgs.msg import FollowJointTrajectoryResult
8  from control_msgs.msg import FollowJointTrajectoryGoal
9  from trajectory_msgs.msg import JointTrajectoryPoint
10 from trajectory_msgs.msg import JointTrajectory
11 import math
12 import time
13 from std_msgs.msg import String
14
15 from InvRobot import *
16 from findBricks import *
17
18 visionCoor = []
19
20 def getCoordinates(coord):
21     global visionCoor
22
23     coord_str = str(coord)
24     coor = coord_str.replace("data: ", "")
25     visionCoor = [float(s) for s in coor.split(',')]
26
27     if visionCoor[0] == 0 and visionCoor[1] == 0:
28         print "Nu skal programmet stoppe"
29         visionCoor = []
30         print "VisionCoor"
31         print visionCoor
32
33 def main():
34     Jobactive = True
35     Job = False
36
37     while Jobactive == True:
38         print "V3"
39         print visionCoor
40         xy = visionCoor
41         if len(xy) > 0:
42             mirrorCube(xy)
43             Job = True
44         else:
45             Jobactive = False
46             if Job == True:
47                 RobotLowFive()
48                 print "Jobs done"
49             else:
50                 print "There wasnt any job "
51
```

```

52 if __name__ == "__main__":
53     rospy.init_node("InvRobot")
54     rospy.Subscriber("Coordinates", String, getCoordinates)
55     setupGrabberPressureSensor()
56
57     top = invkin([0,0, 54.1])
58     RobotDo(top,0,0)
59     main()

```

A.2 Vision Node

```

1  #!/usr/bin/env python
2
3  import rospy
4  from findBricks import *
5  from std_msgs.msg import String
6
7  def VisionPublisher():
8
9      pub = rospy.Publisher('Coordinates', String, queue_size=10)
10     rospy.init_node('VisionPublisher', anonymous=True)
11     rate = rospy.Rate(1) # 10hz
12     while not rospy.is_shutdown():
13         try:
14             bricks = find_brick_centers()
15
16             if len(bricks) != 0:
17                 coords_str = "%f,%f"%(bricks[0], bricks[1])
18             else:
19                 coords_str = "%f,%f"%(0, 0)
20             pub.publish(coords_str)
21
22             print("Test af coords_str")
23             print(coords_str)
24         except:
25             print "Error"
26         rate.sleep()
27
28 if __name__ == '__main__':
29     try:
30         VisionPublisher()
31     except rospy.ROSInterruptException:
32         pass

```

A.3 Vision funktioner

```

1  #!/usr/bin/env python
2  import cv2
3  import urllib
4  import numpy as np
5  import math
6
7  def get_from_webcam():
8      """
9      Fetches an image from the webcam
10     """
11     print "try fetch from webcam..."
12     stream=urllib.urlopen('http://192.168.0.20/image/jpeg.cgi')
13     bytes=''
14     bytes+=stream.read(64500)
15     a = bytes.find('\xff\xd8')
16     b = bytes.find('\xff\xd9')
17
18     if a != -1 and b != -1:

```

```

19     jpg = bytes[a:b+2]
20     bytes= bytes[b+2:]
21     i = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8),cv2.CV_LOAD_IMAGE_COLOR)
22     i_crop = i[55:350, 300:610]
23     return i_crop
24 else:
25     print "did not receive image, try increasing the buffer size in line 13:"
26
27 def extract_single_color_range(image,hsv,lower,upper):
28     """
29     Calculates a mask for which all pixels within the specified range is set to 1
30     the ands this mask with the provided image such that color information is
31     still present, but only for the specified range
32     """
33     if len(lower) == 2 and len(upper) == 2:
34         mask0 = cv2.inRange(hsv, lower[0], upper[0])
35         mask1 = cv2.inRange(hsv, lower[1], upper[1])
36         mask = mask0+mask1
37     else:
38         mask = cv2.inRange(hsv, lower, upper)
39     res = cv2.bitwise_and(image,image, mask= mask)
40     return res
41
42 def threshold_image(image):
43     """
44     Thresholds the image within the desired range and then dilates with a 3x3 matrix
45     such that small holes are filled. Afterwards the 'blobs' are closed using a
46     combination of dilate and erode
47     """
48     ret,th1 = cv2.threshold(image,50,255,cv2.THRESH_BINARY)
49     resdi = cv2.dilate(th1,np.ones((3,3),np.uint8))
50     closing = cv2.morphologyEx(resdi, cv2.MORPH_CLOSE,np.ones((5,5),np.uint8))
51
52     return closing
53
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         calculate 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']),
81                             int(moments['m01']/moments['m00'])))
82
83     return centers

```

