Hochschule Bochum
Bochum University
of Applied Sciences
Campus
Velbert/Heiligenhaus

# ABB Robotersimulation
# &
# XBox One Kinect V2

von

**Joshua Latusek, Tobias Wyrwoll, Valentin Hardkop**
Matrikelnummer: 017201686 & 017201226 & 017201258

Hausarbeit im Studiengang
Mechatronik und Informationstechnologie

**Eingereicht am:** 20.09.2021
**Prüfer:** Jan Weber

# Inhaltsverzeichnis

# 1 Einleitende Übersicht

Die Industrie befindet sich im ständigen Wandel hin zu immer stärker automatisierten Prozessen. Industrieroboter spielen hier eine entscheidende Rolle und helfen zum Beispiel dabei Fließbandproduktionen effizienter und kostengünstiger zu gestalten. Nicht alle Branchen setzen auf eine Vollautomatisierung, denn der Einsatz von geschultem Fachpersonal ist weiterhin wichtig. Dennoch ist der Robotereinsatz in vielen Branchen der heutigen Zeit nicht mehr wegzudenken, egal ob in der Industrie, der Logistik oder der Medizin, überall dort kommen Roboter mittlerweile zum Einsatz. In der Medizin können feinmotorische Roboter zum Beispiel Operationen unterstützen und größere Erfolge erzielen. Die Kontrollersteuerung ist nur eine von vielen Möglichkeiten. Eine weitere Möglichkeit stellt die Steuerung über Gesten dar. Spezielle Sensoren erfassen Bewegungen und Positionen, rechnen diese in Positionsdaten um und geben diese an Roboter weiter.
Das Projekt beinhaltet die Ansteuerung eines Roboters über Gesten. Die Robotersimulation des ABB IRB 1600-6/1.2 soll auf der virtuellen Oberfläche *Rviz* gezielt Positionen anfahren. Diese Positionen sollen vom Benutzer über eine XBox One Kinect V2 Kamera vorgegeben und abgebildet werden. Mittels einer definierten Kommunikation und Umrechnung der Positionsdaten, werden diese an den ABB Roboter übergeben. Um ein funktionierendes Einlesen von Gelenkpunkten eines Menschen (im folgenden Skeleton-Tracking genannt) implementieren zu können, müssen die einzelnen Komponenten untereinander kompatibel sein. Diese Komponenten umfassen die ROS-Version, den XBox One Kinect V2 Treiber, die Skeleton-Tracking-Software und den Treiber für den ABB Roboter.
Die Arbeit ist in die folgenden Abschnitte eingeteilt: Der erste Abschnitt befasst sich mit den Grundlagen des Projektes, dabei wird die benötigte Hardware und Software detaillierter erklärt. Im zweiten Abschnitt wird auf die Wahl der richtigen Treiber und Tracker eingegangen. Zusätzlich sind weitere Umsetzungen des Projektes, wie die Kommunikation zwischen Roboter und Skeleton-Tracking mittels Python-Code erklärt. Im nächsten Abschnitt folgt eine Installationsanleitung und abschließend eine Zusammenfassung mit einem Ausblick.

# 2 Grundlagen

Der folgende Abschnitt behandelt die für das Projekt benötigten Grundlagen. Es wird auf die verwendete Hardware und Software zur Umsetzung des Projektes eingegangen.

## 2.1 ROS - Robot Operating System

Bei ROS (Robot Operating System) handelt es sich um ein Framework für persönliche Roboter und Industrieroboter. Die Hauptbestandteile von ROS sind Hardwareabstraktion, Gerätetreiber, Paketverwaltung und Nachrichtenaustausch zwischen Programmen bzw. Programmteilen. Neben dem eigentlichen Betriebssystem ROS, gibt es noch eine große Auswahl an Zusatzpaketen, welche je nach Projekt hinzugefügt werden können. Durch die Möglichkeit einer Full-Desktop Installation können alle Pakete direkt installiert werden. [ROS-Wiki, 2013]

## 2.2 XBox One Kinect V2

Die Kinect V2 ist eine für die XBox entwickelte Kamera zur Erfassung von Gesten. Die verbaute Kamera erfasst das visuelle Umfeld und ein verbauter Tiefensensor stellt Tiefeninformationen. In Kombination mit der Spielekonsole XBox One können Spiele ohne die Verwendung eines Controllers gespielt werden. Die Kinect erfasst die Bewegungsabläufe des Spielers und übersetzt diese in die benötigten Steuerungsbefehle für das entsprechende Spiel. *libfreenect2* ist ein speziell für die Verwendung der Kinect V2 unter Linux entwickelter Treiber.


