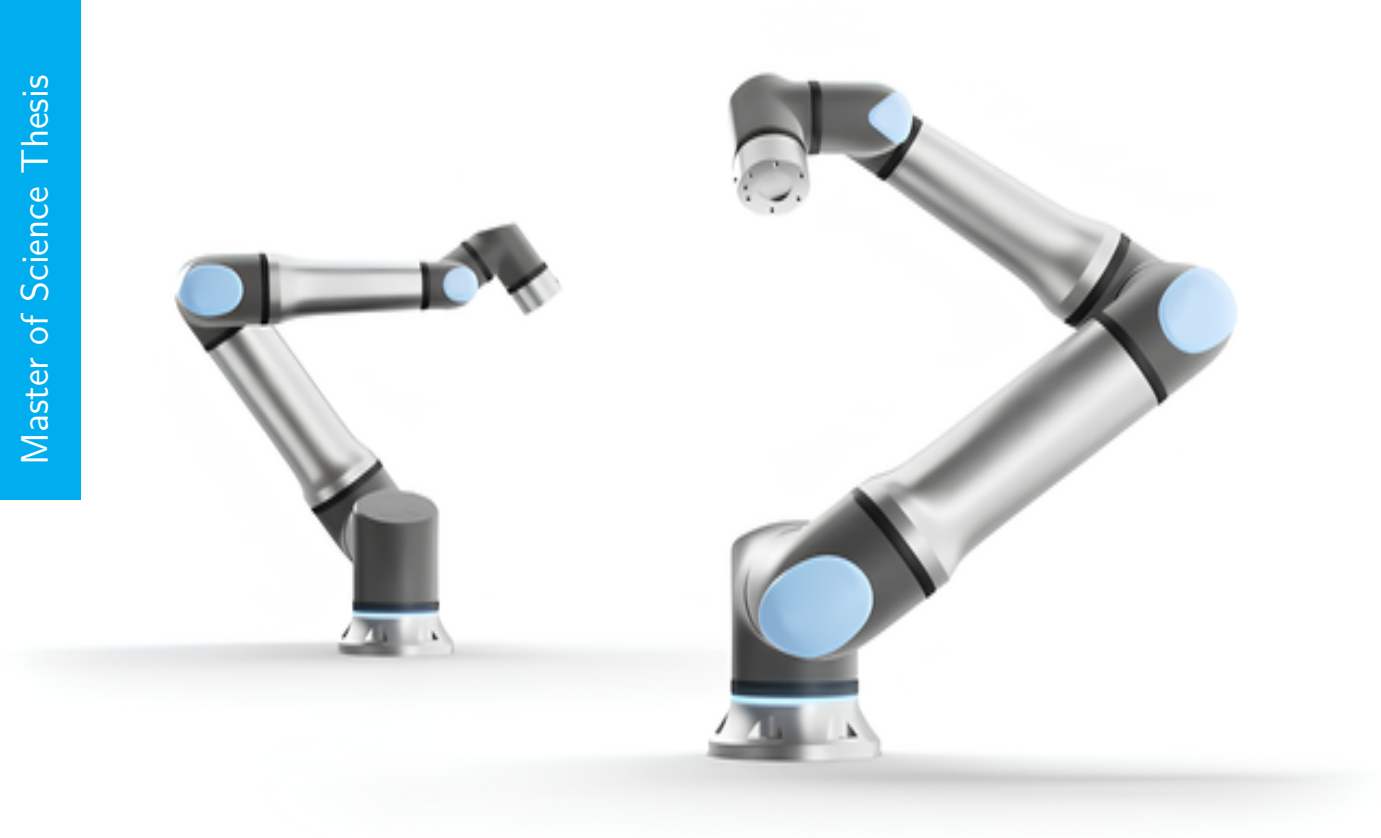
Automated Robotic System for Precision NVH Testing: Recognition, Targeting, and Impact Optimization

Junhwi Mun

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Automated Robotic System for Precision NVH Testing: Recognition, Targeting, and Impact Optimization

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Junhwi Mun

March 18, 2025

Faculty of Mechanical Engineering (ME) · Delft University of Technology

Abstract

Noise, Vibration, and Harshness (NVH) testing plays a pivotal role in assessing the performance and quality of automotive components and machinery. Traditional manual methods often suffer from inconsistency, inefficiency, and a high dependency on operator skill, leading to challenges in ensuring precise impact force, accurate targeting, and optimal impact angles. This thesis introduces an integrated framework that automates NVH testing through the synergistic use of advanced normal vector extraction and multi-objective path planning techniques.

The first contribution of this work is a novel normal vector extraction method grounded in advanced 3D signal processing—a fundamental component of systems and control. This approach robustly computes surface orientations from noisy point cloud data while significantly reducing computational overhead and minimizing data requirements. By leveraging these core 3D signal processing techniques, the method ensures precise alignment of the robotic arm's end effector, enabling impacts to be delivered perpendicularly to the target surface and thereby enhancing the reliability of vibration measurements.

