1 Orbbec Astra

Kamerasysteme zur Ermittlung von Entfernungen können aus mehreren Kameras (Stereo Bilderkennung), oder aus einer Kombination aus einer RGB Kamera, einem Infrarot Sender und Empfänger (TOF) bestehen. Die hier verwendete Kamera wurde aufgrund der Verbindungsmöglichkeit mit ROS, der Kompaktheit und der Preislage gewählt. Zusätzlich wurde diese Version der Kamera bereits in anderen Projekten verwendet und soll weiterhin praktisch genutzt werden können.

Die Orbbec Astra ist eine 3D die auf dem TOF-Prinzip (time of flight) beruht. Der Projektor sendet im Infrarotbereich Lichtpunkte aus, diese Punkte werden von Objekten im Sichtfeld reflektiert und von dem IR Sensor empfangen. Nun wird mittels Winkel und Abstand des Projektors zum Sensor und der Zeit zwischen Senden und Empfangen die resultierende Entfernung errechnet.

Abb. 1: Orbbec Astra

Da der RGB und IR Sensor einen Abstand zueinander haben, sind die resultierenden Bilder nicht direkt über lagerbar, es müssen Anpassungswerte berechnet werden.

2 Treiber Kamera

Für die Treiber der Kamera werden von Orbbec zwei Möglichkeiten genannt. Als erstes die OpenNI2 Umgebung, in der Programme in C++ geschrieben werden können. Die erforderlichen Daten sind auf https://orbbec3d.com/develop/ zum Download verfügbar. Informationen zum Installieren befinden sich in der enthaltenden Anleitung.
Die zweite Möglichkeit ist der in dieser Arbeit verwendete Treiber, welcher eine direkte Verknüpfung mit ROS darstellt. Durch die folgenden Befehle wird die Software heruntergeladen, installiert beziehungsweise kompiliert und ausgeführt.

- Installation
  - 'cd catkin_ws/src'
  - 'git clone https://github.com/orbbec/ros_astra_camera'
  - 'git clone https://github.com/orbbec/ros_astra_launch'
  - 'git clone https://github.com/ros-drivers/rgbd_launch'
  - 'catkin_make'
- Ausführen
  - 'roscore'
  - 'roslaunch astra_launch astra.launch'
  - 'rqt_image_view (Auswahl: 'rgb_raw' oder 'depth_image')'

3 Daten Strukturen der Kamera

Die Kameradaten welche verwendet werden, sind das rohe Farbbild 'rgb/Image_raw' und das Bild welches die Tiefen Informationen enthält 'depth/Image'.

