Westfälische Hochschule

University of Applied Sciences
Gelsenkirchen Bocholt Recklinghausen

Bachelorarbeit

Titel der Arbeit // Title of Thesis

Positionsermittlung eines Werkstücks/ Objekts mit einer 3D-ToF - Kamera, Visualisierung und Verarbeitung der Bilddaten mit dem Robot Operating System (ROS) und Matlab, geregeltes Greifen und Ablegend des Werkstücks auf einer Fertigungslinie mit dem Roboterarm des Kuka-youBots

Detecting the pose of a work piece with a 3D-Tof-Camera, visualisation and processing of the image data using the Robot Operating System (ROS) and Matlab, controlled gripping and depositting on an assembly line using the robotic arm of the Kuka-youBot

Akademischer Abschlussgrad: Grad, Fachrichtung (Abkürzung) // Degree

Bachelor of Engineering (B.Eng.)

Autorenname, Geburtsort // Name, Place of Birth

Flores Karsten, Bocholt

Studiengang // Course of Study

Mechatronik // Mechatronics

Fachbereich // Department

Maschinenbau // Engineering

Erstprüfer // First Examiner

Prof. Dr. Olaf Just

Zweitprüfer // Second Examiner

Prof. Dr. Horst Toonen

Abgabedatum // Date of Submission

16. November 2017

Eidesstattliche Versicherung

Flores Karsten

Name, Vorname // Name, First Name

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem Titel:

„Positionsermittlung eines Werkstücks/ Objekts mit einer 3D-ToF - Kamera, Visualisierung und Verarbeitung der Bilddaten mit dem Robot Operating System (ROS) und Matlab, geregeltes Greifen und Ablegend des Werkstücks auf einer Fertigungslinie mit dem Roboterarm des Kuka-youBots "

„Detecting the pose of a work piece with a 3D-Tof-Camera, visualisation and processing of the image data using the Robot Operating System (ROS) and Matlab, controlled gripping and depositting on an assembly line using the robotic arm of the Kuka-youBot"

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum, Unterschrift // Place, Date, Signature

Inhaltsverzeichnis

Abkürzungsverzeichnis

BSD Berkeley System Distribution

DDR Double Data-Rate

KUKA Keller und Knappich Augsburg

LAN Local Area Network

LTS Long Term Support

MATLAB Matrix Laboratory

NUC Next Unit of Computing

PAP Programmablaufplan

RAM Random-Access Memory

RGB Rot Grün Blau (additiver Farbraum)

ROS Robot Operating System

SSD Solid State Drive

TOF Time Of Flight

WLAN Wireless Local Area Network

Abbildungsverzeichnis

1. Einleitung

Das Verbinden von Sensoren und Aktoren ist ein wichtiger Teil der Mechatronik. Die Automatisierung von Prozessen in vielen industriellen Zweigen nimmt stetig zu. Dazu gehört das gezielte Verarbeiten von Daten die mittels Sensoren erfasst wurden und das Berechnen von darauf resultierenden erforderlichen Aktionen.

Das thematisierte Problem, in dieser Arbeit, ist die Erkennung eines bestimmten Objekts mittels optischen Sensoren sowie die Ansteuerung eines Roboterarms.

Dies wird mittels eines Roboterarms von KUKA umgesetzt. Alle Daten bezüglich der Position des zu greifenden Objekts werden mittels einer TOF-Kamera von Orbbec erfasst und mittels MATLAB berechnet.

Dadurch, dass die Basis des Arms verfahrbar bleiben soll, kommen nur eine am Arm montierte oder eine mittels weiterer Halterung am youBot befestigter Kamera in Frage. Doch die Fixierung am youBot wirft weitere Probleme auf:
Keinen fixen Nullpunkt, Rotation des Arms, ungenaue Positions-Wiederholgenauigkeit und weitere.

Der Schwerpunkt liegt in der Erkennung von Objekten sowie dem Ansteuern eines Roboterarms. Es wird eine Inverse Kinematik zur Berechnung der Winkelstellungen verwendet. Diese ist nur eine vereinfachte geometrische Variante, die nur in angepassten Fällen verwendet werden kann.

Diese Arbeit ist wie folgt strukturiert.
Zunächst wird die Aufgabenstellung spezifiziert und das zu erwartende Ergebnis beschrieben. Danach werden die zur Verfügung stehenden Mittel erläutert. Darauf folgt eine Beschreibung des Grundgedankens zur Lösung der Aufgabe, als auch die zu erwartenden Fehlerquellen und Problemzonen. Des Weiteren wird das Erstellen eines Modellarbeitsplatzes, als auch die Einrichtung der Hard- und Software beschrieben. In dem darauf folgenden Kapitel werden die schon vorhandenen, beziehungsweise selbst erstellten Strukturen, die für den Datenverkehr benötigt werden, erklärt. Im Kapitel 7 werden die Grundstruktur und die wichtigsten Funktionen dokumentiert. Beendet wird die Arbeit mit der Schlussbetrachtung und dem Anhang, welches den Programmcode zu den beschriebenen Funktionen enthält.

Das zentrale Ziel ist es ein funktionierendes Programm zu schreiben welches selbstständig Objekte aufnimmt und in freie Ladestellen ablegt, bei voller Ladung einen Ortswechsel simuliert und dann die Objekte von den Ladestellen auf, beziehungsweise zwischen, Ablagesymbolen ablegt. Dies soll in einem ständigen Kreislauf funktionieren.

2. Aufgabenstellung

Die Unterteilung in drei Bereiche und die Aufgabe dieser wird im folgenden kurz erläutert. Zusätzlich wird eine Erweiterungsmöglichkeit beschrieben.

2.1. Spezifizierte Aufgabe

Die grundlegende Aufgabe ist das Erkennen von Objekten mittels einer Kamera. Die daraus resultierenden Daten werden für die Ansteuerung des Arms benötigt um das Aufheben und Ablegen der Objekte zu realisieren. Diese Aufgabe wird größtenteils von drei Unterprogrammen erledigt:

- Die 'Kreiserkennung' liefert die Daten (XYZ-Koordinaten, Durchmesser, Abstände) bezüglich der Position des Objekts,
- die 'Ik' errechnet aus diesen Daten die Winkelstellungen für den Arm,
- 'GelenkPos' und 'GreiferPos' regeln das Anfahren der Winkelstellungen des Arms, sowie das Öffnen und Schließen des Greifers.

2.2. Verbindung mit weiterer Bachelorarbeit

Weiterführend ist eine Verbindung zu der Bachelorarbeit von Sebastian Flores möglich. Diese Arbeit befasst sich mit dem autonomen Verfahren, der youBot Basis, mittels Laserscanner. Somit ist ein Verfahren zwischen dem Aufheben und Ablegen möglich, was einer realitätsnahen Bestückung und Ablieferung einer Taktstraße bzw. eines Förderbands gleich kommt.

3. Stand der Technik

Die in dieser Bachelorarbeit verwendete Hard- und Software wird im folgenden kurz erläutert und aufgelistet. Im Falle der Software werden die Webseiten zum Herunterladen angegeben.

3.1. youBot

Der verwendete youBot ist ein für die Forschung und Lehre konzipierter Roboter von KUKA.

Der Name KUKA entstand durch das Abkürzen der Gründernamen und des Gründungsorts: **K**eller **u**nd **K**nappich **A**ugsburg. Die Firma wurde 1898 gegründet und hatte zunächst das Ziel günstige Haus- und Straßenbeleuchtungen zu produzieren. Danach folgten Innovationen in der Schweißtechnik und erst 1973 entwickelte KUKA den weltweit ersten Industrieroboter mit sechs elektromechanisch angetriebenen Achsen. [vgl. KUKa]

Die youBot Basis besitzt omnidirektionale Reifen, welche Drehungen ohne Lenkeinschlag sowie direktes seitwärts Fahren ermöglichen. Zusätzlich ist es möglich, auf den zwei Montagestellen, die sich auf dem youBot befinden, jeweils einen KUKA Arm oder eine Ladefläche zu montieren. Da diese Arbeit sich nur mit dem Arm beschäftigt, wird nicht weiter auf die Basis eingegangen.

Der KUKA Arm wurde mit der ROS Umgebung verbunden, dies geschah mittels den youBot Treibern. Damit ist es möglich, durch die Vorgabe von Positionen als auch der Geschwindigkeit, einzelne Gelenke anzusteuern. Die einzelnen Zustände der Gelenke, Positionen und Geschwindigkeiten, können ausgelesen werden. [vgl. Now]

Abb. 3.1.: youBot Arm

youBot KUKA Arm Eigenschaften:

- Anzahl der Freiheitsgrade: 5
- Höhe: 655 mm
- Arbeitsumfeld: 0,513 m²
- Gewicht: 5,8 kg
- Nutzlast: 0,5 kg
- Struktur: Gussform aus Magnesium
- Wiederholgenauigkeit
  in Position: 0.1 mm
- Kommunikation:
  EtherCAT: Frequenz 1ms
- Spannung: 24 V DC
- Leistung auf 80 W beschränkbar
- Axialgeschwindigkeit: 90 Grad/s

[vgl. KUKb]

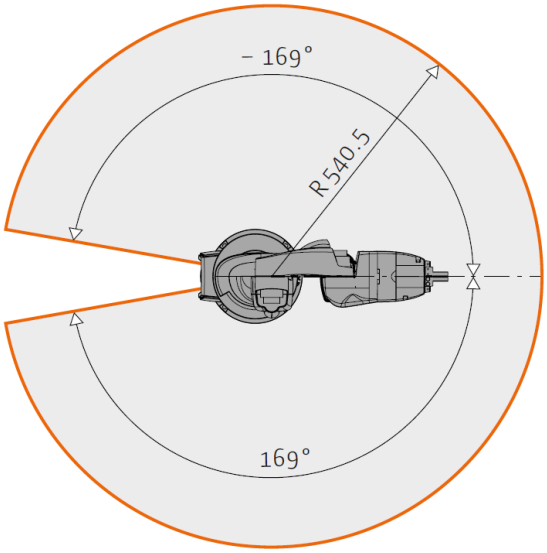

Abb. 3.2.: Gelenke X-Y Ebene

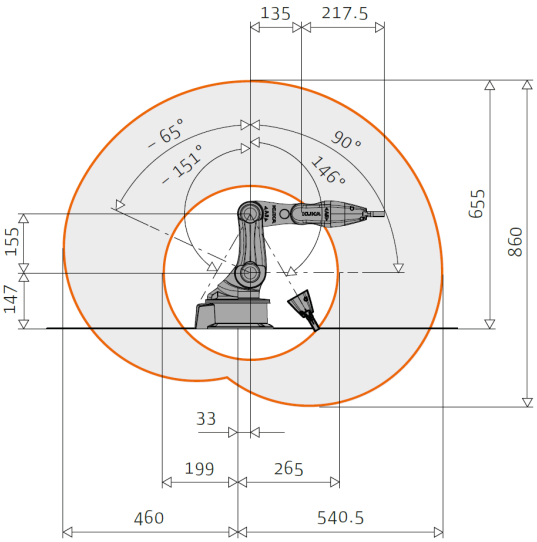

Abb. 3.3.: Gelenke Y-Z Ebene

3.2. IntelNuc

Abb. 3.4.: IntelNUC

Der verwendete PC, ist ein Mini PC aus der Intel NUC Reihe. Die genaue Bezeichnung lautet: NUC7i3BNH. Dieser hat einen Intel®Core™i3 der siebten Generation, eine 128GB Intel SSD Festplatte und 8GB (DDR4) RAM verbaut. Der PC ist bei dem Modell Versuch mittels LAN-Kabel direkt am Arm angeschlossen [vgl. Flo]. Die Berechnung der Erkennung von Objekten als auch die Steuerung des Arms wird von diesem PC vorgenommen. Dies benötigt eine gewisse Geschwindigkeit, der i3 ist für diese Aufgabe geeignet. Schnellere Prozessoren aus der i5 oder i7 Reihe wurden aufgrund des höheren Energiebedarfs nicht verwendet.

3.3. Ubuntu

Das gewählte Betriebssystem ist Linux Ubuntu 16.04LTS. Wichtige Eigenschaften des gewählten Systems: Es bietet LTS (Long Term Support) und unterstützt die aktuelle ROS Version. Es wurde nicht das neuste System 18.04LTS gewählt, da die ROS Umgebung für diese Version noch in der Entwicklung war (Stand: 09.2017).

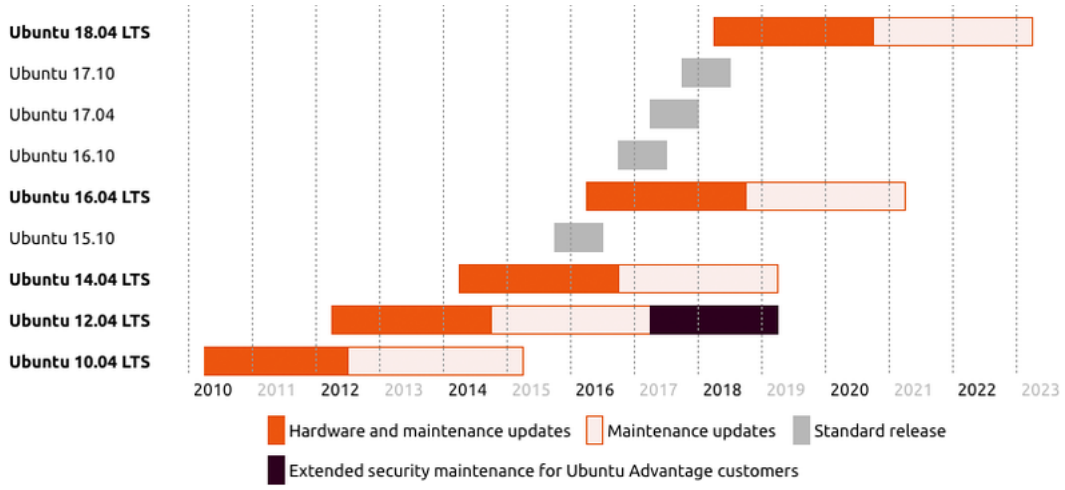

Abb. 3.5.: Ubuntu LTS

3.4. ROS

Die Bibliotheken und Werkzeuge, die das ROS (Robot Operating System) zur Verfügung stellt, ermöglichen die Softwareentwicklung für Roboteranwendungen. Zusätzlich sind in diesem System Hardware Abstraktionen, Gerätetreiber, Visualisierungen, Nachrichtenvermittlungen, Paketverwaltung und weitere Komponenten verfügbar. Es ist unter der Open Source BSD (Berkeley System Distribution) Lizenz veröffentlicht, dadurch ist eine Anwendung und Weiterentwicklung ohne den Kauf gewisser Lizenzen möglich. [vgl. Fab]

Abb. 3.6.: Kinetic Kame ROS

Die verwendete Version hat den Namen 'ROS Kinetic Kame'. Es ist die zehnte veröffentlichte ROS Version, die am 23. Mai 2016 heraus gebracht wurde. Sie wurde primär für die Ubuntu 16.04 Version programmiert. Andere Betriebssysteme von Linux, Mac OS X, Android oder Windows werden nur zu verschiedenen Graden unterstützt. [vgl. Tul]

3.5. MATLAB

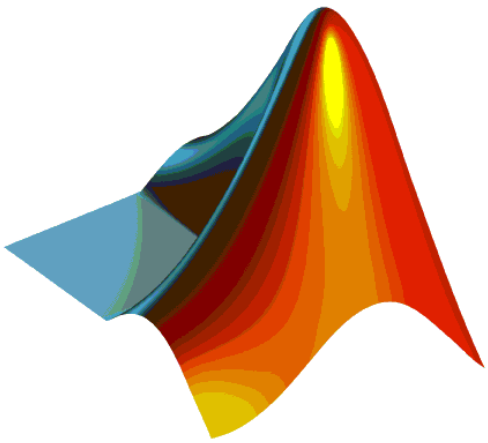

Abb. 3.7.: MATLAB Logo

MATLAB®ist ein Produkt von MathWorks, eine weltweit agierende Firma, die führend in der Entwicklung technischer Software für mathematische Berechnungen und modellbasierter Entwicklung ist. MATLAB®, „the language of technical computing“, besitzt eine Programmierumgebung die für Algorithmen-Entwicklung, Analyse und Visualisierung von Daten als auch, nummerische Berechnungen ausgelegt ist. [vgl. MATb]

3.6. 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. 3.8.: 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.

