



**CZECH TECHNICAL  
UNIVERSITY  
IN PRAGUE**

**F3**

**Faculty of Electrical Engineering  
Department of Control Engineering**

**Bachelor's Thesis**

# **Automation of precision measurement process using laser tracker and industrial robot**

**Václav Kubáček**

**May 2022**

**Supervisor: Ing. Tomáš Jochman**



## Acknowledgement / Declaration

I would like to thank my supervisor Ing. Tomáš Jochman, for his guidance in both solving the thesis goals and composing the thesis. I would also like to thank Ing. Pavel Burget, Ph.D., the director of Testbed for Industry 4.0. Who made it possible for me to conduct my bachelor's project on the robotic workstation at Testbed.

I declare that I have worked independently on the submitted thesis and that I have listed all the sources of information used in accordance with the methodological guideline on the observance of ethical principles in the preparation of university theses.

In Prague on May 21, 2022

.....

## Abstrakt / Abstract

Tato bakalářská práce se zabývá automatizací procesu přesného měření s využitím laser trackeru a průmyslového robotu. Úkolem je navrhnout způsob komunikace mezi kontrolérem robotu, kontrolérem laser trackeru a aplikačním softwarem. Další částí práce je navrhnout a testovat proces automatické kalibrace pracovního prostoru robotu a také proces přesného měření výrobku pomocí dotykové sondy na sledovacím zařízení. Na závěr dojde k porovnání výsledků vlastních algoritmů s výsledky získaných ze softwaru třetích stran.

**Klíčová slova:** laser tracker; kalibrace; průmyslový robot, dotyková sonda, přesné měření;

This bachelor thesis deals with the automation of a precision measurement process using a laser tracker and an industrial robot. The task is to design a communication method between the robot controller, the laser tracker controller, and the application software. Another part of the work is to design and test the process of automatic calibration of the robot workspace and accurate product measurement using a touch probe on the tracker. Finally, the results of the developed algorithms will be compared with the results obtained from third-party software.

**Keywords:** laser tracker; calibration; industrial robot, touch probe, precision measurement;



# Contents /

<b>1 Introduction</b>	<b>1</b>	
1.1 Goals . . . . .	1	
<b>2 Main components of the robotic workstation</b>	<b>3</b>	
2.1 Hardware . . . . .	3	
2.1.1 Industrial robot and positioner . . . . .	3	
2.1.2 Laser tracker and accessories . . . . .	4	
2.2 Software . . . . .	5	
2.2.1 Robot programming . . . . .	5	
2.2.2 Metrology software . . . . .	6	
<b>3 Communication with the laser tracker controller</b>	<b>7</b>	
3.1 Connecting the laser tracker . . . . .	7	
3.1.1 EtherCAT . . . . .	8	
3.1.2 Real Time Feature Pack EtherCAT (RTFP-EC) . . . . .	8	
3.1.3 Connection of the Automation Robot Cable . . . . .	9	
3.2 Communication protocol . . . . .	10	
3.2.1 Input frame from Slave to Master . . . . .	10	
3.2.2 Output frame from Master to Slave . . . . .	11	
3.2.3 Algorithm for the laser tracker automated control . . . . .	13	
3.3 Conversion of incoming data . . . . .	15	
3.3.1 IEEE Standard for Floating-Point numbers . . . . .	15	
3.3.2 Implementation of the Real numbers converter . . . . .	15	
3.3.3 Discussion on precision loss . . . . .	16	
3.3.4 Coordinate system conversion for positions . . . . .	17	
3.3.5 Coordinate system conversion for rotations . . . . .	17	
3.3.6 Euler angles conversion limitations . . . . .	18	
3.4 Other configuration option . . . . .	19	
<b>4 Automatic robot workspace calibration</b>	<b>20</b>	
4.1 Description of the workflow . . . . .	20	
4.2 Transformation between the positioner and the laser tracker $\mathbf{T}_l^d$ . . . . .	21	
4.2.1 Features measurement . . . . .	21	
4.2.2 Plane, Axis and Center point alignment . . . . .	22	
4.2.3 Robot programming . . . . .	23	
4.3 Transformation between the laser tracker and robot's root $\mathbf{T}_r^l$ . . . . .	24	
4.3.1 Transformation between the laser tracker and the T-Mac . . . . .	25	
4.3.2 Transformation between the robot and the T-Mac . . . . .	26	
4.3.3 Joining $\mathbf{T}_t^l$ and $\mathbf{T}_t^r$ together . . . . .	27	
4.3.4 Euler angles . . . . .	28	
<b>5 Inspection of the manufactured part</b>	<b>30</b>	
5.1 Measurement of the printed part using Polyworks Inspector software . . . . .	30	
5.1.1 Printed Part . . . . .	30	
5.1.2 Polyworks Inspector program . . . . .	31	
5.1.3 Workflow for automated inspection of the printed part . . . . .	32	
5.1.4 Robot programming . . . . .	33	
5.1.5 Evaluation of the inspection . . . . .	34	
5.2 Measurement of the printed part without metrology software . . . . .	35	
5.2.1 Acquisition of measured data . . . . .	35	
5.2.2 Inspection of the thickness of the lower ring . . . . .	36	
5.2.3 Cylinder inspection . . . . .	38	
<b>6 Comparison of the methods presented in Chapters 5 and 6 with metrology software</b>	<b>40</b>	

6.1 Comparison of calibration methods . . . . .	40
6.1.1 Workflow description in Robodyn . . . . .	40
6.1.2 Comparison of both transformation matrices . .	41
6.1.3 Calibration result . . . . .	43
6.2 Comparison of printed part inspection methods . . . . .	43
6.2.1 Layer thickness inspection .	44
6.2.2 Cylinder inspection . . . .	46
<b>7 Conclusion</b>	<b>48</b>
<b>References</b>	<b>50</b>
<b>A Thesis assignment</b>	<b>53</b>
<b>B Photographs of devices</b>	<b>55</b>

## Tables / Figures

<b>6.1</b>	Layer thickness deviations .....	44	<b>2.1</b>	Robotic workplace .....	3
<b>6.2</b>	Cylinder parameters deviations .....	46	<b>2.2</b>	Perpendicular printing .....	4
			<b>3.1</b>	Configuration with T-Mac .....	7
			<b>3.2</b>	Configuration with RRR.....	8
			<b>3.3</b>	Automation cable .....	9
			<b>3.4</b>	Frame from slave .....	10
			<b>3.5</b>	Inputs in WorkVisual .....	11
			<b>3.6</b>	Frame from master .....	11
			<b>3.7</b>	Outputs in WorkVisual .....	13
			<b>3.8</b>	Float32 .....	15
			<b>3.9</b>	Double64 .....	15
			<b>3.10</b>	Double converter .....	16
			<b>3.11</b>	Whole converter.....	17
			<b>3.12</b>	Configuration with PC.....	19
			<b>4.1</b>	Calibration diagram.....	21
			<b>4.2</b>	DKP coordinate system.....	22
			<b>4.3</b>	Incremental movement .....	24
			<b>4.4</b>	T-Mac face 3 .....	25
			<b>5.1</b>	Printed part .....	31
			<b>5.2</b>	Circle acquisition.....	32
			<b>5.3</b>	Macro Script program .....	33
			<b>5.4</b>	Polyworks Inspector.....	35
			<b>6.1</b>	Robodyn.....	41
			<b>6.2</b>	Plane and points .....	45
			<b>6.3</b>	Layer thicknesses deviations ...	45
			<b>6.4</b>	Cylinder .....	46
			<b>B.1</b>	Laser Tracker.....	55
			<b>B.2</b>	Touch Probe.....	55
			<b>B.3</b>	Automation Interface .....	56
			<b>B.4</b>	Object inspection .....	56



# Chapter 1

## Introduction

The bachelor thesis deals with the automation of a precision measurement process using a laser tracker and an industrial robot. The principle of the precision measurement function is that the robot carries a tool which is 6DoF (Degrees of Freedom) tracking device for automated applications, equipped with a touch probe. On demand, the laser tracker measures the position of the 6DoF tracking device in space.

Thanks to these devices, it is possible to design and develop an application for automatic calibration of the robot workspace. In addition, it is possible to use the device to accurately measure objects in space with a touch probe, as is the case with Coordinate Measuring Machines (CMMs). The advantages over CMMs are greater flexibility in the selection of measurement positions, and when equipped with a conveyor, the capacity for output inspection on the production line can be significantly increased.

A secondary contribution of this work is the use of the results of robot workspace calibration for multi-axis additive manufacturing. Accurate calibration of the robot workspace is required to link the external kinematic chain to the robot controller. Furthermore, it is possible to use the laser tracker in feedback with the robot controller to refine the robot trajectories, especially in linear interpolation.

This project is being developed at the Testbed for Industry 4.0, which is part of the Czech Institute of Informatics, Robotics, and Cybernetics (CIIRC CTU)<sup>1</sup>.

### 1.1 Goals

The main objective of this work is to design and develop a process for automatic robot workspace calibration using a laser tracker and an industrial robot. The next objective is to design and develop a process for accurate measurement of a part by touch probe with the same devices.

To accomplish these goals, there is a need to provide communication between the laser tracker and the robot controller so that the laser tracker can send measured data to the robot controller, and the robot can send commands to the laser tracker. The received data must also be converted to the correct format so that the robot controller can process them.

The result of the automatic calibration of the robot workspace is to find the transformation matrix between the robot root and the root of the positioning table. The second transformation matrix to be found is the transformation matrix between the laser tracker coordinate system and the robot coordinate system. The matrix is needed in order to compare the robot positions with those measured by the laser tracker. This matrix can also be obtained from the Robodyn software, which is primarily used for robot calibration. It is therefore convenient to compare these methods.

An automatic inspection of a manufactured part process can automatically determine if the part has been manufactured within specified tolerances. Part inspection can be

---

<sup>1</sup> <https://ricaip.eu/testbed-prague/>

created both with and without the Polyworks Inspector metrology software. These methods will also be compared with each other. If no third-party software is used, there is a great opportunity to automate the inspection process in the meaning of continuous quality control. Such a solution can be used at the output of a production line and could automatically measure its products and detect faulty ones. In this solution, the laser tracker, alongside the 6DoF tracking device, works similarly to CMM [1].

# Chapter 2

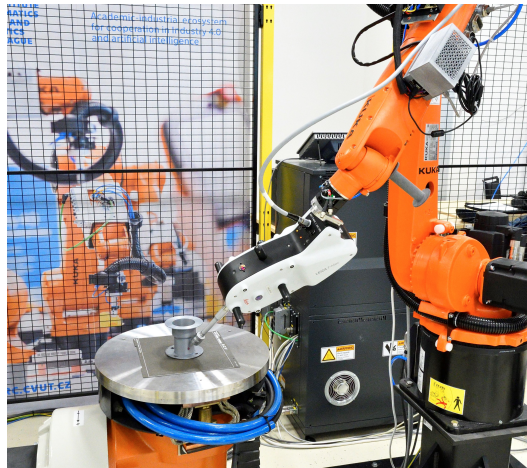
## Main components of the robotic workstation

The hardware and software used at the robotic workstation will be introduced before the solution of the specified goals of the work is presented. Industrial robots are quite common in the automation of manufacturing processes, whereas laser trackers are not often used in manufacturing plants. Therefore, the laser tracker and its components will be presented in more detail. Software products will also be introduced in short. Rather they will be described at the places in the thesis where they are needed and used.

### 2.1 Hardware

#### 2.1.1 Industrial robot and positioner

- **Kuka KR 8 R1620** is a 6-axis robot arm with a rated payload of 8 kg, a maximum reach of 1620 mm, and with pose repeatability of 0.04 mm. The manufacturer does not provide this accuracy for absolute positioning, which is needed in additive manufacturing. This robot is suitable for the workstation because it can carry a 6DoF tracking device and also an extruder for 3D printing. It is worth mentioning that Kuka KRC4 small size 2 controller is used [2].
- **Kuka DKP-400** is a 2-axis positioner external kinematics with pose repeatability 0.1 mm. The rotating axis ( $E1$ ) can rotate in the range  $\pm 90^\circ$ , and the rotating axis ( $E2$ ) can rotate endlessly. It serves as a pad on which it is printing [3]. Workplace can be seen in the Figure 2.1. Thanks to the positioner kinematics, turning and tilting with the printed part is possible. It allows to print perpendicular to the pad, as shown in Figure 2.2. Printing without any supports is possible in contrast to other conventional 3D printers.



**Figure 2.1.** Workplace with Kuka KR 8 R1620 robot and Kuka DKP-400 positioner.



**Figure 2.2.** Example of perpendicular printing.

### 2.1.2 Laser tracker and accessories

- **Absolute Tracker AT960** (Figure B.1) is a laser tracker that measures a point in space using a class 2 laser (composed of multiple class 1 laser beams). As the target can be multiple devices, at the workstation, 1.5'' and 0.5'' red ring reflectors (RRR) are used for measuring a single point, and T-Mac (Tracker-Machine control sensor). When using the T-Mac, it can even measure 6DoF.

The native coordinate system of the measured point is defined in spherical coordinates  $(\varphi, \theta, r)$ . The laser source has two orthogonal rotating axes that can move horizontally and vertically. Using encoders on these axes, the first two coordinates of the point are obtained.  $\varphi$  is the coordinate of the horizontal axis, and  $\theta$  is the coordinate of the vertical axis [4].

Distance  $r$  is measured with the Absolute Interferometer (AIFM) module, which combines an Interferometer (IFM) for precision of a dynamic measurements with an Absolute Distance Meter (ADM) to set the absolute reference distance. Both modules measure the distance to the reflector simultaneously. The IFM finds out the phase between emitted and reflected signals to measure small differences in distance, and it can detect quick changes in distance. The Interferometer uses a visible laser beam. The ADM uses an invisible modulated laser beam. Modulation by a polarization of laser beam is used. It is precise, but it takes some time to analyze the incoming modulated waves, so the absolute position is first determined using the ADM, then the relative deviations from it are dynamically determined using the interferometer [5–6].

The accuracy of the laser tracker is defined by the Maximum Permissible Error (MPE), which is defined by the Automotive Society of Mechanical Engineers (ASME) [7]. It specifies the maximum error value permissible for a given measurement specification and target used. The inaccuracy of the measurement of the distance of a



point in space is theoretically defined as the difference between the measured value and the nominal value. The measurement inaccuracy is dependent on the target distance. With increasing distance, the accuracy decreases. For a 1.5'' RRR, the MPE uncertainty  $U$  of a distance coordinate  $r$  is defined as:

$$U_r = \pm 15 \mu m + 6 \mu m/m.$$

The accuracy of the distance measurement is dependent on the ambient air temperature and atmospheric pressure, so a temperature and pressure sensor must be connected to the laser tracker to compensate for these influences during the measurement [4].

- 1.5'' **RRR** is a passive target composed of three orthogonal mirrors. It is used to measure a point in  $\mathbb{R}^3$ . Its design ensures that the beam is reflected in the same direction as it entered, that is, into the laser tracker. Its diameter is 38.1 mm, and its acceptance angle of the laser beam is  $\pm 30^\circ$  [8].
- **T-Mac Multiside TMC30-M with a touch probe** is an active target for measuring a 6DOF object in  $\mathbb{R}^3$ . Information about the position of the point is obtained in the same way as for an RRR reflector because there is one on its surface. Rotation information is provided in quaternions. There are 7 Infrared Light Emitting Diodes (IR LEDs) distributed over the surface of the T-Mac; several of them are even elevated above the surface to provide 3D distribution. From the position of these LEDs, which illuminate the laser tracker, the rotation of the T-Mac is defined. The T-Mac Multiside differs from a simple T-Mac in that it has 4 faces that can be used for measurement. Each face has its own reflector and a set of 7 IR LEDs [9]. The T-Mac can be seen in Figure 2.1, when it is mounted on the flange of the robot. A **touch probe with a ruby tip** (Figure B.2) can be attached to the T-Mac. This is used to measure the surface of an object; as soon as the probe touches an object, a point, where the probe touched the object is sampled. When several points are measured, a point cloud of the measured object is obtained. The way the probe works is that when it touches the surface, the electrical contact inside the probe breaks; this triggers a measurement. The probe is triggered by a force of 0.1 N [10].
- An **Automation Interface Controller** (Figure B.3) is required to process the signals from the T-Mac and send it to laser tracker. It can be also connected to the robot's controller.

## 2.2 Software

### 2.2.1 Robot programming

Kuka **WorkVisual** is software for hardware configuration, offline programming, and diagnostics of KR C4 control systems. It allows programming the robot and external kinematics in KRL language, configuring inputs, outputs, and field buses, configuring optional packages, and editing TOOL and BASE coordinate systems.

Kuka **KRL** is a procedural programming language used for programming Kuka robots. A KRL program consists of SRC and DAT files. The SRC file contains the commands themselves, and the DAT file is used to store program data, constants or positions. SUB files are special programs that run continuously even when no SRC file is selected. They are used, for example, to parallel monitoring of selected variables or to control interrupts [11].

Kuka **RSI** is an add-on option package to WorkVisual. It is used for cyclic data exchange and processing between sensors and robot controller. For example, it can be used to correct robot trajectories based on incoming sensor data. RSI creates so-called RSI Contexts that run parallel with the SRC program. One cycle of an RSI context is 12 ms or 4 ms; this can be chosen. Its programming is done in a dedicated graphical editor where signals are connected with function blocks. The user can create his own function blocks. It is possible to transfer variables between the RSI context and the SRC file and vice versa [12].

### ■ 2.2.2 Metrology software

**Tracker Pilot** is the software included with the laser tracker and is used for accuracy checks, system maintenance, firmware updates, defining the targets used, and also for measurement. It is a good idea to perform these accuracy checks once in a while to maintain the quality of the measurements. During these procedures, several points must be measured at defined positions. The Tracker Pilot guides the user through these processes. This software cannot process the measured data, only export it to a CSV file [13].

**Robodyn** is metrology software developed by Hexagon Manufacturing Intelligence, which is mainly used for robot calibration. It evaluates the positioning accuracy or can adjust the Denavit–Hartenberg (DH) [14] parameters of the robot. It can also perform ISO tests, for example ISO 9283. These norms test the robot in, for example, position accuracy and repeatability, trajectory velocity characteristics or position stabilization time [15].

**Polyworks** manufactures several products for objects modeling or measurement. The **Inspector** software is used at the robotic workplace. It can be used for comprehensive inspect of some manufactured part. In this software, reference objects can be created; single points or point clouds can be measured; from them can be created whole features. Furthermore, the measured data can be aligned and compared with the reference ones. Lastly, measurement reports can be generated [16]. Polyworks Inspector supports measurements with both RRR and T-Mac targets, including touch probe.