```
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 =
103         extract_single_color_range(image,hsv,lower_green,upper_green)
104     single_color_img_yellow =
105         extract_single_color_range(image,hsv,lower_yellow,upper_yellow)
106     single_color_img_red = extract_single_color_range(image,hsv,lower_red,upper_red)
107
108     single_channel_blue = threshold_image(single_color_img_blue)
109     single_channel_green = threshold_image(single_color_img_green)
110     single_channel_yellow = threshold_image(single_color_img_yellow)
111     single_channel_red = threshold_image(single_color_img_red)
112
113     cont_blue = contours(single_channel_blue)
114     cont_green = contours(single_channel_green)
115     cont_yellow = contours(single_channel_yellow)
116     cont_red = contours(single_channel_red)
117
118     centers_blue = get_centers(cont_blue)
119     centers_green = get_centers(cont_green)
120     centers_yellow = get_centers(cont_yellow)
121     centers_red = get_centers(cont_red)
122
123     centers_cm = []
124     for c in centers_blue:
125         x_koor = float((295.0-c[1])/9.0)
126         y_koor = float(-(c[0])/9.0)
127         centers_cm.append(x_koor)
128         centers_cm.append(y_koor)
129
130     for c in centers_green:
131         x_koor = float((295.0-c[1])/9.0)
132         y_koor = float(-(c[0])/9.0)
133         centers_cm.append(x_koor)
134         centers_cm.append(y_koor)
135
136     for c in centers_yellow:
137         x_koor = float((295.0-c[1])/9.0)
138         y_koor = float(-(c[0])/9.0)
139         centers_cm.append(x_koor)
140         centers_cm.append(y_koor)
141
142     for c in centers_red:
143         x_koor = float((295.0-c[1])/9.0)
144         y_koor = float(-(c[0])/9.0)
145         centers_cm.append(x_koor)
146         centers_cm.append(y_koor)
147
148     return centers_cm
```

A.4 Inverse Robot

```

1  #!/usr/bin/env python
2
3  import rospy
4  import actionlib
5  from control_msgs.msg import FollowJointTrajectoryAction
6  from control_msgs.msg import FollowJointTrajectoryFeedback
7  from control_msgs.msg import FollowJointTrajectoryResult
8  from control_msgs.msg import FollowJointTrajectoryGoal
9  from trajectory_msgs.msg import JointTrajectoryPoint
10 from trajectory_msgs.msg import JointTrajectory
11 import math
12 import time
13 from std_msgs.msg import Int32
14
15 grabberPressure = 0
16
17 def invkin(xyz):
18     """
19     Python implementation of the the inverse kinematics for the crustcrawler
20     Input: xyz position
21     Output: Angels for each joint: q1,q2,q3,q4,q5
22
23     You might adjust parameters (d1,a1,a2,d4).
24     The robot model shown in rviz can be adjusted accordingly by editing
25         au_crustcrawler_ax12.urdf
26     """
27     d1 = 16.8; # cm (height of 2nd joint)
28     a1 = 0.0; # (distance along "y-axis" to 2nd joint)
29     a2 = 17.31; # (distance between 2nd and 3rd joints)
30     d4 = 20; # (distance from 3rd joint to gripper center - all inclusive, ie. also 4th
31             joint)
32
33     x1 = xyz[0]
34     y1 = xyz[1]
35     z1 = xyz[2]
36
37     q1 = math.atan2(y1, x1)
38
39     # Calculate q2 and q3
40     r2 = math.pow((x1 - a1 * math.cos(q1)),2) + math.pow((y1 - a1 * math.sin(q1)),2)
41     s = (z1 - d1)
42     D = (r2 + math.pow(s,2) - math.pow(a2,2) - math.pow(d4,2)) / (2 * a2 * d4)
43
44     q3 = math.atan2(-math.sqrt(1 - math.pow(D,2)), D)
45     q2 = math.atan2(s, math.sqrt(r2)) - math.atan2(d4 * math.sin(q3), a2 + d4 *
46             math.cos(q3)) - (math.pi/2)
47
48     #q4 = 0# not consider rotation so far..
49
50     #print '_____ '
51     #print 'r2 = ' , r2
52     #print 's = ' , s
53     #print 'D = ' , D
54     #print 'q1 = ' , q1, ' : ' , ' q2 = ' , q2, ' : ' , ' q3 = ' , q3
55     #print '_____ '
56
57     return q1,q2,q3
58
59 class RobotExecute:
60
61     N_JOINTS = 5
62
63     #constructur

```

