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

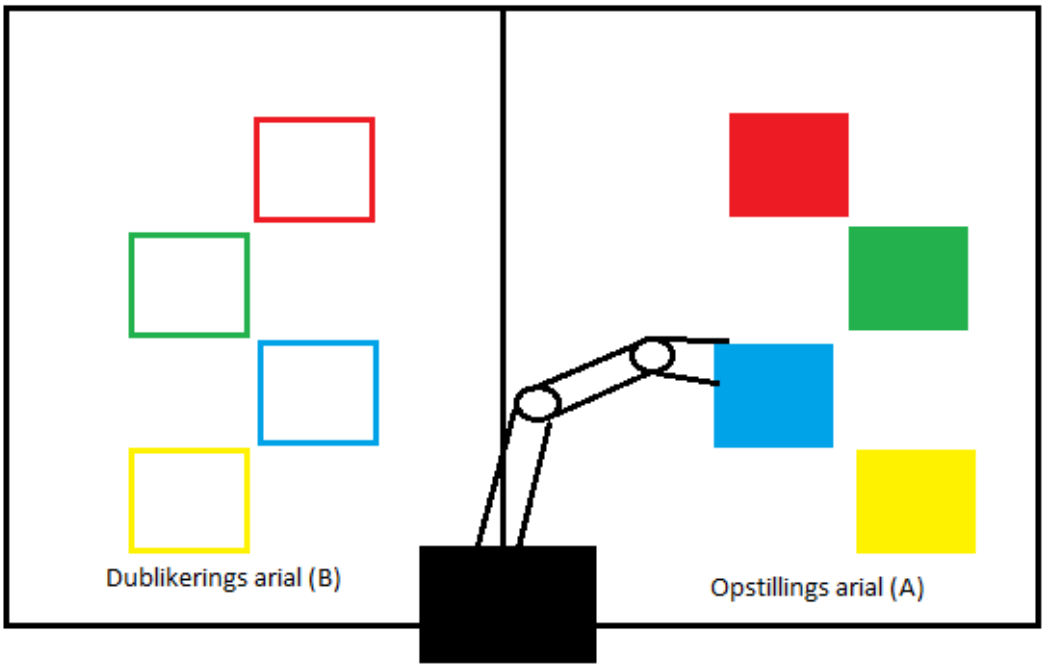
I faget ITROB1 er der blevet stillet en opgave om, at skrive et program, som skal kunne styre den mekaniske robotarm også kaldet CrustCrawler. Selve opgaven blev stillet meget fri, og derfor var det op til gruppen, at bestemme hvordan denne skulle løses. De eneste krav til opgaven var at der skulle oprettes 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 hardcodede 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.