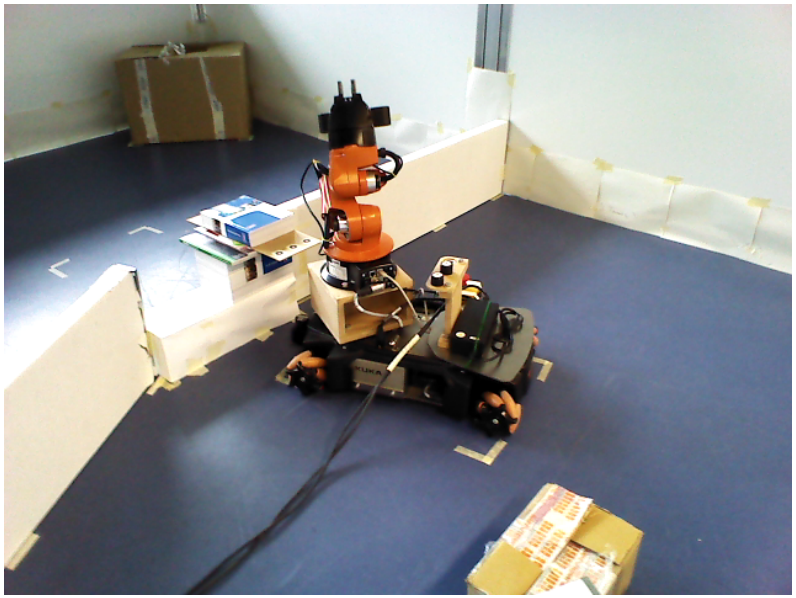


Abb. 2: RGB Bild

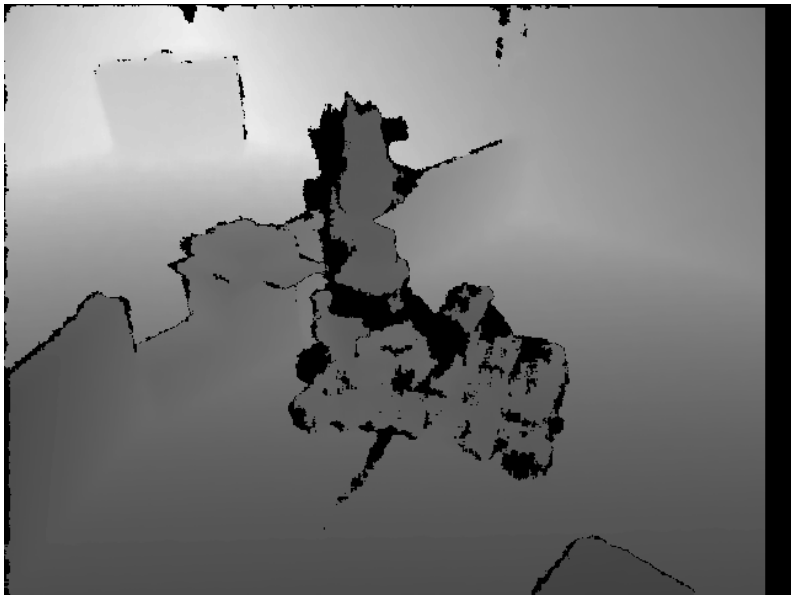
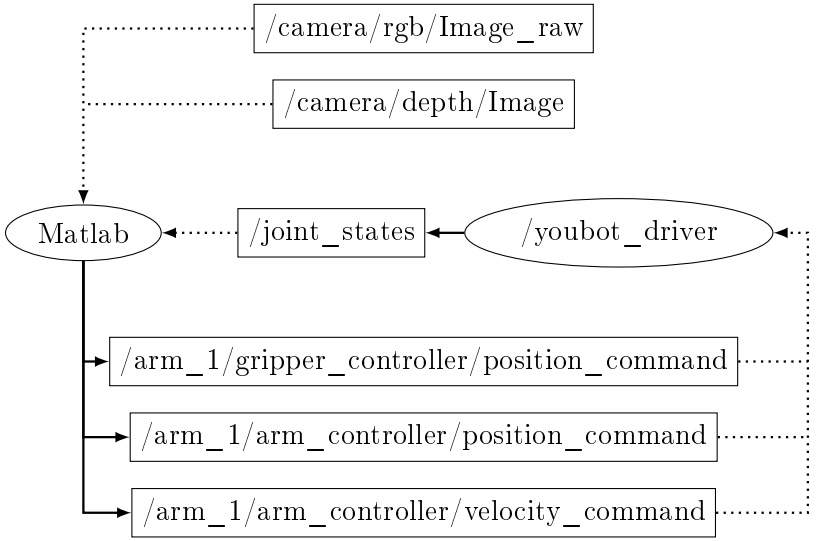


Abb. 3: Tiefen Bild

In dem Tiefenbild wird die Tiefe mittels Graustufen dargestellt, je näher ein Messpunkt der Kamera ist desto dunkler wird dieser angezeigt. An Stellen wo das Infrarotlicht auf reflektierende Oberflächen trifft oder Schatten wirft, werden schwarze Bereiche angezeigt. Beide Bilder haben eine Auflösung von 640px*480px. Es wurde nicht die höhere Auflösung für das Farbbild gewählt, somit besteht eine bessere Berechnungsfähigkeit der Bilder.



Das oben zu sehende Diagramm zeigt eine vereinfachte Darstellung einer ROS Umgebung. Die Treiber der Kamera und des youBots sind aktiv. In der ROS-Node mit dem Namen MATLAB läuft das Hauptprogramm, welches alle relevanten Daten benötigt. Die Daten der Kamera und der Gelenke werden von der MATLAB-Node ausgelesen. Zusätzlich werden Daten zur Steuerung des Arms und Greifers von dieser Node in den zugehörigen Controller geschrieben. Die Verfahrbefehle werden von dem youBot Treiber zu Bewegungen verarbeitet, zusätzlich aktualisiert dieser die Daten zu den Gelenken in 'joint_states'.

4 MATLAB Umgebung

- ROS
  - Kamera
    - RGB Sub.
    - Depth Sub.
  - Arm
    - JointStates Sub.
    - Velocity Pub.
    - Position Pub.
    - Nachricht
    - Info
      - JointMin
      - JointMax
      - JointRes
      - JointUp
      - JointValueMin
      - JointValueMax
      - JointValueRes
      - JointValueUp
  - Greifer
    - Position Pub.
    - Nachricht

Beim Start des Programms wird eine Struktur angelegt die alle wichtigen Daten bezüglich der ROS Verbindungen enthält sowie weitere Informationen zum Arm und zur Ladefläche. Diese Struktur wird beim Aufrufen der Unterprogramme übergeben, damit diese zum Beispiel Daten senden beziehungsweise empfangen können. Die nebenstehende Abbildung gibt einen Überblick der Daten welche sich in dieser Struktur befinden. Die jeweiligen Verbindungsinformationen der Kamera, des Arms und des Greifers sind in drei Teile gegliedert:

1. Sub.(subscriber): enthält die Daten der Verbindung zum Auslesen von zum Beispiel: Bildern.
2. Pub.(publisher): enthält die Daten der Verbindung zum Steuern des Arms beziehungsweise des Greifers.
3. Nachricht: beschreibt die Form des Befehls zum Senden über die Publisher Verbindung.

Zusätzlich sind in Info weitere Daten hinterlegt, wie zum Beispiel die Position der einzelnen Ladestellen oder bestimmte Positionen des Arms.

Als Beispiel einer Unterfunktion, die Aufgerufen wird und ROS benötigt, ist hier die Funktion der Kreiserkennung aufgeführt.

'KreisErkennung(ROS,'w',20);'

Diese benötigt die ROS Informationen 'Kamera.RGBSub' und 'Kamera.DepthSub' zum Auslesen der aktuellen Bilder.

