Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Constantin Titgemeyer

Oberflächenvermessung zum reparativen 3D-Druck mit einem Industrieroboter

Fakultät Technik und Informatik
Department Maschinenbau und Produktion

Faculty of Engineering and Computer Science
Department of Mechanical Engineering and Production Management

Constantin Titgemeyer

Oberflächenvermessung zum reparativen 3D-Druck mit einem Industrieroboter.

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung

im Studiengang Entwicklung und Konstruktion
am Department Maschinenbau und Produktion
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr. Ing Thomas Frischgesell
Zweitprüferin: Dipl. -Ing. Carolina Bohnert

Abgegeben am 23.08.2019

Constantin Titgemeyer

Thema der Bachelorthesis

Oberflächenvermessung zum reparativen 3D-Druck mit einem Industrieroboter.

Stichworte

Industrieroboter, Kuka, KR C4, Agilus, 3D-Druck, Scannen, Micro-Epsilon, scanCONTROL, Python, TCP/IP, XML, Server-Client, KRL, Ethernet-KRL, EKI, PolyWorks, ASCII, Punktwolke

Kurzzusammenfassung

Das Ziel dieser Arbeit ist das dreidimensionale Scannen von Objekten mit einem Industrieroboter. Dazu wird ein Laserlinienscanner an einen Roboter montiert und über Ethernet werden beide Parteien an ein Netzwerk angeschlossen. Um die gescannten Daten mit den Positionsdaten des Roboters zu vereinen, wird ein Computer als Bindeglied verwendet.

Sowohl das Ansteuern des Scanners als auch die Server-Client-Verbindung mit der Robotersteuerung wird mit Python bewerkstelligt. Zur Bedingung des scanCONTROL 2950-100 Laserlinienscanners wird das im SDK enthaltene Python-Binding verwendet. Der Verbindungsaufbau mit der KRC4-Steuerung wird über die EthernetKRL-Schnittstelle realisiert. Hierzu werden über das TCP/IP Datenpakete versendet, die eine festgelegte XML-Struktur besitzen. Aus den empfangenen Daten erstellt die Steuerung automatisch die abzufahrende Trajektorie.

Um die Bedienung des Scansystems zu erleichtern ist eine graphische Benutzeroberfläche erstellt worden.

Schlussendlich werden die erstellten Punktwolken mit PolyWorks in brauchbare Polygonmodelle umgewandelt. Diese sollen in zukünftigen Arbeiten unter anderem dazu verwendet werden, um mit demselben Roboter auf das gescannte Objekt dreidimensional zu drucken.

Constantin Titgemeyer

Title of the paper

Surface measuring for reparative 3D printing with an industrial robot.

Keywords

Industrial robotics, Kuka, KR C4, Agilus, 3D printing, scanning, Micro-Epsilon, scanCONTROL, Python, TCP/IP, XML, Server-Client, KRL, Ethernet-KRL, EKI, PolyWorks, ASCII, point cloud

Abstract

The aim of this thesis is the three-dimensional scanning of objects using an industrial robot. For this purpose a laser line scanner is mounted on a robot and both parties are connected to a network via Ethernet. In order to merge the scanned data with the position data of the robot, a computer is used as a bridge.

Both the controlling of the scanner and the server-client connection to the robot controller is done with Python. For the scanCONTROL 2950-100 laser line scanner the Python binding contained in the SDK is used. The connection to the KRC4 controller is established via the EthernetKRL-Interface. For this purpose, data packets with a defined XML structure are sent using the TCP/IP. The controller automatically creates the trajectory to be executed from the received data.

In order to facilitate the operation of the scanning system, a graphical user interface has been created.

Finally the created point clouds are converted into useful polygon models with PolyWorks. These will be used in future projects to print three-dimensionally on the scanned object using the same robot.

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen Bedanken, die mich bei dieser Arbeit motiviert und unterstützt haben.

Ein besonderer Dank gilt dabei Prof. Dr. Thomas Frischgesell, der den Stein ins Rollen gebracht hat, indem er mir dieses Thema angeboten hat. Dadurch war es mir möglich, tief in diese spannende Thematik einzutauchen und wertvolle Erfahrungen zu sammeln.

Auch möchte ich mich bei Dipl.-Ing. Carolina Bohnert, Ing. Robin Auffermann und Ing. Hasibullah Shafaq bedanken. Alle drei hatten immer ein offenes Ohr für Fragen, haben mir mehrfach geholfen und somit maßgeblich zum Erfolg dieser Arbeit beigetragen.

Ein Dank gilt auch Prof. Dr.-Ing. Günther Gravel und B. Eng. Hendrik Mietzner für das Zurverfügungstellen einer PolyWorks-Lizenz, und bei Monika Riedel möchte ich mich für das Fertigen der Halterung bedanken.

Zuletzt möchte ich mich bei meiner Familie und meinen Freunden bedanken, die mich während meines ganzen Studiums tatkräftig unterstützt haben und an meiner Seite standen.

Constantin Titgemeyer

Hamburg, 23.08.2019

Abkürzungsverzeichnis

CIRC	Kreisbewegung des Manipulators
EKI	EthernetKRL Interface
GMRK	Gestengesteuerte Mensch-Roboter-Kollaboration
GUI	Graphical User Interface
KKS	Kartesisches Koordinatensystem
KLI	KUKA Line Interface
KR 6	KUKA Robot 6
KRC4	KUKA Robot Control 4 (Compact)
KRL	KUKA Robot Language
LIN	Linearbewegung des Manipulators
PIL	Python Image Library
PoE	Power over Ethernet
PTP	Point-To-Point Bewegung des Manipulators
RSI	Robot Sensor Interface
SDK	Software Development Kit
TCP	Tool Center Point bzw. Endeffektor des Manipulators
TCP/IP	Transmission Control Protocol/Internet Protocol
TOF	Time of flight, dt. Laufzeitmessung
UDP	User Datagram Protocol
XML	Extensible Markup Language

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmcodeverzeichnis

1 Einführung

1.1 Motivation

Die digitale Welt wird von Tag zu Tag enger mit der realen Welt verknüpft. Seit den 1980er Jahren werden Objekte mit Computern dreidimensional dargestellt. Fast zeitgleich wurde auch die Barriere zwischen den beiden Welten durchbrochen, indem mit Stereolithografie das erste digitale Objekt in die reale Welt gesetzt wurde. Diese Technikrichtungen haben sich seitdem weit entwickelt. Es gibt etliche Varianten, reale Gegenstände zu vermessen, sie digital darzustellen und digitale Objekte in der Realität herzustellen. Trotzdem ist der Schritt von der einen in die andere Welt noch immer mit verschiedensten Herausforderungen gekoppelt.

In der heutigen Zeit wird man immer öfter mit virtuellen Objekten konfrontiert, die ihren Ursprung aus der realen Welt haben. Neben dem Gebrauch in Onlineshops oder Computerspielen, ist das Digitalisieren von Gegenständen für die Industrie besonders interessant. Durch Scannen eines gefertigten Teils und anschließendem Vergleichen mit der virtuellen Quelldatei kann die Qualität der Fertigung überprüft und gegebenenfalls verbessert werden. Wenn von einem Bauteil kein CAD-Modell vorhanden ist, kann dieses auch durch einen Scann erstellt werden. Scanning ist somit ein fundamentaler Bestandteil des Reverse Engineering.
Das präzise Vermessen eines Objektes und die Rückführung in ein bearbeitbares Modell benötigt abgestimmte Systeme. Diese unterliegen trotz ausgleichender Algorithmen physikalischen Gesetzen.

Im Rahmen eines Forschungsprojektes des Instituts für Mechanik und Mechatronik an der HAW Hamburg wurde daran gearbeitet, mit einem 6-achsigen Industrieroboter Kunststoffe dreidimensional zu drucken. Im Gegensatz zu einem herkömmlichen 3D-Drucker, der nur innerhalb paralleler Ebenen drucken kann, ist es einem 6-

Achsroboter möglich, wesentlich freiere Druckbewegungen auszuführen. Neben dem Filamentextruder wurde an der Halterung auch eine Frässpindel angebracht. Dadurch ist es möglich, mit dem Roboter sowohl additiv als auch subtraktiv zu fertigen.
Aus diesem Projekt ist die Idee entstanden, den Aufbau um eine Scanfunktion zu erweitern. Auf diese Weise wird es möglich, mit einem System die Barriere zwischen Digital und Real in beide Richtungen zu durchdringen.
Aus der Kombination von räumlichem Scannen und dem dreidimensionalen Drucken eröffnen sich neue Arten der Nachbearbeitung. Das Hauptaugenmerk liegt dabei in der Möglichkeit, beschädigte Gegenstände zu reparieren, indem die Oberfläche gescannt und das fehlende Material mit Filament ersetzt wird. Mit der gleichen Methode kann einem Gegenstand eine Funktion hinzugefügt werden. Es kann aber auch die Qualität des gedruckten Objekts automatisch überprüft und dann das überschüssige Material mit der Fräse entfernt werden.

Es existieren schon einige roboterunterstützte Scansysteme. Diese sind allerdings auf das alleinige Vermessen ausgelegt. Dadurch fehlt bei diesen der Raum für eine 3D-Druck- und Fräsfunktion. Diese Arbeit soll ein Scansystem hervorbringen, das mit eben diesen Funktionen erweiterbar ist.

1.2 Zielsetzung

Im Zentrum für Industrierobotik soll ein Laserlinienscanner am Roboterarm eines KUKA Agilus KR6 R900 sixx befestigt werden, um Oberflächen für eine weitere Verarbeitung durch 3D-Druck zu registrieren. Die vom Scanner aufgezeichneten Daten sollen mit den Positionsdaten des Roboters zu einer Punktwolke zusammengefügt werden. Hierzu wird ein PC als Schnittstelle zwischen Scanner und Roboter eingerichtet. Um die Einstellung und Bedienung zu erleichtern, wird eine grafische Benutzeroberfläche erstellt. Anhand der Eingabeparameter soll die vom Roboter zu fahrende Trajektorie automatisch berechnet werden. Mittels eines Referenzkörpers wird die Genauigkeit der gescannten Daten überprüft. Nach Erstellung einer STL- oder OBJ-Datei aus der gescannten Punktwolke soll eine

Möglichkeit aufgezeigt werden, wie diese Datei für die Weiterverarbeitung im 3D-Druck verwendet werden kann.

1.3 Struktur der Arbeit

Zunächst werden in Kapitel 2 die theoretischen Grundlagen des Roboters und des Scanners genauer erläutert. Hier wird nur auf die allgemeinen Grundlagen eingegangen, da die spezifischen an passender Stelle beschrieben werden. Auf die Halterung zur Befestigung des Scanners und ihre Anforderungen wird in Kapitel 3 näher eingegangen.
Da ein fundamentaler Teil der Arbeit aus dem Umgang mit Schnittstellen besteht, sind diese detailliert in Kapitel 4 erörtert. In diesem werden zunächst in den Unterkapiteln 4.1 und 4.2 die Schnittstellen der Komponenten aufgeführt und erklärt. Darauf folgt unter Unterkapitel 4.3 die Erklärung, wie der Computer mit Python als Bindeglied fungiert. Nachdem die gewählten Schnittstellen der einzelnen Systemelemente erläutert wurden, wird das Zusammenspiel im Unterkapitel 4.4 dargelegt.
Der Kern dieser Arbeit besteht aus den erstellten Programmen, die mit der Konfigurationsdatei anschaulich in Kapitel 5 aufgeführt werden. In 5.1 wird die auf Basis von Kapitel 4.1.2 erstellte Konfigurationsdatei erklärt. Danach werden die geschriebenen KRL- und Python-Programme mithilfe von Aktivitäts- und Objektdiagrammen erklärt.
Mit diesen Kapiteln ist die Frage der Umsetzung geklärt, worauf die Untersuchung der Ergebnisse folgt. Dies wird im sechsten Kapitel geschildert. In Kapitel 7 wird anhand eines Beispielscans die Weiterverarbeitung mit einem separaten Programm erläutert.
Vor dem abschließenden Ausblick wird die Arbeit noch einmal zusammengefasst und ein Fazit gezogen.

2 Theoretische Grundlagen

2.1 Theoretische Grundlagen zum Roboter

2.1.1 KUKA Agilus und KR C4

Die für den Scanprozess benötigte mechanische Bewegung wird von einem Industrieroboter ausgeführt. Der verwendete Roboter ist ein Agilus KR6 R900 sixx von der Firma KUKA. Die Agilus-Reihe sind Kleinroboter mit 5 bis 6 Achsen, die auf hohe Geschwindigkeiten ausgelegt sind.

Trotz der hohen Geschwindigkeit besitzt der 6-achsige Agilus KR6 R900 sixx eine hohe Widerholgenauigkeit von ±0.03 mm. Seine maximale Traglast beträgt 6kg. Durch Überschreitung dieser Maximaltraglast wird die Lebensdauer des Roboters beeinträchtigt. Zudem gilt die angegebene Wiederholgenauigkeit nur bei der Nenntraglast von 3kg [vgl. KUKA Roboter GmBH 2014a, KR AGILUS sixx Betriebsanleitung S.26 f.].

Entscheidend für die Anwendbarkeit des Scanprozesses ist der für diese Gewichtsklasse relativ große Arbeitsraum, der in **Fehler! Ungültiger Eigenverweis auf Textmarke.** dargestellt ist. Die Maßangaben sind hierbei in Millimetern. Beim ausgewählten Scanverfahren muss der Arbeitsraum wesentlich größer sein, als das zu scannende Objekt. Da der benötigte Arbeitsraum unter anderem von den frei wählbaren Scanwinkeln abhängig ist und die verbindenden PTP-Fahrten ohne Simulation unvorhersehbar sind, ist es nicht möglich, allein anhand des Arbeitsraums die maximale Größe des zu scannenden Objektes festzulegen.

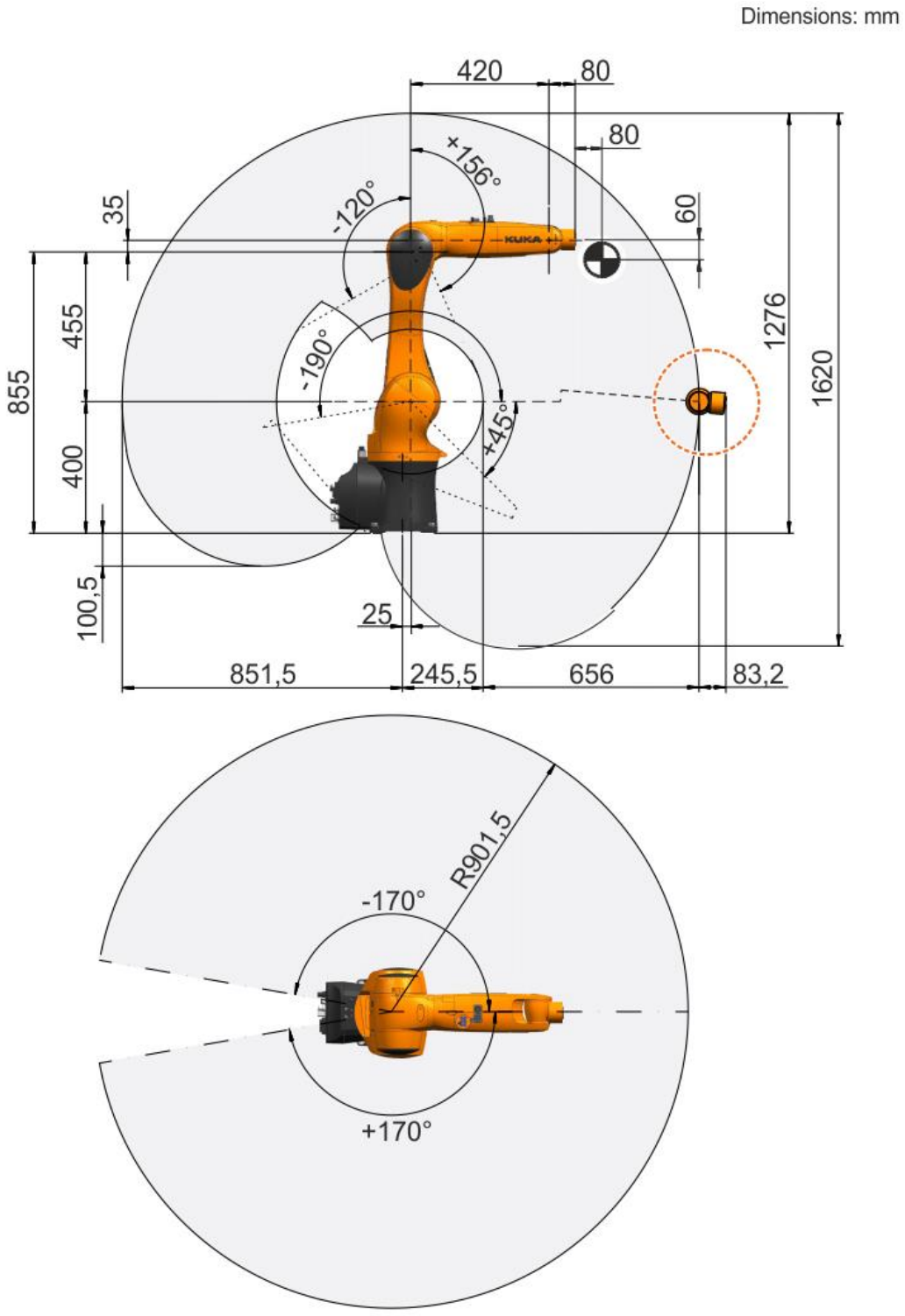


Abbildung 2-1: Arbeitsraum des KUKA Agilus KR6 R900 sixx
[KUKA Roboter GmbH 2014a, S. 23]

Als Steuerung für den Manipulator wird die kompakte Variante der KR C4 verwendet. Dies ist eine KUKA-Steuerung aus der aktuellen Generation, die im Gegensatz zu KR C2 einen integrierten SoftPLC (Programmable Logic Controller/SPS), eine CNC-

Steuerung und eine Steuerung für Sicherheitsmaßnamen besitzt. Sie unterstützt dabei sowohl die eigene Programmiersprache KRL als auch die Ansteuerung über Maschinencode und SPS (intern und extern) [vgl. KUKA Roboter GmBH 2016, Das Steuerungssystem der Zukunft_KR C4].
Zudem ist die Steuerung mit Hardware- und Softwaremodulen erweiterbar und kann so an die individuellen Bedürfnisse angepasst werden. Als Bedienhilfe kann das SmartPad herangezogen werden. Damit ist es möglich, Programme zu schreiben, den Roboter zu konfigurieren und ihn manuell zu bedienen. Nähere Informationen über die Steuerung und den Manipulator können den jeweiligen Dokumentationen von KUKA entnommen werden.

2.1.2 KRL

KUKA Robot Language ist eine von KUKA entwickelte Programmiersprache für die Steuerung von Robotern. Ein KRL-Programm teilt sich auf in eine SRC- und eine DAT-Datei, die zusammen als Modul bezeichnet werden. Der eigentlich durchzuführende Code befindet sich dabei in der SRC-Datei, die auch allein ausgeführt werden kann. Die DAT-Datei hingegen wird meistens automatisch erstellt und dazu genutzt, um Variablen zu hinterlegen. Abhängig davon, wo und wie die Variablen deklariert worden sind, ergibt sich deren Lebensdauer und Gültigkeit. Sind sie nur im SRC deklariert worden, so sind sie auch nur im jeweiligen Programm abrufbar und bearbeitbar. Wurden sie hingegen in der zugehörigen DAT definiert, sind sie in der gesamten SRC-Datei nutzbar, inklusive den Unterprogrammen und Funktionen. Um Variablen in unterschiedlichen Modulen nutzen zu können, müssen sie im User-Bereich der allgemein zugänglichen $Config.dat deklariert werden.

Am Anfang jedes Programms steht die Deklarationsphase, in der allen lokalen Variablen ein Daten-Typ zugeordnet werden muss. Neben den Standarddatentypen BOOL, INT, REAL und CHAR gibt es etliche spezielle Datentypen wie bspw. FRAME, das kartesische Koordinaten mit jeweiligem Winkel beinhaltet. Auch können

Erweiterungen wie EKI zusätzliche Datentypen mit sich bringen. Im Falle von EKI ist das unter anderem EKI_STATUS.

Die meisten Datentypen können zusätzlich als Feld (Array) deklariert werden. Dazu muss nach dem Variablennamen in eckigen Klammern die Feldgröße definiert werden. Mit einer weiteren Größe und einem trennenden Komma in den Klammern kann dem Feld eine Dimension hinzugefügt werden. KRL unterstützt dabei maximal dreidimensionale Felder (Bsp. *dimension3 = [1,2,3]*) [vgl. KUKA Roboter GmBH 2013, Einsatz und Programmierung von Industrierobotern S.190].

Die Deklarationsphase muss direkt nach Beginn des Programms stehen und darf nur einmal vorkommen. Unmittelbar danach findet die Initialisierung statt, bei der den Variablen ein Anfangswert oder Zeichen zugeordnet wird. Erst nach diesen beiden Phasen können die ersten Fahrtbefehle programmiert werden. Diese können dann mit den konventionellen Fahrtbefehlen PTP, LIN oder CIRC ausgeführt werden oder alternativ mit einem Inline-Formular, wie es in Abbildung 2-2 dargestellt ist. Letzteres empfiehlt sich insbesondre, wenn man mit dem SmartPad arbeitet, da sich die Eingabe über Dropdown Menüs schneller gestaltet als über die Bildschirmtastatur.

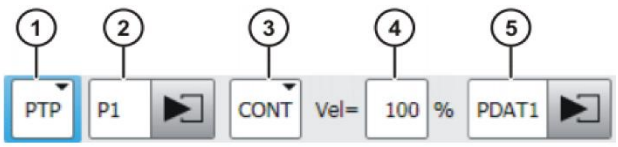


Abbildung 2-2: PTP Inline-Formular
[KUKA Roboter GmBH 2014b, S.318]

1. Bewegungsart
2. Name des Zielpunkts
3. Bei CONT wird überschliffen
4. Geschwindigkeit
5. Name des Bewegungsdatensatzes

Die Eingabemöglichkeiten eines Inline-Formulars verändern sich dynamisch je nach Auswahl. Nähere Informationen können der Dokumentation KUKA System Software 8.3 entnommen werden.

Um den Code übersichtlicher zu gestalten, können sogenannte Folds verwendet werden. Diese beginnen mit dem Kommentarzeichen „;“ und dem Schlüsselwort *FOLD*. Sie enden mit *;ENDFOLD* und können dann eingefaltet werden. Die Ansichten eines KRL-Programms unterscheiden sich, je nachdem, ob es mit dem SmartPad geöffnet wurde oder mit einer Desktop Anwendung wie WorkVisual. Dabei ist die

Ansicht des SmartPads oft stark vereinfacht und manche Daten können nicht bearbeitet werden. Deshalb wurden die meisten KRL-Programme dieser Arbeit mit dem Editor Orange-Edit.free von OrangeApps erstellt, da man in diesem einfach von der SmartPad-Ansicht auf die Desktop-Ansicht wechseln kann. Zudem kann man damit Inline-Formulare am Desktop erstellen und erhält eine rudimentäre Syntax-Überprüfung.

2.1.3 Submit-Interpreter

Auf der KRC4-Steuerung können zwei Prozesse parallel zueinander ablaufen. Zum einen der Roboter-Interpreter, der die Bewegungsprogramme abläuft, sowie der Submit-Interpreter, der in erster Linie die Sicherheitsroutinen durchläuft. Von diesen beiden besitzt der Roboter-Interpreter die höhere Priorität und hat eine regelmäßige Taktzeit von 12ms. Die Taktzeit des Submit-Interpreters ist hingegen unregelmäßig, weshalb damit keine zeitkritischen Anwendungen ausgeführt werden sollten. Die Geschwindigkeit und Wiederholungsrate werden maßgeblich von der Länge des Scripts beeinflusst, das der Interpreter durchläuft. Selbst leere Zeilen haben darauf Einfluss.

Der Submit-Interpreter kann als Soft-SPS benutzt werden, um bei Anwendungen mit geringerem Umfang die Anschaffung einer teuren SPS zu sparen [vgl. KUKA Roboter GmBH 2014b, Kuka System Software 8.3 S.469].

Beim Hochfahren der Steuerung und dem damit verbundenen Starten des Submit-Interpreters wird automatisch das sps.sub Programm angewendet. Darin sind die wichtigsten Routinen standartmäßig programmiert, die der Interpreter durchführen soll. Viele Zusatzpakete von Kuka fügen dieser sub-Datei zusätzliche Routinen hinzu. Um zu vermeiden, dass andere Programmabläufe gestört werden, sind für den Anwender explizite User-Folds ausgewiesen, die er benutzen kann.

Standartmäßig stehen dem User drei Folds zur Programmierung zur Verfügung. Der erste User-Fold ist zur Deklaration von Variablen gedacht, die in den darauffolgenden User-Folds verwendet werden sollen. Der Submit-Interpreter kann jedoch auch auf System-Variablen und Variablen zugreifen, die in der $cofig.dat-Datei global

deklariert wurden. Der zweite Fold ist der Initialisierungs-Fold. Das hier geschriebene Programm wird einmal direkt nach dem Starten der SUB-Datei ausgeführt. Im dritten User-Fold werden die User-Routinen erstellt.
Beim Programmieren in der SUB-Datei muss auf folgende Dinge geachtet werden:
Im SUB-Programm dürfen keine WAIT-Operationen oder Endlosschleifen programmiert werden, da einige Funktionen von den Routinen im sps.sub abhängen, wobei sich manche davon um den einwandfreien Ablauf des Roboters kümmern. Deshalb darf die Routine nicht aufgehalten werden. Vor dem Bearbeiten der sps.sub sollte eine Sicherheitskopie angelegt und die anderen Warnhinweise im Kapitel 12 der Dokumentation KUKA System Software 8.3 beachtet werden.

2.1.4 Konfiguration der Benutzergruppe

Standartmäßig ist bei der KR C4 die Benutzergruppe *Anwender* eingestellt. Diese Benutzergruppe ist in ihren Möglichkeiten sehr begrenzt. So kann sie nur Programme anwählen oder abwählen, jedoch nicht bearbeiten. Um dies tun zu können, muss der Benutzer zunächst in den Einstellungen zu einer Benutzergruppe wechseln, die einen höheren Rang besitzt, bspw. *Experte.* Mit dieser Benutzergruppe ist es auch möglich, den Submit-Interpreter an und ab zu wählen. Standardmäßig wird nach 5 Minuten ohne Verwendung der Benutzer automatisch auf die Benutzergruppe *Anwender* herabgestuft. Muss danach der Submit-Interpreter neu angewählt oder eine Änderung am Programm vorgenommen werden, so muss zunächst wieder ein Benutzergruppenwechsel durchgeführt werden.

Bei häufigem An- und Abwählen des Submit-Interpreters, um die Verbindung neu aufzubauen, ist der Benutzergruppenwechsel zeitaufwändig. Dieses Problem kann durch eine Bearbeitung der Konfigurationsdatei *Authentication.config* gelöst werden. Zu finden im Verzeichnis *C:\\KRC\SmartHMI\Config* der Steuerung kann in dieser Datei sowohl die Zeit bis zum Zurücksetzen des Benutzers eingestellt, als auch die Standardbenutzergruppe definiert werden. Die Zeit kann eingestellt werden, indem

man der Variablen *LeaseTime* eine entsprechende Zahl zuordnet. Wird *LeaseTime* auf null gesetzt, so wird der automatische Benutzerwechsel ausgeschaltet.
Um den Standardbenutzer von *Anwender* auf *Experte* zu setzten, muss in der selben Konfigurationsdatei der *Default-Userlevel* von 5 bzw. 10 auf 20 gesetzt werden. Dies hat allerdings zur Folge, dass jeder, der den Roboter bedient, ohne Passwortabfrage automatisch Zugriffsrechte auf die Expertenebene hat. Bei falscher Handhabung können dadurch irreparable Schäden an Mensch und Maschine entstehen.

2.2 Theoretische Grundlage zum Scanner

2.2.1 Dreidimensionale Oberflächenvermessung

Für das Scannen von dreidimensionalen Oberflächen existieren viele unterschiedliche Verfahrensarten. Am häufigsten werden dabei taktile und optische Messmethoden verwendet. Da das dreidimensionale taktile Vermessen eines Objektes wesentlich zeitaufwändiger ist, werden im Folgenden nur optische Messmethoden genauer beschrieben. Unter optische Messmethoden fallen unter anderem das Lichtschnittverfahren, das Scannen mit Streifenlichtprojektion, die Photogrammetrie und die verschiedenen Arten des TOF-Scannens (Time of flight).
Beim Vermessen mit der TOF-Methode werden Strahlen oder Wellen versendet und deren Reflektion empfangen. Anhand der vergangenen Zeit und der bekannten Geschwindigkeit der Strahlung oder der Wellen, wird die Entfernung berechnet. Das Radar ist dabei eine der bekanntesten TOF-Methoden, die Wellen verwendet. Eine optische TOF-Methode hingegen ist das Lidarverfahren, das Laserstrahlen zur Vermessung verwendet. Damit ist auch dreidimensionales Vermessen möglich. So wird das Mapping von mobilen Robotern oft mithilfe von Lidarsensoren bewerkstelligt.
Es existieren auch TOF-Kameras, die mithilfe von Photomischdetektoren Tiefeninformationen erfassen können [vgl. Norbert Lossau 2002, 3D-Kamera erfasst ihr räumliches Umfeld in Echtzeit].

In der Photogrammetrie wird ein Objekt aus verschiedenen Perspektiven fotografiert und mithilfe dieser Fotografien wird versucht, ein dreidimensionales Abbild des Objekts zu erzeugen. Anhand von markanten Punkten werden die Stellen ermittelt, an denen sich die Bilder überlappen. Mit Hilfe einer Kombination aus den Metadaten der Fotografien und den Positionsdaten der Kamera während des Auslösens wird dann ein Modell berechnet. Um die Genauigkeit zu erhöhen, werden oft Markierungen verwendet. Einsatz findet die Photogrammetrie bspw. bei der Erstellung von topographischen Karten. Auch in der Medienindustrie ist diese Methode beliebt, da sie 3D-Modelle mit passender Textur erstellen kann.
Das Scannen mit Streifenlichtprojektion kombiniert die Messprinzipien der Photogrammetrie und das Triangulationsprinzip des Lichtschnittverfahrens, das in Kapitel 2.2.2 genauer beschrieben wird. Beim Scannen werden mehrere kodierte Streifen mit einem Projektor auf die Zieloberfläche projiziert. Die von der Oberfläche verformten Streifen werden mit mindestens einer Kamera aufgenommen, die in einem definierten Winkel zum Projektor steht. Anhand der Verformung der Steifen kann mit den Bilddaten und Methoden der Photogrammetrie ein 3D-Modell berechnet werden. Die Kodierung der Streifen hat dabei für die Software eine ähnliche Funktion wie Markierungen. Um Lücken durch Beschattungen zu vermeiden, kann entweder das zu scannende Objekt oder das Scansystem bewegt werden [vgl. Handbuch Optische Messtechnik, S.44]. Nähre Informationen bezüglich der verschiedenen Kodierungsarten können im Handbuch optische Messtechnik auf Seite 45 ff. nachgeschlagen werden.

Die meisten TOF-Verfahren und die Photogrammetrie eignen sich hervorragend für die Vermessung sehr großer Objekte, sind dadurch jedoch nur bedingt präzise und eignen sich deshalb nicht für die gewünschte Anwendung. Sowohl zur Streifenlichtprojektion als auch zum Laserscannen existieren bereits Lösungen in Kombination mit Industrierobotern. Eine Lösung die Streifenprojektion benutzt, bietet Gom mit den ATOS Systemen. Creaform bietet mit dem MetraSCAN 3D-R hingegen ein System mit einem 3D-Laserscanner.

Diese Lösungen sind jedoch zu groß und zu schwer, um sie zusammen mit dem 3D-Druck- und Fräswerkzeug an den Kuka Agilus anbringen zu können. Da das Scannen mit Streifenlichtprojektion immer einen Projektor und mindestens eine Kamera benötigt, ist es aus den oben genannten Gründen an sich für diese Anwendung nicht geeignet. Im Vergleich dazu ist ein Laserlinienscanner leicht und kompakt. Die Kombination von Kompaktheit und Präzision war der Hauptgrund, weshalb die Entscheidung auf den scanCONTROL von Micro-Epsilon viel. Einen Kompromiss muss man dabei allerdings eingehen: da der scanCONTROL von Micro-Epsilon Daten nur zweidimensional erfassen kann, muss eine präzise Relativbewegung zum Zielobjekt erzeugt werden.

2.2.2 Funktionsweise des Laserlinienscanners

Das Erfassen von Oberflächen mit Laserlinienscannern gehört zum Lichtschnittverfahren. Dieses Verfahren verwendet das Triangulationsprinzip. Dabei wird eine Laserlinie auf die zu messende Oberfläche projiziert. Die von Objekt reflektierten Strahlen werden von einer Optik erfasst und auf einen lichtempfindlichen Sensor gelenkt.

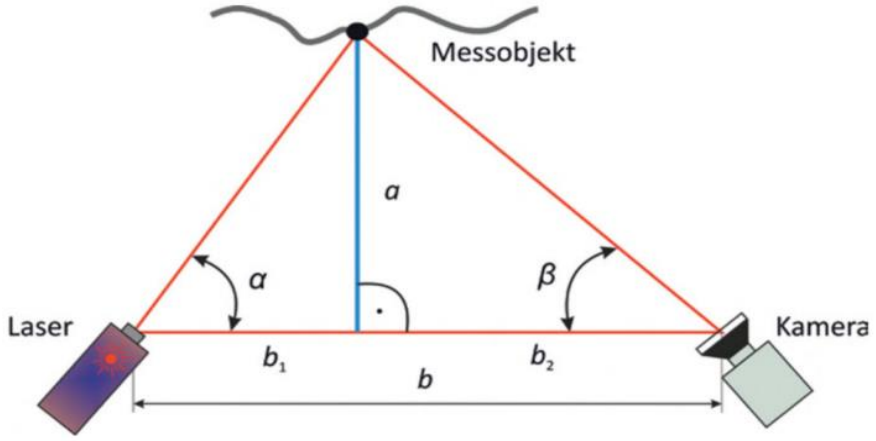


Abbildung 2-3: Triangulationsprinzip

[Optische Messtechnik 2017, S.11 Bild 2.3]

Voraussetzung zur Berechnung der Entfernung zum Objekt ist der Abstand des Sensors zum Laser und die Winkel der jeweiligen Ausrichtung. In der Abbildung 2-3

sind diese Werte als die Länge b und die Winkeln α und β skizziert. Der Abstand a kann mit der folgenden Formel berechnet werden:

$$a = \mathrm{b}\frac{\tan\alpha \cdot \tan\beta}{\tan\alpha + \tan\beta}$$

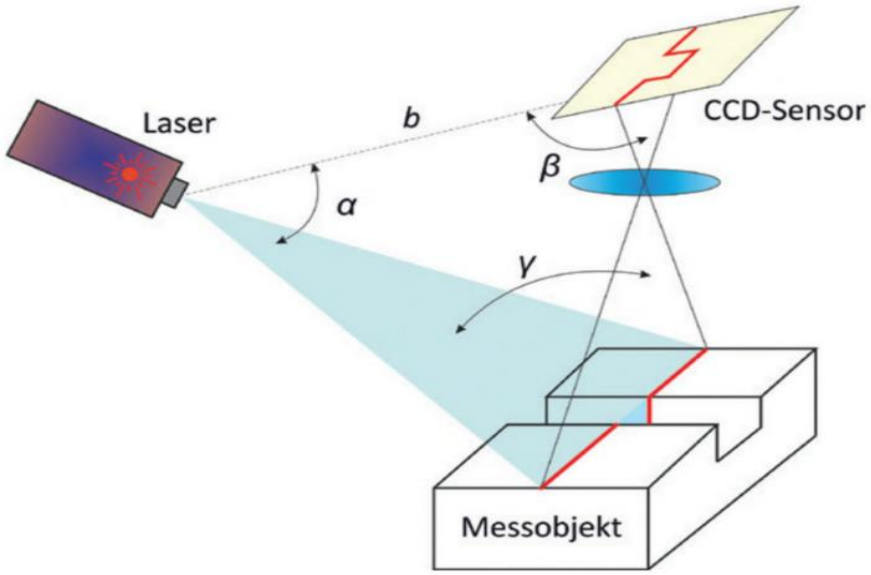


Abbildung 2-4: Prinzip des Lichtschnittsensors
[Optische Messtechnik 2017, S.11 Bild 2.4]

In Abbildung 2-4 wird gezeigt, wie die Reflektion der projizierten Laserlinie auf den Sensor trifft. Den einzelnen Punkten der Linie können dann zweidimensionale Werte zugeordnet werden [vgl. Handbuch Optische Messtechnik, S.11].

Oft sind der Laser und der Sensor gemeinsam in einem kompakten Gehäuse fest verbaut, weshalb ihr Abstand und ihre Winkelstellung zueinander somit bekannt ist. In den Gehäusen der Laserlinienscanner von Micro-Epsilon ist zudem ein Controller eingebaut, der die Daten der Sensormatrix interpretiert und in ein Koordinatensystem überträgt [vgl. Micro-Epsilon, Glossar Laser-Linien-Triangulation].

Ein Problem dieser Messmethode sind die in Abbildung 2-5 gezeigten Abschattungen. Damit der Sensor das an der Oberfläche reflektierte Licht detektieren kann, darf es zwischen Oberfläche und Sensor nicht abgelenkt oder blockiert werden. Die meisten Laserlinien werden erzeugt, indem ein Punktlaser mit einer Optik zu einer Linie aufgefächert wird. Je weiter die Linie von der Optik entfernt projiziert wird, desto länger wird sie. Bei Kanten, die einen steileren Winkel besitzen als der Winkel der Auffächerung, entsteht zwangsweise eine Abschattung. Um solche Abschattungen auszugleichen, muss die Position des Scanners zum Objekt oder v.v. geändert werden [vgl. Micro-Epsilon, Betriebsanleitung scanCONTROL 29xx S.46].

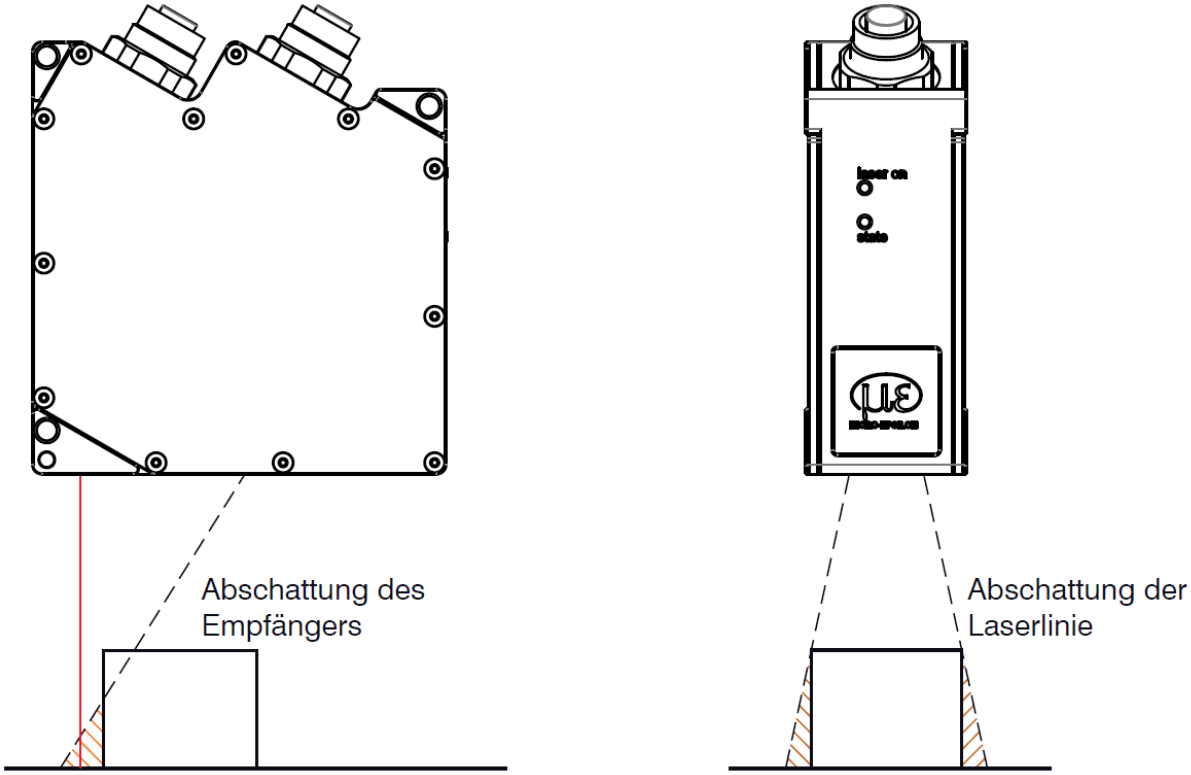


Abbildung 2-5: Abschattungen
[Betriebsanleitung scanCONTROL 29xx S.46 Abb. 23]

2.2.3 Eigenschaften des scanCONTROL 2950-100

Mit der scanCONTROL-Serie bietet Micro-Epsilon eine Vielzahl an leichten und kompakten Laserscannern. Die vierstellige Zahl gibt dabei die Modellreihe und die Klasse an. 29 steht für die Modellreihe mit einer Auflösung von bis zu 1280 Punkten pro Profil. Im Vergleich besitz die Reihe 26 die Hälfte an Punkten und die Reihe 30 das Doppelte. Die 50 steht für die High-Speed-Klasse, die sich in der 29er-Reihe mit bis zu 2000Hz auszeichnet. Am Ende der Modellbezeichnung ist die Kennzahl der Messbereichsgröße angegeben. Die 100 beschreibt dabei die Linienlänge in der Messbereichsmitte in Millimetern. Bei diesem Modell wird damit zusätzlich auch die Größe des Messbereichs in Z-Richtung angegeben. Dies ist der größte Messbereich der

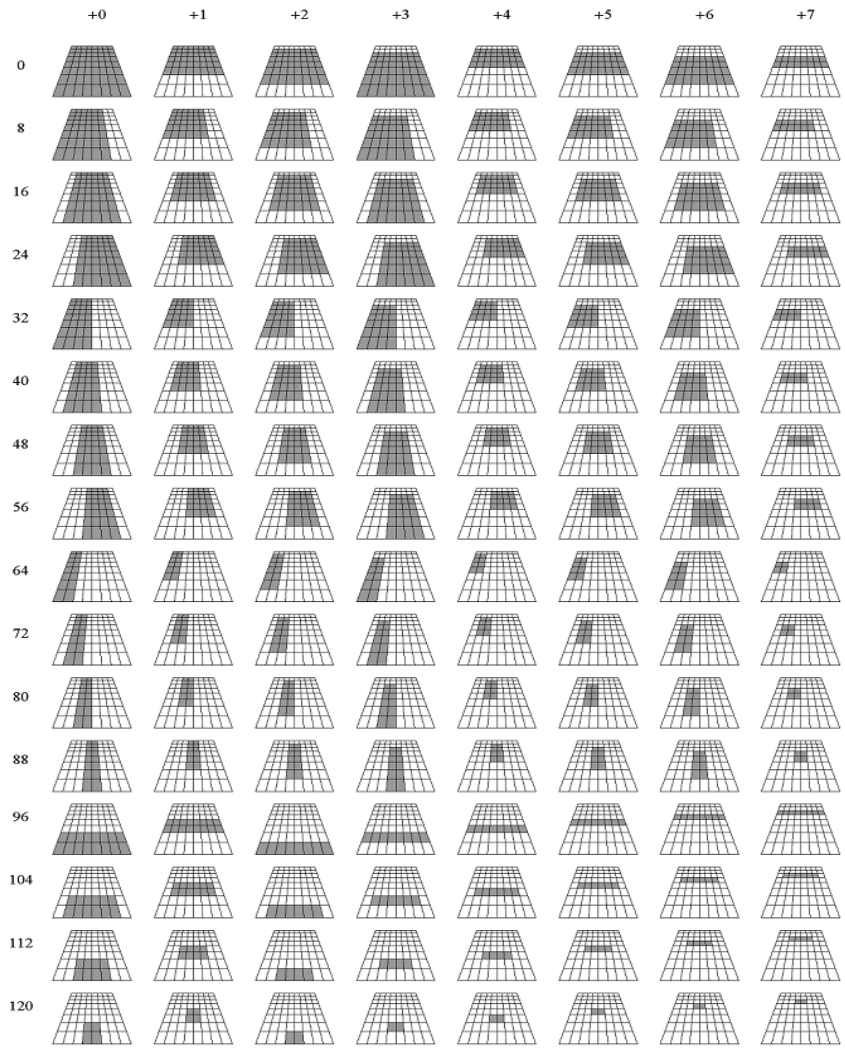


Abbildung 2-6: Vordefinierte Messfelder
[Betriebsanleitung scanCONTROL 29xx, S.39 Abb. 19]

29xx-Reihe. Dabei ist zu beachten, dass trotz der unterschiedlichen Größen die Auflösung innerhalb der Reihe gleich bleibt. Bei einem größeren Messbereich ist die Referenzauflösung somit kleiner. So liegt bspw. die Referenzauflösung bei der Größe 100 bei 12 µm, bei der Größe 10 hingegen bei 1 µm. Trotz der geringeren Genauigkeit wurde das Modell mit eine Linienlänge von 100mm ausgewählt, da bei größeren Objekten weniger Fahrten benötigt werden, um dieses vollständig zu scannen. Beim Zusammenfügen der unterschiedlichen Scanfahrten entstehen Ungenauigkeiten, die den Genauigkeitsvorteil der Größe 10 aushebeln.