Complementing this, the thesis proposes a multi-objective path planning framework that employs rigorous optimization techniques—another cornerstone of systems and control curricula—to navigate complex, dynamic, and obstacle-rich environments. By simultaneously addressing factors such as collision avoidance, accessibility, troubleshooting minimization, and targeting accuracy, the framework optimally selects trajectories for the robotic arm, ensuring consistent and repeatable testing even in the presence of irregular component geometries.

Extensive simulations and real-world experiments demonstrate that the integrated approach not only enhances the precision of NVH measurements but also reduces testing time and operational costs. Overall, this research paves the way for more intelligent, efficient, and adaptable automated NVH testing systems suitable for a wide range of industrial applications.

Table of Contents

Acknowledgements

Throughout my academic and professional journey, I have been fortunate to receive guidance and support from several remarkable individuals. I wish to express my deepest gratitude to my academic supervisor, Prof. Dr. Meichen Guo, for her inspiring mentorship and unwavering encouragement during my studies at Delft University of Technology. Her expert guidance has been instrumental in the successful completion of this work.

I also extend my sincere thanks to my company supervisors, Steven Klaassen and Jelle Boelens, whose practical insights and dedicated mentorship have helped bridge the gap between theory and practice. Their continuous support and constructive feedback have significantly enriched both my professional development and this project.

Lastly, I am profoundly grateful to my peers and friends, whose steadfast support and companionship have made this journey not only more enjoyable but also deeply rewarding.

Delft, University of Technology — Junhwi Mun
March 18, 2025

Chapter 1

Introduction

The field of NVH (Noise, Vibration, and Harshness) is extensively and actively applied across multiple industries, particularly in the automotive sector, where it serves as a key performance indicator. NVH refers to the study and control of mechanical vibrations, airborne noise, and the subjective feel of harshness experienced during the operation of a vehicle or machinery. It encompasses both objective measurements and subjective evaluations that influence customer satisfaction, comfort, and the overall perception of quality.

NVH performance is shaped by a variety of factors, including material properties and structural design. Traditionally, NVH measurements are conducted on fully assembled components or systems. However, addressing NVH at the pre-assembly stage is equally vital. This involves selecting materials and shapes for individual components that meet stringent NVH criteria, which necessitates a combination of advanced methods and experimental validation to ensure optimal outcomes.

NVH Testing Approach

To optimize NVH performance, pre-assembly tests are conducted on components provided in advance. These tests provide valuable guidelines for selecting materials and component designs. Key target areas, such as engines or tire wheels, are identified where vibrations are most likely to occur. To measure vibrations, these components are suspended using elastic bands to isolate them from external interference, and vibration sensors are strategically attached near the target points. An experiment setup with the test component can be found in Fig.1-1.

Figure 1-1: Experiment setup in real life

The testing process involves delivering controlled impacts to the target points using a small hammer. However, manually operated impact testing presents several challenges that can compromise the accuracy and repeatability of vibration measurements:

Importance of the Three Factors in Manual NVH Testing

The three factors—impact angle, target accuracy, and impact force—are crucial in determining the reliability and precision of vibration measurements. Their importance can be summarized as follows:

- **Impact Angle:** The angle at which the hammer strikes the target significantly influences the accuracy of the vibration data. A non-perpendicular impact can introduce undesired lateral forces, causing inconsistent excitation of the component and skewing the vibration measurements. This can result in erroneous identification of natural frequencies and damping properties.
- **Target Accuracy:** Precise targeting ensures that the hammer consistently strikes the designated points where vibrations are to be measured. Deviations from the target point may cause the measured response to differ from the actual vibrational characteristics of the component, leading to unreliable data. Accurate targeting is especially important for complex geometries or small test components where the exact point of impact is critical.
- **Impact Force:** Consistent impact force is essential for generating reproducible excitation levels. Variations in force can lead to discrepancies in the amplitude of the measured vibrations, making it difficult to compare results across multiple tests. Inconsistent force application also complicates the calibration of vibration sensors and the interpretation of data.

These challenges often require testers to perform multiple impacts—sometimes exceeding ten per target point—to obtain reliable data. This process is not only time-consuming but also prone to human error.

Addressing these factors is essential for obtaining high-quality, repeatable vibration measurements. Automating the impact process using robotic systems can help overcome these challenges by ensuring precise control over the angle, accuracy, and force of each impact.

