h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik–

Konzeption und Implementierung einer KI-gestützten Bin-Picking-Anwendung für einen mit einer Tiefenkamera ausgestatteten kollaborativen Roboter

Abschlussbericht: Vertiefende F&E-Studien

Sommersemester 2021

David Leonhardt

david.leonhardt@stud.h-da.de

Matrikelnummer: 745708

Dozent : Prof. Dr. Thomas Horsch

16. Juli 2021

ZUSAMMENFASSUNG

Im Rahmen dieser F&E Studien wurde ein Software-Konzept für das Generieren von möglichen Greifposen für den kollaborativen Panda Roboter der Firma FRANKA EMIKA ausgearbeitet sowie ein konkreter Prototyp implementiert. Für die Erkennung der Werkstücke wurde die Tiefenkamera Intel RealSense LiDAR Camera L515 an dem Roboter montiert.

Da sowohl die Tiefenkamera als auch der Roboter gut in das Robot Operating System (ROS) integriert und daraus ansteuerbar sind, bietet sich ROS als Grundlage für die Softwareentwicklung an. Für das Detektieren der Bauteile und die Ermittlung der zum Greifen nötigen Endeffektorposen (translatorisch und rotatorisch) wurde eine KI integriert, die die von der Tiefenkamera aufgenommenen Punktewolken verarbeitet und mögliche Greifposen generiert und klassifiziert.

Es wurde eine beispielhafte Bin-Picking-Applikation umgesetzt, um die untersuchten Technologien in einem konkreten Einsatzgebiet zu testen. Diese Tests wurden einerseits in der Simulationsumgebung Gazebo und andererseits am realen Roboter durchgeführt. Dabei wurden in der Simulation gute Ergebnisse erzielt. Bei der Übertragung auf den realen Prototyp stellte sich jedoch heraus, dass die Kalibrierung der Kamerapose von hoher Wichtigkeit und nur mit zusätzlichem Aufwand umzusetzen ist.

INHALTSVERZEICHNIS

1 EINLEITUNG

1.1 MOTIVATION

Das Aufgabenfeld von Robotern ist in den letzten Jahren zunehmend anspruchsvoller geworden. Daher ist die Steigerung der „Intelligenz“ des Roboters ein wichtiges Gebiet in der Forschung und Entwicklung. Ein konkretes Beispiel dafür ist der sogenannte „Griff in die Kiste“ (engl. Bin Picking), bei dem der Roboter mit einem ungeordneten Behälter voller Werkstücke konfrontiert wird und diese dynamisch erkennen und greifen soll.

Die größte technologische Herausforderung bei der Umsetzung solcher Anwendung liegt darin, dem Roboter die Wahrnehmung (engl. Perception) seiner Umwelt zu ermöglichen und diese für die jeweilige Anforderung optimal auszuwerten. In diesem Zusammenhang werden zunehmend künstliche Intelligenzen (KI) eingesetzt, um die digitalen Bild- und Tiefendaten der Umwelt zu verarbeiten.

1.2 DEFINITION DER ARBEITSPAKETE

Im Folgenden sind die zu Beginn des Projektes festgelegten Arbeitpakete aufgeführt:

AP 1: Definition der Anforderungen

Zu Beginn sollen die Anforderungen an das System definiert werden. Dazu gehören einerseits Eigenschaften der zu greifenden Objekte sowie deren Lage im Umfeld des Roboters. Andererseits soll der Ablauf des Prozesses definiert werden.

AP 2: Recherche der Lösungsmöglichkeiten

Bevor Implementierungen stattfinden können, sollen zunächst grundlegende Lösungsmöglichkeiten für alle Teilaufgaben recherchiert werden. Dabei werden Programmierumgebungen, Programmiersprachen, konkrete Bibliotheken und Robotersimulatoren untersucht und auf den Nutzen im System geprüft.

AP 3: Inbetriebnahme der Tiefenkamera

Da die Tiefenkamera die zentrale Hardwarekomponente im System ist, wird diese zuerst in Betrieb genommen. Es soll möglich sein, ihre Daten aus einem Programm heraus auszulesen und weiterzuverarbeiten.

Intel bietet eine SDK, um die Kamera aus einem Programm heraus ansprechen zu können. Alternativ existiert ein ROS Paket von Intel, das die direkte Integration der Kamera in das Robot Operating System ermöglicht.

AP 4: Simulation des Panda Roboters mit Kamera

Um nicht an die reale Hardware gebunden zu sein, soll eine Simulation ermöglich werden. Es sollen dabei sowohl der Roboter als auch die Tiefenkamera simulierbar und analog zu den realen Gegenstücken ansteuerbar sein.

Die für den Panda Roboter oft verwendete Simulationssoftware Gazebo bietet Möglichkeiten, Sensoren und Kameras (auch Tiefenkameras) individuell zu modellieren und in eine Simulation einzubinden. Des Weiteren ist die Integration von Gazebo in ROS sehr einfach und bewährt.

AP 5: Fusion von Punktewolken

Damit ein guter Überblick über die ganze Arbeitsfläche garantiert werden kann, soll ein Verschmelzen mehrerer Punktewolken, die aus verschiedenen Kamerawinkeln aufgenommen wurden, im Programm möglich sein.

AP 6: Objekt- und Greifposenerkennung

Das Erkennen eines zu greifenden Objektes und der zum Greifen am besten geeigneten Endeffektorpose ist das Kernstück der Studie. Es soll möglich sein, unterschiedliche, zufällig angeordnete Objekte zuverlässig zu lokalisieren und eine passende Greifpose zu ermitteln, die ein sicheres Greifen des Objektes ermöglicht.

Generell können verschiedene Verfahren für dieses Vorhaben eingesetzt werden. Dazu zählen bewährte Methoden der Bild- und Punktewolkenverarbeitung. Der Fokus soll dabei auf Varianten liegen, die auf dem Einsatz einer künstlichen Intelligenz basieren.

AP 7: Softwarearchitektur

Es soll ein logisches Softwarekonzept für die Interaktion aller Teilkomponenten erstellt werden. Die Architektur soll so modular wie möglich aufgebaut sein, damit im Nachgang einfach Änderungen oder Erweiterungen an einzelnen Programmteilen vorgenommen werden können.

Das Robot Operating System bietet ein gutes grundlegendes Software-Framework mit vielen für die Robotik und Bildverarbeitung geeigneten Funktionen und Bibliotheken. Es unterstützt sowohl die eingesetzte Tiefenkamera als auch den Panda Roboter und ermöglicht den angestrebten modularen Aufbau der Software.

AP 8: Implementierung des Prototyps

Abschließend soll ein funktionsfähiger Prototyp umgesetzt werden, wobei die realen Hardwarekomponenten für die Realisierung einer beispielhaften Bin Picking Applikation verwendet werden. Die Applikation soll einfach durch den Bediener steuer- und parametrierbar sein, z.B. mit Hilfe einer graphischen Benutzeroberfläche.

1.3 ABLAUF DES PROJEKTS

Das Pojekt startete Anfang April 2021 mit der Anforderungsdefinition (AP 1). Es folgten Recherchen zu möglichen Umsetzungsmöglichkeiten dieser Anforderungen (AP 2). Die Inbetriebnahme der Tiefenkamera (AP 3) fand zunächst ohne eine Anbindung an den Roboter statt. Im Anschluss wurde die Simulationsumgebung des Roboters mit Tiefenkamera (AP 4) ermöglicht. In diesem Zusammenhang wurden parallel die Fusion der Punktewolken (AP 5) und die Greifposenerkennung (AP 6) umgesetzt. Nach dem Abschluss dieser Voraussetzungen wurde eine passende Softwarearchitektur für die gesamte Anwendung (AP 7) erstellt. Die implementierte und in der Simulation getestete Software wurde dann abschließend auf einen realen Prototypen übertragen (AP 8). Das Projekt endete schließlich im Juli 2021 mit einem funktionsfähigen simulierten Bin-Picking-Prozess, bislang ohne einen Behälter. An der realen Hardware lief der Prozess grundlegend, war aufgrund der nicht optimal kalibrierten Kamera jedoch noch nicht ganz zuverlässig.