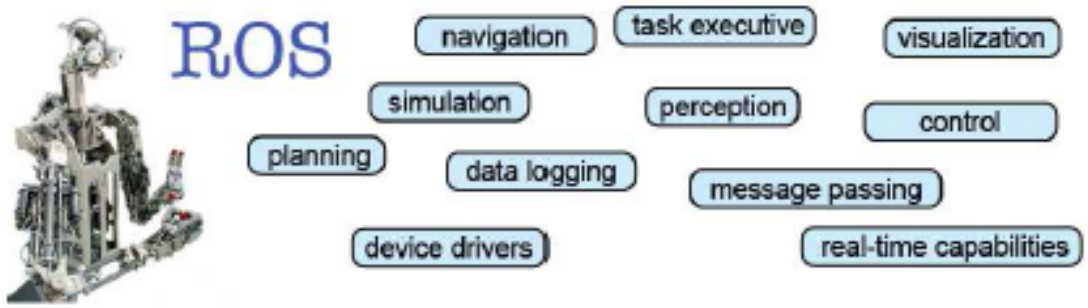
Robot Operating System (ROS) 4

4.1 ROS 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.

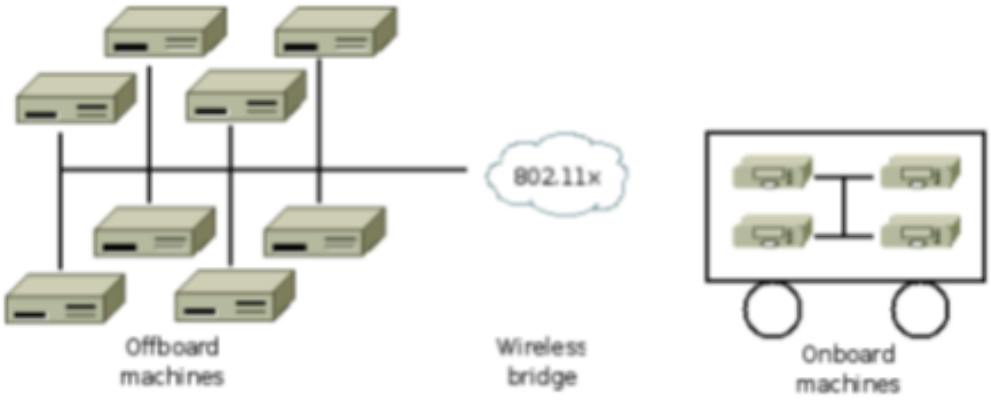


Figur 4.1. ROS design kriterier

Frameworket er designet efter nedenstående kriterier.

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

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 a noder, hvor det kan opdelse således at én node udføre én handling. På store robotter, som bruges i indrustien, er der typsik 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 vission 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 pushed og lageret i subnettet. På nedenstående figur ses hvordan P2P visuelt fungere.



Figur 4.2. ROS netværks konfiguration

Tool-based: For at kunne håndtere komplesiteten 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 supporter disse forskellige programmeringssprog, gør at ROS i sig selv er let anvendeligt for de fleste programmør, 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 alemen bruger, og kræver en linces for at får adgang til.

4.3 ROS virkemåde

De grundlæggende begreber for ROS implementationen er:

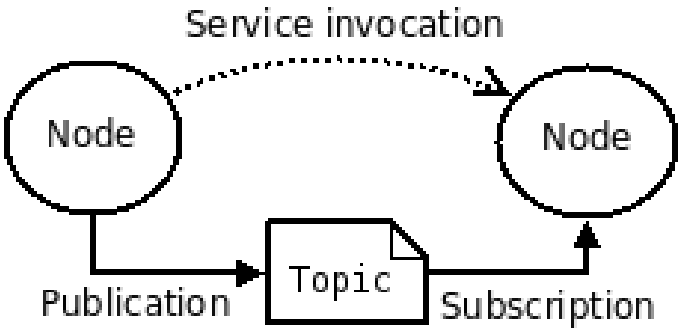
- Nodes
- Messages
- Topics
- Services

Nodes: Noder er de processer, som udfører en databehandling. For at kunne kommunikerer med mange noder, er det praktisk at bruge P2P kommunikationsmetoden. På den måde kommunikerer noder gennem beskeder, og kan køre asynkron fra hinanden, og derved opdeldes arbejdsbyrden fra den enekelt tråd.

Messages: Messages er en strengt type datastruktur, hvor primative standarts typer som integer, floation point, booelan osv. er supportet. Disse beskeder bliver benyttet af noderne til at kommunikerer med hinanden. Beskederne kan sammensættes af flere forskellige beskeder, og derved oprette en message queue.

Topic: Når en node udfører en kommando, sender den en besked afsted. Denne besked bliver så published på et givent topic, hvor beskeden kan indeholde den ønskede type, som er supportet. Når noden har published en besked, lkan en anden node i systemet så subcribe på lige præcis den besked, som den første node sendte afsted. Derved kan den sidste nævnte node modtage de data, som den første node sender afsted med beskeden.

Dette er et public subscribe forhold. Grunden til dette forhold mellem noder er, at den node som publicher og den node som subscriber er ikke afhængige af hinanden, eller de er ikke opmærksomme på deres eksistens, og derved opstår der ingen komplikationer mellem disse noder.
Nedenfor ses en figur over hvordan kommunikationen mellem de to noder fungerer.



Figur 4.4. ROS Topic

Service: Ved publish og subcribe forholdet er dette en meget fleksibel form for kommunikation også kaldet broadcasting. Denne broadcasting er asynkront, og køre derfor uafhængigt af "main" tråden. En service er defineret af et string navn og et par af strengte typer beskeder. Den ene bruges som en request og den anden bruges som en response. Services anvender srv filer, som er kompileret i source koden ved et ROS-client bibliotek.

