F3

Faculty of Electrical Engineering
Department of Cybernetics

Master's Thesis

Utilization of 3D Vision System for Robotic Sanding with Force Feedback Control

Bc. Václav Kubáček
Study programme: Cybernetics and Robotics

May 2024
Supervisor: Ing. Tomáš Jochman

Acknowledgement / Declaration

I would like to express my genuine thanks to my supervisor, Ing. Tomáš Jochman, for his continuous support, guidance, and insightful advice throughout the development of this thesis. His expertise and encouragement were invaluable in the successful completion of this thesis. I also extend my thanks to Pavel Burget, Ph.D., the head of the Testbed for Industry 4.0, where this research was conducted. His support and the resources provided by the department were essential for the practical implementation and testing of the system.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, May 24, 2024

.....

Abstrakt / Abstract

Tato diplomová práce se zabývá vývojem autonomního robotického systému pro broušení a leštění konvexních povrchů s využitím systému 3D vidění a zpětnovazebního řízení. Hlavním cílem je zvýšit adaptabilitu a efektivitu robotického procesu broušení, zejména v prostředí s častými změnami výrobků. Systém integruje systém 3D vidění pro generování mračna bodů povrchu dílu, která jsou následně zpracována pro vytvoření přesných robotických drah. Dráhy jsou generovány ve formě klikatých a spirálových vzorů, aby bylo zajištěno komplexní pokrytí a konzistentní povrchová kvalita s ohledem na kolmé vedení brusného vřetena. K udržování konstantní přítlačné síly se používá senzor síly a momentu, který dynamicky upravuje dráhu robota na základě zpětné vazby v reálném čase, aby se přizpůsobil změnám povrchu. Mezi klíčové úspěchy patří vývoj robustních algoritmů zpracování mračna bodů, zpětnovazebního řídicího mechanismu pro udržování konstantní síly a použití komunikačního protokolu OPC UA pro plynulý a bezpečný přenos dat mezi součástmi systému. Zavedení rozšířené reality (AR) prostřednictvím soupravy Hololens 2 zlepšuje interakci s uživatelem a umožňuje operátorovi vizualizovat brousící dráhy v reálném čase a interagovat s pracovní stanicí. Účinnost systému byla ověřena v reálném průmyslovém prostředí a prokázala jeho potenciál pro zlepšení procesů robotického broušení a leštění ve flexibilních výrobních prostředích.

Klíčová slova: robotické broušení, systém 3D vidění, zpětnovazební řízení síly, zpracování mračna bodů, rozšířená realita

This thesis presents the development of an autonomous robotic system for sanding or polishing convex surfaces using a 3D vision system and force feedback control. The primary objective is to enhance the adaptability and efficiency of robotic sanding processes, especially in environments with frequent product changes. The system integrates a 3D vision system to generate point cloud data of the workpiece surface, which is then processed to create precise robotic paths. The paths are generated in the form of zig-zag and spiral patterns to ensure comprehensive coverage and consistent surface quality concerning the perpendicular alignment of the sanding spindle. A force-torque sensor is employed to maintain a constant pressing force, dynamically adjusting the robot's path based on real-time feedback to accommodate surface variations. Key achievements include the development of robust point cloud processing algorithms, a feedback control mechanism for maintaining uniform force, and the application of the OPC UA communication protocol for seamless and secure data exchange between system components. The introduction of augmented reality (AR) via the Hololens 2 headset enhances user interaction, allowing operators to visualize sanding paths in real-time and interact with the workstation. The system's effectiveness was validated in a real industrial setting, demonstrating its potential for improving robotic sanding and polishing processes in flexible manufacturing environments.

Keywords: robotic sanding, 3D vision system, force feedback control, point cloud processing, augmented reality

Contents /

Tables / Figures

Chapter 1
Introduction

This master thesis explores solutions to `Scan & Sand` problem with an industrial robotic manipulator. This task involves the sanding or polishing of manufactured objects whose shapes and specifications are not previously known. Typically, objects created through processes such as turning or 3D printing require such finishing to attain a smooth surface. Similarly, this approach is also applicable for stripping old paint or varnish from objects. Such objects often have complex shapes and require a skilled person to sand them.

1.1 Motivation

Despite the precision with which robots can follow a set trajectory, equipped with a polishing spindle, their application is currently limited. The trajectories must be meticulously programmed by an operator using CAD/CAM software, significantly reducing the robot's flexibility to adapt to new and varied objects. If the path is predefined, it is difficult for the robot to react to unexpected changes on the workpiece. In the production process, it is likely that the products will easily deviate from each other and will have their own imperfections. In such a case, the robot would not be able to maintain constant pressure, which is very important.

Beyond material processing, these robotic systems could revolutionize additive manufacturing. Printing on any arbitrarily curved surface remains a complex task[1]. However, a robot equipped with an extruder and a depth camera could easily overcome these limitations. When printing material onto an existing part, the surface must have a suitable adhesion and a smooth surface that can be achieved by sanding.

Another focal point of this thesis is the integration of appropriate visualization method, such as augmented reality (AR) headset within robotic workspace. There is a growing belief that AR technology could significantly enhance industrial operations, especially in robotic manufacturing [2]. For instance, unskilled operators could receive real-time, detailed guidance about workplace devices and their operation through an AR headset. Specifically, in sanding applications, operators could view the robot's sanding paths projected directly onto the object's surface. This feature allows for on-the-fly adjustments to the sanding pattern's density or the applied pressure, significantly enhancing the process's efficiency and effectiveness.

1.2 Goals

The primary goal of this thesis is to develop an autonomous robotic cell capable of sanding an unknown convex object with desired force. This process leverages a point cloud obtained by a depth camera, with visualization and user interaction facilitated through the AR headset. This thesis outlines the methods employed and their implementation in an actual industrial setting. The real workplace where the solution was implemented is located in the Testbed for Industry 4.0 at the Czech Institute of Informatics, Robotics

and Cybernetics, CTU in Prague[1]. Although the fundamental difference between sanding and polishing lies in the amount of material removed, this project is structured as a proof of concept; therefore, the specific application—whether for sanding or polishing—is secondary.

A core component of the methodology is an algorithm that processes data from a 3D Vision system. It consists of processing the point cloud and then generating two sanding patterns, namely zig-zag and spiral patterns. The pattern consists of the robot's positions and the end-effector's orientations in such a direction that the spindle is always perpendicular to the object's surface. Between each point the robot traverses in linear motions. Since the presented algorithm must be aware of the robot kinematics, a forward and inverse kinematic problem for a 6-DOF (Degrees of Freedom) robotic manipulator is introduced.

Furthermore, to maintain a constant pressing force—a critical factor in effective sanding—a feedback mechanism adjusts the robot's position based on real-time force measurements. This also requires the development of a straightforward process that determines the gravitational force compensation in the steady state of the tool attached to the force-torque sensor.

Last but not least, the OPC UA (OPC Unified Architecture) communication protocol is briefly introduced. It is used for communication and data transfer between the application computer and the robot controller, as well as for communication and data transfer between the robot and the AR headset.

The implementation section provides a comprehensive overview of the automated sanding process pipeline. Where at the beginning, it is necessary to calibrate the workspace, i.e., to find out the transformation matrix between the robot coordinate system and the depth camera coordinate system. Followed by the detailed steps from point cloud processing to generating executable robot code for the KR C4 controller. Then, a concrete implementation of the feedback control of the pressing force in the RSI (Robot-Sensor Interface) is presented. The possibility of simulating a robot program in RoboDK software is also investigated, which is especially useful for path verification and collision control. Moreover, all the communication protocols that are used in the data transfer between the individual components of the workplace are described. There is even a description of the process of developing an application for AR Headset in Unity with emphasis on the alignment of the whole scene to a specific part being sanded.

[1] `https://ricaip.eu/testbed-prague/`

Chapter 2
Related Work

This chapter provides a comprehensive review of the state of the art in robotic systems for sanding and polishing, with a particular focus on the integration of 3D vision systems and force feedback control. It examines key developments in robotic path planning, dynamic force control, and real-time 3D vision applications. The synthesis of this literature not only outlines the evolution of robotic capabilities but also frames the technological backdrop against which this thesis proposes enhancements to robotic sanding systems.

2.1 Approach to sanding

In their study on automatic polishing for curved surfaces, Fengjie Tian et al. [3] introduce a force control strategy that adeptly manages the application of consistent polishing force through both active and passive compliance mechanisms. Key to this approach is the incorporation of gravity compensation to neutralize the impact of the tool's weight and an explicit force control based on position, ensuring uniform application of force. This is mathematically represented in the paper as $F_m(i) = F_c(i) + F_g$, where $F_m(i)$ is the measured force, $F_c(i)$ is the polishing force, and F_g is the gravitational force on the polishing tool.

A central element of their methodology is the development of a material removal model, which is crucial for achieving uniform material removal across the workpiece. This model is formulated to understand the interaction between the polishing tool and the curved surface, enabling the determination of optimal path spacing. The material removal profile along the vertical feed direction is described by $z(y) = ay^2 + b$, where $z(y)$ is the material removal depth, and a and b are model parameters that account for the tool-workpiece interaction dynamics.

Moreover, Tian et al. propose an optimal path spacing algorithm derived from their material removal model to ensure even material removal and high-quality surface finishes. The algorithm's foundation lies in the equation for path spacing $S = \sqrt{2b/a}$, which plays a pivotal role in minimizing material removal depth variations and achieving the desired surface quality [3].

The paper by Joshua Nguyen, Manuel Bailey, Ignacio Carlucho, and Corina Barbalata [4] presents an innovative system for automated sanding. This system is distinguished by its ability to adjust the manipulator's velocity based on real-time assessment of surface quality, which is inferred from vibration data captured by a force-torque sensor at the end-effector. This methodology leverages two primary control strategies: a variable velocity generation law and a pose regulation-based law. The former dynamically adjusts the robot's speed along the tangential direction of the sanding path, conditional upon the amplitude and frequency of the force signal, indicative of surface roughness.

Their methodology mainly focuses on the correlation between the sanding process's vibrational characteristics and the resultant surface finish. By analyzing the force

signal in the frequency domain using FFT (Fast Fourier Transform), the system can distinguish the surface's smoothness through variations in frequency and amplitude, thus enabling real-time adjustments to the sanding velocity.

One technical contribution is the detailed formulation of the vibration-driven control law, which is inversely proportional to the vibration frequency and amplitude, as observed during the sanding process. This law ensures that the robot spends more time on rougher surfaces and accelerates over smoother areas, optimizing the sanding operation's efficiency. Their Variable Velocity Generation Law is as follows:

$$v_t = \frac{1}{u_\lambda + u_A + \epsilon},$$

where v_t is the velocity of the manipulator in the tangential direction, u_λ is a term that adjusts the velocity based on the difference in the measured sanding frequency λ_m from a predefined target sanding frequency λ_d, u_A adjusts the velocity based on the difference in the measured amplitude A_m from a predefined target amplitude A_d and lastly ϵ is a small constant to prevent division by zero [4].

The research paper by Alberto García et al. [5] introduces a human-robot cooperation system designed to automate surface treatment operations such as sanding, deburring, and polishing. The core of their methodology hinges on synergistic cooperation where the human operator dictates the regions of the workpiece for treatment, using the robot's precision and power for execution. The system utilizes a camera network for real-time, accurate localization of the workpiece within the robot's workspace, facilitating dynamic adjustments to changes, including workpiece repositioning.

The technical foundation of this work is built upon a multi-tiered control strategy, prioritizing

1. a smooth approach and boundary adherence to keep the robot tool within a designated area near the workpiece,
2. maintaining tool orientation to ensure perpendicular contact with the workpiece surface and
3. flexible operational modes for manual and automatic surface treatment.

Specifically, the smooth approach control employs a robust system regulating the tool's approach velocity to decrease as the tool nears the workpiece, adhering to a predefined safety margin. Boundary constraints are enforced through a control system that confines the robot tool within an allowable proximity to the workpiece, thereby preventing unnecessary movements and potential collisions with workspace objects.

Orientation control is achieved through a real-time feedback instrument ensuring the tool's perpendicular alignment with the workpiece surface. Lastly, the integration of manual and automatic operational modes enhances the system's flexibility, allowing operators to manually guide the tool for specific treatments or enabling the robot to autonomously service pre-determined areas [5].

2.2 Paths generation

The paper by Manuel Amersdorfer and Thomas Meurer [6] introduces an innovative equidistant tool path and Cartesian trajectory planning strategy for robotic machining of curved freeform surfaces, leveraging arc-length parameterization. Central to their method is the transformation between 3D Cartesian coordinates and 2D parameters based on surface arc-lengths, facilitating the generation of equidistant machining paths

that maintain consistent distances regardless of local geometric or surface curvature variations. This approach is particularly beneficial for applications requiring uniform coverage, where traditional iso-scallop [7] or iso-planar [8] methods may fall short due to their dependence on the local curvature or the complexity in handling equidistant paths across varying surface geometries.

This transformation is articulated as:

$$u = \int_{xref}^{x} \sqrt{1 + \left(\frac{\partial f_z}{\partial x}\right)^2}\, dx,\ v = \int_{yref}^{y} \sqrt{1 + \left(\frac{\partial f_z}{\partial y}\right)^2}\, dy,$$

where u and v represent the transformed parameters in the arc-length domain, with $\frac{\partial f_z}{\partial x}$ and $\frac{\partial f_z}{\partial y}$ denoting the partial derivatives of the surface function f_z with respect to x and y, respectively. This transformation enables the planning of equidistant paths within a parameter space that is inherently independent of the surface's local curvature.

Further refining their strategy, they introduce an inverse interpolation scheme employing cubic splines to map the planned paths back into Cartesian coordinates, ensuring that the equidistance property is preserved across larger path intervals. This meticulous approach to trajectory planning also includes the consideration of centripetal acceleration limits, ensuring constant velocity along the path through the equation:

$$\dot{\tau}(t) = \frac{v_d}{\|\gamma'(\tau(t))\|},$$

where $\tau(t)$ denotes the path parameter as a function of time, v_d is the desired constant velocity, and $\gamma'(\tau(t))$ represents the derivative of the path with respect to τ, indicating the path's curvature [6].

The paper High Precision Trajectory Planning on Freeform Surfaces for Robotic Manipulators by Renan S. Freitaset et al. [9] presents a comprehensive methodology for off-line automatic trajectory generation for robotic manipulators operating on 3D freeform surfaces.

Central to their methodology is the use of the Radial Basis Function (RBF) [10] for high-precision surface modeling and a marching method algorithm for path generation. The RBF interpolation, a necessary component of their approach, is defined as:

$$f(x) = \sum_{i=1}^{N} \lambda_i \phi(\|\mathbf{x} - \mathbf{x_i}\|),$$

where ϕ is the basis function such as Gaussian $\phi(r) = e^{-cr^2}$ or triharmonic spline $\phi(r) = r^3$, $\mathbf{x_i}$ are the centers of the RBF, and λ_i are the weights determined to fit the surface. This formulation is crucial for accurately modeling the complex geometries of freeform surfaces.

The marching method algorithm [11], utilized for generating paths on these modeled surfaces, iteratively calculates intersection points between the surface and a cutting plane, creating a trajectory path. This method's iterative nature allows for generating precise and continuous paths across the surface of the workpiece.

For trajectory planning, the paper emphasizes the need for discretizing the CAD model and estimating object normal vectors, computing an implicit representation with RBF, and generating paths through the marching method algorithm [9].

Gang Wang et al. in the paper Trajectory Planning and Optimization for Robotic Machining Based On Measured Point Cloud [12] introduces a methodology for robotic

machining path planning, particularly suited for large, complex parts that are predisposed to deformation. The foundation of this method is the transformation of point cloud data, obtained through onsite measurements of deformed parts, into a machining path that compensates for the part's actual deformed geometry.

A key aspect of their methodology is the generation of dual Nonuniform Rational B-Spline (NURBS) curves [13] from the point cloud data. These curves represent the machining path points and the tool axis points, effectively smoothing the initial rough path derived directly from the point cloud. This process involves a least-squares NURBS curve fitting algorithm, formulated as follows:

$$C(u) = \frac{\sum_{j=0}^{m} w_j N_{j,k}(u) \mathbf{d_j}}{\sum_{j=0}^{m} w_j N_{j,k}(u)}$$

where $C(u)$ is the curve prescription, $N_{j,k}(u)$ are the basis functions, $\mathbf{d_j}$ are the control points, and w_j are the weights and $k = 3$ to keep the second derivative of the curve continuous. The prescription of curve $C(u)$ is used to generate both NURBS curves C_P, C_Q for the tool path points and tool axis points, respectively.

To ensure the smoothness and accuracy of the generated path, an objective function for smoothness optimization is established. This function aims to minimize the deformation energy while considering the constraints of deviation from the initial measured path. The optimization integrates both geometric continuity and fidelity to the actual part geometry, which is crucial for maintaining the desired machining quality.

Furthermore, the methodology extends to robot pose optimization, focusing on the dexterity and stiffness of the robotic arm during machining. This step is vital for enhancing the stability and quality of the machining process, ensuring the robot maintains optimal configurations that avoid kinematic limits and ensure efficient material removal [12].

Chapter 3
Methodology

This chapter explains the state-of-the-art methodologies employed in this thesis to enhance the functionality and efficiency of robotic sanding systems through the integration of a 3D vision system and force feedback control. The methods are designed to be as general as possible so that they can be implemented in different workplaces, even with modified tasks. The core of the workplace is a robotic manipulator with 6-DOF. A force-torque sensor and a sanding spindle must be attached to it. The workstation also includes a work table to which the part to be sanded is attached. The part is to be scanned by a 3D vision system, which creates a point cloud on which the sanding paths for the robot are generated. The diagram in Figure 3.1 shows all the essential components of the workstation.

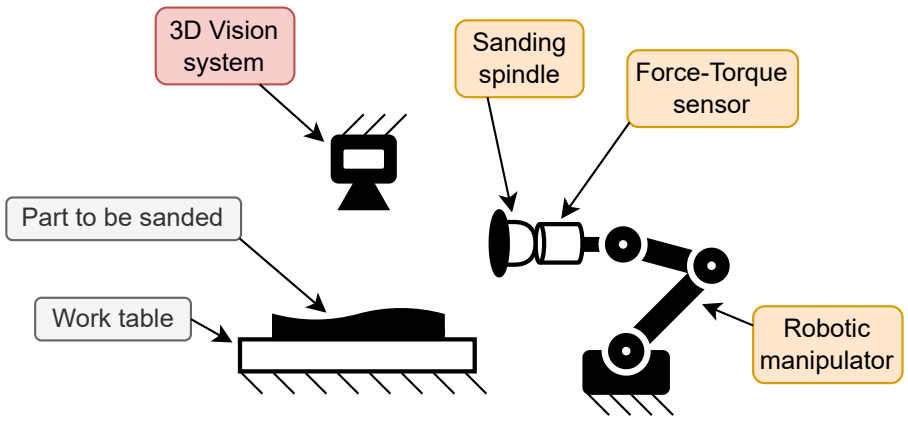

Figure 3.1. Overview of the workstation with fundamental components.

The work process begins with the 3D spatial data acquisition using a depth camera, which serves as the foundation for generating precise robotic paths tailored to the unique contours of each workpiece. The core of the methodology chapter discusses the use of path generation algorithms, specifically the development of zig-zag and spiral sanding patterns that adapt to complex convex geometries.

Subsequent sections explore the dynamic force control strategies implemented to maintain constant pressure during the sanding process, crucial for achieving uniform material removal. This includes an in-depth examination of real-time feedback mechanisms and their integration with the robotic control system to adjust the sanding force dynamically in response to real-time data from the force-torque sensor.

Additionally, the chapter describes the application of the OPC UA communication protocol for data exchange between the robot controller and other system components, enhancing the overall responsiveness and reliability of the sanding operation.

The methodology also covers the calibration processes necessary for aligning the robot's coordinate system with that of the 3D vision system, ensuring accuracy and consistency in the execution of sanding paths.

3.1 Point cloud processing and path generation

This section describes how the point cloud, obtained from a depth cameras, is processed for the purpose of robotic sanding. The algorithms that extract the sanding paths from the point cloud are also presented. These paths are generated in terms of patterns, namely zig-zag and spiral patterns. The concrete process of obtaining the point cloud from the camera is explained in section 4.3.1. It is now assumed that it is being worked with some point cloud of the scene, which has a resolution of approximately 1 mm.

3.1.1 Point cloud processing

The entering point cloud may look like the one in picture 3.2. As it can be seen in the picture, the point cloud is in the camera coordinate system, so it is necessary to transform it into the robot coordinate system first.

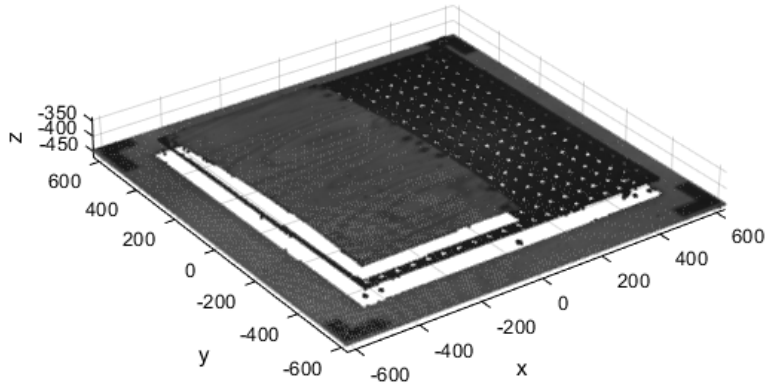

Figure 3.2. Input point cloud into the process of generating robot sanding paths.

This is done by applying a transformation matrix to each point of the point cloud. The transformation matrix $\mathbf{T}$ is obtained by the process of workspace calibration, which is described in section 3.5. Such a matrix consists of a rotation matrix $\mathbf{R}$ and a translation $\mathbf{t}$ between the coordinate system origins. It is important to note that such a matrix $\mathbf{T}$ transforms homogeneous coordinates. Each point must, therefore, be expanded into 4D, transformed, and then returned to 3D; this is achieved by adding 1 as the fourth coordinate. In the following equation, there is such a matrix $\mathbf{T}$ and how the i-th point of the point cloud is transformed from the camera coordinate system ($\mathbf{p}_i^c$) to the robot coordinate system ($\mathbf{p}_i^r$) [14]:

$$\mathbf{T} = \begin{pmatrix} & \mathbf{R} & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{pmatrix} \longrightarrow \begin{pmatrix} \mathbf{p}_i^r \\ 1 \end{pmatrix} = \mathbf{T} \begin{pmatrix} \mathbf{p}_i^c \\ 1 \end{pmatrix}. \tag{1}$$

Once the point cloud is transformed into the robot's coordinate system, it is possible to discard unnecessary objects, i.e., the floor and the table on which the workpiece is

mounted. This is done simply by thresholding, knowing how high the table is relative to the robot, i.e., all points of the point cloud that have a z coordinate less than a certain threshold are removed.

Further, the outliers are removed, which are typically located on the edges of the workpiece, where the camera can see poorly. These points would spoil the algorithm for generating the sanding pattern. They are removed using the statistical removal method. This means that points that are more distant from their neighbours than the average distance of the point cloud are removed [15].

The last thing needed is to estimate the normals. For this, a method has already been prepared in the Open3D library. The covariance analysis algorithm is used there. It is based on Principal Component Analysis (PCA), which estimates surface normals by analyzing the local geometric distribution of points around each point of interest in a point cloud. For each point, it defines a neighborhood either through K nearest neighbors or a radius-based search, then calculates the covariance matrix of these neighborhood points to capture their spread in 3D space. By performing eigenvalue decomposition on this covariance matrix, the algorithm identifies the eigenvector associated with the smallest eigenvalue as the surface normal, since this direction represents the axis of least variance, effectively perpendicular to the local surface. An additional step is unifying the orientation of the estimated normals towards a consistent viewpoint. A virtual point with coordinates $\begin{pmatrix} 1300 & 200 & -1000 \end{pmatrix}^T$ is created, all the normals must be pointing towards this point [15].