2 UMSETZUNG

2.1 DEFINITION DER ANFORDERUNGEN (AP 1)

Es wurden Anforderungen an die zu greifenden Objekte, die Umgebung des Roboters und die Grenzen des mit dem System Möglichen definiert. Diese Anforderungen lauteten wie folgt:

2.1.1 *Art der zu greifenden Objekte*

Grundsätzlich soll das Greifen verschiedenster Arten von Objekten ermöglicht werden. Dabei müssen sie nicht geordnet vorliegen, sondern können sich ungeordnet auf einem Haufen oder in einer Kiste befinden. Da die Erkennung der Greifposen über die 3D-Form der Objekte stattfinden soll, sind die Farben der Objekte zunächst irrelevant. Optional könnte die Farbe für die Detektion eines konkreten Objekts verwendet werden.

2.1.2 *Roboterumgebung und Bedingungen*

Die zu greifenden Objekte befinden sich auf dem Arbeitstisch, auf dem auch der Roboter befestigt ist. Optional kann auch eine Kiste oder ein anderer Behälter verwendet werden, in dem die Objekte auf dem Tisch liegen. Zu greifen sind jedoch nur die Objekte selbst, der Tisch oder ein eventueller Behälter sollen bei der Greifposengenerierung ausgeschlossen werden. Des Weiteren müssen die Objekte „hart“ genug sein, damit sie von einem Greifer ohne abzurutschen greifbar sind und mindestens einen geeigneten Angreifpunkt für ein zuverlässiges Greifen durch den Roboter besitzen.

2.1.3 *Physikalische und technologische Grenzen*

Die Tiefenkamera erfasst einen Bereich von 0,25 bis 9 Metern. Die Objekte müssen also innerhalb dieser Begrenzungen liegen (dies ist auf dem Arbeitstisch immer gegeben). Die Traglast des Panda Roboters beträgt 3 kg, die Objekte dürfen also nicht schwerer sein als dieses Limit, da der Roboter sie andernfalls nicht anheben kann. Außerdem müssen sich die Objekte im Arbeitsbereich des Roboters befinden, so dass dieser sie erreichen kann. Abbildung 2.1 zeigt den vom Roboter erreichbaren Bereich. [1] [2]

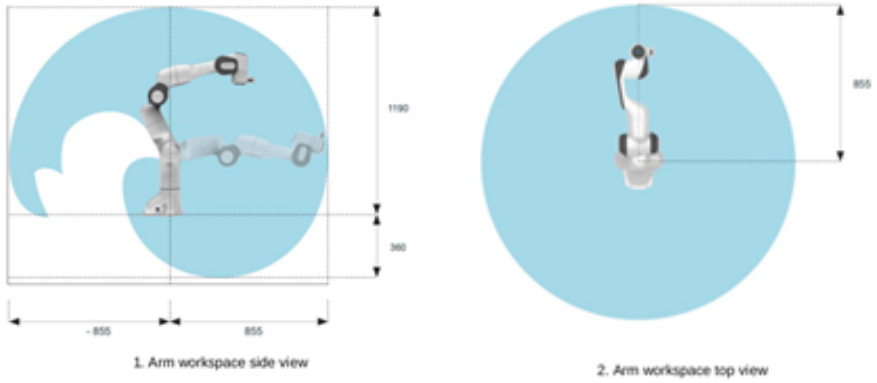


Abbildung 2.1: Franka Panda - Workspace

2.2 RECHERCHE DER LÖSUNGSMÖGLICHKEITEN (AP 2)

In diesem Abschnitt werden die bei der Recherche gefundenen Lösungsmöglichkeiten der einzelnen Teilprobleme dargestellt.

2.2.1 *Ansteuerung des Panda Roboters*

Franka Emika bietet das so genannte Franka Control Interface (FCI) zur Steuerung des Panda Roboters. Dieses unterstützt zwei Arten, auf die mit dem Roboter kommuniziert werden kann:

Die **libfranka** ist die in C++ implementierte Client-Seite des FCI. Sie bietet Schnittstellen für mehrere Funktionen:

- die Ausführung von Nicht-Echtzeit-Befehlen
- die Ausführung von Echtzeit-Befehlen (1 kHz Kontrollschleife)
- das Auslesen des Roboterzustands für Sensordaten
- den Zugriff auf die Modell-Bibliothek, um Kinematiken und Dynamiken zu berechnen

Das ROS Metapaket **franka_ros** integriert die **libfranka** in das Robot Operating System. Es gruppiert mehrere ROS Pakete, die essenzielle Funktionen für die Interaktion mit dem Roboter bieten. Dazu zählen zum Beispiel die Hardwareschnittstellen, Visualisierungsmöglichkeiten und die Roboterbeschreibung.

Für die Bahnplanung und das Ansteuern der Robotercontroller ist außerdem eine MoveIt Konfiguration bereitgestellt. Damit kann sowohl über das Visualisierungstool Rviz oder die C++/Python APIs die Bahnplanung parametriert und aufgerufen werden. [3]

2.2.2 *Auslesen der Intel Realsense Tiefenkamera*

Analog zum Ansteuern des Panda Roboters bestehen für die Interaktion mit der Tiefenkamera zwei Ansteuerungsmöglichkeiten:

Die Realsense SDK **librealsense** unterstützt verschiedenste Programmiersprachen (darunter auch C++ und Python). Mit ihrer Hilfe kann die Kamera konfiguriert und ihre Aufnahmen ausgelesen werden. [4]

Mit dem ROS Metapaket **realsense_ros** können die Funktionen der **librealsense** aus dem ROS Ökosystem heraus aufgerufen werden. Insgesamt beinhaltet es zwei ROS Pakete, eines mit der Kamerabeschreibung und ein anderes mit den ROS Knoten und Scripts für die Ansteuerung der Kamera. Die Knoten werden über ROS Services und Topics angesprochen. [5]

2.2.3 *Punktewolkenfusion*

Für die Verarbeitung von den aus der Kamera ausgelesenen Punktewolken bietet sich die Point Cloud Library (PCL) an. Die PCL unterstützt die Programmiersprachen C++ und Python und ist gut in ROS integriert. Die Bibliothek ermöglicht viele komplexe Punktewolkenoperationen durch simple Funktionsaufrufe. Dazu zählen zum Beispiel das Zusammenfügen mehrerer Punktewolken, sowie das Ausdünnen oder das Transformieren von Punktewolken. [6]

Für die eigentliche Fusion können mehrere Punktewolken aus verschiedenen Winkeln der Kamera aufgenommen und in ein gemeinsames Koordinatensystem transformiert werden. Danach können die Punktewolken zusammengefügt und redundante Informationen entfernt werden.

Die Punktewolken der PCL können auf ein ROS Topic gepublished werden und sind dann z.B. in **Rviz** visualisierbar.

2.2.4 *Robotersimulation*

Es existieren mehrere Simulationsprogramme für Roboter. Der für ROS am häufigsten genutzte und am besten dokumentierte Simulator ist Gazebo. Gazebo bietet eine robuste Physik-Engine, Grafik mit hoher Qualität und eine intuitive grafische Nutzerschnittstelle. Außerdem können in ROS im Unified Robot Description Format (URDF) definierte Robotermodelle und deren Umgebung in die Simulation geladen werden. Dazu gehören auch Möglichkeiten zur individuellen Modellierung von Sensoren (auch Tiefenkameras). Ein weiterer Vorteil ist die direkte Integration in ein ROS System durch Kommunikationsmöglichkeiten mittels ROS Services und Topics. [7]

Von der ROS Community erstellte Roboterbeschreibungen des Panda mit den für die Simulation nötigen Parametern (z.B. Masse und Trägheitsmomente) sind online verfügbar. Ein bekanntes Paket ist der **panda_simulator** [8]. Er bietet Low-level Controller, um den simulierten Roboter per Topics zu steuern und published den aktuellen Roboterzustand in

der Simulation per ROS Topics. Zusätzlich beinhaltet es einige weitere Scripts mit unterschiedlichen Steuerungsmöglichkeiten des simulierten Roboters.