## Chapter 3

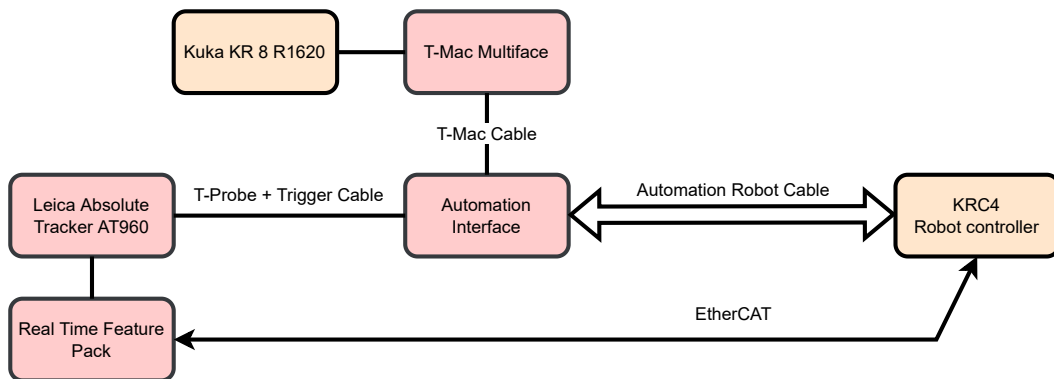
### Communication with the laser tracker controller

For different applications of the laser tracker, various ways of communication exist. The laser tracker can communicate only with an application computer or additionally with the robot controller. For basic 3DoF measurements with the Tracker Pilot or Polyworks Inspector is sufficient connection via Ethernet. When a robot is connected, Real-time communication via EtherCAT protocol is available. For 6DoF measurements is needed to connect T-Mac alongside the Automation interface. It can be connected with the robot's controller digital I/O via the Automation cable [17].

Tracker Pilot and Polyworks Inspector are automatically converting raw received data to preferred coordinate systems [13, 16]. However, when using the real-time module, the received data is in binary, so it must first be converted to decimal format. Then the translation coordinates must be converted from spherical coordinates to cartesian coordinates and the rotation from quaternions to a rotation matrix.

#### 3.1 Connecting the laser tracker

The layout in Figure 3.1 enables measurements of 6DOF without an application PC. That means without any metrology software [17–18]. Measured data is stored in the robot controller, where it can be directly evaluated, or just stored and sent to a desktop PC after measurement is completed. Therefore, this configuration can be used for workspace calibration, where both robot and laser tracker data are saved to the robot's hard drive. This configuration can also be used for automatic inspection of the manufactured part. This will again record the laser tracker data to the robot's hard drive controller and then send and evaluate it on the PC in Matlab. When it is used to correct robot trajectories, only the 3DoF is measured. Passive target RRR is used. Hence, the T-Mac and Automation Interface are removed from the schematic, which reduces to those in Figure 3.2.



**Figure 3.1.** Wiring diagram with T-Mac, Real Time module without application computer.

```

graph LR
    A[Leica Absolute Tracker AT960] --- B[Real Time Feature Pack]
    B <-->|EtherCAT| C[KRC4 Robot controller]
  
```



## E-1 GATE: \_\_\_\_\_ ;

Whether a slave or a master, the EtherCAT device typically has two Ethernet ports. The device receives a frame from the previous device with the first port and with the second port, the device sends the frame to the next device. Often, a single frame goes through the nodes from the master; it goes from one slave to another. Every slave conveys only the part of the frame addressed to him. Data reading and writing in a single node are done within several nanoseconds. The most common network typologies are supported by the EtherCAT, such as star, tree, line, and bus. Ring topology is very useful because a frame sent by the master terminates again in the master node [19–20].

At the current robotic workstation, there is just a single master and a single slave. The laser tracker as a slave automatically returns the frame to the master, the Beckhoff EtherCAT Coupler<sup>1</sup>, which is connected to the robot controller.

Use of the RTFP-EC module is advantageous because all communication programming between the laser tracker and the robot's controller is done in the WorkVisual. There is no need for additional software such as Polyworks Inspector. The laser tracker inputs and outputs are mapped in WorkVisual. Therefore, it is possible to work directly with the raw data from the laser tracker and send commands to the laser tracker to control it. Additionally, another laser tracker can be connected to the network; unfortunately, at the robotic workstation there is not a second laser tracker [18]. The

<sup>1</sup> <https://www.beckhoff.com/cs-cz/products/i-o/ethercat-terminals/ek1xxx-bk1xx0-ethercat-coupler/ek1100.html>

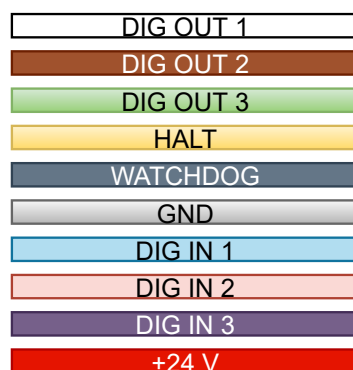
disadvantage is that there is no library from the Leica manufacturer for this communication, so the received data are binary. It was necessary to design a converter to process them.

The Real Time module could provide measured data from a 3DoF or a 6DoF target. Those are sent in a cycle with a frequency up to 1 kHz, but the manufacturer recommends 250 Hz. It can asynchronously receive commands from the robot controller to control the laser tracker. For example, send a laser beam to the specified position or perform a stationary measurement [18].

### 3.1.3 Connection of the Automation Robot Cable

Automation Robot Cable particularly serves for handshake signals between the robot and an application program and for status signals from the probe. Individual signals, shown in Figure 3.3, have the following functions:

- **GND, +24 V**—GND stands for ground, the reference level of signals 0 V. +24 V is a power supply with a current of 300 mA. So the nominal voltage level of all signals is either 0 V or 24 V. Logical “1” is represented by 24 V, and logical “0” by 0 V.
- **DIG IN 1, DIG IN 2, DIG IN 3**—The robot informs the application software about its status through digital inputs. Their function depends on the user. For example, to indicate whether the robot program is active and the robot is ready to go to the next position.
- **DIG OUT 1, DIG OUT 2, DIG OUT 3**—Similarly, with the digital outputs, the application software can give commands to the robot. For example, when DIG OUT 1 is in a *high* state, the robot can move to the next position, and when in *low*, the robot has to stop and wait.
- **HALT**—This signal is practical when the touch probe is connected to the T-Mac. The HALT goes into a *low* state when the probe is triggered; otherwise, it stays in *high*. It provides information that the probe has just touched an object, and the robot should stop.
- **WATCHDOG**—The watchdog is checking the LAN connection between the Automation Interface and the application PC. Another internal watchdog communication takes part between them. When the connection is broken, the Watchdog signal turns to a *low* state; otherwise, it stays in a *high* state [17].



**Figure 3.3.** Signals in the Automation Robot cable.

## 3.2 Communication protocol

### 3.2.1 Input frame from Slave to Master

The frame that a slave (the laser tracker) sends to a master (RSI Context) is 88 bytes long. It contains the part with measured data, the **Cyclic Measurement** (68 bytes), and the second part, **Command Response** (20 bytes), serves as the acknowledgment of the command request from the master. The parts of the Cyclic Measurement are in Figure 3.4. The Command Response part is very similar to the Command Request from a master. It will be explained in Section 3.2.2. The differences are that instead of Command Control, the slave has **Command Status**, and it has only 4 Arguments.

Name	Width
Cyclic Measurements	68 bytes
Command Response	20 bytes

Name	Width	Data Type
Tracker Status	4 bytes	UDINT
AngleHz	8 bytes	LREAL
AngleVt	8 bytes	LREAL
Distance	8 bytes	LREAL
Quaternion0	8 bytes	LREAL
Quaternion1	8 bytes	LREAL
Quaternion2	8 bytes	LREAL
Quaternion3	8 bytes	LREAL
Time Stamp	8 bytes	ULINT

**Figure 3.4.** Laser tracker to the robot controller frame—Input data.

Tracker Status contains information about the taken measurement. For obtaining correct information, it is necessary to interpret the Tracker Status bit by bit. For example, the second least significant bit says if the measurement is valid. Next, there is information on whether accuracy was out of range or which error occurred, even what face of the T-Mac was used, etc. A detailed description can be found in the manual [18]. In the project's current phase, it is only detected if the measurement went successful. What is expected from the laser tracker is known, so controlling other parameters was unnecessary. For optimal solution for automated processes, it will be necessary to look at other parameters.

The next part of the frame is the measured values themselves. Coordinates of the measured point are provided in a spherical coordinate system  $(\varphi, \theta, r)$ . AngleHZ ( $\varphi$ ) gives a horizontal angle in radians. Similarly, AngleVt ( $\theta$ ) gives a vertical angle, and Distance ( $r$ ) is a radius in meters. If T-Mac is connected, the measurement contains information about rotation. This is delivered in the form of quaternions:  $q_0, q_1, q_2, q_3$ .

Those numbers are in format LREAL (Long Real), so the width is 8 byte (64 bit). This format is incompatible with the REAL data type in WorkVisual. The robot controller cannot process Long Real. For this reason, these input numbers cannot be automatically converted to any variable in the WorkVisual. An additional converter in RSI was created to interpret incoming numbers to the format of REAL, which is only 32 bit [11, 18]. The converter is described in Section 3.3.

The last part of the input frame is Time Stamp. It gives information about when the data was written to the frame by the slave. EtherCAT master uses it for synchronization [20].

Figure 3.5 represents I/O mapping in Workvisual. It shows where the data from the laser tracker is mapped. Input addresses from 40 to 680 serve as inputs to the RSI context.

Name	Type	Address	I/O	I/O	Name	Type
CmdIF Response.Argument1	UDINT	1136	← 0x00	→ 0x00	Input40#G	UDINT
CmdIF Response.Argument2	UDINT	1168	← 0x00	→ 0x00	Input72#G	UDINT
CmdIF Response.Argument3	UDINT	1200	← 0x00	→ 0x00	Input104#G	UDINT
CmdIF Response.Argument4	UDINT	1232	← 0x00	→ 0x00	Input136#G	UDINT
CmdIF Response.Command__ID	USINT	1112	← 0x00	→ 0x00	Input168#G	USINT
CmdIF Response.Command__jobID	USINT	1128	← 0x00	→ 0x00	Input176#G	USINT
CmdIF Response.Command__status	BYTE	1120	← 0x00	→ 0x00	Input184#G	BYTE
CmdIF Response.Command__Version	USINT	1104	← 0x00	→ 0x00	Input192#G	USINT
Cyclic Measurements.AngleHz	LREAL	592	← 0x00	→ 0x00	Input200#G	LREAL
Cyclic Measurements.AngleVt	LREAL	656	← 0x00	→ 0x00	Input264#G	LREAL
Cyclic Measurements.Distance	LREAL	720	← 0x00	→ 0x00	Input328#G	LREAL
Cyclic Measurements.Quaternion0	LREAL	784	← 0x00	→ 0x00	Input392#G	LREAL
Cyclic Measurements.Quaternion1	LREAL	848	← 0x00	→ 0x00	Input456#G	LREAL
Cyclic Measurements.Quaternion2	LREAL	912	← 0x00	→ 0x00	Input520#G	LREAL
Cyclic Measurements.Quaternion3	LREAL	976	← 0x00	→ 0x00	Input584#G	LREAL
Cyclic Measurements.Status	UDINT	560	← 0x00	→ 0x00	Input648#G	UDINT
Cyclic Measurements.TimeStamp	ULINT	1040	← 0x00	→ 0x00	Input680#G	ULINT

**Figure 3.5.** Input data mapping in WorkVisual.

### 3.2.2 Output frame from Master to Slave

The robot controller, as a master, can send predefined commands to the laser tracker. The Command IDs can be sent from a PLC<sup>2</sup> (Programmable Logic Controller) for an application for automated measurements. There are driving parameters: Version, Command ID, Command Control, and Job ID. Next, there are arguments that have various uses in a command's specification. The individual parts of the frame are shown in Figure 3.6.

Name	Width	Data Type
Version	1 byte	USINT
Command ID	1 byte	USINT
Command Control	1 byte	USINT
Job ID	1 byte	USINT
Argument 1	4 bytes	
Argument 2	4 bytes	
Argument 3	4 bytes	
Argument 4	4 bytes	
Argument 5	4 bytes	
Argument 6	4 bytes	

**Figure 3.6.** Robot controller to Laser tracker frame—Output data.

The Version variable is useless for the current application. The version number can be incremented with every frame. For example, an error can be trackable when the version number is known. Command ID specifies which command the laser tracker

<sup>2</sup> <https://new.siemens.com/cz/cs/products/automation/systems/industrial/plc.html>



should execute. Command Control says whether some command is active or no action is required. Lastly, Job ID has to be incremented for every new command.

If the master wants to start processing the command, it will set Command Control from 0x00 to 0x01. In contrast, the slave has six states for Command Status:

Entry state:	0x00
Initialized:	0x01
Start:	0x03
Processing:	0x07
Completed:	0x0f
Error:	0x13

There are seven predefined Commands, but the most useful commands are these four:

- **Laser go-to position** will lead the laser tracker head to the defined position specified in Argument1—horizontal angle (in radians), Argument2—vertical angle (in radians), and Argument3—approximate distance (in meters). All these arguments are in 32 bit floats. Argument4 defines whether the laser tracker should find the nearest target: 0 for *no* and 1 for *yes* [18]. However, the only reason for selecting *no* is when the laser is guided to places where it is not assumed the laser tracker will measure. For example, not to shine in people's eyes when it is not measuring.
- **Stationary measurement** will perform one of the three stationary measurements according to Argument1: 0 is for Standard, which lasts 2 s, 1 is for Fast measurement (0.5 s), and 2 for Precise measurement (5 s) [13]. The result of the Stationary measurement will appear in the same positions as the results of the Cyclic measurement. They will stay there as long as the laser tracker is in state Completed.
- **Set Target ID** can preselect target, which is defined in the Tracker pilot. So the laser tracker will know what the target is. It can be useful when targets switch between 1.5'' and 0.5'' reflectors. This feature cannot be performed when T-Mac is used, it is detected automatically.
- **Shutdown Tracker.** This command will shut down the laser tracker and terminates all communication.

For all commands above apply, as a response from the slave comes only in Argument1, it is reserved for an error message if an error occurred. There are 10 error messages. Again, their exact description does not need to be presented, some of them are: some of the driving arguments are invalid, the previous command is pending, the command was aborted, the Job ID was incremented at the wrong time, or an unknown error occurred.

The update rate of the EtherCAT is higher than RSI Context or KRL program. From the slave are coming the same frames until the master responds. Hence the EtherCAT master responds only to changes in frames from the slave. The master will respond when it detects that the slave has changed the value of the Command Status or Job ID parameters [18]. Since it does not matter that the master may not communicate quickly, the outgoing frame can be composed in KRL and sent to the slave.

Lastly, Figure 3.7 represents I/O mapping in WorkVisual. Output addresses from 40 to 256 are output from the RSI context to the laser tracker.



Name	Type	Address	I/O	I/O	Name	Type
CmdIF Request.Argument1	UDINT	12624	→	←	Output40#G	UDINT
CmdIF Request.Argument2	UDINT	12656	→	←	Output72#G	UDINT
CmdIF Request.Argument3	UDINT	12688	→	←	Output104#G	UDINT
CmdIF Request.Argument4	UDINT	12720	→	←	Output136#G	UDINT
CmdIF Request.Argument5	UDINT	12752	→	←	Output168#G	UDINT
CmdIF Request.Argument6	UDINT	12784	→	←	Output200#G	UDINT
CmdIF Request.Command__Control	BYTE	12608	→	←	Output232#G	BYTE
CmdIF Request.Command__ID	USINT	12600	→	←	Output240#G	USINT
CmdIF Request.Command__jobID	USINT	12616	→	←	Output248#G	USINT
CmdIF Request.Command__Version	USINT	12592	→	←	Output256#G	USINT

Figure 3.7. Output data mapping in Workvisual.

### 3.2.3 Algorithm for the laser tracker automated control

This section shows an example algorithm for sending a command to the laser tracker. The idea to the future is that some PLC sends commands to the robot controller, which action should be performed. When the automatic inspection will be installed at the end of the production line, which is controlled by a PLC, it will be above the laser tracker and the robot. Then the PLC will give the measurement commands. As was written in the previous section, there are four possible actions. When the robot is ready to take a measurement, the PLC will send a request, and the robot, together with the laser tracker, will perform the action. When it is done, the robot is ready for a new command from the PLC.

At the moment, all four actions for the laser tracker are programmed in KRL and RSI and can be arbitrarily arranged one behind the other. Unfortunately, the program cannot communicate with the PLC. However, this is no big challenge. The focus was on establishing communication between the robot and the laser tracker and not on PLC programming.

The program for the PLC can take different forms depending on the application. There will be three variables needed in every case: `PlcAction`, `PlcActionRequest`, and `RobotReady`. When the robot is ready for receiving a command, the variable `RobotReady` is set to `True`, thus signaling the PLC its state. `PlcActionRequest` will generate only a single pulse signaling that new command is required and the action number is already ready in integer—`PlcAction`. The range of this variable is from 1 to 4. At this moment, the variable `RobotReady` is set to `False`. The robot cannot receive another command, and the selected action is handled. This sequence runs in a loop.

The algorithm below shows only one command for the laser tracker: Laser go-to position. The other actions vary in sent Arguments. This algorithm is implemented mainly in KRL. The outgoing data frame is created bit by bit, as it is defined. RSI Context is used only for transferring inputs from the laser tracker to KRL variables and vice versa. For clearance of the code, when the variable with the same name comes from the slave, it has at the end character *S*:

```

Set: All variables ← 0x00
wait for CommandStatus = 0x01 and JobIdS = 0x01 then    ▷ tracker initialized
  Version ← 0x11
  loop
    RobotReady ← True    ▷ sends an acknowledgment to the PLC
    wait for PlcActionRequest then
      RobotReady ← False
      switch PlcAction
        case 1 then    ▷ Laser go-to position

```

```

CommandId  $\leftarrow$  0x03
Argument1  $\leftarrow$  HzAngle
Argument2  $\leftarrow$  VAngle
Argument3  $\leftarrow$  Distance
Argument4  $\leftarrow$  0x01
JobId  $\leftarrow$  JobIdS + 1
CommandControl  $\leftarrow$  0x01
wait for CommandId = 0x03 and
    CommandStatus = 0x07 and JobIdS = JobId then
wait for CommandStatus = 0x0f or
    CommandStatus = 0x13 then
if CommandStatus = 0x0f then  $\triangleright$  Action ended correctly
    CommandId  $\leftarrow$  0x01
    CommandControl  $\leftarrow$  0x00
else  $\triangleright$  CommandStatus = 0x13
    Read Error message in Argument1S
    break
end if
case 2 then  $\triangleright$  Stationary measurement
    ...
case 3 then  $\triangleright$  Set Target ID
    ...
case 4 then  $\triangleright$  Shutdown Tracker
    ...
default
    break
endswitch
endloop

```

After the EtherCAT master is turned on and synchronizing with the slave on the EtherCAT bus is done, the communication between them can begin. They both must start with all variables in frames with the value 0x00. Then the tracker sets Command Status to 0x01 and Job ID to 0x01. The tracker is initialized and waits for a command.

The master now fills the outgoing frame with mandatory variables. Command ID says that Laser go-to position is required. Argument1 to Argument3 specify the coordinates where the laser tracker should aim. After that Job ID could be incremented. The slave will start executing the command only after Command Control is set to 0x01. The slave sets the same Version number and Job Id as the master, and the Command Status goes from 0x01 to 0x03—Start and then to 0x07—Processing. The action can be aborted by setting Command Control back to 0x00. After the laser tracker head is in its desired position, Command Status will be either 0x0f—Completed or 0x13—Error. When the master reads the result of the action in the slave's Command response, the Command Control can be turned to state 0x00—Stop.

In the case of the action of positioning the laser tracker, there are no return data. But, when the Stationary measurement is selected, the measured data are in position for Cyclic Measurement data as long as the slave is in state Completed and the master in state Start [18].