**Bild 2.1:** XBox One Kinect V2 [Jamhoury, 2018]

Die Kinect V2 kann bis zu 25 Gelenke (Joints) erkennen und dabei besitzt jedes Gelenk 11 Eigenschaften. Dazu zählen $color(x, y)$, $depth(x, y)$, $camera(x, y, z)$ und $orientation(x, y, z, w)$. [Jamhoury, 2018]

## 2.3 Skeleton-Tracking mittels der Kinect V2

Generell ist das Skeleton-Tracking mit der Kinect V2 möglich. Um die dafür benötigten Daten auszulesen muss ein Tracker als Paket in ROS eingebunden und ein passender Treiber installiert sein. Das folgende Bild 2.2 zeigt eine mögliche Kombination zu erfassender Gelenkpunkte. Auf GitHub stehen unterschiedliche Programme zum Skeleton-Tracking zur Verfügung. Um die Funktionalität sicherzustellen muss ihc Vorhinein eine Überprüfung der Kompatibilität sowohl zwischen Tracker und Treiber als auch Tracker und ROS-Version erfolgen. Das Skeleton-Tracking bietet die Möglichkeit den menschlichen Körper vor der Kamera zu lokalisieren und einzelne Körperteile zu verfolgen. Auf einer visuellen Oberfläche kann dieses Skelett nahezu in Echtzeit ausgegeben werden. [Jamhoury, 2018]

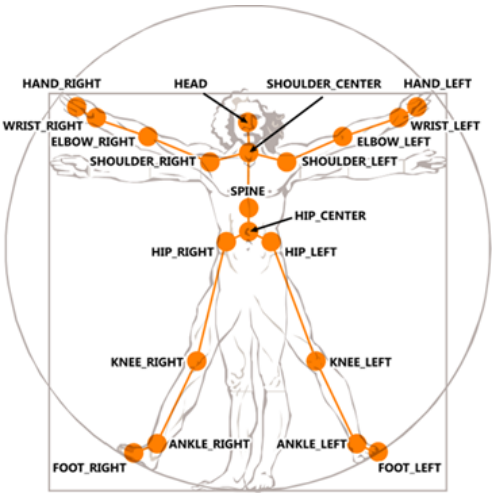


**Bild 2.2:** Skeleton-Tracking Gelenkpunkte [Jamhoury, 2018]

## 2.4 ABB IRB 1600 - Manipulator

Der ABB IRB 1600 ist ein Industrieroboter der Firma ABB und in verschiedenen Größen konfigurierbar. Die im folgenden beschriebene Ausführung des Roboter besitzt die Bezeichnung IRB 1600-6/1.2 und hat eine maximale Reichweite von 1.2 Metern. Der Roboter wird als *Manipulator* bezeichnet und besitzt 6 Achsen Bild 2.3. [Jamhoury, 2018]

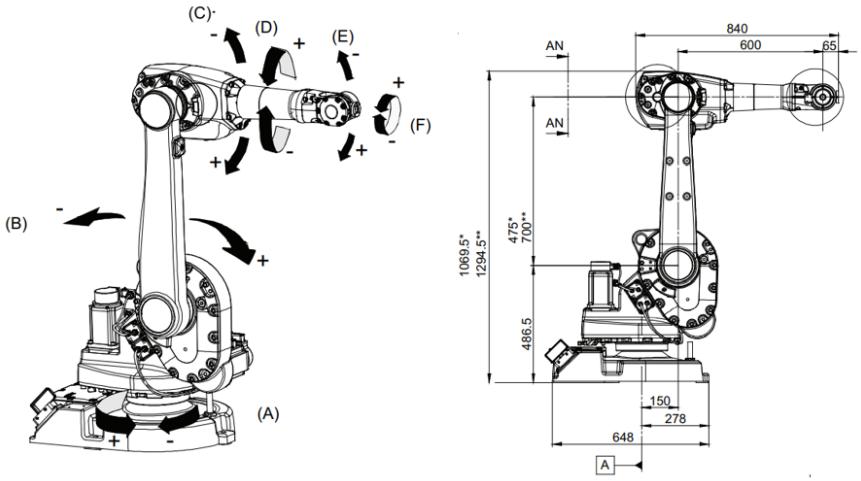


**Bild 2.3:** ABB IRB 1600 6/1.2 - Manipulator [ABB-Engineering, 2020]

Der Roboter kann die in Bild 2.4 dargestellten minimalen und maximalen Positionen auf den entsprechenden Bahnen anfahren.