The processed point cloud, which serves as input to the functions for creating sanding patterns, is shown in Figure 3.3. It can be seen that the points are in the robot's coordinate system, most of the outliers are removed, and all the normals are pointing downwards. For clarity, there is also a zoomed-in detail of the point cloud.

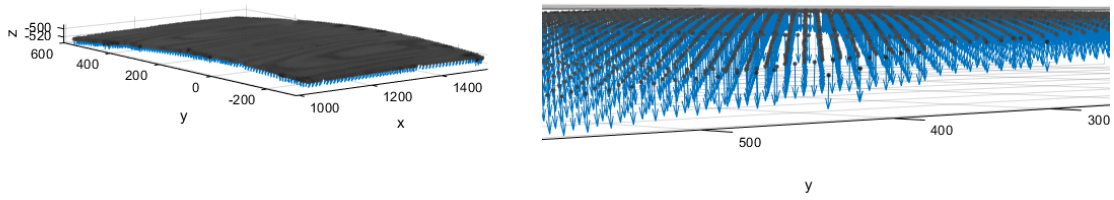

Figure 3.3. Processed point cloud, prepared for a generation of sanding patterns.

3.1.2 Zig-zag pattern creation

The algorithm for creating a zig-zag pattern from a given point cloud for robotic manipulator sanding operations is a process that effectively organizes a complex set of 3D data points into a structured path. This path is designed to guide a robotic arm across a surface in a zig-zag motion, optimizing coverage and efficiency.

The output is an array of transformation matrices as in Equation (1). This defines both the positions and orientations of the tool in the robot coordinate system and is independent of the conventions used by different robot manufacturers. The algorithm

is divided into two main functions: one that generates the positions of the path and a second that adds orientation to each point in the path based on the closest point's normal vector in the point cloud. The pseudo-code of the `zig_zag_pattern` function is provided here:

```
Function zig_zag_pattern(poinct cloud: pcd, tool radius: r):
    Initialize: tolerance_x = r * 0.33
    Initialize: voxel_radius = r * 0.5
    Downsample pcd using voxel_radius to create pcd_voxel_grid
    Convert pcd_voxel_grid points to a numpy array: points

    Initialize: path = []
    Initialize: going_right = True

    While points are not empty:
        Find the point with the minimum x-coordinate: min_x_point
        Points within tolerance_x of min_x_point form: stripe
        Remove stripe from points array

        Average the x-coordinates of stripe points: x_avrg
        Assign x_avrg to all stripe points
        Sort stripe points by y-coordinate

        If not going_right:
            Reverse the order of stripe points

        Toggle the value of going_right
        Append stripe points to path

    Return add_orientation_to_path(path, pcd)
```

The `zig_zag_pattern` function starts by defining a tolerance for the x coordinate and a radius for voxel downsampling. Voxel downsampling is a technique used to reduce the resolution of the point cloud by grouping nearby points into voxels, or 3D pixels, based on the specified radius [15]. These voxels generalize the neighborhood into which the sanding spindle should be driven in order to sand all the surrounding points.

With the point cloud simplified, the function proceeds to generate the zig-zag pattern. It initializes a loop that continues until all points in the downsampled point cloud have been processed. Within each iteration, the algorithm identifies the point with the minimum x coordinate, which serves as a reference for creating a `stripe`, a row of points. These stripes are formed by selecting points within a certain x coordinate tolerance from the reference point. This grouping creates a row parallel to the y axis. The procedure progresses from the closest points to the robot to the most distant ones.

Once a stripe is defined, its points are sorted based on their y coordinates, ensuring they are organized in a linear fashion from one end of the stripe to the other. The algorithm then checks the current direction of movement (left or right) and reverses the order of points in the stripe if necessary. This reversal is what creates the zig-zag pattern: as the path progresses, it alternates direction with each new stripe, mimicking the back-and-forth pattern of a zig-zag.

The array of coordinates must be added with orientations. This is done in the function `add_orientation_to_path`, which is described by the following pseudo-code:

```
Function add_orientation_to_path(Nx3 array: path, poinct cloud: pcd):
    Create a KD Tree from the pcd: pcd_tree
    Initialize: trafos with shape Nx4x4 filled with zeros

    For each point p in path:
        Find the closest point in pcd_tree to p and its normal vector n
        Normalize n

        Define a project vector x as [-1, 0, 0]
        Project x onto the plane defined by n to get x_proj
        Normalize x_proj

        Calculate y as the cross product of n and x_proj
        Normalize y

        Construct a rotation matrix R from x_proj, y, and n
        Assign R and p to the corresponding space in trafos
        Set the last diagonal cell of the matrix to 1

    Return trafos
```

This function's role is to assign an orientation to each point along the path, ensuring the robotic manipulator approaches each point with the correct alignment. The orientation is determined based on the normal vectors of the closest points in the original, high-resolution point cloud.

For each point in the path, the algorithm uses a KD Tree to efficiently find the closest point in the original point cloud [15]. The normal vector $\mathbf{n}$ of this closest point is then used as a basis for calculating the orientation. The process involves projecting a vector $\mathbf{x} = \begin{pmatrix} -1 & 0 & 0 \end{pmatrix}^T$ onto a plane defined by the normal vector:

$$\mathbf{x}_{proj} = \mathbf{x} - (\mathbf{x} \cdot \mathbf{n})\mathbf{n}. \tag{2}$$

Thus, it is guaranteed that the tool will point towards the robot at all points. The last component of the rotation is created by the dot product of $\mathbf{n}$ and $\mathbf{x}_{proj}$:

$$\mathbf{y} = \mathbf{n} \times \mathbf{x}_{proj}. \tag{3}$$

The rotation matrix $\mathbf{R}$ is formed by composing into a matrix these normalized vectors, which are orthogonal to each other [14]:

$$\mathbf{R} = \begin{pmatrix} \mathbf{x}_{proj} & \mathbf{y} & \mathbf{n} \end{pmatrix}. \tag{4}$$

This rotation matrix, along with the point's coordinates, forms a transformation matrix $\mathbf{T}$ that represents the manipulator's position and orientation in 3D space. These transformation matrices are compiled for each point along the path, providing a comprehensive guide for the robotic manipulator to follow the zig-zag pattern with proper orientation at each step.

This array of transformation matrices, as a designed path for the robot, is shown in Figure 3.4, and its zoom is in Figure 3.5, where the individual points are better visible. The individual coordinate systems are suitable for visualization because it is possible to see how the sanding spindle should be oriented. It can be seen that the tool (z axis) is always perpendicular to the object's surface, and the tool's x axis is always pointing towards the robot.

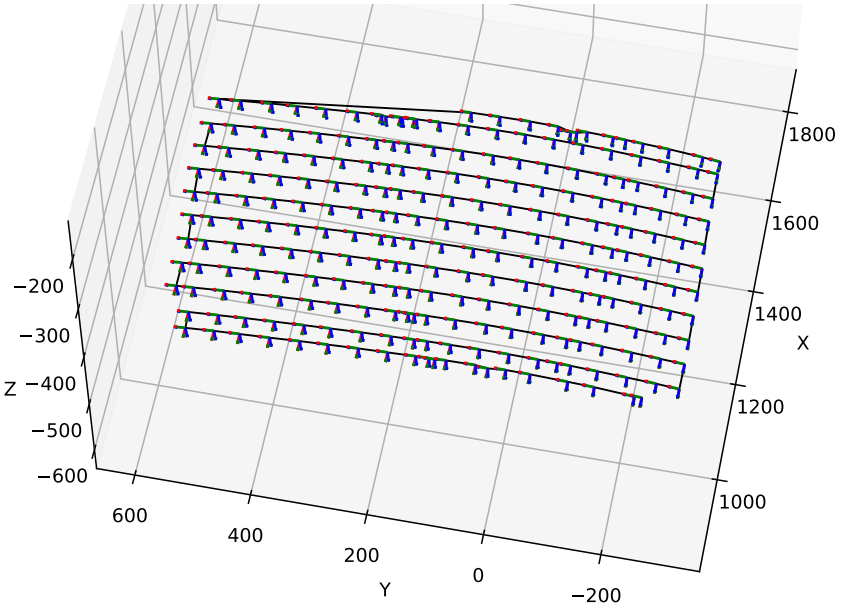

Figure 3.4. Generated path for robot as zig-zag pattern.

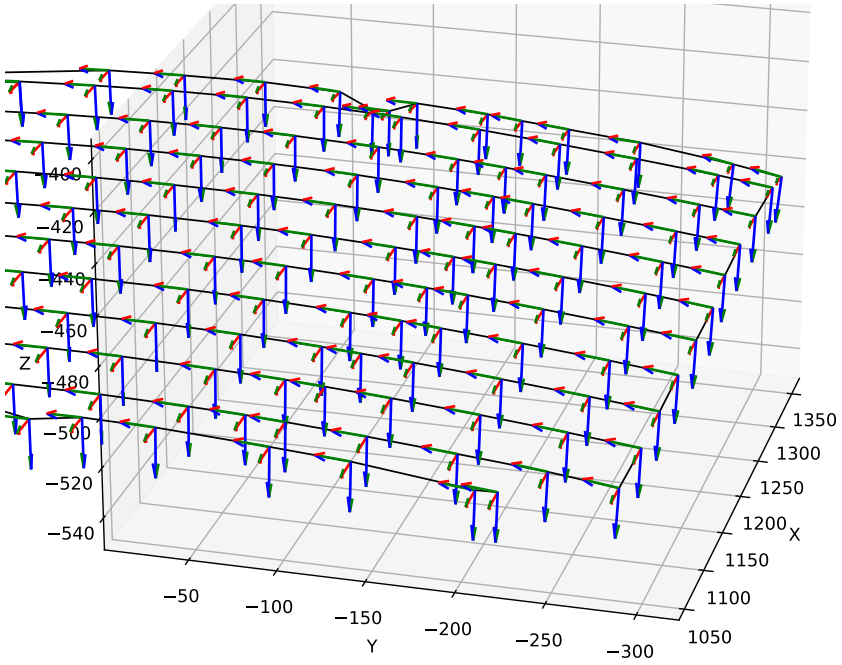

Figure 3.5. Zoomed path for robot as zig-zag pattern.

3.1.3 Spiral pattern creation

Another pattern that is suitable for robotic sanding or polishing is the spiral pattern, where the robot starts at the outer part of the object to be sanded and gradually spirals toward its center. The sequence of points to be traversed by the robot is first properly determined, and then the orientation of the tool is calculated. The algorithm consists of the main function `spiral_pattern` and auxiliary functions `distance_point_to_segment`, `is_point_near_hull`. First, the auxiliary functions are shown:

```
Function distance_point_to_segment(point: p, p1, p2):
```

```
    v = p2 - p1  // Line segment vector from p1 to p2
    w = p - p1   // Vector from p1 to point p

    // Project w onto v
    c1 = dot_product(w, v)
    If c1 is less or equal to 0:
        Return norm(p - p1)  // p projects before p1 on the line
    c2 = dot_product(v, v)
    If c2 is less or equal to c1:
        Return norm(p - p2)  // p projects beyond p2 on the line

    // Calculate the projection point p along the line segment
    b = c1 / c2
    pb = p1 + (b * v)  // Projection p on the segment

    Return norm(p - pb)
```

This function calculates the shortest distance from a point p to a line segment defined by two points, p_1 and p_2. It employs the projection of p onto the line segment and determines the closest point on the segment to p, returning the Euclidean distance between p and this closest point. The function handles boundary cases where the closest point on the line defined by p_1 and p_2 falls outside the segment, in which case the distance to the nearest endpoint (p_1 or p_2) is returned. The vector $\mathbf{v} = p_2 - p_1$ defines the line segment. The vector from p_1 to p is denoted as $\mathbf{w} = p - p_1$. The projection scalar is calculated: $c_1 = \mathbf{w} \cdot \mathbf{v}$, and the squared magnitude of $\mathbf{v}$ is calculated: $c_2 = \mathbf{v} \cdot \mathbf{v}$. The projection point p_b on the line is found using the scalar projection $b = c_2/c_1$, resulting in $p_b = p_1 + b \cdot \mathbf{v}$. The distance from p to the segment is the Euclidean distance between p and p_b: $Distance = \|p - p_b\|$.

The second help function is described here:

```
Function is_point_near_hull(point: p, hull_vertices, tolerance):
    For each vertex i in hull_vertices:
        p1 = hull_vertices[i]
        p2 = hull_vertices[(i + 1) % len(hull_vertices)] // Next vertex

        If distance_point_to_segment(p, p1, p2) is less than tolerance:
            Return True  // p is within tolerance distance of the hull

    Return False  // p is not near the hull
```

This function assesses whether a given point is within a specified tolerance distance from any edge of a convex hull defined by `hull_vertices` [16]. It iterates over each segment of the hull, utilizing `distance_point_to_segment` to compute the distance from the point to the current segment. If the distance is less than the specified tolerance, the function concludes that the point is near the hull. This determination is essential for identifying points that are close to the boundary of a layer in the spiral path generation process.

The previous function is used in the algorithm to create a spiral path for a robot from the point cloud. Its pseudo-code and detailed description is presented here:

```
Function spiral_pattern(pont cloud: pcd, radius: r):
    tolerance = r * 0.25
```

```
    voxel_radius = r * 0.5
    Downsample pcd using voxel_radius to create pcd_voxel_grid

    Convert pcd_voxel_grid points to a numpy array: points
    Initialize: path = []

    While there are at least 3 points remaining:
        project points to XY plane: points_xy
        Compute the convex hull of the points_xy
        Extract the boundary points: hull_vertices

        Initialize: layer_points = []
        For each point in the points_xy:
            If is_point_near_hull(point, hull_vertices, tolerance):
                Add the point to layer_points

        Remove layer_points from the points // use 3D points

        Sort layer_points clockwise relative to their centroid
        Append sorted layer_points to path // use 3D points

    Return add_orientation_to_path(path, pcd)
```

The `spiral_pattern` function is the core of the spiral path generation algorithm. Its arguments on the input are a point cloud `pcd` and a radius `r`, which influences the granularity of the voxel downsampling and the tolerance used for path generation. The process begins by converting the point cloud into a voxel grid, reducing the complexity of the cloud by merging nearby points into voxels.

The function then enters a loop that continues as long as there are enough points (at least 3) to form a convex hull. Within each iteration, it computes the convex hull of the remaining points, using these hull vertices to identify a `layer` of the spiral. Points near this hull, determined by the `is_point_near_hull` function, are considered part of the current layer. These points are then removed from the consideration for subsequent layers, effectively peeling the point cloud layer by layer.

Points in each layer are sorted in a clockwise direction based on their angles relative to the layer's centroid. This sorting is crucial for generating a spiral path that navigates around the layer in a coherent, continuous manner. When working with a convex hull, the points are projected into the XY plane. Only when sorted points are added to the path are their 3D equivalents added.

The sorting process organizes the points in a clockwise direction based on their angular positions relative to the centroid $C = (C_x, C_y)$ of the layer. It is computed as the arithmetic mean of the coordinates of all points in the layer:

$$C_x = \frac{1}{N}\sum_{i=1}^{N} x_i,\ C_y = \frac{1}{N}\sum_{i=1}^{N} y_i, \tag{5}$$

where $(x_i,\ y_i)$ are the coordinates of the i-th point in the layer, and N is the total number of points in the layer. For each point $p_i = (x_i,\ y_i)$ in the layer, its angular position θ_i relative to the centroid C is calculated using the $arctan2$ function, which is a variant of the $arctan$ function that considers the sign of both arguments to determine

the correct quadrant of the angle:

$$\theta_i = arctan2(y_i - C_y,\ x_i - C_x). \tag{6}$$

Once all angular positions θ are calculated, the points in the layer are sorted based on these values. This sorting arranges the points in a clockwise direction around the centroid.

After sorting, the points of the current layer are added to the path. This process repeats, layer by layer, until all points are incorporated into the path. Finally, orientations are added to each path point using the `add_orientation_to_path` function from the previous section. It is again obtained an array of transformation matrices, which is visualized in Figure 3.6, and its zoom is in Figure 3.7.

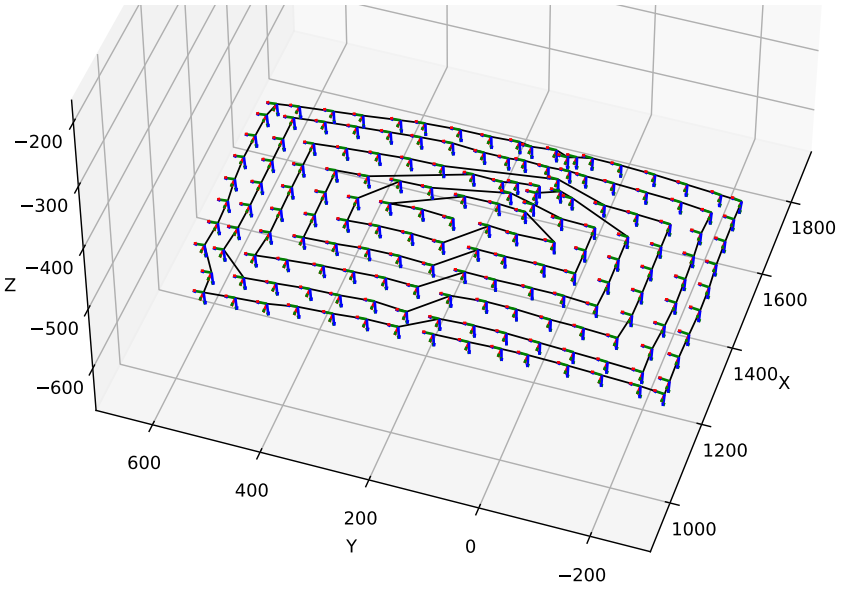

Figure 3.6. Generated path for robot as spiral pattern.

3.2 Controller of the sanding force

A force-torque (F/T) sensor is a device capable of measuring the forces and torques applied along multiple axes. In the context of industrial robots, F/T sensors are integral for tasks requiring precision, such as assembly, machining, and delicate manipulation. They provide robots with the necessary feedback to adjust their actions in real-time, enabling them to perform complex tasks with high precision and adaptability.

Force-torque sensors typically operate on the principle of strain measurement. They consist of a series of strain gauges arranged in a specific configuration. These strain gauges, which are essentially resistors whose resistance changes under the application of strain (deformation), are bonded to a flexible material or structure within the sensor. When an external force or torque is applied, it causes deformation in the sensor material, leading to a change in the resistance of the strain gauges. This change in resistance is then converted into a signal, which is processed to calculate the magnitude and direction of the applied forces and torques. There are six degrees of freedom along which force and torque can be measured: three linear axes (x, y, and z) for forces and three rotational axes (around x, y, and z) for torques [17].

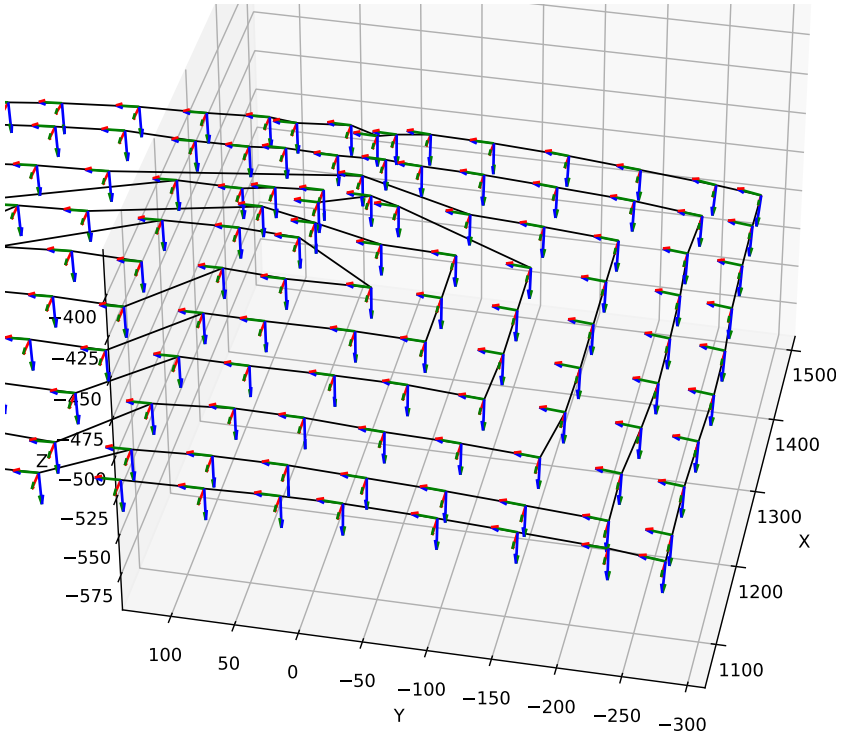

Figure 3.7. Zoomed path for robot as spiral pattern.

In picture 3.8, such a sensor is shown, and its coordinate system is also illustrated. The z axis is essential for the sanding application, which is the direction in which the end-effector is mounted to the F/T sensor. Both z axes are aligned to each other in this way. In this and all subsequent cases, the axes of the coordinate system are indicated in color: red, green, and blue for the x, y, and z axes. In the case of robotic sanding, only the measured forces on the axes are relevant because it is assumed that the robot runs at a constant speed during the sanding process, and its purpose is to maintain a constant pressing force.

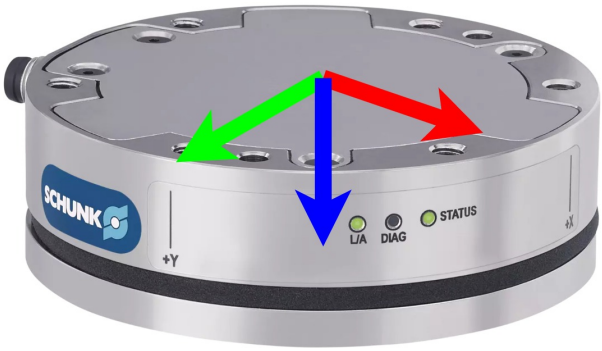

Figure 3.8. The coordinate system of a force-torque sensor.

3.2.1 Sensor Gravity compensation

An essential aspect of working with force-torque sensors is compensating for the effects of gravity. Gravity compensation is necessary because the weight of the sensor itself and any attached tooling or payloads can introduce constant bias in the measurements,

affecting the accuracy of force and torque readings. Gravity compensation involves subtracting the gravitational forces from the sensor readings [18].

This can be achieved through calibration processes where the sensor is oriented in various known positions to measure the gravitational effects. These measurements are then used to mathematically model the gravitational bias, which can be subtracted from subsequent readings to obtain accurate force and torque measurements.

The calibration of the sensor is performed by moving the robot with the attached force-torque sensor and the tool to positions where only one axis is actually measuring, and the other two are not active and measure zero force. In addition, the forces in both the positive and negative directions of sensitivity are measured for each axis. Thus, six values are measured: F_x^+, F_x^-, F_y^+, F_y^-, F_z^+ and F_z^-. These are the forces on each axis in both positive and negative directions. The offset compensation is based on the assumption that for an ideal sensor it holds $(F^+ + F^-)/2 = 0$. However, for a real sensor, this is not true, so it is necessary to find an offset that corrects this to zero. The individual offsets on the axes are obtained as follows:

$$offset_x = \frac{F_x^+ + F_x^-}{2},\ offset_y = \frac{F_y^+ + F_y^-}{2},\ offset_z = \frac{F_z^+ + F_z^-}{2}. \tag{7}$$

These offset values must be subtracted from the raw force-torque sensor measurements.

However, it is also required to ensure that the total force acting on the sensor at a standstill is zero. Again, these six measuring positions are traversed, this time with compensated offsets, and the individual forces are logged. The absolute value is taken, and the arithmetic mean is calculated:

$$offset_{norm} = \frac{|F_x^+| + \ldots + |F_z^-|}{6}. \tag{8}$$

Afterward, the magnitude of the force applied on the force-torque sensor is zero in arbitrary position of the tool. This compensates for the gravitational force being applied to the tool and the force-torque sensor. In the sanding process, the force vector norm of all three components must first be calculated, and then $offset_{norm}$ must be subtracted from it to obtain the real force that the sanding spindle is pressing on the object.