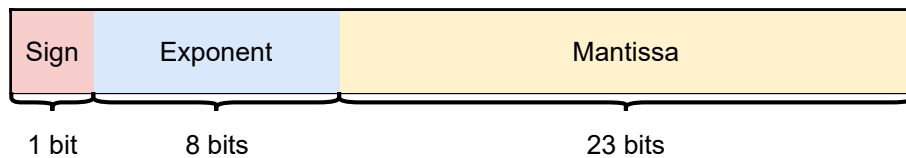
### 3.3 Conversion of incoming data

This section will show how to convert the measured data, which is obtained from the laser tracker, to a suitable format. The measured data are in the wrong decimal number format and in a different coordinate system format. The laser tracker controller sends 64 bit double, but the robot controller can handle only 32 bit float. A position is represented in spherical coordinates and a rotation in quaternions. It has to be converted to Cartesian coordinates and rotation matrix. The position coordinates along with the rotation are used in the robot workspace calibration in Chapter 4 and in the automatic part inspection in Chapter 5.

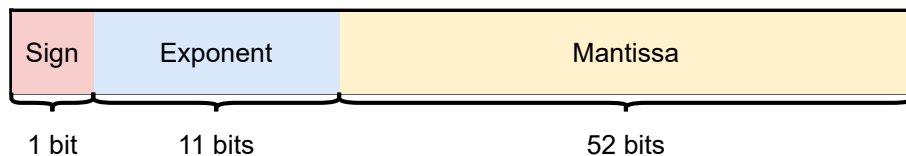
#### 3.3.1 IEEE Standard for Floating-Point numbers

The IEEE 754 standard defines several rules for notating real numbers on computers. From the laser tracker are incoming decimal numbers in 64 bit double precision format (Figure 3.9) but the robot controller can handle only 32 bit single precision format (Figure 3.8). The binary notation of a floating-point number is divided into three parts:

- **The Sign**—For both notations, it is one bit. 1 is for negative numbers, and 0 represents positive numbers.
- **The Exponent**—In single precision format occupies 8 bits, and in double 11 bits. Because both positive and negative exponent representation is needed, there is an offset to which the exponent is added. The offset is  $2^{n-1} - 1$ , where  $n$  is number of bits of exponent (8 or 11). The biases are 127 and 1023.
- **The Mantissa**—23 bit for single precision float and 52 bit for double precision. It is the part with significant digits. In order to be written correctly, it must be normalized, for example 1.01101. Since the first digit is always 1, to save bits, it is omitted [21].



**Figure 3.8.** Scheme of the binary representation of the single-precision floating-point number.

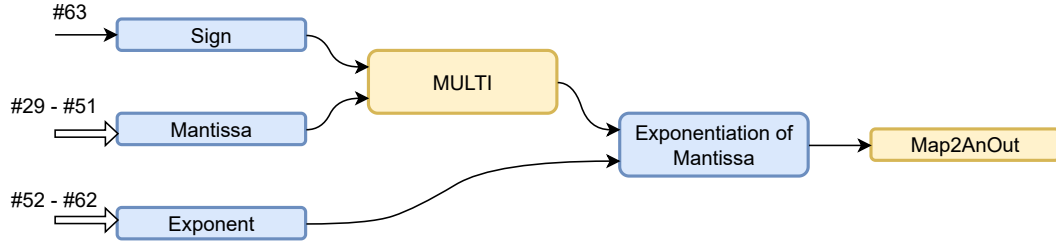


**Figure 3.9.** Scheme of the binary representation of the double-precision floating-point number.

#### 3.3.2 Implementation of the Real numbers converter

The first task of the converter is to represent raw measured data that arrives in a frame from the slave. On the input there are 64 bits, and the output is a REAL type number. This is common to all coordinates of position and rotation. The second part takes these numbers and converts them to Cartesian coordinates and rotation matrix. Next, they are mapped to the variable \$ANOUT[index] in KRL. These two operations are implemented in RSI using several custom-made function blocks. For

testing purposes, the converted numbers are just sent to the KRL and saved to a text file so they can be checked for accuracy. Figure 3.10 schematically describes the first operation—converting input to a number suitable for further calculations. The yellow blocks are functions already implemented RSI, and the blue boxes are custom-made function blocks.



**Figure 3.10.** Schematic description of the algorithm converting a number from the laser tracker to a format suitable for KRL or RSI.

Function block Sign represents equation

$$s = -2 \cdot n + 1,$$

where  $s$  is the required sign and  $n$  incoming bit. The output is either 1 or  $-1$ . For input  $n = 0$  (positive number) the output is  $s = 1$ ; for input  $n = 1$  (negative number) the output is  $s = -1$ .

Function block Mantissa is an implementation of equation

$$m = 1 + (\#51 \cdot 2^{-1}) + (\#50 \cdot 2^{-2}) + \dots + (\#29 \cdot 2^{-23}).$$

It composes a mantissa from binary numbers into a decimal number. The 1 is added because the normalized form of the floating point number always starts with a 1 and therefore is not included in the data frame, it must now be added back. Function block Exponent is quite similar to the Mantissa. It just does not add 1, it subtracts 1023, which is the offset for the exponent [21]. The equation is

$$e = (\#52 \cdot 2^0) + (\#53 \cdot 2^1) + \dots + (\#62 \cdot 2^{10}) - 1023.$$

Function block MULTI is simply multiplying two numbers. Furthermore, the function block Exponentiation of Mantissa multiplies the output of the MULTI function block by  $2^e$ . It was necessary to create this function block because in RSI does not exist a function block that can do a power of some number with a variable exponent [12]. Overall, the converter calculates the equation

$$result = s \cdot m \cdot 2^e.$$

### 3.3.3 Discussion on precision loss

It is important to question whether cutting off half of the bits will result in the loss of important information. When working with single bits, it is not a problem to read a 64 bit number, but it should be stored in the REAL data type, so only the half of the 64 bit number that can fit into REAL. The exponent cannot exceed the range from -126 to 127. There is also less room for significant digits. Mantissa has only 23 bits available instead of 52 bits [21]. However, this is still enough to record the numbers from the laser tracker. The measurement resolution of both the ADM (Absolute Distance Meter)

and the encoders are much worse than the float32 range. The laser tracker gives us distance information in  $\mu m$  [4], so an example of how to write a  $1 \mu m = 1 \times 10^{-6} m$  in float32 type is presented:

0 01101011 00001100011011110111101.

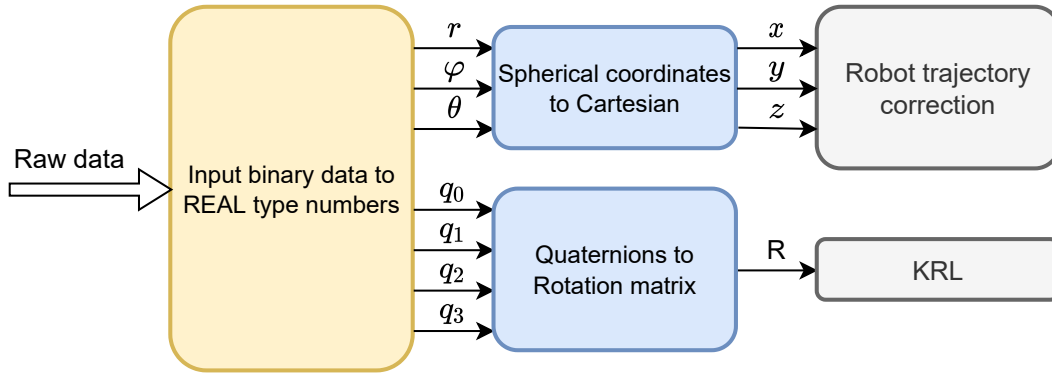
In reality, this binary representation corresponds to a number

$$9.999999974752427078783512115478515625 \times 10^{-7},$$

which is a much better resolution of  $1 \times 10^{-6}$  than what the laser tracker sensors can provide.

### 3.3.4 Coordinate system conversion for positions

As soon as all numbers are in the correct format, they can be converted into the proper coordinate systems. Again, two function blocks in RSI were created, one for converting position from spherical coordinates to Cartesian coordinates and the second for converting quaternions to rotation matrix. Figure 3.11 shows the complete converter.



**Figure 3.11.** Diagram of complete converter of binary data from the laser tracker.

The function block for converting position has three inputs, distance— $r$ , horizontal angle— $\varphi$ , and vertical angle— $\theta$ . The output is coordinates  $x$ ,  $y$ , and  $z$  relative to the laser tracker coordinate system [22–23]. In the RSI context, there are implemented these equations:

$$x = r \cdot \sin(\varphi) \cdot \cos(\theta), \quad (1)$$

$$y = r \cdot \sin(\varphi) \cdot \sin(\theta), \quad (2)$$

$$z = r \cdot \cos(\varphi). \quad (3)$$

### 3.3.5 Coordinate system conversion for rotations

The rotation in the space  $\mathbb{R}^3$  can be defined in several ways, the axis-angle ( $\mathbf{s}$ ,  $\theta$ ) notation can be used as a default. This notation assumes that any rotation in  $\mathbb{R}^3$  can be performed as a rotation by the angle  $\theta$  around the vector  $\mathbf{s}$  [24]. There are equivalent notations to this one called quaternions.

Quaternions are complex numbers with three complex units. They are notated as  $q = q_0 + iq_1 + jq_2 + kq_3$ , where  $q_0$ ,  $q_1$ ,  $q_2$ , and  $q_3$  are real numbers; and  $i$ ,  $j$ , and  $k$  are the complex units [25]. Let us represent the quaternions by a vector in  $\mathbb{R}^4$ :

$\mathbf{q} = (q_0 \ q_1 \ q_2 \ q_3)^T$ . Let the components of the vector  $\mathbf{s}$  be  $\mathbf{s} = (s_x \ s_y \ s_z)^T$ , then the conversion relation from the axis angle notation is

$$\mathbf{q} = \left( \cos\left(\frac{\theta}{2}\right) \quad \sin\left(\frac{\theta}{2}\right)s_x \quad \sin\left(\frac{\theta}{2}\right)s_y \quad \sin\left(\frac{\theta}{2}\right)s_z \right)^T. \quad (4)$$

However, conversion relationship from quaternions to axis angle notation is needed. The angle  $\theta$  can be first isolated:

$$\theta = 2 \cdot \arccos(q_0). \quad (5)$$

$q_0$  belongs to the interval  $\langle -1, 1 \rangle$  which is the same interval as the domain of the function  $\arccos(x)$ . Knowing the angle  $\theta$ , it is not difficult to calculate the remaining parameters:

$$s_x = \frac{q_1}{\sin\left(\frac{\theta}{2}\right)}, \quad s_y = \frac{q_2}{\sin\left(\frac{\theta}{2}\right)}, \quad s_z = \frac{q_3}{\sin\left(\frac{\theta}{2}\right)}. \quad (6)$$

The axis angle notation is not used in any application at the robotic workstation. The rotation must be further converted to a rotation matrix  $\mathbf{R}$  [26]. This can be done using Rodrigues' formula:

$$\mathbf{R} = \mathbf{I}_3 + \sin(\theta)\mathbf{S} + (1 - \cos(\theta))\mathbf{S}^2, \quad (7)$$

where  $\mathbf{I}_3$  is identity matrix and the matrix  $\mathbf{S}$  is the matrix of the cross product  $\mathbf{s} \times \mathbf{x}$ , where  $\mathbf{x} \in \mathbb{R}^3$  [27]. The matrix  $\mathbf{S}$  is skew symmetric.

$$\mathbf{S} = \begin{pmatrix} 0 & -s_z & s_y \\ s_z & 0 & -s_x \\ -s_y & s_x & 0 \end{pmatrix} \quad (8)$$

### 3.3.6 Euler angles conversion limitations

Before the conversion to the rotation matrix, the conversion to Euler angles was implemented. However, this was not an ideal procedure because part of the solution is lost. The solution was reworked to convert the quaternions to the rotation matrix. The Euler angles can be defined as

$$\mathbf{R} = \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x. \quad (9)$$

That is, as a sequential multiplication of the rotation matrix around the  $z$  axis, then around the  $y$  axis, and finally around the  $x$  axis [28–29]. The parameters of these matrices are  $r_z$ ,  $r_y$ ,  $r_x$ . Euler angles are further discussed in Section 4.3.4. The conversion from quaternions to the Euler angles was done according to these formulas [30]

$$r_z = \arctan\left(\frac{2(q_0q_3 + q_1q_2)}{1 - 2(q_2^2 + q_3^2)}\right), \quad (10)$$

$$r_y = \arcsin(2(q_0q_2 - q_3q_1)), \quad (11)$$

$$r_x = \arctan\left(\frac{2(q_0q_1 + q_2q_3)}{1 - 2(q_1^2 + q_2^2)}\right). \quad (12)$$

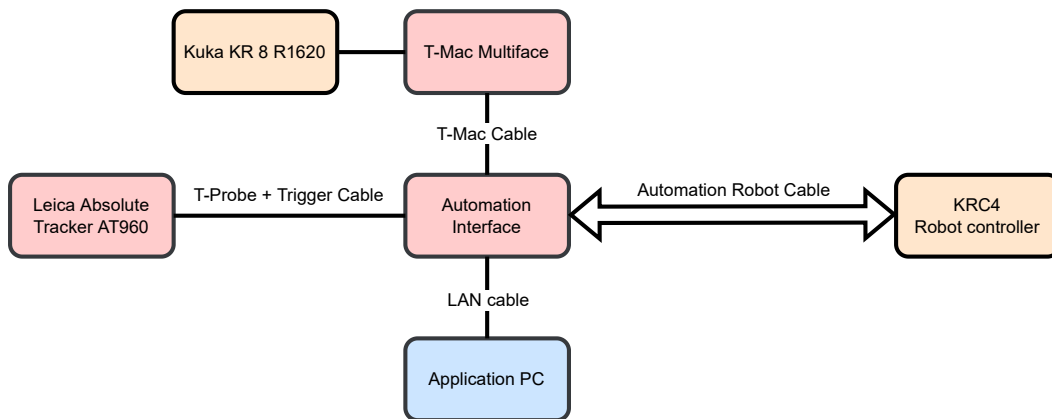
Function  $\arctan\left(\frac{y}{x}\right)$  returns results only in the interval  $\langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$ , so some results can be lost. Moreover, when  $x$  is close to zero, the problem with dividing by zero occurs. This problem solves the function  $\text{atan2}(y, x)$ , which returns a result in the interval  $(-\pi, \pi)$  [31]. However, the problem is with the  $\arcsin(x)$  function, which returns values only in the interval  $\langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$ . Actually, the problem in the conversion from quaternions to

Euler angles can be deduced from the fact that it is a transformation from  $\mathbb{R}^4$  to  $\mathbb{R}^3$ , so some information must be lost.

Another problem with the Euler angles is the so-called Gimbal lock. This is a singularity; it occurs when  $r_y = \pm 90^\circ$ . Then  $r_z$  and  $r_x$  have an identical effect on rotation, which is an undesirable condition [32].

### 3.4 Other configuration option

At the end of the chapter, the wiring of the laser tracker without the Real Time module is presented. The layout in Figure 3.12 also enables measurements of 6DoF [18]. This configuration is used when on an application PC runs either Polyworks Inspector or Tracker Pilot and the measured data are sent there.



**Figure 3.12.** Wiring diagram with T-Mac connected and without Real Time module.

## Chapter 4

### Automatic robot workspace calibration

The aim of this chapter is to design the procedure to get the transformation matrix between the robot root and the positioner root. Workstation calibration is important to ensure accurate additive manufacturing and it is also used to calibrate the parameters of the workstation's digital twin. Since digital twins are widely used for offline motion planning, their parameters must correspond as closely as possible to those of the real workstation.

The base of the positioner can indeed be taught by touching several points with TCP. It is done right in the Kuka SmartPad. Nevertheless, this method cannot provide as accurate base parameters as they are required. It is because the input data for this method are from the robot encoders at each axis, from which direct kinematics is computed. Moreover, this transformation is calculated from only 3 points [11], but in the presented method, it is calculated from much more points to make the method more robust.

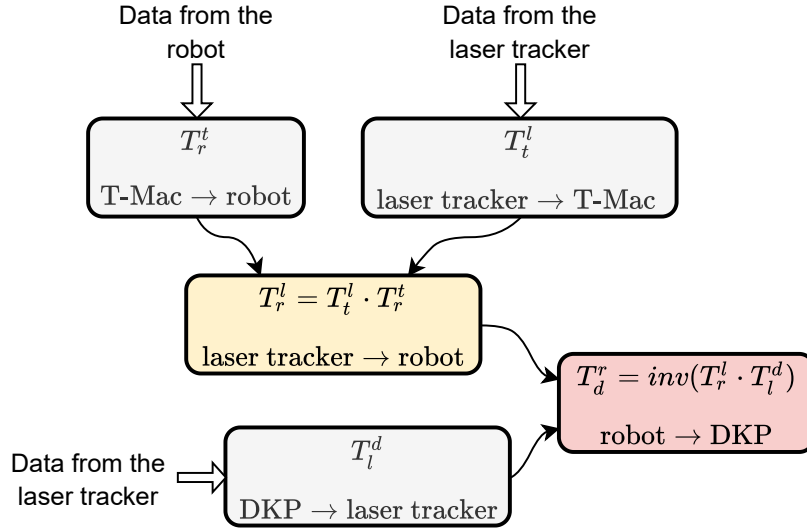
#### 4.1 Description of the workflow

As it has already been written, this chapter aims to obtain the transformation matrix between the root of the robot and the root of the positioner. This task consists of two parts: the method of obtaining the transformation between the laser tracker and the robot root  $\mathbf{T}_r^l$ , and obtaining the transformation matrix between the positioner and the laser tracker  $\mathbf{T}_l^d$ . After that, the two transformation matrices will be multiplied among themselves and inverted, resulting in the transformation matrix between the robot root and the positioner root.

$$\mathbf{T}_d^r = (\mathbf{T}_r^l \cdot \mathbf{T}_l^d)^{-1} \quad (1).$$

A workflow description of obtaining all the necessary transformation matrices is shown in Figure 4.1. As it is shown there, to get the transformation matrix  $\mathbf{T}_r^l$ , two more transformation matrices need to be combined, that is, a transformation matrix between the T-Mac and the robot  $\mathbf{T}_r^t$  and a transformation matrix between the laser tracker and the T-Mac  $\mathbf{T}_t^l$  [14].





**Figure 4.1.** Workflow diagram of obtaining the transformation matrix from the robot root coordinate system to the DKP root coordinate system  $T_d^r$  from matrices  $T_r^t$ ,  $T_t^l$ , and  $T_l^d$ .

## 4.2 Transformation between the positioner and the laser tracker $T_l^d$

This part of the task is more straightforward than the second one. The main idea is to align the coordinate system of the laser tracker with the coordinate system of the positioner and get the transformation matrix between them. The coordinate system can be defined by Plane, Axis, Center point method [16]. The computational part is done in Matlab. Points acquisition is automated; they are recorded by Tracker Pilot. The tasks of communication (Chapter 3) and workspace calibration were developed simultaneously, so Tracker Pilot was primarily used for data acquisition in the development of the calibration process. Measurements in the Tracker Pilot are triggered by the touch of the probe. The method described in Chapter 3 could also measure those points, and it will be used instead of the Tracker Pilot.