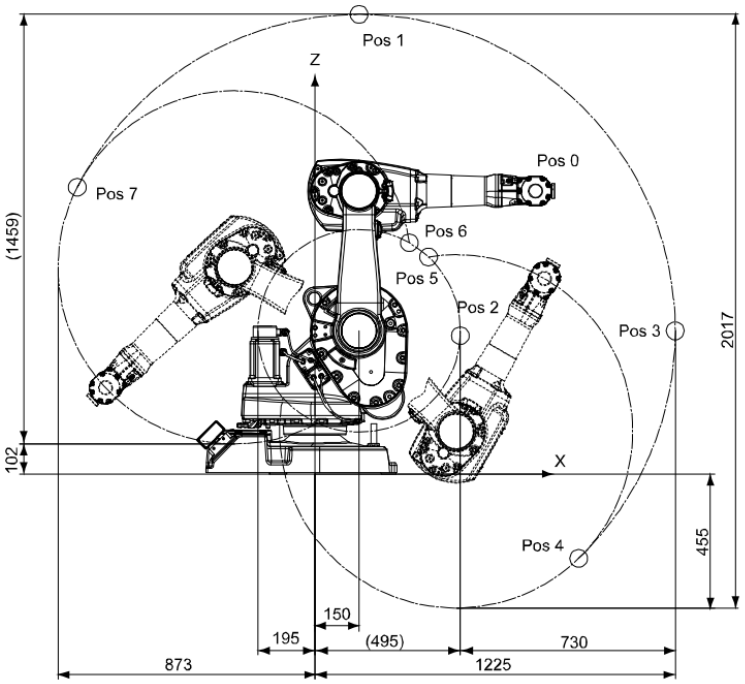


**Bild 2.4:** Positionen am Handgelenksmittelpunkt bei 1.2m Reichweite [ABB-Engineering, 2020]

## 2.5 ABB IRB 1600-6/1.2 in Rviz visualisiert

Rviz ist ein 3D Visualisierungstool für ROS Applikationen und bietet die Möglichkeit Robotermodelle abzubilden. Es kann sowohl echte und simulierte Roboter als auch Sensorinformationen darstellen. Mittels eines GitHub repositorys ist es möglich, den im vorherigen Abschnitt angesprochenen ABB IRB 1600-6/1.2 in Rviz als Robotersimulation darzustellen und anzusteuern. [ROS-Wiki, 2018]

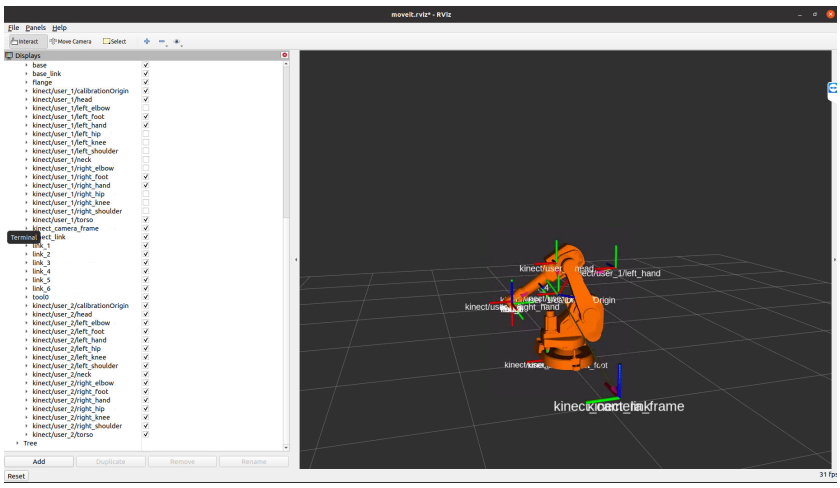

**Bild 2.5:** ABB IRB 1600-6/1.2 in Rviz

# 3 Umsetzung

Der folgende Abschnitt befasst sich mit der Umsetzung des Projektes. Hierzu zählt die Auswahl der richtigen Treiber und Tracker, sowie die benötigten Software-Pakete für die Installation.

## 3.1 Wahl des Treibers & Trackers

Für ein funktionierendes Skeleton-Tracking ist die Wahl des richtigen Treibers ebenfalls so wichtig, wie die Wahl des Trackers. Der Treiber muss sowohl mit dem Betriebssystem, als auch mit dem Tracker kompatibel sein. Hierfür wird im folgenden der verwendete Treiber beschrieben und aus einer Auswahl ein passender Tracker ausgewählt.

### 3.1.1 Treiber

Aufgrund seiner Popularität findet in dieser Arbeit der *libfreenect*2 Treiber Verwendung. Hiermit können Farbbildverarbeitungen, Registrierung von Farb- und Tiefenbildern, sowie mehrere Hardwarebeschleunigungsimplemetierungen für die Bildverarbeitung mit der Kinect genutzt werden. Zusätzlich muss das Framework *OpenNI*2 für das Skeleton-Tracking installiert sein. *OpenNI*2 ermöglicht einen generischen Zugriff auf die Kinect-Funktionen, hauptsächlich aus Bildströmen bestehend. Es bietet die Möglichkeit der Szenensegmetierung, Gestenerkennung, Handerkennung und des Skeleton-Tracking. Das GitHub repository für *libfreenect*2 ist unter dem folgenden Link zu finden: *https* : *//github.com/OpenKinect/libfreenect*2

### 3.1.2 Tracker

Im vorherigen Abschnitt wurde das Problem der Kompatibilität bezüglich des richtigen Trackers angesprochen. Neben der Kompatibilität mit der ROS-Version, muss der Tracker mit dem vorher installierten Treiber lauffähig sein. Die nachfolgende Tabelle beinhaltet die am weitesten verbreiteten Tracker und stellt einen Vergleich der Übereinstimmung der benötigten Anforderungen dar. Der gesuchte Tracker muss auf der Ubuntu 20.04 Version mit ROS Noetic, *OpenNI*2 und *libfreenect*2 lauffähig sein. Kompatibilitätsprobleme können trotzdem jederzeit aufgrund von Systemaktualisierungen auftreten und müssen während der Installation beobachtet werden.

| **Tracker** | **ROS Noetic kompatibel** | **Weiteres** |
|---|---|---|
| *skeleton_markers* | Nein | max. Ubuntu 14.04 |
| *openni*2*_tracker* | Ja | x |
| *kinect_v*2*_skeleton_tracking* | Nein | benötigt *OpenNI*2 |
| *kinect*2*_tracker* | Ja | läuft mit *libfreenect*2 |

Der *skeleton_markers*- und der *kinect_v2_skeleton_tracking*-Tracker sind aufgrund der Inkompatibilität und der benötigten Ubuntu Version keine Option für das Projekt. Während hingehen der *openni2_tracker* und der *kinect2_tracker* beide für die ROS Noetic-Version geeignet sind. Aufgrund der Kompatibilität der beiden Tracker, stehen diese in der engeren Auswahl für die spätere Installation. Hierbei könnte der *kinect2_tracker* durch seine direkte Kompatibilität mit *libfreenect2* von Vorteil sein.