Before logging values from the force-torque sensor, it is useful to wait for a little while until the robot's movement stops and values have stabilized.

3.2.2 Controller for applying a constant pressing force

The purpose of this controller is to hold a constant force ($Reference$) for the whole time the robot is sanding the specified object. It can happen that some defect occurs on the object, which is not accented in the path planning, and at the same time, it controls the force between the points to be traversed by the robot. It should run at the highest possible frequency to achieve the best results. A diagram of the entire control loop is shown in Figure 3.9.

The control loop receives the current values of the forces on the individual axes of the force-torque sensor ($Raw\ F_x$, F_y, F_z). From these raw values the corresponding offsets are subtracted, which were computed in the previous section and are stored directly in the robot controller. From the values of the forces on the axes, the magnitude of the total force vector is calculated: $F = \sqrt{F_x^2 + F_y^2 + F_z^2}$. Even if it is assumed that the force-torque sensor will be active only in the z axis direction, all axes are used

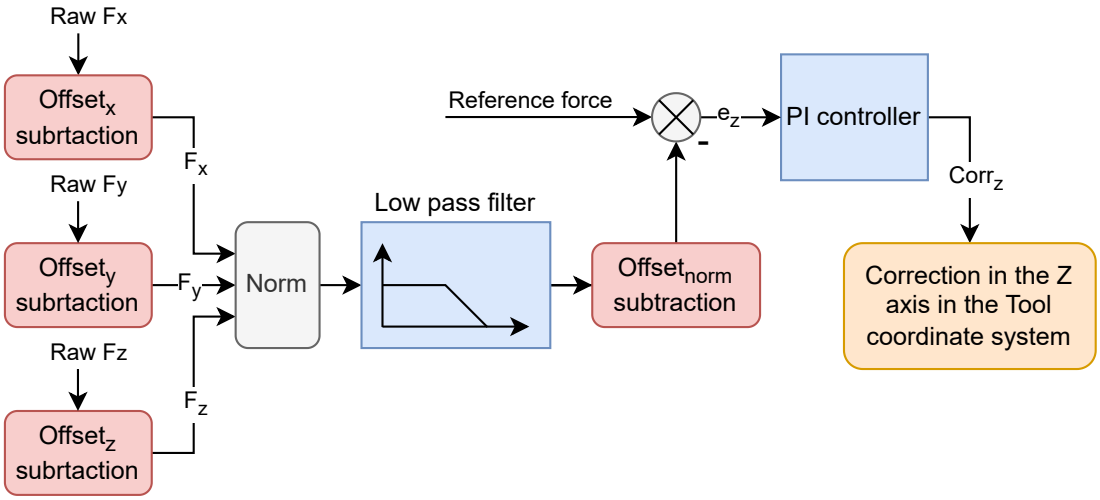

Figure 3.9. Controller for keeping constant pressing force on an object.

to calculate the force vector, especially to avoid inaccuracies in the gravitational force offset. In addition, it is useful for safety. If a force is detected on the x or y axis of the sensor that is not accounted for, the robot will react by moving away from the workpiece.

The magnitude of the force is then filtered by a low pass filter. The output of the force-torque sensor is noisy, and the controller would react too wildly. In a concrete implementation, an Infinite Impulse Response filter with a Bessel second-order function is used [19]. After this signal smoothing, the contribution of the gravitational force ($Offset_{norm}$) is subtracted from the force.

The modified input signal in the form of the force applied to the workpiece is subtracted from the reference force, which is requested by the operator to be constant throughout the sanding process, i.e. it is a reference tracking task. The subtraction of the forces produces the error e_z, which the controller should keep at zero. This error is used as an input to the `PI` controller [20]. It is chosen for the reason that it converts the force magnitude proportionally to the position and, in addition, keeps the information about the error from the past due to the integration component. The output of the controller is therefore the $Corr_z$ magnitude, which is the value by which it should be shifted in the z axis in the end-effector coordinate system. Kuka robots can work directly with this value and can incorporate it into the robot's motion [21]. The controller is set up so that when the error is greater than zero, the robot approaches the object. This means that it is not applying enough force to the object, and its z tool coordinates must be decremented. Conversely, when e_z is less than zero, the robot applies too much force to the workpiece and has to move away from it.

The concrete results of the controllers are in the section 4.6. The specific values of the `PI` controller and the time sequence of the pressing force, when it is controlled in the feedback loop, are given there.

3.3 Forward and Inverse kinematic task for a 6-DOF robotic manipulator

For the automated generation of robot paths, the path generation algorithm needs to know the kinematics of the robot. Two kinematic tasks are defined: a forward and

an inverse task. These mathematical frameworks are essential for understanding and programming the movements of a robot to achieve desired positions and orientations in space. In the context of robotic manipulators, kinematics focuses on the relationship between the joint parameters (such as angles for revolute joints or displacements for prismatic joints) and the position and orientation of the robot's end-effector [22].

In this application, a robotic manipulator with six revolute joints is used, and a sanding spindle is attached to the flange as an end-effector. There are two possibilities how to assign a tool to the robot. The first is that the coordinate system of the flange and the tool is offset only in the z axis. The second option is to attach the tool perpendicular to the flange.

The first option is shown in Figure 3.10. From the image, it is evident that the sixth axis is not used since the sanding spindle is symmetrical about the z axis and, therefore, does not depend on the rotation of the axis 6. However, the advantage is that it is a more straightforward mathematical description of such a system, and the end-effector is only offset in the z axis from the flange so that the inverse kinematic task can be solved analytically quite easily. Another advantage is the simple force control. It is clearly determined that the z axis of the force-torque sensor is always perpendicular to the surface and a simple force reading is ensured. Figure 3.10 further shows all significant coordinate systems (csys) on the robot and the tool.

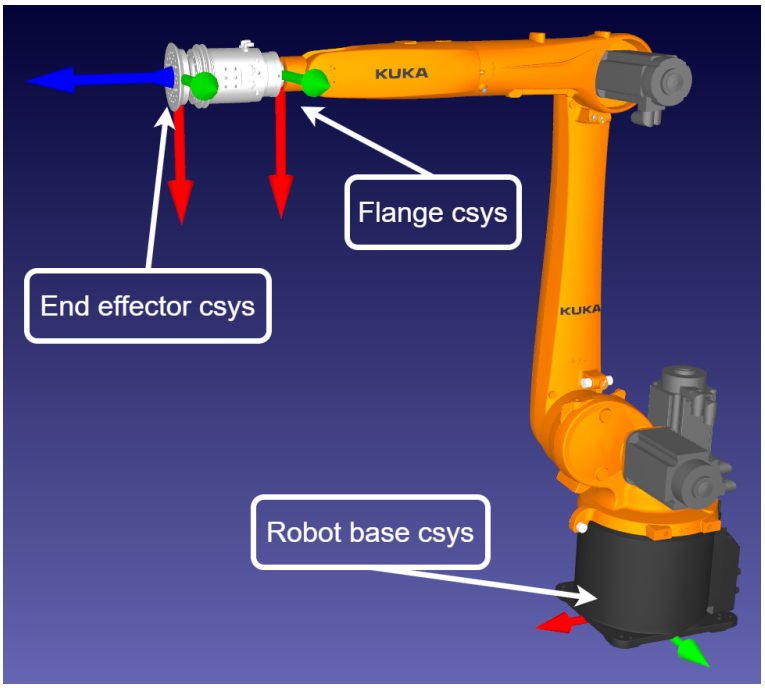

Figure 3.10. Kinematic system with parallel attachment of the tool to the robot flange offset only in the z axis.

Picture 3.11 shows a more complex mounting of the tool to the flange. Its disadvantages are both a more complex description of the kinematics and a more complex task of correcting the robot position with respect to the force-torque sensor measurement. The main advantage is the versatility in angling. Mounting the spindle perpendicular to the flange allows the robot to approach the workpiece from various angles more easily, enhancing the versatility of sanding different surface orientations. Typically, when traversing an edge, it is sufficient to apply axis 6. However, with the previous kinematics, all other axes would have to be engaged, and their joint speeds would increase

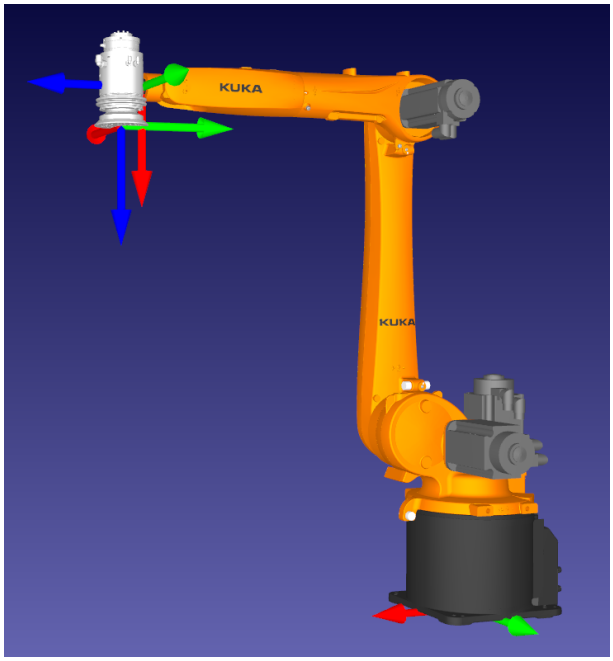

Figure 3.11. Kinematic system with perpendicular attachment of the tool to the robot flange.

significantly to achieve a constant tool velocity, which significantly increases the robot's power consumption.

The first approach was chosen because of the simplicity of the description of both the kinematic model of the system and the measurements on the force-torque sensor. However, it would be interesting to modify the system to the second case in future work because it can provide greater variability in robotic sanding.

3.3.1 Forward kinematic task

The objective of forward kinematics is to compute the position and orientation of the robot's end-effector from its joint parameters (angles for revolute joints) [22]. This task is crucial as it allows the control system to determine where the end-effector will be positioned in the workspace after certain joint movements.

A robotic arm with 6 joints, described by a set of joint parameters $\mathbf{q} = (q_1, \ldots, q_6)$, should be considered, where each q_i represents the joint angle for revolute joints or joint displacement for prismatic joints. The goal is to compute the transformation matrix $\mathbf{T}$ that maps these joint parameters to the pose of the end-effector. Matrix $\mathbf{T}$ contains translations between the robot's origin and TCP (Tool center point), as well as the rotation of the tool.

The Denavit-Hartenberg (DH) convention is used to obtain the matrix $\mathbf{T}$. The DH notation is a standardized method to systematically represent the kinematic chains of robots. It simplifies the mathematical modeling of robotic arms by using a minimal set of parameters. It enables the calculation of transformation matrices between consecutive joint coordinate frames. The parameters θ_i, d_i, a_i, and α_i are defined as follows for each joint i from 1 to 6 in a 6-DOF robot:

- θ_i: The joint angle, which is the rotation around the z axis of the previous joint's frame to align the x axis with the common normal.

- d_i: The link offset, which is the distance along the previous z axis to the common normal. This is effectively the translation along the z axis of the previous joint's frame.
- a_i: The link length, which is the distance from the previous z axis to the current z axis, measured along the common normal (x axis).
- α_i: The link rotation, which is the angle from the previous z axis to the current z axis, measured around the common normal (x axis).

In DH notation, each joint in the robotic manipulator is associated with a coordinate frame. The frames are set as follows:

- The z axis of each frame is aligned with the axis of the joint (axis of motion).
- The x axis is aligned along the common normal between the current z axis and the next z axis. If the axes are parallel, the common normal that provides the shortest distance is selected.
- The origin of each frame is placed at the intersection of the x axis with the z axis of the previous joint.

Each joint i contributes a transformation $\mathbf{T_i}$ based on its parameters using the DH convention. The final transformation is the transformation between the robot flange and TCP. The resulting matrix is then obtained as

$$\mathbf{T} = \mathbf{T_1} \cdot \ldots \cdot \mathbf{T_6} \cdot \mathbf{T_{TCP}}. \tag{9}$$

Each joint i contributes a transformation $\mathbf{T_i}$ based on its parameters using the DH convention. This local transformation from joint i to joint $i+1$ is given by:

$$\mathbf{T_i} = \mathbf{Rot_z}(\theta_i) \cdot \mathbf{Transl_z}(d_i) \cdot \mathbf{Transl_x}(a_i) \cdot \mathbf{Rot_x}(\alpha_i), \tag{10}$$

where **Rot** denotes the matrix of rotation, either about the z axis or about the x axis and **Transl** denotes the matrix of translation in the z axis or in the x axis. These matrices are as follows:

$$\mathbf{Rot_z}(\theta_i) = \begin{pmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{Transl_z}(d_i) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{Transl_x}(a_i) = \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{Rot_x}(\alpha_i) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

After multiplying these matrices the matrix $\mathbf{T_i}$ is obtained[22]:

$$\mathbf{T_i} = \begin{pmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & a_i cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_i) & a_i sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{11}$$

It is important to mention that these θ_i are de facto θ_i offsets because if the axis coordinates $\mathbf{q}$ are taken into account in the DH notation, they are added to these theta parameters ($\theta_i = \theta_i + q_i$). Therefore, the z axis must always be an axis of motion.

Since the process of constructing DH parameters depends on the specific dimensions of the robot, it is not meant to present some general process of how to get DH parameters. However, it is preferable to show a concrete solution as an example. There may

even be more valid solutions to describe a robot using DH parameters. The solution presented is for a robot with a sanding tool on which this thesis was developed.

The manufacturer's datasheet (3.12) is used to determine the DH parameters, where the dimensions of the robot are given. In the process of acquiring the DH parameter, it is simply necessary to obey the rules presented in the previous paragraphs. Then the solution may look like this:

	θ_i $[rad]$	d_i $[mm]$	a_i $[mm]$	α_i $[rad]$
$\mathbf{T_1}$	0	450	0	π
$\mathbf{T_2}$	0	0	150	$\frac{\pi}{2}$
$\mathbf{T_3}$	0	0	810	0
$\mathbf{T_4}$	$\frac{\pi}{2}$	0	-20	$-\frac{\pi}{2}$
$\mathbf{T_5}$	0	-660	0	$\frac{\pi}{2}$
$\mathbf{T_6}$	0	0	0	$-\frac{\pi}{2}$
$\mathbf{T_{TCP}}$	0	-290	0	π

Table 3.1. DH parameters of the robot at the workplace.

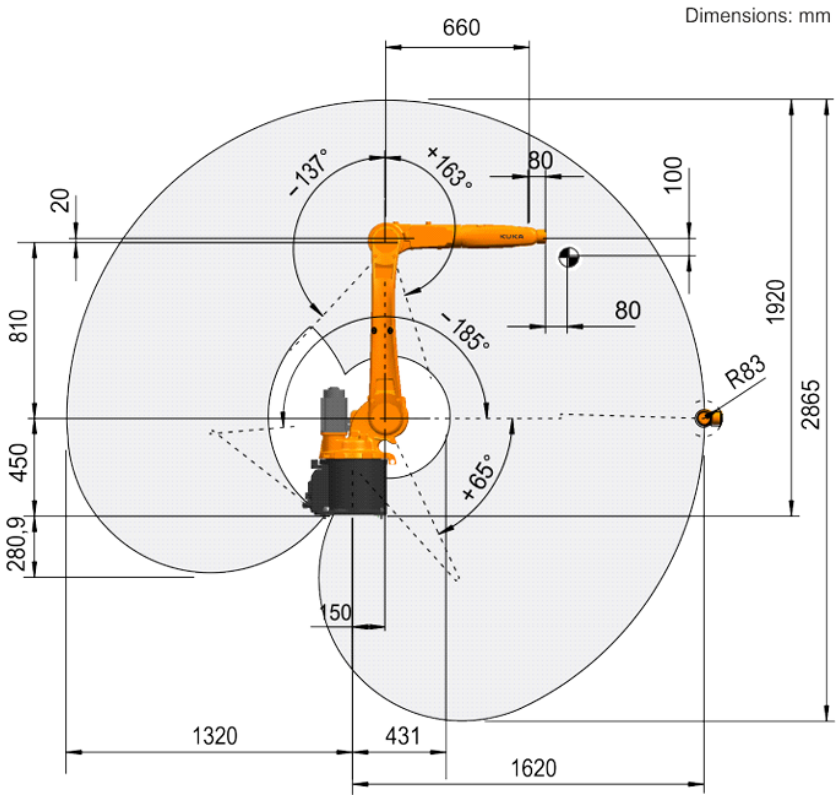

Figure 3.12. Dimensions of the robot in the workplace [23].

It is essential to mention that the offset of the tool from the flange, which is 210 mm, is already taken into account in this case. Furthermore, the robot in Figure 3.12 does not have all the axis coordinates set to zero, but $q_2 = -\frac{\pi}{2}$ rad and $q_3 = \frac{\pi}{2}$ rad.

The process of obtaining the DH parameters can be clarified in Figure 3.13, which shows the coordinate systems of all robot axes. Also the joint coordinates of axes 2 and 3 are added to appear as in the datasheet image.

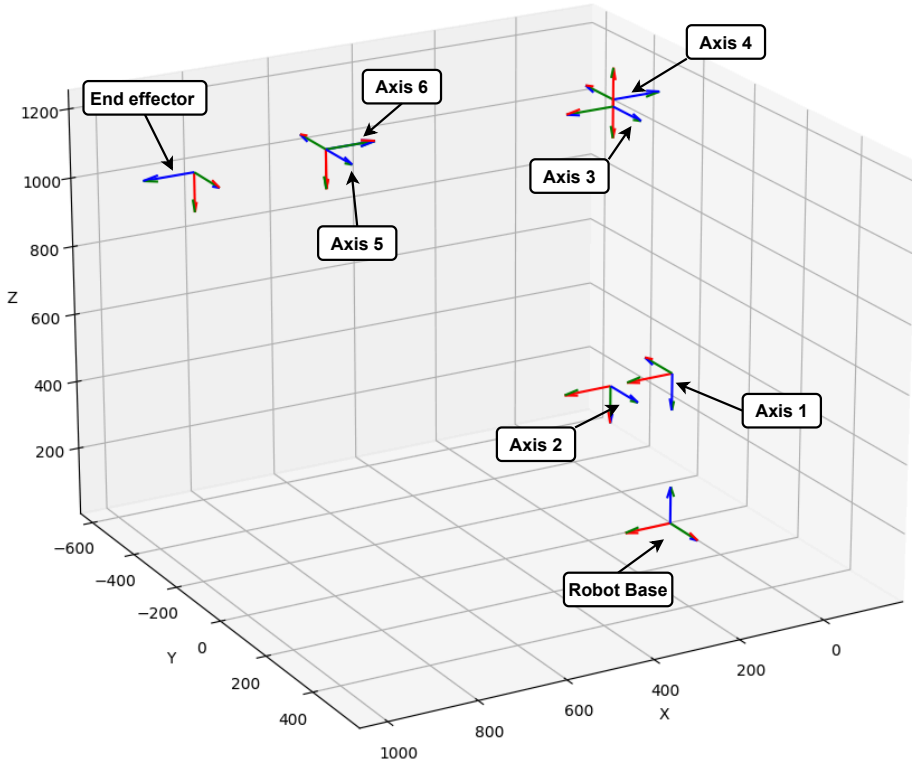

Figure 3.13. Coordinate systems of all robot axes and robot base and end-effector coordinate frame.

Thus, a mathematical description of the kinematics of the tool robot was obtained, where from an arbitrary configuration of the joint coordinates $\mathbf{q}$, the position and rotation of the tool in the robot base can be obtained.

3.3.2 Inverse kinematic task

The inverse kinematic task (IK) is, as the name suggests, the opposite of the forward kinematic task. It solves the problem of determining the configuration of the robot from the known position and orientation of the tool. The position and orientation of the end-effector are given as a transformation matrix $\mathbf{T}$, which should result in the robot's configuration as a sequence of joint coordinates $\mathbf{q}$ [22].

The inverse kinematics problem is generally more complex than forward kinematics due to its nature:

- Non-linearity and high degrees of freedom: Most robotic systems have multiple joints and links, leading to non-linear equations with multiple possible solutions, or sometimes no solution at all.
- Redundancy: A robot with more joints than necessary for a task (redundant degrees of freedom) can have a multiple or an infinite number of solutions. Choosing the optimal solution among them based on criteria like minimizing energy use, avoiding joint limits, or avoiding singular configurations is non-trivial.
- Existence and Uniqueness: Not all desired end-effector positions and orientations may be achievable due to physical constraints, and some positions may be achievable in multiple ways [22].

Various methods have been developed to solve the IK problem, each with its advantages and limitations:

- Analytical solutions: These are closed-form solutions derived using algebraic methods, providing exact joint configurations for given end-effector poses. Analytical methods are fast and precise but only feasible for simpler robotic structures or specific configurations due to the mathematical complexity.
- Numerical solutions: When analytical solutions are not feasible, numerical methods such as iterative algorithms are used. These methods approximate the solution by minimizing the difference between the desired and the actual end-effector position and orientation. Numerical methods can handle more complex robotic structures and are more flexible but can be slower and may not always converge to a solution.
- Optimization-based methods: These involve formulating the IK problem as an optimization problem where an objective function is minimized. This function could include terms for the distance from the desired position, joint limits, obstacle avoidance, and other criteria.
- Heuristic and Learning-Based Approaches: Heuristics such or techniques like Artificial Neural Networks and Deep Learning are used, especially when dealing with highly complex or poorly defined systems [24].

The system of a six-axis robot with all the joints revolute 3.12 is not so complex that it cannot be solved analytically, therefore it is possible to construct algebraic equations for solving IK. The procedure is demonstrated again on a concrete robot since the system of equations again depends on the construction of the specific robot and cannot be well generalized.