Metode 5

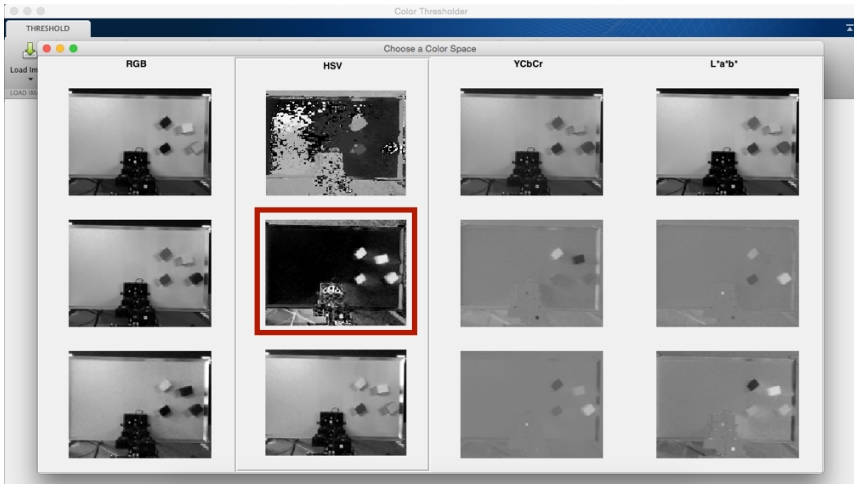
5.1 Vision system

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

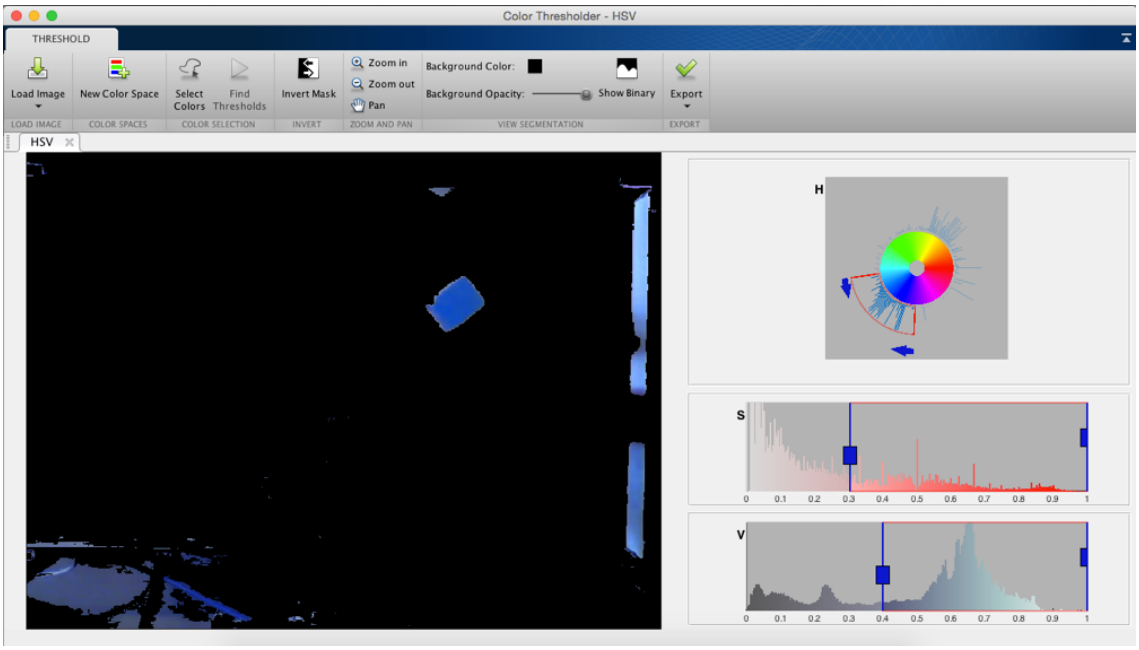
MATLAB's applikation Color Thresholder er blevet brugt til at bestemme, hvilket color space og hvilke grænseværdier, der er mest optimale til at identificere de farvede klodser med webcamet.

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.