### 4.2.1 Features measurement

During the measurement, both axes of the positioner must be in zero positions. The workflow consists of the robot going around the positioner and measuring a plane-surface of the positioner ( $p_1$ ) and two circles ( $c_1, c_2$ ).  $c_1$  is the larger circle in the middle of the positioner, as shown in Figure 4.2, and  $c_2$  is the smaller circle. From both circles are extracted its centers ( $s_1, s_2$ ). The method of measuring the plane and center of the circle is the same as in Section 5.2, so it is explained in detail there. To obtain the plane, 12 points on its surface are measured and the center of the circle is obtained from 8 points. The equation of the plane is as follows:

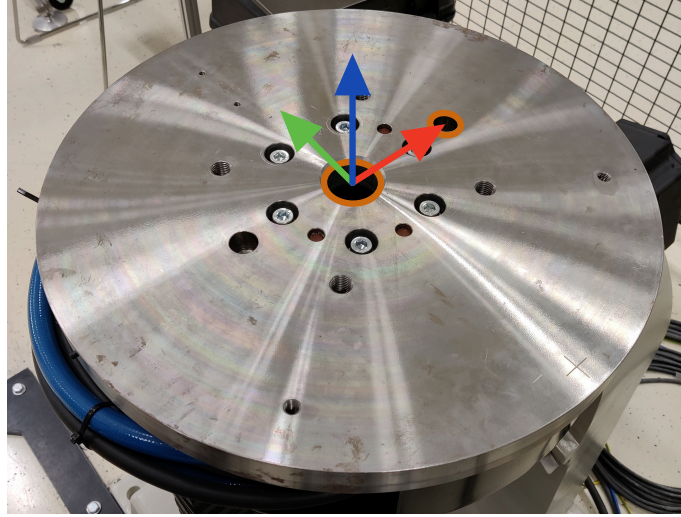
$$p_1: ax + by + cz + d = 0. \quad (2)$$

In this application, it is required that the constraining plane of both circles is the plane  $p_1$ . The centers of both circles are therefore projected into this plane. The  $z$  coordinates of the centers  $s_1$  and  $s_2$  are replaced by the value according to the Equation (2):

$$z = -\frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}. \quad (3)$$

In the alignment method, the  $s_1$  serve as the center point of the coordinate system. Between points  $s_1$  and  $s_2$  is created the axis  $a_1$ . The coordinate system on the top of the DKP table is created from features  $p_1, a_1, s_1$ . These features can be used to perform alignment according to the Plane, Axis Center point method.

It must also be noted that the root of the DKP is not on top of the positioner but 259 mm below it in direction  $z$  axis [3]. This number is subtracted from the translation in axis  $z$ . In Figure 4.2 can be seen the two circles marked in orange and the coordinate system on the table surface. The notation of the coordinate system:  $x$  axis—red,  $y$  axis—green,  $z$  axis—blue.



**Figure 4.2.** Marked coordinate system on the surface of the DKP with marked the two measured circles. The negative  $x$  axis points towards the robot root, and the  $y$  axis points towards the laser tracker.

#### 4.2.2 Plane, Axis and Center point alignment

Before the actual implementation of the method is presented, the theoretical basis of alignment is described. This alignment method is borrowed from Polyworks Inspector. Unfortunately, its manual does not say anything about the theoretical basis of this method, so its process must have been invented. A small hint is presented on the Autodesk PowerInspect site [33]. They say that the center point determines the translation; the plane locks the rotation around the  $x$  and  $y$  axes. Finally, the line determines the rotation around the  $z$  axis. It can be deduced how the alignment method is performed from this information. In this task, the transformation matrix is defined in form of

$$\mathbf{T} = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (4)$$

Where  $r_{1,1}, \dots, r_{3,3}$  are elements of the rotation matrix  $\mathbf{R}$  and  $t_1, \dots, t_3$  are elements of the translation vector  $\mathbf{t}$  [14]. This vector is easy to obtain. The coordinates of the laser tracker origin are  $\mathbf{O} = (0 \ 0 \ 0)^T$ . The translation between these positions is obtained by subtracting the origin from the position vector of the center point:

$$\mathbf{t} = \mathbf{s}_1 - \mathbf{O} = \mathbf{s}_1. \quad (5)$$

To determine the rotation matrix, it is used the fact that it is equivalent to basis vectors [26]. The condition is that the base must be orthonormal, which means that all

vectors are unit vectors and orthogonal to each other. Thus, the rotation matrix will consist of column vectors stacked one behind the other:

$$\mathbf{R} = (\mathbf{x} \quad \mathbf{y} \quad \mathbf{z}). \quad (6)$$

The plane in the form of (2) directly gives normal vector of the plane  $\mathbf{n} = (a \ b \ c)^T$  [34]. It is worth mentioning that plane has two possible normal vectors,  $\mathbf{n}$  and  $-\mathbf{n}$ . The one that points up from the plane (with positive  $z$  coordinate) is chosen. This vector is normalized. The Euclidean norm of the vector is  $\|\mathbf{n}\| = \sqrt{a^2 + b^2 + c^2}$ . This normalized vector is designated as basis vector  $\mathbf{z}$ :

$$\mathbf{z} = \begin{pmatrix} \frac{a}{\|\mathbf{n}\|} \\ \frac{b}{\|\mathbf{n}\|} \\ \frac{c}{\|\mathbf{n}\|} \end{pmatrix} \quad (7)$$

The second basis vector can be obtained easily as well. The centers of the two measured circles lie conveniently on the  $x$  axis of the table coordinate system. The Axis is defined as  $\mathbf{a} = \mathbf{s}_2 - \mathbf{s}_1$ , so after this vector is normalized, it can be declared a basis vector  $\mathbf{x}$ . Because normal vector  $\mathbf{z}$  is perpendicular to all vectors in plane  $p_1$ , it is perpendicular also to the vector  $\mathbf{x}$ , so the conditions for the basis vector are met.

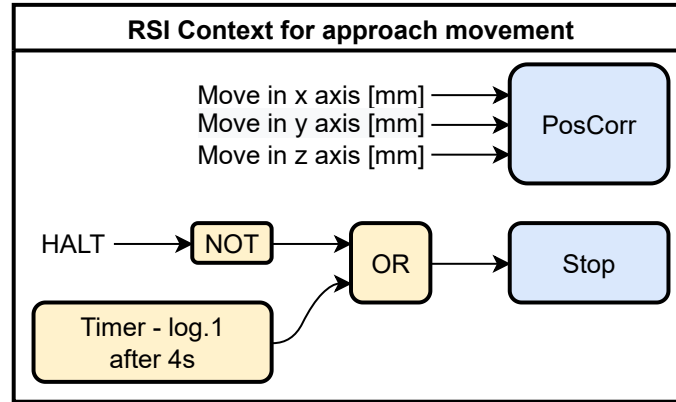
After the two basis vectors are known and a vector perpendicular to both of them is sought, the cross product of the vectors  $\mathbf{z}$  and  $\mathbf{x}$  can be used. The order of multiplication must not be confused to maintain the right-hand coordinate system:  $\hat{\mathbf{y}} = \mathbf{z} \times \mathbf{x}$ . After  $\hat{\mathbf{y}}$  is normalized, it can be declared a basis vector  $\mathbf{y}$ . So all the components of the rotation matrix  $\mathbf{R}$  are known, and it can be composed.

To get the transformation matrix from the laser tracker to the table's root, it has to be subtracted 259 mm from the element  $t_3$  in the matrix  $\mathbf{T}$ , which corresponds to translation in the  $z$  axis.

### 4.2.3 Robot programming

The robot has to go around the table and measure those three objects using T-Mac with the touch probe. The program is partially written in KRL and RSI. The RSI is useful when the robot is close to the target and is about to touch the table's surface. It can quickly respond to contact with the table and immediately stop the robot.

The robot first measures the plane, calculated from 12 measured points. All measured points are predefined and may be more or less random. The robot stops approximately 10 mm away from the surface where a single point is measured. Next, the function for approach movement is started; the inputs are three parameters that tell in which direction it should be moving and by how much. Inside this function are commands, which prepare, activate, and subsequently deactivate an RSI Context. The RSI Context is schematically described in Figure 4.3. Once it is turned on, it runs in a loop and slowly moves towards the surface till the probe tip hits the table; at this moment, the Tracker Pilot samples the point, and the robot stops.



**Figure 4.3.** Diagrammatic description of the RSI Context.

Inside the RSI Context is a function block **PosCorr**, which can relatively change the robot's position or rotation depending on the base currently in use. Even though it is possible to correct the rotation, it is unnecessary to correct it. Right before the approach movement, the probe is in sufficient rotation configuration.

The correction is executed in every cycle of the RSI Context that lasts only 12 ms, so this correction cannot be in millimeters, but in smaller units. The robot is not dynamic enough to move several mm in such a short period of time. Moreover, each coordinate can be corrected maximum of 25 mm for one lifecycle of the RSI Context. For these reasons, the correction is set to  $\pm 0.05$  mm when moving in a single direction and  $\pm 0.035$  mm for each coordinate when the robot moves in two directions simultaneously. This means that the robot moves for this short distance in every cycle of the RSI context.

The second essential function block in the RSI context is **Stop**; it stops the robot in a moment of logical "1" in the input [12]. The signal HALT from the Automation interface sends this trigger signal. But, it is logical "1" when the probe is steady-state and turns to logical "0" when the tip has touched [18]. So before the input to the function block, it is necessary to negate the HALT signal. If the Automation Interface does not send the HALT signal, but the probe has touched, the Stop could also be triggered from the **Timer**. The output turns to logical "1" after 4 s. This Timer is not reset in every RSI Context cycle, but it lasts as long as the RSI context is active. For this reason, there is **OR** logical operation right on the input of the Stop [12].

Once the plane is measured, the robot measures the circle in the middle of the table and the second circle. To determine the circle, 8 points in a single level are needed to measure. The probe will go slightly under the table surface in the hole, and it will gradually touch the measuring points, whereas the plane measured before is used as a constraining plane so that the measured circles will appear in the plane.

### 4.3 Transformation between the laser tracker and robot's root $T_r^l$

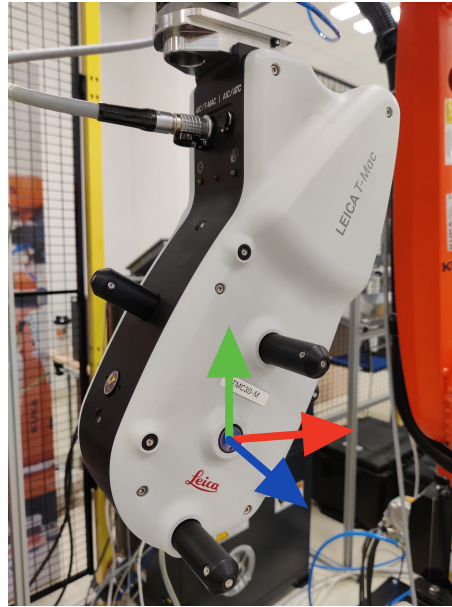
The main idea is to compose this transformation matrix with the transformation between the tracker and the T-Mac and subsequently the transformation between T-Mac and the robot's root. The tracker's measured position and rotation create the first transformation matrix. And the second transformation is obtained from the robot's forward kinematics. Several measurements are made and then evaluated by statistical

methods. Firstly, obtaining a single transformation is going to be described, and after that, how to combine multiple measurements into one result.

#### 4.3.1 Transformation between the laser tracker and the T-Mac

This method does not use the touch probe but the T-Mac alone. The probe must be dismounted before taking measurements. The T-Mac has four faces; there is a reflector on each face. On each reflector's surface in the middle is an origin of a coordinate system of that face. During measurement, *face3* is used, and its coordinate system is in Figure 4.4. Face, where the T-Mac cable is plugged in, is *face4*. On the opposite side of *face4* is *face2* and on the opposite side of *face3* is *face1*. Each side has its own coordinate system, with the origin at the center of the reflector on the surface of the T-Mac. The  $z$  axis direction always comes out of the reflector perpendicular to the surface of the T-Mac, and the negative  $y$  axis direction always points directly towards the touch probe adapter. The  $x$  axis is added so that it is a right-handed coordinate system [9].

By taking a measurement, the laser tracker provides 6DoF measurement. That is a position and rotation of T-Mac's coordinate system related to the tracker's coordinate system. This can be done in Tracker Pilot; the standard measurements are taken. In the future, the acquisition of measured data from the laser tracker according to chapter 3 will also be implemented in this solution. However, use of Tracker Pilot is convenient because measured data can be exported to a CSV file, and they are post-processed in Matlab.



**Figure 4.4.** T-Mac *face3* coordinate system.

A single measurement consists of 7 parameters, three coordinates of translation, and quaternions represents rotation. Those parameters are processed in a custom-made function  $create\_trafo(t_x, t_y, t_z, q_0, q_1, q_2, q_3)$ , where  $t_x$ ,  $t_y$ , and  $t_z$  are translations in each axis;  $q_0, q_1, q_2, q_3$  are rotation parameters. This function composes a transformation matrix  $\mathbf{T}$  according to (4). First, it is created as a unit matrix, and then the proper parameters are inserted into it. The rotation matrix  $\mathbf{R}$  is obtained from the quaternions using the same equations as in Section 3.3.5.

The matrix  $\mathbf{R}$  is inserted into the matrix  $\mathbf{T}$  to positions  $r_{1,1}, \dots, r_{3,3}$ . For making the matrix complete, it is placed  $t_x$  at position  $t_1$ ,  $t_y$  at position  $t_2$ , and  $t_z$  at position  $t_3$

[14]. This gives the transformation from the laser tracker to the T-Mac *face3* coordinate system, it is represented by a matrix  $\mathbf{T}_t^l$ , is obtained.

### 4.3.2 Transformation between the robot and the T-Mac

When the tracker is taking the measurement, the robot records its TCP position. The TCP coordinates system is the coordinates system of T-Mac. The transformation between flange and TCP was read from the digital twin. At this point, it is worth noting that Kuka uses Euler angles [11] to describe rotation:

$$\mathbf{R} = \mathbf{R}_z \cdot \mathbf{R}_y \cdot \mathbf{R}_x. \quad (8)$$

Where the individual matrices represent the rotation in  $\mathbb{R}^3$  around each axis. First around the  $z$  axis by the parameter  $r_z$ , then around the  $y$  axis by the parameter  $r_y$  and lastly around the  $x$  axis by the parameter  $r_x$ . This definition of Euler angles is also called *yaw-pitch-roll*, it is used for aircraft rotation [29]. Rotation matrices around individual axes [26]:

$$\mathbf{R}_z(r_z) = \begin{pmatrix} \cos(r_z) & -\sin(r_z) & 0 \\ \sin(r_z) & \cos(r_z) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad (9)$$

$$\mathbf{R}_y(r_y) = \begin{pmatrix} \cos(r_y) & 0 & \sin(r_y) \\ 0 & 1 & 0 \\ -\sin(r_y) & 0 & \cos(r_y) \end{pmatrix}, \quad (10)$$

$$\mathbf{R}_x(r_x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(r_x) & -\sin(r_x) \\ 0 & \sin(r_x) & \cos(r_x) \end{pmatrix}. \quad (11)$$

When the robot stops, the Stationary measurement in Tracker Pilot is taken [13], and right before starting to the next position, so that the robot's position is steady, the separate coordinates of the TCP are saved in a text file on the robot controller's hard drive. The robot variable `$POS_ACT_MES` is recorded. It is an *E6POS* data type that provides the measured position of TCP in the currently used base coordinate system, which is the robot root base. The individual coordinates can be accessed via dot notation. `$POS_ACT_MES.X`, `$POS_ACT_MES.Y`, and `$POS_ACT_MES.Z` indicates the position of TCP and `$POS_ACT_MES.A`, `$POS_ACT_MES.B`, and `$POS_ACT_MES.C` gives the rotations. *A* is Kuka's parameter for rotation around the  $z$  axis, *B* is a parameter of rotation around the  $y$  axis, and *C* is a parameter of rotation around the  $x$  axis [35].

The text file is opened after a first position is reached, a current position is always written on a new line, and it is closed and saved after all points are measured. It is saved in the robot's controller under `C:\KRC\ROBOTER\UserFiles`. The KRC4 controller runs on Windows operating system, which can be used to transfer files between the robot's controller and a PC. A new Administrator account is created in the controller with read and write rights. Then under the properties of the *UserFiles* file, it is selected that this folder can be shared with the new account, including read and write rights.

Once the measured values are written to the text file, the file can be accessed from the PC. It is necessary to open Windows File Explorer, and after "\\\" it has to be typed KRC4 controller's IP address. Then, it can be selected to log in to the newly created account [36] Login details can be remembered so that they do not have to be entered each time. Now, the text file with the recorded positions can be copied to the root folder of the Matlab project and imported to the Matlab script.



There are only 6 parameters from the robot describing the position and rotation of the T-Mac. Three for position  $(t_x, t_y, t_z)$  and three for rotation  $(r_z, r_y, r_x)$ . The transformation matrix  $\mathbf{T}$  is obtained from a custom-made function *create\_trafo\_euler* $(t_x, t_y, t_z, r_z, r_y, r_x)$ . Again, the identity matrix is filled with the rotation matrix  $\mathbf{R}$ , that is created according to (8). Then the translation parameters are inserted at positions  $t_1$ ,  $t_2$ , and  $t_3$ . The transformation matrix  $\mathbf{T}_t^l$  that represents the transformation from the robot's root coordinate system to the T-Mac *face3* is created.

### 4.3.3 Joining $\mathbf{T}_t^l$ and $\mathbf{T}_r^l$ together

The transformation matrix, which represents the transformation of the coordinate system from the laser tracker to the robot's root, can be easily obtained since we know the transformation between robot and the T-Mac and between the laser tracker and the T-Mac. It is obtained as follows:

$$\mathbf{T}_r^l = \mathbf{T}_t^l \cdot \mathbf{T}_t^r, \quad (12)$$

where  $\mathbf{T}_r^l$  is unknown but is acquired by inverting the matrix  $\mathbf{T}_t^r$ . In order to perform the inversion, we need to verify whether the matrix  $\mathbf{T}$  is invertible, that is if it has a nonzero determinant. Transformation matrices are usually invertible [14]. The resulting transformation matrix is obtained from the relation:

$$\mathbf{T}_r^l = \mathbf{T}_t^l \cdot (\mathbf{T}_t^r)^{-1}, \quad (13)$$

In order to better determine the transformation between the robot and the laser tracker, it is calculated from multiple points. A position generator was created in Matlab to create linear motions that go around the entire workspace with different rotations of the T-Mac. More specifically, 30 points are generated. T-Mac goes between positions

$$x \in \langle 72, 612 \rangle, \quad y \in \langle 551, 956 \rangle, \quad z \in \langle 778, 937 \rangle,$$

and rotations are within intervals

$$r_z \in \langle 61, 96 \rangle, \quad r_y \in \langle -39, 15 \rangle, \quad r_x \in \langle 42, 99 \rangle,$$

these TPC positions are in the robot's coordinate system. These positions are located just above the DKP, where the robot is required to be as accurate as possible. The rotation parameters are chosen in such a way that the optical connection of the laser beam between the laser tracker and the T-Mac is not interrupted. The content of the created text file with positions is then only copied into the KRL program.