Now let consider the concrete analytical solution of IK. The problem will be divided into two parts. In the first part, the possible values of the first three joints (q_1, q_2, q_3) are found. These determine the position of the `wrist`, which is the intersection of motion axes $A4$, $A5$, $A6$ and is responsible for the position of the end-effector. In figure 3.13, it is shown as the origin of axis 5 ($A5$) or axis 6 ($A6$). In the second part, the values of the joint axes $A4$, $A5$, $A6$ are obtained (q_4, q_5, q_6). They are responsible for the orientation of the tool. There is the procedure:

1. The wrist position is calculated by subtracting the effect of the last joint's length from the tool's position, considering the orientation. So it is calculated according to the equation
$$x_{wrist} = x_{tool} + d_{TCP} \cdot \mathbf{R}[:, 2], \tag{12}$$
where x_{tool} and $\mathbf{R}$ are parts of the transformation matrix $\mathbf{T}$ that defines the position and orientation of the end-effector and d_{TCP} is the value of the DH parameter from Table 3.1. $\mathbf{R}[:, 2]$ indicates that only the last column of the rotation matrix is used, i.e. the direction of the z axis in the orientation of the end-effector. Normally $d_{TCP} \cdot \mathbf{R}[:, 2]$ should be subtracted from x_{tool}, but d_{TCP} is already in DH notation with a negative sign. It is essential to note that it is possible to reach the position of the wrist in up to four different configurations.
2. Joint 1 (Base Rotation): Joint coordinate of axis 1 (q_1) is derived from the x, and y coordinates of the wrist relative to the robot base, using $arctan2$ for robust angle calculation:
$$q_1 = -arctan2(x_{wrist,y}, x_{wrist,x}) \tag{13}$$
The second possible configuration of the robot is adding 180° to the q_1. Figure 3.14 visualizes the dependence of q_1 on the wrist position.
3. Joint 2 and 3 (Shoulder and Elbow): Finding the values of q_2 and q_3 is more difficult in the sense that suitable trigonometric relationships formed by the robot's shoulder, elbow, and wrist joint must be found. The sketch in Figure 3.15 serves for a better understanding of trigonometric dependencies. Variable r accounts for

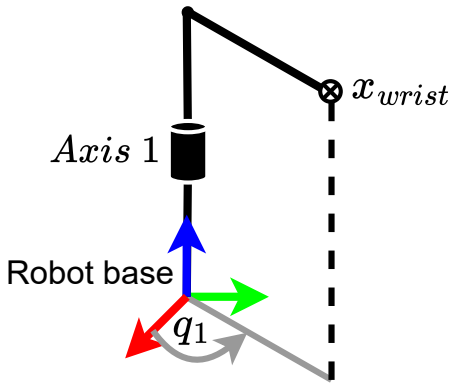

Figure 3.14. Visualization of how the position of the wrist is dependent on the q_1 value of axis 1.

the reach from the base to where the wrist would be horizontally, subtracting a constant horizontal offset represented by a_2 from DH notation:

$$r = \sqrt{x_{wrist,x}^2 + x_{wrist,y}^2} - a_2. \tag{14}$$

The variable s quantifies the straight-line distance from the shoulder joint to the wrist position. It is calculated as the hypotenuse of the triangle formed by the vertical displacement from the base to the wrist (along the z axis, offset by d_1) and the radial distance r:

$$s = \sqrt{(x_{wrist,z} - d_1)^2 + r^2}. \tag{15}$$

The variable l represents the effective length of the robot's arm from the elbow to the wrist. It's the hypotenuse of a right triangle formed by a_4 and d_5:

$$l = \sqrt{a_4^2 + d_5^2}. \tag{16}$$

The angle α is derived from the triangle formed by the vertical and horizontal distances from the base (or shoulder) to the wrist:

$$\alpha = arctan2(x_{wrist,z} - d_1, r). \tag{17}$$

The angle β represents the elbow angle and is determined using the law of cosines within the triangle formed by the arm segments and the line from the shoulder to the wrist (distance s). The law of cosines relates these three sides of the triangle to the angle opposite l:

$$\beta = arccos\left(\frac{a_3^2 + s^2 - l^2}{2a_3 s}\right). \tag{18}$$

This equation uses again the law of cosines to calculate the angle between the upper arm a_3 and the hypotenuse l:

$$\gamma = arccos\left(\frac{a_3^2 + l^2 - s^2}{2a_3 l}\right). \tag{19}$$

δ is the angle of offsets in the joint connection, particularly from the forearm a_4 to the wrist d_5:

$$\delta = arctan2(d_5, a_4). \tag{20}$$

By this all angles necessary to calculate the articulated coordinates q_2 and q_3 are sufficiently described. The angle q_2 (Shoulder joint angle) is primarily influenced by α and β. It is designed to lift and rotate the arm's shoulder joint to align the arm appropriately to reach the wrist position from the shoulder pivot point:

$$q_2 = -(\alpha + \beta). \tag{21}$$

The angle q_3 (Elbow joint angle) connects the upper arm to the forearm and positions them so that the wrist can be placed correctly, and it is calculated as follows:

$$q_3 = \frac{\pi}{2} - (\gamma + \delta), \tag{22}$$

where the term $\frac{\pi}{2}$ is used to normalize the angle q_3, to start from a default position according to the Figure 3.15.

For the second configuration of the shoulder, (q_2) and elbow (q_3) joints in a robotic arm is also considered an alternative kinematic solutions that involve mirroring the elbow's position relative to the shoulder, often referred to as `elbow up` vs. `elbow down` configurations in robotic terminology. This approach allows the arm to reach the same end-effector with the elbow positioned differently. The equations for the angles will be slightly different, but the essence of the composition of the equations is the same. In combination with the two axis 1 configurations, a total of four configurations are achieved in which the robot can reach the wrist position [22].

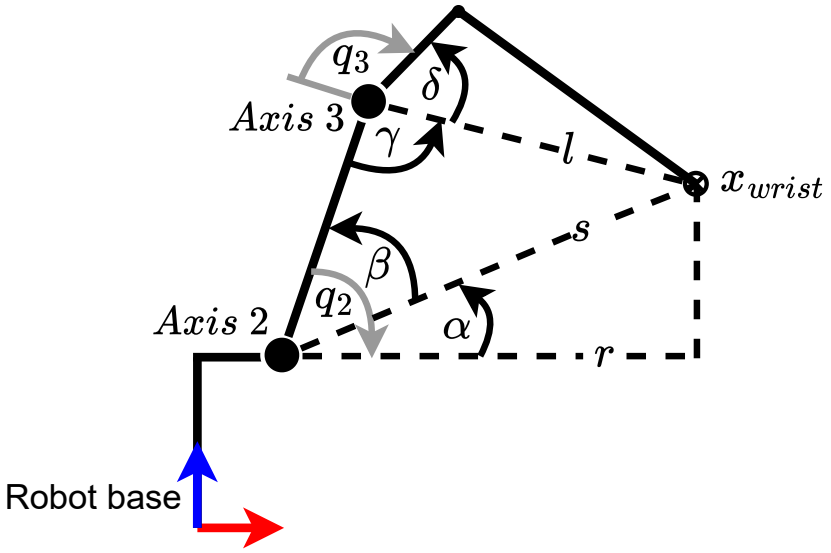

Figure 3.15. Visualization of how the position of the wrist is dependent on the q_2 and the q_3 values.

4. When all the configurations of the wrist are determined, it is possible to proceed to the determination of the joint coordinates q_4, q_5, and q_6, which control the orientation of the wrist and, ultimately, the end-effector. These angles are crucial for ensuring that the end-effector aligns correctly with its desired target orientation.

   It is based on the principle that the resulting rotation matrix $\mathbf{R}$ can be divided into two parts, that is,

$$\mathbf{R} = \mathbf{R}_{1-3} \cdot \mathbf{R}_{4-6}, \tag{23}$$

where $\mathbf{R}_{1-3}$ depends on the contributions of the first three axes (A1, A2, A3), and the $\mathbf{R}_{4-6}$ matrix needs to be further calculated. It can be obtained from the following equation:

$$\mathbf{R}_{4-6} = \mathbf{R}_{1-3}^{T} \cdot \mathbf{R}, \tag{24}$$

taking advantage of the fact that the inverse of a rotation matrix is its transpose because it is an orthogonal matrix.

For computing $\mathbf{R}_{1-3}$ it can be used the robot's kinematic chain (Denavit-Hartenberg parameters) to calculate the rotation matrix from the base frame to the third joint's frame. This is done using FK up to joint 3: $\mathbf{R}_{1-3} = \mathbf{R}_1 \cdot \mathbf{R}_2 \cdot \mathbf{R}_3$. Each $\mathbf{R}_i$ is a rotation matrix that depends on the corresponding q_i. These can be easily obtained from equation (11) if only the first three rows and columns are taken into account.

Once the matrix $\mathbf{R}_{1-3}$ and the rotational matrix $\mathbf{R}$ are known, the concrete matrix $\mathbf{R}_{4-6}$ is calculated, from which the joint coordinates q_4, q_5 and q_6 must be extracted. The equations for them are constructed by comparing the symbolic solution of the matrix $\mathbf{R}_{4-6}$ and the numerical solution calculated. Again, the first three rows of the three columns of equation (11) are taken, where values from Table 3.1 are substituted for α_i values, and the values of θ_i are left symbolic:

$$\mathbf{R_4} = \begin{pmatrix} cos(\theta_4) & 0 & sin(\theta_4) \\ sin(\theta_4) & 0 & -cos(\theta_4) \\ 0 & 1 & 0 \end{pmatrix}, \ \mathbf{R_5} = \begin{pmatrix} cos(\theta_5) & 0 & -sin(\theta_5) \\ sin(\theta_5) & 0 & cos(\theta_5) \\ 0 & -1 & 0 \end{pmatrix},$$

$$\mathbf{R_6} = \begin{pmatrix} cos(\theta_6) & sin(\theta_6) & 0 \\ sin(\theta_6) & -cos(\theta_6) & 0 \\ 0 & 0 & -1 \end{pmatrix}.$$

Now, these matrices are multiplied between themselves: $\mathbf{R}_{4-6} = \mathbf{R_4} \cdot \mathbf{R_5} \cdot \mathbf{R_6}$. This results in the following symbolic matrix (For excessive matrix size, the *sine* is written only as s and the *cosine* as c):

$$\mathbf{R}_{4-6} = \begin{pmatrix} c(\theta_4)c(\theta_5)c(\theta_6) - s(\theta_4)s(\theta_6) & c(\theta_4)c(\theta_5)s(\theta_6) + s(\theta_4)c(\theta_6) & c(\theta_4)s(\theta_5) \\ s(\theta_4)c(\theta_5)c(\theta_6) + c(\theta_4)s(\theta_6) & s(\theta_4)c(\theta_5)s(\theta_6) - c(\theta_4)c(\theta_6) & s(\theta_4)s(\theta_5) \\ s(\theta_5)c(\theta_6) & s(\theta_5)s(\theta_6) & -c(\theta_5) \end{pmatrix}. \tag{25}$$

The matrix $\mathbf{R}_{4-6}$ now contains the information needed to set axes 4, 5, and 6 to achieve the required orientation, especially the third row and the third column. The actual angles q_4, q_5, and q_6 are derived from $\mathbf{R}_{4-6}$:

$$q_4 = arccos(\mathbf{R}_{4-6}[2,2]), \tag{26}$$

$$q_5 = arctan2(\mathbf{R}_{4-6}[1,2], \mathbf{R}_{4-6}[0,2]), \tag{27}$$

$$q_6 = arctan2(\mathbf{R}_{4-6}[2,1], \mathbf{R}_{4-6}[2,0]), \tag{28}$$

where the elements of the matrix $\mathbf{R}_{4-6}$ are numbered as $\mathbf{R}_{4-6}[row, column]$ and the indices start from zero.

In some cases, the calculation might hit a singularity, where certain orientations can't be uniquely determined (*gimbal lock problem*). This happens when $\mathbf{R}_{4-6}[2,2] = 1$ and thus θ_5 is in the zero position, then the axes 4 and 6 are simultaneously eclipsed and thus there are infinitely many solutions for q_4 and q_6. It can be solved by arbitrarily setting one of the axes and calculating the remaining one [22].

Even in this case, there are two possible configurations of axes 4, 5, and 6. If $q_4 \geq 0°$, there is a second solution where $q'_4 = q_4 - 180°$, $q'_5 = -q_5$ and $q'_6 = q_6 - 180°$. Otherwise, when $q4 < 0°$, the second solution is $q'_4 = q_4 + 180°$, $q'_5 = -q_5$ and $q'_6 = q_6 + 180°$. In the general case we get eight possible configurations of the robot as an IK solution.

5. This could be considered a basic robot configuration. It is necessary to pay attention to the limits of the axes. It could happen that in the process of IK computation, some joint coordinates could fall out of their interval. Such a solution must be marked as invalid. The robot in picture 3.12 has the following limitation on the axes [23]:

$$q_1 = [-170°, 170°],\ q_2 = [-185°, 65°],\ q_3 = [-137°, 163°],$$
$$q_4 = [-185°, 185°],\ q_5 = [-120°, 120°],\ q_6 = [-350°, 350°].$$

From the limits of the joint coordinates it can be seen that axes 4 and 6 have a larger interval than $\pm 180°$. This may add another possibility to each solution. To give an example, if the joint coordinate is in the interval $q_4 \in [175, 180]$, the exact same configuration is achieved with $q_4 \in [-185, -180]$.

3.4 OPC UA communication protocol

Open Platform Communications Unified Architecture (OPC UA) is a communication protocol developed by the OPC Foundation [25], designed for industrial automation and other control systems. It provides a robust, secure, and scalable framework for information exchange in the manufacturing industry, and is integral in the drive towards the Internet of Things (IoT) and Industry 4.0.

3.4.1 Overview of OPC UA

OPC UA is built around a client-server architecture, where the server exposes objects that clients can access. The architecture is comprised of several layers:

- Abstract Interface Layer: Defines the high-level interface independent of implementation details.
- Services Layer: Where the core functionality of OPC UA is implemented, handling data modeling, discovery, access, and historizing services.
- Communication Layer: Ensures secure and reliable messaging between clients and servers.
- Transport Layer: Defines the protocols used for transmission of messages, typically using TCP/IP, HTTP, or other protocol bindings.

The data modeling capabilities of OPC UA allow for the detailed and structured representation of real-world processes. The basic building blocks of the OPC UA data model are `nodes`, which represent variables, data types, and methods. These nodes' `attributes`, such as value, status, and timestamps, describe their properties and state. Nodes are interlinked through `references`, enabling the creation of complex data structures.

OPC UA supports multiple types of communications. `Client-server` is the traditional model where clients request, and servers respond. The second option is the `Pub/Sub (publish-subscribe)` model, which is a newer addition where data is broadcasted by publishers and received by subscribers, suitable for high-speed and large-scale communications.

Security is a cornerstone of OPC UA, designed to address authentication, confidentiality, and integrity:

- Authentication: Ensures that only authorized clients can access the server. It supports certificates and user/password mechanisms.
- Encryption: Secures data against unauthorized access using algorithms such as RSA (Rivest–Shamir–Adleman) and AES (Advanced Encryption Standard).
- Data Integrity: Utilizes digital signatures to ensure data has not been tampered with during transmission [26].

3.4.2 Utilization of OPC UA at the sanding workplace

The OPC UA is used in two cases at the sanding robot workplace. After the paths are generated by the algorithm in the application computer, an array of positions is sent from the PC to the server, which is the robot controller. It works by overwriting the already initialized variable in the `dat` files. Any variable or array can be there overwritten, but the OPC UA server no longer has access to the `src` files. The second part of the communication is between the robot and the AR headset. The AR headset reads the paths from the server in the robot so that they can be visualized in the headset. Furthermore, the headset writes to a variables in the robot's controller that indicates that the robot should start moving and traverse the path the operator selects.

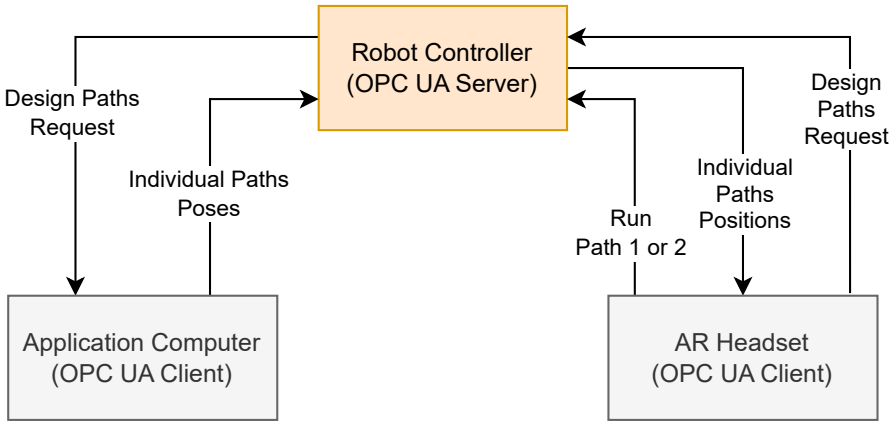

Figure 3.16. Diagram of the use of communication at the workplace via OPC UA.

The communication between the robot and the application computer is introduced first. This part is programmed in Python. In the beginning, the connection must be established. An OPC UA client is created and connected to an OPC UA server using the specified URL:

```
(opc.tcp://OpcUaOperator:kuka@xxx.xxx.xxx.xxx:xxxx/) // Replace
// 'xxx' with actual IP address and 'xxxx' with actual port number
```

This URL includes the protocol type (`opc.tcp`), user credentials (`OpcUaOperator:kuka`), and server address with port (for example: `10.100.0.114:4840`).

The next phase is fetching for nodes. Specific nodes on the server are accessed using their Node IDs. These nodes represent various control points for a robot, such as whether the robot can receive data (`CanRecieveData`), the coordinates of a path (`PathCoordinate`), and the indices in an array (`ArrayIndex`), among others. The script also retrieves nodes for the trajectory type (`TrajectoryType`), data readiness (`DataReady`), and the end of the write operation (`EndWrite`). These are either boolean or integer variables. Address of a specific node is shown here:

```
node_can_write = opcua_client.get_node('ns=5;s=MotionDeviceSystem.
ProcessData.R1.Program.Scan_and_Sand.ScanAndSandTemplate.
CanRecieveData')
```

`CanRecieveData` is a variable of the robot, which indicates that the robot is ready to accept individual positions of the path.

When this variable is `True`, the writing process is started. The script enters a loop to send multiple positions to the robot. For each position it waits if the robot is ready for writing, then it is initialized the variable of type `uatype.Pos()`, which is a structure consisting of $\{X, Y, Z, A, B, C, S, T\}$, where X, Y, Z are the coordinates of the end-effector, and A, B, C are the Euler angles corresponding to the *yaw*, *pitch*, *roll* convention. S and T are Status and Turn, which indicate the configuration of the robot. More about them is in the section about the processing paths for the robot controller 4.4. Then the index to which the position is to be written and the type of path, if it is a zig-zag or spiral pattern, is sent. Lastly, a flag is set to indicate that the data are ready. This loop repeats until all positions of the path are written.

Even integer and boolean variables have their OPC UA variant and must be written, for instance, as follows:

```
node_arr_idx.write_value(uatype.DataValue(uatype.
Variant(i, uatype.VariantType.Int32))) // integer i
node_data_ready.write_value(uatype.DataValue(uatype.
Variant(True, uatype.VariantType.Boolean))) // boolean True
```

To end the communication, the `opcua_client.disconnect()` command is only called [27].

The second part involves OPC UA communication between the robot and the Hololens 2 AR headset. The task now is to read the path positions from the robot so that they can be displayed in the headset. After that, a command has to be written to the robot, which pattern the robot has to execute. The AR headset application was developed in Unity, whose native programming language is C#, so the OPC UA library for C# had to be used for communication with the headset. Except for small differences, it was the same as programming communication in Python [28].

After turning on the application in the headset, the connection to the OPC UA server in the robot is immediately established. And then the array of positions is read. The library has not been programmed with a structure for reading the array of `POS` values, therefore only a one-dimensional array is read, where only the coordinates of the path point are stacked. However, this array is read as `string[]`, so it is necessary to parse it into a $2D$ array of type `Vector3[]`, where the elements are single path positions of type float.

After displaying the paths in the headset, the user chooses which type of pattern he wants the robot to perform, which is written as an integer value. The connection to the server is terminated at the moment of shutting down the application in the headset.

3.5 Workspace calibration

The workstation with the robot and the camera must be calibrated in the sense that the transformation matrix between the robot base and the camera coordinate system is known. The desired transformation matrix consists of the rotation matrix **R** and the translation vector **t**. It is necessary that the point cloud, which is obtained from

the camera system, is aligned to the coordinate system of the workstation, which has its origin in the robot base.

The workspace calibration's foundation is computing the best-fit transformation between two sets of points. Given two sets of corresponding points **A** and **B** in 3-dimensional space, where both **A** and **B** are represented as $3\times N$ matrices, the objective is to find a rotation matrix **R** and a translation vector **t** such that [29]:

$$\min_{R,t} \sum_{i=1}^{N} \|B_i - (\mathbf{R} \cdot A_i + \mathbf{t})\|^2. \tag{29}$$

Here, A_i and B_i are the columns representing individual points in matrices **A** and **B**, respectively. The matrix norm is here meant in the sense of the Frobenius norm.

3.5.1 Obtaining the corresponding points

In order to solve problem (29), it is necessary to fill the matrices **A** and **B** with the corresponding points that match each other. This is done by attaching a special calibration artifact (Fig. 3.17) directly to the sanding spindle, which is supplied directly by the manufacturer of the 3D vision system, Keyence. Alternatively, when using a different depth camera, it is possible to use a different calibration artifact, the camera only needs to be able to localize it in space.

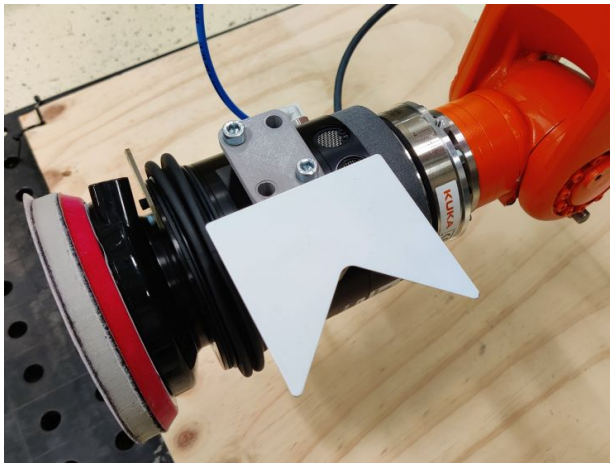

Figure 3.17. Calibration artifact attached to the sanding spindle.

The point that needs to be obtained on the calibration artifact is its center, where the apex of the cutout is. The coordinates of this point in the robot's coordinate system are easy to obtain. A new TCP, which is the center of the artifact, is defined in its controller. Then it is only necessary to read the current position of the TCP from the robot controller. By measuring from the robot, the elements of matrix **B** are obtained, where the individual measured points are arranged in a row.

As already written, the work on this workstation is simplified by the fact that the camera automatically detects the calibration artifact. In Figure 3.18, the interface for the camera can be seen where the calibration takes place. The robot is programmed to traverse a 3x3 matrix of points in three layers, making a total of 27 points. At each point, the position of the calibration artifact is read from both the robot controller and the camera. The calibration in Figure 3.18 contains grid points spaced 10 mm apart in each coordinate [30]. This procedure is used to fill the matrix **A**, where the columns contain the individual positions of the artifact measured by the camera.

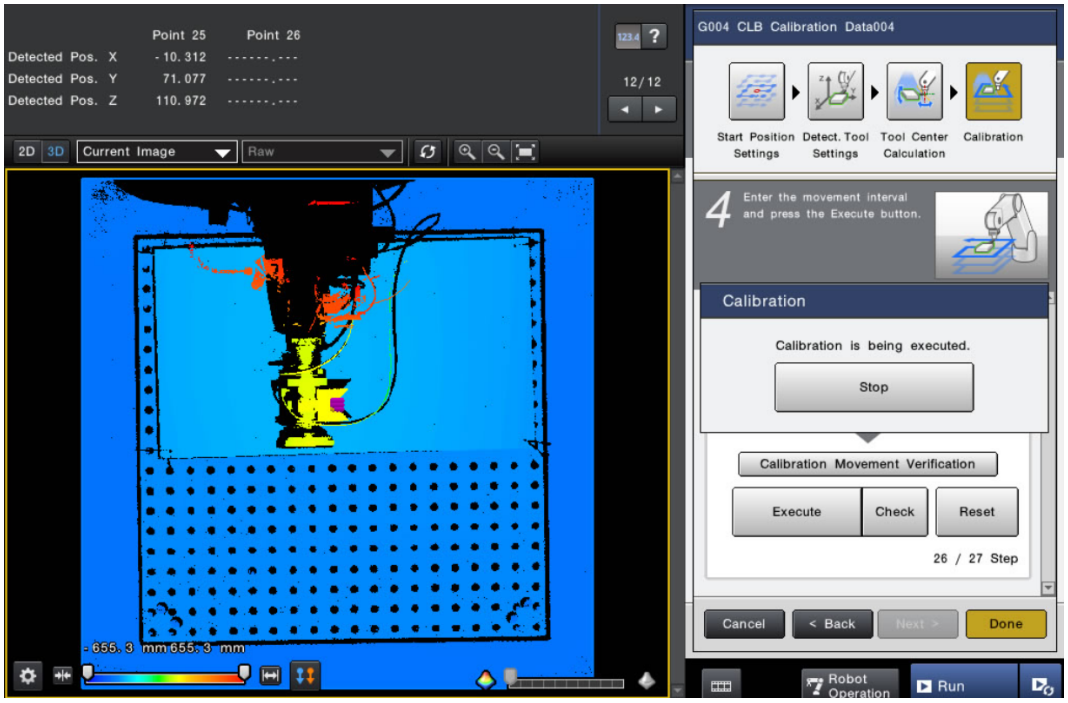

Figure 3.18. Procedure for obtaining a pairs of calibration artifact points.