In Abbildung 2-6 sind die von Micro-Epsilon vordefinierten Messfelder gezeigt. Das gewünschte trapezförmige Messfeld kann während der Scannerkonfiguration mit dem jeweiligen Zahlenwert eingestellt werden. Dabei ist zu beachten, dass die hohen Abtastraten der High-Speed-Klasse nicht mit jedem konfigurierbaren Messfeld erreicht werden können. Hierbei gilt, je kleiner das Messfeld, desto größer kann die Abtastrate werden. Die maximale Profilfrequenz von 2000Hz kann dabei nur mit den kleinsten Messfeldern erreicht werden. Wie hoch die maximale Abtastrate der jeweiligen Messfelder ist, steht in der Tabelle 1 [ScanCONTROL 2900 Quick Reference 2014].

Maximale Abtastrate in Abhängigkeit von den Messfeldnummern								
	+0	**+1**	**+2**	**+3**	**+4**	**+5**	**+6**	**+7**
0	144Hz	192Hz	192Hz	192Hz	284Hz	284Hz	284Hz	552Hz
8	181Hz	239Hz	239Hz	239Hz	354Hz	354Hz	354Hz	680Hz
16	181Hz	239Hz	239Hz	239Hz	354Hz	354Hz	354Hz	680Hz
24	181Hz	239Hz	239Hz	239Hz	354Hz	354Hz	354Hz	680Hz
32	241Hz	318Hz	318Hz	318Hz	469Hz	469Hz	469Hz	892Hz
40	241Hz	318Hz	318Hz	318Hz	469Hz	469Hz	469Hz	892Hz
48	241Hz	318Hz	318Hz	318Hz	469Hz	469Hz	469Hz	892Hz
56	241Hz	318Hz	318Hz	318Hz	469Hz	469Hz	469Hz	892Hz
64	362Hz	476Hz	476Hz	476Hz	694Hz	694Hz	694Hz	1282Hz
72	362Hz	476Hz	476Hz	476Hz	694Hz	694Hz	694Hz	1282Hz
80	362Hz	476Hz	476Hz	476Hz	694Hz	694Hz	694Hz	1282Hz
88	362Hz	476Hz	476Hz	476Hz	694Hz	694Hz	694Hz	1282Hz
96	552Hz	552Hz	1030Hz	1030Hz	1030Hz	1030Hz	1030Hz	1030Hz
104	680Hz	680Hz	1265Hz	1265Hz	1265Hz	1265Hz	1265Hz	1265Hz
112	892Hz	892Hz	1612Hz	1612Hz	1612Hz	1612Hz	1612Hz	1612Hz
120	1282Hz	1282Hz	2000Hz	2000Hz	2000Hz	2000Hz	2000Hz	2000Hz

Tabelle 1: Maximale Abtastrate
[ScanCONTROL 2900 Quick Reference 2014, Sensor Hardware]

Im scanCONTROL 2950-100 ist, wie im vorherigen Kapitel erwähnt, ein Controller integriert, der die Sensordaten interpretiert und in ein scannereigenes Koordinatensystem übersetzt. Der Ursprung der Z-Achse liegt an der Austrittsstelle des Lasers und verläuft dabei positiv in Ausbreitungsrichtung parallel zur aufgespannten Laserebene. Der Nullpunkt der X-Achse befindet sich in der Mitte der Laserlinie.

3 Halterung

Um den Scanner am Roboterarm zu befestigen, wird eine Halterung benötigt. Theoretisch würde hier eine einfache Adapterplatte mit Bohrungen für die Gewinde am Roboter und dem Scanner ausreichen, um den Scanner am Roboter betreiben zu können. Da der Scanner jedoch ohne Werkzeugwechsel gleichzeitig mit dem 3d-Druckaufsatz und der Fräse betrieben werden soll, ist eine passende Halterung erforderlich. Bei dieser sind folgende Dinge zu beachten: Zum einen muss die

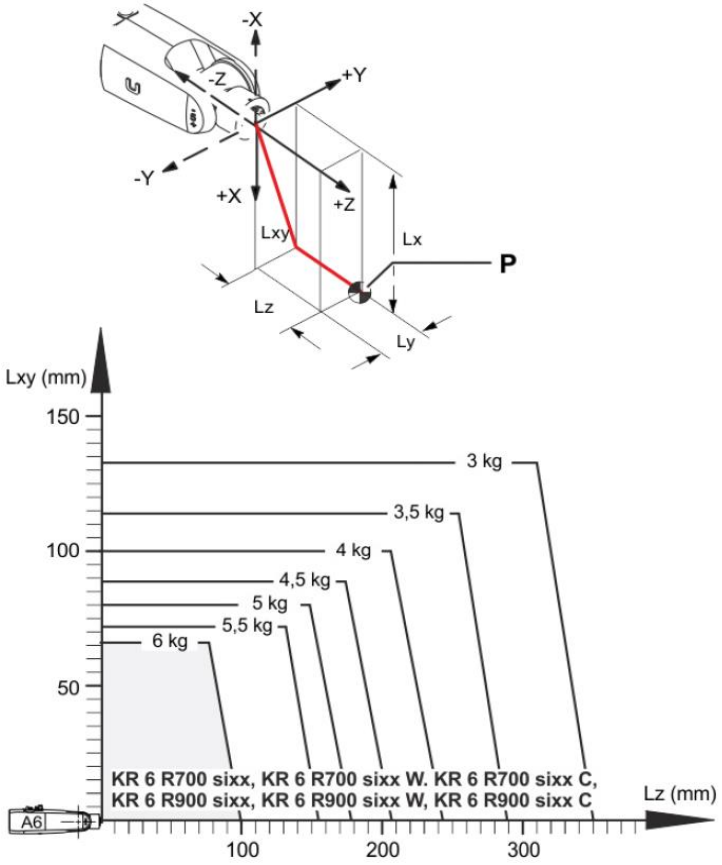


Abbildung 3-1: Traglastdiagramm des KR6

[KUKA KR Agilus sixx Betriebsanleitung 2014, S.27 Abb.4-8]

Halterung sowohl selbstständig den Scanner halten, als auch im Zusammenhang mit der bestehenden Halterung des Extruders und der Fräse. Zum anderen muss der Agilus das Gesamtgewicht präzise bewegen können. Um dies zu gewährleisten, wird das Traglastdiagramm in Abbildung 3-1 herangezogen. Der Schwerpunkt des

Werkzeuges kann hierbei anhand der maximalen Traglast festgelegt werden und umgekehrt.

Nach der Schwerpunktberechnung von Herrn Gaertner, die im Zuge seines Masterprojektes entstanden ist, beträgt das Gesamtgewicht des Werkzeuges ohne Scanner 4.13 kg und der Schwerpunkt liegt in z-Richtung bei 74mm. Hierbei wurden viele Vereinfachungen getroffen und die Verkabelung sowie das eventuelle Filament vernachlässigt.

Um soweit wie möglich von der maximalen Traglast entfernt zu bleiben, wurde die Halterung so dünn wie möglich konzipiert. Sie besteht aus zwei 6mm G.AL C250 Platten, die im rechten Winkel mit drei M4-Schrauben zueinander verschraubt werden. Die Grundplatte, die an das Handgelenk geschraubt wird, besitzt an den Oberseiten einen Lochkreis mit 7 Durchgangsbohrungen für die M5-Schrauben und einer 5H7-Passung für den Passstift. Unterhalb des Lochkreises ist ein länglicher Steg, an dessen Ende sich drei Durchgangsbohrungen für M4-Schrauben befinden. Diese dienen der rechtwinkligen Befestigung der Adapterplatte des Scanners an der Grundplatte, wie es auf dem linken Bild der Abbildung 3-2 dargestellt ist. Die Adapterplatte besitzt dafür die passenden M4-Gewindebohrungen und zusätzlich drei vertikale Durchgangsbohrungen für die M5-Schrauben, die den Scanner befestigen. Dabei wird der Scanner, wie im rechten Bild gezeigt, zum Handgelenk hin auf die Platte montiert. Dadurch ist der Schwerpunkt zentraler und zudem erleichtert es die Montage, wenn das gesamte Werkzeug an den Roboter angebracht ist. Die

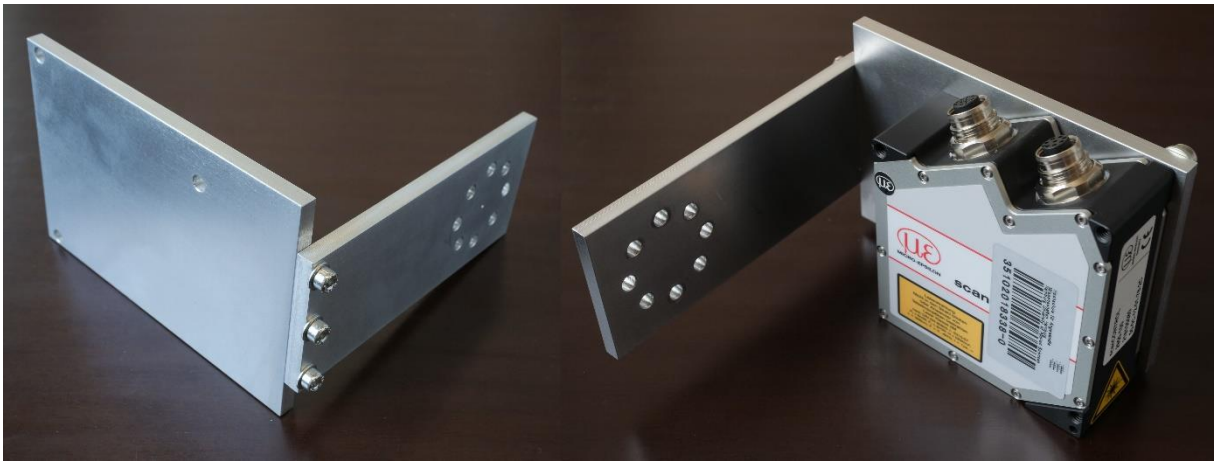


Abbildung 3-2: Halterungsplatten mit 3 M4-Schrauben verbunden (links) Halterung mit amgbrachtem Scanner (rechts)

Grundplatte kann sowohl direkt an das Handgelenk geschraubt werden als auch zwischen Handgelenk und Adapterplatte der Extruder-/Fräsenhalterung geklemmt werden. Der Steg ist dabei lang genug, damit der Scanner zwischen seiner Adapterplatte und der Fräsenhalterung passt.

Die gefertigte Halterung hat ein Gewicht von 0.28kg. Der gesamte Aufbau am Handgelenk des Manipulators hat somit ein Gewicht von 4,78kg. Um den Schwerpunkt zu ermitteln, wird als Grundlage die Berechnung von Herrn Gaertner genommen. Die Berechnung ist in Tabelle 2 abgebildet. Wegen der Verschiebung durch die neue Grundplatte wurde den Schwerpunkten aus der alten Berechnung jeweils 6mm hinzugefügt.

Komponente	**Gewicht**	**Schwerpkt.**	**Kommentar**
	m_i [kg]	$z_{s,i}$ [mm]	
Motorspindel	2,5	58	
Halterung Motorspindel	0,53	58	
Druckkopf	0,717	176	
Schrauben am Flansch	0,026	12	M5 x 18, 7 Stück
Adapterplatte Flansch	0,217	12	
Adapterplatte Druckkopf	0,148	176	
Grundplatte	0,098	3	
Adapterplatte Scanner	0,169	68,57	
Scanner	0,36	86	
Schrauben am Scanner	0,0086	97,81	M5 x 12, 3 Stück
Schrauben der Halterung	0,0054	6	M4 x 12, 3 Stück
Summen ∑ =	**4,78**	**78,4**	

Tabelle 2: Schwerpunktberechnung der Halterung

Der neu errechnete Gesamtschwerpunkt liegt nun bei 78,4mm vom Handgelenk entfernt. Die Sicherheit vor Überlast ist somit trotz der Vereinfachungen gegeben. Da jedoch die nominale Traglast von 3kg schon ohne den Scanner überschritten wird, kann die Nenngenauigkeit vom KUKA Agilus nicht mehr garantiert werden.

4 Schnittstellen

Schnittstellen sind ein essenzieller Teil dieser Arbeit. Sie bieten die Möglichkeit, unterschiedliche Geräte miteinander zu verknüpfen und damit einen Austausch von Daten zu ermöglichen.
Um das dreidimensionale Abtasten eines Körpers mit Hilfe einer Kombination aus mechanischen Bewegungen und optischen Messungen durchzuführen, ist das Verwenden von Schnittstellen unerlässlich. Nicht nur zur Koordinierung, sondern auch zum Sammeln und Auswerten der Daten werden unterschiedliche Arten von Schnittstellen benötigt.

4.1 Schnittstelle zur Kommunikation mit dem Roboter

4.1.1 KUKA Zusatzpakete

Wie in Kapitel 2.1.1 erwähnt, kann die Steuerung um Softwaremodule erweitert werden. Für die Kommunikation mit externen Systemen bietet KUKA zwei unterschiedliche Module als separat erhältliche Technologiepakete an. Zum einen das Robot Sensor Interface (RSI) und zum anderen EthernetKRL. Die Kommunikation vom RSI wird dabei entweder über das IO-System oder über Ethernet bewerkstelligt. Da keine externe SPS in dieser Arbeit verwendet wird, würde am ehesten die Verbindung über Ethernet in Frage kommen. Dazu könnten die Ethernetschnittstellen X66 und X67.1-3 verwendet werden, wobei die verwendete KR C4 compact nur die X66 Schnittstelle besitzt.
RSI benötigt eine Echtzeitübertragung und unterstützt nur UDP als Kommunikationsprotokoll. Das hat zur Folge, dass der Kommunikationspartner ebenfalls echtzeitfähig sein muss. Zudem muss die verwendete Software wegen des Datentransfers mittels UDP fehlende oder fehlerhafte Daten tolerieren können.

Da für diese Arbeit keine Echtzeitkommunikation benötigt wird und fehlende oder fehlerhafte Daten für die gewünschte Anwendung kritisch sein können, wurde die Kommunikation zwischen Computer und Steuerung stattdessen über EthernetKRL bewerkstelligt. Aus diesem Grund wird RSI hier nicht weiter beschrieben. Sollte Interesse an diesem Technologiepaket bestehen, können nähere Informationen aus der Dokumentation KUKA System Technology KUKA.RobotSensorInterface 3.2 entnommen werden.

EthernetKRL unterstützt sowohl UDP/IP als auch TCP/IP. Aus dem oben genannten Grund wird die Kommunikation über TCP/IP ausgeführt. Dazu muss eine Server-Client-Verbindung aufgebaut werden. Die Steuerung kann hierbei sowohl als Server als auch als Client betrieben werden. Das EthernetKRL Interface (EKI) kann als Server jedoch nur einen Client gleichzeitig besitzen. Soll trotzdem eine Verbindung mit mehreren Systemen aufgebaut werden, kann EKI als Client fungieren oder einen weiteren Server anlegen. Insgesamt können so 16 Verbindungen eingegangen werden. Die Datenpakete können nach Verbindungsaufbau entweder als binärer Datensatz (mit fester oder variabler Länge) verschickt werden oder als XML-Struktur [vgl. KUKA Roboter GmBH 2014c, Kuka.EthernetKRL 2.2 S.12].

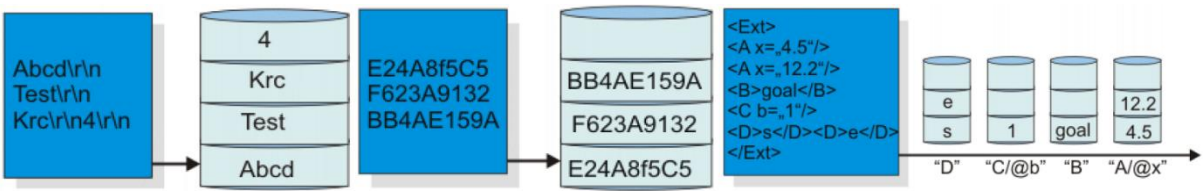


Abbildung 4-1: Daten-Speicher Binär mit Endzeichenfolge (links) Binär mit fester Länge (mitte) und XML-Daten-Speicher(rechts) [KUKA Roboter GmBH 2014c, S.11 Abb.2-2 & 2-3]

Neben der generell unterschiedlichen Struktur unterscheiden sich die möglichen Datenpaketarten teilweise auch in der Speicherung. In Abbildung 4-1 wird gezeigt, wie die empfangene Struktur in den Speicher eingepflegt wird. Bei den binären Verfahren werden die Daten nacheinander in denselben Speicher geschrieben. Im Vergleich dazu werden bei der Speicherung der XML-Datenpakete die Daten extrahiert und für jedes Element oder Attribut in einen eigenen Speicher geschrieben.

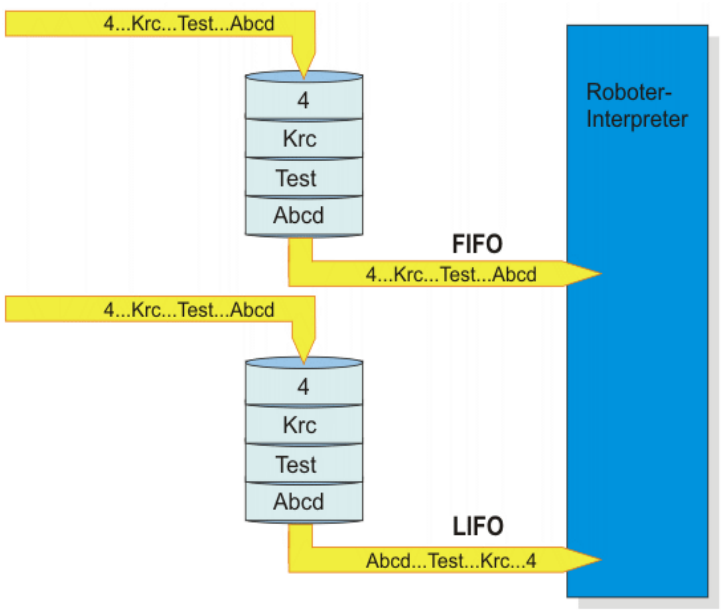


Abbildung 4-2: Übersicht der Ausleseverfahren
[KUKA Roboter GmBH 2014c, S.12 Abb.2-4]

Der Speicher kann dann, wie in Abbildung 4-2 gezeigt, mit zwei verschiedenen Verfahren ausgelesen werden. Beim FIFO-Verfahren (First In First Out) wird das zuerst ausgelesen, was auch zuerst in den Speicher geschrieben wurde. Beim alternativen LIFO-Verfahren (Last In First Out) hingegen wird der letzte Eintrag im Speicher als erstes ausgelesen. Wird in der Konfigurationsdatei nichts anderes festgelegt, wird der Speicher mit dem FIFO-Verfahren ausgelesen [vgl. KUKA Roboter GmBH 2014c, Kuka.EthernetKRL 2.2 S.11].
Da XML-Stuckaturen einfacher als binäre Datensätze lesbar sind und keine der Paketarten Nachteile bei der Übertragung haben, fiel die Wahl in dieser Arbeit auf die XML-Struktur. Trotz der Intoleranz beim Parsen von fehlerhaften XML-Paketen gestaltet sich die Fehlersuche wesentlich einfacher, als bei der Verwendung binärer Pakete.

Jede Verbindung, die über EKI aufgebaut werden soll, muss zuvor konfiguriert werden. Unabhängig davon, ob binäre Datenpakete oder XML-Datenpakete versendet und empfangen werden, geschieht dies mit einer XML-Datei, die in Kapitel 4.1.2 näher erläutert wird. Für jede Verbindung muss dabei eine eigene Konfigurationsdatei erstellt werden.

4.1.2 XML: Konfigurationsdateien und Datenpakete

Extensible Markup Language (XML) ist eine standardisierte Auszeichnungssprache und wie HTML von der Standard Generalized Markup Language (SGML) abgeleitet worden. Durch die standardisierte Form können mit ihr unabhängig von Plattform oder Programmiersprache Daten hinterlegt, ausgetauscht und ausgelesen werden. Ein großer Vorteil dieser Sprache ist dabei, dass die Struktur vom Inhalt und der Darstellung getrennt ist. Dies verleiht XML eine hohe Flexibilität in der praktischen Anwendung [vgl. Helmut Vonhoegen 2018, XML S. 31f. & S. 617].

Nach dem Prolog, in dem unter anderem die XML-Version und das Encoding festgelegt werden, beginnt die XML-typische Baumstruktur aus Elementen. Ein Element besteht aus einem Start-Tag, dem Elementinhalt und einem End-Tag. Die Tags selbst bestehen aus dem Namen des Elements, der von „<“ und „>“ eingeschlossen wird. Um es zu kennzeichnen, besitzt das End-Tag zusätzlich vor dem Namen ein „/“. Der Elementinhalt kann eine beliebige Zeichenfolge sein und somit kann auch ein Element selbst ein Elementinhalt sein. Elemente können somit beliebig oft ineinander verschachtelt werden. Dabei bezeichnet man das jeweilig äußere Element als Elternelement und das innere als Kindelement. Elternelemente können beliebig viele Kinder besitzen, aber ein Kindelement kann zwangsweise nur ein Elternelement haben. Das einzige Element, das kein Elternelement besitzt, ist das Wurzelelement [vgl. Helmut Vonhoegen 2018, XML S. 53f.]. Zur Veranschaulichung

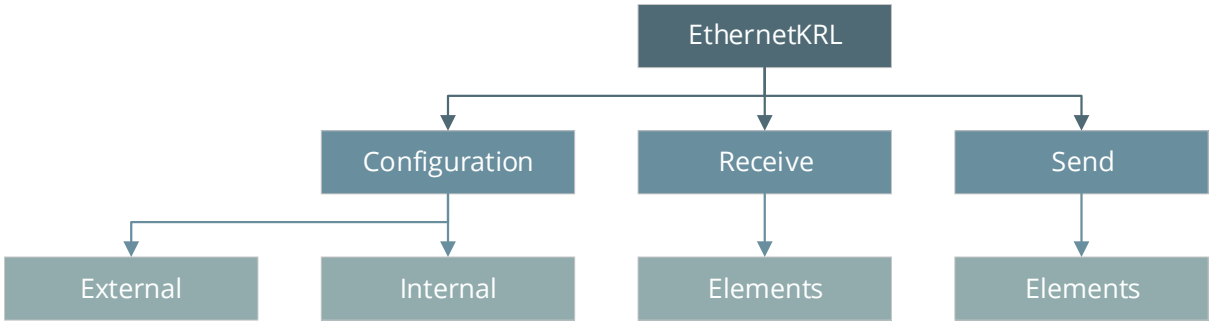


Abbildung 4-3: Elementenbaum der Konfigurationsstruktur

ist in Abbildung 4-3 der Elementenbaum der von KUKA vorgegebene Struktur der Konfigurationsdatei dargestellt. „EthernetKRL“ ist hierbei das Wurzelelement.
Neben Elementen werden in XML auch Attribute verwendet. Diese werden meist eingesetzt, um Elementen Zusatzinformationen zu geben. Im Gegensatz zu Elementen werden diese Informationen nicht zwischen Tags gesetzt, sondern im Attribut selbst eingepflegt. Ein Attribut beginnt mit einem „<“, gefolgt vom Attributtitel, den Informationen und endet mit einem „/>“. Attribute können wohl mehrere Informationen beinhalte, können aber keine Unterelemente oder Unterattribute enthalten. [vgl. Helmut Vonhoegen 2018, XML S. 60].

Wie im oberen Kapitel erwähnt, wird XML sowohl für die Konfigurationsdatei als auch für die Datenpakete selbst verwendet. Dies sind wohl beides XML-Strukturen, die für die Kommunikation relevant sind, müssen aber getrennt voneinander betrachtet werden. Die Konfigurationsdatei enthält die wichtigsten Parameter, die für den Verbindungsaufbau benötigt werden und gibt zudem an, welche Struktur die versendeten Pakete haben sollen. Beim Aufbau der Konfigurationsdateien muss sich an die vorgeschriebene Standardform gehalten werden und die Dateien müssen im Verzeichnis *C:\KRC\ROBOTER\Config\User\Common\EthernetKRL* abgelegt werden. Die Standardform wurde zuvor schon in Abbildung 4-3 als Baumstruktur gezeigt und ist im Folgenden nochmal als XML-Struktur aufgeführt:

```
<ETHERNETKRL>
    <CONFIGURATION>
        <EXTERNAL></EXTERNAL>
        <INTERNAL></INTERNAL>
    </CONFIGURATION>
    <RECEIVE>
        <ELEMENTS></ELEMENTS>
    </RECEIVE>
    <SEND>
        <ELEMENTS></ELEMENTS>
    </SEND>
</ETHERNETKRL>
```

Programmcode 1: XML, Grundstruktur der EthernetKRL-Konfiguration

Im Zweig CONFIGURATION werden die Parameter, die für Verbindungsaufbau relevant sind, eingetragen. Dabei wird nach Intern und Extern unterschieden.

Innerhalb des Elements EXTERNAL wird eingetragen, ob das externe System als Server oder Client fungiert, welche IP-Adresse es besitzt und über welchen Port die Verbindung aufgebaut werden soll.
In INTERNAL werden dementsprechend die Verbindungsparameter für die EKI-Schnittstelle festgelegt. Unter anderem können darin die Speichereinstellungen definiert werden und falls die Steuerung als Server agiert, muss hier auch die IP-Adresse und der Port hinterlegt werden. Das Kindelement ENVIROMENT wird in Kapitel 5.1 noch näher behandelt.
Der zweite große Zweig ist RECEIVE. Das einzige direkte Kind von RECEIVE legt fest, ob eine XML-Struktur (<XML></XML>) oder ob binär (<RAW></RAW>) empfangen werden soll. Die Konfiguration für RAW wird hier nicht näher betrachtet. Bei der Wahl von XML wird mittels XPath die Struktur der Datenpakete definiert. Jeder zu empfangende Informationsträger wird in einem separaten Element definiert. Innerhalb des jeweiligen Attributs *Tag* wird die Baumstruktur dabei im Adresssyntax von XPath beschrieben. Ein weiteres Attribut *Type* dient der Beschreibung des Datentyps. Dieser kann STRING, REAL, INT, BOOL und FRAME sein. Optional kann für jedes Element noch das Setzen von Flags oder die Art des Speicherauslesens definiert werden. Um die Information als Attribut zu empfangen, muss vor der letzten Bezeichnung in der Adresse ein @ gesetzt werden.
Im letzten großen Zweig SEND werden die zu versendenden Pakete definiert. Ähnlich wie bei RECIEVE wird hier auch XPath benutzt. Allerdings wird hier nur das Attribut *Tag* definiert. Weitere Informationen zur Konfiguration können in der KUKA Dokumentation Kuka.EthernetKRL 2.2 nachgelesen werden.

Der Versand der XML-Pakete hält sich im Normalfall an die in der Konfigurationsdatei festgelegte Struktur. Dabei kann mit der Funktion *EKI_Send()* entweder die gesamte Struktur oder ein Teil der Struktur versendet werden. In die Funktion kann aber auch eine unabhängige XML-Struktur als Zeichenkette geschrieben werden, die dann genau so versendet wird. Der Funktion *EKI_Send()* müssen zwei Parameter übergeben werden, die beide in Anführungszeichen gesetzt

werden müssen. Zum einen der zu verwendende EKI-Kanal und zum anderen der Pfad bzw. die Zeichenkette.

Da für das Empfangen der Daten vorher ein Speicherplatz reserviert werden muss, ist das Empfangen und Auslesen der Daten im Gegensatz zum Senden an die XML-Struktur der Konfigurationsdatei gebunden. Zu jedem Datentyp muss die passende EKI_Get-Funktion verwendet werden. Bspw. für eine reale Zahl *EKI_GetReal()*. Diesen Funktionen müssen drei Parameter übergeben werden. Analog zu *EKI_Send()* sind die ersten beiden der Kanal und der Pfad. Als drittes wird eine zuvor deklarierte und initialisierte Variable übergeben, der die Werte oder die Zeichen zugeordnet werden können. In Kapitel 5.2 und 5.2.3 wird die Anwendung dieser Funktionen näher beschrieben.

4.1.3 Verbindungskonfiguration

Genauso wie in der Bachelorthesis „Gestengesteuerte Mensch-Roboter-Kollaboration“ (im folgenden GMRK) wurde die Verbindung zwischen Steuerung und PC zunächst über den EKI-Testserver und mit dem KRL-Programm XMLCallBack aufgebaut. Dieser Verbindungsaufbau ist ausführlich in der Thesis beschrieben, weshalb hier nur auf die Unterschiede in der Konfiguration eingegangen wird.

Statt den PC direkten über die Ethernetschnittstelle der Steuerung anzuschließen, wird in dieser Arbeit die Verbindung zur KRC4 über einen Router aufgebaut. Um das zu erreichen, müssen beide Verbindungspartner eine feste IP-

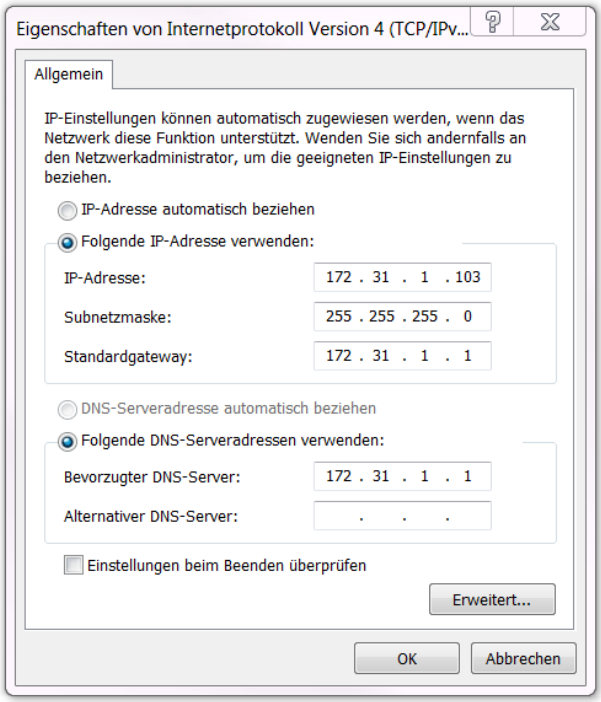


Abbildung 4-4: Einstellung der IPv4 Eigenschaften

Adresse zugewiesen bekommen. Da die Kommunikation über den Router stattfindet, muss seine IP-Adresse als Standardgateway eingestellt werden.

Unter Windows muss man dafür in das Netzwerk- und Freigabecenter, um die entsprechende Ethernetverbinung auszuwählen. Unter „Eigenschaften“ kann man dann die Eigenschaften von IPv4 öffnen. Diese sind in Abbildung 4-4 gezeigt. Möchte man gleichzeitig noch mit dem PC aufs Internet zugreifen, kann die IP-Adresse des Routers in die Zeile des Bevorzugten DNS-Servers eingetragen werden.

Bei der Netzwerkkonfiguration der Steuerung wird im Gegensatz zur Arbeit GMRK, nicht das Interface „virtual6(EKI)“ zur Kommunikation genutzt, sondern „vitual5(KLI)“. Da dieses zum Verbindungsaufbau mit WorkVisual genutzt wurde, ist die feste IP sowie das Subnetz schon eingetragen und es muss nur noch das Standardgateway hinzugefügt werden. Die smartPAD-Ansicht der Netzwerkkonfiguration ist in Abbildung 4-5 dargestellt. Wie bei GMRK muss für den Verbindungsaufbau ein Port im Reiter NAT hinzugefügt werden. In dieser Arbeit wurde der Port 54610 hinzugefügt.

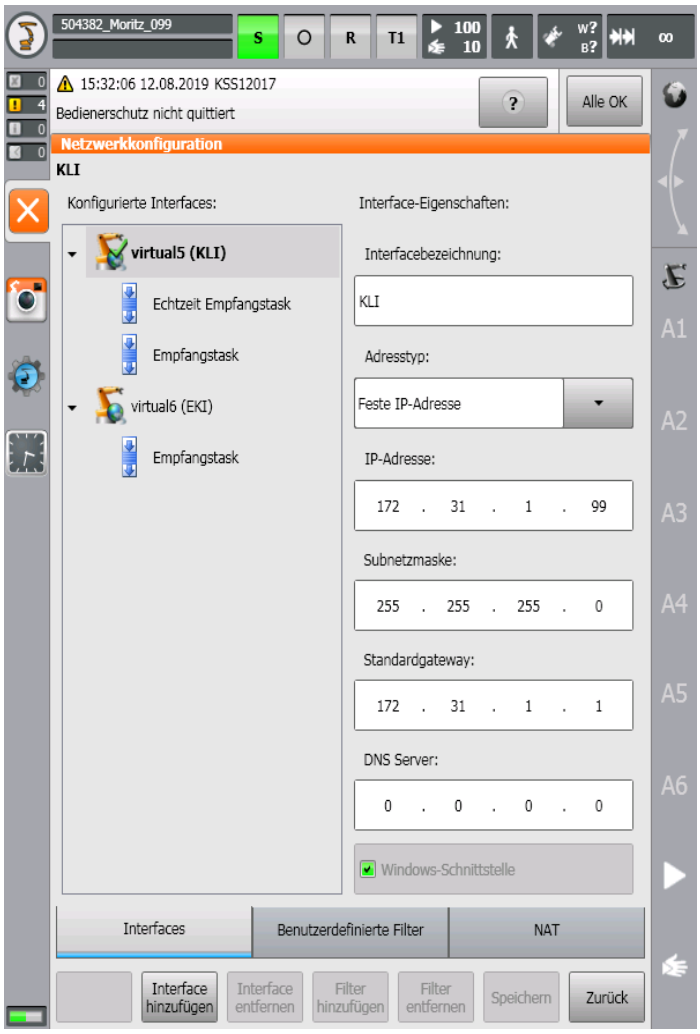
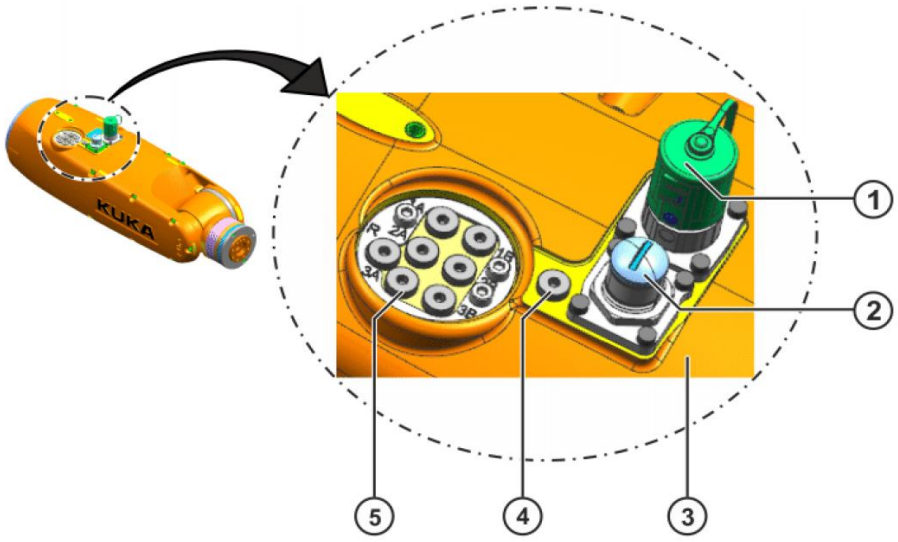


Abbildung 4-5: Netzwerkkonfiguration der Steuerung

4.2 Schnittstellen des Laserscanners

4.2.1 Anschlüsse des scanCONTROL 2950-100

Der scanCONTROL 2950-100 besitzt zwei Anschlüsse. Die erste Anschlussmöglichkeit ist eine 12-polige Multifunktionsbuchse mit Gewinde für einen Schraubstecker. Über die Multifunktionsbuchse kann sowohl die Stromversorgung als auch der Datentransfer bewerkstelligt werden. Zusätzlich ist eine RS422-Schnittstelle integriert, die unter anderem zur Synchronisation mehrerer Scanner verwendet werden kann [vgl. Micro-Epsilon, Betriebsanleitung scanCONTROL 29xx S.28].



1 Anschluss X41
2 Anschluss XPN41
3 Zentralhand
4 Luftleitung AIR2
5 Luftanschlüsse

Abbildung 4-6: Agilus Zentralhand mit Lupenansicht der Schnittstelle A4
[KUKA KR Agilus sixx Betriebsanleitung 2014, S.110 Abb.6-8]

Die 12-polige Buchse kann mit dem mitgelieferten PC2600/2900-5 Multifunktionskabel und einem 651springtec Stecker an den 12-poligen Anschluss X41 angeschlossen werden. Der Anschluss X41 befindet sich, wie in Abbildung 4-6 gezeigt, auf der Oberseite der Zentralhand des Manipulators. Der große Vorteil bei der

Verwendung dieser Schnittstelle ist die ab diesem Punkt intern verlaufende Kabelführung zur KR-C4-Steuerung. Dadurch müsste das Anschlusskabel für den Scanner nicht außen am Roboterarm entlanggeführt werden. Dadurch wird die Wahrscheinlichkeit von Bruch- oder Rissschäden des Kabels enorm reduziert.

Die zweite Anschlussmöglichkeit an den Scanner ist eine 8-polige Ethernet-Schnittstelle, die ebenfalls ein Gewindestecksystem besitzt. Diese Schnittstelle unterstützt sowohl eine 100Mbit Übertragung, wie auch eine Gbit Übertragung, wobei der Gigabit-Verbindung bei hohen Datenraten zwangsweise der Vorzug gegeben wird. [vgl. Micro-Epsilon, Betriebsanleitung scanCONTROL 29xx S.32]. Zum einen kann die Stromversorgung über die Multifunktionsbuchse erfolgen und zum anderen unterstützt der Scanner eine Stromversorgung über PoE (Power over Ethernet).
Da der Manipulator keine Ethernet-Schnittstelle an der Zentralhand besitzt, muss ein Kabel entlang des Manipulators geführt werden, um den Scanner anschließen zu können. Dies erhöht dementsprechend die Risiken von Materialschäden am Kabel. Um zumindest dem Kabelbruch vorzubeugen, wird das robotertaugliche Gigabit-Ethernet-Kabel SCR2600/2900-15 verwendet. Dieses besitzt auf der einen Seite einen 8-pol. Schraubstecker, auf der anderen Seite einen RJ45 Stecker und ist darauf ausgelegt, viele Biegewechsel zu bewältigen.