## 3.2 Kommunikation zwischen Skeleton-Tracking und Robotersimulation

Die Kommunikation zwischen den erfassten Daten des Skeleton-Trackings und der Robotersimulation ist durch den Datenaustausch mittels einer Ansteuerung über ein Python-Script realisiert.

## 3.3 Ausrichtung der Koordinatensysteme

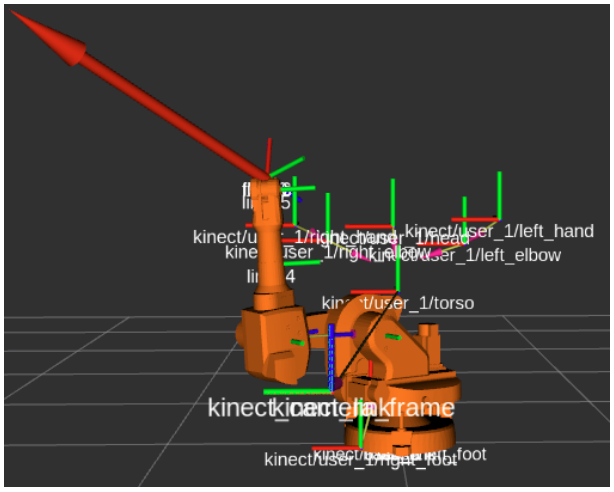


**Bild 3.6:** Skeleton-Tracking mit ausgerichtetem Roboter zur Handposition

Die von der Kinect V2 getrackten Gelenkpunkte des Skelett sind mit Hilfe des *kinect2_trackers* in Rviz visualisiert. Hierbei ist zu beachten, dass der im Ursprung fixierte Punkt der *kinect_camera_frame* ist, welche die Position der Kinect Kamera im Raum darstellt. Das Skelett steht von diesem Punkt in gleicher Entfernung wie der Nutzer von der Kamera. Damit die Roboteransteuerung im weiteren Verlauf besser umgesetzt werden kann, gibt es verschiedene Möglichkeiten das Skelett an den Roboter zu fixieren. Im folgenden finden sich zwei Lösungsansätze.
Im ersten Lösungsansatz ist ein Schultergelenk im Ursprung des Roboters (base_link) positioniert. Der Roboter mit seiner Armreichweite von 1.20m kann vom Skelettarm simuliert werden. Hierzu können über die verschiedenen Gelenkpunkte des Armes die

Achsen des Roboters angesteuert werden.
Bei dem zweiten Lösungsansatz wird der linke Fuß des Skelettes in den Ursprung (base_link) des Roboters gelegt. In diesem Fall kann der Roboter zum Beispiel über die rechte oder linke Hand im Raum bewegt werden. Die von der Hand ausgelesene Position wird skaliert, da der Roboter eine größere Reichweite hat, der durchschnittliche Nutzer. Im Pythonscript wird das Körperteil zur Steuerung des Roboters festgelegt. Zusätzlich zu der reinen Translation des erkannten Skelettes ist eine Rotation nötig, um die Ausrichtung von Roboter und Skelett gleichzusetzen. Die Verbindung zwischen linken Fuß und base_link, sowie die Rotationen können über den folgenden Befehl ausgeführt werden.

```
rosrun tf static_transform_publisher 0 0 0 -0.5 0.5 0.5 0.5
    kinect/user_1/left_foot base_link 10
```