5 Programm Kreiserkennung

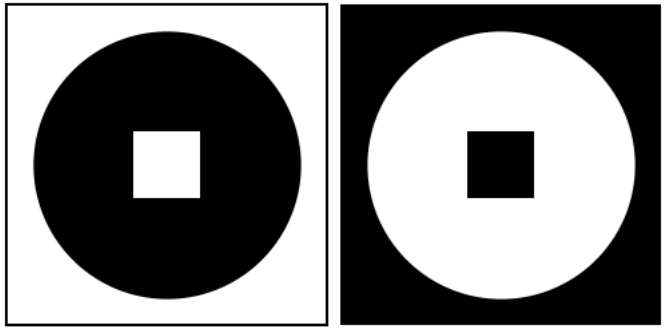


Abb. 4: Symbole der Kreiserkennung

Die Anforderungen die ein Symbol zum Erkennen erfüllen muss:

- hoher Kontrast zur Umgebung
- so groß, dass bei der weitesten Entfernung eine Erkennung noch möglich ist
- so klein, dass es auf den aufzuhebenden Objekten angebracht werden kann
- eine Markierung besitzt, mittels der dieses Symbol von anderen leicht zu unterscheiden ist

Die MATLAB Funktion, die das Erkennen von Kreisen übernimmt, nennt sich
'imfindcircles'. Diese benötigt bestimmte Einstellungen um, für dieses Szenario, das beste Ergebnis zu liefern.

'[centers, radii] = imfindcircles(
imgGray,[Rmin Rmax],
'ObjectPolarity',ColorCode,
'Sensitivity',Sensitivity,
'Method','twostage');'

Die Eingabewerte 'Rmin' und 'Rmax' geben den Bereich an, in dem sich die zu erkennenden Radien befinden müssen. Die Einstellung was erkannt werden soll, Schwarze oder Weiße Kreise, wird mittels der 'ObjektPolarity' eingestellt, hier 'ColorCode'. Dies wird durch die Eingabe entweder auf 'dark' oder 'bright' eingestellt. 'Sensitivity' gibt die Sensibilität an, ab wann eine runde Form als Kreis erkannt wird. Die Methode 'twostage' wurde gewählt, da diese bei schwierigeren Hintergründen ein besseres Ergebnis erzielt als die Standardeinstellung.

Die Funktion 'KreisErkennung' benötigt mindestens vier Eingaben und gibt eine Struktur zurück, welche alle Daten zu den erkannten Kreisen enthält. Diese Funktion kann mit vier bis zwölf Eingaben ausgeführt werden. Die unterschiedliche Anzahl der Eingaben wird mit 'varargin' verwirklicht. Dies bedeutet 'Variable-length input argument list'. In dieser werden alle weiteren Eingaben gespeichert und nacheinander bearbeitet.
Pflicht Eingaben sind:

- ROS-Informationen
- 'w' oder 's' = weiß oder schwarz
- '1' oder '2' = Kreise oder Abstand zwischen Kreisen
- Durchmesser
- (falls '2') Abstand

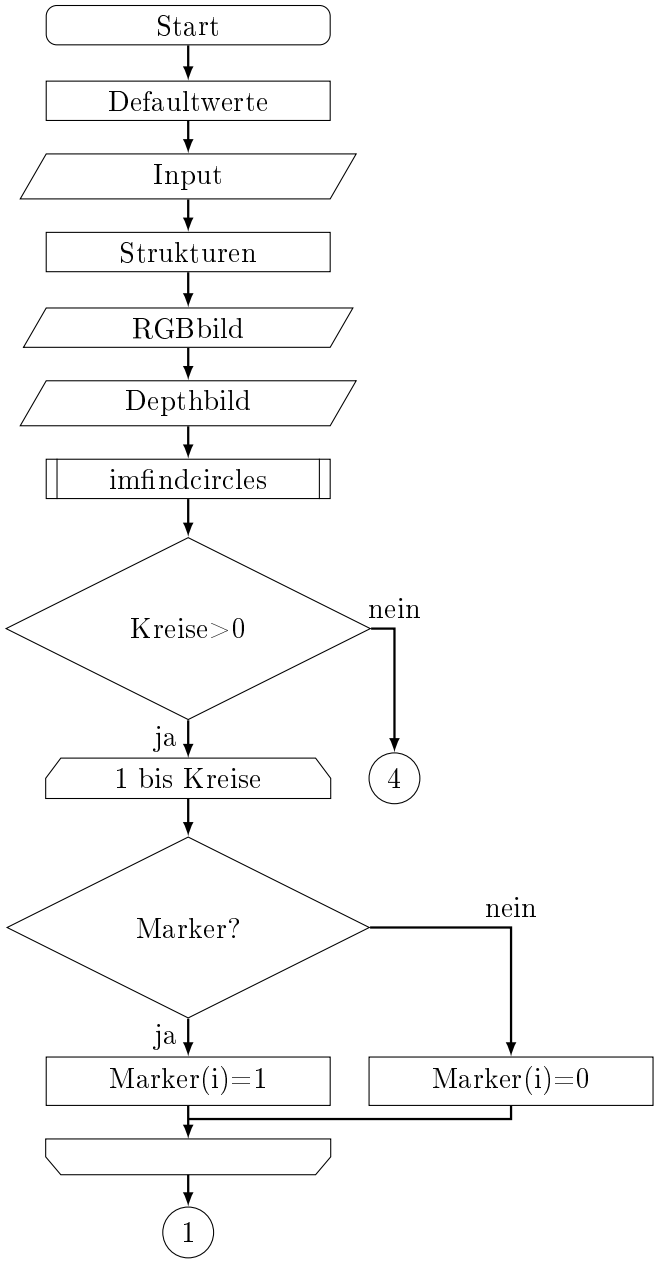
Weitere Eingaben können Standardwerte anpassen oder Ausgaben erzeugen, diese sind:

- 'Dtol',Wert = Durchmesser Toleranz
- 'Atol',Wert = Abstandstoleranz
- 'Sens',Wert = Sensibilität
- 'Bild' = Ausgabe

Ein vollständiger Aufruf dieser Funktion könnte wie folgt aussehen:

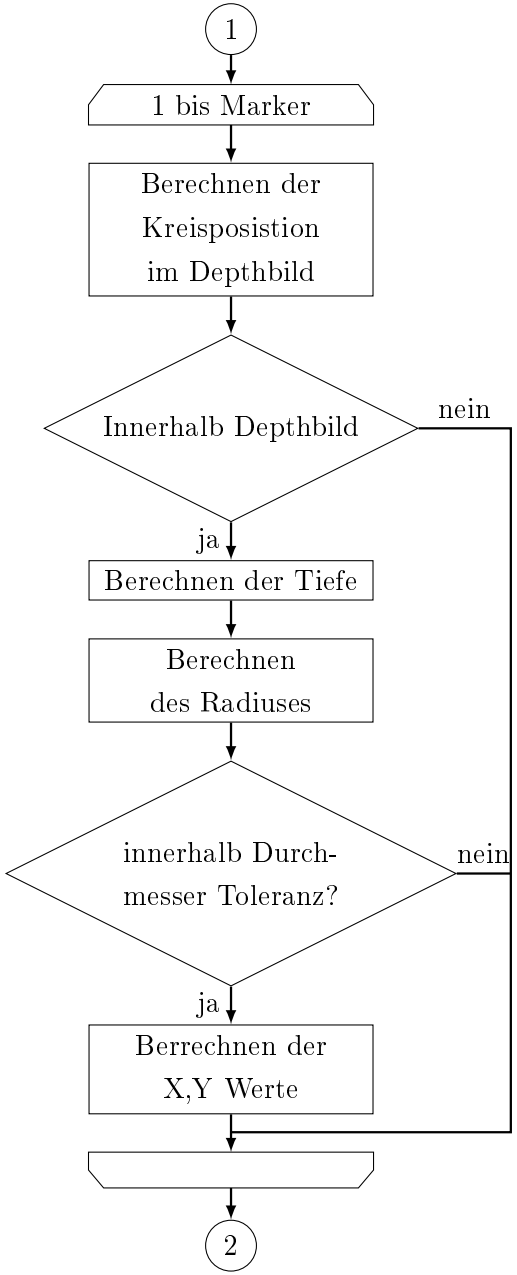
'Ablagestellen=KreisErkennung(ROS,'s','2',20,52,'Dtol',5,'Atol',10,'Sens',0.7,'Bild');'

Das Beispiel bedeutet das schwarze Kreise mit einer Sensibilität von 0.7 erkannt werden. Die Mittelpunkte zwischen diesen Kreisen werden berechnet, falls diese einen Durchmesser von 20±5mm und einen Abstand von 52±10mm zueinander haben. Zum Schluss werden die Mittelpunkte in einem Bild dargestellt und die berechneten Werte ausgegeben.



'Ausgabe=KreisErkennung(ROS,varargin)'

Der Ablauf der Funktion beginnt mit dem Setzten der Standardwerte, gefolgt von der Verarbeitung der Eingabe, die gegebenenfalls die Standardwerte verändert.
Danach werden Strukturen angelegt in denen die Daten gespeichert, verarbeitet und ausgegeben werden.
Mit den ROS-Umgebungsdaten werden das Farbbild und das Tiefenbild ausgelesen. Das Farbbild wird in ein Graustufenbild umgewandelt und der Funktion 'imfindcircles', mit den eingestellten Parametern, übergeben. Falls die Funktion keine Kreise, welche den Parametern entsprechen, gefunden hat wird das Programm beendet.
Falls Kreise gefunden wurden, wird bei jedem Kreis überprüft ob dieser eine Markierung besitzt und dementsprechend abgespeichert.



Für jeden erkannten Kreis, der auch eine Markierung besitzt, wird die Position berechnet die der Kreis im Tiefenbild hat, siehe Formeln 1 und 2. Die Verschiebung kommt zustande, da die RGB Kamera zum IR Sensor versetzt ist, siehe Abbildung 5.

$$DX = KX + (KX - 130) * 0.1 \tag{1}$$

$$DY = KY + (KY - 60) * 0.1 \tag{2}$$

Falls die berechnete Position sich innerhalb des Tiefenbilds befindet, wird erst die Tiefe mittels des Tiefenbilds berechnet und in Millimeter umgewandelt, sowie die Differenz zu der Nullpunktverschiebung gebildet, siehe Formeln 3 bis 5.

$$KZ = imgDepth(DY, DX) \tag{3}$$

$$Z0 = 1000 * KZ \tag{4}$$

$$RZ = 564 - Z0 \tag{5}$$

Anschließend wird mit der Tiefe und dem von 'imfindcircles' ermittelten Radius, der echte Radius berechnet. Die Entfernung zur Kamera wird mittels eines Faktors berücksichtigt. Dieser errechnet sich aus den Punkt an dem der erkannte Radius dem echten Radius entspricht. Zusätzlich wird der echte Durchmesser errechnet, siehe Formeln 6 bis 9.