Der Hauptgrund für die Wahl einer Verbindung über Ethernet ist, dass mit dem Ethernet-Kabel in Kombination mit einem PoE-Switch bzw. einer PoE-Netzkarte eine quasi „Plug and Play“-Lösung geboten wird. Da dies die einfachste und schnellste Lösung war und zudem problemlos funktionierte, wurde die Verbindung über die Multifunktionsbuchse weder eingerichtet noch getestet. Würde nämlich die X41 Buchse verwendet werden, müssten die Scandaten über die Steuerung an den PC gesendet werden, was zu einer zusätzlichen Zeitverzögerung und erhöhtem Fehlerpotential führen würde.

Bei der bestehenden Verbindung über Ethernet muss jedoch bei jeder neuen Trajektorie genaustens darauf geachtet werden, dass weder das Kabel noch der Anschluss durch mögliches Ausreißen beschädigt werden.

4.2.2 scanCONTROL Windows SDK 3.7

Um den Scanner direkt verwenden zu können, bietet Micro-Epsilon „Configuration Tools“ zur Bedienung an. Darin sind die gängigsten Messaufgaben hinterlegt und es müssen keine zusätzlichen Treiber installiert werden. Zur Erstellung von dreidimensionalen Daten kann die Software „3D-View“ verwendet werden. Hierbei wird die dritte Dimension durch Triggerung, Encoderdaten oder durch die Angabe der Relativgeschwindigkeit zum Objekt berechnet [vgl. Micro-Epsilon, 3D-View: Inte raktive 3D-Visualisierungssoftware].
Da der Quellcode von keinem der beiden Softwares zur Verfügung steht, würde sich die Einbindung in den Workflow sehr umständlich gestaltet, weshalb die Wahl auf das Windows SDK fiel.

Bei einem Software Development Kit (SDK) handelt es sich um eine Zusammenstellung von Tools und Dokumentationen, die das Entwickeln einer Software auf einer bestimmten Plattform oder einem speziellen Gerät erleichtern soll [vgl. Golem Media GmbH, SDK - Software Development Kit].
Das SDK, das Micro-Epsilon zur Verfügung stellt, beinhaltet mehrere Bibliotheken, dazugehörige Dokumentationen und Beispielprogramme zum Ansprechen der Laserlinienscanner aus der Produktionsreihe scanCONTROL. Hierbei wird sowohl C# als auch C++ unterstützt. Um die Scanner auch mit Python ansprechen zu können, sind außerdem python-bindings für C++ enthalten. Mithilfe von Bindings können Bibliotheken benutzt werden, die in einer anderen Programmiersprache geschrieben wurden. Für C und C++ ist ab Python 2.5 die Bibliothek ctypes integriert, die Funktionen beinhaltet, welche das Aufrufen von DLLs und geteilten Bibliotheken ermöglicht. Zudem besitzt sie mit C und C++ kompatible Datentypen [vgl. Python Software Foundation, ctypes - A foreign function library for Python]. Um nun die C++-

Bibliothek LLT.dll von Micro-Epsilon mit Python benutzen zu können, ist zusätzlich die Bibliothek pyllt aus dem Python-Binding der SDK erforderlich.
Als zusätzliche Hilfestellung bietet Mirco-Epsilon kostenlos das scanCONTROL Developer Tool an. Mit diesem lässt sich der Laserscanner einfach ansteuern und zudem wird einem die Funktion der DLL angezeigt, wenn die Maus über das entsprechende Funktionsfeld gehalten wird.

4.3 Python als Bindeglied

4.3.1 Ansteuerung des Scanners und verarbeiten der Daten

Als Basisprogramm zur Ansteuerung wird das Pythonskript *full_profiles_poll.py* aus dem Beispielordner der SDK verwendet. Dieses Programm deklariert zunächst alle benötigten Variablen in C++ kompatiblen Datentypen mithilfe von ctypes. Danach wird die Instanz hLLT erstellt und anhand dieser eine Verbindung mit dem Scanner aufgebaut. Nach dem Konfigurieren der Scharfeinstellung wird der Datentransfer eingeleitet und der Scanner für eine Zehntelsekunde vorgewärmt. Ist der Aufwärmprozess beendet, wird der im Scanner integrierte Buffer ausgelesen und die darin enthaltenen Daten in X-/Z-Koordinaten und Intensität konvertiert. Im Anschluss wird ein Zeitstempel erstellt, der Transfer beendet, die Verbindung abgebrochen und die Geräteinstanz entfernt. Zum Schluss werden die gescannten Daten in zwei Graphen dargestellt.
Um dieses Programm für das dreidimensionale Scannen nutzbar zu machen, müssen ein paar Anpassungen durchgeführt werden. Dabei gilt es in erster Linie, zwei Hauptprobleme zu lösen. Zum einen ist eine konstante Profilabfrage zu erstellen. Dies gelingt relativ einfach mithilfe einer while-Schleife, die zunächst durchlaufen wird, bis die Zählervariable den gewünschten Wert erreicht hat und im späteren Verlauf, bis sich die Orientierung der Fahrt ändert. Hierbei muss beachtet werden, dass bei zu schneller Folgeabfrage der Buffer noch nicht wieder beschrieben wurde. Ist dies der

Fall, erzeugt die in *full_profiles_poll.py* einprogrammierte if-Abfrage eine Fehlermeldung:

```
ret = llt.get_actual_profile(hLLT, profile_buffer, len(profile_buffer),
                        llt.TProfileConfig.PROFILE, ct.byref(lost_profiles))
if ret != len(profile_buffer):
    raise ValueError("Error get profile buffer data: " + str(ret))
```

Programmcode 2: Python, Profile aus Buffer auslesen

Eine Anfrage bei Micro-Epsilon ergab jedoch, dass es sich hierbei weniger um einen Fehler als um einen Hinweis handelt. Nichtsdestotrotz wird das Programm damit gestoppt, weshalb die if-Abfrage entfernt werden muss. In diesem Fall wird 1/Frequenz lange gewartet und der Buffer danach wieder ausgelesen. Dadurch erhält man eine konstante Profilabfrage mit vernachlässigbar kleinen und seltenen Lücken [vgl. Anhang 5.1 Schriftverkehr; Daniel Raucher].

Die zweite Herausforderung ist das Hinzufügen einer weiteren Dimension. Da die Scannerdaten in eine Punktlinie mit X- und Z-Koordinaten konvertiert werden, bietet es sich an, die zusätzliche Dimension in Y-Richtung hinzuzufügen, um eine Punktwolke in kartesischen Koordinaten zu erhalten. Geht man davon aus, dass sich der Scanner relativ zum zu scannenden Objekt mit einer konstanten Geschwindigkeit in Y-Richtung bewegt, ist der Y-Wert der Punktprofile anhand der vergangenen Zeit leicht zu berechnen.

Die angefallenen Profile können somit zusammen mit den errechneten Y-Werten in einen Speicher geschrieben werden. Dabei gibt es zwei Möglichkeiten. Entweder werden die Daten im Arbeitsspeicher vorgehalten oder in einem asynchronen Thread auf einen Datenträger geschrieben. Um die Datenmenge nicht vom vorhandenen Arbeitsspeicher beschränken zu lassen, wurde ein Schreibthread eingeführt. Diesem Thread werden in jedem Schleifendurchlauf die X-, Z- und Intensitäts-Arrays mit dem entsprechenden Y-Wert übergeben. In einer for-Schleife, die so lange durchlaufen wird, wie das Array lang ist, wird reihenweise additiv in eine ASC-Datei geschrieben. Die Spalten sind dabei X, Y, Z und die Intensität.

Ein Nachteil dieser Methode ist, dass das einzelne Auslesen der Profile aus dem Scannerbuffer nur bei geringeren Abtastfrequenzen gut funktioniert. Ab Frequenzen von 1000Hz wird empfohlen, die Profile in Containern zu übertragen. Auf die Übertragung von Containern wurde jedoch verzichtet, da bei höheren Frequenzen und den daraus resultierenden kleineren Messfeldern, kürzeren Belichtungszeiten und höheren Verfahrgeschwindigkeiten die Qualität der Scans bei den meisten Oberflächen signifikant sinkt.

4.3.2 Threading in Python

Ein Programm kann einen oder mehrere Prozesse gleichzeitig ausführen. Werden mehrere Prozesse gleichzeitig ausgeführt, kann ein dynamischeres Programm erstellt werden. Dies wird oft bei der Erstellung von GUI-Anwendungen gemacht, da die Oberfläche auch weiter Informationen anzeigen soll, wenn im Hintergrund andere Arbeiten verrichtet werden. Häufiges Beispiel dafür ist ein Ladebalken, der den Fortschritt anzeigt. Auch kann die Geschwindigkeit von Anwendungen enorm erhöht werden, wenn mehrere Aufgaben gleichzeitig durchgearbeitet werden. Dies kann auch auf unterschiedlichen Prozessorkernen geschehen, um die Hardwareressourcen besser ausnutzen zu können. Die Prozesse selbst werden vom Betriebssystem verwaltet. Problematisch ist dabei jedoch, dass die Prozesse voneinander sehr abgekapselt sind. Dies erschwert die Kommunikation und Synchronisation untereinander. Einen Kompromiss beim Parallelisieren bietet dabei das Threading. Jeder Prozess besteht standardmäßig aus einem Thread. In einem Prozess können jedoch auch mehrere Threads ausgeführt werden. Dies geschieht allerdings nur pseudoparallel. Im Gegensatz zum Parallelisieren mit Prozessen können alle in einem Prozess befindlichen Threads auf dieselben globalen Variablen zugreifen. Dadurch gestaltet sich das Synchronisieren einfacher [vgl. Python 3 Das umfassende Handbuch, S.548-549].

Selbst wenn vom Programm mehrere Threads ausgeführt werden, bedeutet das nicht zwangsweise, dass es dadurch schneller wird. Wie viele Threads real gleichzeitig ablaufen können, hängt vom Prozessor, dem Betriebssystem und der genutzten

Programmiersprache ab. Normalerweise kann pro Prozessorkern nur ein Thread gleichzeitig ausgeführt werden. Manche Prozessoren bieten durch bspw. Hyper-Threading die Möglichkeit, zwei Threads von einem Kern gleichzeitig durchzuführten. Um solches Multithreading der Hardware auch nutzen zu können, muss dies allerdings sowohl vom Betriebssystem als auch von der Programmiersprache unterstützt werden.

„In CPython gibt es aus technischen Gründen derzeit leider keine Möglichkeit, verschiedene Threads auf verschiedenen Prozessoren oder Prozessorkernen auszuführen. Dies hat zur Folge, dass selbst Python-Programme, die intensiv auf Threading setzten, nur einen einzigen Prozessor oder Prozessorkern nutzen können.“ [Python 3 Das umfassende Handbuch, S.550]
Das gleichzeitige Ausführen von mehreren Threads wird vom „global interpreter lock“ (auch GIL) verhindert. Dies ist erforderlich, da das Speichermanagement von CPython nicht auf reales Multithreading ausgelegt ist [vgl. wiki.python.org/moin/GlobalInterpreterLock].

Python bietet zur Parallelisierung unter anderem die Module threading und multiprocessing an. Wie es sich vom Namen schon ableiten lässt, bietet das Modul threading die Möglichkeit, in Python Threads zu erstellen und zu managen. Analog dazu bietet multiprocessing Ähnliches mit Prozessen. Da der Zugriff auf gemeinsame Variablen die Anwendungen in dieser Arbeit sehr vereinfachen, wurde in den Programmen das Modul threading statt multiprocessing benutzt. Auch wenn multiprocessing Werkzeuge zur Kommunikation untereinander bietet, so ist dies mit threading doch wesentlich leichter zu bewerkstelligen. Darum wird in dieser Arbeit auch nur das threading näher behandelt.
Das Modul threading basiert auf dem Modul *_thread*. Der Vorteil besteht darin, dass threading eine objektorientierte Programmierschnittstelle und einige Zusatzfunktionen bietet. Bei *_thread* werden die verschiedenen Threads nicht wie Objekte, sondern ähnlich wie Funktionen behandelt.

Die Erstellung eines neuen Threads wird mit der Funktion *start_new_thread()* wie folgt durchgeführt:

```
_thread.start_new_thread(writ, (nx, ny, nz, intensities, W))
```

Programmcode 3: Python, Thread starten

Dieser Funktion wird zum einen die im Thread auszuführende Funktion ohne Klammern übergeben und zum anderen in Klammern die Werte, die diese Funktion benötigt. Diese einzelne Zeile reicht aus, um einen neuen Thread zu eröffnen. So ist *start_new_thread* auch das einzige Werkzeug aus dem *_thread* Modul, das in dieser Arbeit verwendet wird. Weitere Informationen über die weitumfassenden Module *_thread* und *threading* sind der Dokumentation von Python zu entnehmen.

4.3.3 Socket-Programmierung mit Python

Socket-Programmierung im Allgemeinen dient der Kommunikation zwischen verschiedenen Plattformen. Hierbei dienlich ist der Datagrammdienst IP, der zwei große Untergruppen als Transportschicht besitzt [vgl. informatik.uni-hamburg.de/TKRN/world/lernmodule/LMvk/Popup/IP.htm]. Zum einen das UDP und zum anderen das TCP. Beim UDP kann weder garantiert werden, ob die Datenpakete beim Empfänger ankommen, noch ob sie das in der richtigen Reihenfolge tun. Der große Vorteil bei diesem Protokoll ist jedoch die mögliche Geschwindigkeit der Übertragung. Bisher sind Echtzeitanwendungen verschiedener Geräte, die über Ethernet miteinander verbunden sind, nur mit UDP möglich. Hierzu ist jedoch ein tolerantes Programm von Nöten, dass die eventuell auftretenden Fehler in der Kommunikation ausgleicht.
Das sichere aber verhältnismäßig langsame TCP bildet das Gegenstück zu UDP. Durch den Aufbau einer Verbindung wird gewährleistet, dass die Datenpakete vollständig und korrekt beim Empfänger ankommen. Sollte dennoch ein Paket fehlen oder fehlerhaft sein, fordert das Protokoll automatisch zum Wiederholen der Sendung auf [vgl. Python 3 Das umfassende Handbuch, S. 630].

Beide Protokolle eignen sich für eine Server-Client-Kommunikation über Ports. Python 3 beinhaltet standardmäßig das Modul Socket, in dem Funktionen zum Aufbau dieser beiden Protokolle zur Verfügung gestellt werden. Da die Wahl seitens der Steuerung nicht auf RSI sondern auf EKI viel, wurde für den Python-Server TCP/IP verwendet, da nur dies vom EKI-Client unterstützt wird.
Im Folgenden sind die essenziellen Codezeilen für den TCP/IP-Servers in Python aufgezeigt:

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((socket.gethostname(), 59152))
s.listen(1)
conn, addr = s.accept()
while True:
    data = conn.recv(1024)
    if not data: break
conn.close()
```

Programmcode 4: Python, TCP/IP Server

Das Initialisieren des Socket geschieht hierbei in den ersten zwei Zeilen. In der ersten wird mit AF_INET und SOCK_STREAM definiert, dass der Socket mit dem IPv4 und dem Transmission Control Protokoll (TCP) erstellt werden soll. Mit der Funktion *bind()* in der zweiten Zeile, wird die IP-Adresse und der Port übergeben. Die Funktion *socket.gethostname()* ermittelt automatisch die IP-Adresse des verwendeten PCs und der Port kann ab 49152 problemlos willkürlich gewählt werden. Mit *listen()* wird die Anzahl der Verbindungen festgelegt und danach wartet das Programm mit *accept()* auf eine Verbindung. Ist diese aufgebaut, wird *conn* der Socket übergeben, *addr* erhält das Adressobjekt des Clients und die Endlosschleife *while-True* wird ausgeführt. In dieser werden solange Daten aus dem Buffer ausgelesen, bis keine Daten mehr empfangen werden und die Schleife dadurch abgebrochen wird [vgl. Python 3 Das umfassende Handbuch, S. 630-631].

4.4 Kommunikationsprinzip

Da das Kommunikationsprinzip dem der Bachelorthesis GMRK von Herrn Shafaq, sehr ähnelt, bietet es sich an, die Darstellungen analog dazu zu gestalten. Das Äquivalent zur Abbildung 3-8 aus der Arbeit von Herrn Shafaq ist dabei Abbildung 4-7 und das Sequenzdiagramm der Abbildung 3-9 ist vergleichbar mit Abbildung 4-8.

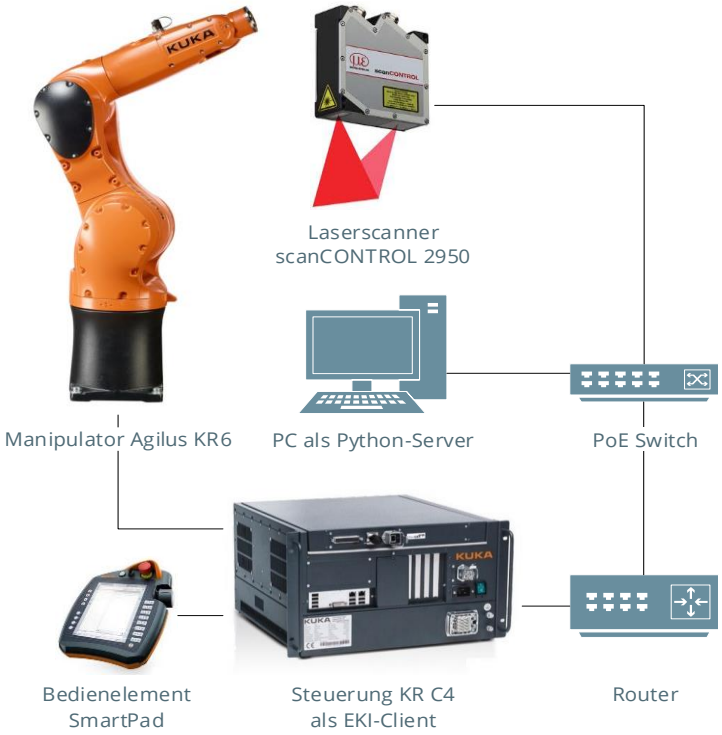


Abbildung 4-7: Aufbau und Vernetzung

Abbildung 4-7 zeigt die Hardwarekomponenten und deren Vernetzung untereinander. Bis auf die Verbindungen der KUKA-Peripherie mit der Steuerung sind alle Verbindungen mit Ethernet realisiert. Die KR C4 Steuerung ist über den Anschluss X66 mit dem Router verbunden. Somit können alle Computer im lokalen Netzwerk mithilfe von WorkVisual KUKA-Projekte auf der Steuerung bearbeiten.
Der Anschluss des Scanners und des PCs an einen PoE-Switch ist hier rein praktischen Gründen geschuldet und hinsichtlich der Übertragungsgeschwindigkeit nicht ideal. Um die maximale Geschwindigkeit zu erhalten, ist es empfehlenswert, stattdessen eine PoE-Netzkarte zu benutzen und den Scanner direkt anzuschließen. Bei den

Abtastraten, die im Zuge der Arbeit verwendet wurden, ist der auftretende Geschwindigkeitsunterschied jedoch irrelevant.
Der größte Unterschied zum Kommunikationsprinzip der GMRK ist das Verwenden eines Python-Servers mit EKI-Client, statt eines EKI-Servers mit Matlab-Client. In dieser Arbeit wird der Server mit Python programmiert, da die Socket-Programmierung unter Python flexibler ist als die in der EKI gebotenen Funktionen des Serverbetriebs. Somit müssen bspw. bei einem Verbindungsabbruch nur die KRL-Programme neu gestartet werden und nicht der gesamte Server, vorausgesetzt der Abbruch wurde nicht von der Python-Seite ausgelöst.

Im Sequenzdiagramm in Abbildung 4-8 wird die Kommunikation der Komponenten untereinander und die Interaktion mit dem Anwender gezeigt. Zunächst muss dieser den Python-Server starten und die Eingabe der Daten in der GUI vollenden. Sobald dies geschehen ist, verbindet sich das Programm mit dem Laserscanner und konfiguriert diesen mit den Werten aus der Eingabe. Wurde das erfolgreich abgeschlossen, wird ein Socket mit definierter IP-Adresse und Port angefordert und auf den Client gewartet. Wurde die Robotersteuerung erst danach hochgefahren, muss der Benutzer nur noch das Hauptprogramm öffnen. Wenn nicht, muss er zuvor den Submit-Interpreter neu starten. Das sub-Programm initialisiert automatisch einen EKI-Kanal und verbindet sich dann als Client mit dem Server. Das Hauptprogramm nutzt denselben Kanal und kann somit das XML-Datenpacket vom Server empfangen und parsen. Anhand der Objekt- und Fahrtdaten kann die Trajektorie berechnet werden und vom Hauptprogramm abgearbeitet werden. Bei LIN-Fahrten wird nach Erreichen einer gewissen Distanz zum Anfangspunkt eine *$FLAG* gesetzt. Bei PTP-Fahrten wird diese unmittelbar gesetzt. Der Submit-Interpreter fragt den Zustand dieser *$FLAGs* in seiner Routine ab und sendet bei gesetzter *$FLAG* die Positionsdaten als XML-Paket an den Server. Das Paket enthält ebenfalls die Information, ob es sich um eine LIN oder PTP Fahrt handelt. Bei einer LIN-Fahrt wird der Laser eingeschaltet, der Scanvorgang gestartet und es werden durchgängig die gemessenen Profile vom Server ausgelesen. Erreicht die LIN-Fahrt ihren Endpunkt und geht über in eine PTP-Fahrt,

wird dies dem Server mitgeteilt. Der Scan wird dann vom Server gestoppt und der Laser wird ausgeschaltet. Dieser Prozess wird wiederholt, bis der Endpunkt der letzten LIN-Fahrt erreicht wird. Danach werden die Verbindungen getrennt und das Programm beendet.

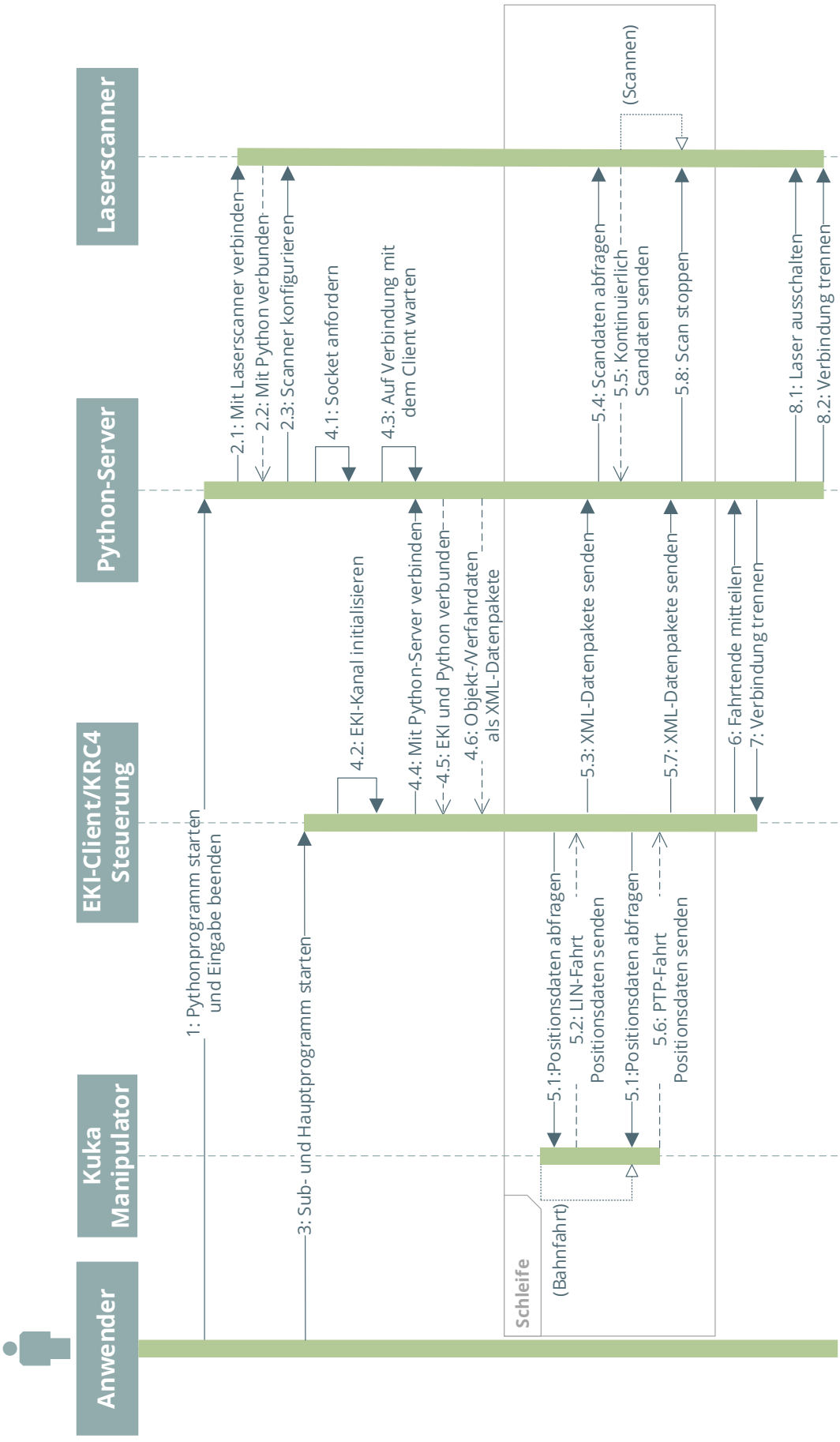


Abbildung 4-8: Sequenzdiagramm zum Kommunikationsablauf

5 Programme und Konfigurationen

5.1 XML-Konfigurationsdatei

Im Folgenden ist die erstellte XML-Konfiguration aufgeführt:

```
<ETHERNETKRL>
    <CONFIGURATION>
        <EXTERNAL>
                <IP>172.31.1.103</IP>
                <PORT>59152</PORT>
        </EXTERNAL>
        <INTERNAL>
            <ENVIRONMENT>Submit</ENVIRONMENT>
            <BUFFERING Mode="FIFO" Limit="512"/>
            <BUFFSIZE Limit="32768"/>
            <ALIVE Set_Flag="1"/>
        </INTERNAL>
    </CONFIGURATION>
    <RECEIVE>
        <XML>
            <ELEMENT Tag="Sensor/Read/PAnzahl" Type="INT"/>
            <ELEMENT Tag="Sensor/Read/Hoehe" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/Breite" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/Laenge" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/Scannerbreite" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/Scannerhoehe" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/Winkel" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/Geschwindigkeit" Type="REAL"/>
            <ELEMENT Tag="Sensor/Read/xyzabc" Type="FRAME"/>
            <ELEMENT Tag="Sensor" Set_Flag="998"/>
        </XML>
    </RECEIVE>
    <SEND>
        <XML>
            <ELEMENT Tag="Robot/Data/ActPos/X"/>
            <ELEMENT Tag="Robot/Data/ActPos/Y"/>
            <ELEMENT Tag="Robot/Data/ActPos/Z"/>
            <ELEMENT Tag="Robot/Data/ActPos/A"/>
            <ELEMENT Tag="Robot/Data/ActPos/B"/>
            <ELEMENT Tag="Robot/Data/ActPos/C"/>
            <ELEMENT Tag="Robot/Winkel"/>
            <ELEMENT Tag="Robot/Orientierung"/>
            <ELEMENT Tag="Robot/Fin"/>
        </XML>
    </SEND>
</ETHERNETKRL>
```

Programmcode 5: XML, Konfiguration der Server-Client-Verbindung

Wie in Kapitel 4.1.2 beschrieben, dient sie zum einen der Einstellung von Verbindungsparametern und zum anderen der Strukturierung der XML-Pakete. Da

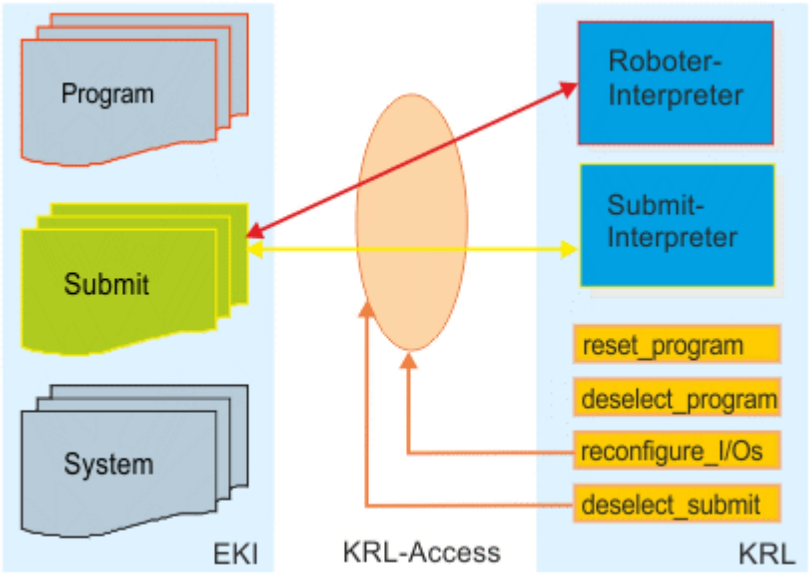


Abbildung 5-1: Verbindungskonfiguration "Submit"
[KUKA Roboter GmBH 2014c, S. 31 Abb. 6-2]

die Steuerung als Client fungiert, muss unter EXTERNAL die IP-Adresse und der Port des Python-Servers hinterlegt werden.

In der internen Konfiguration wird mit BUFFERING *Limit="512"* die maximale Anzahl an Datenelementen eingestellt, die ein Speicher aufnehmen kann. Dies und die Verdopplung des Defalt-Werts der BUFFSIZE sollen einen möglichen Pufferüberlauf und die damit verbundene Unterbrechung der Verbindung verhindern.

In Abbildung 5-1 ist die Submit Konfiguration des ENVIROMENT-Elements dargestellt. Durch diese Konfiguration wird die Verbindung nicht mehr automatisch gelöscht, sobald das Programm zurückgesetzt oder abgewählt wird. Stattdessen wird die Verbindung gelöscht, wenn der Submit-Interpreter abgewählt wird. Wird bspw. vor dem ersten Scannen das Sicherheitssystem aktiviert und der Manipulator dadurch gestoppt, kann nach dem Wiederherstellen der Sicherheit das Programm einfach neu gestartet werden, ohne dass eine neue Verbindung aufgebaut werden muss.

Eine weitere Eigenschaft der Submit-ENVIROMENT ist, dass ein im Submit-Interpreter geöffneter EKI-Kanal parallel auch vom Roboter-Interpreter benutzt werden kann.

Die XML-Struktur der zu empfangenden Datenpakete ist relativ einfach aufgebaut. Sie wird von dem ELEMENT-Attribut *Tag* mittels XPath definiert. Das Wurzel-element ist hierbei *Sensor* mit dem einzigen direkten Kindelement *Read*. Unter *Read* sind alle für die Trajektorie benötigten Daten separat als Kindelement definiert. Bis auf *xyzabc*, das als FRAME definiert ist, besitzen alle Elemente den Typ *REAL*. Auch wenn *Winkel* als Array empfangen wird, hat dies keinen Einfluss auf die Konfigurationsdatei.

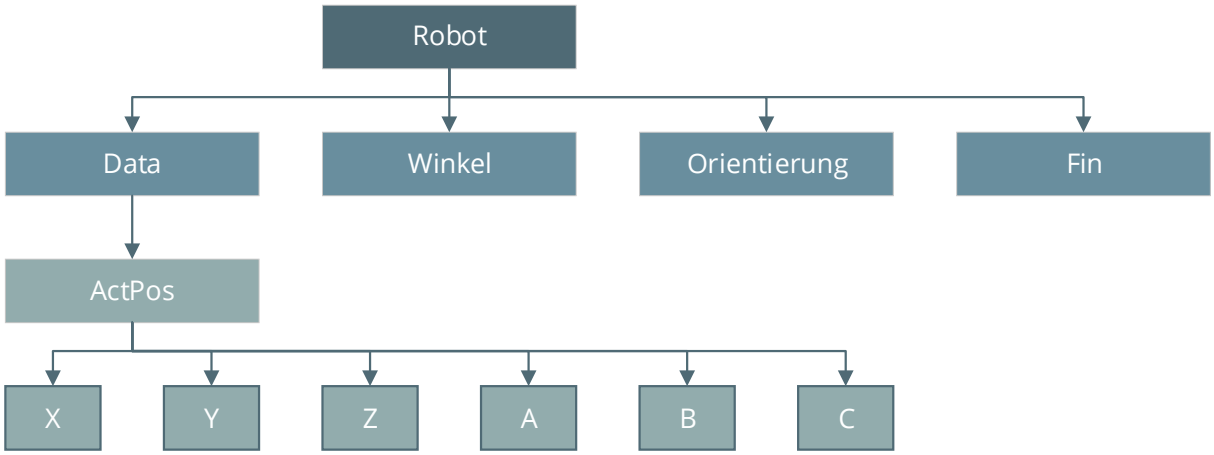


Abbildung 5-2: Baumstruktur der Sendedaten

Die Daten zum Versenden besitzen im Vergleich zu den empfangenen Daten eine etwas komplexere XML-Struktur, die zur Übersicht in Abbildung 5-2 dargestellt ist. Das Wurzelelement *Robot* besitzt vier Kinder, wovon allerdings nur *Data* mit *ActPos* ein Kind besitzt. Die aktuelle Position wird aufgeteilt in die Achsenpositionen und die Winkel zu den Achsen. Diese werden dann als separate Elemente unter *ActPos* versendet. Die Informationen, die aus dem Skript herausgenommen und nicht vom Manipulator abgefragt werden, versendet *Data* als Geschwisterelemente. *Winkel* ist dabei anders als beim Empfangen nur der aktuell gefahrene Winkel und somit kein Array.

5.2 KRL-Programme

5.2.1 Trajektorie

Um ein Objekt mit einem Linienscanner dreidimensional zu vermessen, können verschiedene Relativbewegungen zwischen Objekt und Scanner zum Einsatz kommen. Am einfachsten sind die Scandaten bei gradlinigen Relativbewegungen zu verwerten. Da in dieser Arbeit mit einem 6-Achsenroboter gearbeitet wird, bietet es sich an, die Bewegung mit diesem auszuführen und das Objekt unbewegt zu lassen. Kuka bietet für gradlinige Bewegungen mit konstanter Geschwindigkeit die LIN- und SLIN-Befehle an, wobei die SLIN-Bewegung dabei eine optimierte Variante der LIN-Bewegung darstellt. Um Objekte abscannen zu können, die breiter und länger sind als das maximale Messfeld des Linienscanners, müssen mehrere gradlinige Scanfahrten durchgeführt werden. Komplexere Oberflächen müssen zudem mit verschiedenen Winkeln gescannt werden, um die entstehenden Abschattungen auszugleichen. Das einzelne Erstellen einer solchen Trajektorie für jedes zu scannende Objekt ist sehr aufwändig. Komfortabler ist dagegen eine Trajektorie, die anhand von übergebenen Parametern automatisch erstellt wird.

Um eine Trajektorie zu programmieren, müssen zunächst die abzufahrenden Punkte berechnet werden. Da sich die Fehlersuche und Tests bei KRL-Programmen sehr aufwendig gestalten, wurde die Logik zur Erstellung der Punkte zunächst mit einem Python-Skript programmiert.
Nach dem Definieren der Objektdaten und der Position (Eckpunkte PE) werden einige Variablen für die Berechnung definiert und die Anzahl an benötigten Fahrten berechnet. Der Vorlauf ist dabei eine Überschlagsrechnung, wie lange der Roboter braucht, um die gewünschte Geschwindigkeit zu erreichen, addiert mit einer Pauschale, um dem Scanner genügend Zeit zum Vorwärmen zu geben. Die Anzahl der Fahrten ist mit „j“ in die Y-Richtung und mit „k“ in die X-Richtung aufgeteilt und

wird aus der Objektbreite bzw. -länge und der minimalen Breite des verwendeten Messfeldes inklusive Überlappung gerundet berechnet.

Im Folgenden ist der Code-Abschnitt gezeigt, in dem die Punktkoordinaten der Y-Fahrten berechnet werden:

```
Vorlauf = 1.1*100/500+15 #1.1*Geschwindigkeit^2/Beschleunigung+15
scan = True
Bias = 1
Anzahl = 0
j = int(round(B/ScanBreite+0.5))
k = int(round(L/ScanBreite+0.5))
i = 0
PHx = [PEx[i] + ScanBreite / 2]
PHy = [PEy[i] - Vorlauf]
PHz = H + ScanHoehe
i += 1

while i < j * 2:
    if scan == True:
        PHx += [PHx[i-1]]
        PHy += [PHy[i-1] + Bias*(L + 2*Vorlauf)]
        i += 1
        Bias = -Bias
        scan = False

    elif scan == False:
        PHx += [PHx[i-1] + ScanBreite]
        PHy += [PHy[i-1]]
        i += 1
        scan = True
```

Programmcode 6: Python, Punkte der Y-Fahrten

Nachdem der X- und Y-Wert des ersten Punktes anhand des ersten Eckpunktes, der halben Scanbreite und dem Vorlauf berechnet wurde, werden die nächsten Punkte innerhalb einer while-Schleife errechnet. In dieser Schleife wird zunächst die Variable *scan* abgefragt. Ist diese *True*, handelt es sich um eine Scanfahrt. Dann ist der X-Wert des neuen Punktes gleich dem vorigen und der Y-Wert wird gegenüber dem alten um die Länge des Objekts und dem zweifachen Vorlauf verschoben. Die Richtung der Verschiebung bestimmt dabei der Bias, der nach jeder Scanfahrt sein Vorzeichen wechselt. Ist *scan* bei der Abfrage *False*, wird der Punkt für eine Verbindungsfahrt berechnet, indem der X-Wert und die Scanbreite summiert werden. Dadurch wird eine mäandrisch verlaufende Verfahrlinie generiert. Die Erstellung der Punkte für die Scanfahrten in X-Richtung verläuft analog dazu.

Zum Erstellen der Punkte der angewinkelten Fahrten werden bei den Y-Fahrten die Y-Werte und bei den X-Fahrten die X-Werte kopiert. Den jeweils anderen Werten wird die Messfeldmitte mal dem Sinus des Winkels aufaddiert.

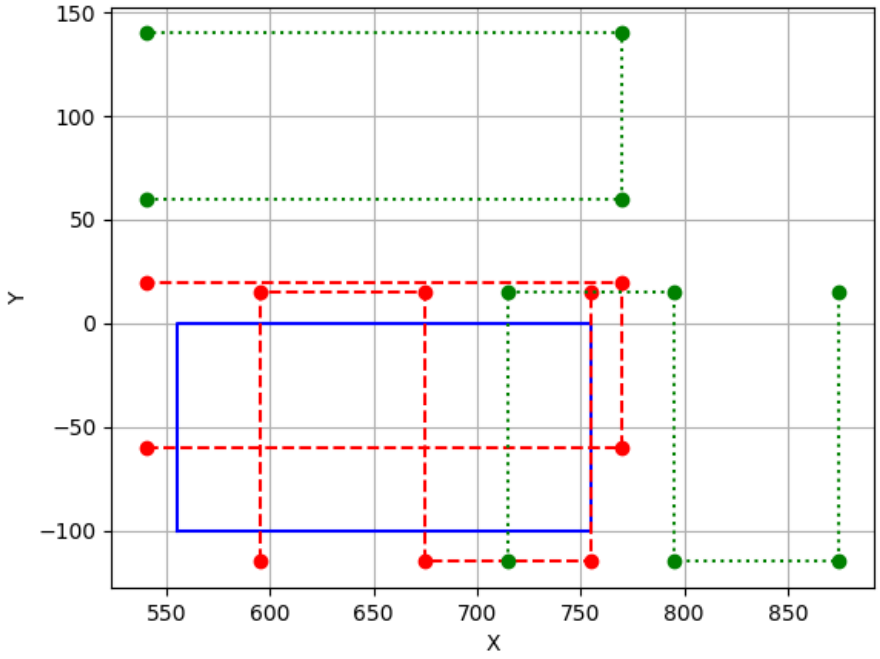


Abbildung 5-3: Punkte der Trajektorie

Gegenüber KRL bietet Python den Vorteil mit bspw. Matplotlib die Punkte wie in Abbildung 5-3 graphisch darstellen zu können. Hierbei wurde ein hypothetisches Objekt mit einer Länge von 100mm und einer Breite von 200mm auf der Eckposition der Standardeinstellungen (X=555, Y=-100) eingefügt. Die Eckpunkte der Packmaße sind in der Darstellung mit blauen Linien verbunden. Die roten Punkte mit gestrichelter Linie stellen die Verfahrpunkte ohne Winkel dar und die grünen Punkte mit gepunkteter Linie geben die Verfahrpunkte mit einem Winkel von 30° an. Durch die graphische Darstellung können Anpassungen im Skript visuell auf Fehler geprüft werden, bevor die Logik in KRL übersetzt wird.

Um die Logik in ein KRL-Programm zu übersetzten, müssen einige Anpassungen durchgeführt werden. Im Gegensatz zu Python können bei KRL die Felder nicht beliebig erweitert werden. Deshalb muss die Punktanzahl am Anfang festgelegt

werden. Da die Punkte nicht grafisch dargestellt werden, müssen die X- und Y-Werte nicht in einzelnen Listen, sondern können in einem FRAME-Feld definiert werden. Darin können auch die Z-Koordinaten und die Winkel zu den Achsen hinterlegt werden, die im Python-Skript keine relevante Rolle spielen. Da sich der Roboter räumlich bewegen und der Scanner passend ausgerichtet werden muss, sind diese zusätzlichen Daten im KRL-Programm unverzichtbar.
Ein weiterer Unterschied zum Python-Programm ist, dass die erstellten Punkte zusätzlich mit Fahrbefehlen verbunden werden müssen. Dabei werden die Scanfahrten mit dem SLIN-Befehl und die Verbindungsfahrten mit dem SPTP-Befehl durgeführt. Um die Verbindung zwischen LIN und PTP-Bewegung flüssiger zu gestalten, können die Verfahrpunkte überschliffen werden. Hierbei fährt der Roboter den Punkt nicht direkt an, sondern bewegt sich nur innerhalb eines definierten Bereichs um den Punkt.
Dieses Überschleifen ist bei einer reinen Trajektorie problemlos. Da der Scanvorgang ausgehend von den Positions- und Fahrtdaten des Roboters eingeleitet werden muss, ergeben sich mit dem Überschleifen allerdings Probleme, die im nächsten Kapitel näher betrachtet werden.

Eine Alternative zur Trajektorie, die aus LIN-Fahrten besteht, ist die Anwendung von zirkularen Fahrten. Das Scannen mit einer kreisförmigen Bewegung ist wesentlich schneller als das Abscannen mit mehreren linearen Fahrten. KUKA bietet mit CIRC in KRL auch die passende Bahnbewegung. Damit der Scanner auch immer zum Objekt gerichtet ist, muss in diesem Fall die Orientierung auf *$ORI_TYPE=#CONSTANT* und der Bezug mit *$CIRC_TYPE=#PATH* auf die Bahnfahrt gesetzt werden. Somit bleibt die Orientierung des TCPs während der Kreisfahrt bezogen auf die Kreislinie gleich. Wie beim Scannen mit den LIN-Fahrten werden die Kreisfahrten mit unterschiedlichen Winkeln mittels PTP-Fahrten verbunden. Analog zur LIN-Fahrt sendet der Submit-Interpreter nach Erreichen der gewünschten Geschwindigkeit und am Ende der Fahrt dem Pythonserver die Positionsdaten. Das Zusammenfügen der empfangenen Daten des Scanners und des Roboters gestaltet sich etwas komplizierter

als bei einfachen LIN-Fahrten. Zunächst muss die aktuelle Position des TCP kontinuierlich aus den empfangenen Positionsdaten, der Geschwindigkeit, der vergangenen Zeit und dem Durchmesser der momentanen CIRC-Fahrt berechnet werden. Anhand der ausgerechneten momentanen Position können dann die Punktdaten der X-Z-Ebene des Scanners vektoriell verrechnet und in eine ASC-Datei geschrieben werden.

Ein weiterer Vorteil der zirkularen Scanfahrt neben der Geschwindigkeit ist die Möglichkeit, große hohle Objekte von innen heraus abscannen zu können. Hierzu muss lediglich die vektorielle Verrechnung angepasst werden, indem sie entgegengesetzt zum Kreismittelpunkt durchgeführt wird. Auch wenn bei einer solchen Scanfahrt genaustens auf die Bewegung des Manipulators geachtet werden muss, ist es im Vergleich dazu mit LIN-Fahrten wesentlich schwieriger umzusetzen.

Das Scannen mit zirkularen Fahrten bringt mit der vorhandenen Hardware aber auch Nachteile mit sich. Ein großes Problem stellt die Komplexität der Bewegung dar. Da die TCP-Position nur am Anfang gesendet wird, sind die restlichen Positionen, die die Grundlage der Berechnung der Punktwolke darstellen, sehr stark von der Genauigkeit der Bahnfahrt, der Konstanz der Geschwindigkeit, der Steifigkeit der Halterung und der Vermessung des TCPs abhängig. Diese Faktoren beeinflussen natürlich auch die Genauigkeit der linearen Scanfahrten, jedoch in geringerem Maße. Zudem können die Ergebnisse einzelner LIN-Fahrten separat abgespeichert und somit besser nachbearbeitet werden. Ein weiteres Problem kann die inhomogene Punktwolke darstellen. Durch das Abscannen mit einem Laserlinienscanner bei kreisförmigen Bewegungen ergibt sich zur Kreismitte hin eine höhere Punktdichte, als nach außen. Beim Umwandeln in eine STL-Datei können dadurch fehlerhafte Oberflächen entstehen.

Wegen der geringeren Fehlerhaftigkeit der Umsetzung wurde in dieser Arbeit die Verfahrmethode mit LIN-Befehlen gewählt.

5.2.2 Haupt und Unterprogramm

Bevor eine Variable initialisiert werden kann, muss diese zuvor in KRL deklariert werden. Das muss im Deklarationsbereich geschehen, der am Anfang des Programms steht. Jedes Programm darf nur eine Deklarationsphase besitzen. Um in einem Programm an unterschiedlichen Stellen deklarieren zu können, muss an dieser Stelle ein Unterprogramm bzw. eine Funktion eingefügt werden. Hierbei müssen die Lebensdauer und die Gültigkeit der Variablen beachtet werden. Wird eine Variable lokal in der SRC-Datei deklariert, so kann sie nur in diesem Programm oder Unterprogramm benutzt werden. Nach dem Erreichen der END-Zeile wird der reservierte Speicherplatz wieder freigegeben. Um auf eine Variable aus einem anderen Programm zugreifen zu können, muss diese in einer .dat-Datei deklariert werden. Dazu kann die lokale .dat oder die *$config.dat* im Systemordner verwendet werden. Wird die *$config.dat* verwendet, so ist die Variable zwangsweise global und alle Programme können diese auslesen und überschreiben [vgl. KUKA Roboter GmBH 2013, Einsatz und Programmierung von Industrierobotern S.180ff.].
Um ein Objekt zu scannen, von dem nur die Packmaße bekannt sind, müssen zunächst die Verfahrpunkte der Trajektorie ermittelt werden. Die benötigte Anzahl von Punkten kann aus der Breite und Länge des Objekts, sowie der Breite eines Scanvorgangs berechnet werden. Da hier viele Größen unbekannt sind, würde es sich anbieten die Länge des Punkte-Arrays/Felds der Anzahl von benötigten Punkten anzupassen. Da die Länge jedoch in der ersten Deklarationsphase nicht abzusehen ist, müsste das Feld in einem Unterprogramm deklariert werden, nachdem die relevanten Größen im Hauptprogramm über EKI empfangen wurden. Diese Methode ist jedoch unter KRL nicht durchführbar, da keine Variablen, sondern nur ganze positive Zahlen zur Definition der Feldlänge verwendet werden können. Um trotzdem so viele Scanfälle wie möglich abdecken zu können, muss die Feldlänge mit möglichst großer Anzahl von Punkten festgelegt werden, die aber nicht unnötig groß sein soll. Hierfür wird die Länge und Breite der Scanbasis durch die minimale Breite des kleinsten Messfeldes inklusive Überlappung geteilt und das Ergebnis verdoppelt. Die kleinste Scanbreite beträgt beim Messfeld Nr.127 ca. 20,8mm. Bei einer 600x800mm Basis und

einer überlappenden Scanbreite von 19mm ergeben sich 74 Fahrten und somit 148 Punkte. Auch die Anzahl der Winkel muss vor der dem eigentlichen Programm in der Deklarationsphase definiert werden und kann somit nicht variabel sein. Hier wurde willkürlich die Anzahl von 19 Winkeln festgelegt.

Das eigentliche Hauptprogramm ist, wie in Abbildung 5-4 zu sehen, relativ einfach aufgebaut. Vor dem Ausführen des Hauptprogramms muss der Submit-Interpreter gestartet und das *sps.sub* Programm angewendet werden. Dadurch ist der EKI-Kanal schon initialisiert. Um beim gleichzeitigen Zugriff auf einen EKI-Kanal Fehler zu vermeiden, ist der Datenaustausch mit dem Server in *Empfangen* und *Senden* aufgeteilt. Das Empfangen wird dabei vom Hauptprogramm übernommen. Die empfangenen Daten werden dann in die Variablen, die mit *Scan_* anfangen geschrieben. Diese wurden in der *$config.dat global* definiert und sind somit von allen Programmen auslesbar und beschreibbar. Beim Parsen werden, bis auf die Winkel und *xyzabc,* alle Werte mit der Funktion *EKI_GetReal()* ausgelesen. Dabei wird der entsprechende Speicherplatz im Buffer wieder freigegeben. Die Winkel werden mit dem Befehl *EKI_GetRealArray()* aus dem Speicher ausgelesen, bis kein gesendeter Winkel mehr vorhanden ist und *xyzabc* wird mit *EKI_GetFrame()* auf einmal ausgelesen [vgl. KUKA Roboter GmBH 2014c, Kuka.EthernetKRL 2.2 S.65].

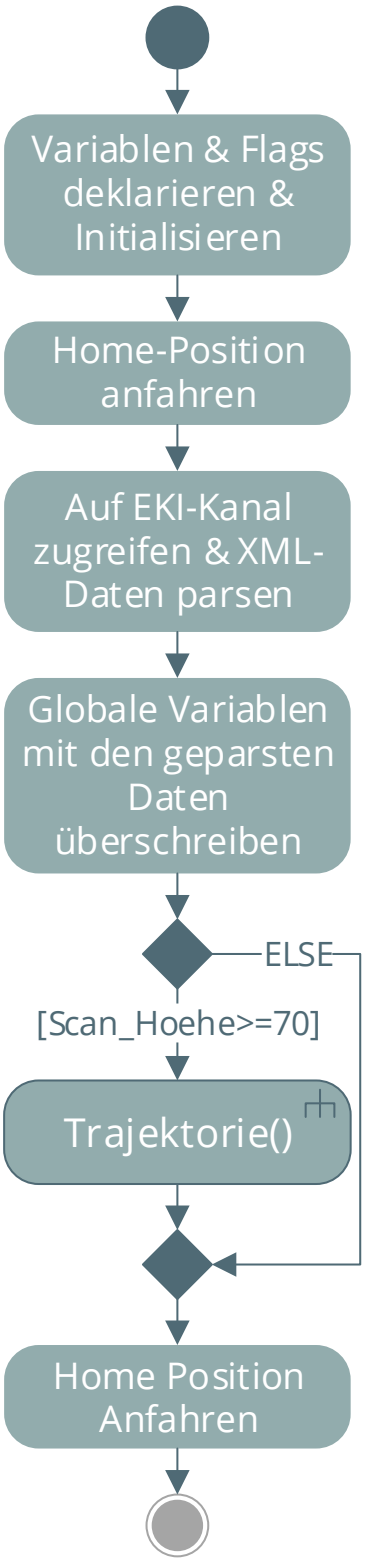


Abbildung 5-4: Aktivitätsdiagramm des KRL-Hauptprogramms

Die Unterfunktion *Trajektorie()* wird nur ausgeführt, wenn das Parsen erfolgreich war und die empfangene *Scan_Hoehe* größer als 70mm ist. Die *Scan_Hoehe* ist die Addition der Objekthöhe und der angegebenen Z-Position. Der Wert 70mm ergibt sich aus der Höhe der Scanplattform (ohne Gummimatte). Um Kollisionen zu vermeiden, wird bei Nichterfüllen der Bedingung das Hauptprogramm beendet und müsste nach Behebung des Problems erneut ausgeführt werden. Sollte eine niedrigere Plattform als Basis eingebaut werden, so müsste diese IF-Abfrage angepasst werden.

Bei der verwendeten Scan-Methode werden während der Trajektorie vor und nach der LIN-Fahrt Flags gesetzt, um eine IF-Abfrage im Submit-Interpreter auszulösen. Wenn *$FLAG[990]==TRUE* oder *$FLAG[991]==TRUE* ist, sollen die aktuellen Positionsdaten, der momentan eingestellte Winkel und die Orientierung der Fahrtrichtung dem Server gesendet werden. Problematisch ist hierbei jedoch der Vorlaufzeiger des Roboter-Interpreters. Dieser führt Zeilen des KRL-Programmcodes aus, bevor der Roboterzeiger diese erreicht. Das wird benötigt, um bspw. das Überschleifen von Punkten zu ermöglichen. Standartmäßig ist der Vorlaufzeiger auf drei Zeilen eingestellt und kann im Programmverlauf unvorhersehbar variieren. Um Punkte überschleifen zu können, muss er mindesten auf 1 eingestellt sein. Ist er das, so werden die Flags kurz vor der LIN-Fahrt gesetzt und es wird eine falsche Richtungsorientierung, sowie eine falsche Position gesendet, die sich irgendwo innerhalb des Überschleifbereichs befindet. Somit können die empfangenen Daten nicht mehr zur Verrechnung der gescannten Daten dienen.
Das lässt sich verhindern, indem der Vorlaufzeiger auf 0 gestellt wird. Dadurch werden die Flags erst gesetzt, wenn die jeweiligen Punkte erreicht werden. Somit würde sowohl die richtige Orientierung als auch die richtigen Positionsdaten gesendet werden. Ein Nachteil hierbei ist jedoch, dass diese Punkte nicht mehr überschliffen werden. Das hat zur Folge, dass jeder Punkt genau angefahren wird und der Roboter jedes Mal kurz anhält. Dadurch erhöht sich bei schnellen Fahrgeschwindigkeiten der Verschleiß des Roboters. Zudem muss der Roboter beim Anfahren zunächst beschleunigen, um auf die gewünschte Geschwindigkeit zu kommen und der Verlauf

dieser Beschleunigung ist in der Regel nicht linear, sondern sinusförmig. Das erschwert die genaue Berechnung der Position des Scanners.
Eine mögliche Lösung dieses Problems besteht darin, Bahnschaltfunktionen zu programmieren. Diese werden benutzt, um bei einem bestimmten Punkt innerhalb einer LIN- oder CIRC-Fahrt einen Ausgang zu setzen. Das Besondere hierbei ist, dass die Fahrt nicht unterbrochen wird und der Punkt durch Entfernung und zeitlichen Versatz eingestellt werden kann. Letzteres ist besonders nützlich, wenn man bspw. einen Kunststoffextruder eine bestimmte Zeit vorheizen muss, bevor Filament ausgegeben werden soll. Im Falle des Scanvorgangs wird jedoch nur der Streckenversatz genutzt, um eine definierte Anfangsposition mit erreichter Betriebsgeschwindigkeit zu senden. Hierzu wird das Standard-Inlineformular leicht modifiziert um nicht *$OUT[1]=TRUE*, sondern *$FLAG[990]=TRUE* zu setzten [vgl. KUKA Roboter GmBH 2013, Einsatz und Programmierung von Industrierobotern S.145ff.].

Das in KRL übersetzte Programm der Trajektorie, die im Kapitel 5.2.1 behandelt wurde, ist als Aktivitätsdiagramm in Abbildung 5-5 dargestellt. Wie schon zuvor erwähnt ist einer der Unterschiede die Verwendung des Frame-Datentyps als Feld. Die einzelnen Zeilen können darin im Gesamten oder separat angesprochen werden. Fürs Erste muss nach dem Variablennamen die jeweilige Zeile in eckigen Klammern folgen. Um dann eine Zelle anzusprechen, muss der Buchstabe der Zelle, getrennt von einem Punkt, dahinter geschrieben werden (Bsp. *P[1].x*). Im Gegensatz zur Größendefinition in der Deklarationsphase, kann die Feldzeile im Programmablauf auch mit einer Variablen oder Formel beschrieben werden. Im Aktivitätsdiagramm ist in der ersten Schleife zu erkennen, wie dies benutzt wird, um bestimmte Zellen in allen relevanten Zeilen des Feldes zu bearbeiten.
Das Unterprogramm ist nach der Deklaration und Initialisierung in 5 *FOLDs* aufgeteilt. Deren Anfänge sind in Abbildung 5-5 als Kommentare markiert und beschrieben. Im ersten *FOLD* wird der erste Punkt für die Y-Fahrten analog zum Pythonprogramm berechnet. Hierfür werden die im Hauptprogramm geparsten

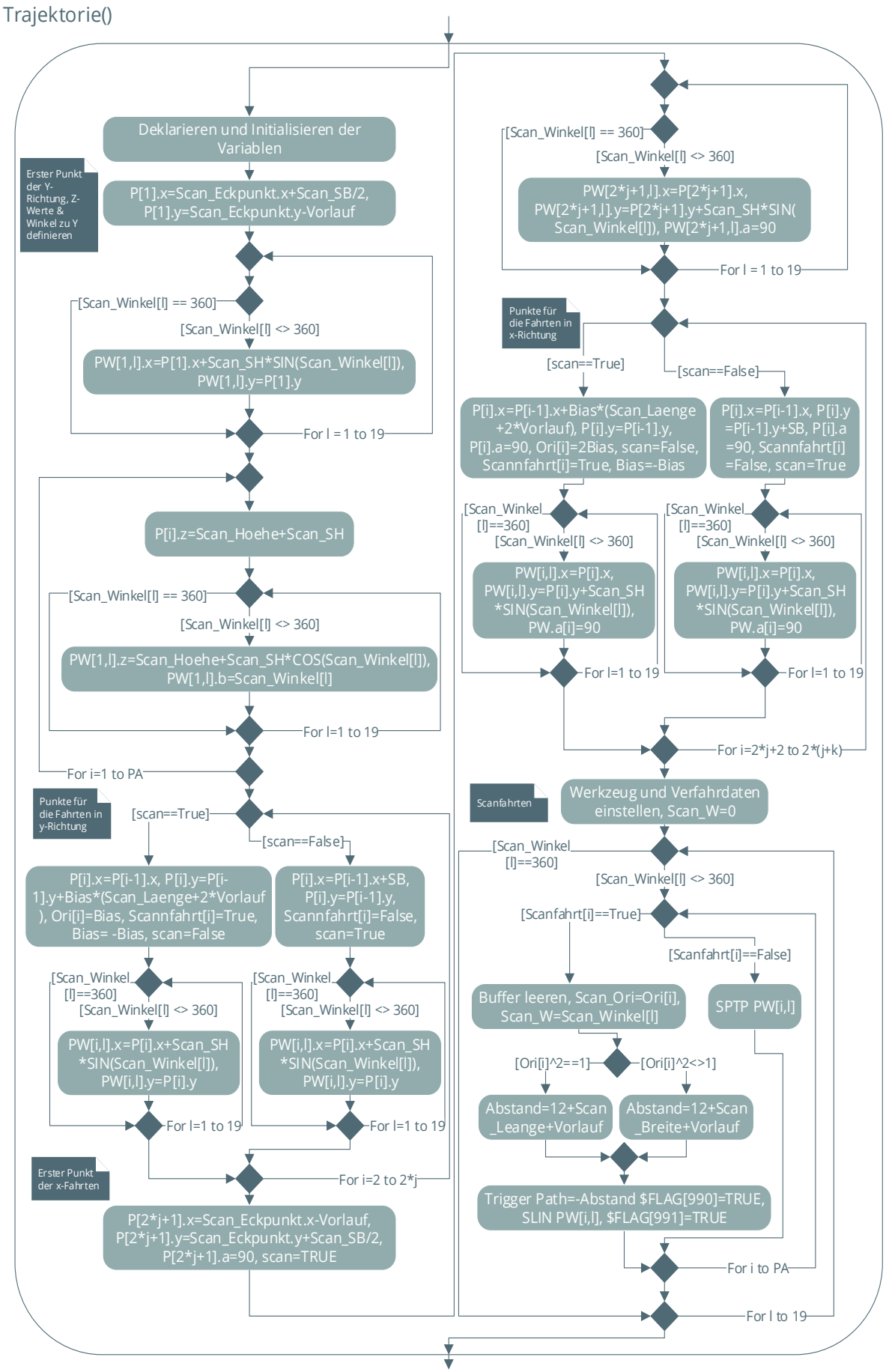


Abbildung 5-5: Aktivitätsdiagramm des Unterprogramms Trajektorie()

globalen Variablen, die mit *Scan_* anfangen, verwendet. Im Vergleich zum Pythonprogramm werden hier jedoch bis zu 19 Winkel berücksichtigt. Alle 19 Winkel werden im Hauptprogramm mit 360 initialisiert und nach dem Parsen als Array werden die jeweiligen Zellen mit den empfangenen Werten überschrieben. Werden weniger als 19 Winkel empfangen, so behalten die übrigen Zellen den Wert 360.

Im zweiten *FOLD* werden, basierend auf dem ersten Punkt, die restlichen Verfahrpunkte der Y-Fahrt berechnet. Hierzu werden in einer For-Schleife abwechselnd die Punkte für die Scan- und Verknüpfungsfahrt berechnet.
Der dritte und vierte *FOLD* berechnen die Punkte für die X-Fahrten nach demselben Schema. Vergleicht man im Aktivitätsdiagramm die Bereiche, fallen die Unterschiede auf. Zum einen sind Vorlauf und die halbe Scanbreite mit einander vertauscht und zum anderen wird der Winkel A bei den X-Fahrten auf 90° gesetzt. Die Winkel um Achsen sind bei Kuka alphabetisch in umgekehrter Reihenfolge benannt worden. C dreht um X, B dreht um Y und A dreht somit um Z.
Während der Punkteberechnung wurde in jedem Schleifendurchlauf der jeweiligen Zelle des Feldes *Scanfahrt* der Zustand *FALSE* oder *TRUE* zugeordnet. Im letzten *FOLD* werden bei *TRUE* die Scanfahrten mit den zuvor beschriebenen Bahnschaltfunktionen ausgeführt. Da sich der Wert beim Schalten mit PATH auf den Zielpunkt bezieht, muss je nach Fahrtrichtung der Wert von *Abstand* angepasst werden. Um die Fahrtrichtung zu ermitteln, wird der Wert von *Ori* abgefragt. Bei Y-Fahrten ist der Wert von *Ori* gleich 1, bzw. -1 bei Fahrten in negative Richtung. Bei X-Fahrten ist er 2 oder -2. Sollte der Zustand der aktuellen Zelle von *Scanfahrt FALSE* sein, wird der nächste Punkt in der Reihe mit einer SPTP-Fahrt angefahren.

In früheren Versionen dieses Programms wurden nur die Punkte ohne Winkel berechnet und abgefahren. Die Punkte mit Winkel kamen erst später dazu. Damit man als Benutzter die Option hat, ohne den Winkel 0° zu scannen, wurde im *FOLD Fahrten* die Fahrten ohne Winkel auskommentiert. Wenn man mit dem Winkel 0° scannen möchte, kann dieser einfach wie die anderen Winkel eingestellt werden. Theoretisch

ist damit das Berechnen der Punkte P redundant und kann in die Berechnung der Punkte PW eingegliedert werden. Um den Quelltext leicht nachvollziehbar zu halten, wurde auf dieses Eingliedern verzichtet und die Punkte P werden, obwohl sie nicht angefahren werden, trotzdem berechnet. Da die Berechnung der Punkte PW auf den jeweiligen Punkten P basiert, wird in den Bereichen zur Erstellung der Punkte im Aktivitätsdiagramm nebeneinander die gleiche Schleife dargestellt.

5.2.3 Programm des Submit-Interpreters

Wie schon in Kapitel 2.1.3 erwähnt wurde, sind in der *sps.sub* Datei auch Programmzeilen enthalten, die automatisch erstellt wurden. Diese variieren je nach installierten Zusatzpaketen und werden hier nicht weiter behandelt.
Am Anfang des SUB-Programms werden, wie bei einem SRC-Programm, diejenigen Variablen deklariert, die im Programmablauf benötigt werden. In diesem Fall wird nur die Variable *RET* als *EKI_STATUS* deklariert, da die anderen verwendeten Variablen global in der *$config.dat* deklariert sind. Innerhalb des *USER INIT-Folds* werden alle in der Anwendung verwendeten *$FLAGs* auf *FALSE* gesetzt, der EKI-Kanal initialisiert und geöffnet. Da vor dem Anwählen der *sps.sub,* der Python-Server auf eine Verbindung warten sollte, ist es unmittelbar nach dem Öffnen möglich, Daten zu transferieren. Das SRC-Programm sollte erst geöffnet werden, nachdem der Submit-Interpreter die Verbindung erfolgreich aufgebaut hat. Dieser zeitliche Versatz im Starten der beiden Programme ist im Aktivitätsdiagramm der Abbildung 5-6 als Höhenunterschied der Interpreter dargestellt.
Ist die Verbindung aufgebaut, wird die Routine des Submit-Interpreters ausgeführt. Bei dieser wird in jedem Durchlauf der Status der *$FLAG[990], [991]* und *[992]* abgefragt. Solange alle *$FLAGs FALSE* sind, wird die Standardroutine ungestört fortgesetzt. Sollte die Funktion *Trajektorie() [990]* oder *[991]* auf *TRUE* setzten, wird der mittlere Strang im Aktivitätsdiagramm ausgeführt. Darin wird zunächst mit *$POS_ACT* und der Funktion *EKI_SetReal()* bzw. *EKI_SetInt()* die aktuelle Position, der Winkel und die Orientierung in die XML-Struktur eingepflegt und danach mit *EKI_Send()* gesendet. Das Senden der Daten am Ende der Scanfahrt ist für den

Scanprozess an sich irrelevant und wird nur zwecks Kontrolle durchgeführt. Um das Senden von falschen Daten zu vermeiden, werden danach die beiden *$FLAGs* wieder auf *FALSE* gesetzt und der Speicher geleert.
Ist der letzte Verfahrpunkt angefahren und damit die Scanfahrt beendet, wird von *Trajektorie()* die *FLAG[992] TRUE* und die Variable *Scan_Fin* wird auf 1 gesetzt. Dadurch wird in der Userroutine der rechte Pfad eingeschlagen, in dem die Orientierung und Fin gesendet werden. Danach wird der EKI-Kanal gelöscht und die Verbindung damit abgebrochen. Im Gegensatz zum Hauptprogramm, endet *sps.sub* nicht nach dem Scannen, sondern durchläuft weiterhin die Standardroutine. Möchte man erneut scannen, muss man nach dem Erstellen des Servers den Submit-Interpreter ab- und wieder anwählen.

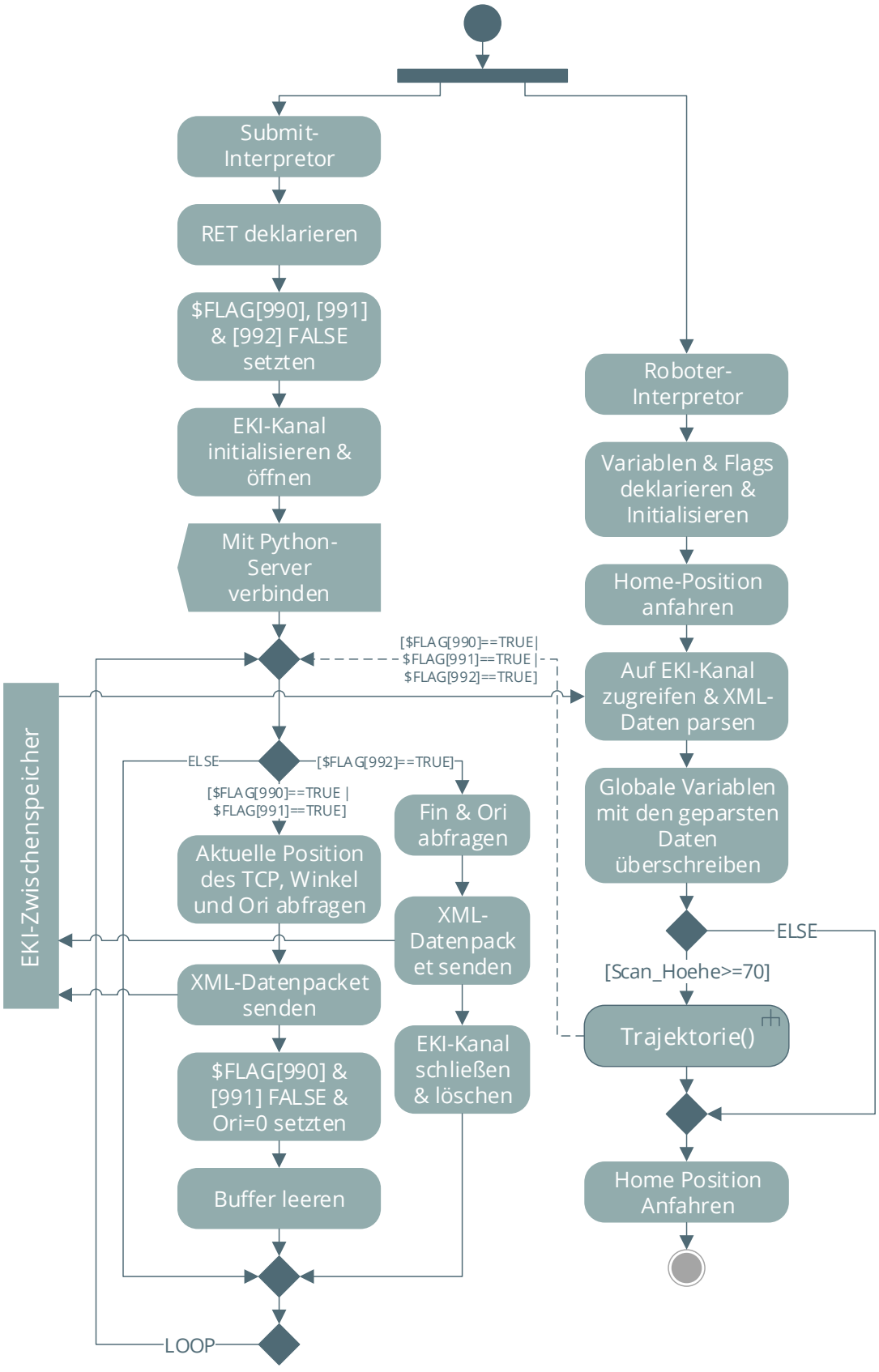


Abbildung 5-6: Aktivitätsdiagramm des Hupt- und Sub-Programm

5.3 Bestandteile des Python-Programms

Um besonders in der Entwicklungsphase die Funktionsfähigkeit zu gewährleisten und die Fehlersuche zu erleichtern, wurden das Python-Programm zunächst in unabhängigen Unterprogramme aufgeteilt, die später zu einem Gesamtprogramm zusammengefügt wurden.

5.3.1 Benutzeroberfläche der Anwendung

Um die Konfiguration des Scanners, der Objekt- und Verfahrdaten zu erleichtern, wurde mithilfe des GUI-Toolkits Tk eine Benutzeroberfläche erstellt. Eine Werteüberprüfung mit eventuellem Anwenderdialog gewährleistet, dass beim Ansprechen des Scanners möglichst keine Error-Meldungen das Programm unterbrechen.

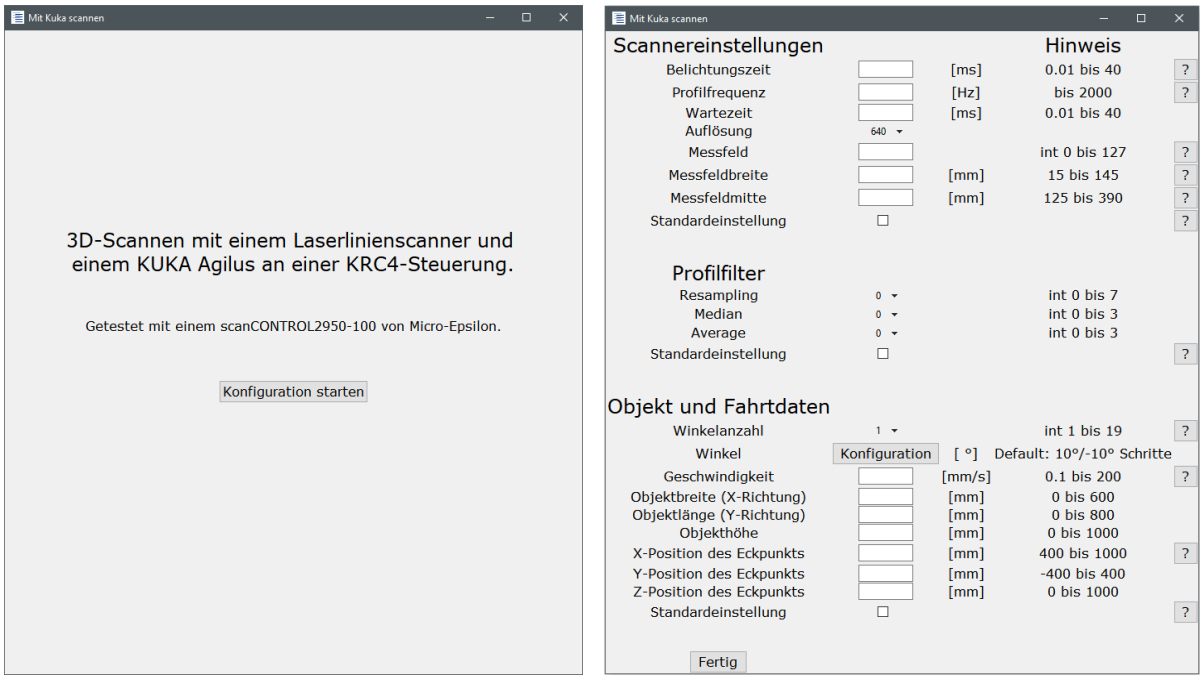


Abbildung 5-7: Startseite (links) und Hauptseite (rechts) der GUI

Hierbei darf die Relevanz dieser Benutzeroberfläche nicht unterschätzt werden. Es ist möglich, die Eingaben manuell im Python-Skript vorzunehmen, wobei vorausgesetzt wird, dass der Anwender die Wertebereiche und Bedingungen der Scannerkonfiguration genau kennt. Bei falscher Konfiguration des Scanners wird eine

Error-Meldung ausgegeben und das Python-Programm bricht ab. Werden Objekt- oder Fahrtdaten falsch eingegeben, könnte entweder die Kommunikation unterbrochen werden oder das KRL-Programm wird während der Scanfahrt abgebrochen.

Nach Ausführen des Programms öffnet sich zunächst die links in Abbildung 5-7 gezeigte Startseite, von der aus man mit einem Knopfdruck auf die Hauptseite gelangt. Die rechts dargestellte Hauptseite unterteilt sich in drei Bereiche. Zur Eingabe der Werte für die Konfiguration des Scanners dient der erste Bereich, für die Einstellung der im Scanner integrierten Filter der zweite und im dritten Bereich werden die Daten für die Scannfahrt und des zu scannenden Objekts eingegeben. Die Bezeichnungen stehen dabei in der ersten und das zugehörige Eingabefeld in der zweiten Spalte. Unter jedem Bereich befindet sich eine Zeile mit der Bezeichnung "Standardeinstellung" und in der jeweiligen Eingabespalte ist ein Ankreuzfeld. Wird ein Kreuz gesetzt, so werden die Standardwerte für den jeweiligen Bereich übernommen und die Eingabefelder müssen nicht komplett ausgefüllt werden. Die Standardwerte beziehen sich auf das Scannen eines relativ hellen Objektes mit geringen Filtereinstellungen. Es wird dabei ein 200mm mal 200mm großen Bereich mit den Winkeln 0°, 10°, -10°, 20° und -20° abgefahren.

Da bei der Auflösung nur vier Werte zur Auswahl stehen, ist hier in der zweiten Spalte ein Dropdownmenü eingebettet. Dasselbe gilt für die kleinen Auswahloptionen der Filtereinstellungen. Da keine SI-Einheiten zur Eingabe verwendet werden, wird in der dritten Spalte die zu verwendende Einheit angegeben. Die vierte und fünfte Spalte soll eine Hilfestellung bei der Eingabe bieten. Hierzu werden in der vierten die Bereiche angegeben, in der sich die Eingabe befinden muss. Der Button mit dem Fragezeichen, der sich in der letzten Spalte befindet, ist eine spezifischere Hilfestellung.

Wird der erste Fragezeichen-Button angewählt, öffnet sich ein Fenster in dem, je nach Beschaffenheit des zu scannenden Objekts empfohlene Bereiche der Belichtungszeit angegeben sind. Diese sind jedoch relativ vage und sollten nur der groben Orientierung dienen. Der Button darunter öffnet eine Tabelle, in der die maximalen

Profilfrequenzen in Abhängigkeit zum Messfeld dargestellt sind. Beim Betätigen des Fragezeichen-Buttons in der Messfeld-Zeile wird die "Abb. 13 Vordefinierte Messfelder" aus der Betriebsanleitung scanCONTROL 29xx von Micro-Epsilons angezeigt, um die Bedeutung der einzugebenden Zahl zu erklären. Der nächste Button öffnet die Abmessungen des Messfelds der 29xx-100er Modelle. Dies dient der Abschätzung der Messfeldbreite und Mitte, was auch nach dem Betätigen des darunterliegenden Knopfes erwähnt wird.

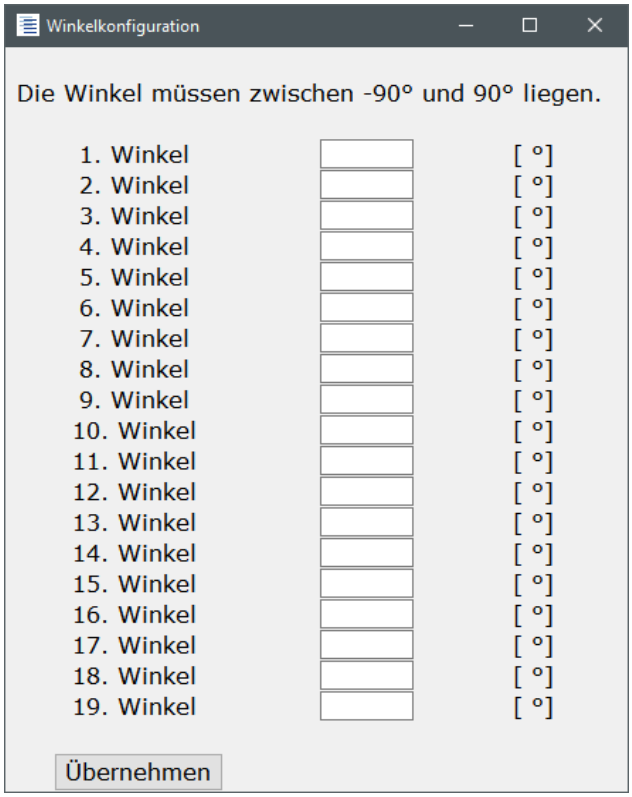


Abbildung 5-8: Winkelkonfigurationsfenster

Der Infobutton der Winkel-Zeile öffnet ein Hinweisfenster, bei dem die Eingabe der Winkel näher erläutert wird. Standardmäßig sind die Winkel in 10°-Schritten abwechselnd positiv und negativ, von 0 bis -90° gestaffelt. Die Winkelanzahl bestimmt die Anzahl der anzufahrenden Winkel. Werden bspw. 4 ausgewählt und die Standardkonfiguration nicht geändert, so werden die Scannfahrten jeweils mit den Winkeln 0°, 10°, -10° und 20° nacheinander durchgeführt. Das in Abbildung 5-8 gezeigte Winkelkonfigurationsfenster, das sich durch Betätigen des „Konfiguration“-Button öffnet, können die Winkel gezielt geändert werden. Zeilen die leer bleiben, behalten den Standardwert und bei mehrmaligem Öffnen des Konfigurationsfensters und beim Übernehmen der Werte werden nur die Winkel der ausgefüllten Zeilen überschrieben. Auch bei veränderten Winkeln ist die Winkelanzahl maßgebend. Nur die Winkel bis zur angegebenen Anzahl werden an den Roboter weitergeben. In der Zeile für die Geschwindigkeit wird beim Anwählen des Buttons ein Hinweisfenster angezeigt, in dem gemeldet wird, dass die Geschwindigkeit mit der Profilfrequenz und Belichtungszeit

abgestimmt werden muss. Das Verhältnis dieser Werte zueinander hat großen Einfluss auf die Punktdichte und Qualität des Scans. Der Hilfe-Button für die Eckpunkteingabe öffnet ein Infofenster, das beschreibt, wie das zu scannende Objekt zu positioniert ist. Der untere Eckpunkt muss die angegebene Position auf 5mm genau besitzen. Der restliche Körper soll dann in positive X- und Y-Richtung des Basiskoordinatensystems ausgerichtet werden. Der Eckpunkt und die Ausrichtung beziehen sich auf einen imaginären Quader, mit den Packmaßen des zu scannenden Objekts, der dieses umschließt.
Die Fragezeichenbuttons in den Standardeinstellungszeilen geben Aufschluss über die als Standard festgelegten Werte.

5.3.2 Funktionsprinzip der GUI

Das eventbasierte Verhalten der GUI ist vereinfacht in der Abbildung 5-9 als Objektdiagramm dargestellt. Hierbei steht nach dem Namen der Elemente, getrennt durch einen Doppelpunkt, die Klasse des jeweiligen Elements. Die Abbildung 5-9 ist ein unkonventionelles Objektdiagramm mit den zusätzlichen Attributen SIGNAL und SLOT. Wird ein Button geklickt, so geht ein Signal zum entsprechenden Slot und eine bestimmte Funktion wird ausgeführt.
Nach dem Öffnen des Programms erscheint zunächst die in Abbildung 5-7 links dargestellte Startseite, auf der sich nur ein Button befindet, der die Funktion *tkraise()* bezogen auf die Hauptseite ausführt. Damit wird der Frame der Hauptseite vor den Frame der Startseite geschoben. Es wird damit kein Fenster geöffnet oder geschlossen. Auf der Hauptseite befinden sich insgesamt 13 Knöpfe, wovon 11 der spezifischen Hilfestellung dienen, einer das Fenster für die Winkelkonfiguration öffnet, und ein Knopf dient dem Abschluss der Eingabe.

Da bei den ersten fünf spezifischen Hilfestellungen dem Benutzer ein größerer Umfang an Informationen dargelegt werden muss, wurde hier auf die Benutzung einfacher Meldefenster der InfoWindow Klasse zu Gunsten einer Bilddarstellung verzichtet. Um Bild-Dateien anzeigen zu können, wird die Python Image Library

verwendet. Diese stammt noch aus Python 2 und ist mit neueren Versionen nicht kompatibel. Um sie trotzdem in Python 3 verwenden zu können, wird die PIL Fork Pillow verwendet. So können bspw. die vordefinierten Messfelder als PNG-Datei in einem eigenen Fenster angezeigt werden. Diese Fenster werden im Objektdiagramm der Klasse ImageWindows zugeordnet und besitzen keine Knöpfe oder Eingabemöglichkeiten.

Das Fenster der Winkelkonfiguration hingegen besitzt sowohl als auch. Beim Betätigen des Übernehmen-Buttons wird die folgende Funktion ausgeführt:

```
def uebernehmen_f():
    halter = 0
    for ii in range(19):
        try:
            wiflo = float(Winkel_eingabe[ii].get())
            if abs(wiflo) < 90:
                Winkel[ii] = wiflo
            else:
                halter += 1
                messagebox.showinfo('Ungültige Eingabe',
                                    'Die Eingabe vom '+str(ii+1)+'. Winkel
                  ' ist ungültige. \n Dieser muss'
                  ' zwischen -90° und 90° liegen.')
        except:
            if str(Winkel_eingabe[ii].get()) == '':
                pass
            else:
                halter += 1
                messagebox.showinfo('Ungültige Eingabe',
                                    'Die Eingabe vom '+str(ii+1)+'. Winkel
                  'ist keine Zahl.')
    print(Winkel)
    if halter == 0:
        windowWK.destroy()
```

Programmcode 7: Python, Funktion uebernehmen_f()

Zur Vereinfachung der Darstellung wird im Objektdiagramm das Abrufen der Winkeleingabe als gesamtes Array gezeigt. Jedoch geschieht dies eigentlich in einer Schleife, bei der jede Eingabe einzeln überprüft wird, ob sie eine Zahl ist und in einem bestimmten Wertebereich liegt. Zunächst wird versucht, die jeweilige Eingabe in das Format Float zu konvertieren. Dies funktioniert nur, wenn die Eingabe eine rationale Zahl ist. Liegt diese dann zusätzlich zwischen -90° und 90°, so wird der Wert der globalen Winkelliste an der entsprechenden Stelle überschrieben. Liegt die Eingabe außerhalb der Grenzen, so wird die Variable *halter* um 1 hochgezählt und ein Informationsfenster wird geöffnet, um den Benutzer über die fehlerhafte Eingabe zu

informieren. Gelingt die Konvertierung in Float nicht, wird überprüft, ob das Eingabefeld leer ist. In diesem Fall wird es ignoriert, wenn dem nicht so ist, wird *halter* hochgezählt und der Benutzer benachrichtigt. Das Fenster wird erst geschlossen, wenn alle Eingaben überprüft wurden und die Variable *halter* trotzdem gleich null ist.
Bei der Anwendung muss darauf geachtet werden, dass die Winkel nacheinander von der aktuellen Eingabe überschrieben werden, nachdem der Übernehmen-Knopf angeklickt wurde. Besitzt bspw. die erste Zeile eine gültige Eingabe, die Zweite nicht und es wird der Übernehmen-Knopf gedrückt, wird trotz fehlerhafter Eingabe der zweiten Zeile, der erste Winkel überschrieben.

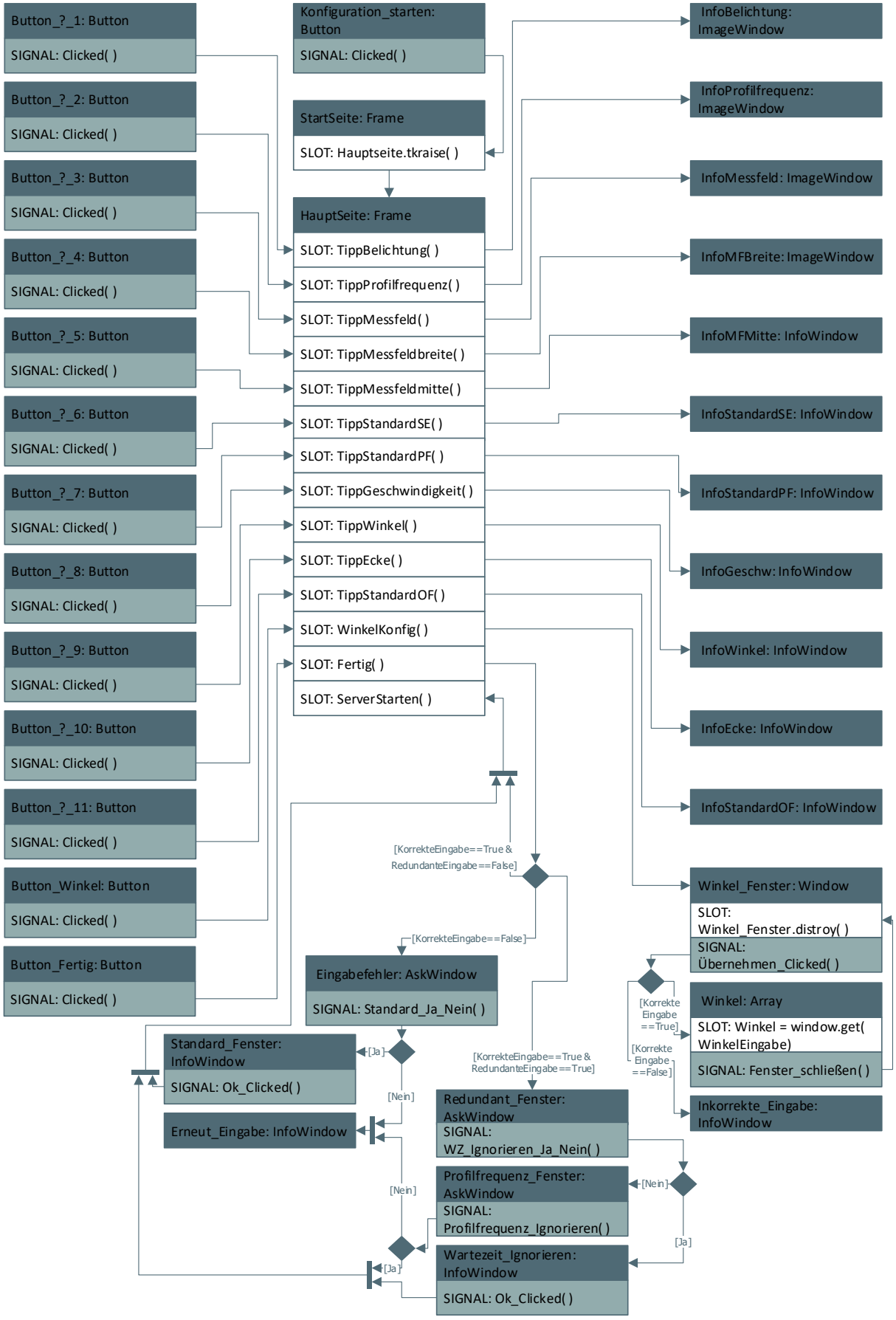


Abbildung 5-9: Objektdiagramm der GUI

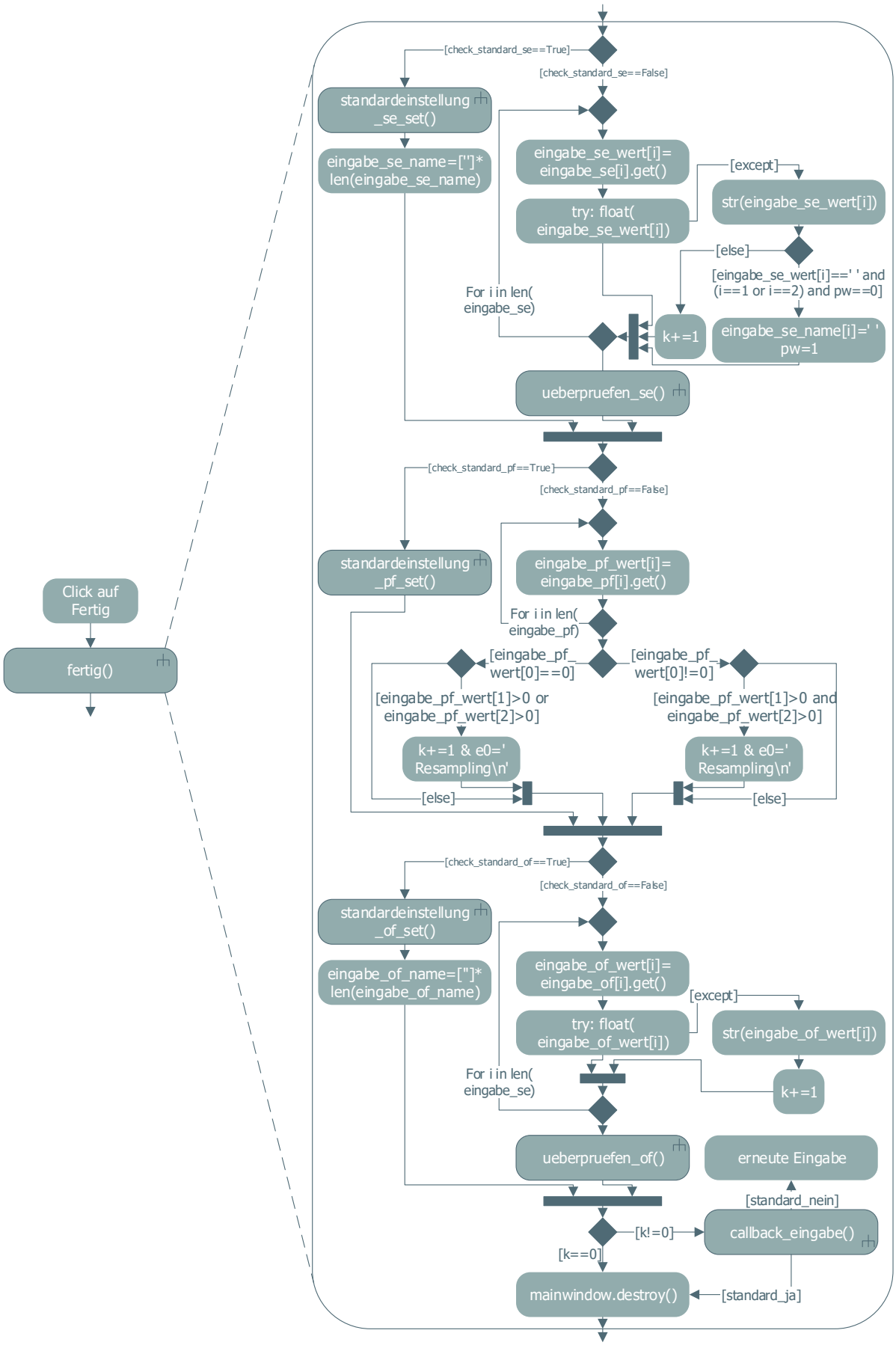


Abbildung 5-10: Aktivitätsdiagramm der Funktion fertig()

Die relativ komplexere Funktion *fertig()* des Fertig-Buttons wird detaillierter in Abbildung 5-10 als Aktivitätsdiagramm dargestellt. Um den Quelltext übersichtlicher zu gestalten, wurde die Funktion entsprechend zur Oberfläche in drei Segmente unterteilt. Am Anfang jedes Segments wird eine IF-Abfrage getätigt, bei der der Status der Check-Boxen für die Standardeinstellungen abgefragt wird. Ist das Ergebnis der Abfrage *True*, so werden die Eingaben des jeweiligen Segmentes ignoriert und es findet auch keine Überprüfung dieser statt. Nach der Einstellung der Standardwerte wird beim ersten und letzten Segment noch das jeweilige Namens-Array nichtig gemacht. Dieses enthält am Anfang die jeweiligen Namen der Eingaben als String und wird bei einer fehlerhaften Eingabe im Dialogfenster angezeigt. Besteht ein Wert in der Liste aus zwei Anführungszeichen ohne Zeichen dazwischen, wird dieser nicht angezeigt. Somit wird nach Hackensetzung kein Name aus dem Bereich im Dialogfenster angezeigt, wenn in einem anderen Bereich eine fehlerhafte Eingabe getätigt wurde.

Die Werteüberprüfung ist in zwei Phasen aufgeteilt. Zunächst wird versucht, alle Werte in Float umzuwandeln. Gelingt dies nicht, wird bis auf eine Ausnahme der Zähler *k* hochgezählt. Die Ausnahme besteht darin, dass es zulässig ist, entweder die Profilfrequenz oder die Wartezeit nicht einzutragen. Da das eine mit dem andern in Zusammenhang mit der Belichtungszeit errechnet wird. Dem Anwender soll dabei freigestellt sein, welchen Wert er dabei angibt. Die zweite Phase ist die eigentliche Überprüfung der Werte, die in der Funktion *überprüfung_se()* stattfindet. In erster Linie wird in dieser untersucht, ob die zulässigen Wetterbereiche eingehalten wurden. In der Funktion der ersten Sektion wird jedoch auch überprüft, ob die Werte zu einander passen. Dies gestaltet sich komplizierter als die Werteüberprüfung der Winkel, da die Belichtungszeit, die Profilfrequenz und die Wartezeit sich gegenseitig bedingen. Die Profilfrequenz ist zudem vom Messfeld abhängig. Wird vom Anwender sowohl die Profilfrequenz als auch die Wartezeit eingegeben, so wird ihm in dieser Überprüfungsfunktion mitgeteilt, dass eine der beiden Eingaben redundant ist und er wird gefragt, welche er ignorieren will.

Bei den Profilfiltern wird die Eingabe nicht in zwei Phasen überprüft. Da nur Integer-Werte eingegeben werden können, wird nur überprüft, ob die ausgewählte Filterkonfiguration möglich ist. Es kann nur entweder Median oder Average eingegeben werden und das nur, wenn Resampling mindestens 1 ist.
Das letzte Segment gestaltet sich ähnlich wie das erste, nur dass in der ersten Phase keine fehlenden Eingaben toleriert werden. Und die Überprüfungsfunktion *überprüfung_of()* kontrolliert nur die jeweiligen Zahlenbereiche der Eingaben.
Bei jedem auftretenden Fehler wird der Zähler *k* um eins hoch gezählt und nur wenn dieser gleich Null ist, wird die gesamte Konfiguration beendet und die Eingabewerte an den Serverabschnitt weitergegeben. Ist der Zähler ungleich Null, wird eine Callbackfunktion ausgeführt, in der die fehlerhaften Eingaben aufgelistet werden und der Benutzer gefragt wird, ob er die Standardeinstellung für die jeweiligen Bereiche verwenden will oder die Werte erneut eingeben möchte.

5.3.3 Python-Server

Das Serverprogramm baut sowohl die Verbindungen zu dem Laserlinienscanner, wie auch zu der KRC4-Steuerung auf. Es ist damit das Kernelement des Gesamtprogramms. Wie in Kapitel 4.4 beschrieben, sind beide Hardwarekomponenten über Ethernet miteinander verbunden und benutzen Sockets zur Kommunikation. Allerdings ist die Socket-Programmierung des Scanners in der pyllt-Bibliothek der zur Verfügung gestellten SDK enthalten. Somit genügt es,

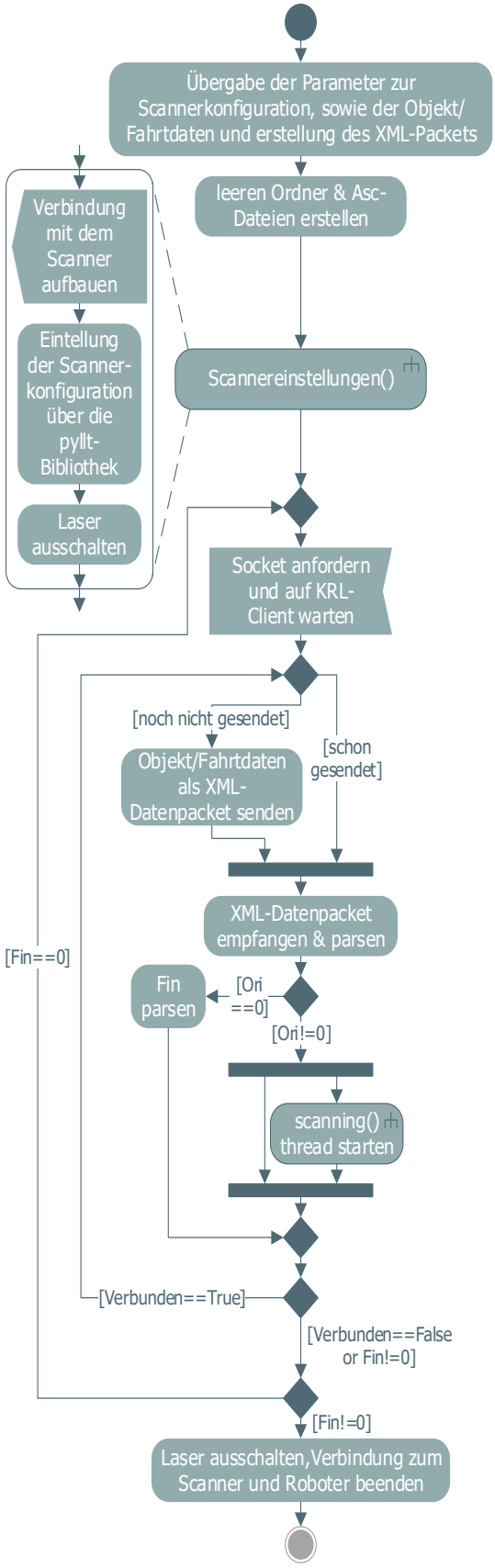


Abbildung 5-11: Aktivitätsdiagramm des Servers

hLLT=llt.create_llt_device(llt.TInterfaceType.INTF_TYPE _ETHERNET) auszuführen und mit *llt.connect(hLLT)* eine Verbindung mit dem Scanner aufzubauen. Das Verbinden und anschließende Konfigurieren geschieht in der Funktion *Scannereinstellungen()*, die in Abbildung 5-11 als Lupendarstellung im Aktivitätsdiagramm gezeigt wird. Am Ende der Funktion wird der Laser ausgeschaltet, um bei unvorhersehbaren Bewegungen der ersten PTP-Fahrt niemanden zu blenden.

Als nächstes wird eine Server-Client-Verbindung mit der Robotersteuerung aufgebaut. Der dazu benötigte Server ist ähnlich aufgebaut wie der in Kapitel 4.3.3 beschriebene. Er besitzt jedoch noch zusätzliche Funktionen. Nach dem erfolgreichen Verbinden wird zunächst vom Auslesen des Buffers einmalig das folgende XML-Paket mit *conn.send(bytearray(xml,'utf-8'))* an den Client gesendet:

```
xml =
'<Sensor>' \
  '<Read>' \
    '<Hoehe>' + str(Hoehe) + '</Hoehe>' \
    '<Breite>' + str(Breite) + '</Breite>' \
    '<Laenge>' + str(Laenge) + '</Laenge>' \
    '<Scannerbreite>' + str(Scannerbreite) + '</Scannerbreite>' \
    '<Scannerhoehe>' + str(Scannerhoehe) + '</Scannerhoehe>' \
    '<Winkel>' + str(Winkel[0]) + '</Winkel>' \
    '<Winkel>' + str(Winkel[1]) + '</Winkel>' \
...
    '<Winkel>' + str(Winkel[18]) + '</Winkel>' \
    '<Geschwindigkeit>'+str(Geschwindigkeit)+'</Geschwindigkeit>' \
    '<xyzabc X="'+str(PEx[0])+'"Y="'+str(PEy[0])+'"Z="0"A="0"B="0"C="0"/>'\
  '</Read>' \
'</Sensor>'
```

Programmcode 8: Python, zu sendendes XML-Paket

Die drei Punkte stehen hier stellvertretend für die Winkel [2] bis [17]. Somit wurden der Steuerung alle vom KRL-Programm benötigten Daten übermittelt. Um dieses Paket nur einmal pro Verbindung zu senden, wird am Anfang der while-True-Schleife die Variable *empfangen* abgefragt. Diese ist vor der Schleife gleich *False*. Ist dies auch während der Abfrage der Fall, wird das XML-Paket gesendet. Nach dem Empfangen des ersten Pakets vom Client, wird *empfangen* gleich *True* gesetzt. Um das Paket jedoch auslesen zu können, muss die XML-Struktur geparst werden. Hierzu wird das Modul *xml.etree.ElementTree* verwendet. Es bietet grundlegende Funktionen zum parsen von

XML-Strukturen und unterstütz dabei rudimentären XPath-Syntax. Das Empfangen und parsen der Daten stellt sich im Quelltext wie folgt dar:

```
data = conn.recv(BUFFER_SIZE)
if not data: break
root = ET.fromstring(data)
try:
    x_r = float(root.find('Data/ActPos/X').text)
    y_r = float(root.find('Data/ActPos/Y').text)
    z_r = float(root.find('Data/ActPos/Z').text)
    a_r = float(root.find('Data/ActPos/A').text)
    b_r = float(root.find('Data/ActPos/B').text)
    c_r = float(root.find('Data/ActPos/C').text)
    W = float(root.find('Winkel').text)
except:
    print('Keine Positionsdaten empfangen.')
try:
    Ori = int(root.find('Orientierung').text)
    empfangen = True
except:
    pass
```

Programmcode 9: Python, Daten empfangen und parsen

Mit der Funktion *fromstring(data)* wird aus dem empfangenen String eine Baumstruktur erstell, durch die man mit XPath navigieren kann. Das geschieht in den darauffolgenden Zeilen mit der Funktion *find()*. Die Schreibweise ist dabei ähnlich wie bei einem Dateipfad, nur dass keine Ordner aufgelistet sind, sondern Knotenpunkte der XML-Struktur. Am Ende des Pfades befindet sich in diesem Fall das jeweils gesuchte Element bzw. der Inhalt des Tags. Für das Extrahieren des Elementinhalts ist das *.text* am Ende zuständig.

Nach dem Empfangen und Parsen der Positionsdaten und der Winkel wird die Orientierung geparst und, wie oben beschrieben, *empfangen* gleich *True* gesetzt. Ist die Orientierung ungleich null, so wird vom Roboter eine LIN-Fahrt ausgeführt und der Thread wird mit der Funktion *scanning()* ausgeführt. Sollte die Orientierung gleich null sein, wird *Fin* abgefragt und somit überprüft, ob die letzte Scannfahrt durchgeführt wurde. Ist dies der Fall, wird wie am Ende der Abbildung 5-11 gezeigt, die Endlosschleife abgebrochen, der Laser ausgeschalten und die Verbindungen sowie das Programm beendet.

Sollte die Endlosschleife durch einen Verbindungsabbruch beendet werden und die Variable Fin ist noch gleich null, so wird *empfangen* wieder *False* gesetzt und der Server wartet erneut auf eine Verbindung.

5.3.4 Scanningthread und Zusammenführung der Daten

Sobald die Variable *Ori* geparst wurde und sich als ungleich null herausstellt, wird ein Thread gestartet, der die Funktion *scanning()* ausführt und die geparsten Positionsdaten sowie den Winkel übergibt. Dieser Vorgang wird auch in Abbildung 5-11: Aktivitätsdiagramm des Servers dargestellt. Die Funktion ist noch einmal detailliert in Abbildung 5-13, ebenfalls als Aktivitätsdiagramm gezeigt.

In der Variablen *start_time* wird die momentane Uhrzeit mit der Funktion *time()* abgespeichert, sobald die Funktion *scanning()* ausgeführt wird. Diese Zeit wird später benutzt, um eine Differenz zu bilden. Danach wird mit den Befehlen aus der *pyllt* der Laser angestellt, der Datentransfer aktiviert. Nachdem der Transfer aktiviert wurde, muss der Scanner 0.1 Sekunde vorgewärmt werden, um am Anfang keine fehlerhaften Messergebnisse zu bekommen. Diese Aufwärmzeit ist im integrierten Vorlauf der Trajektorie berücksichtigt. Ist der Scanner vorgewärmt, wird eine Schleife durchlaufen in der nacheinander die Werte aus dem Scannerbuffer ausgelesen, mit den Positionsdaten verrechnet und dem *schreiben()*-Thread übergeben werden. Diese Schleife wird solange durchlaufen, bis die Variable *Ori* überschrieben wird und dabei gleich null ist. Da die Funktion *scanning()* nur aufgerufen werden kann, wenn *Ori* ungleich null ist und diese Variable in der Schleife nicht verändert wird, wäre es im Normalfall eine Endlosschleife. *Ori* wurde jedoch im Hauptprogramm initialisiert und in Python sind Variablen, die in einer unteren Ebene definiert wurden, für die darauffolgenden Unterprogramme und Funktionen einer höheren Ebene automatisch als *global* definiert. Zudem werden die Daten des EKI-Clients parallel zum Scanningthread empfangen und geparst. Wird vom Client dann die Orientierung null gesendet, wird die Variable *Ori* mit null überschrieben und die Hauptschleife der Funktion *scanning()* wird beendet.

In der Hauptschleife werden die Profile, wie in Kapitel 4.3.1 beschrieben, aus dem Scannerbuffer ausgelesen und in eine Liste mit Float-Werten konvertiert. Dies ist nötig, da die erhaltenen Daten als Double-Array nicht mit den Positionsdaten im Float-Format verrechnet werden können.

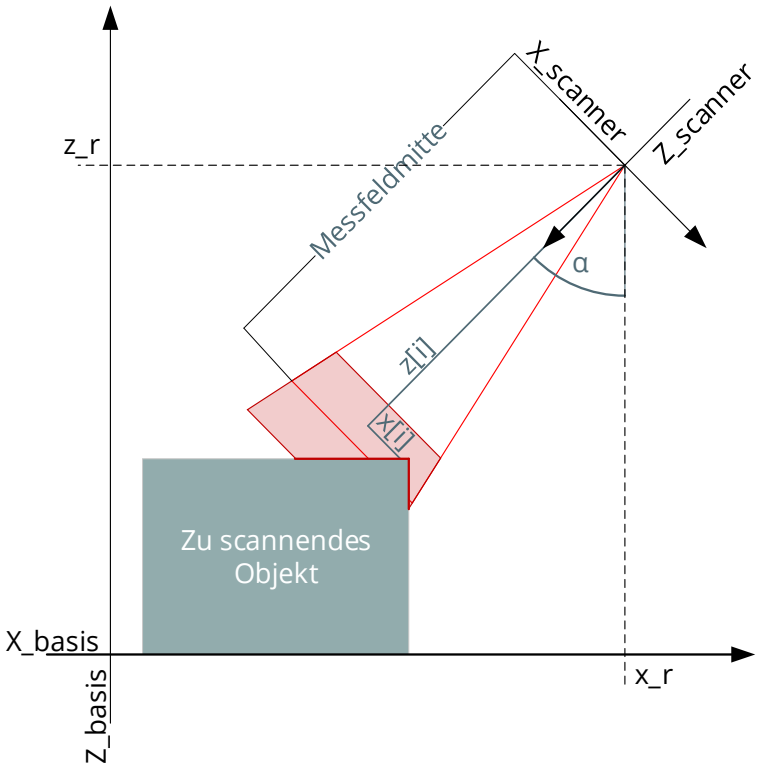


Abbildung 5-12: KKS des Scanners im KKS der Basis

Wie die X- und Y-Werte verrechnet werden, hängt von der Orientierung ab. Wird die Scanfahrt parallel zu Y-Achse des Basiskoordinatensystems ausgeführt, so ist *Ori* gleich 1 oder -1, je nach Richtung der Fahrt. Damit ist der absolute Wert von *Ori* gleich Eins und dadurch werden nach der If-Abfrage, die in Abbildung 5-13 als Verzweigung dargestellt ist, die X-Werte der Scandatenliste mit dem X-Wert der Positionsdaten verrechnet und in die Liste nx geschrieben. Je nach gefahrenem Winkel hat auch der Z-Wert der Scandaten einen Einfluss auf die Werte der nx-Liste.

In Abbildung 5-12 ist eine Scanfahrt mit der Orientierung 1, dem Winkel α um die Y-Achse zweidimensional dargestellt. An dieser Skizze wird deutlich, dass die Verrechnung der Scandaten mit den gesendeten Positionsdaten mittels einfacher trigonometrischer Formeln durchgeführt werden kann. Die jeweiligen nx-Werte werden somit wie folgt berechnet:

$$nx[i] = x_r - z[i] \cdot \sin(\alpha) + x[i] \cos(\alpha)$$

Analog dazu wird auch die nz-Liste mit den gemessenen X- und Z-Werten berechnen. Die Formel dafür lautet:

$$nz[i] = z_r - z[i] \cdot \sin(\alpha) - x[i] \cos(\alpha)$$

Die Berechnung der einzelnen Listenelemente findet dabei in einer For-Schleife statt, die so lange wie das Double-Array ist.
Die Variable ny wird aus dem gesendeten Y-Wert, der Geschwindigkeit der LIN-Fahrt, der Orientierung und der Zeitdifferenz zwischen der aktuellen Zeit und der *start_time* berechnet.
Ist der Absolutwert der Orientierung jedoch Zwei, wird die Scanfahrt parallel zur X-Achse ausgeführt. Im Programm ist somit die Elif-Abfrage (Else-If in Python) $Ori^2==4$ erfüllt und die Werte der nx-Liste werden nicht mit dem empfangenen X-Wert berechnet, sondern mit dem Y-Wert *y_r*. Umgekehrt wird ny dafür mit dem geparsten X-Wert *x_r* berechnet.
Dies scheint zunächst unsinnig, jedoch wird die gleiche If/Elif-Abfrage in der Funktion *schreiben()* durchgeführt, die daraufhin in einem weiteren Thread gestartet wird. Ist $Ori^2==1$, dann wird die ASC-Datei in der Reihenfolge nx, ny, nz beschrieben und bei $Ori^2==4$ wird die Reihenfolge ny, nx, nz verwendet.
Wie zuvor beschrieben, wiederholt sich dieser Vorgang, bis eine Orientierung von 0 gesendet und empfangen wird. Danach wird der Transfer beendet und der Laser des Scanners wird ausgeschaltet.

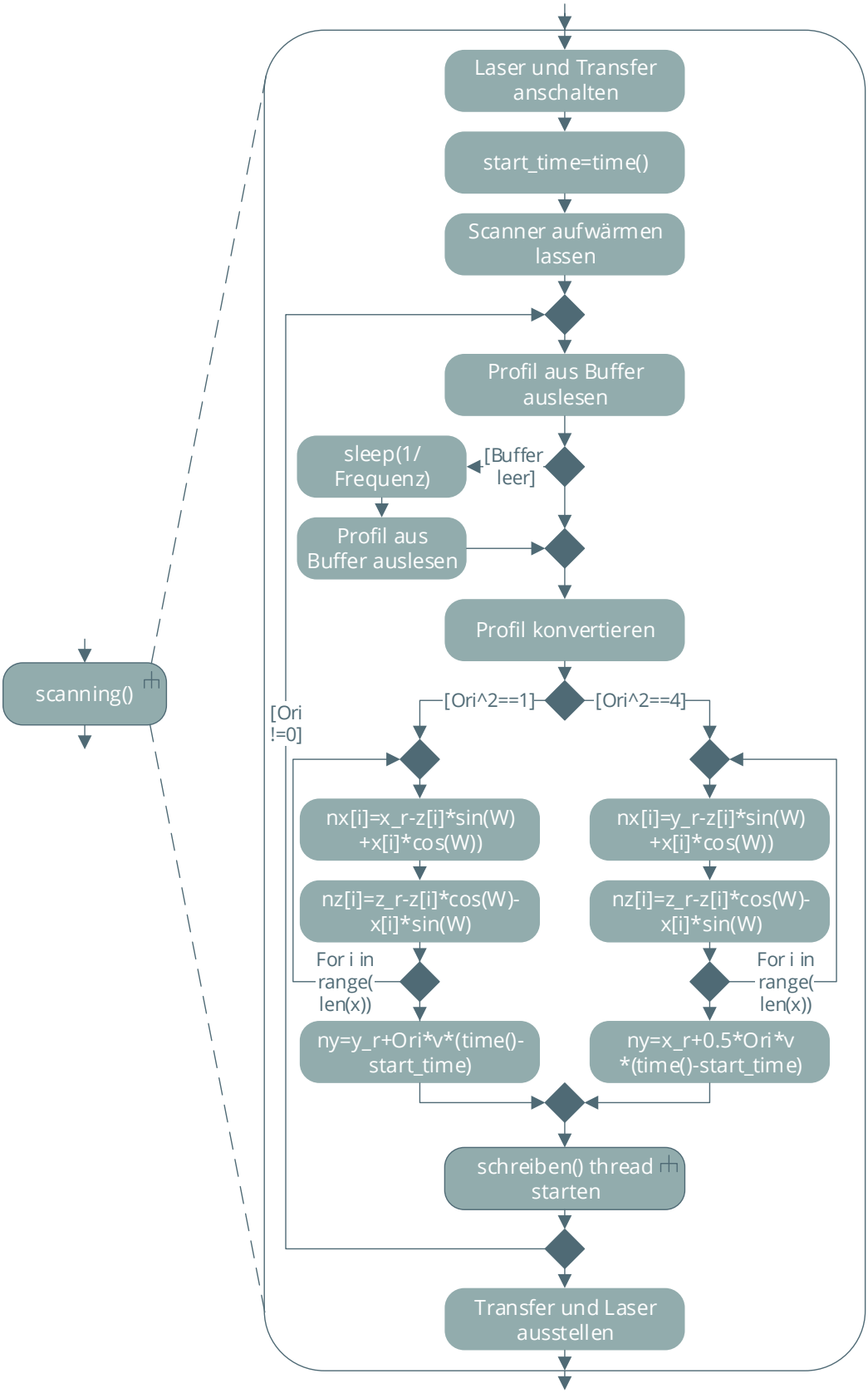


Abbildung 5-13: Aktivitätsdiagramm der Funktion scanning()

5.3.5 Erstellung der Punktwolke

Ein großes Problem beim Erstellen einer Punktwolke ist das zu wählende Format, da noch kein Standardformat existiert. Am verbreitetsten sind ASCII-Punktwolken, jedoch gibt es hier etliche Varianten. Jede Art von Textformat, das ASCII-Zeichen unterstützt, kann zur Erstellung einer ASCII-Punktwolke benutzt werden. Selbst bei gleichem Dateiformat können sich die Punktwolken durch die Trennzeichen, sowie die Art und Reihenfolge der Positionsdaten unterscheiden. Auch die Zusatzinformation der jeweiligen Punkte ist nicht festgelegt. Meistens besitzen die Punkte entweder keine Zusatzinformationen, eine Intensität oder RGB-Werte.
Ein Grund dafür, dass noch kein Standardformat festgelegt wurde ist, dass Punktwolken für viele unterschiedlichen Bereiche genutzt werden, die dementsprechend unterschiedliche Anforderungen an das Format stellen.
In der Archäologie ist bspw. oft die Intensität der gemessenen Punkte eine wichtige Zusatzinformation. Bei der modernen Photogrammetrie spielen meist die RGB-Werte eine entscheidende Rolle, um realistische Texturen erstellen zu können. Wiederum andere Bereiche verzichten zu Gunsten des Speicherbedarfs ganz auf die Zusatzinformationen. Es kann sich auch regional unterscheiden, ob in Dateien Dezimalpunkte oder Dezimalkommas verwendet werden.
Das Verwenden des ASC-Dateiformats ist somit willkürlich gewählt worden. Dieses kann sowohl von MeshLab als auch von PolyWorks geöffnet und bearbeitet werden. Als Struktur der Datei wurde das reihenweise Anordnen der Punkte mit X, Y und Z Spalten verwendet. In einer weiteren Spalte nach der Z-Spalte ist als Zusatzinformation die Intensität der jeweiligen Punkte vermerkt. Getrennt werden die Werte durch Leerzeichen und es werden Dezimalpunkte verwendet.

Je Scanfahrt wird eine ASC-Datei angelegt. Um dies übersichtlich zu gestalten, wird vor dem Erstellen der leeren Dateien ein Ordner angelegt. Der Name des Ordners bildet sich aus dem Erstellungsdatum mit der Uhrzeit und den wichtigsten Werten zur Scannerkonfiguration. Sollte dieser Ordner schon vorhanden sein, wird er überschrieben. Die einzelnen Dateien werden dann in folgender Schleife initialisiert:

```
block = 0  # Block an Dateien mit dem selben Winkel
for i in Fahrten:
    if i >= (j+k)*(block+1):
        block+=1
    dateihandler = open(pfad+'/Scann '+str(jetzt)+' Teil '+str(i+1)
         +' Winkel '+str(round(Winkel[block]))+'.asc', mode='w')
    dateihandler.flush()
    dateihandler.close()
```

Programmcode 10: Python, Erstellung der leeren ASC-Dateien

Das Erstellen von Text-Dateien gestaltet sich in Python sehr einfach durch die Funktion *open()*. Im ersten Übergabeelement der Funktion wird der Dateipfad mit anschließendem Namen und der Dateiendung eingetragen. Wurde nur der Dateiname mit Endung eingetragen, wird diese im selben Ordner wie das Programm abgespeichert. Im zweiten Übergabeelement wird der Bearbeitungsmodus festgelegt. Im Modus ‘w‘ wird die Datei überschrieben, und wenn sie nicht vorhanden ist, neu erstellt. In der oben gezeigten Schleife werden so nacheinander alle benötigten ASC-Dateien erstellt. Der Name der einzelnen Dateien besteht aus dem Zeitpunkt der Erstellung, einer fortlaufenden Nummer und dem eingestellten Winkel. Dabei werden die Dateien in der gleichen Reihenfolge erstellt, wie sie später auch beim Scannen beschrieben werden. Nachdem alle Dateien mit demselben Winkel erstellt wurden, wird der Zähler *block* hochgezählt und die Dateien mit dem nächsten Winkel werden erstellt. Die Namensstruktur soll damit beim späteren Bearbeiten und Vergleichen eine gute Übersicht bieten.

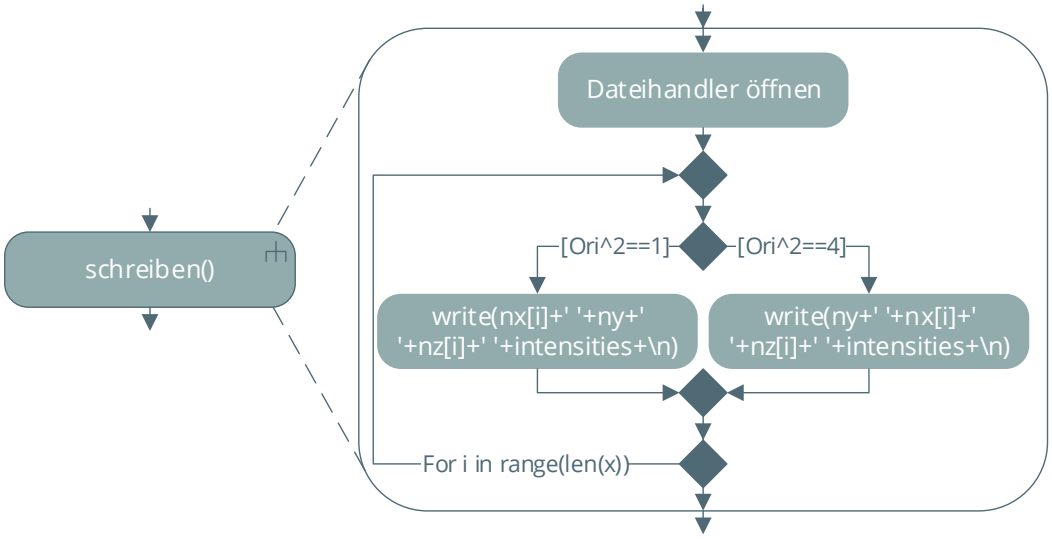


Abbildung 5-14: Aktivitätsdiagramm der Funktion schreiben()

Das eigentliche Beschreiben der ASC-Dateien geschieht während des Scannens in einem eigenen Thread mit der Funktion *schreiben()*. Diese ist in Abbildung 5-14: Aktivitätsdiagramm der Funktion schreiben() als Aktivitätsdiagramm im Detail abgebildet. Wie in Kapitel 5.3.4 beschrieben, wird auch hier die gleiche If/Elif-Abfrage von *Ori* durchgeführt und bei ungleich Eins wird die Position ny mit nx vertauscht. Da in diesen ASC-Dateien die XYZ-Konvention verwendet wird, werden die ny Werte somit in die Spalte für X eingetragen und vice versa. Pro Aufruf der Funktion wird ein Profil in die Datei geschrieben. Dafür wird der Modus 'a' von *open()* verwendet. Dieser funktioniert ähnlich wie 'w', jedoch wird die ASC-Datei nicht überschrieben, sondern die neuen Daten werden an das Ende der alten Daten eingetragen. Nachdem die Datei dann geöffnet wurde, wird das eigentliche Beschreiben von der Funktion *write()* durchgeführt.

6 Auswertung des Scans

Um die Genauigkeit des Scansystems abschätzen zu können, wurde eine Referenzplatte gescannt. Die Kalibrierplatte *GOM/CP40/MV200mm* ist ursprünglich zur Kalibrierung eines Streifenlichtmesssystems von GOM gedacht. Sie besteht aus einer Glasplatte, der ein Punktmuster fotolithografisch aufgedruckt wurde und einem gummierten Randbereich. Die Längen der in Abbildung 6-1 markierten Prüfstrecken beträgt jeweils 250.226 mm [vgl. Dennis Schridde, Werkskalibrierschein].

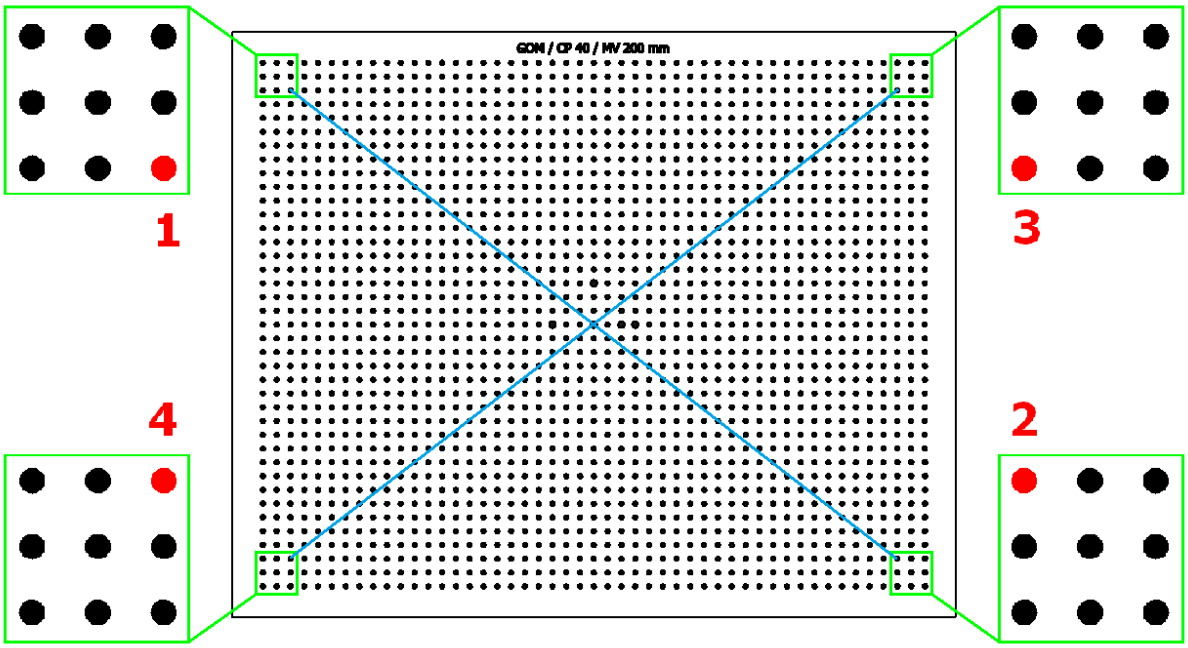


Abbildung 6-1: Kalibrierplatte mit eingezeichneter Prüfstrecke [Anhang 5.2 GOM Werkskalibrierschein CP40/200/100819 S.3 Abb. 2]

Da nur der Werkskalibrierschein zur Verfügung stand, wurden die vertikalen und horizontalen Distanzen aus der Anzahl von Punkten (vertikal 34 und horizontal 44) und dem Satz des Pythagoras wie folgt ermittelt:

$$x = \sqrt{\frac{250.266^2 mm^2}{34^2 + 44^2}}$$

$$x = 4.5001\ mm$$

$$Distanz1zu3 = 44\ x$$

$$Distanz1zu3 = 198.032\ mm$$
$$Distanz1zu4 = 34\ x$$
$$Distanz1zu4 = 153.025\ mm$$

Mit diesen Werten als Basis wurden zunächst die Scanfahrten mit 0° in Y-Richtung geprüft. Die vertikalen Werte sind mit 151.828mm und 152.621mm relativ nah an den geforderten 153.025mm, jedoch weichen die horizontalen Werte mit 201.468mm und 200.481mm weit von den errechneten 198.032mm ab. Da die Horizontale der Plate parallel zur Y-Achse positioniert wurde, lässt sich daraus schließen, dass die Abweichung vom Errechnen der Y-Position aus der Geschwindigkeit resultiert.

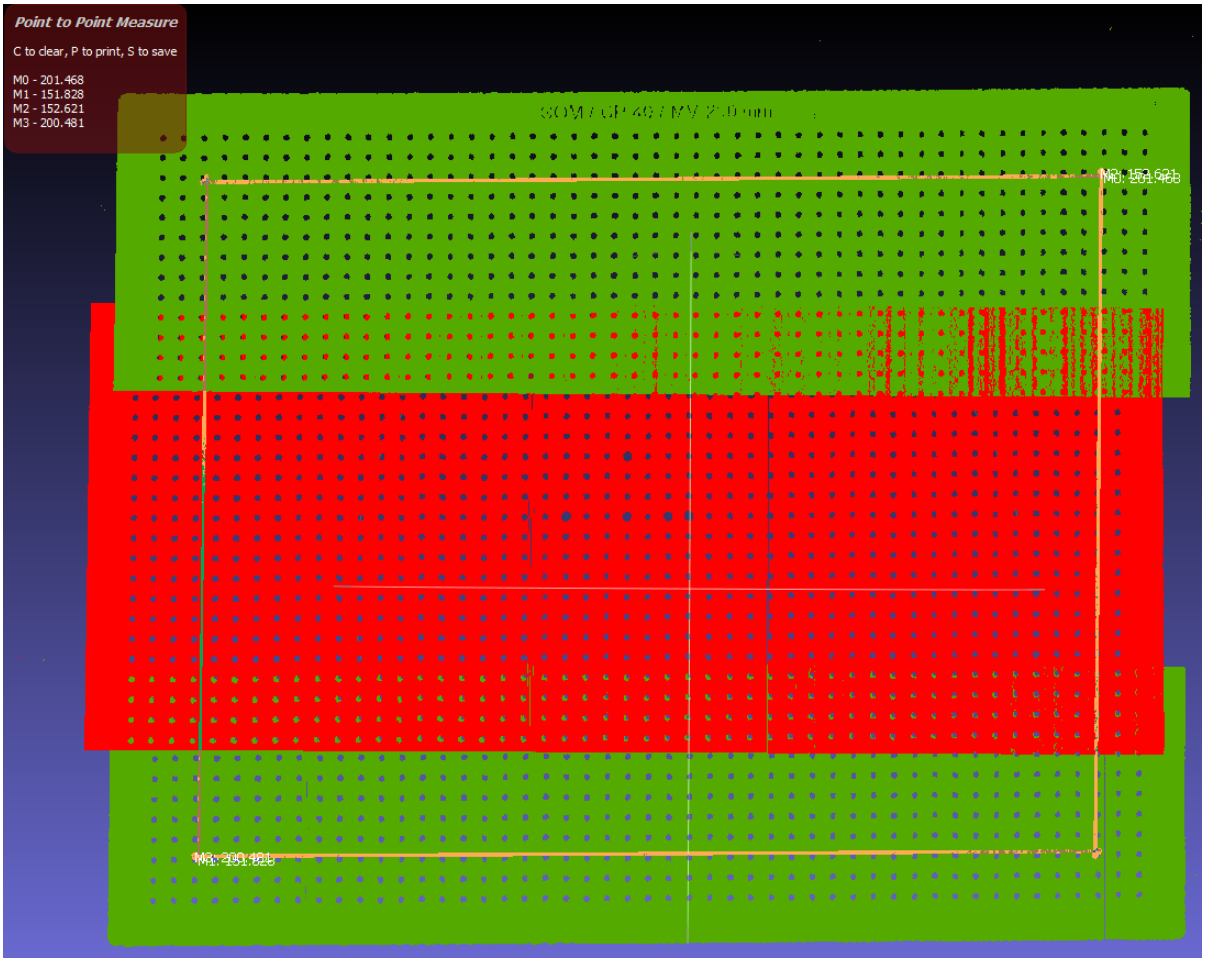


Abbildung 6-2: Mäandrische Scanfahrten bei 0° in positive (grün) und negative (rot) Y-Richtung

Beim Betrachten der Scans in Abbildung 6-2: Mäandrische Scanfahrten bei 0° in positive (grün) und negative (rot) Y-Richtung

fällt jedoch auf, dass die rot markierte Scanfahrt in negative Y-Richtung einen erheblichen Versatz zu den Scanfahrten in positiver Richtung aufweist. Bei der in Abbildung 6-3: Erste Scanfahrt in X-Richtung (gelb) im Vergleich zur Y-Richtung, Draufsicht (links), Seitenansicht (rechts)
gelb markierten Scanfahrt in X-Richtung ist ebenfalls ein deutlicher Versatz in alle Richtungen zu erkennen.

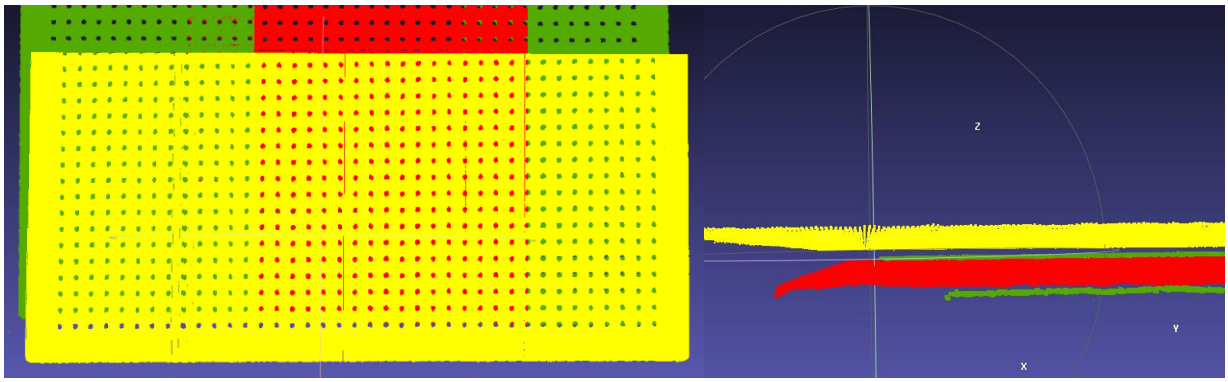


Abbildung 6-3: Erste Scanfahrt in X-Richtung (gelb) im Vergleich zur Y-Richtung, Draufsicht (links), Seitenansicht (rechts)

Um den Grund des Versatzes zu finden, wurden bei der Kommunikation mit der Steuerung die empfangenen Positionen ausgegeben und untersucht. Da die empfangenen Positionen im erwarteten Wertebereich lagen, entstand die Vermutung, dass die real angefahrene Position des Manipulators nicht mit der versendeten übereinstimmt. Um die Vermutung zu überprüfen, wurde die Verfahrlinie in der Trajektorie angepasst, sodass nur noch in positive Richtung gefahren wird. Dadurch ist die Verfahrlinie nicht mehr mäandrisch, sondern zickzack-förmig. Die resultierenden Scans sind in Abbildung 6-4: Zickzack-Scanfahrten Y-Richtung dargestellt. Der Versatz in Y-Richtung konnte mit dieser Methode entfernt werden, was die vorangegangene Vermutung bestätigt. Zusätzlich

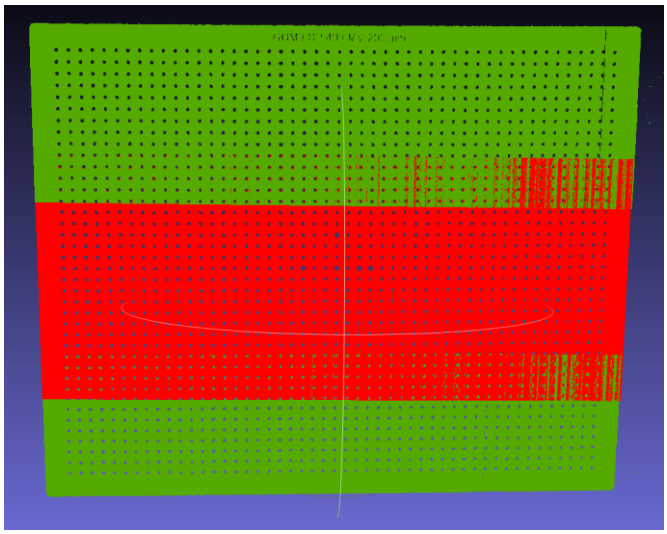


Abbildung 6-4: Zickzack-Scanfahrten Y-Richtung

ist jedoch noch ein leichter Stufenversatz zu erkennen, was darauf schließen lässt, dass die Laserlinie nicht exakt parallel zur X-Achse ist. Da der rechte Winkel der Halterung genau geprüft wurde, liegt diese Abweichung wahrscheinlich an der ungenauen Montage an den Manipulatorflansch.

Im momentanen Zustand des Scansystems ist eine exakte Ermittlung der Genauigkeit nicht möglich. Um exaktere Angaben über die Genauigkeit treffen zu können, müsste zunächst der Manipulator kalibriert, der Scanner genauer ausgerichtet werden und der TCP neu konfiguriert werden. Dann müsste die Genauigkeit der Scangeschwindigkeit untersucht und gegebenenfalls Anpassungen seitens der Software gemacht werden. Dies würde jedoch den Rahmen dieser Arbeit sprängen, weshalb dies in späteren Arbeiten durchgeführt werden muss.

7 Verarbeitung der Punktwolken

Die meisten Programme zur Bearbeitung von Punktwolken sind an das Scansystem gekoppelt. Eine systemunabhängige Open Source Software, die Punktwolken bearbeiten und umwandeln kann ist MeshLab. Mit ihr ist es unter anderem möglich, Punktwolken auszurichten, zu filtern, zu vermessen und in Oberflächen umzuwandeln. Auch sind darin einige Funktion zur Erstellung von photogrammetrischen Modellen enthalten. Hierzu werden jedoch noch Tiefeninformationen zu den Bildern benötigt, die mit einer separaten 3D-Bildrekonstruktion (z.B. mit Arc3D) erstellt werden müssen. Die Anwendung der Vielzahl an Funktionen ist jedoch umständlich, da das Programm vergleichsweise unübersichtlich aufgebaut ist. Ein weiterer Nachteil bei MeshLab ist, dass es in der aktuellen Version 2016 noch keine Möglichkeit gibt, eine Aktion rückgängig zu machen. Wird ein Fehler bei der Bearbeitung gemacht, ist er nur schwer zu korrigieren. Eine kostenpflichtige Alternative zu MeshLab ist PolyWorks. Diese Software bietet ebenfalls viele Funktionen zur Bearbeitung von Punktwolken. Der große Vorteil von Polyworks gegenüber MeshLab ist die annähernd intuitive Bedienung und das Undo-Prinzip. Bei Letzterem kann je nach Toolauswahl die Aktion des Werkzeuges oder die veränderte Perspektive rückgängig gemacht werden. Mit der Anwendung *Modeler* bietet Polyworks auch zahlreiche Funktionen um Polygonnetze zu bearbeiten.
Um die schon sehr hohen Anforderungen an den späteren Anwender nicht unnötig zu erhöhen, wurde die Umwandlung der Punktwolke in ein Polygonmodell in dieser Arbeit mit PolyWorks umgesetzt.

Im Folgenden wird anhand eines gescannten 2x2 Lego Duplo Bausteins eine mögliche Art aufgezeigt, wie man mit PolyWorks ein Polygonmodell aus den gescannten Punktwolken generiert. Es wurde ein Duplo Stein als Testobjekt genommen, da er trotz seiner einfachen Geometrie für die verwendete Scanmethode eine Herausforderung darstellt. Durch die glänzende Kunststoffoberfläche werden einige

Laserpunkte von der Sensorsteuerung fehlinterpretiert und diese Fehler müssen in der Weiterverarbeitung ausgeglichen werden. Zudem müssen mehrere Winkelfahrten angewendet werden, da die harten Kanten des Steins und die Aushöhlung der Noppen Abschattungen verursachen.

Da die Punktwolken der unterschiedlichen Scanfahrten räumliche versetzt sind, müssen die Scans zueinander ausgerichtet werden. Entweder können die Punktwolken selbst zueinander ausgerichtet werden, oder es werden aus den einzelnen Punktwolken Polygonmodelle erstellt und dann ausgerichtet. Das Ausrichten wird dabei im Inspector von PolyWorks durchgeführt. Im Folgenden wird das Vorgehen der Ausrichtung beschrieben, bei dem zuvor ein Polygonmodell erzeugt worden ist. Die Punktwolken können mit der gleichen Methode ausgerichtet werden, jedoch ist die Darstellung der Ausrichtung unübersichtlicher.

Abbildung 7-1: Einstellung der Vernetzung (links), Punktwolke (mittig), vernetzte Oberfläche (rechts)

Um eine Punktwolke zu vernetzten, müssen dem Programm einige Parameter übergeben werden. Diese variieren, je nach den Größenverhältnissen des Scans und sind endscheidend für die Qualität des Polygonmodells. Auf der linken Seite in Abbildung 7-1: Einstellung der Vernetzung (links), Punktwolke (mittig), vernetzte Oberfläche (rechts)

sind die verwendeten Parameter für die Vernetzung eines 20°-Scans des Bausteins (in mm) gezeigt. Bei der Suche, nach den richtigen Werten für die Parameter bietet es sich

an, die Vorschaufunktion zu verwenden. In der Mitte und der rechten Seite der Abbildung 7-1: Einstellung der Vernetzung (links), Punktwolke (mittig), vernetzte Oberfläche (rechts)
ist gut zu erkennen, wie durch die Abschattung Lücken im Netz entstehen. Wie schon zuvor erwähnt, werden die Abschattungen durch Scans mit anderer Orientierung kompensiert. Um die unterschiedlichen Polygonmodelle zueinander auszurichten, kann die *Best-Fit*-Funktion des Inspectors verwendet werden. Zur Ausrichtung an sich wurde dabei die Punktepaar-Methode verwendet. Dabei werden, wie auf der rechten Seite von Abbildung 7-2 dargestellt, markante Punkte, die bei beiden auszurichtenden Oberfläche vorhanden sind, gesetzt. Anhand der gesetzten Punktepaare errechnet *Best-Fit* näherungsweise die optimale Ausrichtung. Je passender dabei die Punkte gesetzt wurden, desto präziser ist danach das Ergebnis.

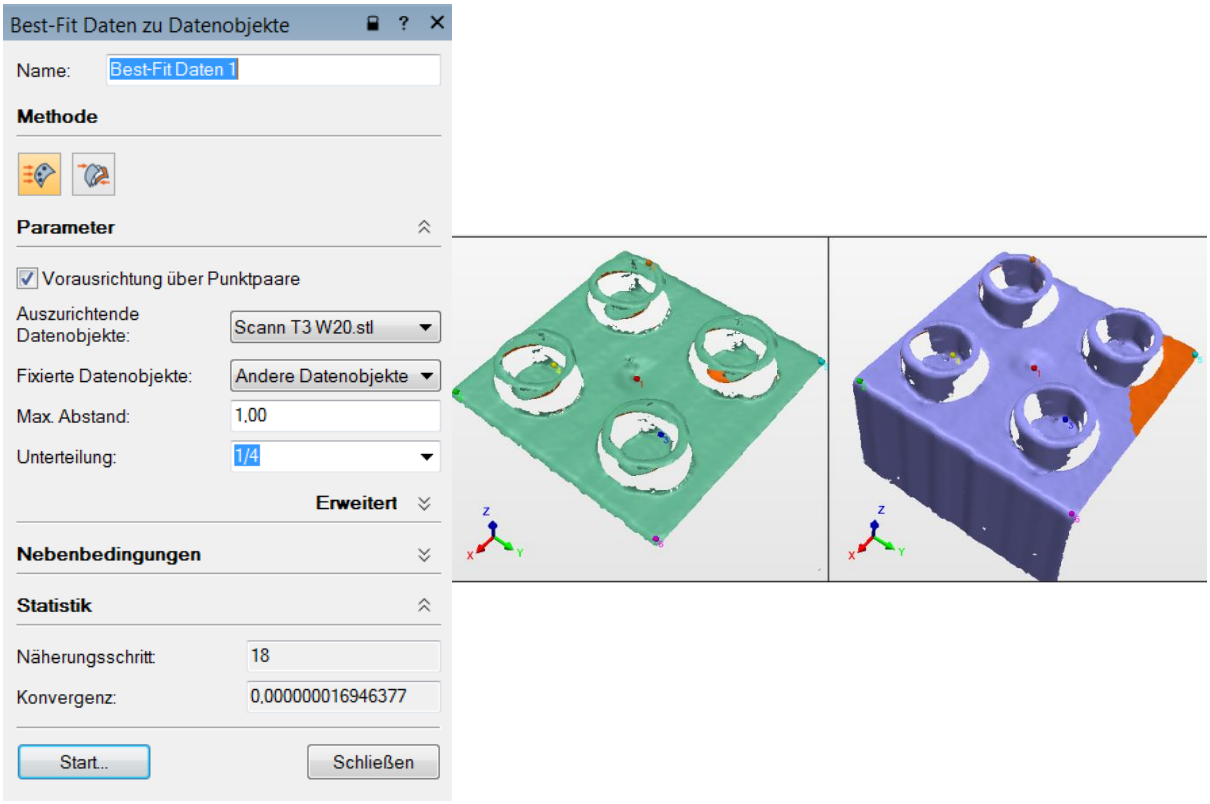


Abbildung 7-2: Einstellung der Best-Fit-Ausrichtung (links), Punktepaar-Ausrichtung (rechts)

Um die ausgerichteten STL-Modelle anschließend zusammenzufügen, muss ein Polygonmodell auf Basis der alten Modelle neu generiert werden. Hierzu werden alle Modelle ausgewählt und die Funktion *Polygonmodell erzeugen* angewendet. Die beiden Parameter, mit deren Hilfe die ausgerichteten Modelle auf der linken Seite der

Abbildung 7-3: Ausgerichtete Scans (links), Erzeugungseinstellungen (mittig), erzeugtes Modell (rechts)

in in das rechte Modell umgewandelt wurden, werden in der Mitte der Abbildung (in mm) angezeigt. Diese Aufgabe ist sehr rechenintensiv, weshalb die Ausführung einige Minuten dauern kann. Durch die komplexe Rechnung kann eine falsche Wahl der Parameter oder unzureichendes Ausrichten durchaus dazu führen, dass das Ergebnis nicht konvergiert und/oder das Programm abstürzt.

Am rechten Modell ist zu erkennen, dass die Lage des Legosteins unvorteilhaft gewählt wurde. Dieser wurde mit den Seitenflächen parallel zur Scanlinie positioniert. Da der Scanner diese nicht richtig aufnehmen konnte, sind die Eckkanten des Modells lückenhaft. Wäre der Stein mit einem Winkelversatz zur Laserlinie positioniert worden, wäre auch die Qualität dieser Kanten höher.

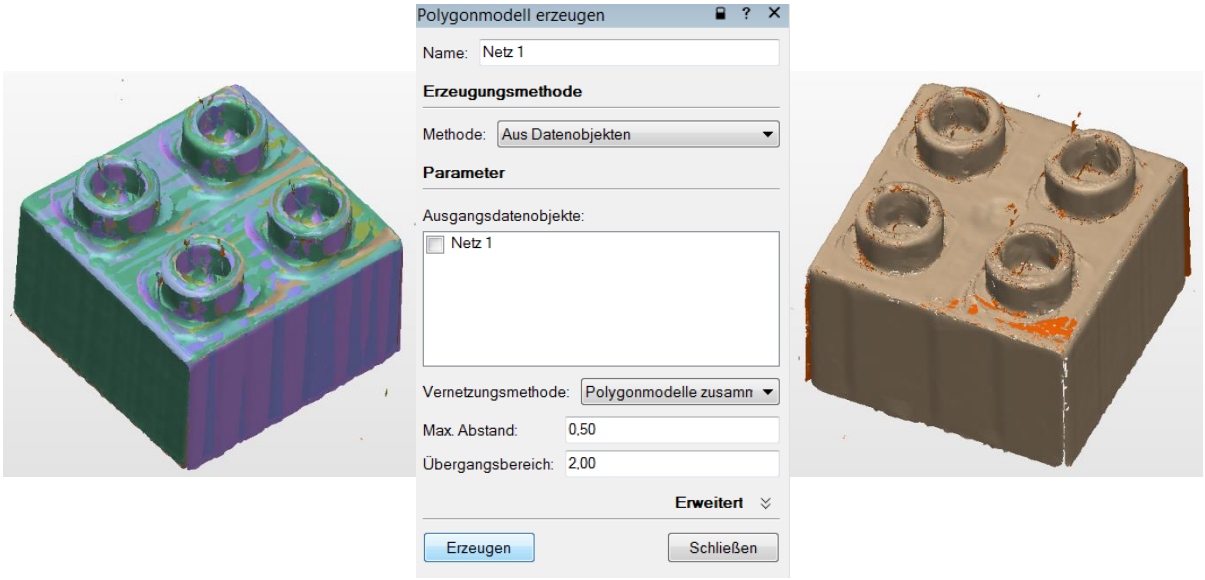


Abbildung 7-3: Ausgerichtete Scans (links), Erzeugungseinstellungen (mittig), erzeugtes Modell (rechts)

Das weitere Aufbereiten des neuen Modells, darunter auch das Reparieren der Kanten, geschieht im Modeler von PolyWorks. Durch nicht entfernte Ausreißer in den Punktwolken entstehen ungewollte Polygonfragmente. Besonders bei den Scans mit größerem Winkel sind diese häufiger anzutreffen. Es empfiehlt sich, die Ausreißer schon bei den Punktwolken auszufiltern, jedoch können auch im späteren Stadium die Fragmente ausgewählt und entfernt werden. Auf der linken Seite der Abbildung 7-4

ist das reparierte Modell aus der rechten Seite von Abbildung 7-3: Ausgerichtete Scans (links), Erzeugungseinstellungen (mittig), erzeugtes Modell (rechts) dargestellt. Dadurch, dass die Punktwolken umgewandelt, ausgerichtet und die entstanden Polygonmodelle neu verbunden wurden, sind an mehreren Stellen die Flächen falsch zusammengefügt worden. Dadurch sind Löcher und ungeordnete Polygonstrukturen entstanden, die auf der linken Seite der Abbildung 7-4 orange markiert und rechts als Detailansicht dargestellt sind. Um diese zu eliminieren, müssen alle fehlerhaften Elemente entfernt und das entstandene Loch geschlossen werden. Dies funktioniert auf ebenen Flächen sehr gut, allerdings können größere Löcher auf abgerundeten Flächen oder an Kanten nur schwierig geschlossen werden.

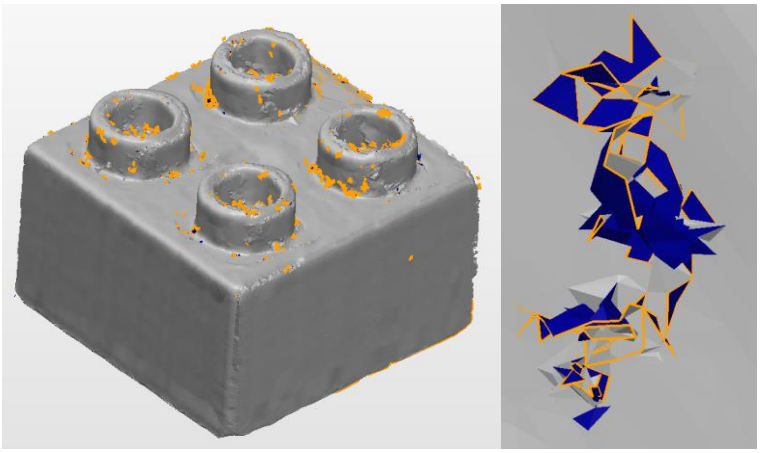


Abbildung 7-4: Repariertes Modell (links), Detailansicht einer fehlerhaften Polygonstruktur (rechts)

Werden die Punktwolken präzise ausgerichtet und danach erst in ein Polygonmodell überführt, entfällt der Großteil solcher Reparaturarbeiten. Auch ist der Rechenaufwand beim Erstellen des Gesamtmodells nicht so groß und fehleranfällig.

Das entstandene STL-Modell kann dann direkt an einen Slicer übergeben werden, um eine Kopie des gescannten Objekts zu drucken. Es kann aber auch mit einem anderen Modell kombiniert und weiter bearbeitet werden, um bspw. auf diesen Legostein einen weiteren zu drucken. Diese Methode kann auch dazu genutzt werden, um den Scan einer Oberfläche, bei der Stücke fehlen, mit der Solloberfläche zu vergleichen. Wird die gescannte Oberfläche von der Solloberfläche subtrahiert, kann das Ergebnis reparativ auf das Objekt aufgetragen werden. Zuvor muss dafür jedoch die Genauigkeit des Scanverfahrens erhöht werden. Während des Drucks muss außerdem vermehrt auf eventuelle Kollisionen geachtet werden.

8 Zusammenfassung und Fazit

Es konnte erfolgreich ein Scansystem aus Roboter, Laserlinienscanner und Computer entwickelt werden, das auf die Anforderungen des Objekts angepasst werden kann. Sowohl die Größe als auch die Oberflächengegebenheiten des Zielkörpers können im Scanprozess berücksichtigt werden. Dazu wird der Scanvorgang über eine Benutzeroberfläche konfiguriert, die dem Anwender eine zusätzliche Hilfestellung bietet. Im Anschluss wird automatisch eine Server-Clientverbindung zwischen Computer und Robotersteuerung über den Router aufgebaut. Die Kommunikation seitens der Steuerung ist dabei in Senden und Empfangen aufgeteilt. Das Senden wird während der Fahrt vom Submit-Interpreter ausgeführt. Das Empfangen findet am Anfang des Hauptprogramm statt und ist somit dem Roboter-Interpreter zugeteilt. Dabei greifen beide Interpreter auf denselben EKI-Kanal zu, weshalb nur eine Konfigurationsdatei benötigt wird und nur einmal eine Verbindung zum Python-Server aufgebaut werden muss.
Anhand der empfangenen Objekt- und Fahrtdaten wird die Trajektorie automatisch in einem Unterprogramm erstellt und ausgeführt. Während den Scanfahrten wird die Position und Orientierung dem Server mitgeteilt, der dann den Scanner ansteuert. Die empfangenen Daten werden mit den gescannten Daten verrechnet und je Scanfahrt wird daraus eine Punktwolke erstellt. Diese kann dann mit einem separaten Programm weiterverarbeitet werden.
Bei der Überprüfung der Scans mit einer Kalibrierplatte hat sich herausgestellt, dass die einzelnen Scanfahrten zueinander versetzt sind. Nach der Überprüfung wurden Wege aufgezeigt, wie es unter Verwendung von PolyWorks trotz der Versätze möglich ist, ein verwendbares Polygonmodell zu erstellen.

Das entwickelte Scansystem ist das erste System, das mit einer KRC4-Steuerung und einem Laserlinienscanner an das Objekt angepasst automatisch dreidimensional scannt und dabei die internen Positionsdaten des Roboters verwendet. Darin liegt

jedoch auch die Schwäche des Systems. Dadurch, dass die Positionsdaten vom Roboter genommen werden, ist die Genauigkeit der Scans an die Genauigkeit eben dieser gekoppelt. Bei den meisten anderen Scansystemen wird die Position des Scanners über Traking ermittelt. Das hat den Vorteil, dass der Roboter nur die Scanbewegung durchführen muss, wodurch diese Systeme unabhängig vom Roboter und seiner Steuerung sind.

Diese Arbeit konnte erfolgreich auf die Vorarbeit der Bachelorthesis GMRK aufbauen und soll wiederum in Kombination mit der Bachelorarbeit von Martin Gewieß und dem Masterprojekt von David Gärtner, die Basis zum reparativen 3D-Druck bilden. Dies wird unter anderem im Kapitel Ausblick vertieft und diskutiert.

9 Ausblick

Auch wenn mit dem momentanen System das Erstellen von 3D-Modellen möglich ist, so bietet es noch viele Verbesserungsmöglichkeiten. Ein Hauptgrund dieser Arbeit ist das reparative 3D-Drucken. Um dieses umzusetzen sind aber noch einige Probleme zu lösen. Insbesondre bei der Genauigkeit müssen sowohl das Scan- als auch das Drucksystem verbessert werden. Hierbei sollte die Genauigkeit des Scans im Verhältnis zum möglichen 3D-Druck stehen. In beiden Fällen muss der Roboter so genau wie möglich kalibriert werden, um entsprechend genau arbeiten zu können.
Ist eine entsprechende Genauigkeit gegeben, kann überlegt werden, inwieweit der Prozess automatisiert werden kann und soll. Ist die Qualität der Punktwolke gut genug, könnte sie mit einem Python-Bindig von MeshLab direkt in eine STL- oder OBJ-Datei umgewandelt werden. Das erzeugte Netz könnten dann mit einem Meshmixer-Binding für Python (Momentan allerdings nur unter Python 2.7 und Meshmixer 3.0 funktionstüchtig) weiterverarbeitet werden. Theoretisch könnte so eine einfache Geometrie in einem flüssigen Workflow repariert werden.

Ein wichtiger Punkt zur Verbesserung des Scanablaufs wäre die automatische Überprüfung der Trajektorie. So könnte direkt nach der Eingabe vom Python-Skript überprüft werden, ob alle abzufahrenden Punkte auch im Arbeitsbereich des Roboters liegen. Auch könnte die Trajektorie komplett simuliert werden, um eventuelle Kollisionen vorauszusagen. So könnte auch herausgefunden werden, ob die Trajektorie zu nah oder sogar durch eine Singularität geht. Ist die Simulation zuverlässig genug, kann sie Überprüfungsfahrt im Modus T1 wegfallen.

Das Programm könnte auch um die alternative Trajektorie erweitert werden, die am Ende von Kapitel 5.2.1 erwähnt wird. Somit könnte, je nach Situation die passendste Trajektorie ausgewählt werden. Hierzu müsste sowohl das Python-Skript als auch die Programme des Roboters angepasst werden.

Eine weiter nützliche Erweiterung wäre eine Funktion zur Positionsfindung. Soll ein Objekt additiv bedruckt werden und ist ein STL-Modell von diesem bereits vorhanden, könnte mit einer solchen Funktion das Modell im Koordinatensystem der Basis passend platziert werden. Auch könnte mit einer ähnlichen Funktion die Position von zu greifenden Objekten ermittelt werden, wobei hier allerdings Kameras mit Tiefenwahrnehmung vorteilhafter wären.

Auch die Benutzeroberfläche des Scansystems bietet Verbesserungspotential. Bspw. wäre das Speichern und Bearbeiten von Scankonfigurationen sehr nützlich. Zudem könnte dem Benutzter der prozentuale Fortschritt und die voraussichtlich verbleibende Zeit bis zum Abschluss des Scans angezeigt werden.

Literaturverzeichnis

KUKA ROBOTER GMBH (HRSG.): *KUKA.RobotSensorInterface 3.2.* Augsburg, 2013

KUKA ROBOTER GMBH (HRSG.): *KR Agilus sixx. Betriebsanleitung*. Augsburg, 2014a

KUKA ROBOTER GMBH (HRSG.): *System Software 8.3.* Augsburg, 2014b

KUKA ROBOTER GMBH (HRSG.): *KUKA.EthernetKRL 2.2.* Augsburg, 2014c

KUKA ROBOTER GMBH (HRSG.): *KR C4 compact: Bedienungsanleitung.* Augsburg, 2014d

KUKA ROBOTER GMBH (HRSG.): *Das Steuerungssystem der Zukunft_KR C4*, 2016

KUKA ROBOTER GMBH (HRSG.): *Einsatz und Programmierung von Industrierobotern*, 2013

Helmut Vonhoegen: *XML Einstieg, Praxis, Referenz 9. Auflage, Bonn 2018*

MICRO-EPSILON GmbH (HRSG.): *Betriebsanleitung scanCONTROL 29xx* - Überprüfungsdatum 2019-07-18

MICRO-EPSILON GmbH (HRSG.): *Supplement B to the scanCONTROL 2900 Manual*, 2015-09-01

MICRO-EPSILON GmbH (HRSG.): *scanCONTROL 2900 Quick Reference 1.15*, 2014-05-09

MICRO-EPSILON GmbH (HRSG.): *Glossar Laser-Linien-Triangulation* URL

https://www.micro-epsilon.de/service/glossar/Laser-Linien-Triangulation.html- Überprüfungsdatum 2019-07-29

MICRO-EPSILON GmbH (HRSG.): *3D-View: Interaktive 3D-Visualisierungssoftware* URL

https://www.micro-epsilon.de/2D_3D/laser-scanner/Software/scanCONTROL-3D-View/- Überprüfungsdatum 2019-08-18

Michael Schuth, Wassili Buerakov: *Handbuch Optische Messtechnik* München 2017

Golem Media GmbH (HRSG.): *SDK - Software Development Kit*. URL https://www.golem.de/specials/sdk/ - Überprüfungsdatum 2019-07-18

Johannes Ernesti ; Peter Kaiser: *Python 3 Das umfassende Handbuch 5. Auflage*. Bonn 2017

UHH FB Informatik (HRSG.): *IP, IP-Protokoll, IP-Adresse* URL http://www.informatik.uni-hamburg.de/TKRN/world/lernmodule/LMvk/Popup/IP.htm - Überprüfungsdatum 2019-07-18

Norbert Lossau: *3D-Kamera erfasst ihr räumliches Umfeld in Echtzeit*, 03.12.2002 URL https://www.welt.de/print-welt/article282990/3D-Kamera-erfasst-ihr-raeumliches-Umfeld-in-Echtzeit.html - Überprüfungsdatum 2019-07-28

SHAFAQ, Hasibullah ; FRISCHGESELL, Thomas ; BOHNERT, Carolina: *Gestengesteuerte Mensch-Roboter-Kollaboration.* Hamburg, 2017

Python Software Foundation (HRSG.): *ctypes - A foreign function library for Python* URL https://docs.python.org/2/library/ctypes.html - Überprüfungsdatum 2019-07-18

Anhang

1 Xml-Konfiguration: KukaScannerClientSPS

```
<ETHERNETKRL>
   <CONFIGURATION>
      <EXTERNAL>
            <IP>172.31.1.103</IP>
            <PORT>59152</PORT>
      </EXTERNAL>
      <INTERNAL>
         <ENVIRONMENT>Submit</ENVIRONMENT>
         <BUFFERING Mode="FIFO" Limit="512"/>
         <BUFFSIZE Limit="32768"/>
         <ALIVE Set_Flag="1"/>
      </INTERNAL>
   </CONFIGURATION>
   <RECEIVE>
      <XML>
         <ELEMENT Tag="Sensor/Read/PAnzahl" Type="INT"/>
         <ELEMENT Tag="Sensor/Read/Hoehe" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/Breite" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/Laenge" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/Scannerbreite" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/Scannerhoehe" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/Winkel" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/Geschwindigkeit" Type="REAL"/>
         <ELEMENT Tag="Sensor/Read/xyzabc" Type="FRAME"/>
         <ELEMENT Tag="Sensor" Set_Flag="998"/>
      </XML>
   </RECEIVE>
   <SEND>
      <XML>
         <ELEMENT Tag="Robot/Data/ActPos/X"/>
         <ELEMENT Tag="Robot/Data/ActPos/Y"/>
         <ELEMENT Tag="Robot/Data/ActPos/Z"/>
         <ELEMENT Tag="Robot/Data/ActPos/A"/>
         <ELEMENT Tag="Robot/Data/ActPos/B"/>
         <ELEMENT Tag="Robot/Data/ActPos/C"/>
         <ELEMENT Tag="Robot/Winkel"/>
         <ELEMENT Tag="Robot/Orientierung"/>
         <ELEMENT Tag="Robot/Fin"/>
      </XML>
   </SEND>
</ETHERNETKRL>
```

2 Userfolds des sps.sub-Programms

```
&ACCESS RVO
&COMMENT PLC on control
DEF sps( )
…
  ;FOLD USER DECL
  ; Please insert user defined declarations
  DECL EKI_STATUS RET
  ;ENDFOLD (USER DECL)
…
  ;FOLD USER INIT
  ; Please insert user defined initialization commands
    $FLAG[990]=FALSE
    $FLAG[991]=FALSE
    $FLAG[992]=FALSE
    RET=EKI_Init("KukaScannerClientSPS")
    RET=EKI_Open("KukaScannerClientSPS")
  ;ENDFOLD (USER INIT)

  LOOP
…
    ;FOLD USER PLC
    ;Make your modifications here
    IF (($FLAG[990]==TRUE) OR ($FLAG[991]==TRUE)) THEN
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Data/ActPos/X",$POS_ACT.X)
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Data/ActPos/Y",$POS_ACT.Y)
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Data/ActPos/Z",$POS_ACT.Z)
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Data/ActPos/A",$POS_ACT.A)
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Data/ActPos/B",$POS_ACT.B)
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Data/ActPos/C",$POS_ACT.C)
      RET=EKI_SetReal("KukaScannerClientSPS","Robot/Winkel",Scan_W)
      RET=EKI_SetInt("KukaScannerClientSPS","Robot/Orientierung",Scan_Ori)
      RET=EKI_Send("KukaScannerClientSPS","Robot")
      Scan_Ori=0
      $FLAG[990]=FALSE
      $FLAG[991]=FALSE
      RET=EKI_ClearBuffer("KukaScannerClientSPS","Robot")
      RET=EKI_ClearBuffer("KukaScannerClientSPS","Sensor/Read")
    ENDIF
    IF ($FLAG[992]==TRUE) THEN
      RET=EKI_SetInt("KukaScannerClientSPS","Robot/Orientierung",Scan_Ori)
      RET=EKI_SetInt("KukaScannerClientSPS","Robot/Fin",Scan_Fin)
      RET=EKI_Send("KukaScannerClientSPS","Robot")
      RET=EKI_Close("KukaScannerClientSPS")
      RET=EKI_Clear("KukaScannerClientSPS")
    ENDIF
      …
  ENDLOOP
  …
```

3 KRL-Programm: KukaScannerClient

```
&ACCESS RV01
&COMMENT
&REL 1
&PARAM DISKPATH = KRC:\R1\Program
&PARAM SensorITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
DEF KukaScannerClient( )
;FOLD Declaration
  INT i
  REAL H,B,L,SB,SH,v
  DECL REAL W[19]
  DECL FRAME PE
  DECL EKI_STATUS RET
;ENDFOLD (Declaration)
;FOLD INI
  ;FOLD BASISTECH INI
  BAS (#INITMOV,0)
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
    USERSPOT(#INIT)
    Scan_Ori=0
    Scan_Fin=0
    Scan_Breite=0
    Scan_Laenge=0
    H=0
    B=0
    L=0
    SB=0
    SH=0
    v=0
    PE = {x 0,y 0,z 0,a 0,b 0,c 0}
    FOR i = 1 TO 19
      W[i] = 360
    ENDFOR
  ;ENDFOLD (USER INI)
;ENDFOLD (INI)
;FOLD PTP HOME Vel=50 % DEFAULT;%{PE}%R 8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:50, 7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,50)
$H_POS=XHOME
PTP XHOME
;ENDFOLD
;FOLD EKI GET
  RET=EKI_GetRealArray("KukaScannerClientSPS","Sensor/Read/Winkel",W[])
  RET=EKI_GetReal("KukaScannerClientSPS","Sensor/Read/Hoehe",H)
  RET=EKI_GetReal("KukaScannerClientSPS","Sensor/Read/Breite",B)
  RET=EKI_GetReal("KukaScannerClientSPS","Sensor/Read/Laenge",L)
  RET=EKI_GetReal("KukaScannerClientSPS","Sensor/Read/Scannerbreite",SB)
  RET=EKI_GetReal("KukaScannerClientSPS","Sensor/Read/Scannerhoehe",SH)
  RET=EKI_GetReal("KukaScannerClientSPS","Sensor/Read/Geschwindigkeit",v)
  RET=EKI_GetFrame("KukaScannerClientSPS","Sensor/Read/xyzabc",PE)
  Scan_Hoehe=H
  Scan_Breite=B
  Scan_Laenge=L
  Scan_SB=SB
  Scan_SH=SH
  Scan_Geschwindigkeit=v
```