Alternativ zu Gazebo existieren weitere Simulatoren wie z.B. CoppeliaSim, die eventuell mehr Simulationsfeatures als Gazebo bieten, jedoch deutlich komplizierter in ROS zu integrieren sind.

2.2.5 *Greifposengenerierung*

Mögliche Greifposen für den Roboter können grundsätzlich auf verschiedene Arten und Weisen ermittelt werden. In diesem Projekt wurde der Fokus auf die Detektion mit Hilfe der von der Tiefenkamera aufgenommenen Punktewolken gelegt.

In vielen Fällen ist der Einsatz einer künstlichen Intelligenz bei diesem Vorhaben von Vorteil. In seinem Paper „Grasp Pose Detection in Point Clouds“ stellt Andreas Ten Pas in diesem Zusammenhand die Grasp Pose Detection Bibliothek vor [9]. Diese Bibliothek basiert auf einem Convulutional Neural Network (CNN) für die Verarbeitung der Punktewolken.

Der Algorithmus der Posengenerierung sieht dabei grob folgenermaßen aus:

Input: Punktewolke, Region von Interesse, Greiferbeschreibung

- Punktewolke vorverarbeiten
- Region von Interesse extrahieren
- viele mögliche Kandidaten sampeln
- Kandidaten klassifizieren
- gültige Kandidaten wählen

Output: Gültige, klassifizierte Greifposenkandidaten

Das verwendete CNN ist bereits mit dem BigBird-Datensatz [10] vortrainiert. Da die Greifposenerkennung unabhängig von den Objekten erfolgt, lassen sich die Ergebnisse einfach auf neue Objekte übertragen.

Die Bibliothek bietet ebenfalls zwei Möglichkeiten für die Verwendung:

Die Grasp Pose Detection C++-Bibliothek **gpd** ermöglicht die Detektion von Greifposen mit 6 Freiheitsgraden (Translation und Rotation). Sie erhält Punktewolken als Input und generiert mögliche Greifposen als Output. Des Weiteren ist sie mit der PCL kompatibel.

gpd_ros ist ein ROS Wrapper für die gpd-Bibliothek. Er ermöglicht die direkte Verarbeitung von auf einem ROS Topic gepublishten Punktewolken. Die generierten Greifposen werden anschließend auf einem anderen Topic gepublished. [11]

2.3 INBETRIEBNAHME DER TIEFENKAMERA (AP 3)

Die Installation der für die Tiefenkamera notwendige Software fand nach der von Intel bereitgestellten Anleitung statt. Dabei wurde sowohl die SDK **librealsense** als auch der ROS Wrapper **realsense_ros** installiert.

Der Realsense-Viewer ist ein graphisches Tool von Intel, mit dem die Kamera einfach angesteuert und ihre Tiefen-, Infrarot- und RGB-Bilder (in 2D und 3D) angezeigt werden können. Zudem bietet es die Möglichkeit ein Firmwareupgrade der Kamera auszuführen. Das Tool wird standardmäßig bei der Installation der Realsense SDK mitinstalliert. Abbildung 2.2 zeigt erste Tests mit der realen Kamera im Realsense-Viewer. [4]

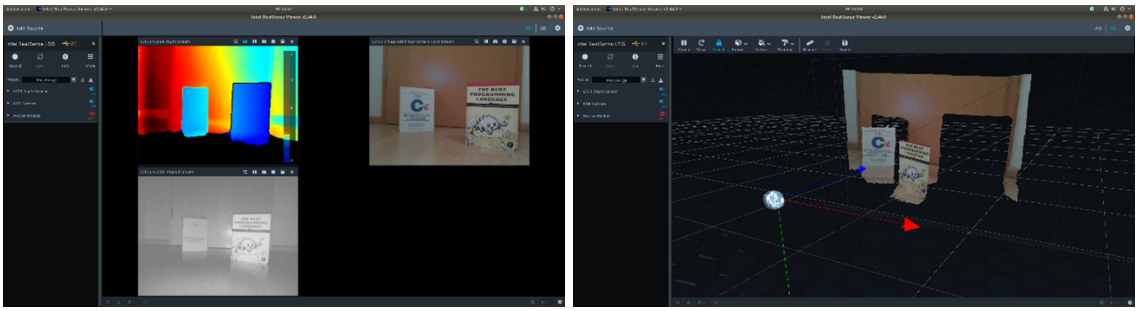


Abbildung 2.2: Franka Panda - Workspace

Das **realsense_ros**-Paket beinhaltet ROS Knoten, mit deren Hilfe die Bild- und Tiefendaten der Kamera ausgelesen und auf einem Topic gepublished werden können. Um den Knoten mit passenden Parametern aufrufen zu können, kann die ebenfalls im Paket enthaltene Launchdatei rs_camera.launch verwendet werden. Dabei ist zu beachten, dass der Knoten nur dann Punktewolken published, wenn der Filter „pointcloud“ im Launchbefehl angegeben wird: *roslaunch realsense2_camera rs_camera.launch filters:=pointcloud*. [5]

Die so auf dem Topic veröffentlichten Punktewolken können in der Visualisierungsumgebung Rviz in Echtzeit dargestellt werden.

Es ist es möglich, die Kamera mit Hilfe von Kommandozeilenargumenten beim Aufruf der Launchdatei zu parametrieren. Für eine ausführlichere Konfiguration kann zunächst im Realsense-Viewer eine Konfiguration erstellt und als JSON exportiert werden. Die JSON-Datei kann der Launchdatei über den Parameter *json_file_path* übergeben werden.

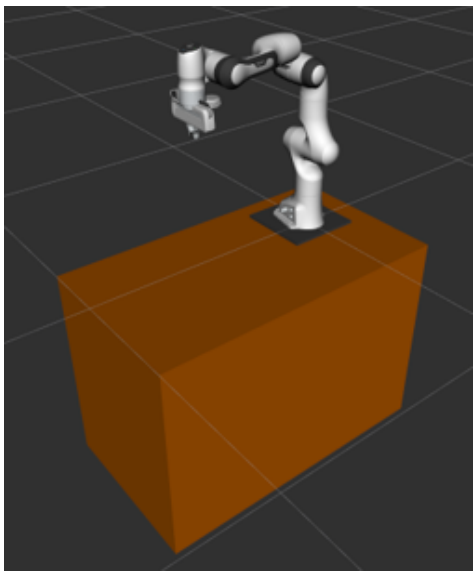
2.4 SIMULATION DES ROBOTERS MIT TIEFENKAMERA (AP 4)

Die Wahl der Simulationsumgebung fiel aufgrund der guten Kompatibilität mit ROS auf Gazebo. Bei ersten Tests mit der Simulationsumgebung wurde das Paket **panda_simulator** von Saif Sidhik verwendet [8]. Es beinhaltet eine für die Simulation angepasste Roboterbeschreibung des Panda Roboters und einige zusätzliche Python-Scripts, mit denen der simulierte Roboter über ROS hinaus auf andere Weisen gesteuert werden kann.

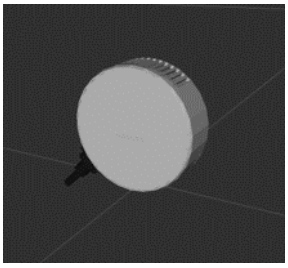
Da einige der dabei automatisch aufgerufenen Scripts jedoch regelmäßig abstürzten und der Start der Simulation daher nicht zuverlässig war, wurde der **panda_simulator** durch eine andere, leichtgewichtigere Roboterbeschreibung für die Simulation ausgetauscht. Sie

ist in dem ROS Paket **franka_ros** von Tahsincan Köse enthalten und besitzt keine zusätzlichen Scripts, bietet allerdings alle wichtigen Features für die Nutzung mit ROS und MoveIt [12].