Orbbec Astra Eigenschaften:

- Größe 165 x 30 x 40 mm
- Gewicht 0,3 kg
- Reichweite 0,6 – 8,0m (Optimal 0,6 – 5,0m)
- Tiefen Bild (Depth) Größe
  - 640*480 (VGA) @ 30FPS
  - 320*240 (QVGA) @ 30FPS
  - 160*120 (QQVGA) @ 30FPS
- Farbbild (RGB) Größe
  - 1280*960 @ 7FPS
  - 640*480 @ 30FPS
  - 320*240 @ 30FPS
- Sichtfeld 60° horizontal x 49.5° vertikal (73° diagonal)
- Datenschnittstelle USB 2.0
- 2 Mikrophone
- Betriebssysteme Windows 7/8/10, Linux, Android
- Stromversorgung über USB 2.0
- Software Astra SDK or OpenNI 2 or 3rd Party SDK

[vgl. Orbb]

4. Planung

Dieses Kapitel befasst sich mit der Festigung der Grundidee und die dabei zu beachtenden Fehlerquellen und Problemzonen.

4.1. Grundgedanke

Der Plan des Ablaufs sieht zwei ähnliche Ablaufstränge vor in denen nur zwischen dem Aufheben beziehungsweise Ablegen unterschieden wird.

- (Ablegen: vorhergehendes Aufheben aus fixierter Ladestelle)
- ausrichten des Arms sodass die Kamera auf den Bereich ausgerichtet ist in dem sich das zu erkennende Symbol befinden sollte
- die Aufnahmen von RGB und Tiefen Bildern
- die Berechnung des Drehwinkels zum nächstliegendsten Symbol
- die Ausrichtung zum Symbol
- neue Aufnahmen von RGB und Tiefen Bildern
- berechnen der Winkelstellungen der Gelenke 2, 3 und 4
- anfahren einer Vorposition (Aufheben: Greifer öffnen)
- anfahren der Greifposition (Aufheben: Greifer schließen, Ablegen: Greifer öffnen)
- anfahren einer Vorposition
- anfahren einer Ruheposition

4.2. Fehlerquellen

Bei der optischen Erkennung kann es vorkommen, dass durch schlechte Beleuchtung falsche oder keine Kreise erkannt werden. Dies wird durch den Versuch der immer gleichen Beleuchtung, Abdunkeln des Raums und Einschalten der Lampen, weitestgehend verhindert. Durch Strukturen im Hintergrund, wie gemusterte Böden oder Schmutz, ist die Erkennung nicht immer fehlerfrei möglich, doch bei dem im Labor vorhandenen Bodenbelag ist dies sehr selten bis gar kein Problem. Bei der Positions-Wiederholgenauigkeit des Arms kann es zu geringen Abweichungen kommen, da diese sich im Bereich von ± 1mm befinden kommt es aber meistens zu keinem Problem.

4.3. Problemzonen

Beim Verfahren des Arms kann es vorkommen, dass ein ausgerechneter Punkt sich in einem Bereich befindet, in dem der Arm in Kontakt mit der Basis oder sich selbst kommt. Im momentanen Programm besteht noch kein Sicherheitssystem, welches eine Kollision mit sich selbst verhindern würde, deswegen ist das Programm nur unter Aufsicht auszuführen. Weiterhin wäre ein Kontakt nicht zwingend mit dem Beschädigen des Roboters verbunden. Da es sich um einen Schulungsroboter handelt, sind alle Gelenke so eingestellt das diese nur eine geringe Kraft aufbringen können. Dies führt zu einer geringen Traglast des youBots. So ist es möglich den Arm mit einer Hand festzuhalten.

5. Vorbereitung

Im folgenden Kapitel wird das Errichten eines Model basierten Arbeitsplatzes, als auch das Einrichten der Software, beschrieben.

5.1. Armstation

Das Einrichten einer Armstation, siehe Abbildung 5.1, welches ein vereinfachtes Modell einer youBot Basis ist, ermöglicht ein getrenntes Arbeiten von der eigentlichen Basis.

Abb. 5.1.: Armstation

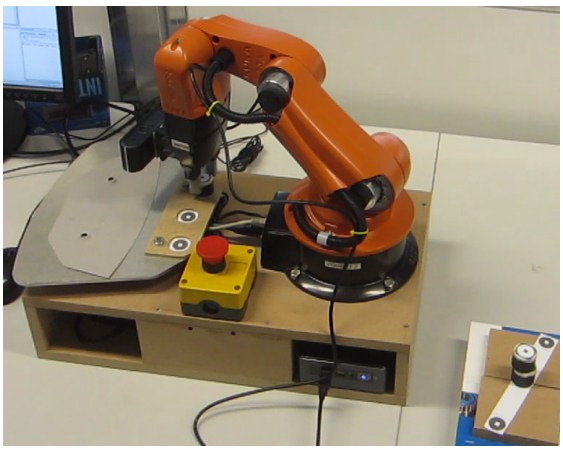

Abb. 5.2.: Versuchsaufbau

Die Maße des Modells wurden so gewählt das sie der Basis nahe kommen. Dadurch können alle auf dem Modell eingerichteten Programme, nur mit geringen Änderungen, auch auf der Basis verwendet werden. In Abbildung 5.2 ist der Versuchsaufbau sichtbar, die Ladestellen auf der Ladefläche als auch die Ablagestelle vor dem Modell sind so positioniert, dass der Arm diese erreichen kann ohne vollends ausgefahren zu sein.

5.2. Kamerahalterung

Die Befestigung der Kamera wird an der dem youBot beiliegenden Lochblechhalterung, die zwischen Arm und Greifer montiert wird, geschraubt.

Abb. 5.3.: Astra Halterung

Die mitgelieferte Halterung der Astra Kamera, siehe Abbildung 5.3, war für eine stabile Positionierung ungeeignet, da sie ein Kippgelenk besitzt und nur durch eine Schraube befestigt werden kann. Durch die Anfertigung einer Verbindungsplatte (Metall) und einer Abstandsplatte (Holz) kann die Kamera fest an das Lochblech montiert werden, siehe Abbildung 5.4. So ist eine Veränderung der Position des Arms relativ zur Kamera nicht möglich, dadurch können alle berechneten Positionen auf einen Nullpunkt bezogen werden. So zeigen sich nur noch Abweichungen durch die Ungenauigkeit des Arms.

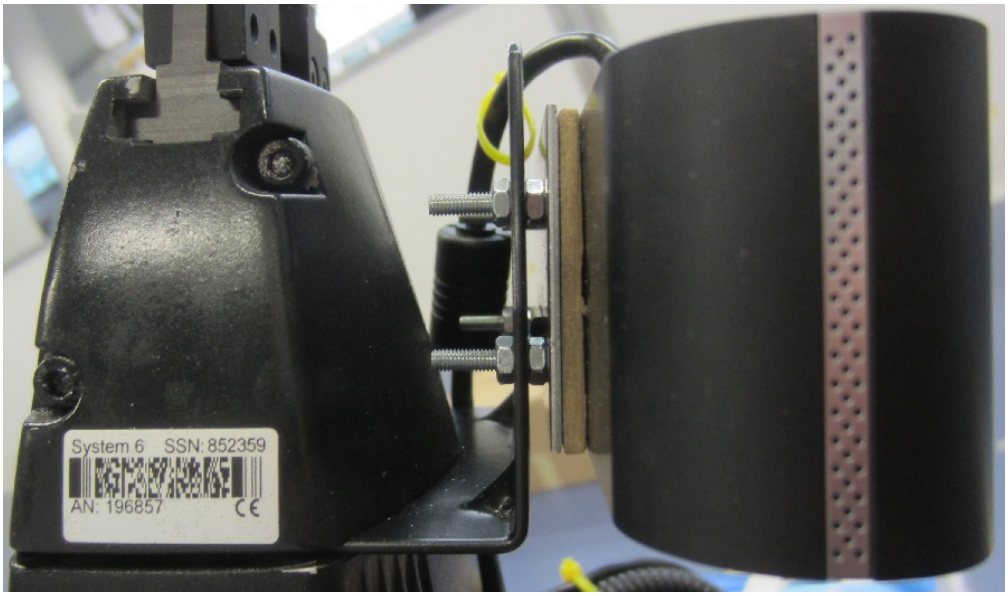

Abb. 5.4.: selbst erstellte Halterung

5.3. Einrichten der Software

Das Einrichten teilt sich in drei Bereiche auf. Der Treiber für den youBot, der Treiber für die Kamera sowie die Add-ons für Matlab. Zusätzlich wird bei den Treibern erklärt wie diese zu starten sind.

5.3.1. Treiber youBot

Der bereits Installierte youBot Treiber wurde den Umständen entsprechend angepasst. Im Versuchsaufbau wurde der Treiber so konfiguriert, dass nur die Teileinstellungen, die die Arm Steuerung benötigt, gestartet werden. In der Datei 'youbot_driver.launch', die sich innerhalb der Catkin Umgebung ('catkin_ws/src/youbot _driver_ros_interface/launch/') befindet, wurde der boolesche Operator 'youbotHasBase' deaktiviert und die 'robot_decription' wurde auf 'youbot_arm_only.urdf.xacro' gesetzt. Die 'youbotHasBase' Variable beschreibt ob der youBot eine Basis besitzt, dementsprechend werden die dafür benötigten Konfigurationen geladen. Die 'robot_decription' enthält Daten über die Dimensionen des youBots, in diesem Fall nur über den Arm.
Zum Starten des Treibers wird der folgende Befehl im Terminal ausgeführt:

'roslaunch youbot_driver_ros_interface youbot_driver.launch'

5.3.2. 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')'

5.3.3. Add-ons Matlab

Die für diese Arbeit benötigten Add-ons beschränken sich auf die 'Robotics System Toolbox', welche alle Befehle für die ROS Verbindung enthält, sowie die 'Image Processing Toolbox', aus dieser wurde nur der Befehl 'imfindcircles' verwendet. Auf diesen wird in Kapitel 7.5, innerhalb der Kreiserkennung, genauer eingegangen.

6. Strukturen

Im folgenden Kapitel werden die Strukturen der Treiber, des youBots und der Kamera, als auch die Struktur die in MATLAB angelegt wurde, beschrieben.

6.1. Daten des youBots

Die ausgelesenen Daten des youBot Arms, beziehen sich auf die Winkelstellungen und Geschwindigkeiten der einzelnen Gelenke. Das Topic, welches dafür benutzt wurde, ist 'Joint_states'. Zum Steuern des Arms werden drei Bereiche innerhalb des Treibers angeschrieben, siehe Abbildung 6.3.

6.2. Daten der Kamera

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

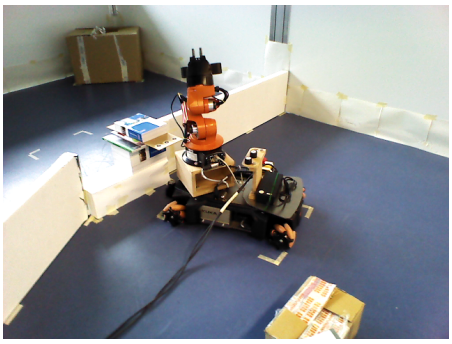

Abb. 6.1.: RGB Bild

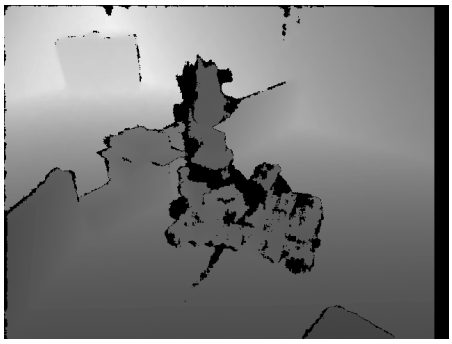

Abb. 6.2.: Tiefen Bild

In Abbildung 6.2 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.

6.3. ROS Umgebung

Zum Erstellen einer Verbindung mit dem ROS-Master Server, entweder über die Lokale Adresse oder über eine WLAN Verbindung, wird eine IP-Adresse benötigt. Dies wird mit der Funktion 'ip=getIP()' erreicht. Diese Funktion greift von MATLAB aus, mittels eines System Befehls, die aktuellen Verbindungen ab.

$$'[\sim, result]=\text{system('ifconfig | grep "inet Adresse"')};'$$
$$'\text{str=strsplit(result,' ',':')};'$$

Wenn keine LAN oder WLAN Verbindung besteht:

str='inet Adresse:127.0.0.1 Maske:255.0.0.0'

Wenn eine LAN oder WLAN Verbindung besteht:

str='inet Adresse:127.0.0.1 Maske:255.0.0.0
inet Adresse:192.168.19.40 Bcast:192.168.19.255 Maske:255.255.252.0'

Mit der daraus resultierenden IP-Adresse wird eine Verbindung mit dem ROS-Server erstellt.

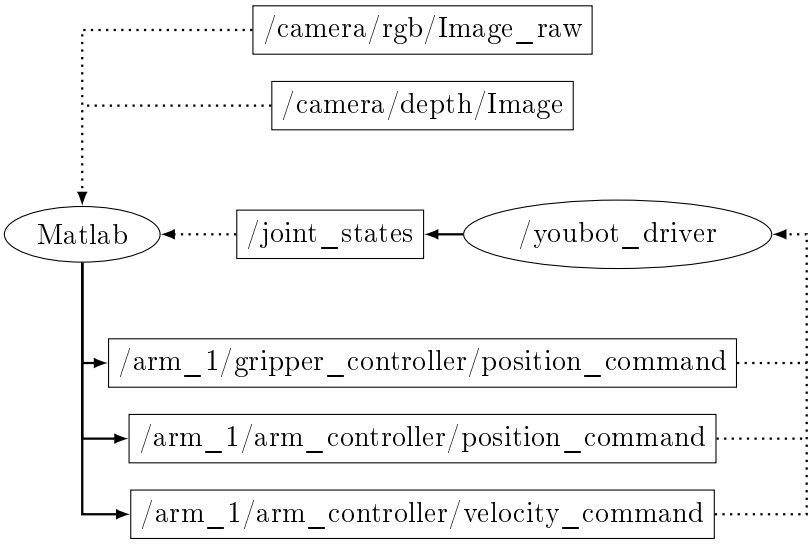

Abb. 6.3.: ROSdiagramm

Das nebenstehende 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'.

6.4. 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
  - Lade
    - Info
      - LadePositionen

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 Abbildung 6.2 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.

7. Programm

Das gesamte Programm besteht aus einem Hauptprogramm und 19 Funktionen.

- Hauptprogramm
  - **Ablauf**
- Funktionen
  - getIP
  - conRos
  - runROS
  - infoArm
  - infoLade
  - **LadeCheck**
  - **KreisErkennung**
  - **VorIk**
  - **Ik**
  - **GelenkPos**
  - **Ablegen**
  - **Aufheben**
  - **GreiferPos**
  - interpolieren
  - PosAufnehmen
  - PosMinDis
  - stop

In diesem Kapitel werden die wichtigsten Funktionen, fett markiert, erklärt und teilweise mittels Programmablaufplänen dargestellt.
Die Funktionen 'getIP', 'conROS', 'runROS', 'infoArm' und 'infoLade' erstellen die Verbindung und legen alle benötigten Strukturen an. Die Unterprogramme 'interpolieren', 'PosAufnehmen' und 'PosMinDis' erledigen weitere Umrechnungen oder Sortierungen. Die Funktion 'stop' sendet einen Befehl an den Arm, der diesen zum stehen bleiben bringt, falls kein noch laufendes Programm diesen Befehl überschreibt. Um das Programm zu starten müssen folgende Schritte ausgeführt werden:

- neues Terminal: 'roscore'
- neuer Tab:
  'roslaunch astra_launch astra.launch'
- neuer Tab:
  'roslaunch youbot_driver youbot_driver_ros_interface.launch'
- MATLAB: Ablauf starten

Das Tastenkürzel 'Strg+Alt+T' kann zum Starten eines neuen Terminals verwendet werden und das Kürzel 'Strg+Shift+T' zum Öffnen eines neuen Tabs im offenen Terminal.