## 3.4 Übergabeform der Parameter

Neben der Verknüpfung der einzelnen Koordinatensysteme stellt sich die Frage nach der Ansteuerung des Roboters. Hierbei handelt es sich um *MoveIt* eine von ROS entwickelte Schnittstelle zur Ansteuerung von Robotern. Sie besitzt verschiedene Funktionen zur Ansteuerung der Roboterfunktionalität. Der folgende Codeabschnitt stellt einen Ausschnitt aus dem Move Group Python Interface Tutorial. [MoveIt, 2021]

```
class MoveGroupPythonInterfaceTutorial(object):
  """MoveGroupPythonInterfaceTutorial"""

  def __init__(self):
  super(MoveGroupPythonInterfaceTutorial, self).__init__()

  ## BEGIN_SUB_TUTORIAL setup
  ##
  ## First initialize `moveit_commander`_ and a `rospy`_ node:
  moveit_commander.roscpp_initialize(sys.argv)
  rospy.init_node("user_1_hand_follower")

  ## Instantiate a `RobotCommander`_ object. Provides information such as the
      robot's
  ## kinematic model and the robot's current joint states
  robot = moveit_commander.RobotCommander()

  ## Instantiate a `PlanningSceneInterface`_ object.  This provides a remote
      interface
  ## for getting, setting, and updating the robot's internal understanding of the
  ## surrounding world:
  scene = moveit_commander.PlanningSceneInterface()

  ## Instantiate a `MoveGroupCommander`_ object.  This object is an interface
  ## to a planning group (group of joints).  In this tutorial the group is the
      primary
  ## arm joints in the Panda robot, so we set the group's name to "panda_arm".
  ## If you are using a different robot, change this value to the name of your robot
  ## arm planning group.
  ## This interface can be used to plan and execute motions:
```

```
group_name = "manipulator"
move_group = moveit_commander.MoveGroupCommander(group_name)

## Create a `DisplayTrajectory`_ ROS publisher which is used to display
## trajectories in Rviz:
display_trajectory_publisher = rospy.Publisher(
"/move_group/display_planned_path",
moveit_msgs.msg.DisplayTrajectory,
queue_size=1,
)
```

Die Klasse *MoveGroupPythonInterfaceTutorial* stellt die benötigten Objekte und Variablen zur Verfügung. Wichtig ist hierbei, die Initialisierung des *moveit_commander* und der *rospy* -Node. In diesem Fall ist der Node *user_1_hand_follower* zu übergeben. Des weiteren ist der richtige Gruppenname, also die Bezeichnung des Roboters einzutragen. Die in diesem Projekt verwendete Robotersimulation hat den Namen *Manipulator*. Das Pythonscript bietet mehrere Funktionen zur Ansteuerung des Roboters. Der folgende Unterabschnitt beschreibt zwei dieser Funktionen genauer.

### 3.4.1 Die Funktionen go_to_pose_goal & set_joint_goal

Die *go_to_pose_goal* -Funktion ist in der Lage durch die Vorgabe einer Zielposition des Roboters diese selbständig mit seinem Greifer anzufahren. Diese Position muss dementsprechend im Bereich der möglichen Anfahrpositionen liegen. Zur Berechnung der fehlenden Gelenkwinkel benutzt MoveIt die inverse Kinematik. Die *set_joint_goal* -Funktion hingegen benötigt alle Gelenkwinkel und richtet den Roboter dementsprechend zur Zielposition aus. Die *go_to_pose_goal* -Funktion bietet eine für den Anfang einfache Umsetzung, da diese lediglich die gewünschte Position benötigt und die Umrechnungen und Positionierung automatisch durchführt. Die *Move_group_class* zusammen mit den im späteren Abschnitt geschriebenen Pythonscripts kontrollieren den Bewegungsablaufes des ABB IRB 1600 [MoveIt, 2021].

## 3.5 Code für die Kommunikation mittels Python

Die Kommunikation zwischen den Daten des Skeleton-Trackers und der ABB Robotersimulation ist mittels eines Python-Scripts und dem zur Verfügung gestellten Programm von MoveIt realisiert. Der Publisher veröffentlicht die von der Kinect V2 erkannte Handpose in dem Script *hand_follower.py*. Die Handpose übergibt er dabei als geometry_msgs.msg.Pose in dem Objekt *hand_pose*. Der *listener* empfängt die Transformation zwischen *base_link* und der *right_hand*. Die ausgelesenen Daten speichert das Programm in *hand*1 ab und veröffentlicht sie mittels des Befehls *hand_pose.publish*(*hand*1). Dies geschieht mit einer Rate von 0.2 Hertz, also alle 5 Sekunden.

```
#!/usr/bin/env python3
import rospy
import tf
import geometry_msgs.msg
import std_msgs.msg
import numpy as np
from math import ceil
from tf.transformations import quaternion_from_euler

if __name__ == '__main__':
  rospy.init_node('user_1_hand_follower')
  rate = rospy.Rate(0.2)
  listener = tf.TransformListener()
  hand_pose = rospy.Publisher('kinect2/hand_pose', geometry_msgs.msg.Pose,
      queue_size=0)
  hand_pose_stamped = rospy.Publisher('kinect2/hand_pose_stamped',
      geometry_msgs.msg.PoseStamped, queue_size=0)
  while not rospy.is_shutdown():
    try:
      (trans,rot) = listener.lookupTransform('base_link',
          'kinect/user_1/right_hand', rospy.Time(0))
    except (tf.LookupException, tf.ConnectivityException,
        tf.ExtrapolationException):
      continue
    try:
      (trans_elbow,rot_elbow) = listener.lookupTransform('base_link',
          'kinect/user_1/right_elbow', rospy.Time(0))
    except (tf.LookupException, tf.ConnectivityException,
        tf.ExtrapolationException):
      continue
```