Die Roboterbeschreibung wurde zusätzlich um die reale Umgebung des Roboters erweitert. Der Tisch wurde als ein Quader mit den originalen Maßen modelliert. Die geometrische Beschreibung der Tiefenkamera stammt aus dem **realsense_ros**-Paket (s. Abbildung 2.3b) [5]. Abbildung 2.3a zeigt die vollständige Roboterbeschreibung mit allen Erweiterungen in der Visualisierungsumgebung Rviz.



(a) Vollständige Roboterbeschreibung



(b) Geom. Beschreibung der Tiefenkamera

Abbildung 2.3: Rviz Visualisierungen

Des Weiteren wurde das Gazebo-Tiefenkamera-Plugin (*libgazebo_ros_openni_kinect*) eingebunden und so konfiguriert, dass es die reale Tiefenkamera so gut wie möglich wiederspiegelt. Dabei ist zu beachten, dass die in Realität verwendete Kamera mit dem Light Detection and Ranging (LiDAR) Verfahren arbeitet. Das verwendete Gazebo-Plugin spiegelt jedoch ein anderes bei vielen Tiefenkameras verwendetes Verfahren wider, weshalb die simulierte Kamera nicht 1 zu 1 an die Realität anpassbar ist. Ein Gazebo-Plugin für das LiDAR Verfahren existiert aktuell nicht öffentlich und müsste zunächst selbst implementiert werden.

Weitere Objekte, z.B. solche, die vom Roboter gegriffen werden sollen, können während der Simulation dynamisch in die Welt eingefügt werden. Dies kann entweder direkt im Gazebo-GUI oder über eine ROS Launchdatei geschehen. Alternativ kann eine Launchdatei erstellt werden, die sowohl Gazebo startet als auch Objekte in die virtuelle Welt lädt. Auf diese Art sind die gewünschten Objekte direkt beim Start des Simulators in der Simulation vorhanden.

Der simulierte Roboter kann analog zum realen Roboter über das Move-Group-Interface von MoveIt gesteuert werden. Zu beachten war dabei, dass der Roboter mit einer Greiferbewegung unter Nutzung der standardmäßig verwendeten *position_controllers/JointTrajectory Controller* die Objekte nicht mit einer bestimmten Kraft greifen konnte und die Objekte da-

her meist abrutschten. Stattdessen wurden *effort_controllers/JointTrajectoryController* eingesetzt und die Gains in der Konfigurationsdatei „ros_controllers.yaml“ angepasst, um das Greifen der Objekte zu ermöglichen. Zusätzlich verbesserte ein Anpassen der Reibungseigenschaften der Objekt-Modelle die Greiffähigkeit.

Abbildung 2.4 zeigt die graphische Nutzerschnittstelle von Gazebo mit eingefügtem Roboter und drei beispielhaften Objekten. Alternativ ist es möglich, Gazebo headless, also ohne Grafik, zu starten. Dies kann vor allem auf schwachen Systemen vorteilhaft sein, da das Gazebo-GUI die Grafikkarte sehr stark beansprucht.

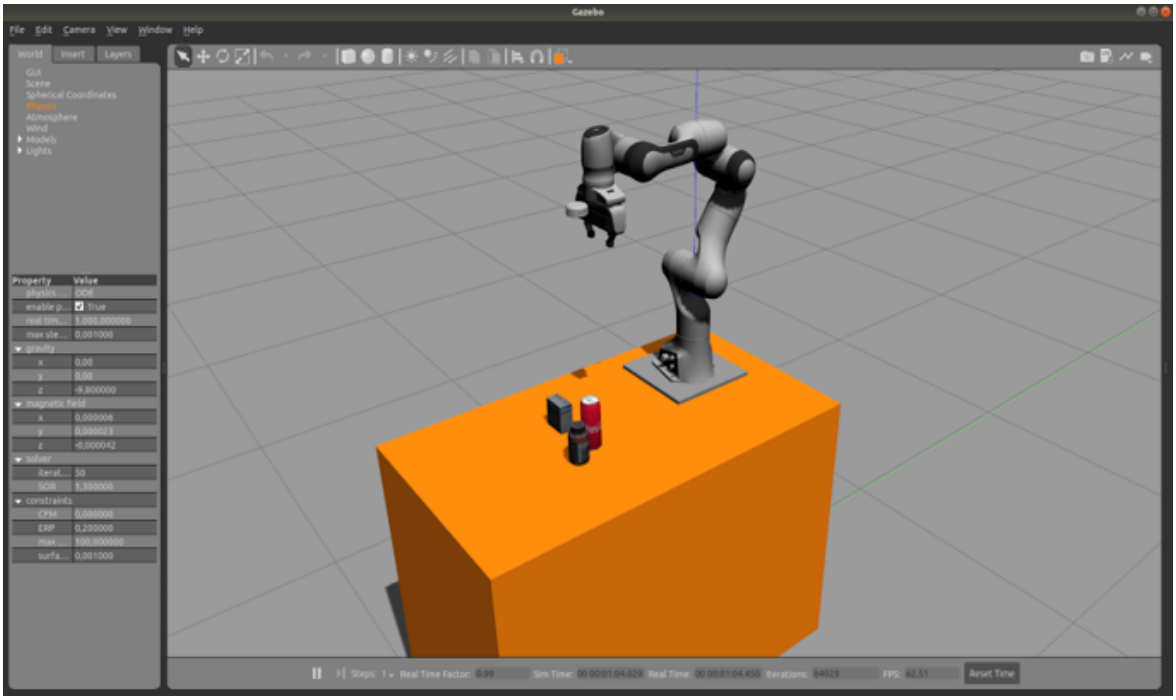


Abbildung 2.4: Simulationsumgebung Gazebo

2.5 FUSION VON PUNKTEWOLKEN (AP 5)

Um den Arbeitsbereich so gut und vollständig wie möglich zu erfassen, sollte das Aufnehmen mehrer Punktewolken aus unterschiedlichen Kameraposen mit einer anschließenden Fusion zu einer Gesamtwolke ermöglicht werden.

Dazu wurde eine zu Beginn jedes Zyklus aufgerufene Detektionsroutine erstellt. In ihr wird der Roboter per MoveIt an verschiedene Posen überhalb des Arbeitsbereiches bewegt. An jeder Pose macht der Roboter halt und die aktuelle Punktewolke der Tiefenkamera wird zuerst mit Hilfe des ROS Pakets **tf** in das Weltkoordinatensystem transformiert und anschließend gespeichert. **tf** kennt die Transformation der Kamera zum Weltkoordinatensystem aus der Roboterbeschreibung im URDF. Abschließend werden alle aufgenommenen Punktewolken unter Nutzung der **PCL** zusammengeführt und, falls nötig, ausgedünnt oder Ausreißer entfernt.

Abbildung 2.5 zeigt eine Gesamtwolke, die aus drei aus verschiedenen Posen (Translation und Rotation) in der Simulation aufgenommenen Punktewolken zusammengesetzt wurde.

Es ist zu erkennen, dass die Zusammenführung sehr genau ist und die Objekte sich in den Punktewolken gegenseitig exakt überlagern.

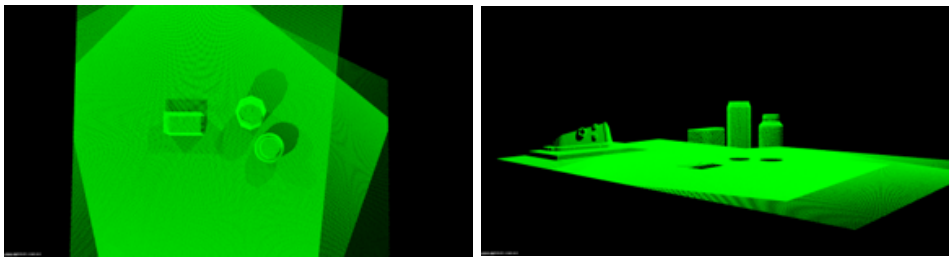


Abbildung 2.5: Fusionierte Punktewolke (Simulation)

Die resultierende Gesamtwolke dient in späteren Schritten einerseits als Eingabe für die Greifposendetektion und andererseits wird sie an die MoveIt-Octomap übergeben, sodass die aufgenommenen Objekte bei der kollisionsfreien Bahnplanung durch Moveit beachtet werden können.