3.5.2 Rotation matrix and translation extraction

Now, both matrices **A** and **B** are filled with calibration artifact points that belong to each other. First the desired rotation matrix is obtained. It is necessary to shift the points to a common origin first. This is done by calculating the centroids of the two matrices and then subtracting each point in the matrix from the corresponding centroid. The centroid is easily calculated as the mean of the points in each of the coordinates individually:

$$A_C = \frac{1}{N}\left(\sum_{i=1}^{N} x_{A,i},\ \sum_{i=1}^{N} y_{A,i},\ \sum_{i=1}^{N} z_{A,i}\right)^T,\ B_C = \frac{1}{N}\left(\sum_{i=1}^{N} x_{B,i},\ \sum_{i=1}^{N} y_{B,i},\ \sum_{i=1}^{N} z_{B,i}\right)^T. \tag{30}$$

The centroids now need to be subtracted from the individual points in the matrix. This can be done by expanding the centroids into the whole matrix: $\mathbf{A_C} = A_C \cdot (1\ \ 1\ \ \ldots\ \ 1)$, $\mathbf{B_C} = B_C \cdot (1\ \ 1\ \ \ldots\ \ 1)$ and subsequently subtracting it from the desired matrix, resulting in the matrix $\mathbf{A'}$ and $\mathbf{B'}$:

$$\mathbf{A'} = \mathbf{A} - \mathbf{A_C}, \quad \mathbf{B'} = \mathbf{B} - \mathbf{B_C}. \tag{31}$$

This transformed all points of the common coordinate system and now matrix $\mathbf{A'}$ differs from matrix $\mathbf{B'}$ only by some rotation. The problem is reformulated such that it needs to be found the optimal rotation matrix **R**:

$$\min_R \sum_{i=1}^{N} \|B'_i - \mathbf{R} \cdot A'_i\|^2. \tag{32}$$

This problem is already well defined mathematically, it is called the Orthogonal Procrustes problem. It is generally defined as a problem that seeks to find the orthogonal matrix **R** that most closely maps one set of points to another. Mathematically, it can be formulated as:

$$\min_R \|\mathbf{Y} - \mathbf{R} \cdot \mathbf{X}\|, \ \textit{subject to } \mathbf{R}^T\mathbf{R} = \mathbf{I}, \tag{33}$$

where $\mathbf{I}$ is the identity matrix, ensuring that $\mathbf{R}$ is orthogonal [29]. This constraint preserves lengths and angles, making it suitable for tasks where rigid transformations (rotations and translations without scaling or shearing) are required. In the case of searching for a rotation matrix, the matrices $\mathbf{B}'$ and $\mathbf{A}'$ can be substituted for $\mathbf{Y}$ and $\mathbf{X}$. Moreover, the task of minimizing the norm or the second power of this norm has the same solution. The solution of the substitution problem (33) is equivalent to the solution of the problem of finding the nearest orthogonal matrix to a given matrix $\mathbf{H} = \mathbf{A}'^T\mathbf{B}'$, i.e. solving the closest orthogonal approximation problem:

$$\min_R \|\mathbf{H} - \mathbf{R}\|, \ \textit{subject to } \mathbf{R}^T\mathbf{R} = \mathbf{I}. \tag{34}$$

The solution of problem (34) leads to the use of the Singular Value Decomposition (SVD) of the matrix $\mathbf{H}$ [31]. To solve this problem:

1. Compute the cross-correlation matrix $\mathbf{H} = \mathbf{A}'^T\mathbf{B}'$ between the translated (centered) matrices $\mathbf{A}'$ and $\mathbf{B}'$.
2. Decompose $\mathbf{H}$ using SVD, $\mathbf{H} = \mathbf{U}\Sigma\mathbf{V}^T$.
3. Calculate the optimal rotation matrix as $\mathbf{R} = \mathbf{V}\mathbf{U}^T$.
4. Check the determinant of $\mathbf{R}$ to ensure a proper rotation without reflection. If the determinant of $\mathbf{R}$ is -1, it suggests the transformation includes a reflection, which is not desirable. The simplest way to adjust $\mathbf{R}$ is to flip the sign of the last row of the matrix $\mathbf{V}$ ($\mathbf{V}[2,:] = -\mathbf{V}[2,:]$). Then, $\mathbf{R}$ must be recalculated.

Thus, information was obtained about how the shifted points are rotated between each other and even how the original points from matrices $\mathbf{A}$ and $\mathbf{B}$ are rotated between themselves. It now remains to be found the translations $\mathbf{t}$ between these points. For this purpose, the centroids (30) already computed are used, and the translation between them is found. The translation vector $\mathbf{t}$ is computed as follows [29]:

$$\mathbf{t} = B_C - \mathbf{R} \cdot A_C. \tag{35}$$

This procedure yields $\mathbf{R}$ and $\mathbf{t}$, satisfying the original problem (29). With the use of this rotation matrix and translation all points obtained by the camera can be efficiently transferred to the coordinate system of the robot base.

Chapter 4
Implementation & Results

This chapter describes the implementation of the proposed solution developed in previous chapter for the application of Scan & Sand in a specific robotic workplace. The actual workplace where the solution was implemented is located in the Testbed for Industry 4.0 laboratory in the building of the Czech Institute of Informatics, Robotics, and Cybernetics.

Initially, the chapter outlines the setup of the experimental environment, including detailed descriptions of the equipment used. The following is an introduction to the entire automated Scan & Sand pipeline. This is viewed both from the operator's point of view, explaining what has to be done and also describing all the communication and data transfer between the individual components of the workplace.

Following the setup, the process of data acquisition is described. This includes the methodologies for capturing and processing the point cloud data. Then, the modification of the generated paths into a program for the Kuka robot and the transfer of this data to its controller is shown. For verification of the paths and collision detection, the simulation process in RoboDK is also explained in this chapter.

Last but not least, the specific parameters of the `PI` controller for maintaining the constant pressing force during sanding and its results in a real operation are presented. For the validation of the algorithm for the generation of robotic paths, its results are also shown on a different workpiece than the one on which the algorithm was developed. It turned out that working with a rotating sanding spindle caused the most difficulties in real implementation, so it is also discussed how these problems were solved. The idea of determining the dependence of the applied force on the removed material is also examined, which would help to optimize the sanding process. Lastly, the development of an AR headset app is presented, which serves as a user environment and for the visualization of the generated sanding paths.

4.1 Workplace setup

First, it is necessary to describe the devices that were used for the implementation of the solution. In picture 4.1 a real robotic workplace can be seen. The components are the robotic manipulator, to which the force-torque sensor and the sanding spindle are attached. There is also an 3D vision system above the workstation, which captures the sanded object below it. For testing purposes, a wooden board was used as sanded object, which is slightly bent in the middle.

The main component of the workstation is the `KUKA KR 8 R1620 HP`, a high-performance industrial robotic arm notable for its precision, agility, and compact design. The robotic manipulator has a maximum payload capacity of 8 kg and a maximum reach of 1620 mm. Its design emphasizes high-speed operation and repeatability, with an accuracy level that significantly enhances production efficiency and quality. Its kinematics have already been described in section 3.3. The robot is commanded by the `Kuka KR C4` controller and is programmed in KRL (Kuka robot

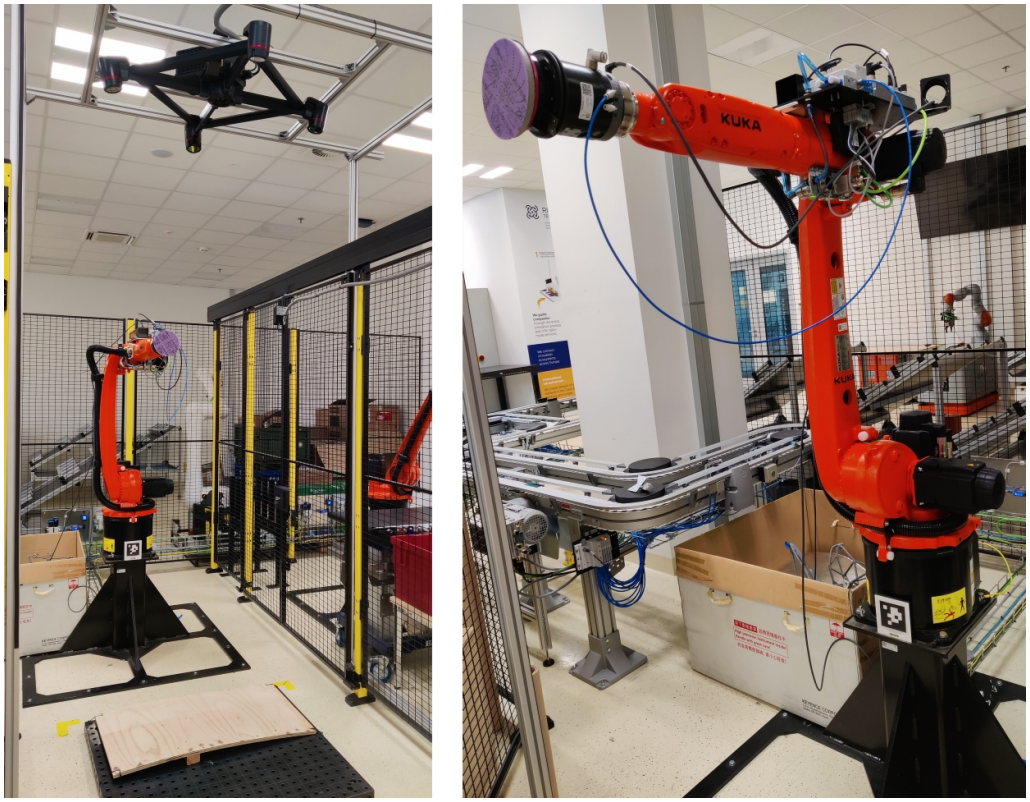

Figure 4.1. Actual workplace for the Scan & Sand application.

language) [32] and RSI (Robot-sensor interface) [21]. The program for the robot is divided into two files: the first is the `src` file, which contain all the instructions, and the second is the `dat` file, which is used to define variables, arrays, or poses.

The robot base also serves as the base of the whole workplace. On the stand in front of the robot there is a marker, which is used to align the scene in the AR headset, which will be further explained in section 4.9.

The force-torque sensor `FT-AXIA` from the manufacturer Schunk is connected between the flange of the robot and the sanding spindle [33]. It is connected via EtherCAT interface directly to the robot controller. Its detailed description has already been included in section 3.2.

Also from the Schunk manufacturer is the sanding spindle of the `AOV` series [34]. It is a pneumatic random orbit sander, so the motor is air driven, and it employs a random orbit sanding action where the sanding disk both rotates and oscillates in a random pattern. The dust that is generated during sanding can be vacuumed out. Polishing wheels can be easily replaced as they are attached via Velcro.

Keyence's high-precision industrial imaging system is used to acquire a point cloud at the workplace. It consists of an `RB-1200` camera system [35] and a `CV-X` controller [30]. This system produces images that combine high-resolution grayscale information with accurate depth data. The depth measurement is facilitated through structured light or time-of-flight (ToF) technology, which projects a pattern onto the object and analyzes how it deforms on surfaces. This method helps in calculating the distance of various points on the object's surface relative to the camera, enabling precise 3D mapping.

The working area of the camera starts 2000 mm below the midpoint of the camera, and its measurement range is 1260 mm x 1260 mm x 1000 mm, where 1000 mm is for

the height (z axis) and the other two parameters for the x and y axes. The resulting point cloud from this camera system has a resolution of 0.616 mm in the x and y axes and a resolution of 0.04 mm in the z axis, i.e. in depth [35].

The CV-X controller processes the data from the RB-1200 with high speed and accuracy. It supports advanced image processing tasks like edge detection, pattern recognition, and automated measurement calculations, essential for precise manufacturing and quality control [30]. However, these possibilities are not used in the workplace. The controller is used only to retrieve raw images from the camera and send them to the application computer and for workspace calibration, which is described in the following section.

4.2 Workstation pipeline

When all the devices, which are in the workplace, have been introduced, it can be shown how they are interconnected and how they interact with each other. Furthermore, this section describes how the whole pipeline of the Scan & Sand solution looks like, where at the beginning, there is a command for the camera to take a picture of the workpiece, and at the end, there is a robot that sands the workpiece. A diagram that shows the interactions between the individual workstation components is shown in Figure 4.2.

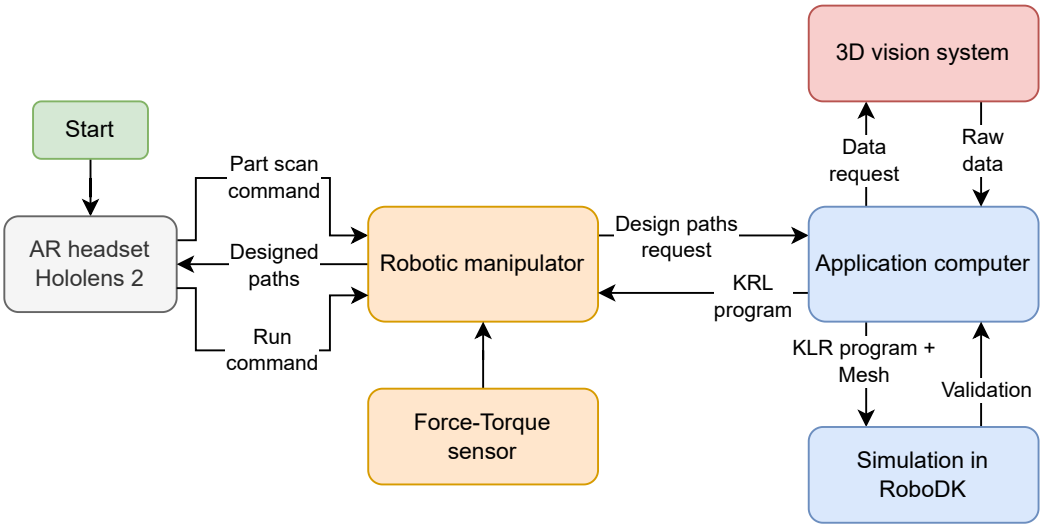

Figure 4.2. Diagram of the interconnection of all devices in the workplace.

It all starts with the operator putting on the AR headset and running the Scan & Sand application. In this application, a simple menu is created to interact with the workplace. The menu appears in the palm of an operator's hand. The palm menu (Fig. 4.16) is displayed in section 4.9, which deals with the usage of AR at the workplace. In that picture, it can be seen four icons:

- Part Scan: This initiates the whole process of designing sanding paths for the robot.
- View Trajectories: Once the paths are loaded in the robot controller, they are also transferred to the AR headset, and this button switches between the visualization of the zig-zag and the spiral pattern.
- Run Trajectory 1: This commands the robot to run the zig-zag pattern.
- Run Trajectory 2: This commands the robot to run the spiral pattern.

The communication between the AR headset and the robot controller is implemented within the OPC UA protocol. The robot waits for one of the three commands: Part Scan, Run Trajectory 1, or Run Trajectory 2. When it receives the Part Scan command, its task is to send this information to the application computer and then prepare to receive the requested poses from the application computer. And when it receives one of the Run Trajectory commands, it simply runs through one of the selected paths.

Now it will be presented how the path generation pipeline works. The Python script `main.py` is executed on the application computer and includes all necessary processes. Other auxiliary functions are programmed in other Python scripts. In `main.py`, the DH notation of a specific robot is hardcoded together with its joint limits. The transformation matrix $\mathbf{T}$, derived in section 3.5, is also defined here. This transforms the points from the robot coordinate system to the robot's base coordinate system.

Next, the `create_pc` function is called, which handles the acquisition of the point cloud from the camera. It first requests the camera to take a picture of the scene and send it to the computer application. And from these images it creates a point cloud of the scene. A detailed description is given in the section 4.3.1.

Since the camera is very precise for this purpose, the resulting point cloud is too dense, so it is then downsampled using `voxel_down_sample(voxel_size=10)` to a computationally acceptable value. The transformation matrix is then applied to all points of the point cloud. Then, unnecessary objects are removed from the scene so that only the sanded workpiece remains in the point cloud. The procedure to create the zig-zag and spiral patterning is then proceeded as described in sections 3.1.2 and 3.1.3. Lastly, a program for the robot controller is made from the designed paths and transmitted to the robot controller. This topic is further explained in section4.4 on the processing paths for the robot controller.

For the purpose of collision detection simulation, a mesh from the point cloud must be generated. RoboDK does not support PLY files, only STL files. For mesh generation, the Open3D library for Python was used, where the method `create_from_point_cloud_ball_pivoting` is directly implemented, which reconstructs the surface by „rolling a ball with a given radius over the point cloud, whenever the ball touches three points a triangle is created [15].“

These are the tasks that are executed by the application computer. It is then convenient to validate the generated paths by sending them to the RoboDK simulation program. There, both the prepared KRL programs and the mesh with the workpiece are sent. The main task of RoboDK is to detect possible collisions with the mesh or the robot itself during robot traversals. If the software simulates the paths successfully, the operator can be sure that no collision will occur in the real workplace and that the code can be executed by the robot. This procedure is discussed further in section 4.5. The transfer of designed paths to the robot controller and the structure of the KRL program itself is described in section 4.4.

Once the paths are ready in the robot controller, the operator can now visualize the two paths in such a way that the paths are displayed on a real workpiece to get the best overview of the sanding process. Once the operator has viewed the two paths, the operator chooses which path to run and then instructs the robot to run that path. This command is again sent via the OPC UA.

After that, there is nothing else to do but to run the robot along the sanding path and thus finish the workpiece. It must not be forgotten that the sanding spindle will

maintain a constant pressure. It receives information about the pressure force from the force-torque sensor and corrects the position of the sanding spindle in real-time. As soon as the action has been performed, the robot is ready for the possible further sanding of the new part.

4.3 Point cloud creation and processing

The 3D vision system, which consists of the CV-X controller and the camera and projector system RB-1200 from the Keyence manufacturer, is used in a real workplace. First, the process of raw data acquisition from the camera system is introduced, followed by editing into the final point cloud.

4.3.1 Obtaining data from the camera

The script initiates communication with the CV-X controller using the TCP/IP protocol. This is facilitated through the Python `socket` module, which allows for the creation and management of network connections. The key parameters for this connection are predefined constants: IP_ADDRESS set to '10.100.0.115', and PORT set to 8500, which are specific to the CV-X device.

The custom made `Send_command(COMM)` function encapsulates the process of sending commands to the controller. It starts by establishing a new socket connection each time a command is sent. This involves setting up a socket with `socket.AF_INET` and `socket.SOCK_STREAM`, which specify the use of IPv4 addresses and a reliable stream-oriented service (TCP), respectively. Once the socket connects to the CV-X controller, it sends the desired command (COMM). The function then listens for a response from the controller using `s.recv(BUFFER_SIZE)`, with the buffer size set to 32 *bytes*, which is sufficient to capture the expected responses from the controller. This response is important as it indicates whether the command was understood and executed by the CV-X. After receiving the response, the socket is immediately closed to free up resources, and the response is returned for further processing [36].

The script contains a dedicated function, `trigger_camera`, designed to ensure the camera is in the correct operational mode and to trigger it for capturing images. The function first checks whether the camera is in setup mode by sending the command `RM\r` (Read Mode). The expected response, `b'RM,0\r'` , would confirm the camera is in setup mode, prompting a command `R0\r` to switch the camera to run mode. The camera actually has two modes: a run mode and a setup mode. In the setup mode, it is possible to set the possible image processing, which is programmed directly by the manufacturer, but none are used for this application. In the run mode, the camera is ready to capture images.

Subsequently, the script verifies the active program on the CV-X by sending a `PR\r` command. If the response indicates a different program is active (anything other than `b'PR,1,500\r'`), the script commands the camera to switch to the desired program `PW,1,500\r`, where 1,500 is a specific program identifier for the Scan & Sand application. The camera contains many programs, and it is possible to switch between them quickly. This allows the camera to be deployed for multiple image processing tasks on a single workstation.

Once the correct mode and program are set, the script sends a `T1\r` command to physically trigger the camera. This is the final step in the command sequence, aimed at initiating the actual image capture process [30].

The camera has now taken a picture of the scene and it can be shown how to transfer the captured data to the application computer. Between sending a command to trigger the camera, it is necessary to wait some time before the camera takes a picture of the scene and stores it in its memory. Sleep for three seconds should be sufficient. The script uses the `FTP` protocol (File Transfer Protocol) to retrieve the captured images. `The download_pointcloud` function handles this by establishing an FTP connection to the same IP address, though on the default port 21, typical for FTP operations.

Once connected, the script logs in as „anonymous“, a common practice for devices allowing non-secure file access. It navigates to the specific directory on the device where the images are stored. Here, it retrieves two specific files, `HEIGHT.png` and `GRAYSCALE.png`, representing different types of data captures (heightmap and grayscale images). Each file is downloaded in binary mode (`retrbinary`) and directly written to local files of the same names. This ensures that the captured data is transferred efficiently and without corruption [37].

4.3.2 Creation of point cloud

The camera data processing function starts by loading two images: a grayscale image (GRAYSCALE.png) and a depth image (HEIGHT.png). The HEIGHT.png image is also in grayscale, which encodes the depth information. The darker the shade, the farther the point is from the camera system. Both have a resolution of 2048x2048 pixels. These images are loaded using the Python Imaging Library (PIL), specifically the `Image.open()`.

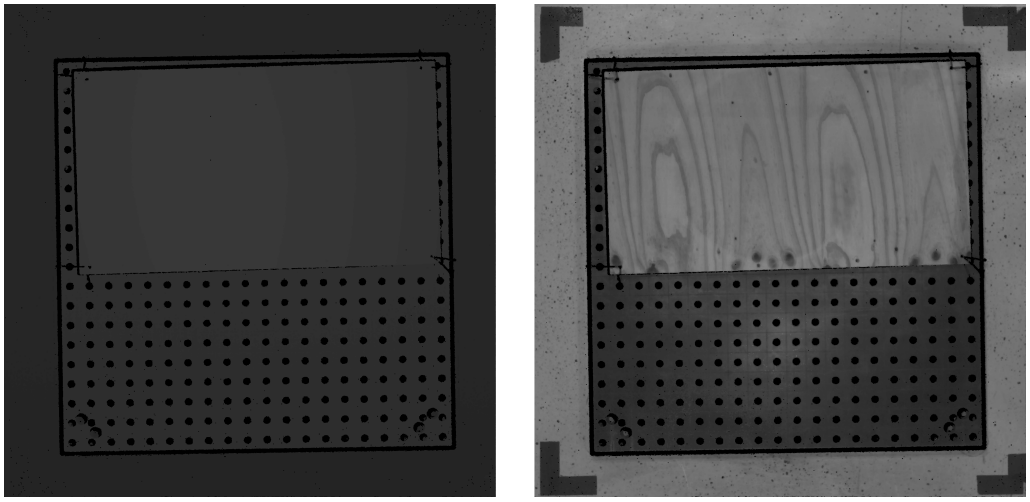

Figure 4.3. The image on the left is the HEIGHT.png and the image on the right is the GRAYSCALE.png.

Once loaded, these images are converted into numpy arrays, which allows for efficient mathematical manipulations. The grayscale image, which initially spans a 0-255 range (8 *bit* values), is normalized to a 0-1 range by dividing by 255. This normalization is necessary for the Open3D library, which is used to represent point clouds and accepts their color in the 0-1 range. The depth image data, stored in a 16 *bit* format, is converted into real-world measurements (millimeters in this case) by multiplying each pixel value by 0.04, reflecting the physical depth measurement per pixel increment.

A new numpy array `points_xyz` is created, an array of 3D points that will serve as a template for the resulting point cloud. The x and y coordinates of the point cloud are scaled by a factor of 0.616 $mm/pixel$, adjusting the spacing between the points to match their actual physical spacing as captured by the camera. This scaling converts the point coordinates from pixel space to real-world metric space. Each x and y coordinate is assigned a value from the heightmap.