$$FZ = \frac{Z0}{530} \tag{6}$$

$$R0 = KR * FZ \tag{7}$$

$$RR = R0 \tag{8}$$

$$RD = R0 * 2 \tag{9}$$

Falls der echte Durchmesser sich innerhalb der Toleranz zu dem gesuchten Durchmesser befindet, werden aus den XY-Werten des RGB-Bilds die echten Werte berechnet. Dazu werden die Kamerawerte auf den Mittelpunkt bezogen, der Tiefen Faktor berücksichtigt und die Nullpunktverschiebung der Kamera hinzugefügt, siehe Formeln 10 bis 13.

$$Y0 = (240 - KY) * FZ \tag{10}$$

$$X0 = (320 - KX) * FZ \tag{11}$$

$$RY = 95 + Y0 \tag{12}$$

$$RX = -12 - X0 \tag{13}$$

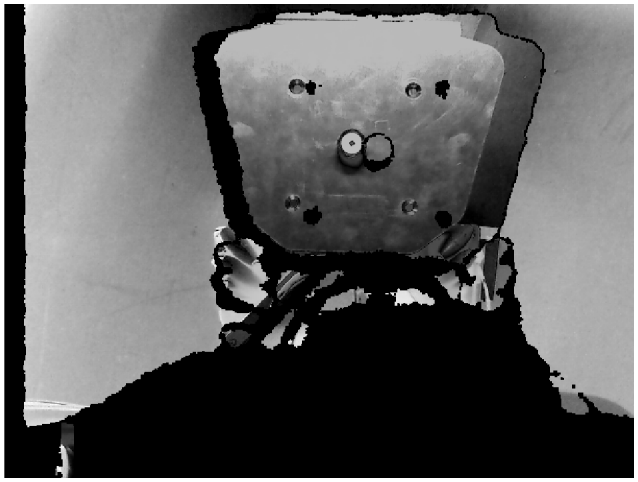
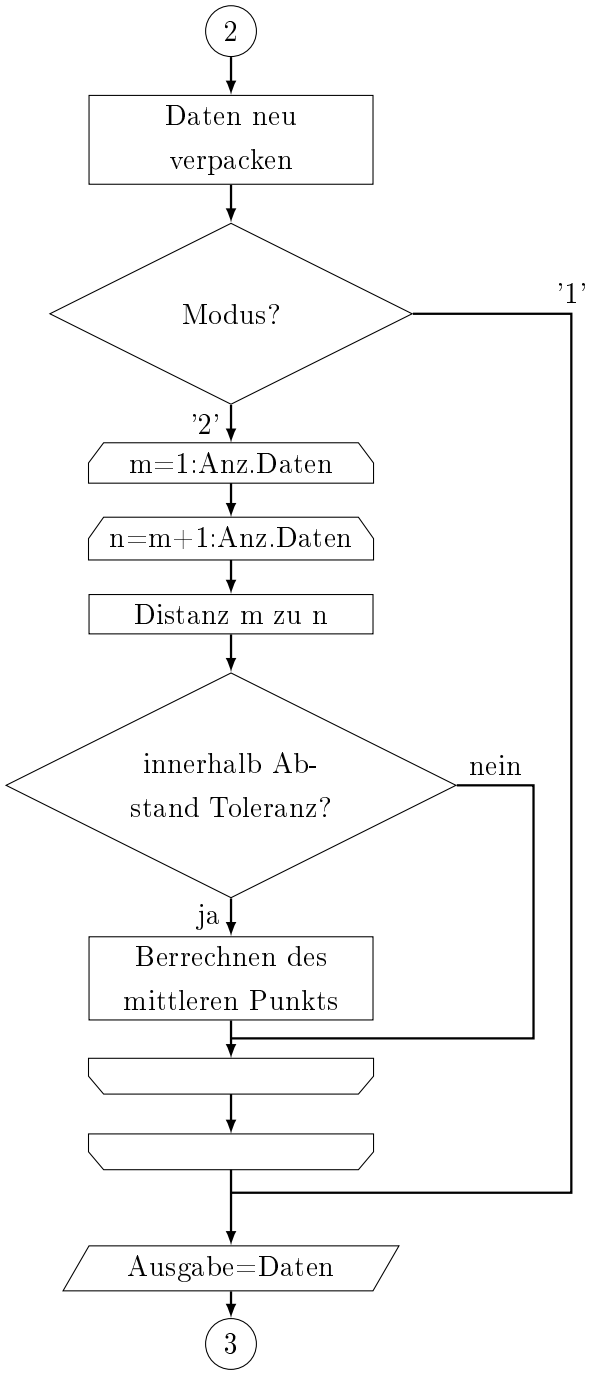