```
  Scan_Eckpunkt=PE
  FOR i = 1 TO 19
    Scan_Winkel[i]=W[i]
  ENDFOR
  RET=EKI_ClearBuffer("KukaScannerClientSPS","Sensor/Read")
;ENDFOLD (EKI GET)

wait for Scan_Breite<>0
wait for Scan_Laenge<>0

IF Scan_Hoehe >= 70 THEN
  TRAJEKTORIE_pb()
ENDIF
;FOLD PTP HOME Vel=50 % DEFAULT;%{PE}%R 8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:50, 7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,50)
$H_POS=XHOME
PTP XHOME
;ENDFOLD
END

DEF TRAJEKTORIE_pb()
;FOLD Declaration
  BOOL scan
  INT PA,Bias,i,j,k,l
  REAL Abstand, Vorlauf, Beschleunigung
  DECL FRAME P[148]
  DECL FRAME PW[148,19]
  DECL INT Ori[148]
  DECL BOOL Scanfahrt[148]
  DECL EKI_STATUS RET
;ENDFOLD (Declaration)
;FOLD INI
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
    j = Scan_Breite/Scan_SB+0.5 ;Anzahl der Fahrten in y-Richtung
    k = Scan_Laenge/Scan_SB+0.5 ;Anzahl der Fahren in x-Richtung
    PA = 2*(j+k)
    i = 1
    Bias = 1
    scan = TRUE
    Abstand = 0
    Beschleunigung = 0.5 ;m/s^2
    Vorlauf = 1.1*Scan_Geschwindigkeit*Scan_Geschwindigkeit/Beschleunigung+15
    ;10% Sicherheit für die nichtlineare Besch. und 15mm Pauschale
    FOR i = 1 TO PA
      P[i] = {x 0,y 0,z 0,a 0,b 0,c 180}
      FOR l = 1 TO 19
        PW[i,l] = {x 0,y 0,z 0,a 0,b 0,c 180}
      ENDFOR
      Ori[i] = 0
      Scanfahrt[i] = FALSE
    ENDFOR
  ;ENDFOLD (USER INI)
;ENDFOLD (INI)
;FOLD Erster Punkt in y-Richtung, z und W
  P[1].x = Scan_Eckpunkt.x+Scan_SB/2
  P[1].y = Scan_Eckpunkt.y-Vorlauf
  FOR l = 1 TO 19
    IF Scan_Winkel[l] <> 360 THEN
```

```
      PW[1,l].x = P[1].x+Scan_SH*SIN(Scan_Winkel[l])
      PW[1,l].y = P[1].y
    ENDIF
  ENDFOR
  FOR i = 1 TO PA
    P[i].z = Scan_Hoehe+Scan_SH
    FOR l = 1 TO 19
      IF Scan_Winkel[l] <> 360 THEN
        PW[i,l].z = Scan_Hoehe+Scan_SH*COS(Scan_Winkel[l])
        PW[i,l].b = Scan_Winkel[l]
      ENDIF
    ENDFOR
  ENDFOR
;ENDFOLD (Erster Punkt in y-Richtung, z und W)
;FOLD Punkte für Fahrten in y-Richtung
  FOR i = 2 TO 2*j
    IF scan == TRUE THEN
      P[i].x = P[i-1].x
      P[i].y = P[i-1].y+Bias*(Scan_Laenge+2*Vorlauf)
      FOR l = 1 TO 19
        IF Scan_Winkel[l] <> 360 THEN
          PW[i,l].x = P[i].x+Scan_SH*SIN(Scan_Winkel[l])
          PW[i,l].y = P[i].y
        ENDIF
      ENDFOR
      Ori[i] = Bias
      Scanfahrt[i] = TRUE
      Bias = -Bias
      scan = FALSE
    ELSE
      P[i].x = P[i-1].x+Scan_SB
      P[i].y = P[i-1].y
      FOR l = 1 TO 19
        IF Scan_Winkel[l] <> 360 THEN
          PW[i,l].x = P[i].x+Scan_SH*SIN(Scan_Winkel[l])
          PW[i,l].y = P[i].y
        ENDIF
      ENDFOR
      Scanfahrt[i] = FALSE
      scan = TRUE
    ENDIF
  ENDFOR
;ENDFOLD (Punkte für Fahrten in y-Richtung)
;FOLD Erster Punkt in x-Richtung
  scan = TRUE
  P[2*j+1].x = Scan_Eckpunkt.x-Vorlauf
  P[2*j+1].y = Scan_Eckpunkt.y+Scan_SB/2
  P[2*j+1].a = 90
    FOR l = 1 TO 19
      IF Scan_Winkel[l] <> 360 THEN
        PW[2*j+1,l].x = P[2*j+1].x
        PW[2*j+1,l].y = P[2*j+1].y+Scan_SH*SIN(Scan_Winkel[l])
        PW[2*j+1,l].a = 90
      ENDIF
    ENDFOR
  Bias = 1
;ENDFOLD (Erster Punkt in x-Richtung)
;FOLD Punkte für Fahrten in x-Richtung
  FOR i = 2*j+2 TO 2*(j+k)
    IF scan == TRUE THEN
      P[i].x = P[i-1].x+Bias*(Scan_Breite+2*Vorlauf)
      P[i].y = P[i-1].y
      P[i].a = 90
      FOR l = 1 TO 19
        IF Scan_Winkel[l] <> 360 THEN
          PW[i,l].x = P[i].x
          PW[i,l].y = P[i].y+Scan_SH*SIN(Scan_Winkel[l])
```

```
          PW[i,l].a = 90
        ENDIF
      ENDFOR
      Ori[i] = Bias*2
      Scanfahrt[i] = TRUE
      Bias = -Bias
      scan = FALSE
    ELSE
      P[i].x = P[i-1].x
      P[i].y = P[i-1].y+Scan_SB
      P[i].a = 90
      FOR l = 1 TO 19
        IF Scan_Winkel[l] <> 360 THEN
          PW[i,l].x = P[i].x
          PW[i,l].y = P[i].y+Scan_SH*SIN(Scan_Winkel[l])
          PW[i,l].a = 90
        ENDIF
      ENDFOR
      Scanfahrt[i] = FALSE
      scan = TRUE
    ENDIF
  ENDFOR
;ENDFOLD (Punkte für Fahrten in x-Richtung)

BAS (#PTP_PARAMS, 10)
$TOOL = TOOL_DATA[2]
$ADVANCE = 0
Scan_W = 0

;FOLD Fahrten
  FOR l = 1 TO 19
    IF Scan_Winkel[l] <> 360 THEN
      FOR i = 1 TO PA
        IF (Scanfahrt[i] == TRUE) THEN
          RET=EKI_ClearBuffer("KukaScannerClientSPS","Sensor/Read")
          RET=EKI_ClearBuffer("KukaScannerClientSPS","Robot")
          Scan_Ori = Ori[i]
          Scan_W = Scan_Winkel[l]
          $ACC.CP = Beschleunigung
          $VEL.CP = Scan_Geschwindigkeit
          $ORI_TYPE = #CONSTANT

          IF ((Ori[i]*Ori[i]) == 1) THEN
            Abstand = 12+Scan_Laenge+Vorlauf
          ELSE
            Abstand = 12+Scan_Breite+Vorlauf
          ENDIF

          TRIGGER WHEN PATH=-Abstand DELAY=0 DO $FLAG[990]=TRUE
          SLIN PW[i,l]

          $FLAG[991] = TRUE
        ELSE
          SPTP PW[i,l]
        ENDIF
      ENDFOR
    ENDIF
  ENDFOR
;ENDFOLD (Fahrten)

Scan_Fin = 1
$FLAG[992] = TRUE

END
```

4 Python-Programm: GUI Server & Scannen

```
from tkinter import *
from tkinter import messagebox
from tkinter import ttk
from PIL import Image
from PIL import ImageTk
import ctypes as ct
import time
import datetime
import os
import pyllt as llt
import _thread
import math
import socket
import xml.etree.ElementTree as ET

Nachkommastellen_in_der_asc = 3
Warmuptime = 0.1
max_geschwindigkeit = 200
Winkel = [0]*19
wi = 1
wj = 1
while wi < 19:
    Winkel[wi] = 10*wj
    Winkel[wi+1] = -10*wj
    wj += 1
    wi += 2
halterWK = 0

################################################
#################### GUI #######################
################################################
font_klein = ("Verdana", 12)
font_gross = ("Verdana", 18)

def schliessfrage():
    if messagebox.askyesno('Fenster schließen',
                           'Fenster wirklich schließen?'):
        root.destroy()

def raise_frame(frame):
    frame.tkraise()

root = Tk()
root.title("Mit Kuka scannen")
root.iconbitmap(default="haw-hamburg-logo.ico")
root.protocol("WM_DELETE_WINDOW",schliessfrage)

f1 = Frame(root)
f2 = Frame(root)
f1.grid(row=0, column=0, sticky='nsew')
f2.grid(row=0, column=0, sticky='nsew')
raise_frame(f1)

def frame1():
    s = ttk.Style()
    s.configure('my.TButton', font=font_klein)
    Label(f1, text='\n\n\n\n\n\n\n', font=font_gross).pack(anchor='n')
    Label(f1, text='3D-Scannen mit einem Laserlinienscanner und \neinem KUKA Agilus an
einer KRC4-Steuerung.\n' , font=font_gross).pack()
```

```
    Label(f1, text='\nGetestet mit einem scanCONTROL2950-100 von Micro-Epsilon.\n\n\n',
font=font_klein).pack()
    ttk.Button(f1, style='my.TButton', text='Konfiguration starten',
command=lambda:raise_frame(f2)).pack(anchor='s')
frame1()

def standardeinstellung_se_set():
    standard_se = []
    standard_se += [0.1]  # Belichtungszeit
    standard_se += [100]  # Profilfrequenz
    standard_se += [(1 / (standard_se[1] * 0.001)) - standard_se[0]]  # Wartezeit
    standard_se += [640]  # Aufloesung_t
    standard_se += [0]  # Messfeld
    standard_se += [80]  # Messfeldbreite
    standard_se += [240]  # Messfeldmitte
    return standard_se

def standardeinstellung_pf_set():
    standard_pf = []
    standard_pf += [1]  # Resampling
    standard_pf += [0]*2  # Median & Average
    return standard_pf

def standardeinstellung_of_set():
    standard_of = []
    standard_of += [4]  # Winkelanzahl
    standard_of += [10]  # Geschwindigkeit
    standard_of += [200]  # Objekbreite X-Richtung
    standard_of += [200]  # Objektlänge Y-Richtung
    standard_of += [50]  # Objekthöhe
    standard_of += [555]  # X-Pos Eckpunkt
    standard_of += [-100]  # Y-Pos Eckpunkt
    standard_of += [72]  # Z-Pos Eckpunkt; Basishöhe(69) + Gummimatte(3); 72
    return standard_of

def frame2():
    Zeile = 0
    eingabe_se_z = 0
    eingabe_pf_z = 0
    eingabe_of_z = 0
    eingabe_se = []
    eingabe_pf = []
    eingabe_of = []

    s = ttk.Style()
    s.configure('my.TButton', font=font_klein)

    Scannereinstellungen = ttk.Label(f2, text="Scannereinstellungen", font=font_gross)
    Scannereinstellungen.grid(column=0, row=Zeile)
    Hinweis = ttk.Label(f2, text="Hinweis", font=font_gross)
    Hinweis.grid(column=3, row=Zeile)

    ######################################################################
Belichtungszeit
    Zeile += 1
    belichtungszeit_label = ttk.Label(f2, text="Belichtungszeit", font=font_klein)
    belichtungszeit_label.grid(column =0, row=Zeile)

    eingabe_se += [ttk.Entry(f2, width=10)]
    eingabe_se[eingabe_se_z].grid(column=1, row=Zeile)
    #eingabe_se[eingabe_se_z].focus
    eingabe_se_z += 1
    belichtungszeit_t = 0.1

    belichtungszeit_einheit = ttk.Label(f2, text="[ms]", font=font_klein)
    belichtungszeit_einheit.grid(column =2, row=Zeile)
    belichtungszeit_hinweis = ttk.Label(f2, text="0.01 bis 40", font=font_klein)
```

```
    belichtungszeit_hinweis.grid(column =3, row=Zeile)

    def belichtungszeit_f():
        windowBF = Toplevel()
        windowBF.title("Empfohlene Belichtungszeichen nach scanCONTROL 2900 Quick
Reference")
        windowBF.geometry("649x700")
        windowBF.configure(background='grey')

        path = "Empfohlene_Belichtungszeit.png"

        # Erstellt ein Tkinter-kompatibles Fotobild
        img = ImageTk.PhotoImage(Image.open(path))
        panel = ttk.Label(windowBF, image=img)
        panel.pack(side="bottom", fill="both", expand="yes")
        windowBF.mainloop()

    belichtungszeit_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=belichtungszeit_f) #
    belichtungszeit_button.grid(column=4, row=Zeile)

    ######################################################################
Profilfrequenz
    Zeile += 1

    profilfrequenz_label = ttk.Label(f2, text="Profilfrequenz", font=font_klein)
    profilfrequenz_label.grid(column =0, row=Zeile)

    eingabe_se += [ttk.Entry(f2, width=10)]
    eingabe_se[eingabe_se_z].grid(column=1, row=Zeile)
    eingabe_se_z += 1

    profilfrequenz_einheit = ttk.Label(f2, text="[Hz]", font=font_klein)
    profilfrequenz_einheit.grid(column =2, row=Zeile)
    profilfrequenz_hinweis = ttk.Label(f2, text="bis 2000", font=font_klein)
    profilfrequenz_hinweis.grid(column =3, row=Zeile)

    def profilfrequenz_f():
        windowPF = Toplevel()
        windowPF.title("Maximale Profilfrequenz in abhängigkeit zum Messfeld nach
scanCONTROL 2900 Quick Reference")
        windowPF.geometry("1176x769")
        windowPF.configure(background='grey')

        path = "maxHz-1.png"
        img = ImageTk.PhotoImage(Image.open(path))
        panel = ttk.Label(windowPF, image=img)
        panel.pack(side="bottom", fill="both", expand="yes")
        windowPF.mainloop()


    messfeldbreite_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=profilfrequenz_f)
    messfeldbreite_button.grid(column=4, row=Zeile)
    ###################################################################### Wartezeit
    Zeile += 1

    wartezeit_label = ttk.Label(f2, text="Wartezeit", font=font_klein)
    wartezeit_label.grid(column =0, row=Zeile)

    eingabe_se += [ttk.Entry(f2, width=10)]
    eingabe_se[eingabe_se_z].grid(column=1, row=Zeile)
    eingabe_se_z += 1

    wartezeit_einheit = ttk.Label(f2, text="[ms]", font=font_klein)
    wartezeit_einheit.grid(column =2, row=Zeile)
    wartezeit_hinweis = ttk.Label(f2, text="0.01 bis 40", font=font_klein)
```

```
    wartezeit_hinweis.grid(column=3, row=Zeile)

    ########################################################################## Auflösung
    Zeile += 1

    aufloesung_label = ttk.Label(f2, text="Auflösung", font=font_klein)
    aufloesung_label.grid(column=0, row=Zeile)

    eingabe_se += [IntVar(f2)]
    choices_Aufloesung = [0,160, 320, 640, 1280]

    aufloesung_popupMenu = ttk.OptionMenu(f2, eingabe_se[eingabe_se_z],
choices_Aufloesung[3], *sorted(choices_Aufloesung))
    aufloesung_popupMenu.grid(column=1, row=Zeile)
    eingabe_se_z += 1

    ########################################################################## Messfeld
    Zeile += 1

    messfeld_label = ttk.Label(f2, text="Messfeld", font=font_klein)
    messfeld_label.grid(column =0, row=Zeile)

    eingabe_se += [ttk.Entry(f2, width=10)]
    eingabe_se[eingabe_se_z].grid(column=1, row=Zeile)
    eingabe_se_z += 1

    messfeld_einheit = ttk.Label(f2, text="", font=font_klein)
    messfeld_einheit.grid(column =2, row=Zeile)
    messfeld_hinweis = ttk.Label(f2, text="int 0 bis 127", font=font_klein)
    messfeld_hinweis.grid(column =3, row=Zeile)

    def messfeld_funktion():
        windowMF = Toplevel()
        windowMF.title("Aus der Betriebsanleitung scanCONTROL 29xx Abb. 13
Vordefinierte Messfelder")
        windowMF.geometry("700x1000")
        windowMF.configure(background='grey')

        path = "MeasuringField.png"
        img = ImageTk.PhotoImage(Image.open(path))
        panel = ttk.Label(windowMF, image=img)
        panel.pack(side="bottom", fill="both", expand="yes")
        windowMF.mainloop()

    messfeld_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=messfeld_funktion)
    messfeld_button.grid(column=4, row=Zeile)
    ##########################################################################
Messfeldbreite
    Zeile += 1

    messfeldbreite_label = ttk.Label(f2, text="Messfeldbreite", font=font_klein)
    messfeldbreite_label.grid(column =0, row=Zeile)

    eingabe_se += [ttk.Entry(f2, width=10)]
    eingabe_se[eingabe_se_z].grid(column=1, row=Zeile)
    eingabe_se_z += 1

    messfeldbreite_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    messfeldbreite_einheit.grid(column =2, row=Zeile)
    messfeldbreite_hinweis = ttk.Label(f2, text="15 bis 145", font=font_klein)
    messfeldbreite_hinweis.grid(column =3, row=Zeile)

    def messfeldbreite_f():
        windowMBF = Toplevel()
        windowMBF.title("Aus der Betriebsanleitung scanCONTROL 29xx Abb.5 Abmaße des
Messfelds")
```

```
        windowMBF.geometry("996x1011")
        windowMBF.configure(background='grey')

        path = "Messfeld_Breite-Hoehe.png"
        img = ImageTk.PhotoImage(Image.open(path))
        panel = ttk.Label(windowMBF, image=img)
        panel.pack(side="bottom", fill="both", expand="yes")
        windowMBF.mainloop()

    messfeldbreite_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=messfeldbreite_f)
    messfeldbreite_button.grid(column=4, row=Zeile)
    ###################################################################### Messfeldmitte
    Zeile += 1

    messfeldmitte_label = ttk.Label(f2, text="Messfeldmitte", font=font_klein)
    messfeldmitte_label.grid(column=0, row=Zeile)

    eingabe_se += [ttk.Entry(f2, width=10)]
    eingabe_se[eingabe_se_z].grid(column=1, row=Zeile)
    eingabe_se_z += 1

    messfeldmitte_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    messfeldmitte_einheit.grid(column=2, row=Zeile)
    messfeldmitte_hinweis = ttk.Label(f2, text="125 bis 390", font=font_klein)
    messfeldmitte_hinweis.grid(column=3, row=Zeile)

    def messfeldmitte_f():
        messagebox.showinfo('Messfeldmitte ', 'Die Messfeldbreite und Mitte hängen von
der ausgewählten Messfeldnummer '
                                              'ab und müssen anhand der Abbildung 5 und
13 aus der Betriebsanleitung '
                                              'scanCONTROL 29xx abgeschätzt werden.')

    messfeldmitte_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=messfeldmitte_f)
    messfeldmitte_button.grid(column=4, row=Zeile)
    ######################################################################
Standardeinstellung SE
    Zeile += 1

    standardeinstellung_SE_label = ttk.Label(f2, text="Standardeinstellung",
font=font_klein)
    standardeinstellung_SE_label.grid(column =0, row=Zeile)

    standardeinstellung_SE_t = BooleanVar()
    standardeinstellung_SE_t.set(False)
    Standardeinstellung_SE_eingabe = ttk.Checkbutton(f2,
variable=standardeinstellung_SE_t)
    Standardeinstellung_SE_eingabe.grid(column=1, row=Zeile)

    def standardeinstellung_SE_f():
        messagebox.showinfo('Standardeinstellung von Scannereinstellungen
','Belichtungszeit: 0.1ms \nProfilfrequenz: '
                                                                    '100Hz
\nWartezeit: (1/(Profielfrequenz'
                                                                    '*0.001))-
Belichtungszeit \nAufloesung: 640 '

'\nMessfeld: 5 \nMessfeldbreite: 80mm '

'\nMessfeldmitte: 240mm \n')

    standardeinstellung_SE_button = ttk.Button(f2, text="?", style='my.TButton',
width=2, command=standardeinstellung_SE_f) #
    standardeinstellung_SE_button.grid(column=4, row=Zeile)
```

```
##########################################################################################
############### Profilfilter

##########################################################################################
############################
    Zeile += 1

    leerzeichen = ttk.Label(f2, text=" ", font=font_gross)
    leerzeichen.grid(column=0, row=Zeile)
    Zeile += 1

    profilfilter_label = ttk.Label(f2, text="Profilfilter", font=font_gross)
    profilfilter_label.grid(column=0, row=Zeile)

    ########################################################################## Resampling
    Zeile += 1

    resampling_label = ttk.Label(f2, text="Resampling", font=font_klein)
    resampling_label.grid(column =0, row=Zeile)

    eingabe_pf += [IntVar(f2)]
    choices_Resampling = range(8)

    resampling_popupMenu = ttk.OptionMenu(f2, eingabe_pf[eingabe_pf_z],
choices_Resampling[0], *sorted(choices_Resampling))
    resampling_popupMenu.grid(column=1, row=Zeile)
    eingabe_pf_z += 1

    resampling_hinweis = ttk.Label(f2, text="int 0 bis 7", font=font_klein)
    resampling_hinweis.grid(column =3, row=Zeile)

    ########################################################################## Median
    Zeile += 1

    median_label = ttk.Label(f2, text="Median", font=font_klein)
    median_label.grid(column =0, row=Zeile)

    eingabe_pf += [IntVar(f2)]

    choices_Median = range(4)

    median_popupMenu = ttk.OptionMenu(f2, eingabe_pf[eingabe_pf_z], choices_Median[0],
*sorted(choices_Median))
    median_popupMenu.grid(column=1, row=Zeile)
    eingabe_pf_z += 1

    median_hinweis = ttk.Label(f2, text="int 0 bis 3", font=font_klein)
    median_hinweis.grid(column =3, row=Zeile)

    ########################################################################## Average
    Zeile += 1

    average_label = ttk.Label(f2, text="Average", font=font_klein)
    average_label.grid(column =0, row=Zeile)

    eingabe_pf += [IntVar(f2)]
    choices_Average = range(4)

    average_popupMenu = ttk.OptionMenu(f2, eingabe_pf[eingabe_pf_z],
choices_Average[0], *sorted(choices_Average))
    average_popupMenu.grid(column=1, row=Zeile)
    eingabe_pf_z += 1

    average_hinweis = ttk.Label(f2, text="int 0 bis 3", font=font_klein)
    average_hinweis.grid(column =3, row=Zeile)
```

```
    ######################################################################
Standardeinstellung PF
    Zeile += 1

    standardeinstellung_PF_label = ttk.Label(f2, text="Standardeinstellung",
font=font_klein)
    standardeinstellung_PF_label.grid(column =0, row=Zeile)

    standardeinstellung_PF_t = BooleanVar()
    standardeinstellung_PF_t.set(False)
    Standardeinstellung_PF_eingabe = ttk.Checkbutton(f2,
variable=standardeinstellung_PF_t)
    Standardeinstellung_PF_eingabe.grid(column=1, row=Zeile)

    def standardeinstellung_PF_f():
        messagebox.showinfo('Standardeinstellung von Profilfilter','Resampling: 1
\nMedian: 0'
                                                                ' \nAverage: 0')

    standardeinstellung_PF_button = ttk.Button(f2, text="?", style='my.TButton',
width=2, command=standardeinstellung_PF_f)
    standardeinstellung_PF_button.grid(column=4, row=Zeile)


##############################################################################################
####### Objekt und Fahrtdaten

##############################################################################################
############################
    Zeile += 1

    leerzeichen2 = ttk.Label(f2, text=" ", font=font_gross)
    leerzeichen2.grid(column=0, row=Zeile)
    Zeile += 1

    obufada_label = ttk.Label(f2, text="Objekt und Fahrtdaten", font=font_gross)
    obufada_label.grid(column=0, row=Zeile)

    ###################################################################### Winkelanzahl
    Zeile += 1
    winkelanzahl_label = ttk.Label(f2, text="Winkelanzahl", font=font_klein)
    winkelanzahl_label.grid(column =0, row=Zeile)

    eingabe_of += [IntVar(f2)]

    choices_Winkelanzahl = range(1,20)

    winkelanzahl_popupMenu = ttk.OptionMenu(f2, eingabe_of[eingabe_of_z],
choices_Winkelanzahl[0], *sorted(choices_Winkelanzahl))
    winkelanzahl_popupMenu.grid(column=1, row=Zeile)
    eingabe_of_z += 1

    winkelanzahl_hinweis = ttk.Label(f2, text="int 1 bis 19", font=font_klein)
    winkelanzahl_hinweis.grid(column =3, row=Zeile)

    def winkelanzah_f():
        messagebox.showinfo('Winkelanzahl und Konfiguration','Die Winkelanzahl bestimmt
die Anzahl an zu fahrenden '
                                                            'Winkeln. Wird bspw. 4
ausgewählt und die Konfiguration '
                                                            'nicht geändert, so wird
mit den Winkeln 0°,10°,-10° und '
                                                            '20° gefahren. \nIn der
Winkelkonfiguration können gezielt'
                                                            ' Winkel geändert werden.
Zeilen die leer bleiben behalten'
                                                            ' den Standardwert und bei
```

```
mehrmaligen Übernehmen, werden '
                                                    'nur die Winkel mit
ausgefüllten Zeile überschrieben.')
    winkelanzahl_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=winkelanzah_f) #
    winkelanzahl_button.grid(column=4, row=Zeile)

    ######################################################################
Winkelkonfiguration
    Zeile += 1

    winkelkonfiguration_label = ttk.Label(f2, text="Winkel", font=font_klein)
    winkelkonfiguration_label.grid(column =0, row=Zeile)
    winkelkonfiguration_einheit = ttk.Label(f2, text="[ °]", font=font_klein)
    winkelkonfiguration_einheit.grid(column =2, row=Zeile)
    winkelkonfiguration_hinweis = ttk.Label(f2, text="Default: 10°/-10° Schritte",
font=font_klein)
    winkelkonfiguration_hinweis.grid(column =3, row=Zeile)

    def winkelkonfiguration_f():
        windowWK = Toplevel()
        windowWK.title("Winkelkonfiguration")
        #windowWK.geometry("700x1000")
        #windowWK.configure(background='grey')
        winkel_label = []
        Winkel_eingabe = []
        winkel_einheit = []

        leerzeichenW1 = ttk.Label(windowWK, text=" ", font=font_klein)
        leerzeichenW1.grid(column=0, row=0)

        winkel_hinweis = ttk.Label(windowWK, text=" Die Winkel müssen zwischen -90° und
90° liegen.", font=font_klein)
        winkel_hinweis.grid(column=0, row=1, columnspan=3)

        leerzeichenW2 = ttk.Label(windowWK, text=" ", font=font_klein)
        leerzeichenW2.grid(column=0, row=2)

        for i in range(19):
            winkel_label += [ttk.Label(windowWK, text=str(i+1)+". Winkel ",
font=font_klein)]
            winkel_label[i].grid(column=0, row=i+3)
            Winkel_eingabe += [ttk.Entry(windowWK, width=10)]
            Winkel_eingabe[i].grid(column=1, row=i+3)
            winkel_einheit += [ttk.Label(windowWK, text="[ °]", font=font_klein)]
            winkel_einheit[i].grid(column=2, row=i+3)

        def uebernehmen_f():
            halter = 0
            for ii in range(19):
                try:
                    wiflo = float(Winkel_eingabe[ii].get())
                    if abs(wiflo) < 90:
                        Winkel[ii] = wiflo
                    else:
                        halter += 1
                        messagebox.showinfo('Ungültige Eingabe',
                                            'Die Eingabe vom '+str(ii+1)+'. Winkel ist
ungültige. '
                                            '\n Dieser muss zwischen -90° und 90°
liegen.')
                except:
                    if str(Winkel_eingabe[ii].get()) == '':
                        pass
                    else:
                        halter += 1
                        messagebox.showinfo('Ungültige Eingabe',
```

```
                                                'Die Eingabe vom ' + str(ii + 1) + '.
Winkel ist keine Zahl.')
            print(Winkel)
            if halter == 0:
                windowWK.destroy()

        leerzeichenW2 = ttk.Label(windowWK, text="  ", font=font_klein)
        leerzeichenW2.grid(column=4, row=i+4)

        uebernehmen = ttk.Button(windowWK, text="Übernehmen", style='my.TButton',
width=11, command=uebernehmen_f)
        uebernehmen.grid(column=0, row=i+5)

        windowWK.mainloop()

    winkelkonfiguration_button = ttk.Button(f2, text="Konfiguration",
style='my.TButton', width=12, command=winkelkonfiguration_f) #
    winkelkonfiguration_button.grid(column=1, row=Zeile)

    ######################################################################
Geschwindigkeit
    Zeile += 1
    geschwindigkeit_label = ttk.Label(f2, text="Geschwindigkeit", font=font_klein)
    geschwindigkeit_label.grid(column =0, row=Zeile)

    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)
    eingabe_of_z += 1

    geschwindigkeit_einheit = ttk.Label(f2, text="[mm/s]", font=font_klein)
    geschwindigkeit_einheit.grid(column =2, row=Zeile)
    geschwindigkeit_hinweis = ttk.Label(f2, text="0.1 bis "+str(max_geschwindigkeit),
font=font_klein)
    geschwindigkeit_hinweis.grid(column =3, row=Zeile)

    def geschwindigkeit_f():
        messagebox.showinfo('Geschwindigkeit','Die Geschwindigkeit muss mit der
Auflösung und Profilfrequenz abgestimmt '
                                              'werden.')

    geschwindigkeit_button = ttk.Button(f2, text="?", style='my.TButton', width=2,
command=geschwindigkeit_f)
    geschwindigkeit_button.grid(column=4, row=Zeile)

    ###################################################################### Objektbreite
    Zeile += 1

    objektbreite_label = ttk.Label(f2, text="Objektbreite (X-Richtung)",
font=font_klein)
    objektbreite_label.grid(column =0, row=Zeile)

    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)
    eingabe_of_z += 1

    objektbreite_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    objektbreite_einheit.grid(column =2, row=Zeile)
    objektbreite_hinweis = ttk.Label(f2, text="0 bis 600", font=font_klein)
    objektbreite_hinweis.grid(column =3, row=Zeile)

    ###################################################################### Objektlänge
    Zeile += 1

    objektlaenge_label = ttk.Label(f2, text="Objektlänge (Y-Richtung)",
font=font_klein)
    objektlaenge_label.grid(column =0, row=Zeile)
```

```
    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)
    eingabe_of_z += 1

    objektlaenge_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    objektlaenge_einheit.grid(column =2, row=Zeile)
    objektlaenge_hinweis = ttk.Label(f2, text="0 bis 800", font=font_klein)
    objektlaenge_hinweis.grid(column =3, row=Zeile)

    ###################################################################### Objekthoehe
    Zeile += 1

    objekthoehe_label = ttk.Label(f2, text="Objekthöhe", font=font_klein)
    objekthoehe_label.grid(column =0, row=Zeile)

    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)
    eingabe_of_z += 1

    objekthoehe_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    objekthoehe_einheit.grid(column =2, row=Zeile)
    objekthoehe_hinweis = ttk.Label(f2, text="0 bis 1000", font=font_klein)
    objekthoehe_hinweis.grid(column =3, row=Zeile)

    ###################################################################### X-Position
    Zeile += 1

    posX_label = ttk.Label(f2, text="X-Position des Eckpunkts", font=font_klein)
    posX_label.grid(column =0, row=Zeile)

    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)
    eingabe_of_z += 1

    posX_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    posX_einheit.grid(column =2, row=Zeile)
    posX_hinweis = ttk.Label(f2, text="400 bis 1000", font=font_klein)
    posX_hinweis.grid(column =3, row=Zeile)

    def pos_f():
        messagebox.showinfo('Position des Eckpunkts','Der untere(kleinstes Z) Eckpunkt
hinten(kleinstes X) '
                                                     'links(kleinstes Y) des Packmaßes
muss auf 5mm genau diese '
                                                     'Position haben.')

    posX_button = ttk.Button(f2, text="?", style='my.TButton', width=2, command=pos_f)
#
    posX_button.grid(column=4, row=Zeile)

    ###################################################################### Y-Position
    Zeile += 1

    posY_label = ttk.Label(f2, text="Y-Position des Eckpunkts", font=font_klein)
    posY_label.grid(column =0, row=Zeile)

    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)
    eingabe_of_z += 1

    posY_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    posY_einheit.grid(column =2, row=Zeile)
    posY_hinweis = ttk.Label(f2, text="-400 bis 400", font=font_klein)
    posY_hinweis.grid(column =3, row=Zeile)

    ###################################################################### Z-Position
    Zeile += 1
```

```
    posZ_label = ttk.Label(f2, text="Z-Position des Eckpunkts", font=font_klein)
    posZ_label.grid(column =0, row=Zeile)

    eingabe_of += [ttk.Entry(f2, width=10)]
    eingabe_of[eingabe_of_z].grid(column=1, row=Zeile)


    posZ_einheit = ttk.Label(f2, text="[mm]", font=font_klein)
    posZ_einheit.grid(column =2, row=Zeile)
    posZ_hinweis = ttk.Label(f2, text="0 bis 1000", font=font_klein)
    posZ_hinweis.grid(column =3, row=Zeile)

    ######################################################################
Standardeinstellung OF
    Zeile += 1

    standardeinstellung_OF_label = ttk.Label(f2, text="Standardeinstellung",
font=font_klein)
    standardeinstellung_OF_label.grid(column =0, row=Zeile)

    standardeinstellung_OF_t = BooleanVar()
    standardeinstellung_OF_t.set(False)
    Standardeinstellung_OF_eingabe = ttk.Checkbutton(f2,
variable=standardeinstellung_OF_t)
    Standardeinstellung_OF_eingabe.grid(column=1, row=Zeile)

    def standardeinstellung_OF_f():
        messagebox.showinfo('Standardeinstellung von Objekt und Fahrtdaten',
'Winkelanzahl: 4                                '
                                                                        '
\nGeschwindigkeit: '

'10mm/s\nObjektbreite: 200mm \nObjektlänge'
                                                                        ':
200mm\nObjekthöhe:40mm \nX-Position: '
                                                                        '555mm\nY-
Position: -100mm \nZ-Position: '
                                                                        '72mm')

    standardeinstellung_OF_button = ttk.Button(f2, text="?", style='my.TButton',
width=2, command=standardeinstellung_OF_f) #
    standardeinstellung_OF_button.grid(column=4, row=Zeile)


##############################################################################
############### Fertig-Zeile

##############################################################################
############################
    Zeile += 1

    leerzeichen2 = ttk.Label(f2, text=" ", font=font_gross)
    leerzeichen2.grid(column=0, row=Zeile)

    ###################################################################### Dialog

    def callback_eingabe(eingabe_se_e, resampling_e0, resampling_e1, eingabe_of_e,
eingabe_se_t, eingabe_pf_t,
                         eingabe_of_t,):
        if messagebox.askyesno('Verify', 'Die follgenden Werte sind nicht gültig:\n' +
eingabe_se_e[0] + eingabe_se_e[1]
                                         + eingabe_se_e[2] + eingabe_se_e[3] +
eingabe_se_e[4] + eingabe_se_e[5] +
                                         eingabe_se_e[6] +
                                         resampling_e0 + resampling_e1 +
                                         eingabe_of_e[0] + eingabe_of_e[1] +
```

```
eingabe_of_e[2] + eingabe_of_e[3] +
                                        eingabe_of_e[4] + eingabe_of_e[5] +
eingabe_of_e[6] + eingabe_of_e[7] +
                                        '\n\nSoll die Standardeinstellung für den
entsprechenden Abschnitt verwendet '
                                        'werden?'):
            kse = 0
            for i in range(len(eingabe_se_e)):
                if eingabe_se_e[i] != '':
                    kse += 1
            if kse != 0:
                eingabe_se_t = standardeinstellung_se_set()

            if resampling_e0 != '' or resampling_e1 != '':
                eingabe_pf_t = standardeinstellung_pf_set()

            kof = 0
            for i in range(len(eingabe_of_e)):
                if eingabe_of_e[i] != '':
                    kof += 1
            if kof != 0:
                eingabe_of_t = standardeinstellung_of_set()

            standardeinstellung_uebernommen = True

            return eingabe_se_t, eingabe_pf_t, eingabe_of_t,
standardeinstellung_uebernommen
        else:
            messagebox.showinfo('No', 'Bitte geben sie die Werte erneut ein')
            standardeinstellung_uebernommen = False
            return eingabe_se_t, eingabe_pf_t, eingabe_of_t,
standardeinstellung_uebernommen

    ######################################################################## Fertig
    def fertig_f():
        global werte_se, werte_pf, werte_of

        k = 0
        eingabe_se_t = []
        eingabe_pf_t = []
        eingabe_of_t = []


############################################################################## SE
        eingabe_se_e = ['Belichtungszeit\n', 'Profilfrequenz\n','Wartezeit\n', '',
'Messfeld\n', 'Messfeldbreite\n',
                        'Messfeldmitte\n']
        #                                  +0    +1    +2    +3    +4    +5    +6    +7
        max_abtastrate_zu_messfeld = [ 144,  192,  192,  192,  284,  284,  284,  552,
# 0
                                       181,  239,  239,  239,  354,  354,  354,  680,
# 8
                                       181,  239,  239,  239,  354,  354,  354,  680,
# 16
                                       181,  239,  239,  239,  354,  354,  354,  680,
# 24
                                       241,  318,  318,  318,  469,  469,  469,  892,
# 32
                                       241,  318,  318,  318,  469,  469,  469,  892,
# 40
                                       241,  318,  318,  318,  469,  469,  469,  892,
# 48
                                       241,  318,  318,  318,  469,  469,  469,  892,
# 56
                                       362,  476,  476,  476,  694,  694,  694, 1282,
# 64
                                       362,  476,  476,  476,  694,  694,  694, 1282,
```

```
# 72
                                    362,  476,  476,  476,  694,  694,  694, 1282,
# 80
                                    362,  476,  476,  476,  694,  694,  694, 1282,
# 88
                                    552,  552, 1030, 1030, 1030, 1030, 1030, 1030,
# 96
                                    680,  680, 1265, 1265, 1265, 1265, 1265, 1265,
# 104
                                    892,  892, 1612, 1612, 1612, 1612, 1612, 1612,
# 112
                                   1282, 1282, 2000, 2000, 2000, 2000, 2000, 2000]
# 120

        eingabe_of_e = ['', 'Eingabe der Geschwindigkeit ist keine Zahl\n', 'Eingabe
der Objekthöhe ist keine Zahl\n',
                        'Eingabe der Objektbreite ist keine Zahl\n', 'Eingabe der
Objektlänge ist keine Zahl\n',
                        'Eingabe der X-Position ist keine Zahl\n', 'Eingabe der Y-
Position ist keine Zahl\n',
                        'Eingabe der Z-Position ist keine Zahl\n']

        def ueberpruefen_se(k,eingabe_se_e, eingabe_se_t):
            # Belichtungszeit
########################################################################
            try:
                if eingabe_se_t[0] >= 0.01 and eingabe_se_t[0] <= 40:
                    eingabe_se_e[0] = ''
                else:
                    k += 1
                    eingabe_se_e[0] = 'Belichtungszeit\n'
            except:
                if eingabe_se_t[0] =='':
                    k += 1
                    eingabe_se_e[0] = 'Eingabe der Belichtungszeit ist keine Zahl\n'

            # Profilfrequenz und Wartezeit
########################################################################

            try:
                eingabe_se_t[1] = float(eingabe_se_t[1])
                eingabe_se_t[2] = float(eingabe_se_t[2])
                if messagebox.askyesno('Profilfrequenz und Wartezeit','Ist eine
Profilfreqeunz angegeben ist die Wartezeit redundant.\n Soll die Wartezeit ignoriert
werden?\n'):
                    messagebox.showinfo('Yes', 'Die Wartezeit wird ignoriert')
                    eingabe_se_t[2] = (1 / (eingabe_se_t[2] * 0.001)) - eingabe_se_t[0]
# int((1/(frequency*0.00001))-shuttertime)
                    if eingabe_se_t[2] < 0.01 or eingabe_se_t[2] > 40:
                        k += 1
                        messagebox.showinfo('Wartezeit', 'Aus der eingegebenen
Belichtungszeit und Profilfrequenz ergibt sich eine unzulässige Wartezeit.\n')
                        eingabe_se_e[1] = ''
                        eingabe_se_e[2] = 'Resultierende Wartezeit\n'
                else:
                    messagebox.showinfo('No', 'Die Profilfrequenz wird ignoriert')
                    eingabe_se_t[1] = 1 / ((eingabe_se_t[0] + eingabe_se_t[2]) * 0.001)
# f[Hz] = 1 / ((s_t[ms] + i_t[ms]) * 0.001)
                    if eingabe_se_t[1] > 2000:
                        k += 1
                        messagebox.showinfo('Profilfrequenz',
                                            'Aus der eingegebenen Belichtungszeit und
Wartezeit ergibt sich eine unzulässige Profilfrequenz.\n')
                        eingabe_se_e[1] = 'Resultierende Profilfrequenz\n'
            except:
                pass
```