Next, $(\,630 \quad 630 \quad 500\,)^T$ is subtracted from all points to ensure that all points of the point cloud are moved to the center of the camera workspace. The dimensions of the workspace are, in fact, 1260 mm x 1260 mm x 1000 mm [35, 30].

The script proceeds to create the base structure for a point cloud using `open3d.geometry.PointCloud()`. This structure will be populated with the 3D coordinates and color information extracted from the depth and grayscale images, respectively. Values from the `points_xyz` array are assigned to this point cloud, and the color information for the point cloud is from the grayscale image. The color for each point is assigned as an RGB (Red, Green, Blue) triplet, where each component is equal to the grayscale value. Lastly, all points with a zero z coordinate are discarded from the point cloud (this indicates a zero or invalid height).

4.4 Path processing and uploading to the robot controller

This chapter summarizes further processing of the designed paths so that they can be executed by a specific Kuka robot. The input to this process is two arrays with transformation matrices representing the paths of the two patterns. This was described in the section 3.1.

4.4.1 Programs for the robot

The `SaS_Control.src` program must be running in the robot controller, which waits in a loop for commands from the AR headset. The pseudo-code that describes the structure of the program is presented here:

```
Def SaS_Control:
    Set base and tool
    Set ADVANCED = 5
    Set APO.CVEL = 95
    Set acceleration and velocity parameters
    Move to HOME position

    Initialize variables:
        Command flags and counters
        Initialize two coordinate arrays for 250 points

    Main Loop:
        Wait for command request

        Switch (Command_Number):
            Case 1: "Data Preprocessing"
                Request Design paths
                Call PreprocessData function
```

```
                Update command completion and readiness flags

            Case 2: "ZIG ZAG Pattern"
                Call ScanandSandTemplate with parameter 1
                Update command completion and readiness flags

            Case 3: "SPIRAL Pattern"
                Call ScanandSandTemplate with parameter 2
                Update command completion and readiness flags

            Default:
                Set error state

        If no command is requested:
            Reset command complete and error flags

        Reset command ready flag

End
```

A `dat` file must be prepared for this `src` program, in which all synchronization flags, `Command_Number`, path arrays and `HOME` position are defined. After starting the `SaS_Control.src` program, the necessary Tool and Base are selected first. Next, the internal variable `$ADVANCED` is set, which tells the robot controller how many movements it should be loaded in advance so that these movements are made without stopping and starting at individual points on the path, and the movements are performed continuously. `$APO.CVEL` sets when the movement to the next point is to be started depending on the change of speed when approaching the target point. Specifically, when the speed drops to 95%, a new movement is started. Then it also sets limits on speed, acceleration, and jerk on individual axes so that the robot does not make too abrupt movements during the sanding process. The robot is then sent to the `HOME` position using point-to-point (PTP) movement [32]. When it reaches the position, the driving flags are initialized. These are used to synchronize communication between the AR headset and the robot, as well as the robot and the application computer. These are boolean variables. Next, two arrays (`PathCoordinates1[250]`, `PathCoordinates1[250]`) of Pos variables are initialized to zero values, which contain the individual path positions.

The main part of the program operates within an infinite loop, continuously monitoring for command inputs. This loop reacts to different command numbers that trigger specific operations such as a data preprocessing and executing zig-zag and spiral patterns. These operations are managed through a switch-case structure, which processes tasks based on the command number received. Each task within the switch-case structure handles the robot's busy state, waits for the robot to complete its current action (monitored through asynchronous state checks), and updates various flags to indicate the readiness and completion of tasks. Error handling is also integrated, and error flags and command results are set as appropriate to ensure robust operation management.

In the case event for preprocess data, the flag for requesting the path design is first set on for the application computer. Next, the `PreprocessData` function is switched to, which handles the receiving of positions from the application computer.

It first waits until the `Data_ready` flag is set on, which indicates that the application computer has already generated the paths and they are ready to be sent to the robot controller. There, gradually, all positions in the two arrays that contain both generated paths are filled. It is important to note that a single pattern does not have to have exactly 250 positions but can have less. This means that the last cells of arrays are copied with the last positions of the path. The robot controller can determine by itself that it skips positions that are the same [32].

Case events `ZIG ZAG Pattern` and `SPIRAL Pattern` call the auxiliary function `ScanandSandTemplate` with either parameter 1 or 2. This function performs the execution of the path itself, which is stored in the PathCoordinates1 or PathCoordinates2 array. Its pseudo-code is here:

```
Function ScanAndSandTemplate(TrajectoryType: Integer):

    Switch (TrajectoryType):
        Case 1:
            Move PTP to the first coordinate in PathCoordinates1

            Create and turn on the RSI Context for force control
            For each coordinate from 1 to 250 in PathCoordinates1:
                Move LIN the coordinate at constant velocity
            Turn off and delete the RSI Context
            Move LIN relative in Z by +100 mm

        Case 2:
            Move PTP to the first coordinate in PathCoordinates2

            Create and turn on the RSI Context for force control
            For each coordinate from 1 to 250 in PathCoordinates2:
                Move LIN the coordinate at constant velocity
            Turn off and delete the RSI Context
            Move LIN relative in Z by +100 mm

    End Function
```

When one or the other path is to be traversed, the first point of the path is first reached by PTP movement. Then the RSI Context is created and activated, in which the force controller from section 3.2.2 is implemented. Then all points in the selected path are traversed. The points are traversed in linear motions (LIN), which differs from PTP motion in that the path between the points must be strictly linear. The points are traversed at the selected constant velocity, which is given in mm/s [32]. Immediately after reaching the last point of the path, the RSI Context is deactivated and deleted. Then, the robot moves away from the ground object by $+100$ mm in the z coordinate.

4.4.2 Programs in the application computer

This section describes how in the application computer the array of transformation matrices must be modified into an array of Pos structures and the protocol how to transfer these arrays to the robot controller.

The Pos structure for Kuka robots is defined as follows: {X, Y, Z, A, B, C, S, T}, where X, Y, Z are the coordinates of the TCP, A, B, C gives information about the

rotation, where A is the rotation around the z axis, B is the rotation around the y axis, and C is the rotation around the x axis. S and T are Status and Turn, which carries information about the specific configuration of the robot.

Position and orientation coordinates are 32 *bit* floats. And both Status and Turn are integers that are composed of bit values arranged one after the other and converted to integers. The Status of a robot is calculated based on the positions of various axes in relation to their coordinate systems or relative positions. This is how the individual bits are determined:

- Bit 0 - Wrist Intersection Position (A4, A5, A6): If the x value of the wrist intersection, relative to the base coordinate system, is negative, it indicates the robot is in the `overhead area.` The Bit 0 is set to 1. If the x value is positive, the robot is in the `basic area,` and the Bit 0 must be set to 0.
- Bit 1 - Axis A3 Position: Bit 3 defines the position of the elbow. If A3 is greater than or equal to 1.74°, Bit 1 is set to 1. Otherwise, it is set to 0.
- Bit 2 - Axis A5 Position: Axis A4 serves as the reference for A5's neutral position. Assume A4 at 0° represents a horizontal line. If A5 is tilted upward relative to this neutral line, Bit 2 is set to 1. In the opposite case, if A5 is either tilted downward or at the neutral position (0° relative to A4), Bit 2 is set to 0.
- Bit 3: This bit is unused and should always be set to 0.
- Bit 4 - Accuracy of Robot: This depends on whether a so-called *absolutely accurate robot* is used, whose accuracy must be certified directly by the manufacturer. In this case, Bit 4 is set to 1. There is no absolutely accurate robot on the workstation in Testbed, so Bit 4 must be set to 0.

These bits are assembled into a binary number in the sense of the bitwise `OR` operator. In its position, it will be either 1 or 0. This number is then represented in decadic form.

The Turn helps specify the rotational direction of the axes and supports motions beyond the typical ±180° constraints. As a general rule, it applies that if the axis angle is positive or 0°, the corresponding bit is set to 0. Opposing, if the axis angle is negative, the corresponding bit is set to 1. For bit allocations, the following applies: Bit 0 for A1, Bit 1 for A2, Bit 2 for A3, Bit 3 for A4, Bit 4 for A5, and Bit 5 for A6. Again, a single binary number is assembled using bitwise `OR`, which is then represented decadically.

When it is known how to determine all the components of the Pos structure, it is possible to proceed to its conversion from the transformation matrix. The process of converting an array of transformation matrices begins by extracting the position and rotation of the individual positions in the path. The position (X, Y, Z) is simply taken from the translation vector. The information about the rotation is obtained by converting the rotation matrix to Euler angles, in the convention A = yaw, B = pitch, and C = roll [32].

When the positions and rotations are known, the whole path is converted into joint coordinates. For each pose in the array, the corresponding joint configuration **q** is computed using an inverse kinematics function. If the inverse kinematic task has no solutions, it indicates that the robot cannot reach the pose. In this case, the conversion of coordinates is terminated, and the previous position is taken as the last one that can be reached. This case can occur quite easily when the algorithm for path design uses positions that the robot cannot reach, for example, because they are outside its working envelope.

Typically, however, a situation arises where multiple valid configurations $\mathbf{q}_i$ are found, then it is iterated through them to select the best one. The best configuration is determined by comparing each configuration to the previously selected one. For the very first position, the HOME = {0°, -90°, 90°, 0°, 90°, 0 } position is the reference. The criteria used is the sum of squared differences between the previous configuration and each configuration $\mathbf{q}_i$, effectively selecting the configuration closest to the previous one in the joint space. The aim is to prevent the robot from reconfiguring itself during the sanding process. Special consideration is given to the fifth joint; if its value is close to zero, the configuration is skipped as it implies a singular configuration, which can lead to control issues. This means that A4 and A6 are in cover and the robot controller cannot handle such a situation [32].

Then, it is only necessary to take this array of joint coordinates and convert them into an array of Pos structures. The TCP position and orientation are easily computed using the forward kinematic task, and the Status and Turn are determined according to the procedure shown in the paragraphs above.

Now, it is left to introduce how the two paths in the two arrays are transferred to the robot's controller. Every single position from each of the arrays is sent individually. It is sent to the OPC UA server in the robot controller, where the arrays must already be initialized. Within the loop over positions, there is a busy waiting that continuously checks if data can be written to a node (`node_can_write` flag). Once the node is ready for writing, a new Pos object of a custom type `uatype.Pos` is created and populated with the corresponding coordinates {X, Y, Z, A, B, C, S, T} from the path list. This Pos object is written to the server. Together with it, the number to which field the Pos must be written (1 for zig-zag, 2 for spiral) and to which index in this field is to be written. When the values are ready on the server, the `data_ready` flag is sent to signal the robot that the new coordinates are ready to be written. This process is repeated until all positions from both paths are written to the robot controller.

4.5 Simulation of the path in RoboDK

RoboDK is a powerful simulation and offline programming software designed for industrial robots. RoboDK boasts a comprehensive library that includes over 500 robots from more than 50 manufacturers, along with various tools and robot peripherals. Its main advantage lies exactly in offline programming. For a user who has to develop a whole robotic cell, it is easier to create a digital version of it first. There, he or she can add all the components of the workplace, program the robot paths, detect collisions and optimize the overall production process. More robots can be added to the workplace in RoboDK. This allows easy and intuitive synchronization of the work cycle of individual robots. After preparing the robot workplace offline, deploying previously developed robotic programs on real robots is easy [38].

Only collision detection and path validation are used in this work. It is convenient to check if the paths generated by the presented algorithms are in a valid condition and if the robot can execute them without collisions with the environment, the workpiece, or itself. It can also reveal singularities in the generated paths. The real robots could not pass through them and would stop in them.

The RoboDK environment is shown in figures 4.4 and 4.5. After starting RoboDK, the robot and sanding spindle models need to be imported into the environment. The spindle must then be selected as the robot's tool. The generated mesh of the

sanding workpiece is also imported. This must be selected as a child of the coordinate system of the robot base. Since RoboDK allows API calls in Python, it is possible to combine these initial imports into a single file, `Prog1.py`. This will perform the necessary imports and alignment to the correct coordinate systems automatically after RoboDK is started.

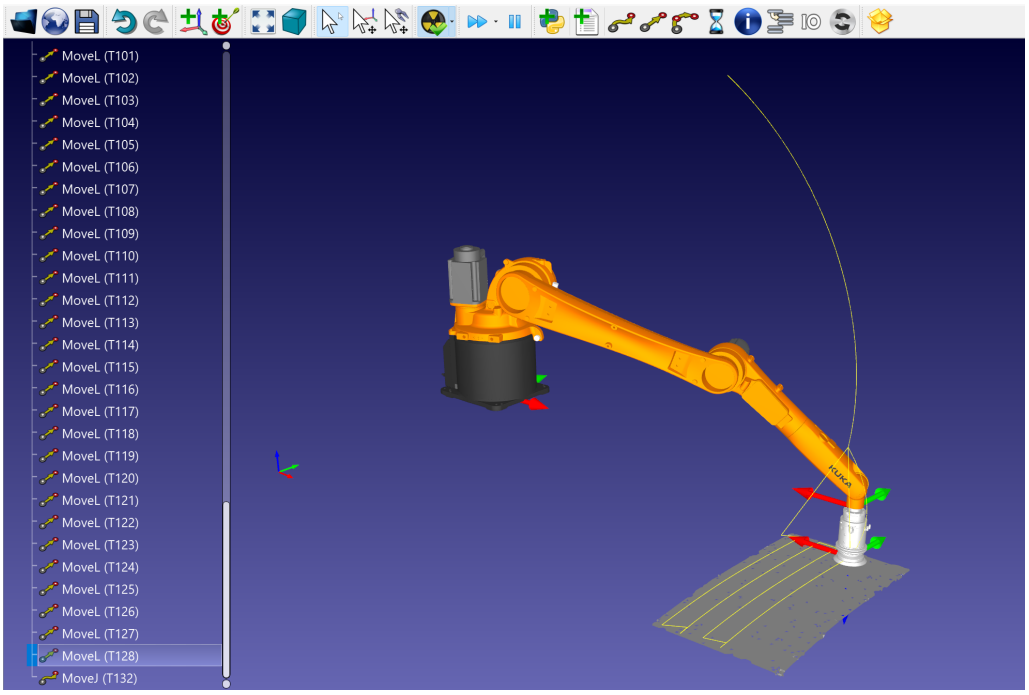

Figure 4.4. A simulation in RoboDK, where the robot traversed a designed path without collision.

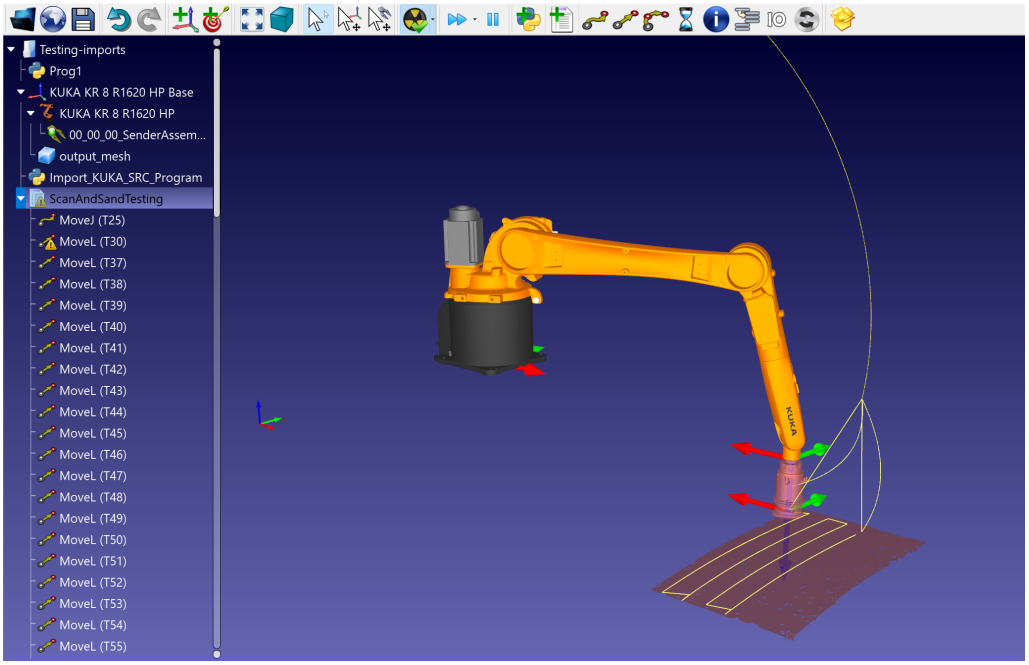

Figure 4.5. Simulation in RoboDK, where the robot is in collision with a workpiece.

In the RoboDK library, there is available another pre-prepared Python script, `Import_KUKA_SRC_Program.py`, which allows to import any `src` file. However, it has limited possibilities, for example, it cannot handle the Status and Turn parameters [39]. For this case, a simple program `ScanAndSandTesting.src` is created when generating the robot paths, which stores only the individual points of the path, simply without the S and T parameters. In the tree structure on the left in RoboDK (Figure 4.5) the individual models and the imported program from the robot are visible. And the imported path is indicated there by a thin yellow line.

After importing the robot program and the part's mesh into RoboDK, it is possible to run a simulation of the robot, where the virtual robot runs through all the points in the program. The collision detection is activated by clicking on the Radiation icon at the top bar of RoboDK. It is natural that the paths were generated in such a way that the sanding spindle touches the part. Therefore, the points in `ScanAndSandTesting.src` are moved 10 mm upper in the z axis. The program simulated in this way can be seen in Figure 4.4, where the robot reached the last point without any issues. And it means that the generated path is valid. On the contrary, if the necessary offset in the z axis were not made, the robot would be in collision at the very first the first point, as can be seen in Figure 4.5.

4.6 Testing the force feedback controller

The controller of the pressing force applied to the surface of the workpiece, which was introduced in section 3.2, is in an actual workplace programmed in the Robot Sensor Interface (RSI) framework. RSI is a real-time communication interface for KUKA robots. RSI is designed to allow external devices such as sensors, computers, or controllers to communicate with the KUKA robot controller at high speeds. This is critical for applications requiring immediate response based on sensor feedback, such as maintaining constant pressing force during sanding. RSI operates with a cycle time of 4 ms during the sanding application. RSI uses an XML format for data exchange between the robot controller and external devices. The interface processes XML data packets that contain both commands to the robot and feedback from the robot. RSI is programmed in a graphical development environment within WorkVisual and is then compiled into an XML file.

The RSI program, called the `RSI context`, is then embedded in the `src` file of the robot program and is triggered each time when the robot reaches the first point of the path on the workpiece and is turned off when the robot traverses the entire path. The whole time it runs, it adjusts the path in real-time. This is achieved by the `PosCorr` function block in the RSI context. The path increment that the robot is to move enters into it. In this case, the z coordinate is being adjusted, and it is relative to the tool coordinate system, i.e., the sanding spindle. The input to the `PosCorr` block is the output of the PI controller. It is the input to the position control loop, which is solved within the internal Kuka system, that solves the interpolation in the Cartesian coordinate system. This adds additional position control requirements in parallel to the processed KRL code.

In a real application, the PI controller function block has the following response [21]:

$$y(k) = y(k-1) + KR \cdot x(k) - KR \cdot \left(1 - \frac{sensor\ cycle}{TN}\right) \cdot x(k-1), \qquad (1)$$

where $y(k)$ is the output of the controller, $y(k-1)$ is the output of the previous cycle and $x(k)$ is the input to the controller, $x(k-1)$ is the input of the previous cycle. The sensor cycle, in this case, is 4 ms. And the controller constants were set to $KR = 5.5 \cdot 10^{-4}$ for the proportional component and $TN = 8$ s for the integration component. Both constants were achieved empirically. The constant of the proportional component is set so that the robot does not react too aggressively and does not overshoot the desired force value. On the other hand, it is set so that the robot quickly returns to the desired force value when it is deviated. The integration constant was derived to increase the control action when the controller still did not reach the required force, but on the other hand, not to overstretch the proportional component.

It is important to mention that in the actual implementation, the output from the `PI` controller is not enabled for `PosCorr` until after 0.5 s. This is because immediately after the startup of the RSI context, the output of the low pass filter is saturated at around 35 N, which is an erroneous value. And it reaches the correct value after a few cycles of the RSI contexts.

First the sanding program was run without controlling the applied force. The measured force in this case is shown in Figure 4.6. There, the initial saturation can be seen at 35 N; then, it can be seen that most of the time, the sanding spindle barely touches the surface of the part. However, there are also visible spikes in the measured force, indicating that the sanding spindle has passed some surface inaccuracy.

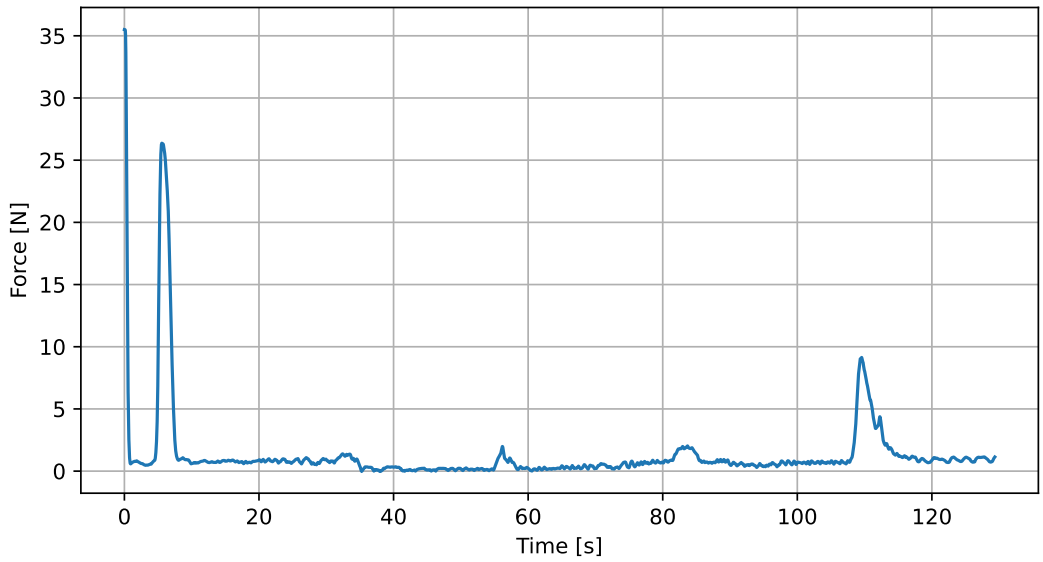

Figure 4.6. Measured force during sanding without the regulator on.

The reference used for testing was 10 N. The robot had to maintain this value of the pressing force throughout the entire run of the RSI context. This value is not enough to talk about sanding but rather about polishing. However, it would have been better to start with a smaller force first. This controller has been tested on a zig-zag pattern path, as seen, for example, in picture 3.4. In Figure 4.7, the force measured by the force-torque sensor can be seen. Furthermore, Figure 4.8 shows the output of the `PI` controller, i.e., its control action.