7.1. Grundstruktur

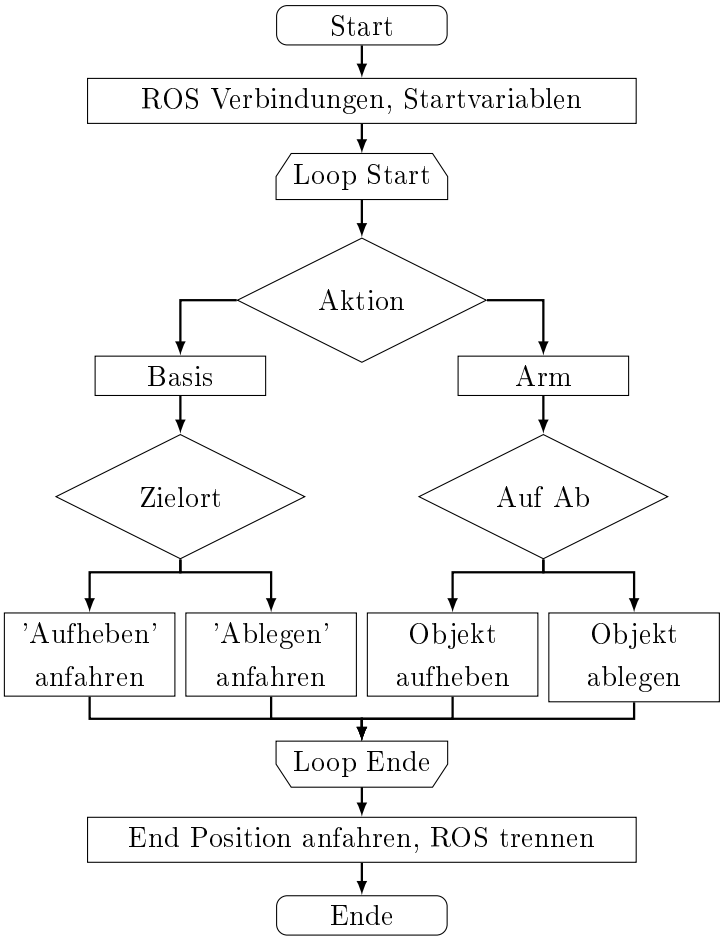

Abb. 7.1.: PAP 'Ablauf'

Der nebenstehende PAP (Programmablaufplan) des Hauptprogramms 'Ablauf', gibt eine Übersicht zu einer möglichen Kombinierten Ausführung des Programms an, wo ein Aufheben und Ablegen mit einem Ortswechsel verbunden wird. Der Zweig in dem die Basis verfährt ist in dieser Arbeit irrelevant, es geht nur um den Pfad in dem der Arm das Aufheben und Ablegen ausführt.

Zu beginn des Programms werden alle Verbindungen mit dem Befehl 'ROS=runROS;' erstellt. Somit befinden sich, wie in Kapitel 6.4 (MATLAB Umgebung) aufgezeigt, alle wichtigen Informationen in der ROS Variable. Der Befehl 'Ladung=LadeCheck;' kontrolliert wie viele Objekte sich auf der Ladefläche befinden. Die Entscheidung welche Aktion als nächstes ausgeführt wird, hängt von der Ladung ab, bei mindestens einem Objekt wird zunächst das Ablegen ausgelöst und bei null dahingegen erst das Aufnehmen. Beim Ablegen und Aufheben wird solange die Tätigkeit wiederholt bis die Ladefläche entweder leer oder voll ist, beziehungsweise sich kein weiteres Objekt im Sichtbereich befindet. Dann wird wiederum die jeweils andere Aktion gestartet, hierzwischen wäre das Verfahren der Basis möglich.

7.2. Ladecheck

'Ladung=LadeCheck(ROS);'

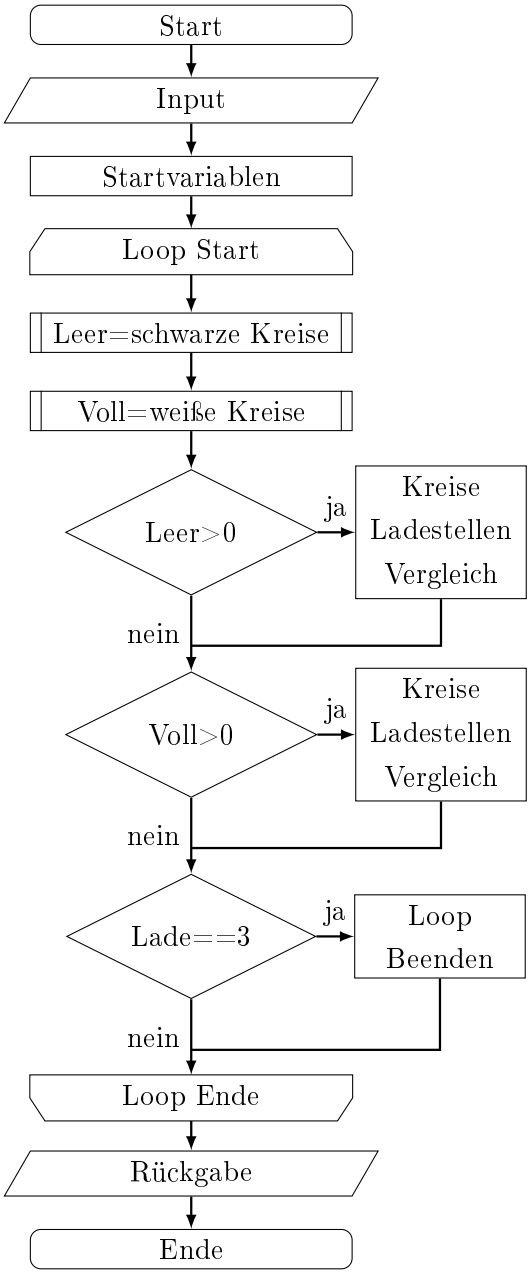

Abb. 7.2.: PAP 'LadeCheck'

Die Funktion 'LadeCheck' wird am Anfang des Hauptprogramms ausgeführt um den Status der Ladefläche zu erhalten. Diese hat eine Eingabe 'ROS' mit dem alle benötigten Informationen übergeben werden sowie eine Ausgabe 'Ladung'. Der Rückgabewert 'Ladung' gibt Auskunft über den momentanen Zustand der Ladefläche. Es ist ein Array mit drei Stellen. Bei einer erkannten leeren Ladestelle wird die zugehörige Position im Array auf '0' gesetzt, bei einer vollen auf '1'. Bei nicht erkennen eines Symbols im Bereich der Ladestelle wird die betreffende Ladestelle auf '-1' gesetzt.

In 'Leer=schwarze Kreise' und 'Voll=weiße Kreise' werden mittels der Kreiserkennung alle weißen und schwarzen Kreise innerhalb des Sichtbereichs mit dem eingestellten Durchmesser gespeichert. Das Vergleichen der Kreismittelpunkte mit den eingespeicherten Ladepositionen erfolgt danach. Falls alle drei Ladestellen direkt auf '0' oder '1' gesetzt werden konnten, wird der Loop beendet. Beim nicht Erkennen einer oder mehrerer Ladestellen wird das Programm nach mehreren Versuchen beendet. Die Ladestellen die nicht gefunden wurden, werden auf '-1' gesetzt.

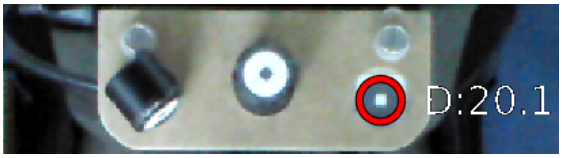

Abb. 7.3.: Schwarze Symbole bezeichnen leere Ladestellen

Abb. 7.4.: Weiße Symbole bezeichnen volle Ladestellen

Da die Position der Ladestellen nicht durch die Kreiserkennung ermittelt werden, sondern nur deren Zustand, sind die Winkelparameter zu den einzelnen Positionen vorher schon bestimmt und gespeichert worden. Der Speicherort, dieser Daten, befindet sich in der MATLAB Umgebung (Kapitel 6.4) innerhalb der ROS-Struktur unter Lade.Info.Ladepositionen. Die drei Positionen sind:

'Ladepositionen(1,:)=[1.25-15 -45.5 -45 -90.25 5];'
'Ladepositionen(2,:)=[1.25 -45.5 -45 -90.25 5];'
'Ladepositionen(3,:)=[1.25+15 -45.5 -45 -90.25 5];'

Dadurch das die Positionen fest eingespeichert sind, wird eine weitere Drehung des Arms um die Y-Achse vermieden. Das bedeutet, dass die Objekte immer in einer Überkopfbewegung von Ablage zu Aufnahme beziehungsweise von Aufnahme zu Ablage transportiert werden.

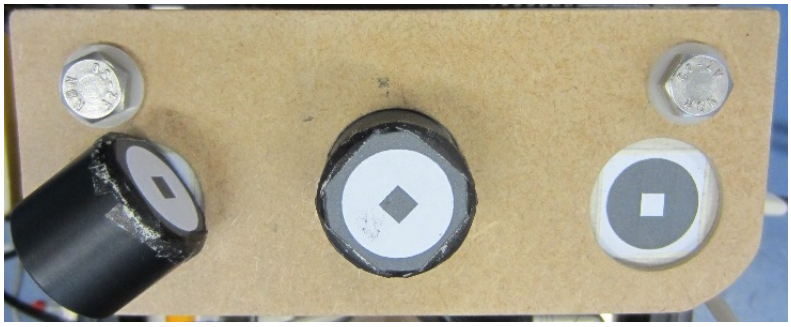

Abb. 7.5.: Ladestellen

Das Beispiel, siehe Abbildung 7.5, würde bedeuten das die erste Ladestelle nicht erkannt werden kann, die zweite voll, und die dritte leer ist. In der Variable 'Ladung' wäre somit '[-1,1,0]' gespeichert.

7.3. Ablegen und Aufheben

'[Abgelegt, LeererSichtbereich]=Ablegen(ROS,nr)'

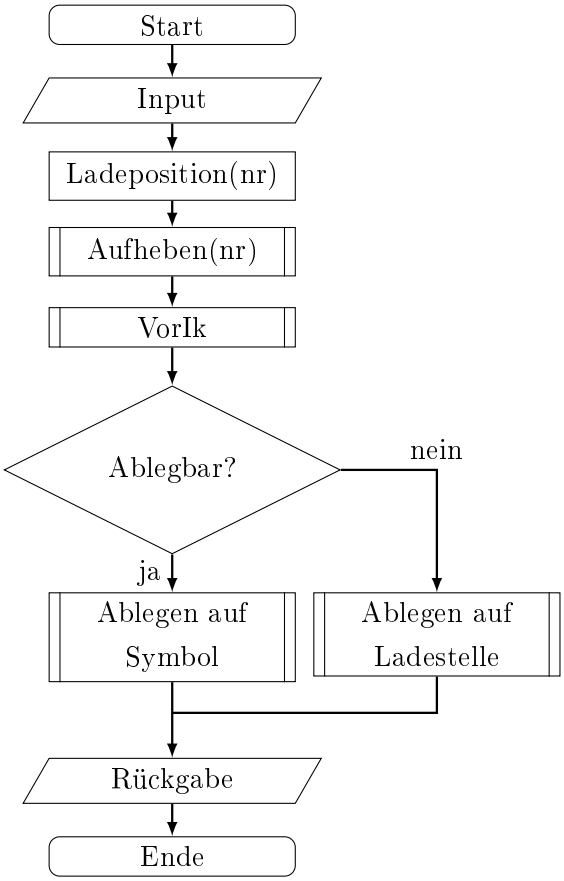

Abb. 7.6.: PAP 'Ablegen'

Die Funktion zum Ablegen von den Objekten benötigt zwei Eingaben, die ROS Informationen und die Ladestellennummer. Die ROS Informationen werden zum Auslesen und Ansteuern benötigt. Die Nummer entscheidet darüber aus welcher Ladestelle ein Objekt heraus genommen wird. Der Rückgabewert 'Abgelegt' verändert in der Variable 'Ladung' an der Position 'nr' den Zustand bei erfolgreichem Ablegen auf '0' und bei scheitern auf '1'. Die Variable 'LeeresSichtbereich' gibt zusätzlich Auskunft über den Sichtbereich. Falls dieser leer ist, das heißt kein Symbol zum Ablegen erkannt wurde (mehrere Versuche), wird in einer höheren Instanz trotz noch nicht leerer Ladefläche die Aktion gewechselt, in diesem Fall von Ablegen zu Aufheben. Das Unterprogramm 'VorIk' richtet den Arm zum nächsten Symbol aus. Danach wird Überprüft ob sich ein Symbol zum Ablegen im Sichtbereich befindet und dies in Reichweite ist. Falls dies zutrifft wird das Objekt auf das Ablagesymbol gelegt, falls nicht wird es in die Ladestelle zurückgelegt.

'[Aufgehoben, LeererSichtbereich]=Aufheben(ROS,nr)'

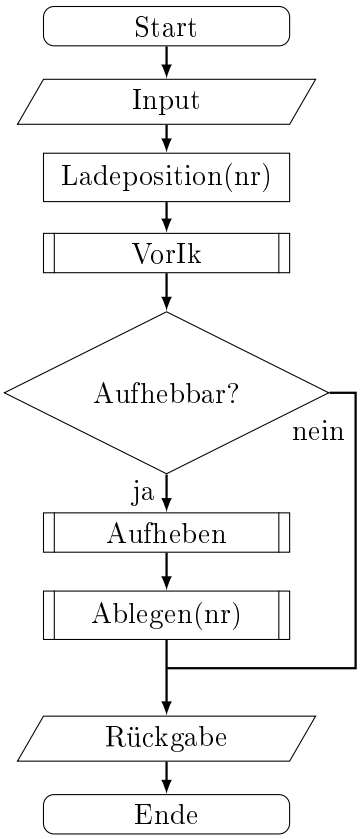

Abb. 7.7.: PAP 'Aufheben'

Die Funktion zum Aufheben funktioniert in größten Teilen genauso wie die zum Ablegen. Die Unterschiede sind der Rückgabewert, hier 'Aufgehoben', ändert bei erfolgreichem Aufheben die Ladestelle auf '1' und bei scheitern auf '0'. Bei dieser Funktion kommt Aufheben und Ablegen in einer anderen Reihenfolge vor.

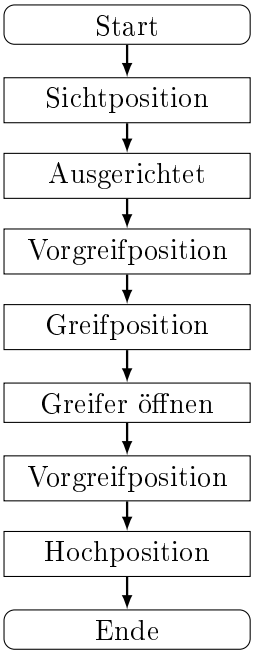

Abb. 7.8.: Bewegungsablauf Ablegen

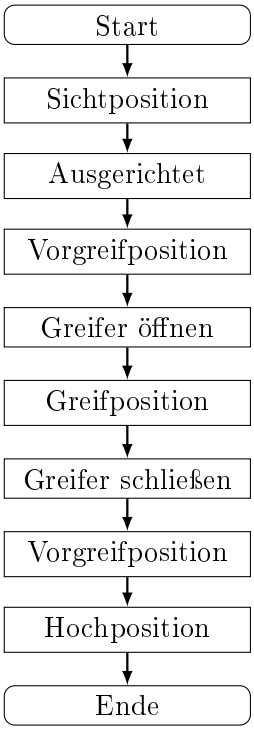

Abb. 7.9.: Bewegungsablauf Aufheben

Der Bewegungsablauf des Ablegens und Aufhebens sind nahezu identisch und unterscheiden sich nur darin wann sich der Greifer öffnet bzw. schließt. Die erste Position, die angefahren wird, ist die Sichtposition. In dieser Position wird die Funktion 'VorIk' gestartet welche den Arm zum Symbol hin ausrichtet. Danach wird in der ausgerichteten Position mittels den Bildern und der Funktion 'Ik' alle benötigten Winkel berechnet. Mit diesen wird die Vorgreifposition angefahren, hier wird beim Aufheben der Greifer geöffnet. Nun wird die eigentliche Greifposition angefahren. In der Greifposition wird beim Ablegen der Greifer geöffnet, sodass das Objekt in der Ladestelle bleibt. Beim Aufheben wird in dieser Position der Greifer geschlossen, um das Objekt aus der Ladefläche herauszunehmen. Anschließend wird erst wieder die Vorgreifposition und dann eine Hochposition angefahren. Somit wurde entweder das Objekt, welches sich im Greifer befand, auf das Ablagesymbol (oder zwischen die Ablagesymbole) gelegt oder das Objekt welches sich im Sichtbereich befand, befindet sich nun im Greifer und kann in eine Ladestelle gelegt werden.