2.6 OBJEKT- UND GREIFPOSENERKENNUNG (AP 6)

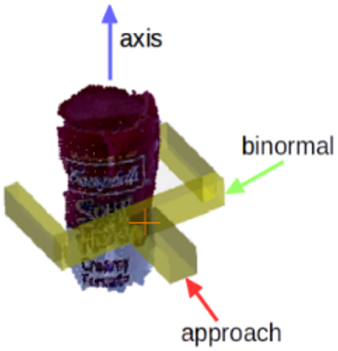
Für die Greifposenerkennung wurde die in 2.2.5 vorgestellte, ein CNN nutzende Open-Source Bibliothek **gpd** in dem ROS Wrapper **gpd_ros** eingesetzt. [11]

Für die optimale Nutzung der Bibliothek mussten zunächst die Einstellungen in einer Konfigurationsdatei „gpd_params.cfg“ den eigenen Anforderungen entsprechend angepasst werden. Konfigurierbar ist dabei einerseits die Greifergeometrie des realen Greifers. Darüber hinaus können Parameter der Vorverarbeitung der Punktewolke oder Parameter des neuronalen Netzes angepasst werden.

Für die eigentlich Greifposengenerierung wird zunächst der Detektionsknoten „detect_grasps“ aus dem **gpd_ros**-Paket gestartet. Diesem wird der Name des Punktewolkentopics, auf dem die zu verarbeiteten Punktewolken veröffentlicht werden, und der Pfad zur angepassten Konfigurationsdatei als Parameter angegeben.

Um Greifposen zu generieren, wird die fusionierte Gesamtwolke auf dem spezifizierten Punktewolkentopic gepublished, woraufhin die Bibliothek diese verarbeitet und mögliche Greifposen klassifiziert. Die generierten Greifposen werden von der Bibliothek anschließend auf einem anderen Topic veröffentlicht und können dort ausgelesen werden.

Zu beachten ist, dass die so generierten Greifposen in dem in Abbildung 2.6a dargestellten Format vorliegen. Dort zeigt die X-Achse der Orientierung auf das Objekt, der Panda-Greifer benötigt jedoch zum Greifen eine Orientierung, bei der die Z-Achse auf das Objekt zeigt. Außerdem befindet sich die generierte Translation im „Ballen“ der Roboterhand, MoveIt benötigt allerdings die Position des Roboterflansches, also des Greiferursprungs. Daher ist eine Transformation des in Abbildung 2.6b als GPD gekennzeichneten Korrdinatensystems in das MoveIt-System nötig.



(a) Konvention in **gpd**

(b) Tranformation von **gpd** zu MoveIt

Abbildung 2.6: Greifposenbeschreibung

2.7 GESAMT-SOFTWAREARCHITEKTUR (AP 7)

Für das Erstellen einer beispielhaften Bin-Picking-Applikation wurde zunächst ein Softwarekonzept entwickelt, sodass alle für die Applikation nötigen Funktionalitäten abgedeckt sind. Die Software wurde dabei, so wie in ROS üblich, grundlegend in mehrere Pakete gegliedert, welche bestimmte Aufgaben logisch voneinander trennen. Abbildung 2.7 stellt die in diesem Zuge neu implementierten Pakete in Grün markiert dar. Bei den in Blau markierten Paketen handelt es sich um die wichtigsten externen, von anderen Autoren bereits erstellten Open-Source-Pakete, die für das Projekt ebenfalls eingesetzt wurden.

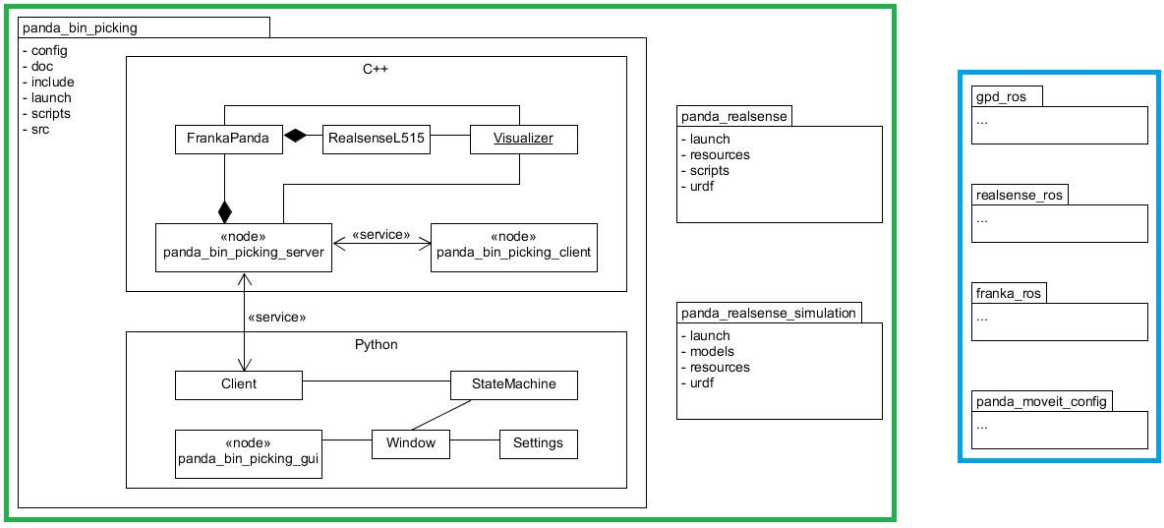


Abbildung 2.7: ROS Paketstruktur

Die erstellten Pakete **panda_realsense** und **panda_realsense_simulation** beinhalten fast keinen Programmcode, sondern stellen lediglich Ressourcen für die Interaktion mit dem Roboter und der Tiefenkamera bereit.

Das **panda_realsense_simulation**-Paket ist Grundlage für die Nutzung der Applikation in der Simulation. Es besitzt Launchdateien zum Starten von Gazebo mit dem vollständigen für die Simulation angepassten Robotermodell und der Tiefenkamera mit dem dafür konfigurierten Gazebo-Plugin. Das mit Kamera und Tisch vervollständigte Robotermodell ist als URDF im Paket enthalten und basiert auf der Roboterbeschreibung aus dem **franka_description**-Paket, das sich in dem Metapaket **franka_ros** befindet. Des Weiteren beinhaltet das **panda_realsense_simulation**-Paket einige beispielhafte 3D-Modelle von möglichen zu greifenden Objekten, die nach Bedarf in die Simulation geladen werden können.

Das Paket **panda_realsense** hat die gleiche Aufgabe wie das **panda_realsense_simulation**-Paket mit dem Unterschied, dass es für den realen Einsatz und nicht zu Simulationszwecken dient. Daher beinhaltet es auch keine 3D-Modelle, stattdessen aber einige in Python geschriebene Scripts mit für den realen Einsatz nützlichen Funktionen (z.B. Kamerakalibrierung). Ein angepasstes Robotermodell im URDF und Launchdateien für den Kommunikationsaufbau mit der Kamera und dem Roborer in der Realität sind ebenfalls enthalten.