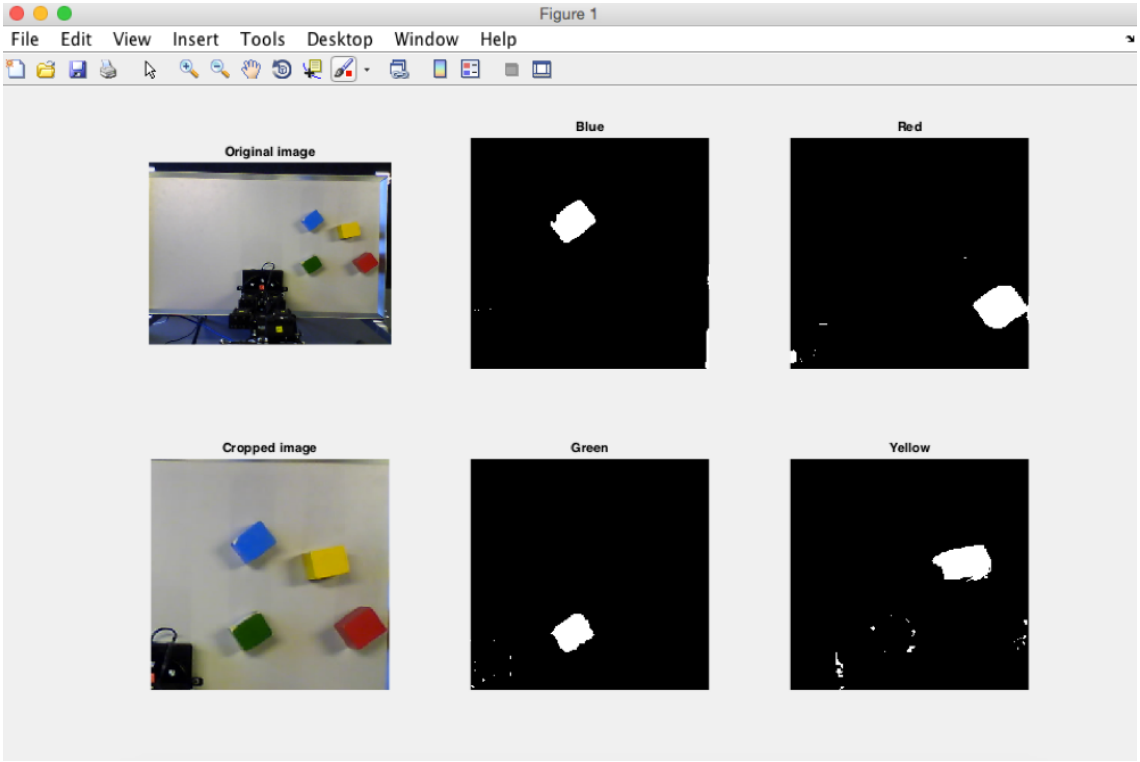
For hver farve blev følgende grænseværdier identificeret (se bilag B på side 28 for funktioner til identifikation af hver farve):

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 B på side 28 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 webcamet og beskærer det så det kun er det relevante område der vises på billedet. Funktionen returnerer det beskårede billede.

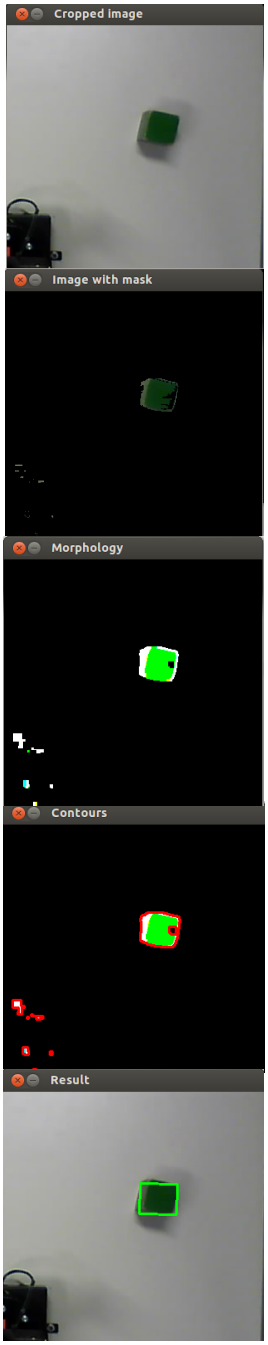
Billedet fra `get_from_webcam` konverteres fra color spacet BGR til HSV ved brug af OpenCV-Python funktionen `cv2.cvtCOLOR`. Efter konverteringen kaldes funktionerne `extract_single_color_range`, `threshold_image`, `contours` og `get_centers` på billedet for hver farve.