7.4. Arm Bewegung

'GelenkPos(ROS,WinkelNeuD)'

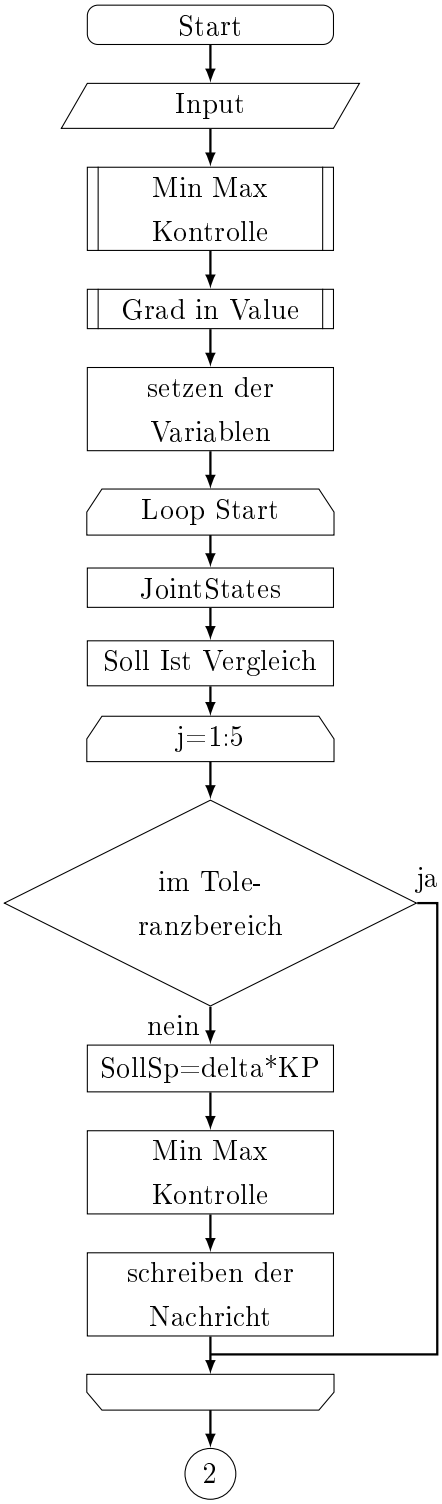

Abb. 7.10.: PAP 'GelenkPos' Teil 1

Die Funktion, die das Regeln der Arm Position bearbeitet, ist die 'GelenkPos'. Die Funktion hat zwei Eingaben, die ROS Informationen und die an zufahrenden neuen Winkelstellungen der Gelenke 1 bis 5. Die Eingabe der neuen Winkelpositionen sieht wie folgt aus: '[1 2 3 4 5]'. Die Werte sind in Grad anzugeben. Falls ein oder mehrere Gelenke nicht verfahren werden sollen, so muss an der jeweiligen Position 'nan' (nan = not a number) angegeben werden. Die Eingabe könnte wie folgt aussehen: '[90 0 nan 45 nan]'.
Die benötigten Informationen über die ROS-Umgebung sind:

- die Informationen des Arms: Minimale und Maximale Winkelwerte der Gelenke
- der aktuelle Status der Gelenke
- das Format der zu sendenden Nachricht
- der Empfänger der Nachricht

Die Funktion beginnt mit dem Kontrollieren der Eingabe. Falls eine Winkeleingabe größer oder kleiner ist als der Grenzwert, wird dieser Wert auf den jeweiligen Grenzwert gesetzt. Danach folgt das Umrechnen in die für den Arm benötigte Skalierung. Die übrigen Variablen die vor dem Start der Schleife gesetzt werden sind:

- Endgeschwindigkeit und Endtoleranz
- minimal und maximal Geschwindigkeit
- P-Regler Werte für die einzelnen Gelenke

In der Regler Schleife werden zunächst die aktuellen Positionen der Gelenke ausgelesen und mit der Eingabe vergleicht (Soll-Ist Vergleich). Danach wird für die Gelenke, die sich noch nicht innerhalb des Toleranzbereichs befinden, eine neue Geschwindigkeit berechnet. Die neue Geschwindigkeit wird erst Kontrolliert, ob diese sich zwischen der Maximal- und Minimalgeschwindigkeit befindet, und dann erst in die Nachricht geschrieben.

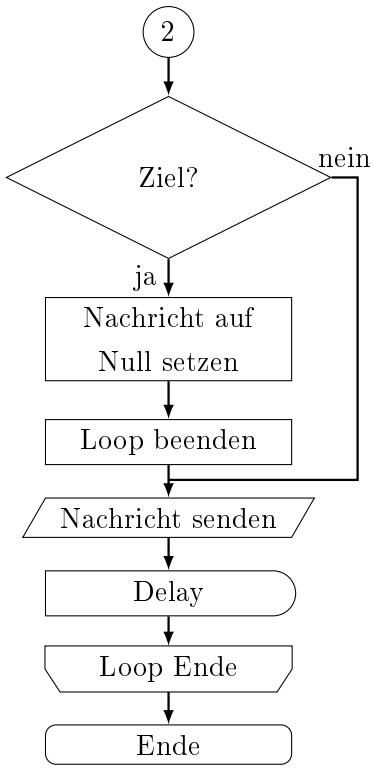

Abb. 7.11.: PAP 'GelenkPos' Teil 2

Nachdem berechnen der Geschwindigkeit und schreiben der Nachricht, wird kontrolliert ob sich die Gelenke schon im Zielbereich befinden und langsamer sind als die Endgeschwindigkeit. Falls dies zutrifft wird die Geschwindigkeit für alle Gelenke in der Nachricht auf null gesetzt und der Loop, nach dem nächsten Durchlauf, beendet. Danach wird die Nachricht gesendet, entweder die berechnete oder die auf null gesetzte. Darauf folgt eine kurze Pause, hier 0.01 Sekunden, um den Arm Zeit zugeben auf die Nachricht zu reagieren.

Die Regelung des Arms beziehungsweise der einzelnen Gelenke geschieht nur mittels eines P-Reglers. Da die Regelstrecke ein I-Anteil, besitzt ist die Regelung so möglich.

'GreiferPos(ROS,Abstand)'
oder
'GreiferPos(ROS,Greifer1,Greifer2)'

Die Funktion zum Ansteuern des Greifers benötigt zwei beziehungsweise drei Eingaben. Die benötigten Informationen über die ROS-Umgebung sind:

- die Informationen des Greifers: Minimal und Maximal Werte
- der aktuelle Status des Greifers
- das Format der zu sendenden Nachricht
- der Empfänger der Nachricht

Die übrigen Eingaben werden in Millimeter gemacht. Eine weitere Eingabe gibt den Abstand der beiden Greifer zueinander an. Falls zwei weitere Eingaben gemacht werden, geben diese den Abstand des jeweiligen Greifers zur Mitte an. Als Beispiel wird eine Greiferseite auf 10mm und die andere Greiferseite auf 30mm gefahren:
'GreiferPos(ROS,10,30)'. Das Programm wartet so lange bis der Greifer die eingegebene Position erreicht hat.

Die Flächen mit denen der Greifer die Objekte umfasst sind zu glatt um die Zylinder richtig zu packen. Deswegen wurden diese mit einem Gummischlauch überzogen. Zusätzlich wurden zwei Kabel, je Greifer Seite, so angebracht das die Zylinder sich im gegriffenen Zustand nicht verdrehen können.

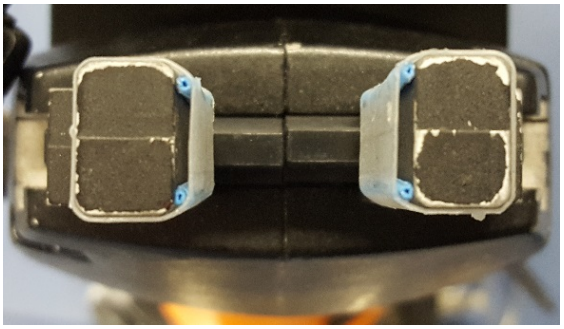

Abb. 7.12.: Greifer Ansicht 1

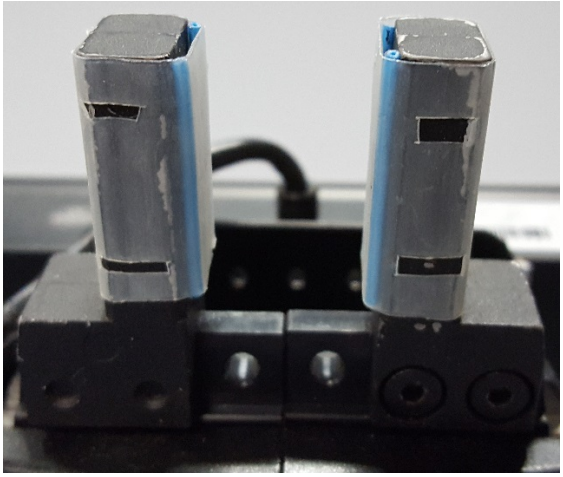

Abb. 7.13.: Greifer Ansicht 2

7.5. Kreiserkennung

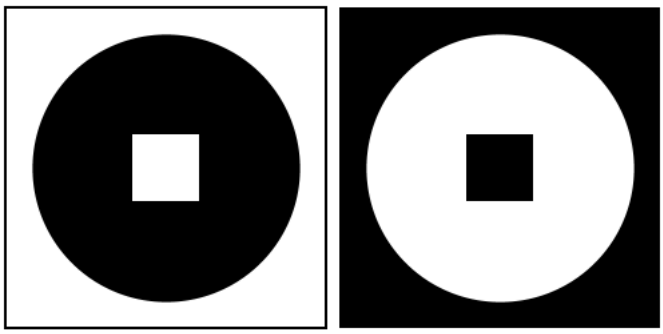

Abb. 7.14.: Symbole der Kreiserkennung

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)'

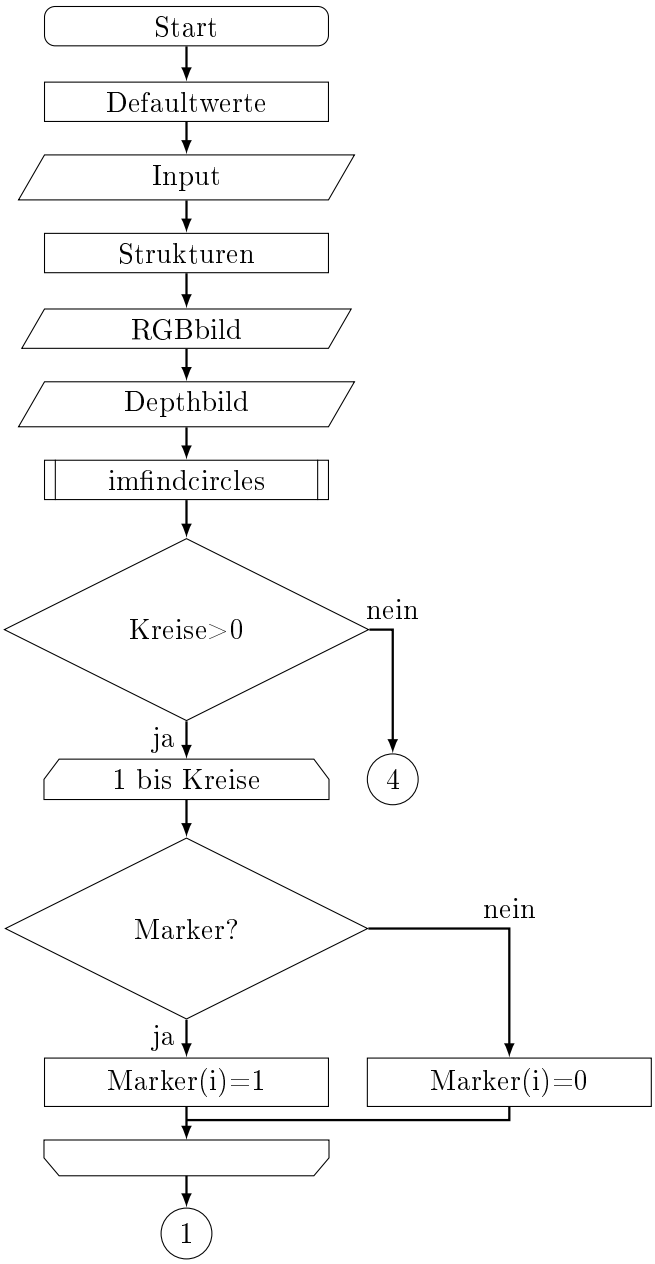

Abb. 7.15.: PAP 'KreisErkennung' Teil 1

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, siehe Abbildung 7.20.
Falls Kreise gefunden wurden, wird bei jedem Kreis überprüft ob dieser eine Markierung besitzt und dementsprechend abgespeichert.

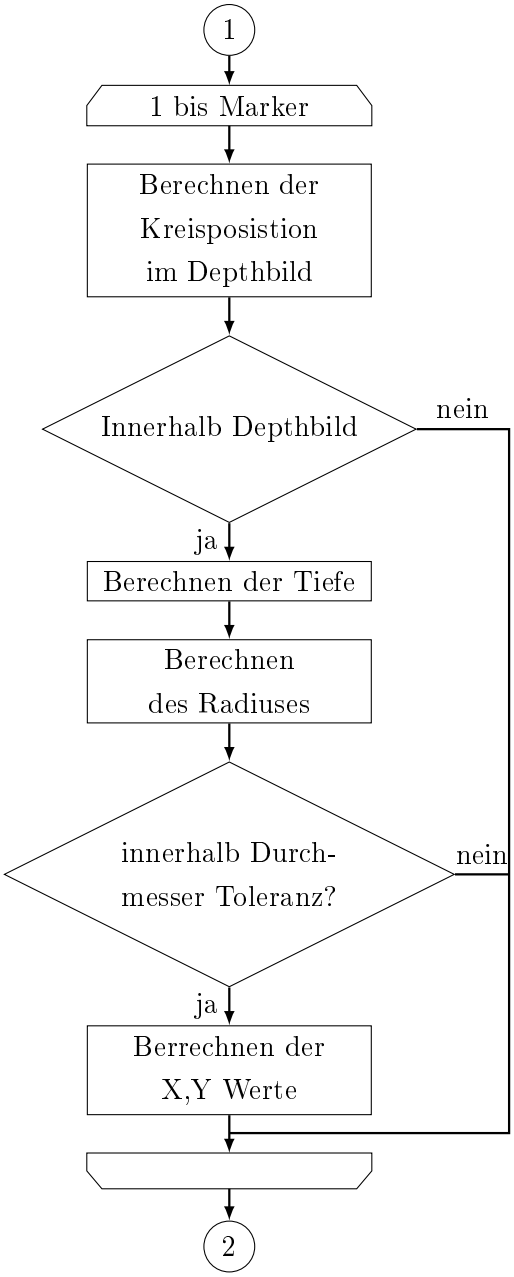

Abb. 7.16.: PAP 'KreisErkennung' Teil 2

Für jeden erkannten Kreis, der auch eine Markierung besitzt, wird die Position berechnet die der Kreis im Tiefenbild hat, siehe Formeln 7.1 und 7.2. Die Verschiebung kommt zustande, da die RGB Kamera zum IR Sensor versetzt ist, siehe Abbildung 7.18.

$$DX = KX + (KX - 130) * 0.1 \tag{7.1}$$

$$DY = KY + (KY - 60) * 0.1 \tag{7.2}$$

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 7.3 bis 7.5.

$$KZ = imgDepth(DY, DX) \tag{7.3}$$

$$Z0 = 1000 * KZ \tag{7.4}$$

$$RZ = 564 - Z0 \tag{7.5}$$

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 7.6 bis 7.9.

$$FZ = \frac{Z0}{530} \tag{7.6}$$

$$R0 = KR * FZ \tag{7.7}$$

$$RR = R0 \tag{7.8}$$

$$RD = R0 * 2 \tag{7.9}$$

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 7.10 bis 7.13.