```

62     def __init__(self, server_name, angles, rotate, gripper):
63         self.client = actionlib.SimpleActionClient(server_name,
64                                                     FollowJointTrajectoryAction)
65
66         self.joint_positions = []
67         self.names = ["joint1",
68                      "joint2",
69                      "joint3",
70                      "joint4",
71                      "gripper"]
72         # the list of xyz points we want to plan
73         joint_positions = [
74             [angles[0], angles[1], angles[2], rotate, gripper]
75         ]
76
77         # initial duration
78         dur = rospy.Duration(1)
79
80         # construct a list of joint positions
81         for p in joint_positions:
82             jtp = JointTrajectoryPoint(positions=p, velocities=[0.5]*self.N_JOINTS,
83                                       time_from_start=dur)
84             dur += rospy.Duration(5)
85             self.joint_positions.append(jtp)
86
87         # create joint trajectory
88         self.jt = JointTrajectory(joint_names=self.names, points=self.joint_positions)
89
90         # create goal
91         self.goal = FollowJointTrajectoryGoal(trajectory=self.jt,
92                                               goal_time_tolerance=dur+rospy.Duration(2))
93
94     def send_command(self):
95         self.client.wait_for_server()
96         #print self.goal
97         self.client.send_goal(self.goal)
98
99         self.client.wait_for_result()
100        #print self.client.get_result()
101
102    def RobotDo(angles, rotate, gripper):
103
104        print 'RobotExecute started: '
105        print '_____ '
106
107        node = RobotExecute("/arm_controller/follow_joint_trajectory", angles, rotate,
108                           gripper)
109
110        node.send_command()
111
112    def mirrorCube(xy):
113
114        air1 = 15
115        air2 = 30
116        table = 6
117        grabber_pos = 0
118        not_rotated = 0
119        rotated = 1.5
120        top = invkin([0, 0, 54.1])
121        time1 = 1
122        time2 = 0.5
123
124        #in air
125        RobotDo(invkin([xy[0], xy[1], air2]), not_rotated, grabber_pos)

```

```

124     time.sleep(time1)
125
126     #to table
127     RobotDo(invkin([xy[0], xy[1], table]), not_rotated, grabber_pos)
128     time.sleep(time1)
129
130     #grab brick. While pressure is low enough, increase grabber position until brick is
131     secured.
132     while grabberPressure < 700:
133         print "grabber pressure is: %d" % grabberPressure
134         grabber_pos += 0.3
135         RobotDo(invkin([xy[0], xy[1], table]), not_rotated, grabber_pos)
136
137     #rotate - to air low
138     RobotDo(invkin([xy[0], xy[1], air1]), rotated, grabber_pos)
139     time.sleep(time2)
140
141     #to air
142     RobotDo(invkin([xy[0], xy[1], air2]), rotated, grabber_pos)
143     time.sleep(time1)
144
145     #to mirrored X Y
146     RobotDo(invkin([xy[0], -xy[1], air2]), rotated, grabber_pos)
147     time.sleep(time2)
148
149     # lower
150     RobotDo(invkin([xy[0], -xy[1], air1]), rotated, grabber_pos)
151     time.sleep(time1)
152
153     #table un rotate
154     RobotDo(invkin([xy[0], -xy[1], table]), not_rotated, grabber_pos)
155     time.sleep(time1)
156
157     #release
158     grabber_pos = 0
159     RobotDo(invkin([xy[0], -xy[1], table]), not_rotated, grabber_pos)
160     time.sleep(time1)
161
162     # to air
163     RobotDo(invkin([xy[0], -xy[1], air1]), not_rotated, grabber_pos)
164     time.sleep(time2)
165
166     # higher air
167     RobotDo(invkin([xy[0], -xy[1], air2]), not_rotated, grabber_pos)
168     RobotDo(top, not_rotated, grabber_pos)
169     time.sleep(2)
170
171 def RobotLowFive():
172     x = 28
173     y = 0
174     z = 40
175     not_rotated = 0
176     rotated = 1.5
177     top = invkin([0, 0, 54.1])
178     i = 0
179
180     dur = rospy.Duration(1)
181
182     RobotDo(top, rotated, 0)
183     time.sleep(0.5)
184     RobotDo(top, not_rotated, 0)
185
186     while i < 31:
187         RobotDo(invkin([x, y, z-i]), not_rotated, 0)
188         i += 10

```