`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 31 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 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.

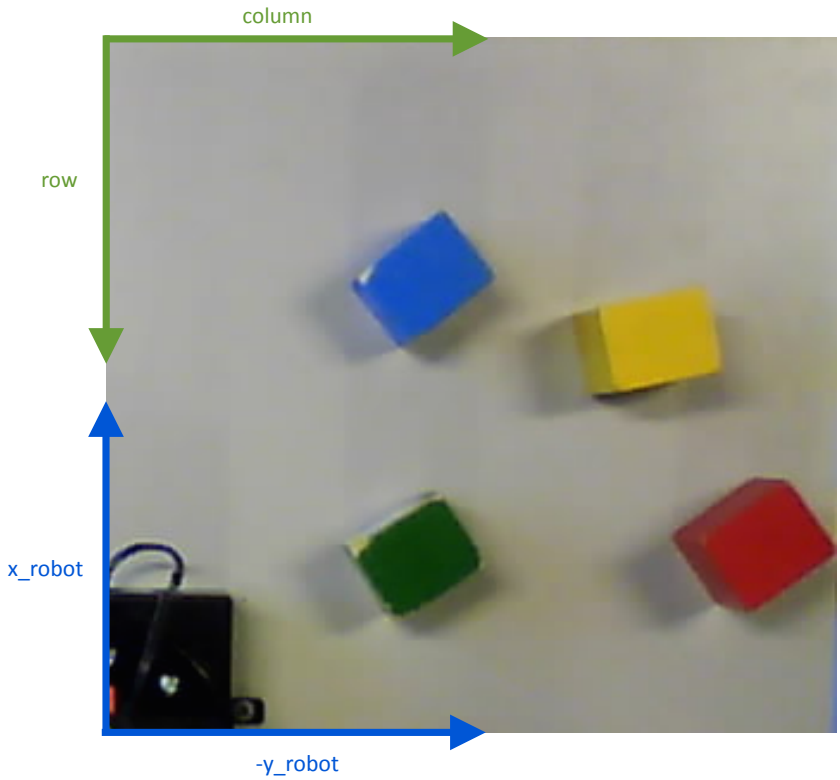
`get_centers` finder for hver kontur en firkantet repræsentation af konturen ved brug af OpenCV-Python funktionerne `cv2.arcLength` og `cv2.approxPolyDP` og finder derefter arealet af disse ved brug af OpenCV-Python funktionen `cv2.contourArea`. Hvis dette areal er større end 500 pixels



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 32 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.

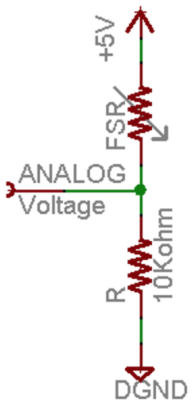
5.2 Pressure sensor system

For at kunne identificere, hvorvidt robotten har grebet fat om en klods, er en tryksensor blevet anvendt. Denne sidder yderst på robottens griber. Til at tolke målingerne fra

sensoren er en Arduino microcontroller blevet anvendt. Til kommunikation mellem Arduinoen og resten af systemet, er Rosserial bibliotiket blevet anvend.

5.2.1 FSR 402 - Tryksensor

Til at måle trykket, er en Interlink FSR 402 Short anvendt. Denne fungerer ved at ændre modstanden over sensoren i forhold til det tryk, der lægges på den. Dette er en meget simpel sensor at anvende, da der udover sensoren blot skal anvendes en enkelt pull-down modstand.



Figur 5.6. Diagram over tilkobling af tryksensor til microcontroller

Som det ses på figur 5.6, er sensoren på det ene ben koblet til microcontrollerens 5v forsyning, og på det andet ben koblet til en 10 KOhm modstand, samt den pin, der rent faktisk måles på. Dette fungerer ved, at efterhånden som modstanden over FSR Sensoren falder, så mindskes den samlede modstand over FSR Sensoren og pull down modstanden. Dette medfører, at strømstyrken over begge modstande øges, og spændingen over den faste modstand stiger. Dermed øges spændingen også til den pin hvorpå målingen foretages.