$$Y0 = (240 - KY) * FZ \tag{7.10}$$

$$X0 = (320 - KX) * FZ \tag{7.11}$$

$$RY = 95 + Y0 \tag{7.12}$$

$$RX = -12 - X0 \tag{7.13}$$

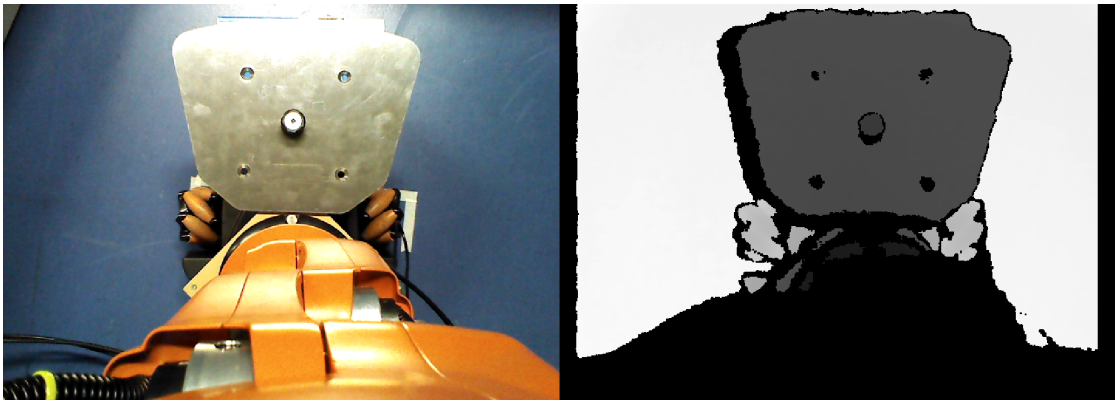

Abb. 7.17.: Objekt im RGB- und Depthbild

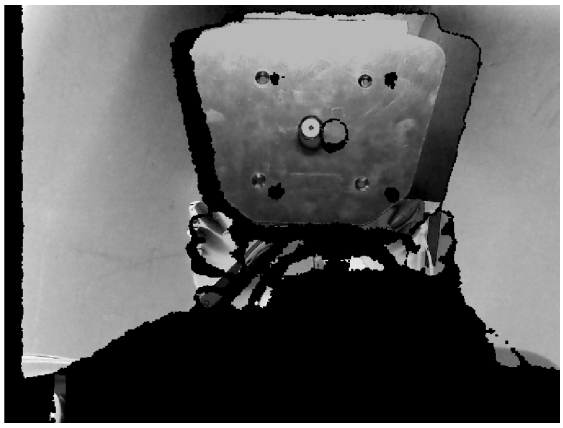

Abb. 7.18.: Unterschied RGB- zu Depthbild

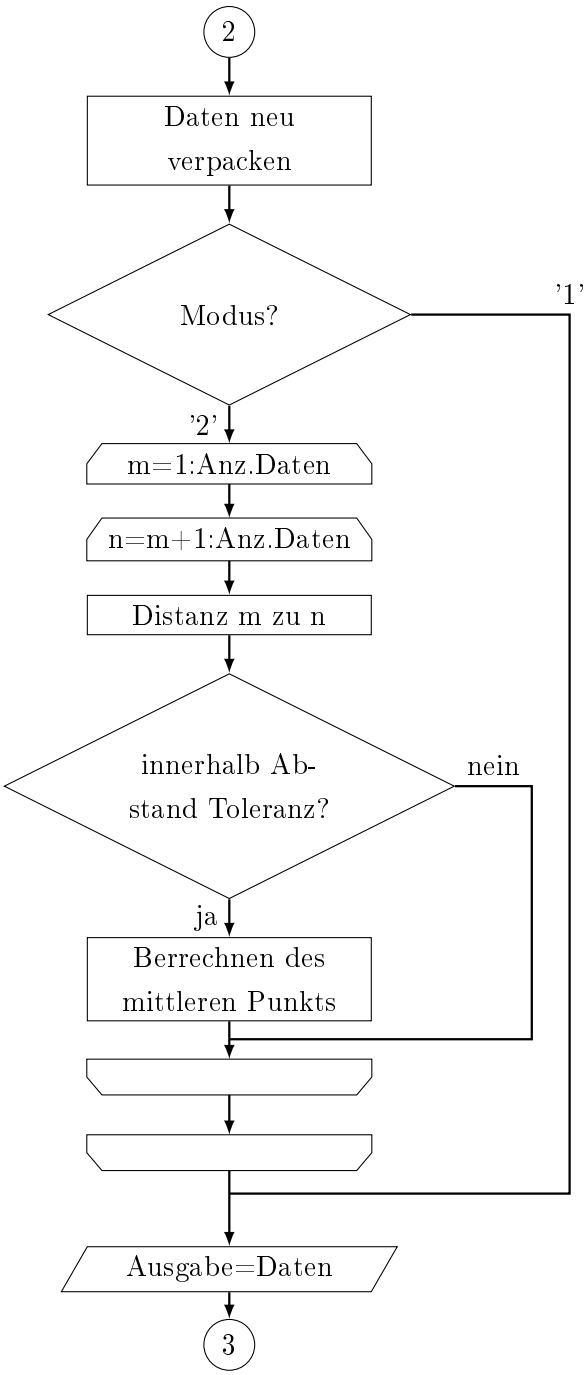

Abb. 7.19.: PAP 'KreisErkennung' Teil 3

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 7.14.

$$Dis = \sqrt{((X(m) - X(n))^2 + (Y(m) - Y(n)^2))} \tag{7.14}$$

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 7.15 bis 7.19. Die Tiefe für diesen Punkt wird nochmal neu mit dem Tiefenbild errechnet, siehe Formeln 7.20 und 7.21.

$$RX = (RX(a) + RX(b))/2 \tag{7.15}$$

$$RY = (RY(a) + RY(b))/2 \tag{7.16}$$

$$RD = (RD(a) + RD(b))/2 \tag{7.17}$$

$$DX = (DX(a) + DX(b))/2 \tag{7.18}$$

$$DY = (DY(a) + DY(b))/2 \tag{7.19}$$

$$DZ = imgDepth(DY, DX) \tag{7.20}$$

$$RZ = 564 - 1000 * (DZ) \tag{7.21}$$

Danach werden die berechneten Daten ausgegeben.

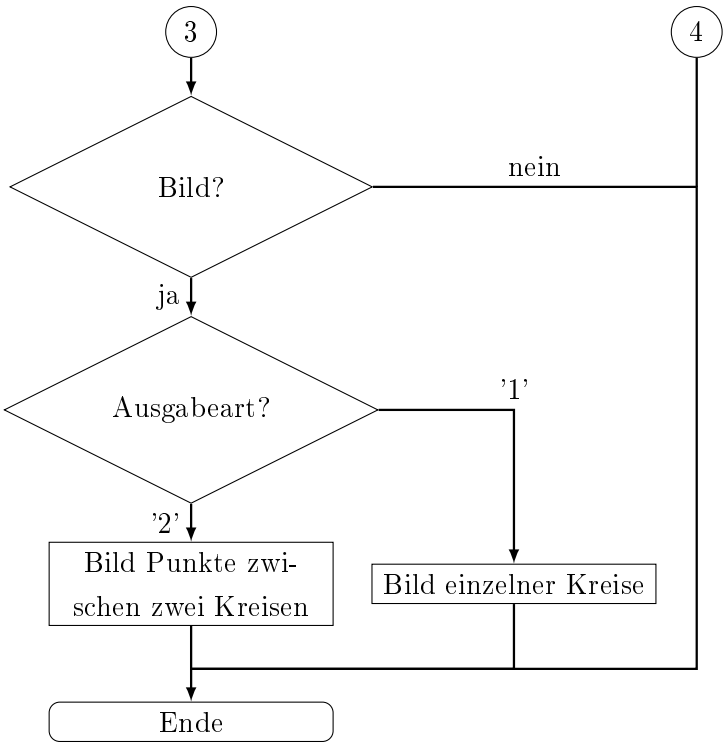

Abb. 7.20.: PAP 'KreisErkennung' Teil 4

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 7.22. Die Anzeige der Abstände zeigt nur die Mittelpunkte von Kreispaaren an, die den richtigen Durchmesser und Abstand zueinander haben, siehe Abbildung 7.21.

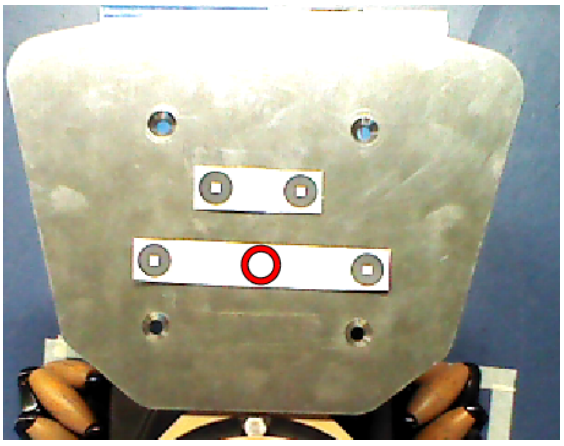

Abb. 7.21.: Anzeige des Mittelpunktes

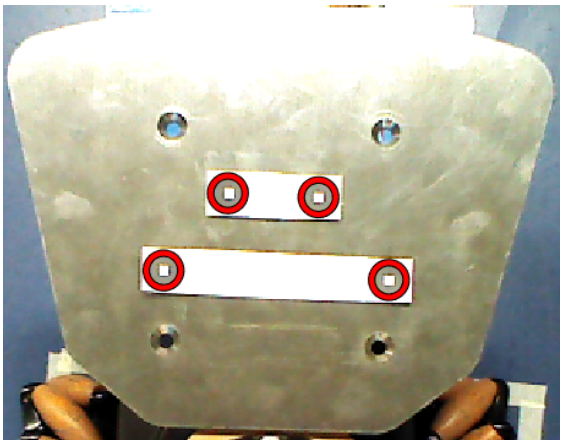

Abb. 7.22.: Anzeige einzelner Kreise

7.6. Inverse Kinematik

Um die erforderlichen Daten auf Y und Z zu reduzieren, wird vor der geometrisch inversen Kinematik der Arm zum Objekt ausgerichtet und damit der X-Wert nahe Null gebracht.

7.6.1. Vorprogramm

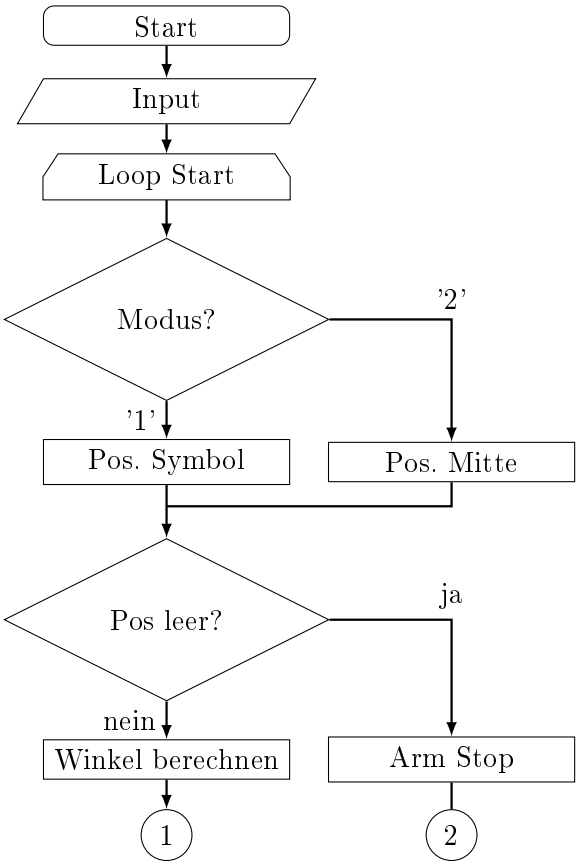

Abb. 7.23.: PAP 'VorIk' Teil 1

Das Programm 'vorIk', startet mit der Übergabe der ROS-Strukturdaten.

In der danach startende Schleife wird zunächst die Position des Ziels ermittelt. Der Modus entscheidet ob ein einzelnes Symbol oder der Punkt zwischen zwei Symbolen angefahren werden soll.

Darauf folgt eine Kontrolle der Daten. Falls kein Ziel sichtbar ist bleibt der Arm stehen und die Schleife startet von vorne. Falls mehrfach keine Daten vorhanden sind bricht das Programm ab. Mit vorhandenen Daten wird der an zufahrende Winkel berechnet, siehe Formeln 7.22 und 7.23.

$$Punkt = [Pos.XPos.YPos.Z] \tag{7.22}$$

$$alpha = atand(Punkt(1)/Punkt(2)) \tag{7.23}$$

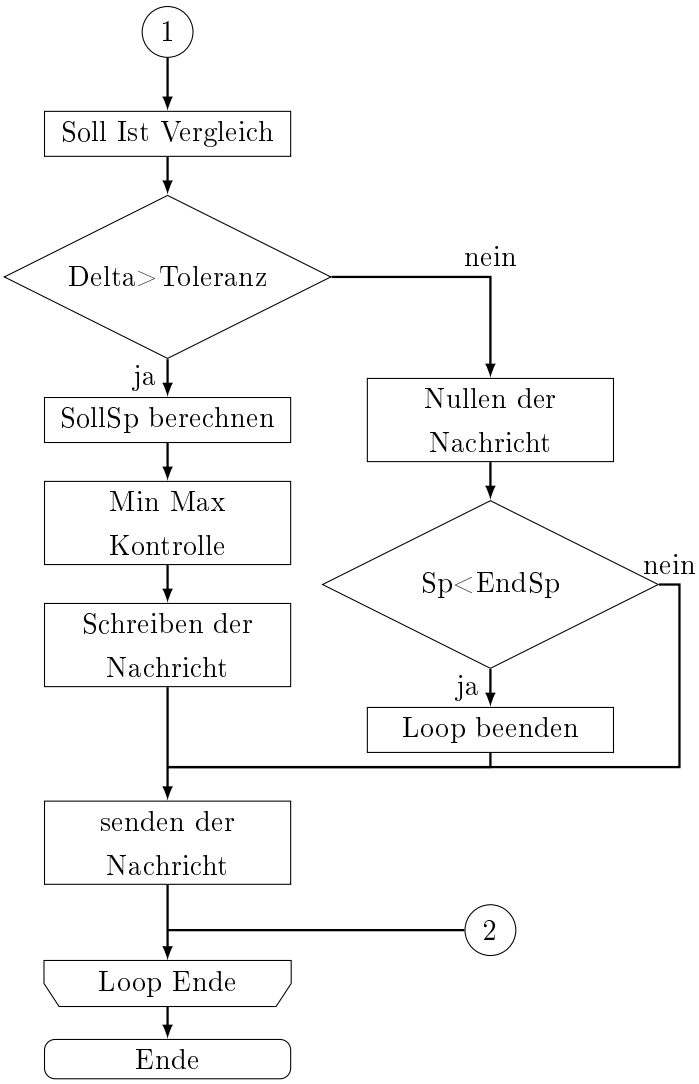

Abb. 7.24.: PAP 'VorIk' Teil 2

Mit der errechneten aktuellen Winkelstellung und dem an zufahrenden Winkel, wird ein Soll Ist Vergleich gemacht.

In dem Fall das der resultierende Deltawert innerhalb der Toleranz liegt, wird die Nachricht für das Gelenk auf null gesetzt. Zusätzlich wird kontrolliert ob die aktuelle Geschwindigkeit kleiner ist als die eingestellte Endgeschwindigkeit. In diesen Fall wird die Schleife nachdem nächsten Durchlauf beendet. Wenn der Deltawert zu groß ist, wird eine neue Armgeschwindigkeit berechnet, siehe Formeln 7.24 bis 7.28. Diese wird auf Minimal- und Maximalgrenzen kontrolliert und in die Gelenknachricht geschrieben.

$$KP = 0.03; \tag{7.24}$$

$$Soll = 0; \tag{7.25}$$

$$Ist = alpha; \tag{7.26}$$

$$deltaPos = Soll - Ist; \tag{7.27}$$

$$SollSp = deltaPos * KP; \tag{7.28}$$

Danach wird die Nachricht gesendet und die Schleife solange wiederholt, bis die Toleranz und Endgeschwindigkeit erreicht wurde.

7.6.2. Geometrisch inverse Kinematik

Die hier gewählte inverse Kinematik ist eine vereinfachte Version einer geometrischen Kinematik. Anders als bei einer vollständig inversen Kinematik, werden bei dieser Variante nur die Winkelpositionen im 1. Quadranten für 2 Gelenke berechnet. Diese entsprechen der Gelenke 2 und 3 beim youBot. Gelenk 4 resultiert aus den errechneten Werten, gehört aber Sinngemäß nicht zu der inversen Kinematik.

$$l_1 = 155 \tag{7.29}$$

$$l_2 = 135 \tag{7.30}$$

$$E_y = Y - 33 \tag{7.31}$$

$$E_z = Z + 217.5 - 147 + 10 \tag{7.32}$$

$$l = \sqrt{E_y^2 + E_z^2} \tag{7.33}$$

$$alpha = \left(atan\left(\frac{E_y}{E_z}\right) - acos\left(\frac{l_1^2 + l^2 - l_2^2}{2 * l_1 * l}\right)\right) * \frac{180}{\pi} \tag{7.34}$$

$$beta = 180 - acos\left(\frac{l_2^2 + l_1^2 - l^2}{2 * l_2 * l_1}\right) * \frac{180}{\pi} \tag{7.35}$$

$$gamma = 180 - (alpha + beta) \tag{7.36}$$

Die Funktion '[alpha, beta, gamma]=Ik(Y,Z);' benötigt zwei Eingaben. Diese geben die Position des Objekts an. X sollte zu dieser Zeit nahe Null sein.
Die Längen des Arms sind fest ein programmiert, siehe Formeln 7.29 und 7.30. Die eingegebenen Koordinaten werden dem Nullpunkt der Berechnung angepasst sowie die Entfernung vom Nullpunkt zum Endpunkt berechnet wird, siehe Formeln 7.31 bis 7.33. Mittels einen umgestellten Kosinussatzes werden die ersten beiden Winkel berechnet, siehe Formeln 7.34 und 7.35. Der letzte Winkel wird durch eine einfache Subtraktion errechnet, siehe Formel 7.36.
Die Rückgabewerte der Funktion beschreiben die anzufahrenden Winkel für die Gelenke 2, 3 und 4.