Once the robot has passed all 30 positions, and it has recorded all positions, and the laser has also measured all positions; they are imported into a Matlab.  $\mathbf{R}$  is the matrix of points measured by the robot. In matrix  $\mathbf{L}$  are stored measured points by the laser tracker. For every 30 positions, the matrix  $\mathbf{T}_r^l$  is calculated.

From every  $\mathbf{T}_r^l$ , the individual parameters of translation and rotation are extracted using the custom-made function *parameters\_from\_trafo* $(\mathbf{T}_r^l)$  and stored in the corresponding row of matrix  $\mathbf{P}$ . This matrix is used to store the parameters of the transformation matrices. The translations are easy to extract since they are at positions  $t_1$ ,  $t_2$ , and  $t_3$ . Converting the rotation matrix to Euler angles is more complicated. It is discussed in section 4.3.4. It is worth mentioning that Euler angles have two solutions in the nonsingular case [37]. So far, the solution that has been chosen is the one that

was expected, which is not a very efficient approach. In the future, this will be converted to quaternions to avoid converting the rotation matrix to Euler angles as much as possible.

Once all 30  $\mathbf{T}_r^l$  matrices are calculated and their parameters stored in the matrix  $\mathbf{P}$ , these parameters can be evaluated. Each parameter is computed from its 30 instances by computing the mean value, that is, the best estimate of the expected value  $E(X)$  [38]. Thus, 6 mean values are calculated ( $\bar{t}_x, \bar{t}_y, \bar{t}_z, \bar{r}_z, \bar{r}_y, \bar{r}_x$ ). From these estimations of expected value, the resulting transformation matrix  $\mathbf{T}_r^l$  between the origin of the laser tracker and the robot's root coordinate system is computed. Again using function *create\_trafo\_euler*( $\bar{t}_x, \bar{t}_y, \bar{t}_z, \bar{r}_z, \bar{r}_y, \bar{r}_x$ ). This process is described in the following pseudocode:

```

R ← readmatrix() ▷ read measured data from the robot
L ← readmatrix() ▷ read measured data from the laser tracker
P ← zeros(30, 6) ▷ matrix of parameters from the resulting transformation matrix
for  $i = 1, \dots, 30$ 
     $\mathbf{T}_t^r \leftarrow \text{CreateTrafo}(\mathbf{R}[i, \text{all columns}])$ 
     $\mathbf{T}_t^l \leftarrow \text{CreateTrafoEuler}(\mathbf{L}[i, \text{all columns}])$ 
     $\mathbf{T}_r^l = \mathbf{T}_t^l \cdot \text{inv}(\mathbf{T}_t^r)$ 
     $\mathbf{P}[i, \text{all columns}] \leftarrow \text{ParametersFromTrafo}(\mathbf{T}_r^l)$ 
end for
m ← zeros(1, 6) ▷ vector of means of transformation matrix parameters
for  $i = 1, \dots, 6$ 
     $\mathbf{m} \leftarrow \text{mean}(\mathbf{P}[\text{all rows}, i])$ 
end for
 $\mathbf{T}_r^l = \leftarrow \text{CreateTrafo}(\mathbf{m})$ 

```

#### 4.3.4 Euler angles

In this section, the conversion of the rotation matrix to Euler angles is introduced. The rotation matrix, which was created using Equation (8), looks like this:

$$\mathbf{R} = \begin{pmatrix} \cos(r_y)\cos(r_z) & R_{1,2} & R_{1,3} \\ \sin(r_z)\cos(r_y) & R_{2,2} & R_{2,3} \\ -\sin(r_y) & \sin(r_x)\cos(r_y) & \cos(r_x)\cos(r_y) \end{pmatrix}, \quad (14)$$

$$R_{1,2} = \sin(r_x)\sin(r_y)\cos(r_z) - \sin(r_z)\cos(r_x),$$

$$R_{1,3} = \cos(r_x)\cos(r_z)\sin(r_y) + \sin(r_x)\sin(r_z),$$

$$R_{2,2} = \sin(r_x)\sin(r_y)\sin(r_z) + \cos(r_x)\cos(r_z),$$

$$R_{2,3} = \sin(r_y)\sin(r_z)\cos(r_x) - \sin(r_x)\cos(r_z).$$

The conversion back to Euler angles ( $r_z, r_y, r_x$ ) is done by using the parameters  $R_{1,1}, \dots, R_{3,3}$ . The case when  $R_{3,1} \neq \pm 1$  is presented first. Then the value of  $r_y$  is immediately obtained:

$$r_{y,1} = -\arcsin(R_{3,1}). \quad (15)$$

As discussed earlier, the function arcsine returns values only in interval  $\langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$ . So there are actually two solutions of  $r_y$ . The second one is shifted by  $\pi$ :

$$r_{y,2} = \pi - r_{y,1}. \quad (16)$$



If  $R_{3,2}$  is divided by  $R_{3,3}$ , and  $R_{2,1}$  is divided by  $R_{1,1}$ , the formula for  $\tan(r_x)$  and  $\tan(r_z)$  is found. Subsequently,  $\text{atan2}()$  is used instead of the arctangent to get the angles  $r_x$  and  $r_z$  :

$$\frac{R_{3,2}}{R_{3,3}} = \frac{\sin(r_x)\cos(r_y)}{\cos(r_x)\cos(r_y)} = \tan(r_x), \quad (17)$$

$$r_x = \text{atan2}\left(\frac{R_{3,2}}{\cos(r_y)}, \frac{R_{3,3}}{\cos(r_y)}\right), \quad (18)$$

$$\frac{R_{2,1}}{R_{1,1}} = \frac{\sin(r_z)\cos(r_y)}{\cos(r_z)\cos(r_y)} = \tan(r_z), \quad (19)$$

$$r_z = \text{atan2}\left(\frac{R_{2,1}}{\cos(r_y)}, \frac{R_{1,1}}{\cos(r_y)}\right). \quad (20)$$

First,  $r_{y,1}$  and then  $r_{y,2}$  can be substituted into equations (18) and (20). This will produce bolt solutions of  $r_{x,1}$ ,  $r_{x,2}$ ,  $r_{z,1}$ ,  $r_{z,2}$  [37].

The case in which  $R_{3,1} = \pm 1$  is the singularity called Gimbal lock ( $\arcsin(1) = 90^\circ$ ). So  $r_y$  is either  $90^\circ$  or  $-90^\circ$ . It means that  $r_z$  and  $r_y$  are causing the same rotation, so one parameter of them is redundant [32]. Thus,  $r_z$  can be set to 0.

When  $r_z = 0$ , and  $R_{3,1} = -1$ , then  $r_y = 90^\circ$ , and then  $R_{1,2}$ ,  $R_{1,3}$  can be simplified:

$$R_{1,2} = \sin(r_x)\cos(r_z) - \cos(r_x)\sin(r_z) = \sin(r_x - r_z) = \sin(r_x), \quad (21)$$

$$R_{1,3} = \cos(r_x)\cos(r_z) + \sin(r_x)\sin(r_z) = \cos(r_x - r_z) = \cos(r_x), \quad (22)$$

Those can be divided between each other, and using  $\text{atan2}()$ , the  $r_x$  is obtained:

$$r_x = \text{atan2}(R_{1,2}, R_{1,3}). \quad (23)$$

The second possibility of singularity is when  $R_{3,1} = 1$ , then  $r_y = -90^\circ$ ,  $r_z$  can be again set to 0, then  $R_{1,2}$ ,  $R_{1,3}$  become

$$R_{1,2} = -(\sin(r_x)\sin(r_z) + \cos(r_x)\cos(r_z)) = -\sin(r_x + r_z) = -\sin(r_x), \quad (24)$$

$$R_{1,3} = -(\cos(r_x)\cos(r_z) - \sin(r_x)\sin(r_z)) = -\cos(r_x + r_z) = -\cos(r_x), \quad (25)$$

Those can again be divided between each other, and using  $\text{atan2}()$ , the  $r_x$  is obtained [37]:

$$r_x = \text{atan2}(-R_{1,2}, -R_{1,3}). \quad (26)$$

# Chapter 5

## Inspection of the manufactured part

The idea of this application is that the robot equipped with the T-Mac and touch probe will be at the output of the production line and will check parameters of manufactured part. Then it will evaluate if the product has been manufactured within the specified tolerances. Measured data will then be compared with the reference CAD model.

At our workstation, the printed part is measured. After the part is printed, the extruder is swapped with the T-Mac with the touch probe. In this task, the robot serves only as a simple tool carrier; no precise positioning is required. An example of the object inspection process is shown in Figure B.4. Together with the laser tracker, T-Mac works similarly to CMM. This solution is less costly, as the user does not have to buy an expensive CMM but can use an industrial robot, which costs significantly less [1]. In addition, there is a greater possibility of automating the process. The output of production lines can be different types of products, and the robot can easily inspect them.

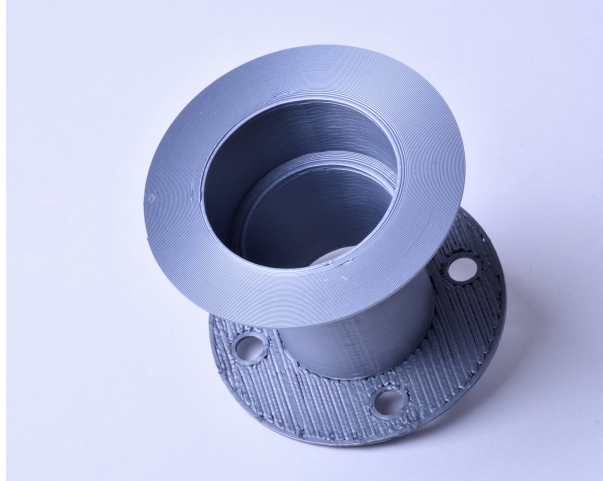
### 5.1 Measurement of the printed part using Polyworks Inspector software

#### 5.1.1 Printed Part

An example of a part that is printed on the workstation is shown in Figure 5.1. In order to print it, the positioner must rotate the first axis to a perpendicular position. Then it is rotated with the second axis all around, and the printing continues. The robot moves only on the  $z$  axis. This demonstrates the advantages of multiple degrees of freedom as opposed to conventional 3D printers<sup>1</sup> with 3DoF (gantry construction). This product would have to be printed with supports on them.

The base of the printed part is a 2 mm high ring (annulus) with a diameter of the inner circle of 50 mm and an outer circle of 100 mm. There are four holes in it with diameters of 10 mm. On the inner circle is a printed cylinder with a diameter of 50 mm. Subsequently, the positioner is rotated in the first axis of 90°, and on top of the cylinder are printed circular layers that create another annulus. This time it has a thickness of 1.5 mm, the width of layer. The used extruder nozzle has a diameter of 1 mm, and the printed material is pressed, so the layer becomes wider. Overall, the annulus has an outer diameter of 63 mm. After this, the first axis goes returns back to the default position. At its edge is printed another cylinder of a height of 30 mm. The positioner is then turned back to the perpendicular position. And the last annulus is printed on the edge of the cylinder. Its outer diameter is 100 mm.

<sup>1</sup> <https://www.prusa3d.com/cs/>



**Figure 5.1.** An example of a printed product that cannot be printed on a conventional 3DoF printer without support, but can be printed at this workstation.

### ■ 5.1.2 Polyworks Inspector program

In Polyworks Inspector, there can be run the custom-made program for so-called **Macro Script editor**. It supports automatic execution of measurement instructions and subsequent data evaluation [16]. There are three sets of instructions. First, instructions specific only to the Leica laser trackers are used to connect to the laser tracker or control the robot [17]. The second set of instructions is common to the entire Polyworks Inspector; there are instructions that provide commands for measurement and evaluation. For example, instruction that starts the plane feature measurement. The last are basic commands common to other programming languages. These instructions can handle, for example, working with variables, loops, and conditions evaluations.

Subsequent creation of the program in Macro Script editor is relatively easy. The best option is that the user first performs the measurement process once manually. In Polyworks Inspector, there is a **Command History window** where the executed commands are recorded in real-time. The user can then see what commands are used to execute the desired operation [16]. Thus, he or she finds out how to assemble the measurement program. The individual parts of the Macro Script program are now introduced:

- **Establishing communication**—The IP address and the IP address of the Automation Interface are needed to establish communication with the laser tracker and Polyworks Inspector. Then it is specified which laser tracker and which target are connected. The specifics are that the T-Mac with a touch probe is used. It is then verified if the laser tracker communicates with the Polyworks Inspector, subsequently, if the T-Mac is indeed detected as the target. Then it is verified whether there is no other problem in establishing communication [17]. Finally, the CHECKPROGACTIVEBIT command tests if the robot's program is running, so the robot is ready to move. This signal is mapped to DIG IN 1(Figure 3.3).
- **Feature measurement**—This section measures a plane and two circles in sequence. When the number of points a feature consists of is specified, Polyworks Inspector knows when the feature acquisition should be complete. The ROBOT CONFIRMEDGO (DIG OUT 1) and ROBOT WAITFORPOSREACHED (DIG IN 2) commands are particularly useful. With the first one, the Polyworks Inspector sends a signal to the robot that it can move to the next position. And with the second one serves as a wait; it waits for the robot to get to the specified position, and only then

can other commands be executed. In Figure 5.2, these commands are used in *for* loop.

- **Measured data evaluation**—Once features are measured, some other features may be necessary to create. For example, from a circle, can be extracted its center point. Next, in this part can be performed object alignment [16]. In Figure 5.2 can be seen used instructions for obtaining the circle and center point from it.

```
#circle in center
TREEVIEW OBJECT SELECT NONE
FEATURE PRIMITIVE CIRCLE PROBE2 ( , "circle_A", )
FEATURE PRIMITIVE CURVE_BASED OPTIONS PROBE USE CONSTRAINING PLANE ( "On" )
FEATURE PRIMITIVE CURVE_BASED OPTIONS PROBE CONSTRAINING PLANE PLANE_FEATURE ( "Plane_A" )
FEATURE PRIMITIVE CIRCLE OPTIONS PROBE MANUAL POINT ACQUISITION STANDARD STANDARD USE FIXED_NB_POINTS ( "On" )
FEATURE PRIMITIVE CIRCLE OPTIONS PROBE MANUAL POINT ACQUISITION STANDARD STANDARD FIXED_NB_POINTS ( 7 )
SET temp 1
WHILE $temp <= 7
  ++ temp
  ROBOT CONFIRMEDGO
  ROBOT WAITFORPOSREACHED
ENDWHILE

#point from circle
TREEVIEW OBJECT SELECT NONE
TREEVIEW OBJECT SELECT ("circle_A", "On")
FEATURE PRIMITIVE POINT FROM_CIRCLES ( "Point_A", )
```

**Figure 5.2.** Screenshot from Macro Script editor showing instructions of measuring a circle followed by extracting its center.

### 5.1.3 Workflow for automated inspection of the printed part

This inspection is performed in Polyworks Inspector using a Macro Script program. A new workspace in Polyworks Inspector must be opened, and then the Macro Script program will do all the work. The Macro Script program cooperates with a robot's program in KRL. The workflow diagram is in Figure 5.3.

Before the program can be started, the robot must send the signal that the robot's program is running. Immediately after starting the Macro Script program, communication with the laser tracker is established. After this, the CAD model of the printed part is imported. It is stored in the root folder of the Polyworks project. The problem is that it is imported into the origin of the world coordinate system that corresponds to the origin of the laser tracker's coordinate system. In order to compare the CAD model with actual measured parameters, they must be in the same coordinate system [16]. A coordinate system identical to the positioner's surface coordinate system is created (Figure 4.2).

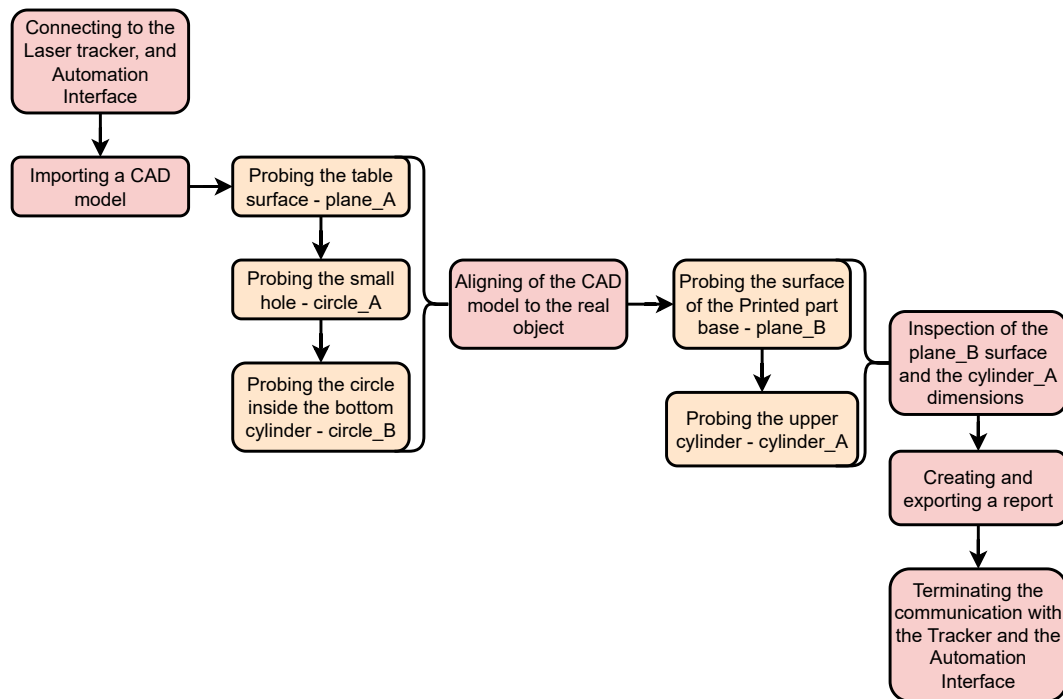
The Plane, Axis and Center point method is used again but this time is done automatically in Polyworks Inspector. First, the plane is obtained—*plane\_A*. It is determined by measuring 12 points. This plane is not exactly the positioner's surface, but a pad of width of 1 mm is attached to it, on which it is printed. A small circle—*circle\_A* is then measured, the center of which lies in the positive direction of the x axis. The probe then goes inside the lower cylinder, where another circle—*circle\_B* is measured at a single level. Both circles have a constraining plane *plane\_A* in order to both center points lie on the pad. When aligning is performed, as the plane is used *plane\_A*, as the axis is used the line between two circle's center points, and as the center point is used the center of the *circle\_B*.

ROBOT CONFIRMEDGO and ROBOT WAITFORPOSREACHED are again used for synchronization between the Macro Script program and the robot. This pair of signals are not used for every robot move as the title suggests, but ROBOT CONFIRMEDGO is sent when the robot starts one particular geometric object. And ROBOT WAITFORPOSREACHED is received only after finishing measuring a whole feature. This use is suitable for waiting for the Polyworks Inspector; it takes some time

to process the measured data and prepare the measurement of the next object. Thus, the robot cannot start measuring other points that would not be recorded [16].

The parameters of the printed part are then evaluated. Basically, any part of the product can be examined, as well as the parameters between the individual parts. For automated inspection testing purposes, the base thickness at each point is examined, then the diameter and the direction vector of the top cylinder is examined as well. Again, the 12 points are used to determine the base of the part. To determine the cylinder, measuring three circles in three different levels is necessary. Each circle consists of 8 points, making the whole cylinder of 24 [39].