Abb. 5: Unterschied RGB- zu Depthbild



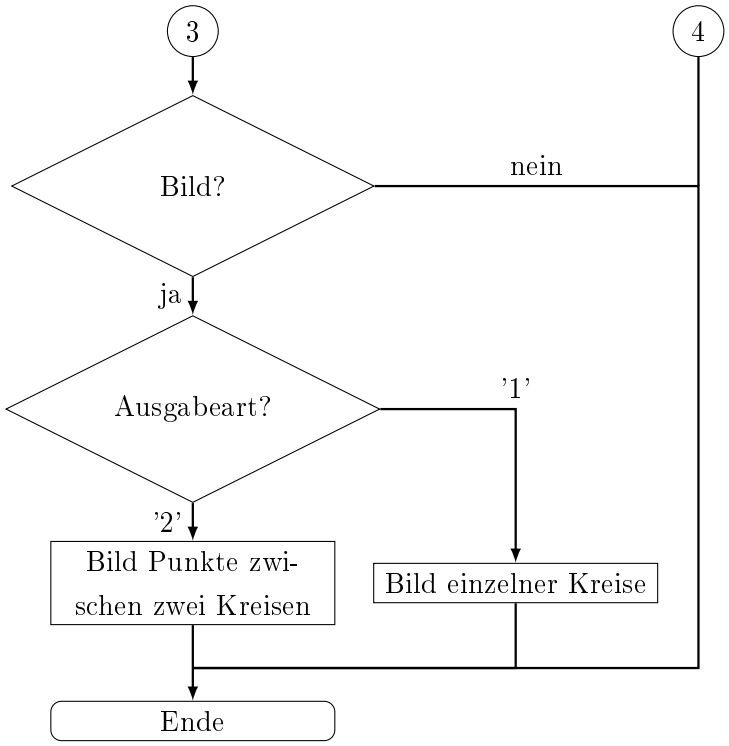
Nun werden alle Kreise, von denen die X,Y,Z,D Daten berechnet wurden, neu verpackt. Bei dem Modus wo die Kreise und deren Daten zur Position und Durchmesser gefragt sind, muss keine weitere Berechnung durchgeführt werden. Für den zweiten Modus, in dem die Abstände gefragt sind, müssen noch die Distanzen von allen Kreisen zueinander berechnet werden, siehe Formel 14.

$$Dis = \sqrt{((X(m) - X(n))^2 + \overline{(Y(m) - Y(n)^2))}} \tag{14}$$

Im Anschluss wird verglichen ob die Distanzen sich innerhalb der Toleranz für den gesuchten Abstand befinden. Falls das zutrifft wird der Mittelpunkt, der Kreise bei denen es zutrifft, (a) und (b), berechnet, siehe Formeln 15 bis 19. Die Tiefe für diesen Punkt wird nochmal neu mit dem Tiefenbild errechnet, siehe Formeln 20 und 21.

$$RX = (RX(a) + RX(b))/2 \tag{15}$$
$$RY = (RY(a) + RY(b))/2 \tag{16}$$
$$RD = (RD(a) + RD(b))/2 \tag{17}$$
$$DX = (DX(a) + DX(b))/2 \tag{18}$$
$$DY = (DY(a) + DY(b))/2 \tag{19}$$
$$DZ = imgDepth(DY, DX) \tag{20}$$
$$RZ = 564 - 1000 * (DZ) \tag{21}$$

Danach werden die berechneten Daten ausgegeben.



Der letzte Teil der Funktion befasst sich mit der möglichen Ausgabe der erkannten Kreise in einer Figur. Falls die Ausgabe gewünscht ist, wird noch unterschieden ob einzelne Kreise oder der Punkt zwischen zwei Kreisen angezeigt werden soll. Bei einzelner Anzeige wird jeder Kreis der innerhalb der Suchparameter liegt rot eingekreist angezeigt, siehe Abbildung 7. Die Anzeige der Abstände zeigt nur die Mittelpunkte von Kreispaaren an, die den richtigen Durchmesser und Abstand zueinander haben, siehe Abbildung 6.