8. Schlussbetrachtung

Zusammenfassend kann gesagt werden, dass die Aufgabe des selbstständigen Erkennens von Objekten, die Berechnung der Winkelstellungen sowie das Verfahren des Arms, Aufheben und Ablegen eines Objekts, mittels dem geschriebene Programm gelöst worden ist.

Das Erkennen wurde mit dem Programm 'KreisErkennung' umgesetzt. Dieses kann durch Einstellungen innerhalb der Eingabe auf verschiedene Symbolgrößen oder Zustände konfiguriert werden. Es berechnet von den gesuchten Symbolen die X, Y und Z Koordinaten sowie den Durchmesser und falls gefragt auch den Abstand von gleich großen Kreisen zueinander.
Die Berechnung der Winkelstellungen erledigt eine vereinfachte geometrisch Inverse Kinematik (Programm 'Ik'). Dieser Funktion wird die ermittelte Position des Objekts übergeben und gibt als Rückgabewert die Winkelstellungen für drei Gelenke aus. Dadurch das es eine vereinfachte angepasste Variante einer Inversen Kinematik ist, funktioniert diese nur für zwei Gelenke in einer Ebene. Das verbliebene dritte Gelenk wird so angesteuert das der Greifer nach unten zeigt.
Die Programme 'GelenkPos' und 'GreiferPos' bewerkstelligen das Verfahren des Arms sowie das Öffnen und Schließen des Greifers. Die von der Inversen Kinematik berechneten Werte werden von dem Programm 'GelenkPos' benötigt.

Das Ergebnis stellt eine Modellm öglichkeit zur Steuerung von Robotern in Bestückungslinien oder Transportsystemen dar. Es müssen Anpassungen in den Bereichen Sicherheit und Fehlerquellen gemacht werden, damit diese Arbeit in der Industrie Anwendung finden kann. So muss zum Beispiel bei der Inversen Kinematik das Berechnen von Punkten, die innerhalb des Arms oder der Basis liegen, verhindert werden.

Projekte die nachfolgend am KUKA youBot entstehen, werden in der Vorbereitung sich mit der geringen Tragfähig des Roboterarms auseinander setzten müssen. Da dieser nur für die Entwicklung von Software und nicht zum Verwenden in einer industriellen Weise entwickelt wurde, ist dies akzeptabel. Zusätzlich bietet das ROS in Verbindung mit MATLAB durch die vielen Werkzeuge und Bibliotheken eine große Anzahl von möglichen Projekten in den Bereichen: Steuerung, Regelung, Sensorik, Kinematik (Invers oder Vorwärts), Prozessanalyse oder ähnlichen.

Bei weiterem Interesse, empfehlen sich folgende Quellen:

- ROS: ROSbyExample [Goe15] oder ROS Wiki [ROS]
- Youbottreiber: GitHub Nowak [Now]
- Kameratreiber: Develop Orbbec [Orba]
- Matlab: Matalabbuch [Sta09] oder MathWorks [Mata]
- Inverse Kinematik: Mathias Brandstötter [Bra]

Literaturverzeichnis

[Bra] BRANDSTÖTTER, Mathias: *Inverse kinematics of serial robots with wrist.* https://de.mathworks.com/matlabcentral/fileexchange/48468-inverse-kinematics-of-serial-robots-with-wrist?s%5C_tid=prof%5C_contriblnk. – zu letzt aufgerufen am 20.10.2017

[Fab] FABIANSACCILOTTO: *ROS.* http://wiki.ros.org/de. – zu letzt aufgerufen am 20.10.2017

[Flo] FLOREK, Jasinska: *Controlling youBot from external PC.* http://www.youbot-store.com/wiki/index.php/Controlling_youBot_from_external_PC. – zu letzt aufgerufen am 20.10.2017

[Goe15] GOEBEL, R. P.: *ROS by Example.* R. Patrick Goebel, 2015

[KUKa] KUKA: *KUKA Geschichte.* https://www.kuka.com/de-de/über-kuka/geschichte. – zu letzt aufgerufen am 20.10.2017

[KUKb] KUKA: *YouBot Datenblatt.* http://www-home.htwg-konstanz.de/~bittel/LaborMobileRoboter/youBot_datenblatt_web_0514.pdf. – zu letzt aufgerufen am 20.10.2017

[Mata] MATHWORKS: *MathWorks.* https://de.mathworks.com/?s_tid=gn_logo. – zu letzt aufgerufen am 20.10.2017

[MATb] MATLAB: *MATLAB About.* https://de.mathworks.com/company/aboutus.html. – zu letzt aufgerufen am 20.10.2017

[Now] NOWAK, W.: *youBot Treiber.* https://github.com/wnowak/youbot_driver_ros_interface. – zu letzt aufgerufen am 20.10.2017

[Orba] ORBBEC: *Develop with Orbbec.* https://orbbec3d.com/develop/. – zu letzt aufgerufen am 20.10.2017

[Orbb] ORBBEC3D: *Astra technical specs.* https://orbbec3d.com/product-astra/. – zu letzt aufgerufen am 20.10.2017

[ROS] ROS, Wiki: *Documentation*. http://wiki.ros.org. – zu letzt aufgerufen am 20.10.2017

[Sta09] Stark, Georg: *Robotik mit MATLAB*. Fachbuchverlag Leipzig, 2009

[Tul] TullyFoote: *ROS Kinetic Kame*. http://wiki.ros.org/kinetic. – zu letzt aufgerufen am 20.10.2017

Anhang

A. Ablauf

```
% Hauptprogramm Main, startet einen Loop in dem der YouBot vom Ablegen_Ort
% zum Aufheben_Ort verfaehrt und immer die jeweilige Aktion ausfuehrt
ROS=runROS;    % Startet eine Struktur in dem alle relevanten ROS-Infos sind
fenster=figure; % Startet eine Figur die Ausgabe, und Schliesser des Programms
    ist
Ladung=LadeCheck(ROS);    % Kontrolle der Ladeflaeche
Aktion='Fahrt';           % Aktion setzten auf Fahren
if sum(Ladung)>0
    Ortbefehl='Ablegen';  % Wenn min 1 dann zuerst Ablage anfahren
else
    Ortbefehl='Aufheben'; % Sonst erst Aufnahme anfahren
end
Ort='unbekannt';          % Der momentane Ort ist unbekannt
run=true; % Starten des Loops
while run
    textStep(Aktion,Ort,Ortbefehl,Ladung); % Ausgabe in der Figur
    switch Aktion     % Entscheiden was gemacht wird Anhand der Aktion
        case 'Fahrt'   % Wenn Fahrt dann erst Arm in Res_Stellung
            GelenkPos(ROS,[-169 -65 146 -102 -167]);
            switch Ortbefehl % Entscheidung wohin gefahren wird
                case 'Ablegen'
                    %Arena_Pfad_fcn(ROS,ROS.Basis.Info.Rechts); % Anfahren des
                        Orts
                    Ort='Ablegen';        % setzten des Aktuellen Orts
                    Aktion='Arm';         % Aendern der Aktion von Fahrt zu Arm
                    Ortbefehl='erreicht'; % Aendern auf erreicht
                case 'Aufheben'
                    %Arena_Pfad_fcn(ROS,ROS.Basis.Info.Links);
                    Ort='Aufheben';
                    Aktion='Arm';
                    Ortbefehl='erreicht';
```

```
30            end
31        case 'Arm'     % Wenn am Ziel Ort Arm ansteuern
32            switch Ort % Entscheiden der Arm Bewegung
33                case 'Ablegen'
34                    nr=0;              % Ladeposition
35                    for i=1:3        % durch zaehlen von 1:3
36                        if Ladung(i)==1 % falls an der Ladepos. ein Objekt ist
37                            nr=i;      % nr=Ladeposition
38                        end
39                    end
40                    if nr~=0   % falls alle Ladepositionen leer sind
41                        [Ladung(nr), LeererSichtbereich]=Ablegen(ROS,nr);
42                    end
43                    % falls Ladung leer oder Sichtbereich leer und Ladund nicht
                          voll ist
44                    if sum(Ladung)==0 || (LeererSichtbereich && sum(Ladung)<3)
45                        Ortbefehl='Aufheben'; % zum Anderen Ort fahren
46                        Aktion='Fahrt';       % Fahren aktivieren
47                    end
48                case 'Aufheben'
49                    nr=0;
50                    for i=1:3
51                        if Ladung(i)==0
52                            nr=i;
53                        end
54                    end
55                    if nr~=0
56                        [Ladung(nr), LeererSichtbereich]=Aufheben(ROS,nr);
57                    end
58                    if sum(Ladung)==3 || LeererSichtbereich
59                        Ortbefehl='Ablegen';
60                        Aktion='Fahrt';
61                    end
62            end
63    end
64    if ~ishandle(fenster) % falls Fenster geschlossen wurde Loop beenden
65        run=false;
66    end
67 end
68 Aktion='Fahrt';       % Fahren aktivieren
69 Ortbefehl='Start_Ende'; % zur Start_Ende Position fahren
```

```
pause(1);
textStep(Aktion,Ort,Ortbefehl,Ladung);      % Ausgabe in der Figur
GelenkPos(ROS,[-169 -65 146 -102 -167]);    % Arm in Res_Stellung
Arena_Pfad_fcn(ROS,ROS.Basis.Info.Start_Ende); % Anfahren des Orts
```

B. getIP

```
% getIp zum Erhalten der aktuellen Ip
function ip=getIP()
% Mittels Systembefehl wird Verbindung ausgelesen
[~,result]=system('ifconfig | grep "inet Adresse"');
% Der ausgelesende String wird getrennt und gespeichert
str=strsplit(result,{' ',':'});
try
    ip.Net=char(str(9));
catch
    ip.Net='0.0.0.0';
end
ip.Local=char(str(4));
end
```

C. conROS

```
% conRos startet die ROS Verbindung
function conROS(nodeName)
ip=getIP();
if (exist('nodeName','var'))
    % eingabe = nodeName
else
    nodeName='Matlab';
end
if robotics.ros.internal.Global.isNodeActive
    % schon aktiv nichts tun
else
    % Verbindung zum Masterserver
    try
        rosinit(ip.Local,'NodeName',nodeName);
    catch
```

```
        rosinit(ip.Net,'NodeName',nodeName);
    end
end
end
```

D. runROS

```
% zum testen des Arms ohne Hauptprogramm,
% erstellen der Verbindung und der ROS-Struktur
function ROS=runROS()
conROS();
pause(1); % kurzer Delay
% Rosverbindungen
DepthSubscriber=rossubscriber('camera/depth/image','sensor_msgs/Image');
RGBSubscriber=rossubscriber('camera/rgb/image_raw','sensor_msgs/Image');
ArmPublisher = rospublisher ('arm_1/arm_controller/velocity_command','
    brics_actuator/JointVelocities');
ArmPublisher2 = rospublisher ('arm_1/arm_controller/position_command','
    brics_actuator/JointPositions');
ArmSubscriber = rossubscriber ('/joint_states','sensor_msgs/JointState');
GreiferPublisher = rospublisher ('arm_1/gripper_controller/position_command','
    brics_actuator/JointPositions');
% Arm Geschwindigkeits Struktur
ArmNachricht = rosmessage(ArmPublisher); % Nachricht konfigurieren
for i=1:5
    Gelenk(i) = rosmessage('brics_actuator/JointValue'); % Gelenk konfigurieren
    Gelenk(i).JointUri = ['arm_joint_' num2str(i)];    % Gelenk Nr
    Gelenk(i).Value = 0.0;                             % Value
    Gelenk(i).Unit = 's^-1 rad';                       % Einheit
end
ArmNachricht.Velocities = Gelenk; % Nachricht verpacken
% Arm Posistions Struktur
ArmNachricht2 = rosmessage(ArmPublisher2); % Nachricht konfigurieren
for i=1:5
    Gelenk(i) = rosmessage('brics_actuator/JointValue'); % Gelenk konfigurieren
    Gelenk(i).JointUri = ['arm_joint_' num2str(i)];    % Gelenk Nr
    Gelenk(i).Value = 0.0;                             % Value
    Gelenk(i).Unit = 'rad';                            % Einheit
end
```

```
ArmNachricht2.Positions = Gelenk; % Nachricht verpacken
% Greifer Positions Struktur
GreiferNachricht = rosmessage(GreiferPublisher); % Nachricht konfigurieren
for j=1:2
    Greifer(j) = rosmessage('brics_actuator/JointValue'); % Greifer
        konfigurieren
    if j==1
        Greifer(j).JointUri = 'gripper_finger_joint_l'; % Greifer Links
    elseif j==2
        Greifer(j).JointUri = 'gripper_finger_joint_r'; % Greifer Rechts
    end
    Greifer(j).Value = 0; % Value auf aktuell setzten
    Greifer(j).Unit = 'm'; % Einheit
end
GreiferNachricht.Positions = Greifer; % Nachricht verpacken
%Verpacken der Daten in eine leichter zu handhabende Struktur
ROS.Kamera.SubDepth=DepthSubscriber;
ROS.Kamera.SubRGB=RGBSubscriber;
ROS.Arm.Sub=ArmSubscriber;
ROS.Arm.Pub=ArmPublisher;
ROS.Arm.Nach=ArmNachricht;
ROS.Arm.Pub2=ArmPublisher2;
ROS.Arm.Nach2=ArmNachricht2;
ROS.Greifer.Pub=GreiferPublisher;
ROS.Greifer.Nach=GreiferNachricht;
%Arm Informationen
ROS.Arm.Info=infoArm();
%Ladestellen Informationen
ROS.Lade.Info=infoLade();
end
```

E. infoArm

```
% Min Max sowie Up/Mid Positionen aller Gelenke und des Greifers
function Out=infoArm()
% Gelenke Min Max Winkel
JointMin=[-169 -65 -151 -102 -167 0 0];
JointMax=[169 90 146 102 167 10 10];
% Spezielle Positionen Winkel
```

```
JointRes=[-169 -65 146 -102 -167 0 0];
JointUp=[0 0 0 0 0 0 0];
% Gelenke Min Max Werte Treiber
JointValueMin=round([0.0100692 0.0100692 -5.02655 0.0221239 0.110619 0
    0]+0.00005,4);
JointValueMax=round([5.84014 2.61799 -0.015708 3.4292 5.64159 0.0115
    0.0115]-0.00005,4);
JointValueUp=zeros(1,7);
JointValueRes=zeros(1,7);
% Berechnen der Speziellen Positionen Werte Treiber
for i=1:7
    JointValueUp(i)=interpolieren(JointUp(i),JointMin(i),JointMax(i),
        JointValueMin(i),JointValueMax(i));
    JointValueRes(i)=interpolieren(JointRes(i),JointMin(i),JointMax(i),
        JointValueMin(i),JointValueMax(i));
end
% Verpacken in Struktur
Out=struct(...
    'JointMin',JointMin,...
    'JointMax',JointMax,...
    'JointUp',JointUp,...
    'JointRes',JointRes,...
    'JointValueMin',JointValueMin,...
    'JointValueMax',JointValueMax,...
    'JointValueRes',JointValueRes,....
    'JointValueUp',JointValueUp);
end
```

F. infoLade

```
% Informationen zu den fixen Ladestellen
function Out=infoLade()
WinkelDiff=15;
WinkelLadeD=zeros(3,5);
% Berechnen der Positionen
for i=1:3
    % WinkelLadeD(i,:)=[1.25+WinkelDiff*(i-2) -45.5 -45 -90.25 5];
    WinkelLadeD(i,:)=[1.25+WinkelDiff*(i-2)-45.5 -45.5 -45 -90.25 5];
end
```

```
% Verpacken in Struktur
Out=struct(...
    'LadePositionen',WinkelLadeD);
end
```