The distance between each point of the base and the pad plane is measured. The optimal distance for each point is 2 mm. The minimal, maximal, mean, and standard deviation can be determined. It can be chosen how much tolerance is allowed and thus evaluate whether the base thickness of the product is printed correctly. It is no problem to add additional points to the inspection for greater measurement accuracy.



**Figure 5.3.** Scheme of the workflow of the Macro Script program for automated inspection.

#### 5.1.4 Robot programming

Since the robot moves in the base of the positioner, to which the manufactured product is always aligned, the robot's movements can be easily planned. So far, no postprocessor for generating robot movements has not been made. Robot program is written in KRL manually. Since each printed object is slightly different, the robot moves the last section, about 10 mm, in an incremental motion. This incremental movement has already been described in Section 4.2.3. Then the program is completed by transitions between the measured points. The incorporation of synchronization signals for the program in Polyworks Inspector must not be forgotten. At the beginning of the program,  $\$OUT[3]$ —robot has finished movement, and  $\$OUT[2]$ —robot program is active, are set to *True*. The robot waits until the  $\$IN[1]$  variable is *True* to start measuring the feature. Immediately afterward, it drops  $\$OUT[2]$  to *False* and returns to *True* when

it has passed all the points to get a particular feature. Then it waits for  $IN[1]$  again [35].

The T-Mac with the touch probe must be tilted during the measurement; otherwise, it will crash into the overhanging parts. If the probe measures an object from the outside, the surroundings of the object to be measured can be divided into 8 segments, for which tilts are defined according to the  $x$  and  $y$  axes. When the measured point is in the proximity of the positive  $x$  axis, it is sufficient to change the parameter defining the rotation around the tool's  $y$  axis in the negative direction; in the case of Kuka, this is parameter  $B$ . For points in the neighborhood of negative  $x$  values, these are positive values of the tilt  $B$ . If the point is in the region of the  $y$  axis, the probe rotates about the  $x$  axis, as indicated by parameter  $C$ . When the measured points are in the neighborhood of positive  $y$  axis, the probe rotates in the positive values of parameter  $C$  and vice versa. The remaining 4 segments combine both the  $x$  and  $y$  coordinates and thus combine the appropriate tilt. The specific tilt values can vary depending on how much the surface overhangs the measured point.

If the inside surface of an object is measured, such as the middle circle during alignment, the parameters  $B$  and  $C$  are the negated parameters  $B$  and  $C$  that correspond to them for measurements from outside of the object. Using these conditions, trajectories to measure almost any feature on the surface of an object can be generated. The problem are places that are too narrow or overhanging; to measure them, the probe would have to approach from the bottom up. In the case of measuring the printed product, the kinematics of the robot would not allow it.

### 5.1.5 Evaluation of the inspection

The purpose of the bachelor thesis was to design a process for inspecting a printed part, not to deal with specific parameters to determine the correctness of a printed part. As already written, when measuring in Polyworks Inspector, it can be chosen from many parameters to investigate. It is, therefore, suitable for in-depth evaluation of measurement data, but not so suitable for automating the process [16]. For example, in the case of a cylinder, concentricity or cylindricity might be interesting to investigate. And for a plane, for example, flatness or parallelism [39]. When inspecting without any metrology software, these parameters will now not be evaluated.

In Figure 5.4 are captured results of the inspection. The interior of the upper cylinder and the thickness of the lower ring at 12 points was examined. The thickness of the lower ring is represented by the  $z$  coordinates.

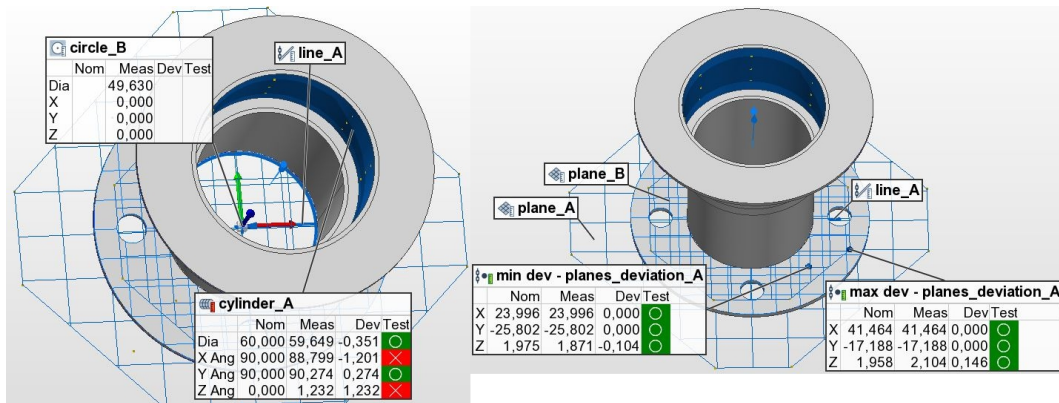
When examining the area, a colormap was created from all the measured points to represent the distance of the measured points from the pad. For 12 points, the colormap is sparse, but for more points, it would be an excellent graphical representation of the thickness. This colormap is then used to find the point with the smallest and largest deviation of the  $z$  axis coordinate from the nominal value [16]. It is good to note that not all nominal values have a  $z$  component of exactly 2 mm either, which is probably due to the inaccuracy of the pad (*plane\_A*) measurements. The measurement shows that the point where the thickness is the greatest from the defined value is 0.146 mm thicker. The point with the smallest layer thickness is found to have a layer is by -0.104 mm thicker than defined. Interestingly, these two points are close together. This may be due to the probe falling into the gap between the two extruder nozzle's paths when measuring the least thick point.

The cylinder consists of three circles in three planes with a distance of 9 mm between each other. Each circle consists of 8 points. Three points are needed to make a circle,



and only two circles would be enough to make a cylinder [39]. More points were added to improve the accuracy of the parameters of the resulting cylinder. The nominal diameter of the inside of the upper cylinder is 60 mm, and the actual diameter is 59.649 mm, so it was printed with a deviation of -0.351 mm. This satisfies the defined tolerance of  $\pm 0.5$  mm.

For the cylinder, the deviation of the direction vector from the defined one is also evaluated. Ideally, it should be aligned with the z axis. That is, it should have an angle between the x and y basis vectors of  $90^\circ$  and from the z basis vector of  $0^\circ$ , which it has not since the cylinder is printed on a ring that is not always perfectly parallel to the bottom ring on the base. Tolerances were set to  $\pm 1^\circ$ , which was only met for the angle between the y-axis and the feature's direction angle [16].



**Figure 5.4.** Polyworks Inspector environment after completed inspection. The figure shows the features needed for alignment, the coordinate system of the printed part, and the results of the measured features.

## 5.2 Measurement of the printed part without metrology software

Metrology software Polyworks Inspector provides a large number of functions that are very useful for detailed inspection, but these functions can be redundant for simple output tolerance checking. Since one of the uses of a T-Mac with a touch probe can be output inspection in a production line, it is more convenient to automatically evaluate the measured data in, for example, a Matlab function. Polyworks Inspector slows down the measurements. It takes some time to prepare the feature measurements and then evaluate the measured data [16]. Secondly, for the user, there is no need to buy an expensive Polyworks license for these simple measurements.

### 5.2.1 Acquisition of measured data

As this task was created simultaneously with the laser tracker data conversion task, it was easier to perform the measurements in Tracker Pilot and export the measured data from there. The Tracker Pilot only provides the data recorded by the laser tracker, it does not evaluate the measured data in any way. The trigger for a measurement acquisition is selected for the touch probe so that the point is recorded when the probe touches the surface [13]. When it will be measured without Tracker Pilot but using Real Time module, trigger option can be specified as one of the Arguments in the Command Request, as described in Section 3.2.3.

The robot's program is the same as for the printed part measurement with the Polyworks Inspector. The control signals that are synchronizing the automatic inspection between the Polyworks Inspector and the robot's moves ( $\$IN[1]$ ,  $\$OUT[2]$ ,  $\$OUT[3]$  variables) are omitted. Also omitted are the robot paths that provide the measurements of the two circles ( $circle\_A$ ,  $circle\_B$ ) needed to align the CAD model to the actual object. The CAD model is not being used here, so the alignment does not matter. The measured data are in the coordinate system of the laser tracker and not in the coordinate system of the positioner. It does not matter that the measured objects are in the coordinate system of the laser tracker. The thickness of the printed layer and the diameter of the cylinder will not change.

The robot while measuring the features touches the pad on 12 points to obtain the plane of the pad, then 12 points on the surface of the bottom ring are measured for getting the thickness of the layer. Lastly, 8 points are measured in two levels inside the top cylinder for obtaining cylinder's axis, and diameter. In this case, only two circles are used to determine the cylinder [39]. So the measurement total consists of 40 points which are exported from the Tracker Pilot.

### 5.2.2 Inspection of the thickness of the lower ring

For this inspection, the first 24 measurement points are imported into Matlab, and the first 12 are used to create the plane of the pad, on which the object is printed. These points are interpolated into the plane using linear regression. The remaining 12 points will then be examined for their distance from the plane. Equation of pad's plane  $\pi$  is

$$\pi: ax + by + cz + d = 0, \quad (1)$$

where  $x, y, z$  are variables and  $a, b, c, d$  are coefficients of the plane [34]. The variable  $z$  will be isolated:

$$\pi: z = -\frac{a}{c}x - \frac{b}{c}y - \frac{d}{c}, \quad (2)$$

new plane's coefficients are created, then the equation of  $\pi$  is

$$\pi: z = -\hat{a}x - \hat{b}y - \hat{d}. \quad (3)$$

If those 12 measured points substituted for variables  $x, y, z$ , it produces a system of 12 equations with 3 unknowns:

$$\begin{aligned} z_1 &= -\hat{a}x_1 - \hat{b}y_1 - \hat{d} \\ z_2 &= -\hat{a}x_2 - \hat{b}y_2 - \hat{d} \\ &\vdots \\ z_{12} &= -\hat{a}x_{12} - \hat{b}y_{12} - \hat{d} \end{aligned}$$

This system can be rewritten into a matrix form  $\mathbf{Ax} = \mathbf{z}$ , where

$$\mathbf{A} = \begin{pmatrix} -x_1 & -y_1 & -1 \\ -x_2 & -y_2 & -1 \\ \vdots & \vdots & \vdots \\ -x_{12} & -y_{12} & -1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} \hat{a} \\ \hat{b} \\ \hat{d} \end{pmatrix}, \quad \mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_{12} \end{pmatrix}.$$

Then the vector  $z$  can be transferred to the other side of the equation:

$$\mathbf{Ax} - \mathbf{z} = \mathbf{0}. \quad (4)$$



This is an overdetermined system, which can be approximated. It is an optimization problem of minimizing the norm of deviations on the left-hand side of equation (4); the problem does not change if it is minimized the square of this norm [40]:

$$\min \|\mathbf{Ax} - \mathbf{z}\|^2 \quad (5)$$

This task is easily solved in Matlab using the a command  $\mathbf{x} = \mathbf{A} \setminus \mathbf{z}$ . This provides the  $\hat{a}$ ,  $\hat{b}$ ,  $\hat{c}$  parameters of the plane  $\pi$  from the equation (3).

Once the equation of the pad's plane has been obtained, the printed layer thicknesses can be evaluated sequentially at 12 points. For each point, the distance from the plane  $\pi$  is calculated. First, the plane  $\pi$  is converted back to the format from (1):

$$\pi: \hat{a}x + \hat{b}y + z + \hat{d} = 0. \quad (6)$$

This form of the equation gives the normal vector  $\mathbf{n}$  of the plane [34]. The normal vector is  $\mathbf{n} = (\hat{a} \ \hat{b} \ 1)^T$ . For later calculations, it is useful to normalize the vector  $\mathbf{n}$ . As it has been done in section 4.2.2.

In the matrix  $\mathbf{P}$  are stored the measured points. So the rows contains points  $\mathbf{p}_i = (x_i \ y_i \ z_i)^T$ , where  $i = 1, \dots, 12$ . Then, any point  $\mathbf{p}_0$  that lies in the plane  $\pi$  has to be found. Subtracting point  $\mathbf{p}_0$  from point  $\mathbf{p}_i$  gives a vector  $\mathbf{q}_i$ . The distance of point  $\mathbf{p}_i$  from the plane  $\pi$  is given as an Euclidean norm of the vector  $\mathbf{q}_i$  projected onto a linear subspace created by the unit normal vector  $\mathbf{n}$  [41].

The point  $\mathbf{p}_0$  can be any point that satisfies equation (6), so it can be chosen a trivial variant

$$\mathbf{p}_0 = (0 \ 0 \ -\hat{d})^T.$$

In general, the formula for the projection of a vector  $\mathbf{u} \in \mathbb{R}^3$  onto a vector  $\mathbf{v} \in \mathbb{R}^3$  is

$$\text{proj}_{\mathbf{v}} \mathbf{u} = \left( \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{v}\|^2} \right) \mathbf{v}, \quad (7)$$

where the  $\mathbf{u} \cdot \mathbf{v}$  denotes the dot product of vectors. When these facts are combined, the definition of the shortest distance  $d_i$  of the point  $p_i$  from the plane  $\pi$  can be found:

$$d_i = \left\| \text{proj}_{\mathbf{n}} \mathbf{q}_i \right\| = \left\| \left( \frac{\mathbf{q}_i \cdot \mathbf{n}}{\|\mathbf{n}\|^2} \right) \mathbf{n} \right\|. \quad (8)$$

The first simplification of equation (8) is that  $\|\mathbf{n}\| = 1$ . So the equation simplifies to the form

$$d_i = \left\| (\mathbf{q}_i \cdot \mathbf{n}) \mathbf{n} \right\|. \quad (9)$$

Next, there is a fact that  $\|x\mathbf{v}\| = x\|\mathbf{v}\|$ , where  $x \in \mathbb{R}$ . Since the dot product  $\mathbf{q}_i \cdot \mathbf{n}$  results in a scalar, this rule can be used [41]. Using again the fact that  $\|\mathbf{n}\| = 1$ , equation simplifies to the form

$$d_i = (\mathbf{q}_i \cdot \mathbf{n}) \|\mathbf{n}\| = \mathbf{q}_i \cdot \mathbf{n}. \quad (10)$$

If the points  $\mathbf{p}_0$  and  $\mathbf{p}_i$  are returned to the vector  $\mathbf{q}_i$ , a simplified formula for the distance of the point  $\mathbf{p}_i$  from the plane  $\pi$  is found:

$$d_i = (\mathbf{p}_i - \mathbf{p}_0) \cdot \mathbf{n}, \quad (11)$$

into which the 12 measured points can be easily substituted in Matlab. This produces 12 distances that determines the thickness of the printed layer at each point  $\mathbf{p}_i$ . From these distances, it can be calculated the minimum and maximum deviation just as in Polyworks Inspector.

### 5.2.3 Cylinder inspection

Just as the bottom ring could be inspected, the parameters of the top cylinder can be inspected without using Polyworks Inspector. The cylinder is made of two circles, each of 8 points. Again, they are measured in the coordinate system of the laser tracker, but in this case, unfortunately inaccuracy of the positioner tilt is mixed together with the inaccuracy of the cylinder's center axis. This problem is easily solved by transforming the measured points into the coordinate system of the positioner, as described in Section 4.2.2. For now, the alignment is not incorporated. It is relied on a mastering of both axes of the positioner, which was performed exactly as it is instructed by Kuka [11]. So, when both axes are at zero position, it is assumed that the error in the positioner position is much smaller than the misalignment of the center axis of the printed cylinder.

The cylindrical surface  $\omega$  in  $\mathbb{R}^3$  can be defined by its radius  $r_\omega$  and a direction vector  $\mathbf{u}$  (center axis). Since each circle is measured in constant  $z$  coordinates, the center of each circle will be at its center of mass. From these two points, the vector  $\mathbf{u}$  is easily obtained. Fortunately, an equation of circles are not needed; it is more tricky to parameterize circle in  $\mathbb{R}^3$ . The radius  $(r_1, r_2)$  of each circle is calculated as the expected value (mean value) of the distances of the measured points from the center of the circle. From these two radiuses is calculated the diameter of the cylinder.

The last 16 measured points are loaded into Matlab. It is divided into two matrices,  $\mathbf{P}_1$  and  $\mathbf{P}_2$ , according to the belonging of each circle. Their rows contain the coordinates of the measured points  $(x, y, z)$ . The center of mass  $(\mathbf{t}_1, \mathbf{t}_2)$  of the two circles is easily calculated; for each component, the mean of the points of the respective component is calculated [38]:

$$t_x = \frac{P_{1,x} + \dots + P_{8,x}}{8}, \quad t_y = \frac{P_{1,y} + \dots + P_{8,y}}{8}, \quad t_z = \frac{P_{1,z} + \dots + P_{8,z}}{8}, \quad (12)$$

where  $P_{1,x}$  denotes the first row of the matrix  $\mathbf{P}$  and its first component, which corresponds to the  $x$  coordinate. A direction vector is created using the points  $\mathbf{t}_2$  and  $\mathbf{t}_1$ :

$$\mathbf{u} = \mathbf{t}_2 - \mathbf{t}_1. \quad (13)$$

Ideally, after normalization of the vector  $\mathbf{u}$ , it should be  $\mathbf{u} = (0 \ 0 \ 1)^T$ . That would mean it was printed perpendicular to the DKP. In real situation, however, the components  $x$ , and  $y$  are not 0. The deviation of the vector  $\mathbf{u}$  from the central axis can be defined as three angles between the vector  $\mathbf{u}$  and vectors  $\mathbf{v}_x = (1 \ 0 \ 0)^T$ ,  $\mathbf{v}_y = (0 \ 1 \ 0)^T$ , and  $\mathbf{v}_z = (0 \ 0 \ 1)^T$ . The angle  $\varphi$  between vector  $\mathbf{u}$  and each vector  $\mathbf{v}$  can be derived from the relation [41]

$$\cos(\varphi) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \rightarrow \varphi = \arccos^{-1} \left( \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|} \right)$$

This is done for all three vectors  $\mathbf{v}$  resulting in angles  $\varphi_x, \varphi_y, \varphi_z$ . The results are then just converted from radians to degrees. The angle  $\varphi_z$  is already the deviation being sought. The  $\varphi_x$  and  $\varphi_y$  angles must be subtracted by  $90^\circ$  to obtain those deviations.

Next, using the points in the matrices  $\mathbf{P}_1$  and  $\mathbf{P}_2$  and the centers of the circles  $\mathbf{t}_1$  and  $\mathbf{t}_2$  can be found the radiuses  $r_1, r_2$  of both circles. Each radius  $r$  is calculated as the average of the distances of the points from  $\mathbf{P}$  to the center  $\mathbf{t}$ . The distance is calculated as the norm of the vector  $\mathbf{t} - \mathbf{P}$ . The circle's radius is

$$r = \frac{\|\mathbf{t} - \mathbf{P}_1^T\| + \dots + \|\mathbf{t} - \mathbf{P}_8^T\|}{8}, \quad (14)$$

where  $\mathbf{P}_i$  denotes the  $i$ th row of the matrix  $\mathbf{P}$ ,  $i = 1, \dots, 8$ . When  $r_1, r_2$  are calculated, the diameter  $d_\omega$  of the cylinder is obtained as the double the radius  $r_\omega$ , which is obtained from the average of  $r_1$ , and  $r_2$ :

$$d_\omega = 2 \frac{r_1 + r_2}{2} = r_1 + r_2. \quad (15)$$

## Chapter 6

# Comparison of the methods presented in Chapters 5 and 6 with metrology software

### 6.1 Comparison of calibration methods

This section compares methods for determining the transformation between the laser tracker's coordinate system and the robot's root coordinate system. The first method is the one presented in section 4.3; this method is not using any metrology software. The second method is using Robodyn, it is a software primarily designed for robot calibration, but as a by-product, it produces the transformation matrix between the laser tracker and the robot's root. It also finds the transformation between the flange and the TCP, which is also useful for accurate measurements in the future [15].