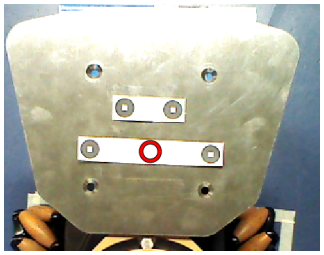


Abb. 6: Anzeige des Mittelpunktes

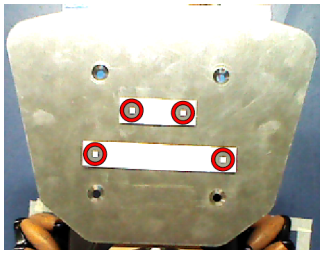


Abb. 7: Anzeige einzelner Kreise

Code KreisErkennung

```
% Kamera Kreiserkennung
% input : ROS, 'weiss'/'schwarz', 'einzel'/'doppel', ...
%          Durchmesser, Abstand, 'Dtol', wert, 'Sens', wert, ...
%          'Bild'
% input : ROS, 'w'/'s', '1'/'2', Durchmesser, Abstand
function Ausgabe=KreisErkennung(ROS,varargin)
%##################################################
% Default Werte
DurchmesserToleranz=3;
Sensitivity=0.7;%85;
ColorCode=[];
Modus=[];
Durchmesser=[];
Abstand=[];
AbstandToleranz=3;
Bildlich=0;
%##################################################
% Input Verarbeitung
% Zusuchende Farbe
inds = find(strcmpi('w',varargin), 1);
if ~isempty(inds)
    ColorCode='bright';
end
inds = find(strcmpi('s',varargin), 1);
if ~isempty(inds)
    ColorCode='dark';
end
```

```
28 % Modus
29 inds = find(strcmpi('1',varargin), 1);
30 if ~isempty(inds)
31     Modus='einzel';
32     Durchmesser=cell2mat(varargin(inds+1));
33 end
34 inds = find(strcmpi('2',varargin), 1);
35 if ~isempty(inds)
36     Modus='doppel';
37     Durchmesser=cell2mat(varargin(inds+1));
38     Abstand=cell2mat(varargin(inds+2));
39 end
40 % Durchmessertoleranz
41 inds = find(strcmpi('Dtol',varargin), 1);
42 if ~isempty(inds)
43     DurchmesserToleranz=cell2mat(varargin(inds+1));
44 end
45 % Abstandtoleranz
46 inds = find(strcmpi('Atol',varargin), 1);
47 if ~isempty(inds)
48     AbstandToleranz=cell2mat(varargin(inds+1));
49 end
50 % Sensibilitaet der Kreiserkennung
51 inds = find(strcmpi('Sens',varargin), 1);
52 if ~isempty(inds)
53     Sensitivity=cell2mat(varargin(inds+1));
54 end
55 % Bildliche Ausgabe
56 inds = find(strcmpi('Bild',varargin), 1);
```

```
57 if ~isempty(inds)
58     Bildlich=1;
59 end
60 %##################################################
61 % Struktur Bearbeitende Werte und Ausgabe
62 Data=struct('KameraX',[],'KameraY',[],'KameraZ',[],'KameraR',[],...
63     'DepthX',[],'DepthY',[],...
64     'RealX',[],'RealY',[],'RealZ',[],'RealR',[],'RealD',[],...
65     'Marker',[],'MValue',[]);
66 tempData=struct('X',[],'Y',[],'Z',[],'D',[],...
67     'KX',[],'KY',[],'KR',[],'DX',[],'DY',[],'Dis',[]);
68 tempData2=tempData;
69 Ausgabe=struct('X',[],'Y',[],'Z',[],'D',[],'Dis',[]);
70 %##################################################
71 % holen der Bilder, RGB und Depth, und umwandeln des RGB in Gray
72 imgDepth=readImage(ROS.Kamera.SubDepth.LatestMessage);
73 imgRGB=readImage(ROS.Kamera.SubRGB.LatestMessage);
74 imgGray=rgb2gray(imgRGB);
75 %##################################################
76 % Kreiserkennung und Markerkontrolle
77 Rmin=6; % Durchmesser Einstellung,
78 Rmax=18; % der zur suchenden [px] einheitlichen Kreise
79 % nur die Kreise suchen die gefragt sind, farblich
80 % aussortieren der Kreise ohne Marker
81 [centers, radii] = imfindcircles(imgGray,[Rmin Rmax],...
82     'ObjectPolarity',ColorCode,'Sensitivity',...
83     Sensitivity,'Method','twostage');
84 if ~isempty(centers)
85     for i=1:size(centers,1)
```

```
86          Data(i).KameraX=centers(i,1);
87          Data(i).KameraY=centers(i,2);
88          Data(i).KameraR=radii(i);
89          Data(i).MValue=imgGray(round(Data(i).KameraY,0),round(Data(i).KameraX,0));
90          % Marker ist bei weiss schwarz und vice versa
91          if strcmp(ColorCode,'bright')
92              if 0<=Data(i).MValue && Data(i).MValue<200
93                  Data(i).Marker=1;
94              else
95                  Data(i).Marker=0;
96              end
97          elseif strcmp(ColorCode,'dark')
98              if 201<Data(i).MValue && Data(i).MValue<=255
99                  Data(i).Marker=1;
100             else
101                 Data(i).Marker=0;
102             end
103         end
104     end
105     %#####################################################
106     % Berechnung des Durchmessers mittels der Tiefe Z
107     % nur bei denen die den Marker haben
108     DepthPosX=zeros(size(Data,2),1);
109     DepthPosY=zeros(size(Data,2),1);
110     for j=1:size(Data,2)
111         if Data(j).Marker==1
112             DepthPosX(j)=round(Data(j).KameraX+(Data(j).KameraX-130)*0.1,0);
113             DepthPosY(j)=round(Data(j).KameraY+(Data(j).KameraY-60)*0.1,0);
114             Data(j).DepthX=DepthPosX(j);
```

```
115             Data(j).DepthY=DepthPosY(j);
116             % Achtung Bildrand, schauen ob wir uns nahe des Rands befinden
117             area=5;
118             if DepthPosX(j)<1+area || DepthPosX(j)>640-area
119                 %disp('x<0 || x>640');
120             elseif DepthPosY(j)<1+area || DepthPosY(j)>480-area
121                 %disp('y<0 || y>480');
122             else
123                 % mehrere Z werte zusammen rechnen und teilen
124                 % fals der mittlerste Werte mal 'nan' ist
125                 anzahl=0;
126                 Zsum=0;
127                 for u=-area:2:area
128                     for v=-area:2:area
129                         if ~isnan(imgDepth(DepthPosY(j)+u,DepthPosX(j)+v))
130                             Zsum=Zsum+imgDepth(DepthPosY(j)+u,DepthPosX(j)+v);
131                             anzahl=anzahl+1;
132                         end
133                     end
134                 end
135                 Data(j).KameraZ=Zsum/anzahl;