```
            try:
                eingabe_se_t[1] = float(eingabe_se_t[1])
                eingabe_se_t[2] = str(eingabe_se_t[2])
                if eingabe_se_t[1] > 0.1 and eingabe_se_t[1] <= 2000 and
eingabe_se_t[2] == '':
                    eingabe_se_e[1] = ''
                    eingabe_se_t[2] = (1 / (eingabe_se_t[1] * 0.001)) - eingabe_se_t[0]
                else:
                    if eingabe_se_t[2] == '':
                        k += 1
                        eingabe_se_e[1] = 'Profilfrequenz\n'
            except:
                pass
            try:
                eingabe_se_t[2] = float(eingabe_se_t[2])
                eingabe_se_t[1] = str(eingabe_se_t[1])
                if eingabe_se_t[2] >= 0.01 and eingabe_se_t[2] <= 40 and
eingabe_se_t[1] == '':
                    eingabe_se_e[2] = ''
                    eingabe_se_t[1] = 1 / ((eingabe_se_t[0] + eingabe_se_t[2]) * 0.001)
                else:
                    if eingabe_se_t[1] == '':
                        k += 1
                        eingabe_se_e[2] = 'Wartezeit\n'
            except:
                pass

            try:
                eingabe_se_t[1] = str(eingabe_se_t[1])
                eingabe_se_t[2] = str(eingabe_se_t[2])
                if eingabe_se_t[1] =='' and eingabe_se_t[2] =='':
                    eingabe_se_e[1] = 'Eingabe der Profilfrequenz ist keine Zahl\n'
            except:
                pass

            #  Messfeld und Profilfreqeunz
#####################################################################

            try:
                eingabe_se_t[4] = int(eingabe_se_t[4])
                if eingabe_se_t[4] >= 0 and eingabe_se_t[4] <= 127:
                    eingabe_se_e[4] = ''
                    if float(max_abtastrate_zu_messfeld[eingabe_se_t[4]]) <
float(eingabe_se_t[1]):
                        k += 1
                        messagebox.showinfo('Messfeld und Profilfrequenz',
                                            'Die eingegebene Profilfrequenz ist zu hoch
für das ausgewählte Messfeld.\n')
                        eingabe_se_e[4] = 'Messfeld/Profilfrequenz\n'
                else:
                    k += 1
                    eingabe_se_e[4] = 'Messfeld\n'
            except:
                if eingabe_se_t[4] == '':
                    eingabe_se_e[4] = 'Eingabe des Messfelds ist keine Zahl\n'
                    k += 1
                else:
                    try:
                        eingabe_se_t[4] = float(eingabe_se_t[4])
                        eingabe_se_e[4] = 'Eingabe des Messfelds ist keine ganze
Zahl\n'
                    except:
                        eingabe_se_e[4] = 'Eingabe des Messfelds ist keine Zahl\n'

            #  Messfeldbreit
#####################################################################
            try:
```

```
                eingabe_se_t[5] = float(eingabe_se_t[5])
                if eingabe_se_t[5] >= 15 and eingabe_se_t[5] <= 145:
                    eingabe_se_e[5] = ''
                else:
                    k += 1
                    eingabe_se_e[5] = 'Messfeldbreite\n'
            except:
                if eingabe_se_t[5] == '':
                    eingabe_se_e[5] = 'Eingabe der Messfeldbreite ist keine Zahl\n'
                    k += 1
            #  Messfeldmitte
########################################################################
            try:
                eingabe_se_t[6] = float(eingabe_se_t[6])
                if eingabe_se_t[6] >= 125 and eingabe_se_t[6] <= 390:
                    eingabe_se_e[6] = ''
                else:
                    k += 1
                    eingabe_se_e[6] = 'Messfeldmitte\n'
            except:
                if eingabe_se_t[6] == '':
                    eingabe_se_e[6] = 'Eingabe der Messfeldmitte ist keine Zahl\n'
                    k += 1
            return k, eingabe_se_e, eingabe_se_t

        if bool(standardeinstellung_SE_t.get()) == True:
            eingabe_se_t = standardeinstellung_se_set()
            eingabe_se_e = ['']*len(eingabe_se_e)

        else:
            poderw = 0
            for i in range(len(eingabe_se)):
                try:
                    float(eingabe_se[i].get())
                    eingabe_se_t += [float(eingabe_se[i].get())]
                except:
                    eingabe_se_t += [str(eingabe_se[i].get())]
                    if (eingabe_se_t[i] == '') and (i == 1 or i == 2) and poderw == 0:
                        eingabe_se_e[i] = ''
                        poderw += 1
                    else:
                        k += 1

            ############################################################## Überprüfung
SE
            pruef_se = ueberpruefen_se(k, eingabe_se_e, eingabe_se_t)
            k += pruef_se[0]
            eingabe_se_e = pruef_se[1]
            eingabe_se_t = pruef_se[2]


############################################################################## PF
        resampling_e0 = ''
        resampling_e1 = ''

        if bool(standardeinstellung_PF_t.get()) == True:
            eingabe_pf_t = standardeinstellung_pf_set()
        else:
            for i in range(len(eingabe_pf)):
                eingabe_pf_t += [int(eingabe_pf[i].get())]

            if eingabe_pf_t[0] == 0:
                if eingabe_pf_t[1] > 0 or eingabe_pf_t[2] > 0:
                    k += 1
                    resampling_e0 = '\nResampling: Wenn Resampling auf 0 gesetzt ist,
müssen auch Median und Average auf 0 gesetzt werden.\n'
            else:
```

```
                if eingabe_pf_t[1] > 0 and eingabe_pf_t[2] > 0:
                    k += 1
                    resampling_e1 = '\nMedian oder Average: Es können nicht beide
Gleichzeitig >0 eingestellt werden.\n'


############################################################################## OF
        def ueberpruefen_of(k, eingabe_of_t):
            # Geschwindigkeit
#####################################################################
            try:
                if eingabe_of_t[1] >= 0.1 and eingabe_of_t[1] <= max_geschwindigkeit:
                    eingabe_of_e[1] = ''
                else:
                    k += 1
                    eingabe_of_e[1] = 'Geschwindigleit\n'
            except:
                pass

            # Objektbreite
#####################################################################
            try:
                if eingabe_of_t[2] >= 0 and eingabe_of_t[2] <= 600:
                    eingabe_of_e[2] = ''
                else:
                    k += 1
                    eingabe_of_e[2] = 'Objektbreite\n'
            except:
                pass

            # Objektlänge
#####################################################################
            try:
                if eingabe_of_t[3] >= 0 and eingabe_of_t[3] <= 800:
                    eingabe_of_e[3] = ''
                else:
                    k += 1
                    eingabe_of_e[3] = 'Objektlänge\n'
            except:
                pass

            # Objekthöhe
#####################################################################
            try:
                if eingabe_of_t[4] >= 0 and eingabe_of_t[4] <= 1000:
                    eingabe_of_e[4] = ''
                else:
                    k += 1
                    eingabe_of_e[4] = 'Objekthöhe\n'
            except:
                pass

            # X-Position
#####################################################################
            try:
                if eingabe_of_t[5] >= 400 and eingabe_of_t[5] <= 1000:
                    eingabe_of_e[5] = ''
                else:
                    k += 1
                    eingabe_of_e[5] = 'X-Posizion des Eckpunkts\n'
            except:
                pass

            # Y-Position
#####################################################################
            try:
                if eingabe_of_t[6] >= -400 and eingabe_of_t[6] <= 400:
```

```
                    eingabe_of_e[6] = ''
                else:
                    k += 1
                    eingabe_of_e[6] = 'Y-Posizion des Eckpunkts\n'
            except:
                pass

            # Z-Position
########################################################################
            try:
                if eingabe_of_t[7] >= 0 and eingabe_of_t[7] <= 1000:
                    eingabe_of_e[7] = ''
                else:
                    k += 1
                    eingabe_of_e[7] = 'Z-Posizion des Eckpunkts\n'
            except:
                pass

            return k, eingabe_of_e

        if bool(standardeinstellung_OF_t.get()) == True:
            eingabe_of_t = standardeinstellung_of_set()
            eingabe_of_e = ['']*len(eingabe_of_e)
        else:
            for i in range(len(eingabe_of)):
                try:
                    eingabe_of_t += [float(eingabe_of[i].get())]
                    eingabe_of_e[i] = ''
                except:
                    k += 1
            ########################################################################## Überprüfung
OF
            pruef_of = ueberpruefen_of(k, eingabe_of_t)
            k += pruef_of[0]
            eingabe_of_e = pruef_of[1]

        if k == 0:
             werte_se = eingabe_se_t
            werte_pf = eingabe_pf_t
            werte_of = eingabe_of_t
            root.destroy()

        else:
            eingabe_se_t, eingabe_pf_t, eingabe_of_t, standardeinstellung_uebernommen =
callback_eingabe(eingabe_se_e, resampling_e0, resampling_e1, eingabe_of_e,
eingabe_se_t, eingabe_pf_t, eingabe_of_t)
            werte_se = eingabe_se_t
            werte_pf = eingabe_pf_t
            werte_of = eingabe_of_t
            if standardeinstellung_uebernommen == True:
                root.destroy()


    Zeile += 1

    fertig_button = ttk.Button(f2, text="Fertig", style='my.TButton', width=6,
command=fertig_f)
    fertig_button.grid(column=0, row=Zeile)

frame2()

root.mainloop()

shuttertime = int(werte_se[0]*100)  # zwischen 1 und 4095; 100=1ms von 0.01ms bis 40ms
frequency = werte_se[1]  # f[Hz]=1/((s_t[ms]+i_t[ms])*0.001)
idletime = int(round(werte_se[2]*100))  # int((1/(frequency*0.00001))-shuttertime)
#zwischen 1 und 4095
```

```
resolution = werte_se[3]  # 160,320,640 oder 1280
scanner_type = ct.c_int(0)
Measuringfield = werte_se[4]  # zw 0 und 127
Scannerbreite = werte_se[5]  # = Messfeldbreite
Scannerhoehe = werte_se[6]  # = Messfeldmitte

Resampling = werte_pf[0] # set Resampling 0-7
Median = werte_pf[1]  # set Median 0-3
Average = werte_pf[2]  # set Average 0-3

Winkelanzahl = werte_of[0]
Geschwindigkeit = werte_of[1]/1000  # m/s
v = werte_of[1]  # geschwindigkeit in mm/s
Breite = werte_of[2]  # X-Richtugn
Laenge = werte_of[3]  # Y-Richtung
Hoehe = werte_of[4]+werte_of[7]

PEx = [werte_of[5]]  # PE: Eckpunkt zum Roboter(y) hin links(x) unten(z) |
Robotersockel geht in x-Richtung bis 105mm
PEy = [werte_of[6]]
PEz = [werte_of[7]]
PEx += [PEx[0]]
PEy += [PEy[0]+Laenge]
PEx += [PEx[0]+Breite]
PEy += [PEy[0]+Laenge]
PEx += [PEx[0]+Breite]
PEy += [PEy[0]]
PEx += [PEx[0]]
PEy += [PEy[0]]

wl = 0
for index in range(len(Winkel)):
    if index <= Winkelanzahl-1:
        wl+=1
    else:
        Winkel[index] = 360

j = int(round(Breite/Scannerbreite+0.5))
k = int(round(Laenge/Scannerbreite+0.5))
Fahrten = range((j+k)*wl)
Teil = 1

Fin = 0
Ori = 0

xml = '<Sensor>' \
         '<Read>' \
            '<Hoehe>' + str(Hoehe) + '</Hoehe>' \
            '<Breite>' + str(Breite) + '</Breite>' \
            '<Laenge>' + str(Laenge) + '</Laenge>' \
            '<Scannerbreite>' + str(Scannerbreite) + '</Scannerbreite>' \
            '<Scannerhoehe>' + str(Scannerhoehe) + '</Scannerhoehe>' \
            '<Winkel>' + str(Winkel[0]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[1]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[2]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[3]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[4]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[5]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[6]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[7]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[8]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[9]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[10]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[11]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[12]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[13]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[14]) + '</Winkel>' \
```

```
            '<Winkel>' + str(Winkel[15]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[16]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[17]) + '</Winkel>' \
            '<Winkel>' + str(Winkel[18]) + '</Winkel>' \
            '<Geschwindigkeit>' + str(Geschwindigkeit) + '</Geschwindigkeit>' \
            '<xyzabc X="' + str(PEx[0]) + '" Y="' + str(PEy[0]) + '" Z="0" A="0" B="0"
C="0"/>' \
        '</Read>' \
    '</Sensor>'

print(xml)
TCP_IP = socket.gethostname()  # Server IP
TCP_PORT = 59152
BUFFER_SIZE = 1024  # max 65534  # Normal 1024

# Init-Profilpuffer und Zeitstempel-Info
profile_buffer = (ct.c_ubyte*(resolution*64))()  # (ct.c_ubyte*(resolution*64))()
timestamp = (ct.c_ubyte*16)()
available_resolutions = (ct.c_uint*4)()
available_interfaces = (ct.c_uint*6)()
lost_profiles = ct.c_int()
shutter_opened = ct.c_double(0.0)
shutter_closed = ct.c_double(0.0)
profile_count = ct.c_uint(0)

# Deklaration des Messdatenarray
x = (ct.c_double * resolution)()
z = (ct.c_double * resolution)()
intensities = (ct.c_ushort * resolution)()

# Null pointer wenn Daten nicht erforderlich sind
null_ptr_short = ct.POINTER(ct.c_ushort)()
null_ptr_int = ct.POINTER(ct.c_uint)()

# Ordner und Dateien erstellen
vi = int(round(v))
fri = int(round(frequency))
jetzt = datetime.datetime.now().strftime('%Y-%m-%d %H_%M')
pfad = 'C:/Users/student.MECHLAB103/Desktop/ASC Scanns/Scann '+str(jetzt)+'
V'+str(vi)+'S'+str(shuttertime)+'Hz'+str(fri)+'F'+str(Resampling)+str(Median)+str(Avera
ge)

if os.path.isdir(pfad) == False:
    os.mkdir(pfad)
else:
    print('Ordner ist vorhanden und wird überschrieben!')

block = 0  # Block an Datein mit dem selben Winkel
for i in Fahrten:
    if i >= (j+k)*(block+1):
        block+=1
    dateihandler = open(pfad+'/Scann '+str(jetzt)+' Teil '+str(i+1)+' Winkel
'+str(round(Winkel[block]))+'.asc', mode='w')
    dateihandler.flush()
    dateihandler.close()

# schreiben thread
def schreiben(wx, wy, wz, wintensities, wwinkel):
    dateihandler = open(pfad+'/Scann '+str(jetzt)+' Teil '+str(Teil)+' Winkel
'+str(round(wwinkel))+'.asc', mode='a')
    for i in range(len(x)):
        if (wz[i] != 0.0):
            if Ori ** 2 == 1:
                dateihandler.write(str(wx[i]) + ' ' + str(wy) + ' ' + str(wz[i]) + ' '
+ str(wintensities[i]) + '\n')
            elif Ori ** 2 == 4:
                dateihandler.write(str(wy) + ' ' + str(wx[i]) + ' ' + str(wz[i]) + ' '
```

```
+ str(wintensities[i]) + '\n')

# Instanz erstellen
hLLT = llt.create_llt_device(llt.TInterfaceType.INTF_TYPE_ETHERNET)

def Scannereinstellung():
    # Verfügbare Schnittstellen abrufen
    ret = llt.get_device_interfaces_fast(hLLT, available_interfaces,
len(available_interfaces))
    if ret < 1:
        raise ValueError("Error getting interfaces : " + str(ret))

    # Set IP
    ret = llt.set_device_interface(hLLT, available_interfaces[0], 0)
    if ret < 1:
        raise ValueError("Error setting device interface: " + str(ret))

    # Verbinden
    ret = llt.connect(hLLT)
    if ret < 1:
        raise ConnectionError("Error connect: " + str(ret))

    # Verfügbare Auflösungen abrufen
    ret = llt.get_resolutions(hLLT, available_resolutions, len(available_resolutions))
    if ret < 1:
        raise ValueError("Error getting resolutions : " + str(ret))

    if resolution not in available_resolutions:
        raise AttributeError("Wrong resolution")

    # Scanner Typ
    ret = llt.get_llt_type(hLLT, ct.byref(scanner_type))
    if ret < 1:
        raise ValueError("Error scanner type: " + str(ret))

    ret = llt.set_resolution(hLLT, resolution)
    if ret < 1:
        raise ValueError("Error setting resolution: " + str(ret))

    # Profile config
    ret = llt.set_profile_config(hLLT, llt.TProfileConfig.PROFILE)
    if ret < 1:
        raise ValueError("Error setting profile config: " + str(ret))

    # Set Messfeld
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_MEASURINGFIELD, Measuringfield)
    if ret < 1:
        raise ValueError("Error setting profile config: " + str(ret))

    # Set shutter time
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_SHUTTERTIME, shuttertime)
    if ret < 1:
        raise ValueError("Error setting shutter time: " + str(ret))

    # Set idle time
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_IDLETIME, idletime)
    if ret < 1:
        raise ValueError("Error idle time: " + str(ret))

    # Set Profilfilter
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_PROFILE_FILTER, Resampling,
Median, Average) # (..., Resampling 0-7, Median 0-3, Average 0-3)
    if ret < 1:
        raise ValueError("Error setting profile filter: " + str(ret))

    # Laser off
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_LASERPOWER, 0)  # 0: Off; 1:
```

```
reduced; 2: standard
    if ret < 1:
        raise ValueError("Error setting Laser Power off: " + str(ret))

Scannereinstellung()

def scanning(x_r,y_r,z_r,W):
    start_time = time.time()

    # Laser on
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_LASERPOWER, 2)  # 0: Off; 1:
reduced; 2: standard
    if ret < 1:
        raise ValueError("Error setting laser power on: " + str(ret))

    # Starte Transfer
    ret = llt.transfer_profiles(hLLT, llt.TTransferProfileType.NORMAL_TRANSFER, 1)
    if ret < 1:
        raise ValueError("Error starting transfer profiles: " + str(ret))

    global Teil

    # Warm-up time
    time.sleep(Warmuptime)

    #       Schleife starten
    while Ori != 0:
        ret = llt.get_actual_profile(hLLT, profile_buffer, len(profile_buffer),
llt.TProfileConfig.PROFILE,
                                     ct.byref(lost_profiles))
        if ret != len(profile_buffer):
            time.sleep(1/frequency)
            ret = llt.get_actual_profile(hLLT, profile_buffer, len(profile_buffer),
llt.TProfileConfig.PROFILE,
                                         ct.byref(lost_profiles))

        ret = llt.convert_profile_2_values(hLLT, profile_buffer, resolution,
llt.TProfileConfig.PROFILE, scanner_type, 0, 1,
                                           null_ptr_short, intensities, null_ptr_short, x,
z, null_ptr_int, null_ptr_int)
        if ret & llt.CONVERT_X is 0 or ret & llt.CONVERT_Z is 0 or ret &
llt.CONVERT_MAXIMUM is 0:
            raise ValueError("Error converting data: " + str(ret))

        nx = [0]*resolution
        nz = [0]*resolution
        ny = 0

        if Ori ** 2 == 1:
            for i in range(len(x)):
                if (float(z[i]) != 0.0):
                    nx[i] = round(x_r - float(z[i]) * (math.sin(math.radians(W)))
                                  + float(x[i]) * (math.cos(math.radians(W))),
Nachkommastellen_in_der_asc)  # hängt von der Pose des Roboters ab:Scanner zeigt in -y
-> x+x_r; Scanner zeigt in +y -> -x+x_r
                    nz[i] = round(z_r - float(z[i]) * (math.cos(math.radians(W)))
                                  - float(x[i]) * (math.sin(math.radians(W))),
Nachkommastellen_in_der_asc)
            ny = round(y_r + Ori * v * (time.time() - start_time),
Nachkommastellen_in_der_asc)
        elif Ori ** 2 == 4:
            for i in range(len(x)):
                if (float(z[i]) != 0.0):
                    nx[i] = round(y_r - float(z[i]) * (math.sin(math.radians(W)))
                                  + float(x[i]) * (math.cos(math.radians(W))),
Nachkommastellen_in_der_asc)  # hängt von der Pose des Roboters ab:Scanner zeigt in -y
-> x+x_r; Scanner zeigt in +y -> -x+x_r
```

```
                    nz[i] = round(z_r - float(z[i]) * (math.cos(math.radians(W)))
                                      - float(x[i]) * (math.sin(math.radians(W))),
Nachkommastellen_in_der_asc)
            ny = round(x_r + 0.5 * Ori * v * (time.time() - start_time),
Nachkommastellen_in_der_asc)
        elif Ori == 0:
            continue
        else:
            print("Error: Orientierung = "+str(Ori)+" beim Scannvorgang")

        try:
            _thread.start_new_thread(schreiben, (nx, ny, nz, intensities, W))
        except:
            print("Error: unable to start writing thread")

    # Stop transmission
    ret = llt.transfer_profiles(hLLT, llt.TTransferProfileType.NORMAL_TRANSFER, 0)
    if ret < 1:
        raise ValueError("Error stopping transfer profiles: " + str(ret))

    # Laser off
    ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_LASERPOWER, 0)  # 0: Off; 1:
reduced; 2: standard
    if ret < 1:
        raise ValueError("Error setting Laser Power off: " + str(ret))

    dateihandler.close()
    Teil += 1

################################################
#################### Server ####################
################################################

while Fin == 0:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.bind((TCP_IP, TCP_PORT))
    s.listen(1)  # Anzahl möglicher Verbindungen
    print('listen to '+str(TCP_IP))
    empfangen = False

    conn, addr = s.accept()
    print ('Connection address:'+ str(addr))
    while True:
        if empfangen == False:
            conn.send(bytearray(xml,'utf-8'))
        data = conn.recv(BUFFER_SIZE)
        if not data: break
        xmlroot = ET.fromstring(data)
        try:
            x_r = float(xmlroot.find('Data/ActPos/X').text)
            y_r = float(xmlroot.find('Data/ActPos/Y').text)
            z_r = float(xmlroot.find('Data/ActPos/Z').text)
            a_r = float(xmlroot.find('Data/ActPos/A').text)
            b_r = float(xmlroot.find('Data/ActPos/B').text)
            c_r = float(xmlroot.find('Data/ActPos/C').text)
            W = float(xmlroot.find('Winkel').text)
        except:
            print('Keine Positionsdaten empfangen.')
        try:
            Ori = int(xmlroot.find('Orientierung').text)
            empfangen = True
        except:
            pass

        if Ori != 0:
            print(Ori,W)
            try:
```

```
                _thread.start_new_thread(scanning, (x_r, y_r, z_r, W))  # scanning-
thread starten
            except:
                print("Error: unable to start scanning thread")
        else:
            try:
                Fin = int(xmlroot.find('Fin').text)
                if Fin != 0:
                    time.sleep(0.1)
                    break
            except:
                pass
        try:
            print('x = '+str(x_r)+' y = '+str(y_r)+' z = '+str(z_r)+' Orientierung =
'+str(Ori))
        except:
            pass

    conn.close()

# Laser off
ret = llt.set_feature(hLLT, llt.FEATURE_FUNCTION_LASERPOWER, 0)  # 0: Off; 1: reduced;
2: standard
if ret < 1:
    raise ValueError("Error setting Laser Power off: " + str(ret))

# Disconnect
ret = llt.disconnect(hLLT)
if ret < 1:
    raise ConnectionAbortedError("Error while disconnect: " + str(ret))

# Delete
ret = llt.del_device(hLLT)
if ret < 1:
    raise ConnectionAbortedError("Error while delete: " + str(ret))

dateihandler.close()
```

5 Dokumente

5.1 Schriftverkehr mit Mirco-Epsilon Support

Hallo Herr Titgemeyer,

kein Problem.

Aus Sensorsicht sind beim 29xx die 200 Hz nur mit etwas eingeschränktem Messfeld zu erreichen.

Am besten mal in den Configuration Tools das Geeignete aussuchen. Siehe dazu die Sensordoku (QuickReference).

Die Doku ist u.A. über die Configuration Tools zugänglich (Toolbar „?" -> Dokumentation)

Bsp. für 284 Hz:

```
llt.set_feature(hLLT, llt.FEATURE_FUNCTION_MEASURINGFIELD, 5)
```

Ansonsten gibt es noch einige Auffälligkeiten im Code die zu Geschwindigkeitseinbußen führen können. An dieser Stelle kann ich nun aber wirklich nur noch Hinweise geben:

- Das time.sleep alleine schränkt die Auslesefrequenz schon auf 100 Hz ein. Das sleep brauchen Sie nur einmalig nach „transfer_profiles"

- Die Daten werden jedes Mal in die Datei geschrieben, was prinzipiell erstmal dauert. Bei einer SSD könnte es sogar funktionieren, aber es wäre besser man sammelt die Daten im RAM auf und schreibt sie anschließend erst in eine Datei oder man führt einen asynchronen Speicherthread ein.

- Falls nach entfernen des sleeps die Funktion get_actual_profile den Fehler -104 wirft, heißt das, dass einfach noch keine neuen Daten vom Sensor da sind. Ist also weniger ein Fehler und mehr ein Hinweis. Es empfiehlt sich daher zwischen zwei Aufrufen dieser Funktion geeignet lang zu warten (~1/f). Für höhere Geschwindigkeiten (>1000 Hz) sollte man sich übrigens das Callbackbeispiel als Grundgerüst nehmen.

- Um einen steten y-Abstand zu haben, empfehlen wir einen Encoder – somit können auch Varianzen in der Bewegungsgeschwindigkeit von Sensor/Target ausgeglichen werden.

Freundliche Grüße / Kind regards

Daniel Rauch, M. Sc.
Applikation & Support 2D/3D Optische Messtechnik
Application & Support 2D/3D Optical Metrology

MICRO-EPSILON MESSTECHNIK GmbH & Co. KG
Königbacher Str. 15 | 94496 Ortenburg | Germany
Tel.: +49 8542 168-597 | Fax: +49 8542 168-90
Daniel.Rauch@micro-epsilon.de
www.micro-epsilon.de

Von: Titgemeyer, Constantin-Julian [mailto:Constantin-Julian.Titgemeyer@haw-hamburg.de]
Gesendet: Montag, 11. März 2019 13:12
An: Rauch, Daniel
Cc: Hanisch, Martin
Betreff: AW: Windows SDK mit Matlab verwenden

Sehr geehrter Herr Rauch,

nachdem mir die Ansteuerung auch mit der zusätzlichen Hilfestellung nicht gelungen ist, bin ich von Matlab auf Python gewechselt. Wie erwartet funktioniert hier das Ansteuern problemlos.

Ich möchte mich an dieser Stelle nochmals für die bisherige Hilfe bedanken.
Während des Schreibens eines Programms zum Erstellen einer .asc Datei mit Ihrem Beispielprogramm „full_profiles_poll.py" als Grundgerüst, bin ich jedoch auf ein anderes Problem gestoßen. Die Wiederholrate der auf die Datei geschriebenen Profildaten ist wesentlich niedriger, als die eingestellte Abtastrate. Eine Vermutung meinerseits ist das der „profile_buffer" bei {Zeile 116 im angehängten scanTest2.py}
„ret = llt.convert_profile_2_values(hLLT, profile_buffer,…, x, z,…)" nur das letzte im Puffer befindliche Profil an x und z übergibt.
Durch die unterschiedlichen Wiederholraten sind mir die Profilabstände leider nicht ersichtlich. Das bereitet mir insofern Schwierigkeiten, da ich im späteren Verlauf mit der gefahrenen Geschwindigkeit den y-Wert berechnen möchte.
Über Hinweise oder Anmerkungen würde ich mich sehr freuen!

Mit freundlichen Grüßen

Constantin Titgemeyer

5.2 GOM Werkskalibrierschein CP40/200/100819

Werkskalibrierschein

Proprietary Calibration Certificate

GOM GmbH
Schmitzstraße 2
38122 Braunschweig
Germany

Kalibrierschein-Nr. *Calibration Certificate no.*	*CP40/200/100819* 2017-07-25

Gegenstand *Object*	Kalibrierplatte *Calibration panel*
Hersteller *Manufacturer*	GOM - Gesellschaft für Optische Messtechnik mbH
Typ *Type*	CP40/200
Serien-Nr. *Serial no.*	CP40/200/100819
Material *Material*	Glas *Glass*
Ausdehnungskoeffizient *Coefficient of thermal expansion*	$3.25*10^{-6}$ K^{-1}
Anzahl der Seiten *Number of pages*	4
Datum der Kalibrierung *Date of calibration*	25.07.2017

Datum der Ausstellung *Date of issue*	25.07.2017
Prüfer *Inspector*	Dennis Schridde

1) Kalibriergegenstand / Calibration object

Der Kalibriergegenstand ist eine Kalibrierplatte vom Typ CP40/200 (siehe Abbildung 1). Sie trägt die Seriennummer CP40/200/100819. Die Kalibrierplatte ist aus dem Material Glas gefertigt. Auf der Kalibrierplatte sind mehrere optische Kreismarken in einer definierten, gitterförmigen Struktur angeordnet. Im Zentrum der Kalibrierplatte besitzen mehrere Kreismarken einen größeren Durchmesser als die übrigen Kreismarken. Sie definieren den Koordinatenursprung des Plattenkoordinatensystems. Zudem ermöglichen sie eine automatische Erkennung des Plattentyps durch die GOM-Software.

The calibration object is a calibration panel of the type CP40/200 (see figure 1). It has the serial number CP40/200/100819. The calibration panel is made out of the material Glass. Several optical circular markers are arranged in a defined, grid shaped structure on the calibration panel. In the center of the calibration panel, several circular markers have a larger diameter than the other circular markers. They define the coordinate origin of the panel coordinate system. They also allow an automatic detection of the panel type through the GOM software.

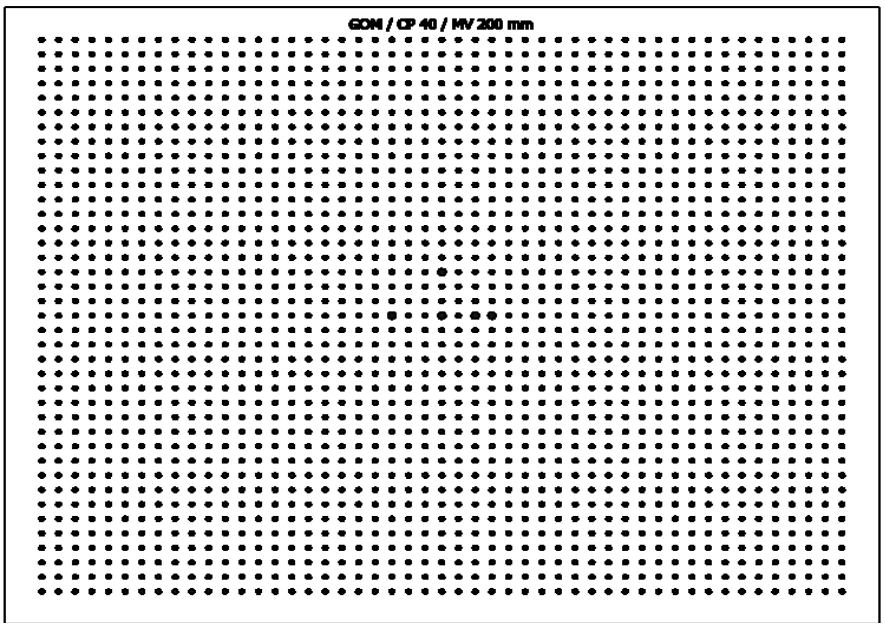


Abbildung 1: Kalibrierplatte vom Typ CP40/200
Figure 1: Calibration panel of the type CP40/200

Die Gitterstruktur der Kalibrierplatte wird durch ein fotolithografisches Verfahren erzeugt. Das Verfahren zeichnet sich durch sehr hohe Fertigungsgenauigkeiten bezüglich der Form und des Maßstabs aus. Gemäß Herstellerangaben werden folgende Genauigkeiten erzielt:

The grid structure of the calibration panel is created via a photolithographic process. The process excels by a very high production accuracy with respect to the shape and the scale. According to manufacturer's data, the following accuracies are achieved:

Eigenschaft *Property*	Fertigungsgenauigkeit *Production accuracy*
Länge *Length*	0.5 µm + 25 * 10^{-6} * L
Orthogonalität *Perpendicularity*	< 3 µrad
Rundheit *Roundness*	0.5 µm (Ø < 20 mm) 1.0 µm (Ø ≥ 20 mm)

Die zu bestimmenden Werte sind die 3D-Koordinaten der Mittelpunkte aller Kreismarken auf der Kalibrierplatte. Zusätzlich werden zwei Prüfstrecken zwischen definierten Kreismarken aus den zugehörigen 3D-Koordinaten berechnet (siehe Abbildung 2).

The measuring values to be determined are the 3D coordinates of the center points of all circular markers on the calibration panel. From the corresponding 3D coordinates, two test distances are additionally computed between defined circular markers (see figure 2).

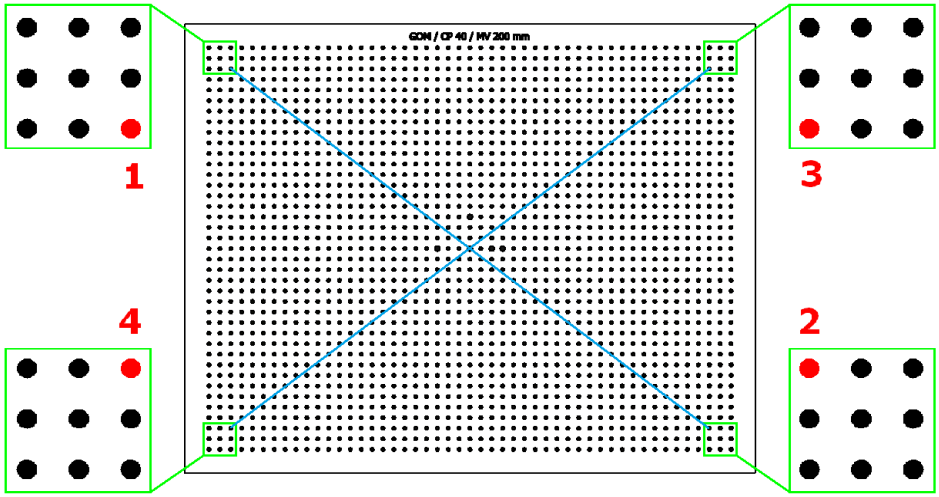


Abbildung 2: Prüfstrecken der Kalibrierplatte
Figure 2: Test distances of the calibration panel

2) Kalibrierverfahren / Calibration procedure

Die Kalibrierplatte wird nicht durch ein Messverfahren gemessen. Stattdessen werden die Sollmaße aus der Konstruktionszeichnung angehalten.

The calibration panel is not measured by a measuring method. Instead, the nominal dimensions from the drawing are used.

3) Messergebnisse / Measuring results

Die 3D-Koordinaten der Kalibrierplatte werden nicht auf dem Werkskalibrierschein angegeben, sondern in Form einer elektronischen Datei im XML-Format gespeichert. Die Datei trägt den Namen CP40-200-100819s.calobj. Sie wird auf einem separaten Speichermedium ausgeliefert.

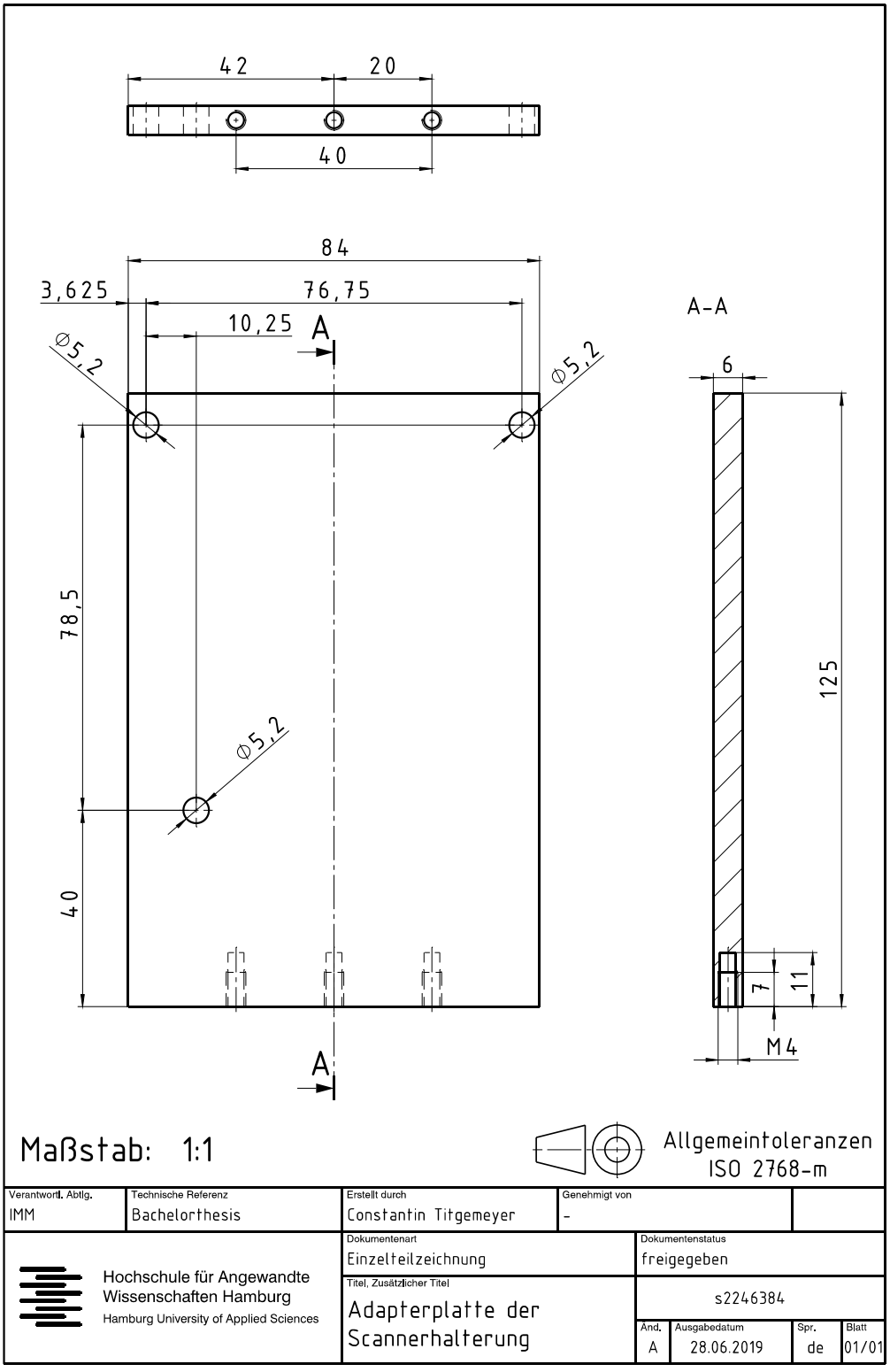
The 3D coordinates of the calibration panel are not listed on the calibration certificate, but saved as an electronic file in XML format. The file is named CP40-200-100819s.calobj*. It is delivered on a separate storage medium.*

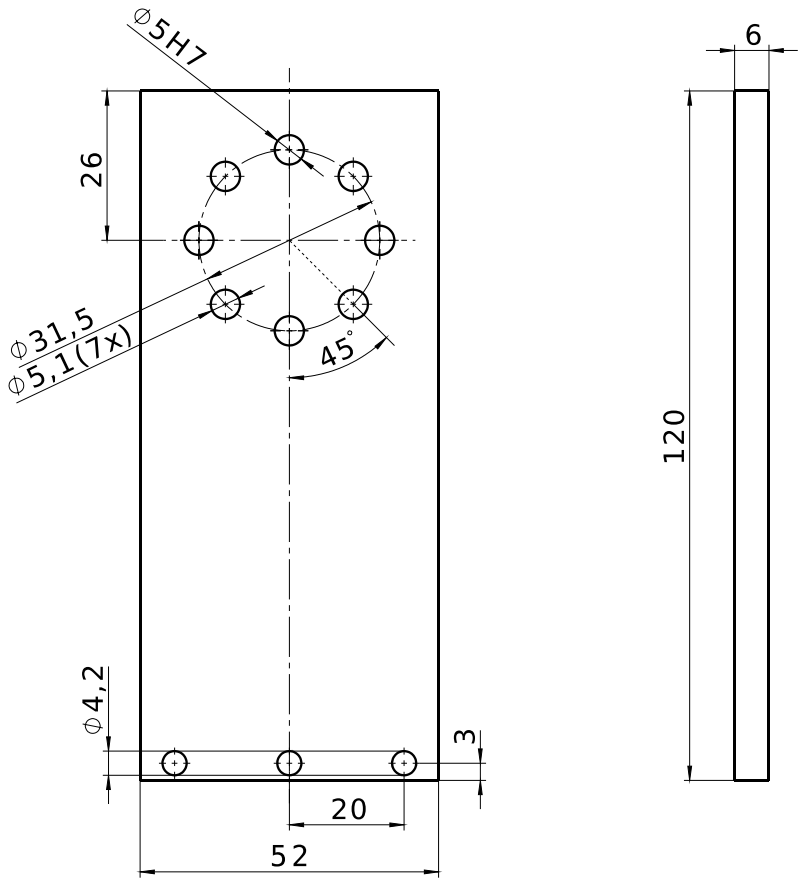
Die aus den zugehörigen 3D-Koordinaten berechneten Längen der Prüfstrecken betragen:

The lengths of the test distances computed from the corresponding 3D coordinates are:

Prüfling *Testing feature*	Messwert in mm *Measured value in mm*	Messwert in inch *Measured value in inch*
Distanz 1-2 *Distance 1-2*	250.226	9.8514
Distanz 3-4 *Distance 3-4*	250.226	9.8514

5.3 Zeichnungen der Halterung





Maßstab: 1:1

Allgemeintoleranzen
ISO 2768-m

Verantwortl. Abtlg	Technische Referenz	Erstellt durch	Genehmigt von	
IMM	Bachelorthesis	Constantin Titgemeyer	-	

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Dokumentenart	Dokumentenstatus
Einzelteilzeichnung	freigegeben

Titel, Zusätzlicher Titel
Grundplatte der Scannerhalterung

s2246384

And.	Ausgabedatum	Spr.	Blatt
A	28.06.2019	de	01/01