Since the main focus of this chapter is to compare the developed method with a method using Robodyn, the transformation matrix between the coordinate system of the positioner and the coordinate system of the laser tracker will be omitted. This is because Robodyn is mainly designed for calibration of open kinematic 6DoF chains and not for two-axis external kinematics [15]. Therefore, the transformation matrix between the positioner and the laser tracker was not determined using Robodyn.

#### 6.1.1 Workflow description in Robodyn

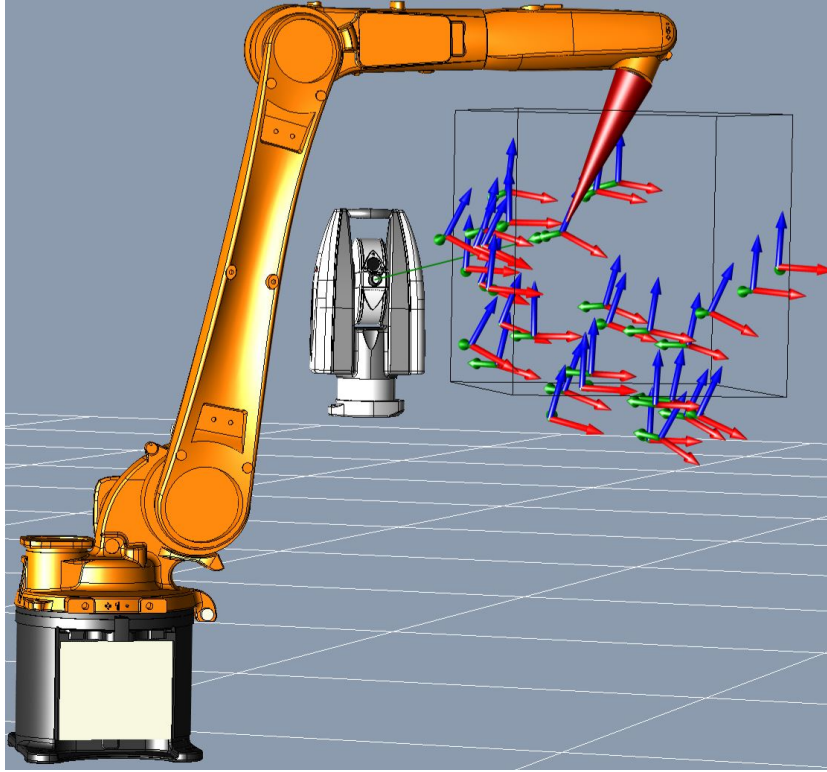
Robodyn is software for calibrating industrial robots from different manufacturers. It only needs to know the DH parameters [14] of the robot and its controller. The data for the definition of the DH parameters of the Kuka KR 1620 robot can be found in its datasheet [2]. In addition to the robot specification, it is essential to choose the correct measuring device (Leica Absolute Tracker AT960) and its target (T-Mac *face3* without the touch probe). The target does not have to be a T-Mac, it can be an ordinary RRR 1.5'' reflector. CAD models of these devices can be imported into Robodyn, the T-Mac model is not needed, it is visualized by its coordinate system (a cone in Figure 6.1). Robodyn works on the principle that the robot passes through several predefined positions, at which the tool rotates differently, and from these, it finds the transformation between the robot and the measuring device [15].

Before the calibration can start, the approximate values of the transformations between the flange and the TCP and between the robot and the laser tracker must be set. This is because Robodyn generates the robot trajectories with respect to the laser tracker position so that there is an unbroken optical link between the laser tracker and the target. Before running the calibration, it is a good idea first to run a simulation to see if there are no collisions in some of the motions.

The calibration points are generated by Robodyn itself depending on the selected workspace; this can be scaled and moved in space at the user's choice. It can be above the DKP, so the workspace is approximately the same as in the method using no metrology software. Once the points are generated, they have to be manually checked, it often happens that the robot configuration changes, which is not ideal because on the robot

are mounted various cables and extruder accessories which could be damaged when the robot is moved to a different configuration. If the robot is in a non-conforming configuration at a given point, the configuration can be changed manually, if Robodyn does not find a conforming configuration, the point can be deleted and a new one generated [15]. Figure 6.1 shows the prepared workspace with the generated measurement points.

Robodyn must connect to the robot controller and the laser tracker before the calibration can start. The robot paths are streamed to the controller using the EthernetKRL package [42]. Hexagon provides both the KRL program and the SPS program for it. So before connecting to the robot in Robodyn, both the SPS program and the KRL program must be running. Then the calibration is started in Robodyn. When the robot has passed all the points, the calibration report is generated.



**Figure 6.1.** Screenshot from Robodyn showing the robot, the laser tracker, generated points, and a red cone showing the target.

### 6.1.2 Comparison of both transformation matrices

Now, both transformation matrices  $\mathbf{T}$  from the robot coordinate system to the laser tracker coordinate system can be compared. Its components can be extracted so that they can be compared with each other.

■ Transformation matrix obtained using Tracker Pilot and Matlab:

$$\mathbf{T}_r^l = \begin{pmatrix} -0.8704 & -0.4924 & 0.0008 & 2448.53 \\ 0.4924 & -0.8704 & 0.0016 & -967.703 \\ -0.0001 & 0.0017 & 1 & 780.573 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (1)$$

$$t_x = 2448.53 \text{ mm}, t_y = -967.703 \text{ mm}, t_z = 780.573 \text{ mm},$$

$$r_z = -0.089^\circ, r_y = 0.043^\circ, r_x = 150.503^\circ.$$

The transformation from the flange to the tool that was used for the measurements was obtained from the NX software<sup>1</sup>, where it the digital twin of the workstation. Its parameters are:

$$x = -7 \text{ mm}, y = 44.7 \text{ mm}, z = 306.5 \text{ mm}, r_z = 180^\circ, r_y = 0^\circ, r_x = 180^\circ.$$

Due to manufacturing tolerances, it may not be exactly as manufacturers of components claims. This transformation between the flange and the TCP can be corrected depending on what Robodyn determines.

■ Transformation matrix obtained by Robodyn:

$$\mathbf{T}_r^I = \begin{pmatrix} -0.8656 & -0.5008 & 0.0007 & 2448.144 \\ 0.5008 & -0.8656 & 0.0016 & -967.8878 \\ -0.0002 & 0.0018 & 1 & 780.614 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad (2)$$

$$t_x = 2448.144 \text{ mm}, t_y = -967.898 \text{ mm}, t_z = 780.614 \text{ mm},$$

$$r_z = 0.101^\circ, r_y = 0.136^\circ, r_x = 149.947^\circ.$$

The transformation parameters provided by Robodyn between the robot's flange and its tool are

$$x = -7.068 \text{ mm}, y = 44.291 \text{ mm}, z = 306.504 \text{ mm},$$

$$r_z = 179.404^\circ, r_y = 0.26^\circ, r_x = 179.921^\circ.$$

The differences of the transformation matrices can be shown to see how much they differ from each other. The components of the transformation matrix obtained using Robodyn is subtracted from the components of the matrix obtained using Tracker Pilot and Matlab:

$$\hat{t}_x = 0.3859 \text{ mm}, \hat{t}_y = 0.1847 \text{ mm}, \hat{t}_z = -0.0407 \text{ mm},$$

$$\hat{r}_z = 0.5562^\circ, \hat{r}_y = 0.0298^\circ, \hat{r}_x = -0.1906^\circ.$$

From these data, it can be seen that the transformation matrices are different. The method of finding the transformation matrix between the robot and the laser tracker will have to be modified. A transformation matrix with such an error cannot be used for real-time corrections to the robot during 3D printing. The measured points have to be converted from the coordinate system of the laser tracker to the coordinate system of the robot in order to compare the laser tracker and robot data with each other.

Of course, the first improvement can be to change the transformation of the tool. Second, the method would not have to rely on the forward kinematics computed by the robot. However, the robot could just provide values from the encoders at robot axes and computation of the forward kinematics [14] could be done alongside the data evaluation in Matlab. Furthermore, the parameters of the transformation matrix could not be computed as the mean of a fixed number of measurements but these parameters could be iteratively improve. The points could be added to the measurements until the parameter deviations fell below some threshold. These points can be sent to the robot via EthernetKRL so more and more points can be generated.

<sup>1</sup> <https://www.plm.automation.siemens.com/global/en/products/nx/>

### 6.1.3 Calibration result

At the end of this section, the calibration result itself is presented. That is the transformation matrix between the coordinate system of the robot root and the coordinate system of the positioner root. Transformation matrix between the DKP and the laser tracker  $\mathbf{T}_l^d$  was created from the Plane, Axis, Center point alignment method and its result is

$$\mathbf{T}_l^d = \begin{pmatrix} -0.501 & -0.865 & -0.0034 & 2161.769 \\ 0.865 & -0.501 & -0.004 & 1147.394 \\ 0.002 & -0.005 & 1 & -772.564 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (3)$$

To obtain the transformation matrix  $\mathbf{T}_d^r$ , matrices  $(\mathbf{T}_r^l \cdot \mathbf{T}_l^d)^{-1}$  from Equation (2) and (3) has to be multiplied:

$$\mathbf{T}_d^r = \begin{pmatrix} 0.010 & -0.943 & -0.003 & 3729.392 \\ 0.999 & 0.042 & 0.006 & -1807.468 \\ -0.005 & -0.004 & 1 & 7.273 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (4)$$

## 6.2 Comparison of printed part inspection methods

In this section, the methods that can be used for an automatic inspection of manufactured parts are compared. One method uses Polyworks Inspector metrology software. The second method first measures all the points and then processes and evaluates the raw data. The second one can be called as *Manual* method. Both methods were used to evaluate the same printed object (Figure 5.4). The measurement was held on the same day, first with Polyworks Inspector, then with Tracker Pilot and Matlab. The cylinder parameters can be compared with each other because it does not matter from which specific points they are composed. The layer thickness was determined at 12 points, but it does not make sense to compare the thickness of a particular layer at a point between the two methods. Although the robot program was not changed and the measured points are the same in both cases, the robot will not travel from the same point twice with absolute accuracy, especially when the last part of the path consists of incremental motion.

When measuring in Polyworks Inspector, the CAD model is aligned to the printed real object before the actual inspection of the features. The Plane, Axis, Center point alignment method was used to find the transformation matrix from the Printed part coordinate system to the laser tracker coordinate system:

$$\mathbf{T}_d^l = \begin{pmatrix} -0.5046 & 0.8634 & 0 & 101.48 \\ -0.8633 & -0.5046 & -0.0058 & 2440.345 \\ -0.005 & -0.0029 & 1 & 525.198 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

This can be decomposed into rotation and translation components:

$$r_z = -120.3046^\circ, \quad r_y = 0.2887^\circ, \quad r_x = -0.1674^\circ,$$

$$t_x = 101.48 \text{ mm}, \quad t_y = 2440.345 \text{ mm}, \quad t_z = 525.198 \text{ mm}.$$

Tracker Pilot provides the raw data from the laser tracker; it is the same as we would get when using the Real Time module. Tracker Pilot does not even compensate for the

radius of the ruby tip. It has a radius of 2.5 mm [13], so when the plane and lower ring points are measured, 2.5 mm is subtracted from their  $z$  coordinate. When the cylinder surface is measured, there is no reason to compensate for the radius of the ruby tip as it does not affect the centers of the circles, hence the center axis of the cylinder. Without the compensated probe tip, the circles are measured with diameters 5 mm smaller, so it should be added that to them. The resulting cylinder will have a correct diameter by then.

### 6.2.1 Layer thickness inspection

The results of the bottom ring thickness can be compared. The equations of the planes of the pad on which it is printed can be compared. In Polyworks Inspector, the equation of the plane can be extracted in the laser tracker's coordinate system.

$$\pi_1: -0.0033x - 0.0039y + z + 524.3903 = 0, \quad (5)$$

$$\pi_2: -0.0036x - 0.038y + 0.9999z + 524.753 = 0. \quad (6)$$

The plane  $\pi_1$  was found by Manual method and  $\pi_2$  by Polyworks Inspector. Comparing the coefficients of  $\pi_1$  and  $\pi_2$ , we can consider that the detection of the  $\pi_1$  plane by the Manual method works well.

The thickness of the printed layer at each points can be also evaluated. Polyworks Inspector unfortunately does not allow access to the points that compose the feature. The points would have to be measured individually and not as part of a plane. Which is not a problem to implement, but at the moment it is not.

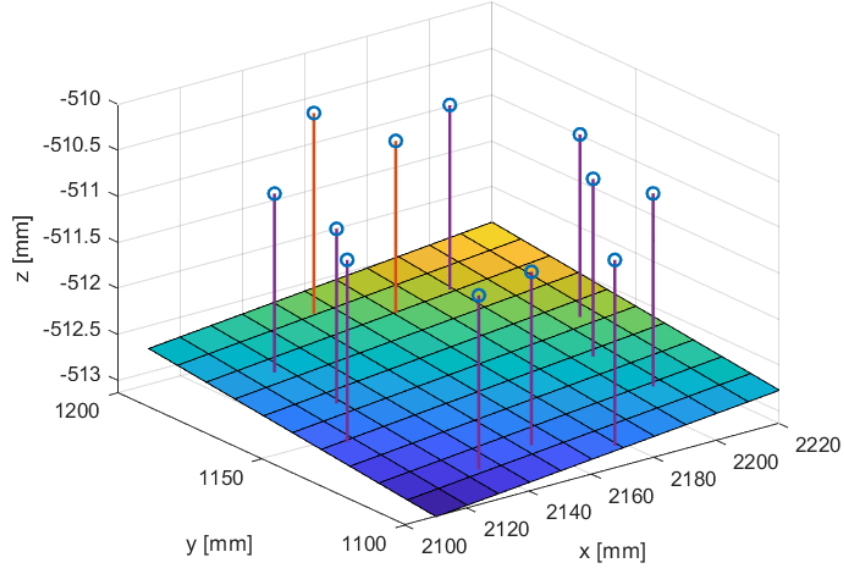
Method	Max Dev [mm]	Min Dev [mm]
Manual	0.212	-0.098
Polyworks	0.146	-0.104

**Table 6.1.** Maximum and minimum deviation from the expected value  $E(D)$ .

Table 6.1 shows the maximum and minimum deviations of the distance of the measured points from the nominal expected value  $E(D) = 2$ , where  $D$  is a random variable representing the distance of the measured point from the plane  $\pi$ . Since the nominal layer thickness is 2 mm,  $E(D) = 2$ . The deviations of the lowest points are almost the same, but the deviations of the highest points differ by approximately 0.07 mm. As written earlier, the robot probably did not measure exactly the same point, but the second time its position was slightly different, so it measured a different part of the surface that was slightly higher.

Figure 6.2 shows the fitted plane of the pad and the measured points on the surface of the printed part. That points and the plane was obtained by the manual method. The distance with the maximum and minimum deviation is highlighted in orange.





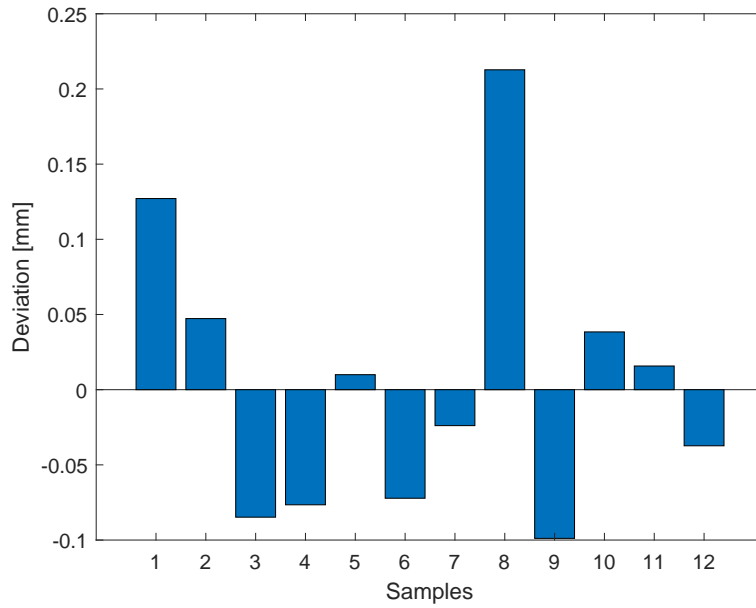
**Figure 6.2.** Fitted plane of the pad, measured points, and distances between them.

In Figure 6.3, it can be seen that the 8th measured point is the thickest in the whole layer, and the 9th measured point is the thinnest. These are the points with coordinates  $p_8 = (2154.224 \ 1190.505 \ -510.459)$ ,  $p_9 = (2170.571 \ 1179.804 \ -510.758)$ .

The standard deviation from the measured points can be calculated. This way, it can be found out the width of the neighborhood of the expected value. This indicates the range in which the majority of the measured points are located. The smaller it is, the less deviated the measured points are from the expected value. Standard deviation  $\sigma$  is defined as the square root of the variance ( $\sigma^2$ ) of a random variable  $D$ . When there are eight measured distances ( $d_1, \dots, d_8$ ) and a  $E(D) = 2$ , it is calculated from the formula [38]:

$$\sigma = \sqrt{\frac{(d_1 - 2)^2 + \dots + (d_8 - 2)^2}{8}}. \quad (7)$$

The standard deviation is  $\sigma = 0.089mm$ .



**Figure 6.3.** Deviations of layer thicknesses at each point.

Method	Dev Diameter [mm]	Dev X angle [°]	Dev Y angle [°]	Z angle [°]
Manual	-0.335	-1.453	0.37	1.499
Polyworks	-0.351	-1.201	0.274	1.232

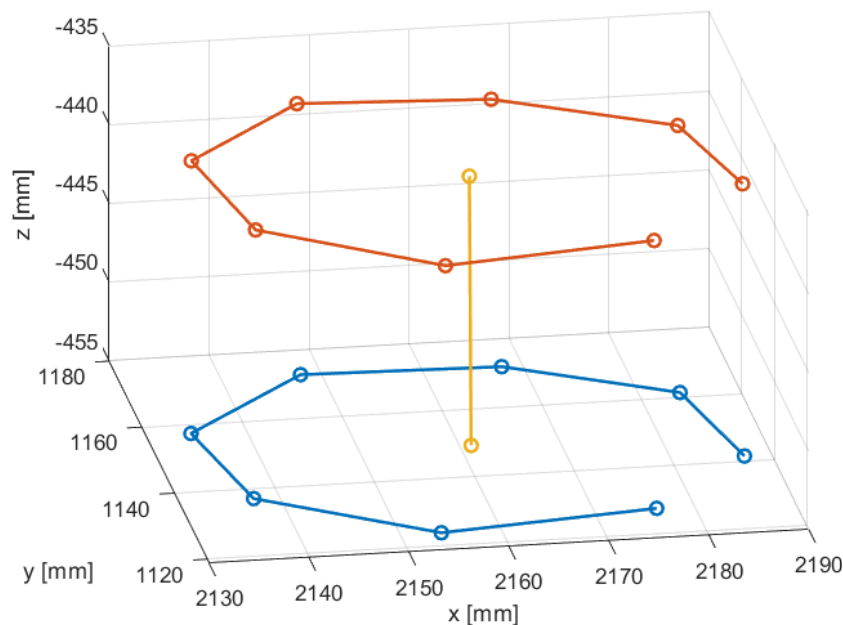
**Table 6.2.** Cylinder parameters deviations that were obtained with Tracker Pilot and Matlab and with Polyworks Inspector.