G. LadeCheck

```
% Zur Kontrolle der Ladestellen : -1 nicht Sichtbar, 1 Voll, 0 Leer
function Ladung=LadeCheck(ROS)
notFoundCnt=0;
GelenkPos(ROS,[136.5 3 0 0 5]);
% GelenkPos(ROS,[-167.5 3 0 0 -10]);
Check=true;
Ladung=[-1 -1 -1];
LadeStellenX=[-68 -10 47];
LadeStellenY=[217.5 223 217.5];
LadeStellenToleranz=20;
while Check
    Leer=KreisErkennung(ROS,'s','1',20);
    Voll=KreisErkennung(ROS,'w','1',20);
    LadestellenCnt=0;
    if size(Leer,2)==1
        if ~isempty(Leer.X)
            for i=1:3
                if Leer.X<LadeStellenX(i)+LadeStellenToleranz ...
                        && Leer.X>LadeStellenX(i)-LadeStellenToleranz ...
                        && Leer.Y<LadeStellenY(i)+LadeStellenToleranz ...
                        && Leer.Y>LadeStellenY(i)-LadeStellenToleranz
                    Ladung(i)=0;
                    LadestellenCnt=LadestellenCnt+1;
                end
            end
        end
    else
        for j=1:size(Leer,2)
            for k=1:3
                if Leer(j).X<LadeStellenX(k)+LadeStellenToleranz ...
                        && Leer(j).X>LadeStellenX(k)-LadeStellenToleranz ...
                        && Leer(j).Y<LadeStellenY(k)+LadeStellenToleranz ...
```

```
                    && Leer(j).Y>LadeStellenY(k)-LadeStellenToleranz
                Ladung(k)=0;
                LadestellenCnt=LadestellenCnt+1;
            end
        end
    end
end
if size(Voll,2)==1
    if ~isempty(Voll.X)
        for i=1:3
            if Voll.X<LadeStellenX(i)+LadeStellenToleranz ...
                    && Voll.X>LadeStellenX(i)-LadeStellenToleranz ...
                    && Voll.Y<LadeStellenY(i)+LadeStellenToleranz ...
                    && Voll.Y>LadeStellenY(i)-LadeStellenToleranz
                Ladung(i)=1;
                LadestellenCnt=LadestellenCnt+1;
            end
        end
    end
else
    for j=1:size(Voll,2)
        for k=1:3
            if Voll(j).X<LadeStellenX(k)+LadeStellenToleranz ...
                    && Voll(j).X>LadeStellenX(k)-LadeStellenToleranz ...
                    && Voll(j).Y<LadeStellenY(k)+LadeStellenToleranz ...
                    && Voll(j).Y>LadeStellenY(k)-LadeStellenToleranz
                Ladung(k)=1;
                LadestellenCnt=LadestellenCnt+1;
            end
        end
    end
end
if LadestellenCnt==3
    Check=false;
else
    notFoundCnt=notFoundCnt+1;
    pause(0.1);
    if notFoundCnt>5
        Check=false;
        warning('Lade Stellen oder Ladung nicht sichtbar');
    end
```

```
    end
end
end
```

H. 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
% Modus
inds = find(strcmpi('1',varargin), 1);
if ~isempty(inds)
    Modus='einzel';
    Durchmesser=cell2mat(varargin(inds+1));
end
```

```
inds = find(strcmpi('2',varargin), 1);
if ~isempty(inds)
    Modus='doppel';
    Durchmesser=cell2mat(varargin(inds+1));
    Abstand=cell2mat(varargin(inds+2));
end
% Durchmessertoleranz
inds = find(strcmpi('Dtol',varargin), 1);
if ~isempty(inds)
    DurchmesserToleranz=cell2mat(varargin(inds+1));
end
% Abstandtoleranz
inds = find(strcmpi('Atol',varargin), 1);
if ~isempty(inds)
    AbstandToleranz=cell2mat(varargin(inds+1));
end
% Sensibilitaet der Kreiserkennung
inds = find(strcmpi('Sens',varargin), 1);
if ~isempty(inds)
    Sensitivity=cell2mat(varargin(inds+1));
end
% Bildliche Ausgabe
inds = find(strcmpi('Bild',varargin), 1);
if ~isempty(inds)
    Bildlich=1;
end
%##################################################
% Struktur Bearbeitende Werte und Ausgabe
Data=struct('KameraX',[],'KameraY',[],'KameraZ',[],'KameraR',[],...
    'DepthX',[],'DepthY',[],...
    'RealX',[],'RealY',[],'RealZ',[],'RealR',[],'RealD',[],...
    'Marker',[],'MValue',[]);
tempData=struct('X',[],'Y',[],'Z',[],'D',[],...
    'KX',[],'KY',[],'KR',[],'DX',[],'DY',[],'Dis',[]);
tempData2=tempData;
Ausgabe=struct('X',[],'Y',[],'Z',[],'D',[],'Dis',[]);
%##################################################
% holen der Bilder, RGB und Depth, und umwandeln des RGB in Gray
imgDepth=readImage(ROS.Kamera.SubDepth.LatestMessage);
imgRGB=readImage(ROS.Kamera.SubRGB.LatestMessage);
imgGray=rgb2gray(imgRGB);
```

```
%##################################################
% Kreiserkennung und Markerkontrolle
Rmin=6; % Durchmesser Einstellung,
Rmax=18; % der zur suchenden [px] einheitlichen Kreise
% nur die Kreise suchen die gefragt sind, farblich
% aussortieren der Kreise ohne Marker
[centers, radii] = imfindcircles(imgGray,[Rmin Rmax],...
    'ObjectPolarity',ColorCode,'Sensitivity',...
    Sensitivity,'Method','twostage');
if ~isempty(centers)
    for i=1:size(centers,1)
        Data(i).KameraX=centers(i,1);
        Data(i).KameraY=centers(i,2);
        Data(i).KameraR=radii(i);
        Data(i).MValue=imgGray(round(Data(i).KameraY,0),round(Data(i).KameraX,0)
            );
        % Marker ist bei weiss schwarz und vice versa
        if strcmp(ColorCode,'bright')
            if 0<=Data(i).MValue && Data(i).MValue<200
                Data(i).Marker=1;
            else
                Data(i).Marker=0;
            end
        elseif strcmp(ColorCode,'dark')
            if 201<Data(i).MValue && Data(i).MValue<=255
                Data(i).Marker=1;
            else
                Data(i).Marker=0;
            end
        end
    end
    %##################################################
    % Berechnung des Durchmessers mittels der Tiefe Z
    % nur bei denen die den Marker haben
    DepthPosX=zeros(size(Data,2),1);
    DepthPosY=zeros(size(Data,2),1);
    for j=1:size(Data,2)
        if Data(j).Marker==1
            DepthPosX(j)=round(Data(j).KameraX+(Data(j).KameraX-130)*0.1,0);
            DepthPosY(j)=round(Data(j).KameraY+(Data(j).KameraY-60)*0.1,0);
            Data(j).DepthX=DepthPosX(j);
```

```
            Data(j).DepthY=DepthPosY(j);
            % Achtung Bildrand, schauen ob wir uns nahe des Rands befinden
            area=5;
            if DepthPosX(j)<1+area || DepthPosX(j)>640-area
                %disp('x<0 || x>640');
            elseif DepthPosY(j)<1+area || DepthPosY(j)>480-area
                %disp('y<0 || y>480');
            else
                % mehrere Z werte zusammen rechnen und teilen
                % fals der mittlerste Werte mal 'nan' ist
                anzahl=0;
                Zsum=0;
                for u=-area:2:area
                    for v=-area:2:area
                        if ~isnan(imgDepth(DepthPosY(j)+u,DepthPosX(j)+v))
                            Zsum=Zsum+imgDepth(DepthPosY(j)+u,DepthPosX(j)+v);
                            anzahl=anzahl+1;
                        end
                    end
                end
                Data(j).KameraZ=Zsum/anzahl;
                Z0=1000*Data(j).KameraZ;%mm
                FaktorZ=(Z0/530);
                R0=round((Data(j).KameraR*FaktorZ),2);
                Data(j).RealZ=564-Z0;
                Data(j).RealR=R0;
                Data(j).RealD=R0*2;
                %##################################################
                % Berechnen der X Y Position
                % nur bei denen die den Richtigen Durchmesser haben
                if Data(j).RealR<(Durchmesser/2)+(DurchmesserToleranz/2) &&...
                        Data(j).RealR>(Durchmesser/2)-(DurchmesserToleranz/2)
                    Y0=round((240-Data(j).KameraY)*FaktorZ,2);% mm 240 ==
                        Bildhoehe/2
                    X0=round((320-Data(j).KameraX)*FaktorZ,2);% mm 320 ==
                        Bildbreite/2
                    Data(j).RealY=95+Y0;  % Nullpunktverschub
                    Data(j).RealX=-12-X0; % Nullpunktverschub
                end
            end
        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;
```

```
                    tempData2(auscnt).D=(tempData(m).D+tempData(n).D)/2;
                    tempData2(auscnt).KX=(tempData(m).KX+tempData(n).KX)/2;
                    tempData2(auscnt).KY=(tempData(m).KY+tempData(n).KY)/2;
                    tempData2(auscnt).KR=(tempData(m).KR+tempData(n).KR)/2;
                    tempData2(auscnt).DX=round((tempData(m).DX+tempData(n).DX)
                        /2,0);
                    tempData2(auscnt).DY=round((tempData(m).DY+tempData(n).DY)
                        /2,0);
                    area=3;
                    anzahl=0;
                    Zsum=0;
                    for p=-area:2:area
                        for q=-area:2:area
                            if ~isnan(imgDepth(tempData2(auscnt).DY+p,tempData2(
                                auscnt).DX+q))
                                Zsum=Zsum+imgDepth(tempData2(auscnt).DY+p,
                                    tempData2(auscnt).DX+q);
                                anzahl=anzahl+1;
                            end
                        end
                    end
                    tempData2(auscnt).Z=564-round(1000*(Zsum/anzahl),2);%mm
                    auscnt=auscnt+1;
                end
                discnt=discnt+1;
            end
        end
        for o=1:size(tempData2,2)
            Ausgabe(o).X=tempData2(o).X;
            Ausgabe(o).Y=tempData2(o).Y;
            Ausgabe(o).Z=tempData2(o).Z;
            Ausgabe(o).D=tempData2(o).D;
            Ausgabe(o).Dis=tempData2(o).Dis;
        end
    end
    %#####################################################
    % Bildliche Ausgabe
    if Bildlich==1
        imshow(imgRGB);
        % Einzeichnen der Kreise, '1' die echten, '2' die Mittelpunkte
        if strcmp(Modus,'einzel')
```

```
            for g=1:size(tempData,2)
                viscircles([tempData(g).KX tempData(g).KY], ...
                    tempData(g).KR,'LineStyle','-','EdgeColor','r');
                txt=strcat('D:',num2str(Ausgabe(g).D));
                text(tempData(g).KX+10,tempData(g).KY,txt,'Color','black','
                    FontSize',14)
            end
        elseif strcmp(Modus,'doppel')
            for h=1:size(tempData2,2)
                viscircles([tempData2(h).KX tempData2(h).KY], ...
                    tempData2(h).KR,'LineStyle','-','EdgeColor','r');
                txt=strcat('D:',num2str(Ausgabe(h).D),' Dis:', num2str(Ausgabe(h)
                    .Dis));
                text(tempData2(h).KX+10,tempData2(h).KY,txt,'Color','black','
                    FontSize',14)
            end
        end
    end
end
end
```

I. VorIk

```
% Drehen sodass der Arm in einer Flucht zum Objekt steht
% zum vereingfachen der inversen Kinematik
function LeererSichtbereich=VorIk(ROS,modus)
run=true;
EndSp=0.05;
LeererSichtbereich=false;
% Min Max
MinSp=0.01;
MaxSp=1;
notfoundCnt=0;
while run
    switch modus
        case '1'
            %Pos=KreisErkennung(ROS,'w','1',20);
            Pos=PosMinDisSimpel(ROS,'1');
            %Pos=PosAufnehmen(ROS);
```

```
        case '2'
            %Pos=KreisErkennung(ROS,'s','2',20,150);
            Pos=PosMinDisSimpel(ROS,'2');
    end
    if isempty(Pos.X)
        ROS.Arm.Nach.Velocities(1).Value=0.0;
        send(ROS.Arm.Pub,ROS.Arm.Nach);
        notfoundCnt=notfoundCnt+1;
        pause(0.1);
        if notfoundCnt>10
            run=false;
            LeererSichtbereich=true;
        end
    else
        Punkt=[Pos.X Pos.Y Pos.Z];
        alpha=atand(Punkt(1)/Punkt(2)); % anzufahrender Winkel=IST 0=SOLL
        omega=0;
        % Regler
        KP=0.03;
        % Soll/Ist Vergleich
        Soll=omega;
        Ist=alpha;
        deltaPos=Soll-Ist;
        if abs(deltaPos)>0.05
            % P-Regler reicht hier aus weil die Regelstrecke I Glied besitzt
            SollSp=deltaPos*KP;
            % Speed Min und Max
            if abs(SollSp)<MinSp
                if SollSp<0
                    SollSp=-MinSp;
                else
                    SollSp=MinSp;
                end
            elseif abs(SollSp)>MaxSp
                if SollSp<0
                    SollSp=-MaxSp;
                else
                    SollSp=MaxSp;
                end
            end
            ROS.Arm.Nach.Velocities(1).Value=-SollSp;
```

```
        else
            ROS.Arm.Nach.Velocities(1).Value=0.0;
            if abs(ROS.Arm.Sub.LatestMessage.Velocity(1))<EndSp
                run=false;
            end
        end
        send(ROS.Arm.Pub,ROS.Arm.Nach);
    end
end
end
```

J. Ik

```
% Simple geometrische Inverse Kinematik fuer 2+1 Gelenke
function [alpha, beta, gamma]=Ik(Pos)
% Berechnung geht von G2 als Startpunkt aus deswegen die Verschiebungen
l1 = 155;
l2 = 135;
% 33 G1<->G2
Ey=Pos(1)-33+1;
% 217.5 G4<->Greifer, 147 0Ebene<->G1, 10 Sicherheits Abstand Boden
Ez=Pos(2)+217.5-147+10;
% Distanz von G2 zu G4
l=sqrt(Ey^2+Ez^2);
% Winkel von G2
alpha=(atan(Ey/Ez)-acos((l1^2+l^2-l2^2)/(2*l1*l)))*180/pi;
% Winkel von G3
beta=180-(acos((l2^2+l1^2-l^2)/(2*l2*l1))*180/pi);
% Winkel von G4 sodass senkrecht zum Boden
gamma=180-(alpha+beta);
end
```

K. GelenkPos

```
% Regler zum Verfahren der Gelenke
function GelenkPos(ROS,WinkelNeuD)
WinkelNeuV=zeros(1,5);
% Max Min Abgleich und Korektur
```

```
for i=1:5
    if WinkelNeuD(i)>ROS.Arm.Info.JointMax(i)
        WinkelNeuD(i)=ROS.Arm.Info.JointMax(i);
    elseif WinkelNeuD(i)<ROS.Arm.Info.JointMin(i)
        WinkelNeuD(i)=ROS.Arm.Info.JointMin(i);
    end
    % wenn nan nimm den Aktuellen Winkel, sonst berechne Value
    if ~isnan(WinkelNeuD(i))
        WinkelNeuV(i)=interpolieren(...
            WinkelNeuD(i),...
            ROS.Arm.Info.JointMin(i),...
            ROS.Arm.Info.JointMax(i),...
            ROS.Arm.Info.JointValueMin(i),...
            ROS.Arm.Info.JointValueMax(i));
    else
        WinkelNeuV(i)=ROS.Arm.Sub.LatestMessage.Position(i);
    end
end
% SollSp und EndSp sowie EndToleranz
SollSp=[0 0 0 0 0];
EndSp=0.01;
EndTol=0.01;
% Min Max
MinSp=0.1;
MaxSp=0.8;
% Regler
KP=[.9 .9 .7 .9 .5];
% Programm arbeitet solange run=true
run=true;
while (run)
    WinkelAltV(1:5)=ROS.Arm.Sub.LatestMessage.Position(1:5);
    % Soll/Ist Vergleich
    Soll=WinkelNeuV;
    Ist=WinkelAltV;
    deltaPos=Soll-Ist;
    for j=1:5
        if abs(deltaPos(j))>EndTol
            % P-Regler reicht weil die Regelstrecke I Anteil besitzt
            SollSp(j)=deltaPos(j)*KP(j);
            % Speed Min und Max
            if abs(SollSp(j))<MinSp
```