Bei dem letzten neu erstellten Paket handelt es sich um das **panda_bin_picking**-Paket. In diesem befindet sich der gesamte Quellcode für die eigentliche Ausführung des Bin-Picking-Prozesses. Die Ansteuerung der Hardware wurde dabei in einem in C++ geschriebenen Knoten implementiert. Dieser *panda_bin_picking_server*-Knoten dient als Server und bietet die für das Bin Picking benötigten Teilfunktionalitäten als ROS Services an, die von Clients aufgerufen werden können. Intern verwendet der Knoten zwei Klassen, die jeweils eine Abstraktion der Hardware darstellen. Die Klasse *RealsenseL515* bildet die verwendete Tiefenkamera ab und stellt Methoden zum Auslesen der Punktewolken von dem Topic, auf dem die Kamera published, und zur Verarbeitung dieser Punktewolken bereit. Die *FrankaPanda*-Klasse abstrahiert den Roboter und besitzt eine Instanz der Tiefenkamera. Sie bietet öffentliche Methoden für die übergeordneten Teilschritte des Bin-Picking (Aufnahme von Punktewolken, Greifposendetektion, Bewegung zur Greifpose, usw.) sowie private Methoden zum Planen und Ausführen von Roboterbewegungen unter Nutzung des MoveIt-Frameworks. Mit Hilfe der statischen Klasse *Visualizer* können Greifposen und Roboterbahnen in Rviz dynamisch visualisiert werden. Des Weiteren wurde der Knoten *panda_bin_picking_client* angelegt, der die Services des Servers aufruft und so die Steuerung der Bin-Picking-Applikation über die Kommandozeile ermöglicht.

Zusätzlich wurde eine Graphische Nutzerschnittstelle (GUI) in Form des Python-Knotens *panda_bin_picking_gui* implementiert. Dieser nutzt mehrere zusätzlich erstellte Klassen, wobei die Ablaufsteuerung des Prozesses über einen Zustandsautomaten implementiert wurde. Über die *Client*-Klasse werden aus dem GUI heraus die vom *panda_bin_picking_server*-Knoten angebotenen Services aufgerufen. Abbildung 2.8 zeigt das erstellte GUI.

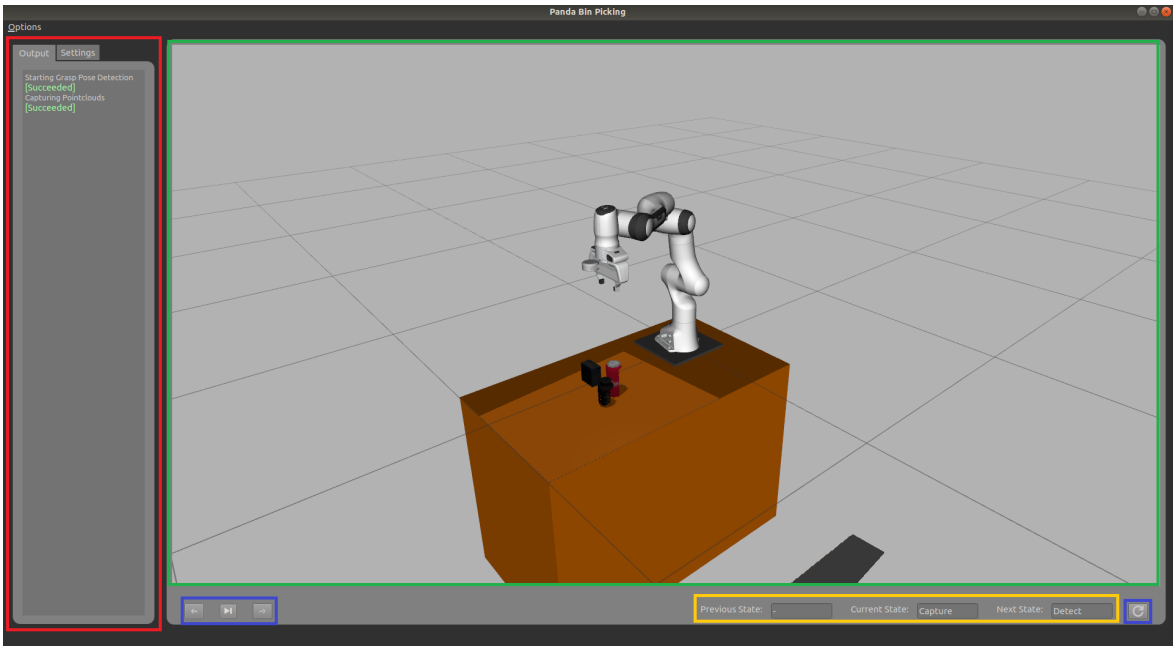


Abbildung 2.8: GUI

Auf der linken Seite (rot) befindet sich eine Debugausgabe über den aktuellen Prozessschritt. Alternativ können dort in einem anderen Tab die Einstellungen der Greifposenerkennung angepasst werden. Im Zentrum der Applikation befindet sich die Live-Visualisierung (grün). Bei dieser handelt es sich um eine in das GUI eingebettete Rviz-Visualisierung, in der der aktuelle Prozesszustand graphisch dargestellt wird. Mit den in Blau markierten Aktionsknöpfen kann der Zustandsautomat und somit generell der Prozess gesteuert werden, wobei der Autoplay-Button noch nicht funktionsfähig ist. Informationen über den aktuellen sowie Folge-/Vorzustand sind ebenfalls enthalten (gelb).

2.8 IMPLEMENTIERUNG DES PROTOTYPS (AP 8)

Da in der Simulation auch die realen von MoveIt anzusteuernden Robotercontroller simuliert werden, mussten bei der Übertragung der Bin-Picking-Applikation auf den realen Roboter nur geringfügige Anpassungen am Code in der *FrankaPanda*-Klasse vorgenommen werden.

2.8.1 *Objekte greifen*

Vorallem das Greifen der Objekte unterscheidet sich in Simulation und Realität. Für die Simulation wurden wie in 2.4 erläutert die *effort_controllers/JointTrajectory* anstelle der *position_controllers/JointTrajectory* verwendet, sodass der Roboter mit einer normalen durch MoveIt geplanten Greiferbewegung das Objekt mit einer festgelegten Kraft packen kann. Die Umstellung der Controller wäre für den realen Roboter evtl. ebenfalls möglich, jedoch bietet Franka eine bessere Alternative. Dabei handelt es sich um das Paket **franka_gripper** aus dem **franka_ros**-Metapaket. Dieses beinhaltet einen Knoten, der die sogenannte *Gras-*

pAction anbietet, welche in der Simulation nicht unterstützt wird. Diese Aktion kann mit einer beliebigen Kraft, Geschwindigkeit, Objektbreite und Toleranz parametriert und aufgerufen werden. Anschließend greift der Roboter das Objekt unter Berücksichtigung dieser Einstellungen. [3]

Da die Klasse *RealsenseL515* lediglich Punktewolken auf einem Topic empfängt und daher unabhängig von der konkreten Kamera ist, mussten hier keine Codeanpassungen vorgenommen werden.

2.8.2 *Kameraposenkalibrierung*

Die größte Herausforderung bei der Übertragung auf den realen Prototyp stellte die Ermittlung der Transformation vom Roboterflansch zur Tiefenkamera dar. Diese ist extrem wichtig, da sie für die optimale Transformation der aufgenommenen Punktewolken in das Weltsystem so exakt wie möglich sein sollte. Schon leichte Abweichungen dieser Transformation können zu enormen Abweichungen der Punktewolken von der Realität führen.

Da das Ausmessen der Kamerapose per Hand mit Hilfe eines Zollstocks sehr ungenaue Ergebnisse lieferte, wurde ein Verbesserungsversuch durch eine Selbstkalibrierung der Kamera durchgeführt. Grundlage dafür stellen das Computer-Vision-Verfahren „Pose-Estimation“ [13] sowie die anhand der realen Maße des Tisches ausgerechnete Position eines zur Kalibrierung verwendeten Schachbrettmusters in Bezug auf die Roboterbasis dar. Damit sind alle für die Berechnung der Transformation vom Roboterflansch zur Kamera nötigen Transformationen vorhanden, da die Transformation von der Roboterbasis zum Flansch von ROS aus der URDF-Roboterbeschreibung extrahiert werden kann. Abbildung 2.9 zeigt alle für die Berechnung nötigen Transformationen.