5.2.2 Arduino - Microcontroller

Som microcontroller er en Arduino UNO blevet anvendt. Valget faldt på denne, da den er let at anvende sammen med analoge sensorer, samt let at integrere med ROS, da der findes et officielt bibliotek til kommunikation mellem Arduino og ROS. Rosserial biblioteket anvendes til at oprette en node og publisere målinger. Til dette anvendes en nodehandler og en publisher.

```
1
2  //variables
3  ros::NodeHandle nh;
4  std_msgs::Int32 pressureMsg
5  ros::Publisher publisher("grabber_pressure", &pressureMsg);
6
7  void setup()
8  {
9    // init node
10   nh.initNode();
11
12   // setup publisher
```

```
13   nh.advertise(grabber_strain_gauge_publisher);
14
15   //init serial comm
16   Serial.begin(57600);
17 }
```

Læsning af sensoren og publisering af den data er meget simpel. Der foretages en læsning med `analogread()` metoden, der er en standard Arduino metode. Parametret der gives med er blot den pin, der foretages en læsning på. Denne læsning sendes med som data attributten på den message, der publiseres.

```
1
2  int checkStrain()
3  {
4   return analogRead(strainGaugePin);
5  }
6
7  void publish(int val) {
8        pressureMsg.data = val;
9        publisher.publish(&pressureMsg);
10 }
```

For at få et kompromis mellem kontinuerlige data og spamming af beskeder, er der sat 10 millisekunders delay mellem hver læsning og publisering af data.

5.2.3 Kommunikation med ROS

Rosserial er som nævnt anvendt til kommunikation mellem microcontrolleren og resten af ROS systemet. Microcontrolleren skal dog stadig være sat til en pc, hvorpå en fuld ROS installation kører. Dette er nødvendigt, da Rosserial ikke i sig selv er en featurekomplet ROS installation, og derfor har brug for at der fra pc siden initialeres en forbindelse til den. Dette gøres vha. følgende kommando.

```
rosrun rosserial_python serial_node.py /dev/AMC0 _baud:=57600
```

De sidste to argumenter specificerer henholdsvis den port, som microcontrolleren befinder sig på, og den baud rate, som forbindelsen skal oprettes med.

Resultater 6

7 Diskussion

Konklusion 8

Python Kode A

A.1 Main

```
#!/usr/bin/env python

import rospy
import actionlib
from control_msgs.msg import FollowJointTrajectoryAction
from control_msgs.msg import FollowJointTrajectoryFeedback
from control_msgs.msg import FollowJointTrajectoryResult
from control_msgs.msg import FollowJointTrajectoryGoal
from trajectory_msgs.msg import JointTrajectoryPoint
from trajectory_msgs.msg import JointTrajectory
import math
import time
from std_msgs.msg import String

from InvRobot import *
from findBricks import *

visionCoor = []

def getCoordinates(coord):
   global visionCoor

   coord_str = str(coord)
   coor = coord_str.replace("data: ","")
   visionCoor = [float(s) for s in coor.split(',')]

   if visionCoor[0] == 0 and visionCoor[1] == 0:
      print "Nu skal programmet stoppe"
      visionCoor = []
      print "VisionCoor"
      print visionCoor

def main():
   Jobactive = True
   Job = False

   while Jobactive == True:
      print "V3"
      print visionCoor
      xy = visionCoor
      if len(xy) > 0:
         mirrorCube(xy)
         Job = True
      else:
         Jobactive = False
         if Job == True:
            RobotLowFive()
            print "Jobs done"
         else:
            print "There wasnt any job "

```

```
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          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
```