*deltax*, *deltay* und *deltaz* beinhalten jeweils die Differenz der Koordinaten zwischen der Handposition und der Ellenbogenposition des rechten Armes. Das Programm berechnet darauf folgend die eulerschen Differenzwinkel aus den Koordinatendifferenzen. Tritt bei dieser Berechnung ein Fehler auf (z.B. bei Division durch Null) ersetzt es den Winkel durch Null. Die so berechneten Differenzwinkel $(roll, pitch, yaw)$ übergibt das Programm der Funktion $quaternion\_from\_euler()$, welche die Orientierung als Quaternion zurückgibt. Den Positionseigenschaften der Pose-Nachricht *hand*1 weist das Programm die Transformation zwischen *base_link* und *kinect/user/right_hand* zu und skaliert und rundet diese. Das zuvor berechnete Quaternion übergibt das Programm unverändert der Orientierung von *hand*1. Zudem erstellt das Programm ein PoseStamped-Objekt dem die selben Eigenschaften, sowie ein Zeitstempel zugewiesen werden.

```
    hand1 = geometry_msgs.msg.Pose()

    deltax = trans[0] - trans_elbow[0]
    deltay = trans[1] - trans_elbow[1]
    deltaz = trans[2] - trans_elbow[2]

    try:
    yaw = np.arctan(deltay/deltax)
    except:
    yaw = 0
    try:
```

```
        roll = -np.arctan(deltaz/deltay)
        except:
        roll = 0
        try:
        pitch = -np.arctan(deltaz/deltax)
        except:
        pitch = 0

        x, y, z, w = quaternion_from_euler(roll, pitch, yaw)

        hand1.position.x = round(trans[0] * 1.2, 2)
        hand1.position.y = round(trans[1] * 1.2, 2)
        hand1.position.z = round(trans[2] * 1.2, 2)

        hand1.orientation.x = x
        hand1.orientation.y = y
        hand1.orientation.z = z
        hand1.orientation.w = w

        pose = geometry_msgs.msg.PoseStamped()
        pose.header = std_msgs.msg.Header()
        pose.header.stamp = rospy.Time.now()
        pose.header.frame_id = "base_link"
        pose.pose = hand1

        hand_pose.publish(hand1)
        hand_pose_stamped.publish(pose)

        rate.sleep()
```

Das Script *abb_mover.py* liest über einen *Subscriber* die in der *geometry_msgs.msg* abgespeicherten Daten der rechten Hand aus und bindet diese an das Objekt *pose_goal*, welches die Daten über die Callback-Funktion erhält. Aus dem erstellten *move_group_object* der $MoveGroupPythonInterfaceTutorial()$ Klasse ruft das Programm die *go_to_pose_goal* - Funktion auf. Dieser Methode werden die vom Subscriber erhaltenen Daten der rechten Hand übergeben.

```
#!/usr/bin/env python3
import rospy
import geometry_msgs.msg
from move_group_class import MoveGroupPythonInterfaceTutorial
import moveit_commander

def callback(data):
  pose_goal = geometry_msgs.msg.Pose()
  pose_goal.orientation.x = data.orientation.x
  pose_goal.orientation.y = data.orientation.y
  pose_goal.orientation.z = data.orientation.z
  pose_goal.orientation.w = data.orientation.w
  #pose_goal.orientation.w = 1
  pose_goal.position.x = data.position.x
  pose_goal.position.y = data.position.y
  pose_goal.position.z = data.position.z
  move_group_object.go_to_pose_goal(pose_goal)

def listener():
  rospy.init_node('user_1_hand_follower')
  rospy.Subscriber('kinect2/hand_pose', geometry_msgs.msg.Pose, callback)
```

```
    rospy.spin()

if __name__ == '__main__':
    move_group_object = MoveGroupPythonInterfaceTutorial()
    listener()
```

Das von MoveIt zur Verfügung gestellte *move_group_class.py* -Script ist in der Lage, die über die *go_to_pose_goal* -Funktion übergebenen Parameter mit der Robotersimulation anzufahren.

```
    def go_to_pose_goal(self, pose):
    # Copy class variables to local variables to make the web tutorials more clear.
    # In practice, you should use the class variables directly unless you have a
        good
    # reason not to.
    move_group = self.move_group

    ## BEGIN_SUB_TUTORIAL plan_to_pose
    ##
    ## Planning to a Pose Goal
    ## ^^^^^^^^^^^^^^^^^^^^^^^
    ## We can plan a motion for this group to a desired pose for the
    ## end-effector:
    pose_goal = pose

    move_group.set_pose_target(pose_goal)

    ## Now, we call the planner to compute the plan and execute it.
    plan = move_group.go(wait=True)
    # Calling `stop()` ensures that there is no residual movement
    move_group.stop()
    # It is always good to clear your targets after planning with poses.
    # Note: there is no equivalent function for clear_joint_value_targets()
    move_group.clear_pose_targets()

    ## END_SUB_TUTORIAL

    # For testing:
    # Note that since this section of code will not be included in the tutorials
    # we use the class variable rather than the copied state variable
    current_pose = self.move_group.get_current_pose().pose
    return all_close(pose_goal, current_pose, 0.01)
```

# 4 Installation

Der folgende Abschnitt beschreibt die Installation der für das Projekt benötigten Softwarepakete. Vor der Installation muss eine ROS-Version auf dem PC installiert sein. Die Installation ist unter der Full-Desktop-Version von ROS-Noetic installiert. Der verwendete Computer nutzt Ubuntu 20.04. Die anfängliche Installation von libfreenect2, TurboJPEG, OpenGL, CUDA und VAAPI sind von der Website [Peng, 2018] mit entsprechendem Reporsitory entnommen.