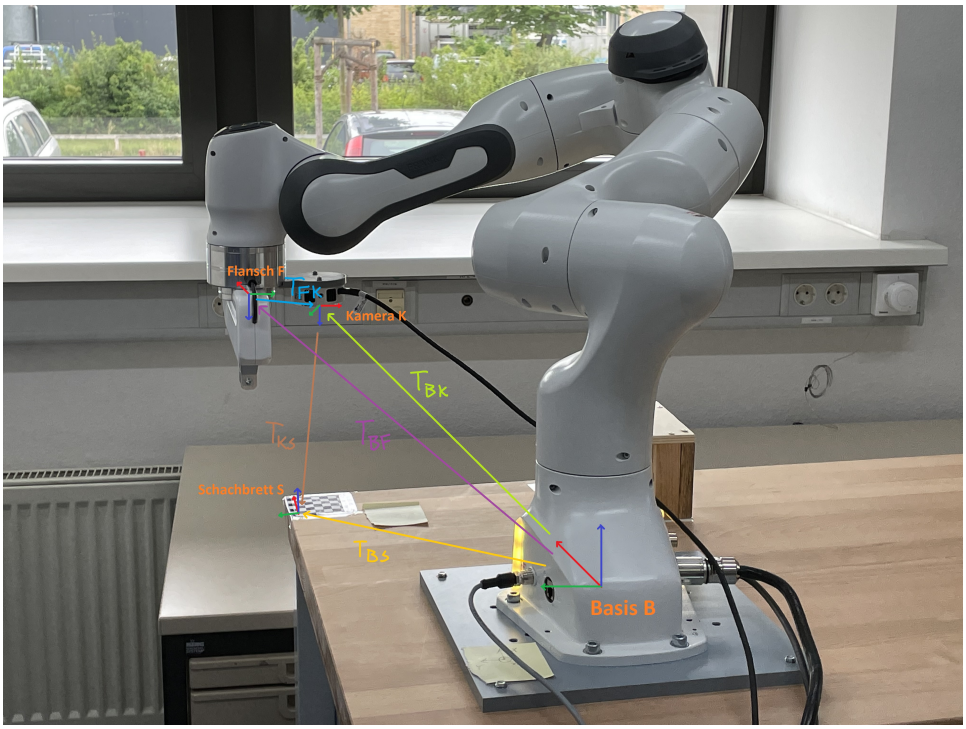


Abbildung 2.9: Kameraposenkalibrierung: Transformationen

Die gesuchte Transformation T_{FK} vom Flansch F zur Kamera K kann ultimativ durch eine Transformation vom Flansch zur Basis B (also die inverse Transformation von T_{BF}) und der anschließenden Transformation T_{BK} von der Basis zur Kamera berechnet werden. Die Transformation T_{BK} wird zunächst aus den Transformationen T_{BS} von der Basis zum Schachbrettmuster S und der inversen der Transformation T_{KS} von der Kamera zum Schachbrettmuster zusammengesetzt. Daraus ergeben sich folgende Formeln:

$$T_{BK} = T_{BS} * T_{KS}^{-1}$$
$$T_{FK} = T_{BF}^{-1} * T_{BK}$$

T_{BF} wird von ROS berechnet, T_{BS} wird anhand der Tischmaße händisch berechnet und T_{KS} ist das Resultat der mit Hilfe der Bibliothek OpenCV durchgeführten Pose-Estimation. In Abbildung 2.10 ist der Vorgang der Pose-Estimation im 2D-Kamerabild zu sehen. Im unteren linken Bereich des Schachbrettes befindet sich das Koordinatensystem der geschätzten Schachbrettpose.

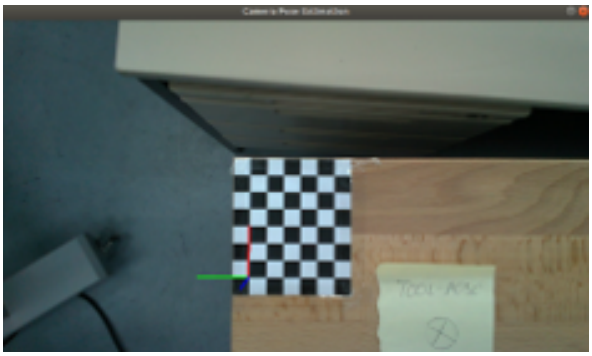


Abbildung 2.10: OpenCV: Pose-Estimation

Die durch die Kalibrierung erhaltene Transformation T_{FK} hat bereits deutliche Verbesserungen im Vergleich zur komplett per Hand ausgemessenen Transformation ergeben. Dennoch sind die Ergebnisse noch nicht optimal, da nun die Schachbrettposition per Hand gemessen ist und daher nicht genau der Realität entspricht.

2.8.3 *Aufnahme und Fusion von Punktewolken*

Das letzte Problem, das in der Simulation nicht auftrat, jedoch am realen Roboter ungenaue Resultate produzierte, war die Aufnahme von Punktewolken aus unterschiedlichen Posen. Vor allem die Orientierung spielte dabei eine große Rolle. Bei einer Aufnahme eines Objektes, bei der die Kamera nicht direkt auf das Objekt gerichtet ist, ist die resultierende Punktewolke stark verzerrt. Dadurch sind die Wolken einerseits sehr ungenau, andererseits ist die Gesamtwolke nach der Fusion inkonsistent und die Objekte an unterschiedlichen Stellen in der Punktewolke vorhanden (s. Abbildung 2.11).

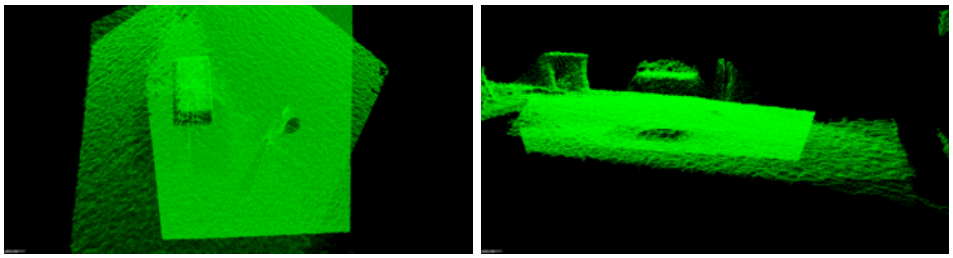


Abbildung 2.11: Fusionierte Punktewolke (Realität, verschiedene Aufnahmewinkel)

Als Lösungsansatz wurden nur noch Punktewolken der Objekte senkrecht von oben nach unten aufgenommen und auch nur kleinere Ausschnitte pro Punktewolke betrachtet. Der Nachteil ist, dass die Wolken so nicht alle Informationen der Objekte beinhaltet. Dennoch wurde damit das grundlegende Problem der verzerrten Objekte vorerst behoben. Eine mit dieser Methode aufgezeichnete, fusionierte Punktewolke ist in Abbildung 2.12 zu sehen. Dabei lässt sich erkennen, dass die Wolken immer noch nicht optimal übereinanderliegen, was auf die nicht exakte Kameraposenkalibrierung zurückzuführen ist.