```

189
190     RobotDo(top, not_rotated, 0)
191
192     # setup subscriber for pressure sensor
193     def setupGrabberPressureSensor():
194         rospy.Subscriber("grabber_pressure", Int32, handleReadPressure)
195
196     # callback method for pressure sensor
197     def handleReadPressure(val):
198         global grabberPressure
199         grabberPressure = val.data
200
201         #debug

```

A.5 Strain Gauge

```

1  #include <ros.h>
2  #include <std_msgs/Bool.h>
3
4  ros::NodeHandle nh;
5  std_msgs::Bool grabbedSomethingMsg;
6
7  boolean grabbedSomething = false;
8  boolean shouldCheck = true;
9
10 ros::Publisher grabber_strain_gauge_publisher("grabbed_something",
        &grabbedSomethingMsg);
11 int strainGaugePin = A0;
12
13 void handleCheckPressureCb(const std_msgs::Bool& msg) {
14     shouldCheck = msg.data;
15 }
16
17 ros::Subscriber<std_msgs::Bool>
        grabber_strain_gauge_subscriber("grabber_check_pressure", &handleCheckPressureCb);
18
19 void setup()
20 {
21     nh.initNode();
22     nh.advertise(grabber_strain_gauge_publisher);
23
24     //for debug
25     Serial.begin(57600);
26
27
28 }
29
30 void loop()
31 {
32     if(shouldCheck){
33         checkStrain();
34     }
35
36
37     nh.spinOnce();
38
39     delay(10);
40 }
41
42 void checkStrain() {
43
44     //read pressure, publish if high enough. TODO: test values
45     if(analogRead(strainGaugePin) > 400) {
46         publish(true);
47     } else {

```

```
48         publish(false);
49     }
50 }
51
52 void publish(boolean val) {
53     grabbedSomethingMsg.data = val;
54     grabber_strain_gauge_publisher.publish( &grabbedSomethingMsg );
55 }
```

MATLAB Vision B

B.1 VisionSimulation.m

```
1 %% Vision simulation in Matlab
2 image = imread('image3.jpg'); % Read image
3
4 subplot(2,3,1)
5 imshow(image);
6 title('Original image');
7
8 imageCrop = imcrop(image,[300 45 310 300]); % Crop image
9
10 subplot(2,3,4)
11 imshow(imageCrop);
12 title('Cropped image');
13
14 imageBlue = findBlue(imageCrop); % findBlue function created by Color Thresholder App
15 subplot(2,3,2)
16 imshow(imageBlue);
17 title('Blue');
18
19 imageRed = findRed(imageCrop); % findRed function created by Color Thresholder App
20 subplot(2,3,3)
21 imshow(imageRed);
22 title('Red');
23
24 imageGreen = findGreen(imageCrop); % findGreen function created by Color Thresholder App
25 subplot(2,3,5)
26 imshow(imageGreen);
27 title('Green');
28
29 imageYellow = findYellow(imageCrop); % findYellow function created by Color Thresholder
    App
30 subplot(2,3,6)
31 imshow(imageYellow);
32 title('Yellow');
```

B.2 findBlue

```

1 function [BW,maskedRGBImage] = findBlue(RGB)
2 % Auto-generated by colorThresholder app on 05-Oct-2016
3 %-----
4 % Convert RGB image to chosen color space
5 I = rgb2hsv(RGB);
6
7 % Define thresholds for channel 1 based on histogram settings
8 channel1Min = 0.515; %Conversion to openCV range channel1Min*179
9 channel1Max = 0.790; %Conversion to openCV range channel1Max*179
10
11 % Define thresholds for channel 2 based on histogram settings
12 channel2Min = 0.300; %Conversion to openCV range channel2Min*255
13 channel2Max = 1.000; %Conversion to openCV range channel2Max*255
14
15 % Define thresholds for channel 3 based on histogram settings
16 channel3Min = 0.400; %Conversion to openCV range channel3Min*255
17 channel3Max = 1.000; %Conversion to openCV range channel3Min*255
18
19 % Create mask based on chosen histogram thresholds
20 BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
21      (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
22      (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
23
24 % Initialize output masked image based on input image.
25 maskedRGBImage = RGB;
26
27 % Set background pixels where BW is false to zero.
28 maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

```

B.3 findRed