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
           au_crustcrawler_ax12.urdf
25     """
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
           joint)
31
32     x1 = xyz[0]
33     y1 = xyz[1]
34     z1 = xyz[2]
35
36     q1 = math.atan2(y1, x1)
37
38     # Calculate q2 and q3
39     r2 = math.pow((x1 - a1 * math.cos(q1)),2) + math.pow((y1 - a1 * math.sin(q1)),2)
40     s = (z1 - d1)
41     D = (r2 + math.pow(s,2) - math.pow(a2,2) - math.pow(d4,2)) / (2 * a2 * d4)
42
43     q3 = math.atan2(-math.sqrt(1 - math.pow(D,2)), D)
44     q2 = math.atan2(s, math.sqrt(r2)) - math.atan2(d4 * math.sin(q3), a2 + d4 *
           math.cos(q3))-(math.pi/2)
45
46     #q4 = 0# not consider rotation so far..
47
48     #print '_____________'
49     #print 'r2 = ' , r2
50     #print 's = ' , s
51     #print 'D = ' , D
52     #print 'q1 = ', q1, '::' , ' q2 = ' , q2, '::' , ' q3 = ' , q3
53     #print '_____________'
54
55     return q1,q2,q3
56
57 class RobotExecute:
58
59     N_JOINTS = 5
60
61     #constructur
```

```
62      def __init__(self,server_name, angles, rotate, gripper):
63          self.client = actionlib.SimpleActionClient(server_name,
                FollowJointTrajectoryAction)
64
65          self.joint_positions = []
66          self.names =["joint1",
67                       "joint2",
68                       "joint3",
69                       "joint4",
70                       "gripper"
71                       ]
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,
                    time_from_start=dur)
83              dur += rospy.Duration(5)
84              self.joint_positions.append(jtp)
85
86          # create joint trajectory
87          self.jt = JointTrajectory(joint_names=self.names, points=self.joint_positions)
88
89          # create goal
90          self.goal = FollowJointTrajectoryGoal( trajectory=self.jt,
                goal_time_tolerance=dur+rospy.Duration(2) )
91
92      def send_command(self):
93          self.client.wait_for_server()
94          #print self.goal
95          self.client.send_goal(self.goal)
96
97          self.client.wait_for_result()
98          #print self.client.get_result()
99
100
101 def RobotDo(angles, rotate, gripper):
102
103     print 'RobotExecute started: '
104     print '_____________________'
105
106     node = RobotExecute("/arm_controller/follow_joint_trajectory", angles, rotate,
            gripper)
107
108     node.send_command()
109
110 def mirrorCube(xy):
111
112     air1 = 15
113     air2 = 30
114     table = 6
115     grabber_pos = 0
116     not_rotated = 0
117     rotated = 1.5
118     top = invkin([0, 0, 54.1])
119     time1 = 1
120     time2 = 0.5
121
122     #in air
123     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
            secured.
131     while grabberPressure < 700:
132        print "grabber pressure is: %d" % grabberPressure
133        grabber_pos += 0.3
134        RobotDo(invkin([xy[0], xy[1], table]), not_rotated, grabber_pos)
135 
136     #rotate - to air low
137     RobotDo(invkin([xy[0], xy[1], air1]), rotated, grabber_pos)
138     time.sleep(time2)
139 
140     #to air
141     RobotDo(invkin([xy[0], xy[1], air2]), rotated, grabber_pos)
142     time.sleep(time1)
143 
144     #to mirrored X Y
145     RobotDo(invkin([xy[0], -xy[1], air2]), rotated, grabber_pos)
146     time.sleep(time2)
147 
148     # lower
149     RobotDo(invkin([xy[0], -xy[1], air1]), rotated, grabber_pos)
150     time.sleep(time1)
151 
152     #table un rotate
153     RobotDo(invkin([xy[0], -xy[1], table]), not_rotated, grabber_pos)
154     time.sleep(time1)
155 
156     #release
157     grabber_pos = 0
158     RobotDo(invkin([xy[0], -xy[1], table]), not_rotated, grabber_pos)
159     time.sleep(time1)
160 
161     # to air
162     RobotDo(invkin([xy[0], -xy[1], air1]), not_rotated, grabber_pos)
163     time.sleep(time2)
164 
165     # higher air
166     RobotDo(invkin([xy[0], -xy[1], air2]), not_rotated, grabber_pos)
167     RobotDo(top, not_rotated, grabber_pos)
168     time.sleep(2)
169 
170 def RobotLowFive():
171 
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

```
%% 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');
```

B.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;
```

B.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;
```

B.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;
```

B.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 principper C

C.1 Masker

C.2 Morphologi

C.2.1 Erode

C.2.2 Dialate

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)