Die folgenden Schritte müssen für die Installation abgearbeitet werden:

Das Git von Libfreenect2 downloaden:

```
cd catkin_ws
cd src
git clone https://github.com/OpenKinect/libfreenect2.git
cd libfreenect2
```

Herunterladen der upgrade deb files:

```
cd depends; ./download_debs_trusty.sh
```

Installieren der build tools:

```
sudo apt-get install build-essential cmake pkg-config
```

Installieren von libusb - die Version muss neuer 1.0.20 sein:

```
sudo apt-get install libusb-1.0-0-dev
```

Installieren von TurboJPEG:

```
sudo apt-get install libturbojpeg0-dev
```

Installieren von OpenGL für Intel CPU:

```
sudo apt-get install beignet-dev
```

Die Installation von CUDA erfolgt im ersten Schritt über die Seite von NVIDIA:

```
https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64
    &Distribution=Ubuntu&target_version=20.04&target_type=deb_local
```

Die folgenden Schritte sind auf der NVIDIA Seite wiederzufinden:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/
    x86_64/cuda-ubuntu2004.pin
sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget https://developer.download.nvidia.com/compute/cuda/11.4.1/local_installers/
    cuda-repo-ubuntu2004-11-4-local_11.4.1-470.57.02-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2004-11-4-local_11.4.1-470.57.02-1_amd64.deb
sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/7fa2af80.pub
sudo apt-get update
sudo apt-get -y install cuda
```

Installieren von VAAPI:

```
sudo apt-get install libva-dev libjpeg-dev
```

Installieren von OpenNI2:

```
sudo apt-get install libopenni2-dev
```

Build durchführen:

```
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/freenect2
make
make install
```

CMake benötigt eine Spezifikation:

```
cmake -Dfreenect2_DIR=$HOME/freenect2/lib/cmake/freenect2
```

Setzen der udev Regeln für den Gerätezugriff:

```
sudo cp ../platform/linux/udev/90-kinect2.rules /etc/udev/rules.d/
```

In separaten Fenster wird Roscore gestartet:

```
roscore
```

An dieser Stelle muss ein build- und bash-Befehl durchgeführt werden:

```
cd catkin_ws
catkin_make_isolated
source devel_isolated/setup.bash
```

Testprogramm in *catkin_ws/src/libfreenect2/build* laufen lassen :

```
./bin/Protonect
```

Das Testprogramm zeigt vier Kamerabilder, welche die möglichen Arten der Kameraaufnahme darstellen.


**Bild 4.7:** Das libfreenect2 Kameratestbild

Als nächstes wird NiTE2 installiert. Hierzu müssen folgende Schritte durchgeführt werden:

```
cd PATHTO/HandTrack/src
wget http://jaist.dl.sourceforge.net/project/roboticslab/External/nite
    /NiTE-Linux-x64-2.2.tar.bz2
tar xjvf NiTE-Linux-x64-2.2.tar.bz2 && rm NiTE-Linux-x64-2.2.tar.bz2
cd NiTE-Linux-x64-2.2
sudo ./install.sh
cd ..
cp libfreenect/build/lib/OpenNI2-FreenectDriver/libFreenectDriver.so
    NiTE-Linux-x64-2.2/Samples/Bin/OpenNI2/Drivers/
cp OpenNI-Linux-x64-2.2/Redist/libOpenNI2.so NiTE-Linux-x64-2.2/Samples/Bin
source setup_nite.bash
```

Der im vorherigen Abschnitt ausgewählte Tracker wird aus dem entsprechenden Git in den */src* -Ordner heruntergeladen.

```
cd catkin_ws
cd src
git clone https://github.com/mcgi5sr2/kinect2_tracker
cd ..
```

An dieser Stelle muss ein build- und bash-Befehl durchgeführt werden.

```
cd catkin_ws
catkin_make_isolated
source devel_isolated/setup.bash
```

Der Tracker kann mittels dem Launch-file getestet werden:

```
roslaunch kinect2_tracker tracker.launch
```

Das vom Tracker aufgezeichnete Skelett soll später mit der Robotersimulation ABB IRB 1600-6/1.2 verknüpft werden. Hierzu wird der Treiber *abb_egm_driver* für die Robotersimulation in Rviz aus dem folgenden GitHub in */src* geklont [Weber, 2021].

```
sudo apt update
sudo apt dist-upgrade
sudo apt install git swig libnlopt-cxx-dev ros-noetic-nlopt ros-noetic-catkin
    ros-noetic-moveit
git clone https://github.com/ros-industrial/industrial_core.git
cd industrial_core
sudo apt install git-extras
git pr 258
cd ..
cd src
git clone --recursive https://gitlab.cvh-server.de/jweber/abb_egm_driver.git
catkin build
cd src
git clone https://bitbucket.org/traclabs/trac_ik.git
```

Es muss die Zeile 35 in trac_ik/trac_ik_lib/include/nlopt_ik.hpp durch den folgenden Ausdruck ersetzt werden.

```
#include <nlopt/nlopt.hpp>
```

An dieser Stelle muss ein build- und bash-Befehl durchgeführt werden.