As can be seen in Figure 4.7, the control of the pressing force is rather fluctuating. On the other hand, it is worth mentioning that the input data is very noisy, this is mainly due to the rotational motion of the spindle, which makes a random orbit

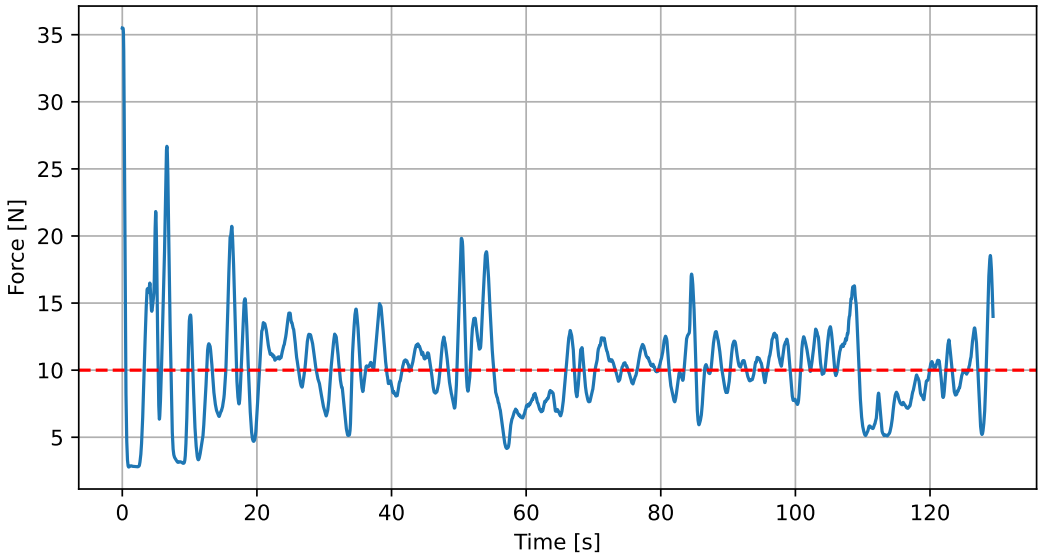

Figure 4.7. Measured force on the force-torque sensor with the RSI context running.

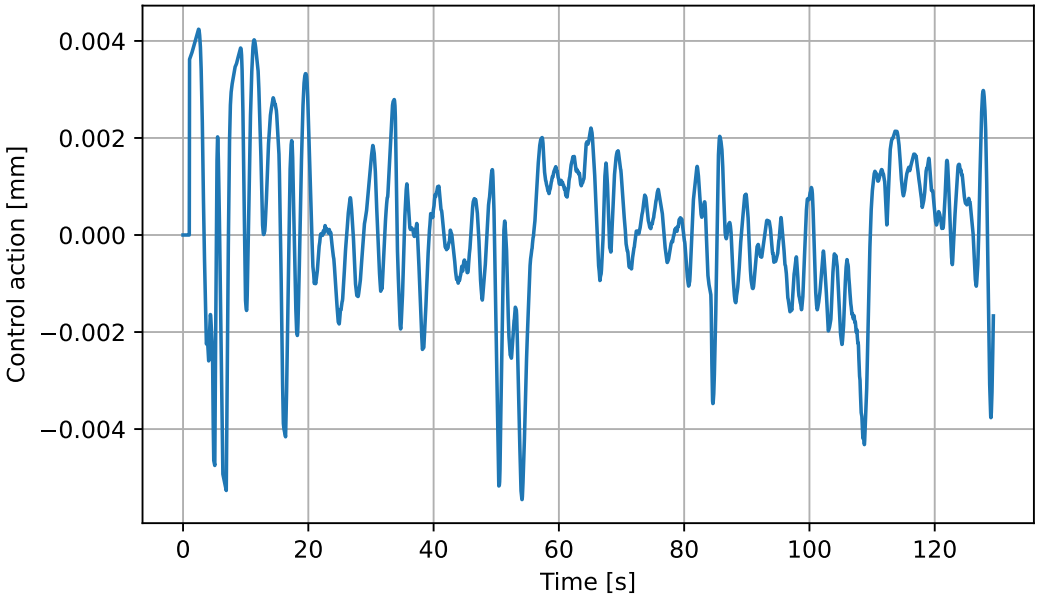

Figure 4.8. Output from the `PI` controller.

movement. This could be considered a great result, as it has proven its concept. In addition, most of the time, the force was maintained in the range of ± 5 N, which is a solid result. Furthermore, the controller compensates well for the inaccuracies of the point cloud. When an increased force appears there, the regulator tends to intervene in the opposite direction. And in spots where the force was originally almost zero, the controller tries to maintain the 10 N of applied force.

The second graph (Fig. 4.8) shows the output of the controller. These are the values by which the z axis of the tool must be moved in one cycle of the RSI context. These are relatively small increments; if they were a bit larger, there would be a risk that the motion would be too dynamic and the robot would break because of the speed overload on the axes. Another good feature is that the size of the control action is immediately reflected in the size of the pressing force.

4.7 Verification of the path generation algorithm

It is convenient to verify the algorithm that generates the sanding paths on different parts than the ones on which it was developed and tested. Therefore, another work table was brought to the workplace for this purpose, on which a new part was placed. The part used to verify the algorithm was a plastic hemisphere. A picture of this configuration can be seen in Figure 4.9. This part was chosen to be more curved than the former pilot part and preferably to be curved in more directions, which this hemisphere fulfills.

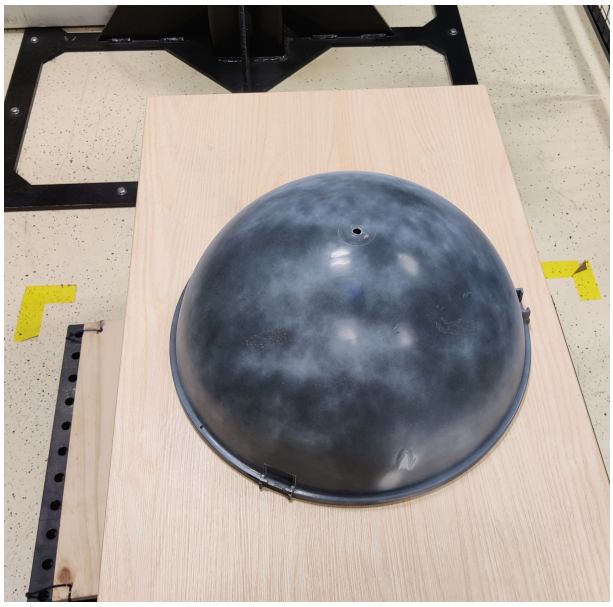

Figure 4.9. Spherical workpiece used for verification of the algorithm for the generation of grinding paths.

However, during the initial scanning of the object, it was revealed that the hemisphere was too shiny. This complicates the scanning process, as such a shiny object scatters and reflects in uncontrollable directions the rays of the swept patterning of the 3D vision system. This problem has been largely eliminated by applying a chalk spray on the surface of the hemisphere, as it makes the surface matte. Minor problems remained only around the rim of the hemisphere, which are almost vertical surfaces oriented towards the camera and thus remain partially occluded by the camera system.

Since the spherical object is slightly smaller than the former object, the internal radius parameters in the algorithm have been changed for better visibility of the paths. Specifically, for the zig-zag pattern, `voxel_radius = radius * 0.45` and `tolerance_x = radius * 0.25`. For the spiral pattern, the adjustments are `voxel_radius = radius * 0.5` and `tolerance = radius * 0.1`. It is also important to mention that the threshold parameter which removes the work table from the point cloud has been modified. This parameter had to be moved up towards the camera system. Then, the resulting generated paths are for the zig-zag pattern in Figure 4.10 and for the spiral pattern in Figure 4.11.

As can be seen on both paths, the necessary condition that the individual positions for the robot's end-effector are always perpendicular to the surface of the part has been fulfilled. In the zig-zag pattern image, the single layers of the path going

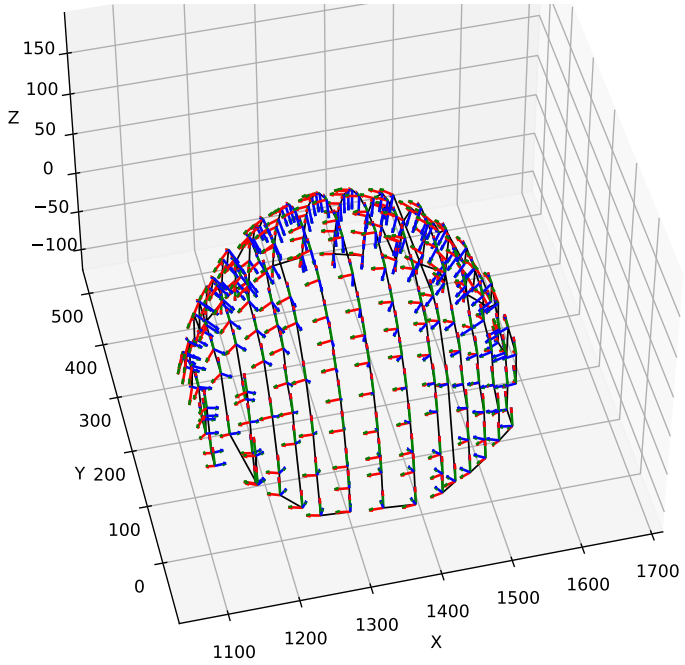

Figure 4.10. Generated zig-zag pattern on the sphere.

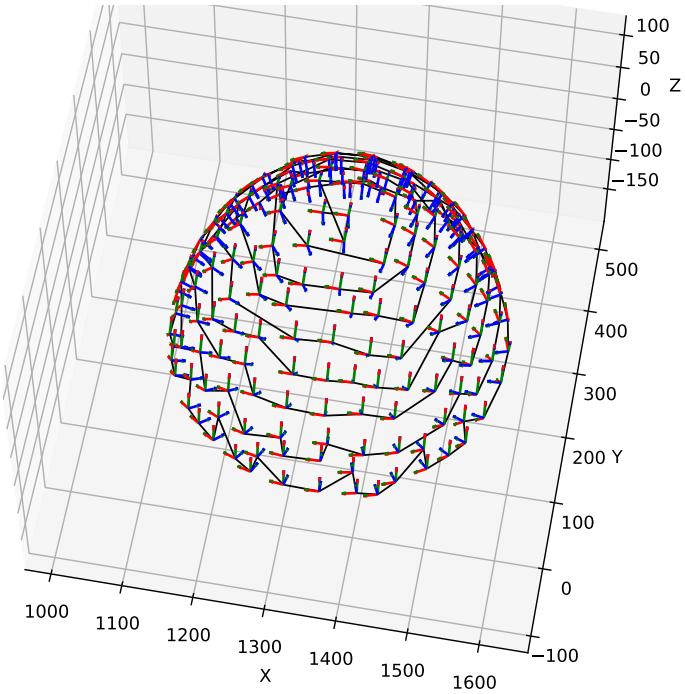

Figure 4.11. Generated spiral pattern on the sphere.

from left to right can clearly be seen traversing the entire hemisphere. The spiral pattern had some minor issues, such as some points around the rim of the hemisphere missing, which disturbed the bottom contour of the spiral pattern, and this distortion propagated to the inner layers. However, it is not a significant inaccuracy; it only makes the spiral pattern slightly distorted, and the robot would have run the sanding paths in order.

4.8 Challenges encountered in the real workplace

When the proposed methods were transferred to an actual workplace, certain difficulties occurred which could not be anticipated. It was, therefore, necessary to refine the developed methods further and thus optimize the sanding process for the real world.

4.8.1 Issues with the rotating spindle

The most significant problems arose from the rotation of the grinding spindle. It turned out that the random orbit motion generates high-frequency noise. Both the gravity compensation process and the force feedback controller did not take it into account and produced poor results. Both programs in RSI had to be modified to take this into account.

First, additional low-pass filters were added to the RSI context, which logs the F/T sensor measurements on the individual axes. Low pass filters were added right to each of the three sensor outputs. Again, they were employed with the Bessel function, this time of the tenth order. This helped to get rid of the high-frequency noise, but the outputs were still fluctuating a lot, even when the sensor was at rest and the robot was not moving. So, the outputs were averaged in this window. When the robot had settled down, instead of taking one reading, ten readings were taken and then averaged. These adjusted values then entered the formulas (7), (8) for determining the offsets $offset_x$, $offset_y$, $offset_z$, $offset_{norm}$.

In such a way, the compensation of offsets that enter the regulator of the applied force in the sanding process has been improved. Nevertheless, in the second RSI context, which implements this controller from Figure 3.9, the same three low-pass filters were added right at the outputs of the F/T sensor. The low pass filter of the second-order from the diagram has been preserved; it still suppresses larger fluctuations of the signal that enters the PI controller. Next, in the actual RSI context, a protective function block was also added in comparison to the diagram in Figure 3.9, which immediately stops the robot when the measured force on the F/T sensor exceeds 30 N. Since the second-order low pass is saturated at the very beginning of the RSI context, this function block is again active only after 0.5 s after the RSI context had been started.

4.8.2 Correlation between applied force and removed material

In order to optimize the force that needs to be applied to the sanded part, it would be helpful to find an exact relation between the force applied and the removed material. It has been thought that a 3D vision system with a z axis resolution of 0.04 mm should produce point clouds on which the removed layer can be seen. However, it turned out that this accuracy was not sufficient. In Figure 4.12, it is possible to see the measured difference between the two point clouds before and after sanding. The force applied to the part was again 10 N.

In reality, one could see that the part was sanded, but in the point cloud differences, only areas of the part at the beginning of the sanding path were visible. There, the force displacement was larger than required, and in addition, the regulator was fluctuating. More worrying, however, are the points with x coordinates of 1350 mm and more, which the robot could never run into because it is beyond its reach, and there should have been a difference in sanding of exactly 0 mm. The 3D vision system has difficulty with such small resolutions and produces noisy outputs. For a

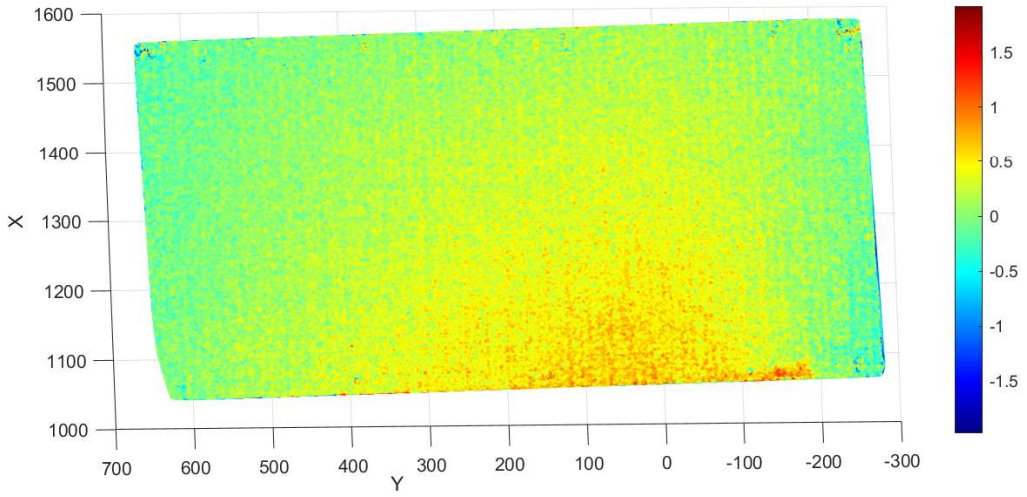

Figure 4.12. The difference between point clouds before and after sanding.

better study of the material removed, it is therefore necessary to use a more accurate scanning device.

4.9 Hololens 2 app for path visualisation

This section provides an introduction to Augmented reality (AR) for industrial applications. Specifically for visualizing sanding or polishing patterns on a real workpiece and as a user interface for controlling the workstation. By leveraging the headset's capabilities, users can view, adjust, and refine the robotic paths in a three-dimensional space, directly overlaying the virtual paths onto the actual target surfaces. This method of visualization facilitates a more intuitive understanding of complex mechanical processes [2].

4.9.1 Introduction to Augmented Reality and Unity

Augmented reality is a technology that overlays digital content onto the real world, enhancing one's perception and interaction with their environment. Unlike virtual reality, which replaces the real world with a simulated one, AR integrates digital components into the user's view of their surroundings.

The Microsoft HoloLens 2 (Fig. 4.13) is an AR headset that embodies the practical application of AR technologies. The HoloLens 2 offers a combination of high-definition visuals, spatial mapping, and real-time interaction, making it an ideal tool for projecting programmed trajectories onto a physical workpiece. It features a set of see-through holographic lenses (waveguides) that utilize a projection system to render digital holograms onto the user's field of view. This headset is equipped with multiple sensors, including depth cameras and an inertial measurement unit (IMU), which facilitate spatial understanding and allow the device to accurately map and interact with the physical environment. The HoloLens 2 operates on Microsoft's custom-built Holographic Processing Unit (HPU) and an ARM processor, which handles the extensive data processing required for real-time AR [40]. Its interaction model is based on gaze, gesture, and voice inputs, making it exceptionally intuitive.

Unity is a versatile and widely used game engine and development platform that supports the creation of both 2D and 3D content. Its set of features allows developers to create detailed environments, scripting in C#, and integrate various media

Figure 4.13. Augmented reality headset Hololens 2 from Microsoft.

assets. Unity's compatibility with multiple platforms and its comprehensive toolkit for rendering, physics, and networking make it an excellent choice for developing AR applications. Its integration with the HoloLens 2 is facilitated through a range of plugins and SDKs (software development kit) provided by both Microsoft and third-party developers, allowing for advanced features such as spatial mapping, gesture recognition, and real-time interactions to be implemented effectively.

4.9.2 Hololens 2 app development in Unity

The process begins with setting up the Unity environment to support HoloLens development. Unity provides a comprehensive platform for creating interactive 3D applications, and with the addition of the Mixed Reality Toolkit (MRTK), developers can use a suite of tools specifically designed to facilitate AR development for Hololens 2. Within Unity, the Universal Windows Platform (UWP) development build support is essential, as it allows the application to run on Windows 10 and Windows 11 devices, including Hololens 2.

With Unity and MRTK set up, developers can start constructing the application. This involves creating scenes, which are containers for all objects, lights, cameras, and scripts in a Unity project. For Hololens 2 applications, the camera is configured to follow the user's head movements, providing a first-person view of the AR world. Necessary objects can be added to the scene. These are the robot model with a stand, the model of the sanding spindle, the palm menu, and the Vuforia marker.

The models are in OBJ format. The Palm menu is already pre-prepared by the MRTK toolbox, and the icons and button labels can be modified. On pressing the button events, functions are mapped from the script `OPCUADataTransfer.cs` [41–42]. In this script, the communication via OPC UA with the server in the robot controller is also implemented, as described in section 3.4.2. The synchronization flags are always sent, that an action must be performed, and together with them, the command number is sent, which is used to control the case switch in the robot program. The development environment in Unity is shown in Figure 4.14.

A Vuforia tag is now introduced, which is essential for aligning the scene with the real environment when the application is running. Vuforia Engine is a SDK that enables anchoring digital content in the physical world. Vuforia's precise tracking capabilities ensure that the holographic overlays remain aligned with the physical

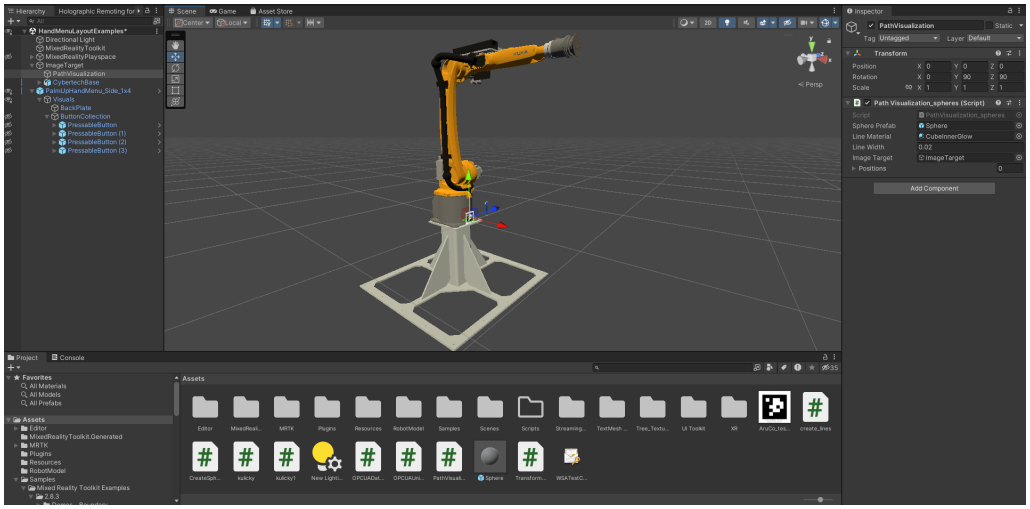

Figure 4.14. Screenshot from the Unity development environment.

machinery, regardless of environmental changes or different viewing angles. Vuforia uses markers or targets to place and sustain AR content in the real world. The target is some ArUco marker, which has to be added to the Vuforia database together with its actual dimensions. This will ensure that when the AR headset sees this tag in the real environment, all holographs are aligned to it in the real world. The advantage is that even if the tag is subsequently occluded, the holographs remain still visible and aligned [43]. The Vuforia tag locataed in the Unity scene and palm menu is shown in Figure 4.15.

If the user has already requested path generation, it will also be possible to visualize the sanding paths. The paths are visualized in the headset by rendering the individual points as little spheres and are linked with line segments. In this way, it is possible to see the points which the robot will traverse and how they are in sequence. In demand to align everything correctly, the OBJ models and these spheres must be in the tree hierarchy as children of this Vuforia tag.

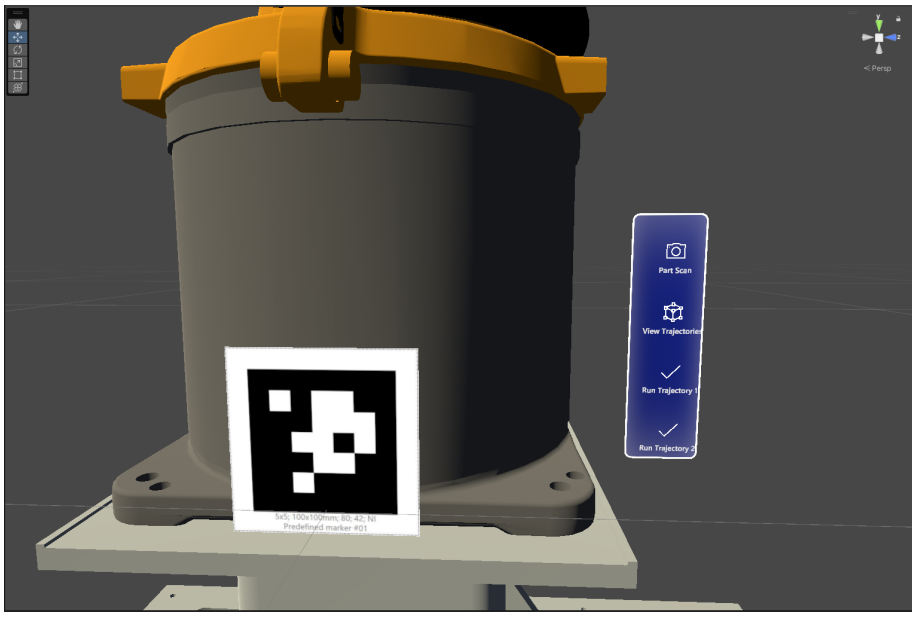

Figure 4.15. Detail of Vuforia tag and Palm menu in Unity.

4.9.3 Resulting visualization

The resulting visualization and user environment can now be presented in the AR headset. When the application is built in Unity and uploaded to Hololens 2, a stand-alone application is there created that runs separately from the application computer.

The palm menu is shown in Figure 4.16 and is displayed whenever the user places their palm in front of the AR headset. A hologram of the robot model and sanding spindle is shown in Figure 4.17. These images are screenshots directly from Hololens 2, so they show faithfully what the operator sees. Through the hologram the Vuforia tag by which the scene was aligned is also visible. It shows that the alignment inaccuracy is, at most, in the order of *mm*, which is reasonably sufficient to give the operator a good overview of the workplace.