```
46                  if SollSp(j)<0
47                      SollSp(j)=-MinSp;
48                  else
49                      SollSp(j)=MinSp;
50                  end
51              elseif abs(SollSp(j))>MaxSp
52                  if SollSp(j)<0
53                      SollSp(j)=-MaxSp;
54                  else
55                      SollSp(j)=MaxSp;
56                  end
57              end
58              % schreiben der errechneten Geschwindigkeit in die Nachricht
59              ROS.Arm.Nach.Velocities(j).Value=SollSp(j);
60          else
61              ROS.Arm.Nach.Velocities(j).Value=0.0;
62          end
63      end
64      % Kontrolle ob Geschwindigkeiten und Abstand zum Ziel klein ist
65      if sum(abs(ROS.Arm.Sub.LatestMessage.Velocity))<EndSp && sum(abs(deltaPos))
            <(EndTol*5)
66          for k=1:5
67              ROS.Arm.Nach.Velocities(k).Value=0.0;
68          end
69          run=false;
70      end
71      % senden der geschriebenen Nachricht
72      send(ROS.Arm.Pub,ROS.Arm.Nach);
73      pause(0.01);
74  end
75  % Nachkorrektur mittels Pos-Befehl
76  for j=1:5
77      ROS.Arm.Nach2.Positions(j).Value=WinkelNeuV(j);
78  end
79  % senden der geschriebenen Nachricht
80  send(ROS.Arm.Pub2,ROS.Arm.Nach2);
81  % Kontrolle ob Geschwindigkeiten kleiner EndSp sind
82  run=true;
83  EndSp=0.01;
84  while run
85      if sum(abs(ROS.Arm.Sub.LatestMessage.Velocity))<EndSp/10
```

```
        run=false;
    end
    pause(0.01);
end
end
```

L. Ablegen

```
% Programm Ablegen, zum Ablegen durch 'nr' defienierter Objekte auf
% voreingestellten Symbol
function [Abgelegt, LeererSichtbereich]=Ablegen(ROS,nr)
% Aufheben von Ladestelle nr
W=ROS.Lade.Info.LadePositionen(nr,:);      % Winkel der Ladeposition (1,2,3)
GelenkPos(ROS,[W(1) 0 0 0 W(5)]);          % drehen des untersten Gelenks
GelenkPos(ROS,[W(1) W(2)+5 W(3) W(4)-2 W(5)]); % anfahren Vorposition
GreiferPos(ROS,20);                        % Oeffnen des Greifers
GelenkPos(ROS,[W(1) W(2) W(3) W(4) W(5)]); % anfahren Greifposition
GreiferPos(ROS,10);                        % schliessen des Greifers
GelenkPos(ROS,[W(1) W(2)+5 W(3) W(4)-2 W(5)]); % anfahren Vorposition
% Suchen nach Ablagestelle
GelenkPos(ROS,[-45 3 0 0 5]);    % anfahren Suchposition
LeererSichtbereich=VorIk(ROS,'2'); % ausrichten zum Symbol
if LeererSichtbereich % falls der Sichtbereich leer ist Objekt zuruecklegen
    Abgelegt=1;       % Abgelegt bleibt dann auf 1 => Ladung[0 1 0]
    AblegenSimpel;
else
    search=true;  % falls ausgerichtet zum Symbol
    searchCnt=0;  % nochmals ein Suchzaehler falls Symbol jetzt nicht mehr
        sichtbar
    while search
        Pos=PosMinDis(ROS,'2');      % erhalten von Positionsdaten
        if ~isempty(Pos.X)           % falls Daten vorhanden
            search=false;            % suche beenden
            if isnan(Pos.Z)          % falls aber der Z Wert nicht ermitelt
                werden kann zuruecklegen
                LeererSichtbereich=true;
                Abgelegt=1;
                AblegenSimpel;
            end
```

```
30          end
31          pause(0.1);
32          searchCnt=searchCnt+1;
33          if searchCnt>10              % falls der Suchzaehler 10 erreicht, Objekt
                 zuruecklegen
34              LeererSichtbereich=true;
35              Abgelegt=1;
36              AblegenSimpel;
37          end
38      end
39      if LeererSichtbereich==false     % falls das Symbol gefunden wurde
40          % Ablage Stelle Berechnen und Anfahren
41          [alpha, beta, gamma]=Ik([Pos.Y Pos.Z+45]);
42          GelenkPos(ROS,[nan alpha beta+5 gamma+1 nan]);
43          [alpha, beta, gamma]=Ik([Pos.Y Pos.Z+20]);
44          GelenkPos(ROS,[nan alpha beta+5 gamma+1 nan]);
45          GreiferPos(ROS,20);
46          [alpha, beta, gamma]=Ik([Pos.Y Pos.Z+45]);
47          GelenkPos(ROS,[nan alpha beta+5 gamma+1 nan]);
48          GelenkPos(ROS,[nan 3 0 0 5]);
49          Abgelegt=0;                  % Ladung[1 2 3] setzen
50      end
51  end
52  end
53  function AblegenSimpel(ROS,W)
54  GelenkPos(ROS,[W(1) 0 0 0 W(5)]);
55  GelenkPos(ROS,[W(1) W(2)+5 W(3) W(4)-2 W(5)]);
56  GelenkPos(ROS,[W(1) W(2) W(3) W(4) W(5)]);
57  GreiferPos(ROS,20);
58  GelenkPos(ROS,[W(1) W(2)+5 W(3) W(4)-2 W(5)]);
59  GelenkPos(ROS,[W(1) 3 0 0 5]);
60  end
```

M. Aufheben

```
1  % Programm Aufheben, zum Aufheben mit Symbol gekenzeichnete Objekte in durch
2  % 'nr' defienierter Ladestellen
3  function [Aufgehoben, LeererSichtbereich]=Aufheben(ROS,nr)
4  GelenkPos(ROS,[-45 3 0 0 5]);    % anfahren Suchposition
```

```
LeererSichtbereich=VorIk(ROS,'1'); % ausrichten zum Objekt
if LeererSichtbereich % falls Sichtbereich leer ist nichts machen
    Aufgehoben=0;
else
    search=true;
    searchCnt=0;
    while search
        Pos=PosMinDis(ROS,'1');  % suche nach dem Objekt
        if ~isempty(Pos.X)       % falls Daten vorhanden
            search=false;        % suche beenden
        end
        pause(0.1);
        searchCnt=searchCnt+1;
        if searchCnt>10          % falls der Suchzaehler 10 erreicht
            search=false;        % suche beenden
        end
    end
    if ~isempty(Pos.X)    % falls Daten vorhanden
        % Aufheben
        [alpha, beta, gamma]=Ik([Pos.Y Pos.Z+25]); % Vorposition berechnen
        GelenkPos(ROS,[nan alpha beta+5 gamma+1 nan]); % Vorposition anfahren
        GreiferPos(ROS,20);                        % Greifer Oeffnen
        [alpha, beta, gamma]=Ik([Pos.Y Pos.Z-25]); % Greifposition berechnen
        GelenkPos(ROS,[nan alpha beta+5 gamma+1 nan]); % Greifposition anfahren
        GreiferPos(ROS,10);                        % Greifer schliessen
        [alpha, beta, gamma]=Ik([Pos.Y Pos.Z+25]); % Vorposition berechnen
        GelenkPos(ROS,[nan alpha beta+5 gamma+1 nan]); % Vorposition anfahren
        GelenkPos(ROS,[nan 3 0 0 5]);
        % Ablegen
        W=ROS.Lade.Info.LadePositionen(nr,:);          % Winkel der Ladeposition
            (1,2,3)
        GelenkPos(ROS,[W(1) 0 0 0 W(5)]);              % drehen des untersten
            Gelenks
        GelenkPos(ROS,[W(1) W(2)+5 W(3)+5 W(4)-2 W(5)]); % anfahren Vorposition
        GelenkPos(ROS,[W(1) W(2)-1 W(3)+5 W(4)-4.5 W(5)]); % anfahren
            Greifposition
        GreiferPos(ROS,20);                            % Oeffnen des Greifers
        GelenkPos(ROS,[nan 3 0 0 5]);
        Aufgehoben=1;
    else
        LeererSichtbereich=true;
```

```
        Aufgehoben=0;
    end
end
end
```

N. GreiferPos

```
% Regler zum Verfahren des Greifers
function GreiferPos(ROS,varargin)
% den varargin mathematisch verarbeitbar machen
input=cell2mat(varargin);
% nach schauen ob eine oder beide Pos angegeben sind
% sowie berechnen der daraus folgende mm Eingaben
if size(input,2)==1
    if input>20
        input=20;
    elseif input<0
        input=0;
    end
    PosNeuMM=[input/2 input/2];
elseif size(input,2)==2
    for j=1:2
        if input(j)>10
            input(j)=10;
        elseif input(j)<0
            input(j)=0;
        end
    end
    PosNeuMM=[input(1) input(2)];
end
% Berechnen der Values der eingegebenden Position
PosNeuV=zeros(1,2);
for i=1:2
    PosNeuV(i)=interpolieren(...
        PosNeuMM(i),...
        ROS.Arm.Info.JointMin(5+i),...
        ROS.Arm.Info.JointMax(5+i),...
        ROS.Arm.Info.JointValueMin(5+i),...
        ROS.Arm.Info.JointValueMax(5+i));
```

```
end
% Schreiben und senden der Nachricht
for k=1:2
    ROS.Greifer.Nach.Positions(k, 1).Value=PosNeuV(k);
end
send(ROS.Greifer.Pub,ROS.Greifer.Nach);
% Weil in for schleife weiter definiert, hier vor definiert
Soll=zeros(1,2);
Ist=zeros(1,2);
deltaPos=zeros(1,2);
% EndToleranz
EndTol=0.00001;
% Programm arbeitet solange run=true
run=true;
while (run)
    % Soll/Ist Vergleich
    for l=1:2
        Soll(l)=PosNeuV(l);
        Ist(l)=ROS.Arm.Sub.LatestMessage.Position(5+l);
        deltaPos(l)=Soll(l)-Ist(l);
    end
    % Kontrolle ob Abstand zum Ziel klein ist
    if sum(abs(deltaPos))<(EndTol*2)
        run=false;
    end
    pause(0.1);
end
end
```

O. interpolieren

```
% Funktion zum Interpolieren
% Wert einer Skala in eine Skala umrechnen
function Out=interpolieren(In,Min1,Max1,Min2,Max2)
Out=round((In-Min1)/(Max1-Min1)*(Max2-Min2)+Min2,4);
end
```

P. PosAufnehmen

```
% Aufnehemen nur mit richtiggen Marker
function PosEnd=PosAufnehmen(ROS)
posMarker=PosMinDisSimpel(ROS,'2');
posObjekt=KreisErkennung(ROS,'w','1',20);
PosEnd=struct('X',[],'Y',[],'Z',[],'D',[],'Dis',[]);
Toleranz=5;
if size(posMarker,2)==1
    if ~isempty(posMarker.X)
        if size(posObjekt,2)==1
            if ~isempty(posObjekt.X)
                if posObjekt.X < posMarker.X+Toleranz && ...
                       posObjekt.X > posMarker.X-Toleranz && ...
                       posObjekt.Y < posMarker.Y+Toleranz && ...
                       posObjekt.Y > posMarker.Y-Toleranz
                   PosEnd=posObjekt;
                end
            else
                disp('kein Objekt');
            end
        elseif size(posObjekt,2)>1
            for i=1:size(posObjekt,2)
                if posObjekt(i).X < posMarker.X+Toleranz && ...
                       posObjekt(i).X > posMarker.X-Toleranz && ...
                       posObjekt(i).Y < posMarker.Y+Toleranz && ...
                       posObjekt(i).Y > posMarker.Y-Toleranz
                   PosEnd=posObjekt(i);
                end
            end
        end
    else
        disp('keinen Marker');
    end
else
    disp('mehrere Marker');
end
```

Q. PosMinDis

```
% nur Kreis mit geringsten Abstand zum Nullpunkt ausgeben
function PosMinEnd=PosMinDis(ROS,modus)
PosMinMulti=struct('X',[],'Y',[],'Z',[],'D',[],'Dis',[]);
PosMinEnd=PosMinMulti;
l=0;
for k=1:5
    switch modus
        case '1'
            Pos=KreisErkennung(ROS,'w','1',20);
        case '2'
            Pos=KreisErkennung(ROS,'s','2',20,52);
    end
    if size(Pos,2)>1
        Dis=zeros(1,size(Pos,2));
        for i=1:size(Pos,2)
            Dis(i)=sqrt(Pos(i).X^2+Pos(i).Y^2);
        end
        Min=Dis(1);
        MinPos=1;
        for j=2:size(Dis,2)
            if Dis(j)<Min
                Min=Dis(j);
                MinPos=j;
            end
        end
        PosMin=Pos(MinPos);
    else
        PosMin=Pos;
    end
    if ~isempty(PosMin.X)
        l=l+1;
        % mehrere Positionen speichern
        % und Vergleichen und die Mittigste nehmen
        PosMinMulti(l).X=PosMin.X;
        PosMinMulti(l).Y=PosMin.Y;
        PosMinMulti(l).Z=PosMin.Z;
        PosMinMulti(l).D=PosMin.D;
        PosMinMulti(l).Dis=PosMin.Dis;
    end
```

```
end
% Median Wahl
TempX=zeros(l,1);
TempY=zeros(l,1);
for m=1:l
    TempX(m)=PosMinMulti(m).X;
    TempY(m)=PosMinMulti(m).Y;
end
TempXMed=sort(TempX);
TempYMed=sort(TempY);
MedianX=TempXMed(round((l/2)+0.5,0));
MedianY=TempYMed(round((l/2)+0.5,0));
Toleranz=1;
Anzahl=0;
PosSumX=0;
PosSumY=0;
PosSumZ=0;
PosSumD=0;
PosSumDis=0;
for n=1:l
    if PosMinMulti(n).X<MedianX+Toleranz && ...
            PosMinMulti(n).X>MedianX-Toleranz && ...
            PosMinMulti(n).Y<MedianY+Toleranz && ...
            PosMinMulti(n).Y>MedianY-Toleranz
        Anzahl=Anzahl+1;
        PosSumX=PosSumX+PosMinMulti(n).X;
        PosSumY=PosSumY+PosMinMulti(n).Y;
        PosSumZ=PosSumZ+PosMinMulti(n).Z;
        PosSumD=PosSumD+PosMinMulti(n).D;
        PosSumDis=PosSumDis+PosMinMulti(n).Dis;
    end
end
PosEndX=PosSumX/Anzahl;
PosEndY=PosSumY/Anzahl;
PosEndZ=PosSumZ/Anzahl;
PosEndD=PosSumD/Anzahl;
PosEndDis=PosSumDis/Anzahl;
if abs(PosMinMulti(1).X)-abs(PosMinMulti(l).X)<5 || ...
        abs(PosMinMulti(1).Y)-abs(PosMinMulti(l).Y)<5
    PosMinEnd.X=PosEndX;
    PosMinEnd.Y=PosEndY;
```

```
    PosMinEnd.Z=PosEndZ;
    PosMinEnd.D=PosEndD;
    PosMinEnd.Dis=PosEndDis;
end
end
```

R. PosMinDisSimpel

```
% nur Kreis mit geringsten Abstand zum Nullpunkt ausgeben
function PosMin=PosMinDisSimpel(ROS,modus)
switch modus
    case '1'
        Pos=KreisErkennung(ROS,'w','1',20);
    case '2'
        Pos=KreisErkennung(ROS,'s','2',20,52);
end
if size(Pos,2)>1
    Dis=zeros(1,size(Pos,2));
    for i=1:size(Pos,2)
        Dis(i)=sqrt(Pos(i).X^2+Pos(i).Y^2);
    end
    Min=Dis(1);
    MinPos=1;
    for j=2:size(Dis,2)
        if Dis(j)<Min
            Min=Dis(j);
            MinPos=j;
        end
    end
    PosMin=Pos(MinPos);
else
    PosMin=Pos;
end
end
```

S. stop

```
% Zum Stoppen des Arms, auch falls noch keine Verbindung besteht
function stop()
conROS();  % Verbinden
pause(0.1); % kurzer Delay
ArmPublisher = rospublisher ('arm_1/arm_controller/velocity_command',...
'brics_actuator/JointVelocities');
ArmSubscriber = rossubscriber ('/joint_states','sensor_msgs/JointState');
ArmNachricht = rosmessage(ArmPublisher); % Nachricht konfigurieren
for i=1:5
    Gelenk(i) = rosmessage('brics_actuator/JointValue'); % Gelenk konfigurieren
    Gelenk(i).JointUri = ['arm_joint_' num2str(i)];    % Gelenk Nr
    Gelenk(i).Value = 0.0;                             % Value
    Gelenk(i).Unit = 's^-1 rad';                       % Einheit
end
ArmNachricht.Velocities = Gelenk;       % Nachricht verpacken
send(ArmPublisher,ArmNachricht);      % Nachricht senden
end
```