```
cd catkin_ws
catkin_make_isolated
source devel_isolated/setup.bash
```

In einem separaten Fenster wird das Skelett in das Koordinatensystem des Roboters gelegt.

```
rosrun tf static_transform_publisher 0 0 0 -0.5 0.5 0.5 0.5
    kinect/user_1/left_foot base_link 10
```

Der letzte Befehl öffnet Rviz und bildet die Robotersimulation und das aufgezeichnete Skeleton-Tracking ab.

```
roslaunch abb_irb1600_6_12_moveit_config demo.launch
```

# 5 Launch-File & Git-Repository

Nach der Installation der im vorherigen Abschnitt geschilderten Schritte, kann der Skeleton-Tracker und die Robotersimulation für eine Ansteuerung des ABB Roboters mittels der Kinect V2 über das folgende Launch-File gestartet werden:

```
roslaunch kinect2_tracker abb_kinectv2_control.launch
```

Sollte die Kinect V2 nicht direkt das vom Benutzer gewünschte Skelett darstellen, muss sich dieser frontal vor der Kinect positionieren und leichte Bewegungen ausführen. Die Kinect benötigt einen Moment zur Erkennung des Skeletts und ist dann Einsatzbereit. Durch eine Handbewegungen der rechten Hand, kann der ABB IRB 1600-6/1.2 jetzt gesteuert werden.

Der proprietäre Code ist unter dem folgenden Link im Git-Reporsitory wiederzufinden:

https://github.com/VHardkop/abb_kinectv2_control

# 6 Zusammenfassung & Ausblick

Ziel dieser Arbeit war es die Ansteuerung einer Robotersimulation eines ABB IRB 1600-6/1.2 über Skeleton-Tracking mithilfe einer XBox One Kinect V2. Dabei musste das Robot Operating System (ROS) als Basis unter Ubuntu Verwendung finden.

Nach erfolgreicher Implementierung ist es jetzt möglich die Position und Orientierung des Greifers mithilfe des rechten Armes zu manipulieren. Dafür ist neben der Installation unterschiedlicher Pakete unter ROS eine Schnittstelle zwischen einem Skeleton-Tracker und der Robotersimulation implementiert worden. Das so entstandene Git ermöglicht dem Nutzer das Starten der Rviz-Umgebung und des Trackers mithilfe der Verwendung eines launch-files.

Nach anfänglichen Problemen mit der Kompatibilität der einzelnen Installationspakete für ROS, verfolgt die Arbeit einen Installationsansatz aus dem Internet. Dieser hat die Ansteuerung der Kinect V2 ermöglicht, jedoch das Skeleton-Tracking als unmöglich angesehen. Durch die Wahl eines zu diesem Zweck entwickelten Trackers ($kinect2\_Tracker$) zusammen mit $libfreenect2$ bietet das launch-file die Möglichkeit das menschliche Skelett zu Tracken und in Rviz zu visualisieren. Die Kommunikation zwischen dem ABB EGM Driver und dem $kinect2\_Tracker$ ist mittels eines Python-Scripts umgesetzt und übergibt die benötigten Daten an die Robotersimulation.

Folgende Projekte können sich intensiver mit der Ansteuerung der Robotersimulationen beschäftigen und verschiedenste Lösungsansätze bezüglich der Orientierung und Positionierung umsetzen. Eine Möglichkeit zur Verbesserung bietet die Anpassung der Roboterparameter und Trajektorienplanung dahingehend, dass der Roboter in der Lage ist der Hand zu folgen statt einzelne Positionen anzufahren.

# Literatur- und Quellenverzeichnis

ABB-Engineering. Produktspezifikation IRB 1600, 2020. https://search.abb.com/library/Download.aspx?DocumentID=3HAC023604-003&LanguageCode=de&DocumentPartId=&Action=Launch.

L. Jamhoury. Understanding Kinect V2 Joints and Coordinates System, 2018. https://lisajamhoury.medium.com/understanding-kinect-v2-joints-and-coordinate-system-4f4b90b9df16.

MoveIt. MoveIt Tutorials, 2021. https://ros-planning.github.io/moveit_tutorials/.

Peng. Modern Skeleton Tracking Benchmarking, 2018. https://msr-peng.github.io/portfolio/projects/skeleton_tracking/.

ROS-Wiki. Was ist ROS?, 2013. http://wiki.ros.org/de/ROS/Introduction.

ROS-Wiki. rviz, 2018. http://wiki.ros.org/rviz.

J. Weber. abb egm driver, 2021. https://gitlab.cvh-server.de/jweber/abb_egm_driver.

# Abbildungsverzeichnis

# Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst und keinen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Die Regelungen der geltenden Prüfungsordnung zu Versäumnis, Rücktritt, Täuschung und Ordnungsverstoß habe ich zur Kenntnis genommen.

Diese Arbeit hat in gleicher oder ähnlicher Form keiner Prüfungsbehörde vorgelegen.

Düsseldorf, den 20.09.2021 T. Wyrwoll

Unterschrift

Mannhagen, den 20.09.2021

Unterschrift

Mettmann, den 20.09.2021

Unterschrift