136                 Z0=1000*Data(j).KameraZ;%mm
137                 FaktorZ=(Z0/530);
138                 R0=round((Data(j).KameraR*FaktorZ),2);
139                 Data(j).RealZ=564-Z0;
140                 Data(j).RealR=R0;
141                 Data(j).RealD=R0*2;
142                 %#####################################################
143                 % Berechnen der X Y Position
```

```
144             % nur bei denen die den Richtigen Durchmesser haben
145             if Data(j).RealR<(Durchmesser/2)+(DurchmesserToleranz/2) &&...
146                     Data(j).RealR>(Durchmesser/2)-(DurchmesserToleranz/2)
147                 Y0=round((240-Data(j).KameraY)*FaktorZ,2);% mm 240 == Bildhoehe/2
148                 X0=round((320-Data(j).KameraX)*FaktorZ,2);% mm 320 == Bildbreite/2
149                 Data(j).RealY=95+Y0;  % Nullpunktverschub
150                 Data(j).RealX=-12-X0; % Nullpunktverschub
151             end
152         end
153     end
154 end
155 %#####################################################
156 % Nach Modus entscheiden was gemacht wird
157 tempcnt=1;
158 for l=1:size(Data,2)
159     if ~isempty(Data(l).RealX)
160         tempData(tempcnt).X=Data(l).RealX;
161         tempData(tempcnt).Y=Data(l).RealY;
162         tempData(tempcnt).Z=Data(l).RealZ;
163         tempData(tempcnt).D=Data(l).RealR*2;
164         tempData(tempcnt).KX=Data(l).KameraX;
165         tempData(tempcnt).KY=Data(l).KameraY;
166         tempData(tempcnt).KR=Data(l).KameraR;
167         tempData(tempcnt).DX=Data(l).DepthX;
168         tempData(tempcnt).DY=Data(l).DepthY;
169         tempcnt=tempcnt+1;
170     end
171 end
172 if strcmp(Modus,'einzel')
```

```
173         % Kreise einzeln mit X Y Z D Werte ausgeben
174         for o=1:size(tempData,2)
175             Ausgabe(o).X=tempData(o).X;
176             Ausgabe(o).Y=tempData(o).Y;
177             Ausgabe(o).Z=tempData(o).Z;
178             Ausgabe(o).D=tempData(o).D;
179         end
180     elseif strcmp(Modus,'doppel')
181         % Kreise Abstand zueinander berechnen
182         % bei dem wo der Abstand stimmt Mittelpunkt berechnen und ausgeben
183         discnt=1;
184         auscnt=1;
185         Distanz=zeros(sum(1:size(tempData,2)-1),1);
186         for m=1:size(tempData,2)
187             for n=m+1:size(tempData,2)
188                 Distanz(discnt)=sqrt( (tempData(m).X-tempData(n).X)^2+...
189                     (tempData(m).Y-tempData(n).Y)^2)+2;
190                 if Distanz(discnt)<Abstand+AbstandToleranz && ...
191                         Distanz(discnt)>Abstand-AbstandToleranz
192                     tempData2(auscnt).Dis=Distanz(discnt);
193                     tempData2(auscnt).X=(tempData(m).X+tempData(n).X)/2;
194                     tempData2(auscnt).Y=(tempData(m).Y+tempData(n).Y)/2;
195                     tempData2(auscnt).D=(tempData(m).D+tempData(n).D)/2;
196                     tempData2(auscnt).KX=(tempData(m).KX+tempData(n).KX)/2;
197                     tempData2(auscnt).KY=(tempData(m).KY+tempData(n).KY)/2;
198                     tempData2(auscnt).KR=(tempData(m).KR+tempData(n).KR)/2;
199                     tempData2(auscnt).DX=round((tempData(m).DX+tempData(n).DX)/2,0);
200                     tempData2(auscnt).DY=round((tempData(m).DY+tempData(n).DY)/2,0);
201                     area=3;
```

```
202                 anzahl=0;
203                 Zsum=0;
204                 for p=-area:2:area
205                     for q=-area:2:area
206                         if ~isnan(imgDepth(tempData2(auscnt).DY+p,tempData2(auscnt).DX+q))
207                             Zsum=Zsum+imgDepth(tempData2(auscnt).DY+p,tempData2(auscnt).DX+q);
208                             anzahl=anzahl+1;
209                         end
210                     end
211                 end
212                 tempData2(auscnt).Z=564-round(1000*(Zsum/anzahl),2);%mm
213                 auscnt=auscnt+1;
214             end
215             discnt=discnt+1;
216         end
217     end
218     for o=1:size(tempData2,2)
219         Ausgabe(o).X=tempData2(o).X;
220         Ausgabe(o).Y=tempData2(o).Y;
221         Ausgabe(o).Z=tempData2(o).Z;
222         Ausgabe(o).D=tempData2(o).D;
223         Ausgabe(o).Dis=tempData2(o).Dis;
224     end
225 end
226 %#################################################
227 % Bildliche Ausgabe
228 if Bildlich==1
229     imshow(imgRGB);
230     % Einzeichnen der Kreise, '1' die echten, '2' die Mittelpunkte
```

```
231         if strcmp(Modus,'einzel')
232             for g=1:size(tempData,2)
233                 viscircles([tempData(g).KX tempData(g).KY], ...
234                     tempData(g).KR,'LineStyle','-','EdgeColor','r');
235                 txt=strcat('D:',num2str(Ausgabe(g).D));
236                 text(tempData(g).KX+10,tempData(g).KY,txt,'Color','black','FontSize',14)
237             end
238         elseif strcmp(Modus,'doppel')
239             for h=1:size(tempData2,2)
240                 viscircles([tempData2(h).KX tempData2(h).KY], ...
241                     tempData2(h).KR,'LineStyle','-','EdgeColor','r');
242                 txt=strcat('D:',num2str(Ausgabe(h).D),' Dis:', num2str(Ausgabe(h).Dis));
243                 text(tempData2(h).KX+10,tempData2(h).KY,txt,'Color','black','FontSize',14)
244             end
245         end
246     end
247 end
248 end
```