### 6.2.2 Cylinder inspection

The second feature that was inspected is the upper cylinder. Its parameters are invariant to the coordinate system. The parameters of the cylinder measured with the Polyworks Inspector can be seen in Figure 5.4.

Table 6.2 records the deviations of the radiuses and angular deviations of the center axes in each axis. The two methods produced almost identical diameters of the cylinder. But the center axis position is different. Probably the circle's centers were determined poorly. The deviation of the direction vector can be seen; it is more likely caused by the robot's poor positioning. The differences between the two methods could be resolved if more points and overall circles were added to the measurements and the centers of the circles were determined by some iterative method such as RANSAC (Random Sample Consensus) [43].

Figure 6.4 shows in orange and blue the measured points on the surface of the cylinder. They are interlinked with straight lines just to visualize the contour of the circle. The points consist of circles with a 5 mm smaller diameter because of the uncompensated tip; nevertheless, the 5 mm is added to the diameters of the measured circles during data evaluation. The yellow points are the detected centers of the circles, and the line between them is the center axis of the cylinder.



**Figure 6.4.** The measured points of the top cylinder, the centers of the circles, and the axis of the cylinder.

For the sake of completeness, the parameters of the cylinder, which were determined with manual method are now presented:

$$d_l = 59.684 \text{ mm}, d_u = 59.646 \text{ mm}, d = 59.665 \text{ mm},$$

$$\mathbf{u} = (-0.11 \quad 0.431 \quad 16.996)^T, \quad \mathbf{u}_n = (-0.0065 \quad 0.0254 \quad 0.9997)^T.$$

Where  $d_l$  is the diameter of the lower circle,  $d_u$  is the diameter of the upper cylinder, and  $d$  is the diameter of the cylinder. The nominal value of all diameters is  $\hat{d} = 60 \text{ mm}$ .  $\mathbf{u}$  is the direction vector of the cylinder; after its normalization, it produces  $\mathbf{u}_n$ , which can be better compared with the ideal variant of this vector, which is  $\hat{\mathbf{u}}_n = (0 \quad 0 \quad 1)^T$ .

## Chapter 7

### Conclusion

In this bachelor thesis, the main objectives were to design and develop a process for automatic robot workspace calibration and accurate measurement of a part with a touch probe using a laser tracker and an industrial robot. This thesis created a foundation for these solutions and explored the possibilities of using a laser tracker in cooperation with a robot. These tasks were successful, and it can be seen that the laser tracker has its place in the robotic workplace. This work has opened up great possibilities for future work.

- First, used equipment and software were introduced. Particular attention was given to the laser tracker and its equipment, which is not a common device like the industrial robot. The software used was also not emphasized as it would be difficult to describe all their functionalities. Rather, the workflow was described only in the sections where they were really needed and, moreover, where it was possible to see their real use.
- Furthermore, the data transfer and communication between the laser tracker and the robot controller or application computer was presented. Since the laser tracker should work autonomously with the robot on the application software, the receiving of measured data from the laser tracker by the robot program was developed in this part. At the same time, a protocol for sending commands to the laser tracker was implemented. The native format of the numbers sent by the laser tracker is 64 bit double, which, however, KRL cannot handle. It can only handle the REAL number format, which is a 32 bit float. So this problem had to be solved first. Then a converter from spherical point coordinates to Cartesian coordinates and a converter of quaternions to rotation matrix for the rotation parameters had to be created. These parts had to be implemented in the RSI since they had to be computed at a high frequency. The last point in the communication is sending the laser tracker requests. This could be implemented in KRL because there is no need to send commands to the slave (the tracker) as fast as possible. Commands to perform a special type of measurement (e.g., longer acquisition time), send the laser tracker to a specified position and turn off the tracker have been implemented.
- In the following part, a process for obtaining the transformation matrix between the coordinate system of the robot root and the laser tracker and then between the coordinate system of the laser tracker and the positioner was developed. These transformations are multiplied together to create a transformation matrix between the robot root and the positioner coordinate system. This matrix is necessary for the workspace calibration, so it was found where the table is positioned in the real world. This transformation matrix was also used to calibrate the digital twin of the workstation. The process of obtaining the transformation matrix between the robot and the positioner was developed independently of third-party software.

The transformation itself between the robot and the laser tracker could be used to correct the robot trajectories in the future. The measured points from the laser tracker are converted into the robot's coordinate system using this transformation

matrix, and then we can compare these points with those measured by the robot itself and adjust its trajectory.

- In the next chapter has been developed a procedure for the automatic inspection of manufactured parts. First, in Polyworks Inspector, an autonomous part inspection was developed, which starts with importing the CAD model, followed by the alignment of the real part, then the measurement of the specified points and features and their evaluation, and finally, a report is generated from the measurements. A method without using Polyworks Inspector has also been developed. The emphasis was on the possibility of inspecting the manufactured part without metrology software. The fitting of points to the plane and to the cylinder was tackled. This does not yet include alignment of the CAD model, any work with the CAD model is not expected, but a transformation of the measured data into the positioner coordinate system is added so that the measured points are in a uniform coordinate system.
- The methods from Chapters 4 and 5 were then tested. First, the accuracy of the transformation matrix between the robot and laser tracker coordinate systems which was obtained with Tracker Pilot and Matlab was compared with that obtained using Robodyn. However, its accuracy is not sufficient, differing by up to 0.39 mm and 0.56 ° in the individual components of translation and rotation. Next, the automatic inspection methods were compared. The accuracy of the inspection of the bottom layer thickness was sufficient, whereas the accuracy of the cylinder parameter detection was worse. The deviations were not sufficient. Probably, the method requires more points in more circles and an iterative method of cylinder fitting.

The possibilities to further develop the methods presented in this thesis are great. In particular, to improve their accuracy to take full advantage of the laser tracker's potential and the completion of the communication protocol between the laser tracker, the robot controller, and the PLC. Then the measurement flow could be controlled from the PLC.

Furthermore, the process of obtaining the transformation matrix between the robot coordinate system and the laser tracker has to be improved. It is necessary to ensure that the points measured by the laser are converted into the robot's coordinate system as accurately as possible so that the robot's positions can be compared with those measured by the laser tracker.

For automatic inspection, there is no need to refine methods for measuring specific features. Of course, the objects to be measured change, and the methods for their inspection will be solved ad hoc. For automatic inspection, it would be useful to create a robot path generator. It is tedious to program them one by one, this way, the user would choose which points will be measured, and the path generator would generate the trajectory considering the geometry of the object to be measured.

## References

- [1] *Coordinate Measuring Machines Measurement System Types and Characteristics Measurement Fundamentals* KEYENCE America.  
<https://www.keyence.com/ss/products/measure-sys/measurement-selection/type/3d.jsp>.
- [2] *KR CYBERTECH nano*. KUKA Roboter GmbH.  
[http://www.wtech.com.tw/public/download/manual/kuka/KUKA%20CYBERTECH\\_nano\\_en.pdf](http://www.wtech.com.tw/public/download/manual/kuka/KUKA%20CYBERTECH_nano_en.pdf).
- [3] *KUKA Positioner*. KUKA Roboter GmbH.  
[https://s3-eu-central-1.amazonaws.com/centaur-wp/theengineer/prod/content/uploads/2014/04/11134900/15\\_Spez\\_DKP\\_400\\_en.pdf](https://s3-eu-central-1.amazonaws.com/centaur-wp/theengineer/prod/content/uploads/2014/04/11134900/15_Spez_DKP_400_en.pdf).
- [4] *Leica AT930/AT960, User Manual*. Leica Geosystems AG.
- [5] Ron Eng. *Leica Absolute Distance Meter*.  
<https://manualzz.com/doc/33278244/leica-absolute-distance-meter>.
- [6] *Leica Absolute Interferometer*. . Hexagon Metrology.
- [7] K.M. Nasr, A.B. Forbes, B. Hughes, and A. Lewis. ASME B89.4.19 standard for laser tracker verification – experiences and optimisations. *International Journal of Metrology and Quality Engineering*. 2012, 3 (2), DOI 10.1051/ijmqe/2012014.
- [8] *Red Ring Reflector 1.5"*.  
[https://shop.hexagonmi.com/na/en\\_US/USD/Catalog/Laser-Tracker/Reflectors/Red-Ring-Reflector-1-5%22/p/575784](https://shop.hexagonmi.com/na/en_US/USD/Catalog/Laser-Tracker/Reflectors/Red-Ring-Reflector-1-5%22/p/575784).
- [9] *Leica T-Mac, User Manual*. Leica Geosystems AG.
- [10] *TESASTAR-p Touch Trigger Probes For Probe Heads*.  
[https://grafker.hu/wp-content/uploads/2013/09/Q\\_2010\\_EN\\_TESASTAR\\_Probe\\_Head.pdf](https://grafker.hu/wp-content/uploads/2013/09/Q_2010_EN_TESASTAR_Probe_Head.pdf).
- [11] *KUKA System Software 8.3*. KUKA Roboter GmbH.  
<http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKA%20KS S-8.3-Programming-Manual-for-SI.pdf>.
- [12] *KUKA.RobotSensorInterface 3.1*. KUKA Roboter GmbH.  
<https://manualzz.com/doc/44608163/kuka.robotsensorinterface-3.1>.
- [13] *LEICA TRACKER PILOT, FOR AT9X0*. Hexagon AB.
- [14] Bruno Siciliano, Lorenzo Sciacivco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. London: Springer, 2010. ISBN 978-1-84628-641-4.
- [15] *RoboDyn User Manual*. New River Kinematics.
- [16] *Reference Guide: PolyWorks/Inspector*. INNOVMETRIC SOFTWARE INC..
- [17] *Leica Integrated Solutions, Configuration Manual*. Leica Geosystems AG.
- [18] *Leica Absolute Tracker AT960/AT930, RTFP-EC Developers Guide*. Leica Geosystems AG.

- [19] *EtherCAT User's Manual*. Estun Automation Technology CO., LTD.  
<https://www.estuneurope.eu/wp-content/uploads/download/Manuali/Fielbus/EtherCAT-User-s-Manual-V1-08.pdf>.
- [20] RealPars. *What is EtherCAT?*.  
<https://www.youtube.com/watch?v=tYA12jkaB8Q>.
- [21] IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*. 2019, 1–84. DOI 10.1109/IEEESTD.2019.8766229. Conference Name: IEEE Std 754-2019 (Revision of IEEE 754-2008).
- [22] Eric W. Weisstein. *Spherical Coordinates*.  
<https://mathworld.wolfram.com/SphericalCoordinates.html?affiliate=1>. From MathWorld—A Wolfram Web Resource.
- [23] Christopher Stover, and Eric W. Weisstein. *Cartesian Coordinates*.  
<https://mathworld.wolfram.com/CartesianCoordinates.html>. From MathWorld—A Wolfram Web Resource.
- [24] Eric W. Weisstein. *Rotation Formula*.  
<https://mathworld.wolfram.com/RotationFormula.html>. From MathWorld—A Wolfram Web Resource.
- [25] Eric W. Weisstein. *Quaternion*.  
<https://mathworld.wolfram.com/Quaternion.html>. From MathWorld—A Wolfram Web Resource.
- [26] Eric W. Weisstein. *Rotation Matrix*.  
<https://mathworld.wolfram.com/RotationMatrix.html>. From MathWorld—A Wolfram Web Resource.
- [27] Serge Belongie. *Rodrigues' Rotation Formula*.  
<https://mathworld.wolfram.com/RodriguesRotationFormula.html>. From MathWorld—A Wolfram Web Resource, created by Eric W. Weisstein.
- [28] Eric W. Weisstein. *Euler Angles*.  
<https://mathworld.wolfram.com/EulerAngles.html>. From MathWorld—A Wolfram Web Resource.
- [29] *Aircraft Rotations*.  
<https://www.grc.nasa.gov/WWW/K-12/airplane/rotations.html>.
- [30] James Diebel. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. 35.
- [31] *Four-quadrant inverse tangent - MATLAB atan2*.  
<https://www.mathworks.com/help/matlab/ref/atan2.html>.
- [32] Evan G. Hemingway, and Oliver M. O'Reilly. Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments. *Multibody System Dynamics*. 2018, 44 (1), 31–56. DOI 10.1007/s11044-018-9620-0.
- [33] *To create a PLP (Plane, Line, Point) alignment PowerInspect 2019 Autodesk Knowledge Network*.  
<https://knowledge.autodesk.com/support/powerinspect/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/PWRI-ReferenceHelp/files/GUID-09496C7F-8475-4F90-A314-7766B6963CEA-htm.html>.
- [34] Eric W. Weisstein. *Plane*.  
<https://mathworld.wolfram.com/Plane.html>. From MathWorld—A Wolfram Web Resource.

- [35] *System Variables: For KUKA System Software 8.1, 8.2 and 8.3*. KUKA Roboter GmbH.  
<http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKA%20System%20Variables%208.1%208.2%208.3.pdf>.
- [36] *Networking with a KUKA Control PC Tutorial*.  
<https://www.youtube.com/watch?v=qu1jxcGFuNI>.
- [37] Gregory G Slabaugh. Computing Euler angles from a rotation matrix. 7.
- [38] Mirko Navara. *Pravděpodobnost a matematická statistika*. Centrum strojového vnímání katedra kybernetiky FEL ČVUT,  
[https://cmp.felk.cvut.cz/~navara/stat/PMS\\_print.pdf](https://cmp.felk.cvut.cz/~navara/stat/PMS_print.pdf).
- [39] Jan Leinveber, and Pavel Vávra. *Strojnické tabulky: pomocná učebnice pro školy technického zaměření*. Uvaly: Albra, 2011. ISBN 978-80-7361-081-4. OCLC: 776515128.
- [40] Tomáš Werner. *Optimalizace: Elektronická skripta přemětu B0B33OPT*. Katedra kybernetiky Fakulta elektrotechnická České vysoké učení technické,  
[https://cw.fel.cvut.cz/b211/\\_media/courses/b0b33opt/opt.pdf](https://cw.fel.cvut.cz/b211/_media/courses/b0b33opt/opt.pdf).
- [41] Jiří Velebil. *Abstraktní a konkrétní lineární algebra*. České vysoké učení technické v Praze Fakulta elektrotechnická,  
[https://math.fel.cvut.cz/en/people/velebil/files/akla/akla\\_2022\\_02\\_11.pdf](https://math.fel.cvut.cz/en/people/velebil/files/akla/akla_2022_02_11.pdf).
- [42] *KUKA.Ethernet KRL 2.1: For KUKA System Software 8.2*. KUKA Roboter GmbH.  
<http://www.wtech.com.tw/public/download/manual/kuka/krc4/KST-Ethernet-KRL-21-En.pdf>.
- [43] Konstantinos G Derpanis. *Overview of the RANSAC Algorithm*. 2010.  
[http://www.cse.yorku.ca/~kosta/CompVis\\_Notes/ransac.pdf](http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf).



# Appendix A

## Thesis assignment



### BACHELOR'S THESIS ASSIGNMENT

#### I. Personal and study details

Student's name: **Kubá ek Václav** Personal ID number: **483422**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Control Engineering**  
Study program: **Cybernetics and Robotics**

#### II. Bachelor's thesis details

Bachelor's thesis title in English:

**Automation of precision measurement process using laser tracker and industrial robot**

Bachelor's thesis title in Czech:

**Automatizace procesu p esného m ení s využitím laser trackeru a pr myslového robota**

Guidelines:

- 1) Learn the different parts of the robotic workstation and the principle of the main components.
- 2) Familiarize yourself with the communication method between the robot controller, the laser tracker controller and application software. Analyse the type of data variables used and, if necessary, propose their conversion to ensure compatibility. Provide wiring diagrams.
- 3) Design and develop a process for automatic calibration of the robot workspace using a Leica T-Mac (Tracker-Machine control sensor) equipped with a touch probe. Implement algorithms for processing the measured data and its evaluation in Matlab software. Provide a flow chart of the workstation functions.
- 4) Design and develop a process to measure a part accurately using a Leica T-Mac device equipped with a touch-sensing probe. Use the application software of the laser tracker to evaluate the measured data and compare them a reference CAD model. Provide a flow chart of the workstation functions.
- 5) Test and compare two methods given in 3) and 4). Focus on determining the accuracy of the measurements and the advantages and disadvantages of each method.

Bibliography / sources:

- [1] Bruno Siciliano, Oussama Khatib (Eds.). Springer Handbook of Robotics. 2008. ISBN: 978-3-540-23957-4.
- [2] KUKA DEUTSCHLAND GMBH. KUKA.RobotSensorInterface 4.1: For KUKA System Software 8.6. Germany, 2019, 129 s. KTS RSI 4.1 V2.
- [3] LEICA GEOSYSTEMS AG- METROLOGY PRODUCTS. Leica Absolute Tracker AT960/AT930: RTFP-EC Developers Guide. 2016, 27 s. Document Revision: 1.0.

Name and workplace of bachelor's thesis supervisor:

**Ing. Tomáš Jochman Testbed CIIRC**

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **31.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until:  
**by the end of summer semester 2022/2023**

Ing. Tomáš Jochman  
Supervisor's signature

prof. Ing. Michael Šebek, DrSc.  
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

### III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Appendix B

### Photographs of devices



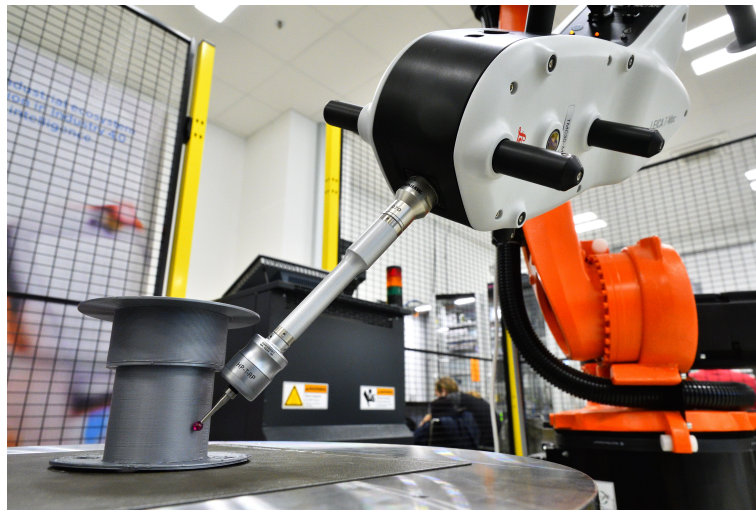
**Figure B.1.** Laser tracker AT960 on a tripod.



**Figure B.2.** Touch probe with a ruby tip attached to the T-Mac.



**Figure B.3.** Automation Interface Controller.



**Figure B.4.** An example of the inspection process of a manufactured part with a touch probe.