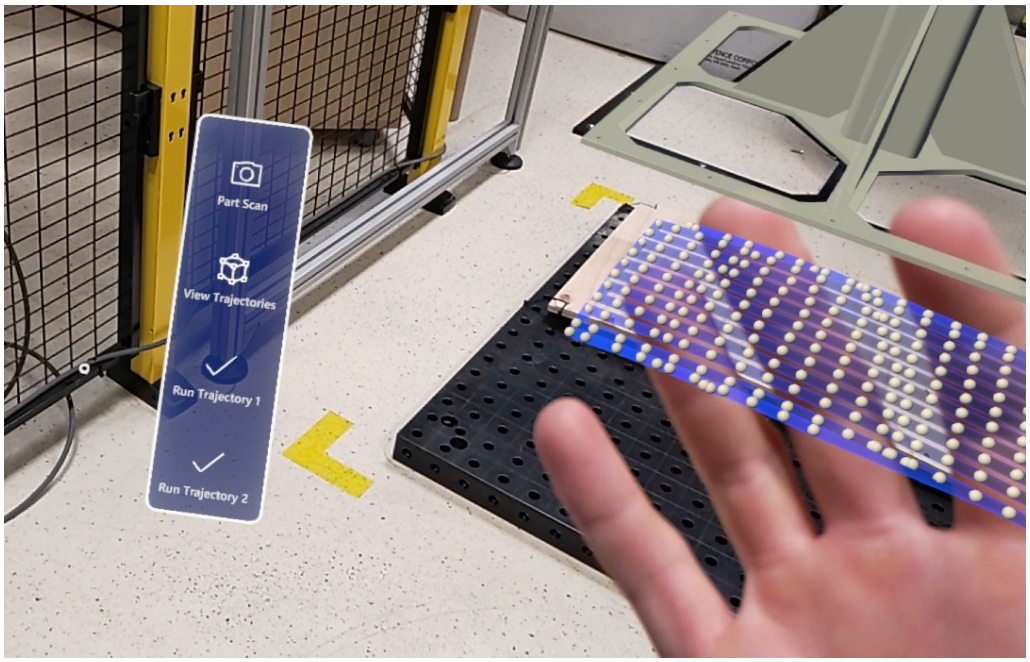

Figure 4.16. A pop-up menu in the palm of the hand to control the workplace.

The projected holograms of the zig-zag and spiral patterns can be seen in Figures 4.18 and 4.19. The individual points as spheres, and the line segments connecting them are clearly visible. Again, the inaccuracy is, at most, in the order of mm, giving the operator an excellent overview of how the sanding path will be executed by the robot.

Figure 4.17. Holograms of the robot's and sanding spindle's models in Hololens 2 headset.

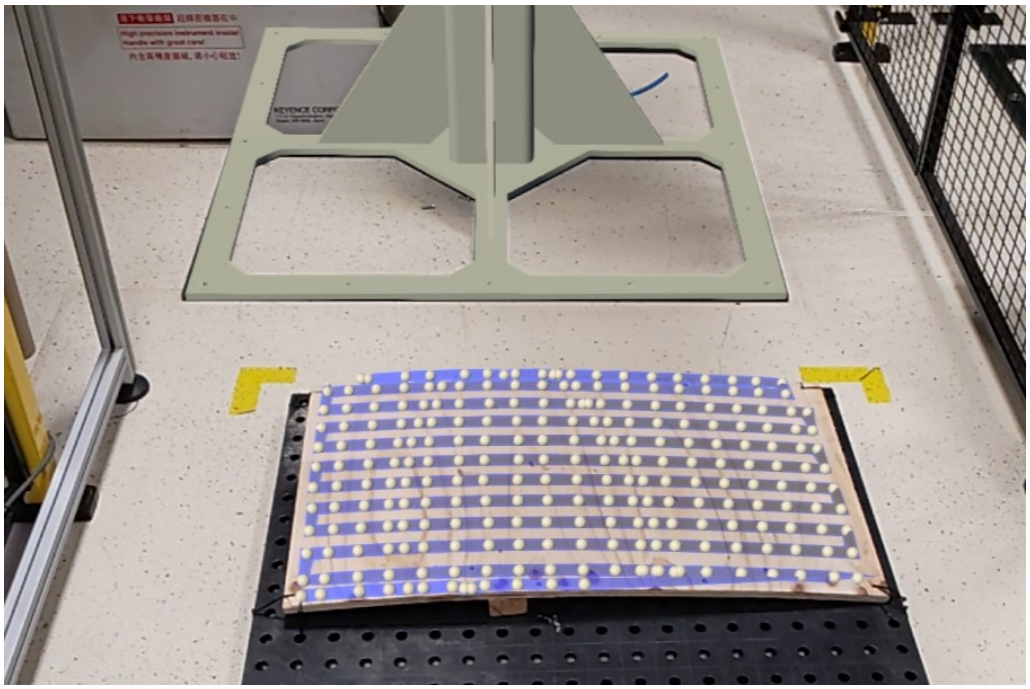

Figure 4.18. Hologram of zig-zag pattern projected onto real workpiece.

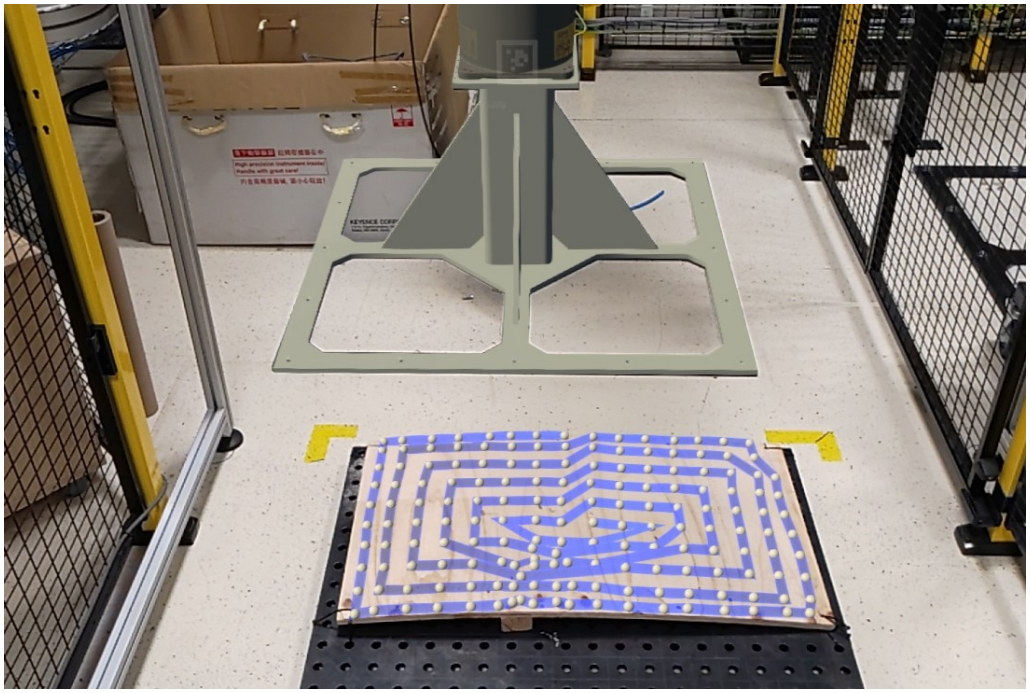

Figure 4.19. Hologram of spiral pattern projected onto real workpiece.

Chapter 5
Conclusion

The primary goal of this thesis was to enhance the efficiency and adaptability of robotic sanding systems using a 3D vision system coupled with force feedback control. Through meticulous research and development, this goal has been comprehensively achieved, marking a significant advancement in robotic automation technologies. Such a developed system can be deployed in smaller production where manufactured products are changed frequently. It can be used for sanding and polishing, depending on the required quantity of material to be removed.

5.1 Results summary

Key achievements of this research include the development of a robust methodology for point cloud processing and path generation, which ensures that the robotic manipulator can adaptively conform to variable convex surface topographies without human intervention. The implementation of zig-zag and spiral sanding patterns through calculated robotic paths not only maximizes coverage but also maintains a consistent force, which is critical for achieving uniform surface finishes. This capability significantly reduces the time and effort required to program the robot for new objects, enhancing the system's adaptability to varied manufacturing tasks.

The thesis established a robust control system that maintains a constant pressing force during the sanding process. This system dynamically adjusts the robot's path based on real-time feedback, ensuring consistent quality regardless of any imperfections or variations in the material being sanded. Together with this, a process for compensating the gravitational force that affects the force-torque sensor at resting state has been developed.

The research also highlighted the importance of precise workspace calibration. The procedure of obtaining the transformation matrix between the depth camera's coordinate system and the robot base's coordinate system, which serves as the base of the whole workplace, was introduced. This is important for the individual point clouds to be mapped to the common coordinate system.

Implementing the OPC UA protocol facilitated efficient and secure data transfer between the robotic controller and the application interfaces. This not only enhances the reliability of the system operations but also data integrity in industrial automation systems. This protocol is used to synchronize the process and data transfer between the AR headset and the robot controller and simultaneously for communication between the robot controller and the application computer, where all calculations and path generation take place.

Furthermore, the introduction of augmented reality (AR) technology in the form of the Hololens 2 headset provides an innovative user interface, enabling operators to visualize and dynamically adjust the sanding paths in real-time. This fusion of AR with robotic control represents a notable leap forward in interactive manufacturing

processes, potentially setting a new standard for industrial applications requiring high precision and adaptability.

The utilization of RoboDK to simulate the robotic paths and operations was pivotal in the project's development phase. It provided a safe and effective environment to test movements before running on a real robot, thereby minimizing the risks of running into singular configurations or collisions with the environment, which may have been neglected during path generation.

The methodologies and systems developed were rigorously tested in a real-world environment, specifically designed to emulate industrial conditions. Even the inaccuracies of the force-torque sensor measurements, caused by the random orbit movement of the sanding spindle, were successfully suppressed. This practical application demonstrated the system's effectiveness and reliability, confirming its readiness for broader industrial adoption.

5.2 Future work

The research presented in this thesis opens several routes for further exploration and development. To enhance the capabilities of the robotic sanding system and extend its applications.

Future work could also focus on developing algorithms that can generate sanding paths for even more complex geometries, potentially using advanced computational geometry techniques. It would be interesting to expand the range of possible patterning that advanced CAD/CAM software allows. It might also be useful to investigate the optimization of the sanding pattern, aiming to minimize the traveled path, which would lead to problems such as the Traveling salesman problem. This would be used on complex surfaces that can be perforated and would minimize the robot's travels between perforations.

Future research could also focus on developing more sophisticated adaptive control algorithms that further refine the force feedback mechanism. Implementing machine learning techniques could enable the system to predict and compensate for variations in material properties and wear on the sanding tools, optimizing the applied sanding force and speed in real-time based on the data collected during operations.

A model of the relation between the applied pressing force and the removed material would be needed. This would provide a better understanding of the robotic sanding process, and the user could choose the sanding force according to the need for the removed material. For this, a more accurate scanning device is needed. It would be necessary to have a clear view of the removed material layer on the workpiece before and after sanding.

It would also be useful to further expand the functionality of the AR headset app. The operator would be able to edit the designed paths directly in the headset. Before the robot executes the path, the operator could adjust the density of the patterning and choose the pressing force to better suit his/her specific application. Next, it would be useful to simulate the trajectory directly in the AR headset. The kinematics of the robot is well described in this thesis and could be used for this. In Unity, it is easy to animate the robot model, and the user can simulate the whole sanding cycle before sending a command to the real robot to run the path. This would bypass the simulation in RoboDK, which cannot be fully automated.

References

[1] Nils Bausch, David P. Dawkins, Regina Frei, and Susanne Klein. 3D Printing onto Unknown Uneven Surfaces. *IFAC-PapersOnLine*. 2016, 49 (21), 583–590. DOI 10.1016/j.ifacol.2016.10.664.

[2] *Virtual Reality & Augmented Reality in Industry*. Berlin, Heidelberg: Springer, 2011. ISBN 978-3-642-17375-2 978-3-642-17376-9. `http://link.springer.com/10.1007/978-3-642-17376-9`.

[3] Fengjie Tian, Chong Lv, Zhenguo Li, and Guangbao Liu. Modeling and control of robotic automatic polishing for curved surfaces. *CIRP Journal of Manufacturing Science and Technology*. 2016, 14 55–64. DOI 10.1016/j.cirpj.2016.05.010.

[4] Joshua Nguyen, Manuel Bailey, Ignacio Carlucho, and Corina Barbalata. *Robotic Manipulators Performing Smart Sanding Operation: A Vibration Approach*. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022. 2958–2964. `https://ieeexplore.ieee.org/document/9812029`.

[5] Alberto García, Luis Gracia, J. Ernesto Solanes, Vicent Girbés-Juan, Carlos Perez-Vidal, and Josep Tornero. Robotic assistance for industrial sanding with a smooth approach to the surface and boundary constraints. *Computers & Industrial Engineering*. 2021, 158 107366. DOI 10.1016/j.cie.2021.107366.

[6] Manuel Amersdorfer, and Thomas Meurer. Equidistant Tool Path and Cartesian Trajectory Planning for Robotic Machining of Curved Freeform Surfaces. *IEEE Transactions on Automation Science and Engineering*. 2022, 19 (4), 3311–3323. DOI 10.1109/TASE.2021.3117691. Conference Name: IEEE Transactions on Automation Science and Engineering.

[7] Fusheng Liang, Chengwei Kang, Zhongyang Lu, and Fengzhou Fang. Iso-scallop tool path planning for triangular mesh surfaces in multi-axis machining. *Robotics and Computer-Integrated Manufacturing*. 2021, 72 102206. DOI 10.1016/j.rcim.2021.102206.

[8] S. Ding, M. A. Mannan, A. N. Poo, D. C. H. Yang, and Z. Han. Adaptive iso-planar tool path generation for machining of free-form surfaces. *Computer-Aided Design*. 2003, 35 (2), 141–153. DOI 10.1016/S0010-4485(02)00048-9.

[9] Renan S. Freitas, Eduardo E. M. Soares, Ramon R. Costa, and Breno B. Carvalho. *High precision trajectory planning on freeform surfaces for robotic manipulators*. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. 3695–3700. `https://ieeexplore.ieee.org/abstract/document/8206216`. ISSN: 2153-0866.

[10] R. K. Beatson, W. A. Light, and S. Billings. Fast Solution of the Radial Basis Function Interpolation Equations: Domain Decomposition Methods. *SIAM Journal on Scientific Computing*. 2001, 22 (5), 1717–1740.

DOI 10.1137/S1064827599361771. Publisher: Society for Industrial and Applied Mathematics.

[11] Ramadhan Abdo Musleh Alsaidi. Two Methods for Surface /Surface Intersection Problem Comparative Study . *International Journal of Computer Applications* . 2014 , 92 (5), 1-8 . DOI 10.5120/16002-4989 .

[12] Gang Wang, Wenlong Li, Cheng Jiang, Dahu Zhu, Zhongwei Li, Wei Xu, Huan Zhao, and Han Ding. Trajectory Planning and Optimization for Robotic Machining Based On Measured Point Cloud. *IEEE Transactions on Robotics*. 2022, 38 (3), 1621–1637. DOI 10.1109/TRO.2021.3108506. Conference Name: IEEE Transactions on Robotics.

[13] Ali Munira, Nur Najmiyah Jaafar, Abdul Aziz Fazilah, and Z. Nooraizedfiza. *Review on Non Uniform Rational B-Spline (NURBS): Concept and Optimization*. In: *Manufacturing Engineering*. Trans Tech Publications Ltd, 2014. 338–343.

[14] Marsh Duncan. *Homogeneous Coordinates and Transformations of Space*. London: Springer London, 2005. ISBN 978-1-84628-109-9. https://doi.org/10.1007/1-84628-109-1_3.

[15] *Open3D primary (5c982c7) documentation*. https://www.open3d.org/docs/latest/index.html.

[16] *scipy.spatial.ConvexHull — SciPy v1.13.0 Manual*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.ConvexHull.html.

[17] Max Yiye Cao, Stephen Laws, and Ferdinando Rodriguez y Baena. Six-Axis Force/Torque Sensors for Robotics Applications: A Review. *IEEE Sensors Journal*. 2021, 21 (24), 27238–27251. DOI 10.1109/JSEN.2021.3123638. Conference Name: IEEE Sensors Journal.

[18] Yongqiang Yu, Ran Shi, and Yunjiang Lou. Bias Estimation and Gravity Compensation for Wrist-Mounted Force/Torque Sensor. *IEEE Sensors Journal*. 2022, 22 (18), 17625–17634. DOI 10.1109/JSEN.2021.3056943. Conference Name: IEEE Sensors Journal.

[19] Thomas Kugelstadt. *Op Amps for Everyone (Third Edition), Chapter 20 - Active Filter Design Techniques*. Boston: Newnes, 2009. ISBN 978-1-85617-505-0. https://www.sciencedirect.com/science/article/pii/B978185617505000020X.

[20] Daniel E. Rivera, Manfred Morari, and Sigurd Skogestad. *Internal model control: PID controller design*. 1986. https://doi.org/10.1021/i200032a041. Archive Location: world Publisher: American Chemical Society.

[21] *KUKA.RobotSensorInterface 4.1*. Augsburg, Germany: KUKA Deutschland GmbH, 2019. https://www.kuka.com/.

[22] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics*. London: Springer, 2009. ISBN 978-1-84628-641-4 978-1-84628-642-1. http://link.springer.com/10.1007/978-1-84628-642-1.

[23] *KUKA KR 8 R1620 HP*. Augsburg, Germany: KUKA Deutschland GmbH, 2021. https://www.kuka.com/-/media/kuka-downloads/imported/8350ff3ca11642998dbdc81dcc2ed44c/0000258989_en.pdf.

[24] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir. Inverse Kinematics Techniques in Computer Graphics: A Survey. *Computer Graphics Forum*. 2018, 37 (6), 35–58. DOI 10.1111/cgf.13310. : https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13310.

[25] *Unified Architecture*.
https://opcfoundation.org/about/opc-technologies/opc-ua/.

[26] Salvatore Cavalieri, and Ferdinando Chiacchio. Analysis of OPC UA performances. *Computer Standards & Interfaces*. 2013, 36 (1), 165–177. DOI 10.1016/j.csi.2013.06.004.

[27] *Python OPC-UA Documentation — Python OPC-UA 1.0 documentation*.
https://python-opcua.readthedocs.io/en/latest/.

[28] *UA Bundle SDK .NET: Unified Automation .NET OPC UA SDK API Reference*.
https://documentation.unified-automation.com/uasdknet/2.1.2/html/index.html.

[29] D.W. Eggert, A. Lorusso, and R.B. Fisher. Estimating 3-D rigid body transformations: a comparison of four major algorithms. *Machine Vision and Applications*. 1997, 9 (5), 272–290. DOI 10.1007/s001380050048.

[30] *CV-X Series, User's Manual*. Osaka, Japan: KEYENCE CORPORATION, 2022.
https://www.keyence.com.

[31] Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*. 1966, 31 (1), 1–10. DOI 10.1007/BF02289451.

[32] *KUKA System Software 8.7*. Augsburg, Germany: KUKA Deutschland GmbH, 2023.
https://www.kuka.com/.

[33] *FT-AXIA*.
https://schunk.com/us/en/automation-technology/force/torque-sensors/ft-axia/c/PGR_3907.

[34] *AOV*.
https://schunk.com/us/en/automation-technology/machining-tools/aov/c/PGR_5498.

[35] *3D VISION-GUIDED ROBOTICS, CONNECTION MANUAL, KUKA Roboter GmbH EDITION*. Osaka, Japan: KEYENCE CORPORATION, 2020.
https://www.keyence.com.

[36] *socket — Low-level networking interface*. Python documentation,
https://docs.python.org/3/library/socket.html.

[37] *ftplib — FTP protocol client*. Python documentation,
https://docs.python.org/3/library/ftplib.html.

[38] *Basic Guide - RoboDK Documentation*.
https://robodk.com/doc/en/Basic-Guide.html##Guide.

[39] *RoboDK API for Python — RoboDK API Documentation*.
https://robodk.com/doc/en/PythonAPI/index.html.

[40] *HoloLens 2—Overview, Features, and Specs Microsoft HoloLens*.
https://www.microsoft.com/en-us/hololens/hardware.

[41] Kevin Semple, MaxWang-MS, Annebelle Olminkhof, Mohit Patel, Vinnie Tieto, Christopher McClister, and Qian Wen. *MRTK2-Unity Developer Documentation - MRTK 2*. 2022.
https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05.

[42] Unity Technologies. *Unity - Manual: Unity User Manual 2022.3 (LTS)*.
https://docs.unity3d.com/Manual/index.html.

[43] *Getting Started with Vuforia Engine in Unity Vuforia Library*.
https://developer.vuforia.com/library/getting-started/getting-started-vuforia-engine-unity.

Appendix A
Thesis assignment

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kubáček Václav** Personal ID number: **483422**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master’s thesis details

Master’s thesis title in English:

Utilization of 3D Vision System for Robotic Sanding with Force Feedback Control

Master’s thesis title in Czech:

Využití systému 3D vidění pro robotické broušení se silovou zpětnou vazbou

Guidelines:

The aim of the thesis is to design a robotic system that automatically generates trajectories for polishing convex surfaces using 3D vision. The robotic system is equipped with a polishing spindle and a force-torque sensor on the robot flange, which will be used for feedback force control of the robot.
1. Get acquainted with the function of the individual components and use this information to design the polishing process. Use information from recommended literature to familiarize yourself with the algorithms used.
2. Develop an algorithm for processing point cloud data from a 3D vision system. Extend the algorithm to generate robot trajectories, for which feasibility verification must then be performed using simulation.
3. Design a feedback system to correct robot trajectories based on the data obtained from the force-torque sensor.
4. Implement the software components on the robot workstation required to test the system.
5. Test the robotic system and optimize the polishing process. Choose the appropriate approach to visualize robot trajectories.

Bibliography / sources:

[1] Amersdorfer, Manuel, and Thomas Meurer. “Equidistant Tool Path and Cartesian Trajectory Planning for Robotic Machining of Curved Freeform Surfaces.” IEEE Transactions on Automation Science and Engineering. Institute of Electrical and Electronics Engineers (IEEE), October 2022. https://doi.org/10.1109/tase.2021.3117691.
[2] Nguyen, Joshua, Manuel Bailey, Ignacio Carlucho, and Corina Barbalata. “Robotic Manipulators Performing Smart Sanding Operation: A Vibration Approach.” 2022 International Conference on Robotics and Automation (ICRA). IEEE, May 23, 2022. https://doi.org/10.1109/icra46639.2022.9812029.
[3] Veronika Putz, Michael Stangl, Christian Kohlberger, Ronald Naderer, Computer Vision Approach for the Automated Tool Alignment of an Orbital Sanding Robot, IFAC-PapersOnLine, Volume 52, Issue 15, 2019, Pages 19-24, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2019.11.643.
[4] Lakshminarayanan, S., Kana, S., Mohan, D.M. et al. An adaptive framework for robotic polishing based on impedance control. Int J Adv Manuf Technol 112, 401–417 (2021). https://doi.org/10.1007/s00170-020-06270-1
[5] Fengjie Tian, Chong Lv, Zhenguo Li, Guangbao Liu, Modeling and control of robotic automatic polishing for curved surfaces, CIRP Journal of Manufacturing Science and Technology, Volume 14, 2016, Pages 55-64, ISSN 1755-5817, https://doi.org/10.1016/j.cirpj.2016.05.010.

Name and workplace of master's thesis supervisor:

Ing. Tomáš Jochman Testbed CIIRC

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

Date of master's thesis assignment: **16.01.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Tomáš Jochman	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master'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 master'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
Abbreviations

API	Application programming interface
AR	Augmented reality
CAD	Computer aided design
CAM	Computer aided manufacturing
csys	Coordinate system
DH parameters	Denavit-Hartenberg parameters
DOF	Degrees of Freedom
F/T sensor	Force-torque sensor
FK	Forward kinematic task
FTP	File Transfer Protocol
IK	Inverse kinematic task
KD Tree	K-Dimensional Tree
KRL	KUKA Robot Language
LIN	Linear movement
OPC UA	Open Platform Communications Unified Architecture
PTP	Point-to-Point movement
RSI	Robot Sensor Interface
SDK	Software development kit
SVD	Singular Value Decomposition
TCP	Tool center point