```

1 function [BW,maskedRGBImage] = findRed(RGB)
2 % Auto-generated by colorThresholder app on 05-Oct-2016
3 %-----
4 % Convert RGB image to chosen color space
5 I = rgb2hsv(RGB);
6
7 % Define thresholds for channel 1 based on histogram settings
8 channel1Min = 0.900; %Conversion to openCV range channel1Min*179
9 channel1Max = 0.080; %Conversion to openCV range channel1Max*179
10
11 % Define thresholds for channel 2 based on histogram settings
12 channel2Min = 0.300; %Conversion to openCV range channel2Min*255
13 channel2Max = 1.000; %Conversion to openCV range channel2Max*255
14
15 % Define thresholds for channel 3 based on histogram settings
16 channel3Min = 0.0; %Conversion to openCV range channel3Min*255
17 channel3Max = 1.000; %Conversion to openCV range channel3Min*255
18
19 % Create mask based on chosen histogram thresholds
20 BW = ( (I(:,:,1) >= channel1Min) | (I(:,:,1) <= channel1Max) ) & ...
21      (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
22      (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
23
24 % Initialize output masked image based on input image.
25 maskedRGBImage = RGB;
26
27 % Set background pixels where BW is false to zero.
28 maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

```

B.4 findGreen

```

1 function [BW,maskedRGBImage] = findGreen(RGB)
2 % Auto-generated by colorThresholder app on 05-Oct-2016
3 %-----
4 % Convert RGB image to chosen color space
5 I = rgb2hsv(RGB);
6
7 % Define thresholds for channel 1 based on histogram settings
8 channel1Min = 0.200; %Conversion to openCV range channel1Min*179
9 channel1Max = 0.415; %Conversion to openCV range channel1Max*179
10
11 % Define thresholds for channel 2 based on histogram settings
12 channel2Min = 0.300; %Conversion to openCV range channel2Min*255
13 channel2Max = 1.000; %Conversion to openCV range channel2Max*255
14
15 % Define thresholds for channel 3 based on histogram settings
16 channel3Min = 0.000; %Conversion to openCV range channel3Min*255
17 channel3Max = 1.000; %Conversion to openCV range channel3Min*255
18
19 % Create mask based on chosen histogram thresholds
20 BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
21      (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
22      (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
23
24 % Initialize output masked image based on input image.
25 maskedRGBImage = RGB;
26
27 % Set background pixels where BW is false to zero.
28 maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

```

B.5 findYellow

```

1 function [BW,maskedRGBImage] = findYellow(RGB)
2 % Auto-generated by colorThresholder app on 05-Oct-2016
3 %-----
4 % Convert RGB image to chosen color space
5 I = rgb2hsv(RGB);
6
7 % Define thresholds for channel 1 based on histogram settings
8 channel1Min = 0.115; %Conversion to openCV range channel1Min*179
9 channel1Max = 0.210; %Conversion to openCV range channel1Max*179
10
11 % Define thresholds for channel 2 based on histogram settings
12 channel2Min = 0.300; %Conversion to openCV range channel2Min*255
13 channel2Max = 1.000; %Conversion to openCV range channel2Max*255
14
15 % Define thresholds for channel 3 based on histogram settings
16 channel3Min = 0.403; %Conversion to openCV range channel3Min*255
17 channel3Max = 1.000; %Conversion to openCV range channel3Min*255
18
19 % Create mask based on chosen histogram thresholds
20 BW = (I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
21      (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
22      (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
23
24 % Initialize output masked image based on input image.
25 maskedRGBImage = RGB;
26
27 % Set background pixels where BW is false to zero.
28 maskedRGBImage(repmat(~BW,[1 1 3])) = 0;

```

Vision principper C

C.1 Masker

C.2 Morphologi

C.2.1 Erode

C.2.2 Dialate

Koordinatsystem konvertering

(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)