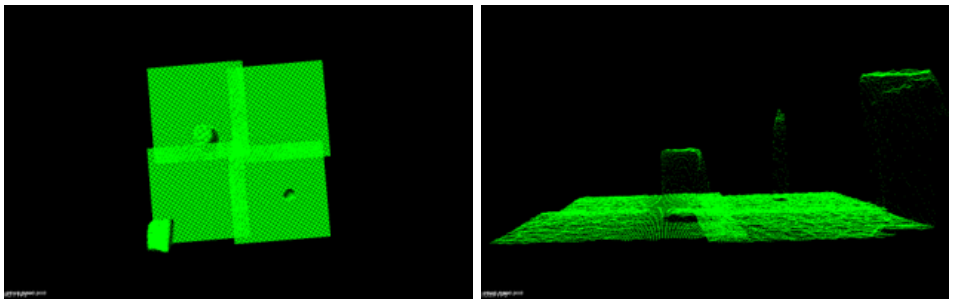


Abbildung 2.12: Fusionierte Punktewolke (Realität, gleicher Aufnahmewinkel)

3

ERGEBNISSE

3.1 SIMULATION

Die Bin-Picking-Applikation läuft in der Simulation stabil und zuverlässig. Das in Gazebo verwendete Robotermodell entspricht exakt dem realen Roboter und kann direkt über MoveIt gesteuert werden. Die simulierte Tiefenkamera basiert nicht genau auf der gleichen Technologie wie die reale Kamera, ist jedoch so originalgetreu wie möglich nach ihren Eigenschaften konfiguriert und liefert gute Punktewolken. Die erhaltenen Puntkewolken lassen sich problemlos fusionieren und erzeugen nach der Auswertung durch die Greifposendetektion sehr gute Greifposen. Der gesamte Prozess läuft ohne Abstürze und terminiert, sobald keine zu greifenden Objekte mehr vorhanden sind. Der Prozess kann sowohl über die Kommandozeile als auch per GUI gesteuert werden.

3.2 REALITÄT

Am realen Prototyp läuft der Bin-Picking-Prozess weitestgehend stabil und ohne Abstürze. In seltenen Fällen stürzt der Tiefenkameraknoten aus dem **realsense_ros**-Paket ab, was nur durch einen Neustart der gesamten Applikation behoben werden kann. Die Detektion der Greifposen ist nicht optimal, was an den ungenauen Punktewolken aus der Tiefenkamera liegt. Hauptgrund für die Ungenauigkeit ist die noch nicht vollständig ausgereifte Kameraposenkalibrierung. Das Greifen der Objekte ist, wenn valide Greifposen generiert wurden, durch den Einsatz der *GraspAction* problemlos möglich, wobei aktuell die Greifkraft für jedes Objekt identisch ist und sich nicht dynamisch ändern lässt. Auch hier kann die Applikation sowohl über die Kommandozeile als auch per GUI gesteuert werden, wobei im User-Interface sogar der aktuelle Zustand des realen Roboters virtuell visualisiert wird.

3.3 VERGLEICH DER ANGESTREBTEN UND ERREICHTEN RESULTATE

Die grundlegend angestrebten Ziele wurden erreicht. Der Roboter ist in der Simulation und Realität in der Lage, Punktewolken aufzuzeichnen, zu fusionieren, Greifposen zu detektieren und damit Objekte zu greifen. Außerdem ist die Steuerung des Prozesses durch das GUI intuitiv möglich.

Der Behälter, der dem Bin-Picking seinen Namen gibt, ist bislang nicht vorgesehen. Das heißt, es können aktuell nur auf dem Tisch verstreute Objekte gegriffen werden, die sich in keinem Behälter befinden. Die Auswertung der Punktewolken ist am realen Roboter leider noch etwas unzuverlässig und entspricht dort noch nicht ganz den angestrebten Zielen. Im GUI sind nicht alle vorgesehenen Funktionen implementiert, jedoch ist bereits die Möglichkeit zum Steuern des gesamten Prozesses auf mindestens eine Art gegeben.

4

FAZIT UND AUSBLICK

4.1 FAZIT

Die grundlegend angestrebten Anforderungen an die Simulation und den realen Roboter wurden erfüllt. Es wurde eine funktionsfähige Bin-Picking-Anwendung implementiert, die alle Technologien, die es in diesen F&E-Studien zu untersuchen galt, kombiniert. Während des Projektes stellte sich dabei heraus, dass der Umgang mit der realen Tiefenkamera mehr Aufwand erfordert als das in Gazebo simulierte Gegenstück. Grund dafür war hauptsächlich das Problem der Kameraposenkalibrierung, die durch ihre momentane Ungenauigkeit noch für nicht optimale Ergebnisse sorgt. Generell läuft der gesamte Prozess in der Simulation reibungsloser als am realen Prototyp.

4.2 AUSBLICK

Bislang fehlt der Bin-Picking-Anwendung der Bin. Dies lässt sich allerdings mit ein wenig zusätzlichem Aufwand ändern. Eine Möglichkeit, diese Funktion hinzuzufügen, wäre zum Beispiel die Segmentierung des Behälters in der Punktewolke mit der PCL, bevor diese an die Greifposendetektion geschickt wird.

Ein weiterer Punkt, der aktuell noch nicht beachtet wird, ist das Ablegen der gegriffenen Objekte. Auch hier könnten Erweiterungen vorgenommen werden, die den Gesamtprozess verbessern. Zusätzlich sind im GUI noch nicht alle Funktionen vollständig implementiert (z.B. Autoplay).

Das Hauptproblem der Applikation ist allerdings wie beschrieben die ungenaue Kameraposenkalibrierung. Um das Problem zu lösen, könnte eine externe Applikation entwickelt werden, die die Kalibrierung auf eine exaktere Art umsetzt. Die damit berechnete Transformation müsste dann lediglich in die URDF-Roboterbeschreibung eingefügt werden, um die Ergebnisse zu verbessern.

Bei einem zukünftigen Umstieg von ROS1 auf ROS2 könnte möglicherweise eine andere Grasp-Bibliothek weitere Vorteile bringen. Intel bietet zum Beispiel selbst das ROS2 Paket **ros2_grasp_library** an [14]. Dieses basiert auf dem Deep-Learning-Toolkit OpenVINO und bietet sogar Hilfsmittel für die Kameraposenkalibrierung.

LITERATUR

[1] FRANKA EMIKA. *Panda TECHNICAL DATA*. URL: https://www.generationrobots.com/media/panda-franka-emika-datasheet.pdf (besucht am 15. 07. 2021).

[2] Intel. *LiDAR Camera L515 Datasheet*. URL: https://dev.intelrealsense.com/docs/lidar-camera-l515-datasheet (besucht am 15. 07. 2021).

[3] FRANKA EMIKA. *Franka Control Interface*. URL: https://frankaemika.github.io/docs/ (besucht am 15. 07. 2021).

[4] Intel. *Intel® RealSense™ Documentation*. URL: https://dev.intelrealsense.com/docs?_ga=2.118304843.313673718.1626358985-246731359.1617748080 (besucht am 15. 07. 2021).

[5] Intel. *ROS Wrapper for Intel® RealSense™ Devices*. URL: https://github.com/IntelRealSense/realsense-ros (besucht am 15. 07. 2021).

[6] Point Cloud Library. *Getting Started*. URL: https://pointclouds.org/ (besucht am 15. 07. 2021).

[7] GAZEBO. *ROS overview*. URL: http://gazebosim.org/tutorials?tut=ros_overview (besucht am 15. 07. 2021).

[8] Saif Sidhik. *Panda Simulator*. URL: https://github.com/justagist/panda_simulator (besucht am 15. 07. 2021).

[9] Andreas ten Pas, Marcus Gualtieri, Kate Saenko und Robert Platt Jr. "Grasp Pose Detection in Point Clouds". In: *CoRR* abs/1706.09911 (2017). arXiv: 1706.09911. URL: http://arxiv.org/abs/1706.09911.

[10] Arjun Singh, James Sha, Karthik Narayan, Tudor Achim und Pieter Abbeel. *BigBIRD: (Big) Berkeley Instance Recognition Dataset*. URL: https://rll.berkeley.edu/bigbird/ (besucht am 15. 07. 2021).

[11] Andreas ten Pas. *ROS Wrapper for GPD*. URL: https://github.com/atenpas/gpd_ros (besucht am 15. 07. 2021).

[12] Tahsincan Köse. *ROS integration for Franka Emika research robots*. URL: https://github.com/tahsinkose/franka_ros (besucht am 15. 07. 2021).

[13] OpenCV. *Pose Estimation*. URL: https://docs.opencv.org/4.5.2/d7/d53/tutorial_py_pose.html (besucht am 15. 07. 2021).

[14] Intel Corporation. *Welcome to ROS2 Grasp Library Tutorials*. URL: https://intel.github.io/ros2_grasp_library/docs/index.html (besucht am 15. 07. 2021).