1-1 Research Problem of Automation with Robotic Systems

Manual testing in NVH (Noise, Vibration, and Harshness) assessments often suffers from limitations such as inconsistency, inefficiency, and dependence on highly skilled professionals. To overcome these challenges, we propose an automated vibration testing system utilizing a robotic arm. This system aims to enhance precision, consistency, and overall testing efficiency by automating the hammering process. By ensuring each impact is delivered with uniform force, direction, and distance, the robotic arm minimizes the number of impacts required per target point, thereby reducing the reliance on manual intervention and increasing the reliability of the testing outcomes.

System Design and Implementation

The proposed automated system equips the robotic arm with a small hammer as its end effector. The robot is programmed to position the hammer at a fixed distance from the target point, maintaining a perfectly perpendicular orientation relative to the surface under test. Controlled impacts are then delivered with consistent force to gather high-quality vibration data. A simplified simulation setup demonstrating the robotic arm with an automatic hammer is illustrated in Fig.1-2.

Challenges in Deployment

Deploying an automated robotic system for vibration testing entails addressing several engineering and planning challenges:

- **Collision Avoidance:** The robotic arm must maneuver around obstacles such as sensor cables and the intricate geometry of components to reach target points without interference.
- **Handling Complex Geometries:** Many components possess non-planar or irregular shapes, necessitating precise hammer positioning to maintain perpendicular alignment with diverse target surfaces.
- **Ensuring Consistency and Repeatability:** The system must deliver uniform impact force and direction across all target points, irrespective of the component's orientation, to ensure reliable and repeatable testing results.

A further complication arises from the variability of test components. Unlike standardized items typically used in manufacturing settings, the components in NVH testing are often confidential, custom-designed, and not yet mass-produced. This variability renders pre-programmed, shape-specific solutions impractical. Consequently, the automation system must

be adaptable, capable of handling a wide range of component geometries without extensive reprogramming. Unlike factory automation, which can leverage detailed product information, our system prioritizes flexibility to accommodate new and unique component designs for each test.

Research Problem

The integration of robotic systems into NVH testing introduces several critical research challenges essential for ensuring precision, reliability, and adaptability. This study focuses on addressing the following key problems:

1. **Fast and Robust Extraction of Normal Vectors from Point Clouds**
   Accurately extracting the normal vectors of target surfaces is crucial for orienting the robotic arm's end effector perpendicular to the target, ensuring optimal impact force and direction. However, this extraction from point cloud data is computationally demanding and susceptible to errors due to noise, occlusions, and irregular geometries. Developing a method that is both rapid and resilient is vital for real-time applications and accommodating the diverse shapes of components. This requires advanced 3D data processing techniques that enhance precision while minimizing processing time.
2. **Trajectory Optimization and Adaptability for Diverse Component Geometries**
   Optimizing the robotic arm's trajectory for NVH testing demands balancing multiple objectives such as collision avoidance, accessibility, and targeting accuracy, all while accommodating the unique and often irregular geometries of custom-designed test components. The trajectory planning must navigate around obstacles—including sensor cables and complex component shapes—ensuring that the hammer consistently aligns perpendicularly to the target surface for optimal impact. Simultaneously, as NVH components frequently feature non-planar surfaces and varied designs, the automation system must be capable of dynamically adapting and recalibrating to new configurations without manual intervention.

By tackling these research problems, this study aims to bridge the gap between traditional manual NVH testing and a fully automated solution, ultimately improving the quality and efficiency of vibration measurement processes.

1-2 Outline of the thesis

This thesis is structured into two main sections, each focusing on a distinct yet interconnected aspect of automating NVH testing with robotic systems:

Normal Vector Extraction

The first section delves into the methodologies for extracting surface normal vectors from point cloud data, a fundamental task in ensuring that the robotic arm's end effector aligns

correctly with the target surface. We introduce a novel approach for normal vector extraction that is optimized for real-time operation and non-planar components.

Multi-Objective Optimization for selecting optimal path

The second section addresses the complexities of path planning for the robotic arm, particularly focusing on balancing multiple objectives such as collision avoidance, accessibility, and targeting accuracy. Traditional path planning often emphasizes single-objective optimization, such as minimizing travel time or avoiding obstacles. However, real-world NVH testing requires a balance between multiple objectives to ensure both safety and adaptability. We propose a multi-objective optimization framework based on sampling-based path planners, tailored for robotic arms that must frequently adapt to new test components and dynamic environments. This approach ensures safe navigation around varying obstacles while optimizing for performance metrics relevant to specific NVH testing applications.

Organization of the Thesis

Following this introduction, the thesis is organized as follows:

- **Chapter 2: Relevant Work**: This chapter reviews existing literature and related methodologies in the field.
- **Chapter 3: Preliminary**: This chapter introduces the fundamental concepts and background information necessary for understanding the subsequent sections.
- **Chapter 4: Normal Vector Extraction**: This chapter details the traditional PCA method and its limitations, followed by our proposed approach. It also describes the integration of the depth camera (Intel RealSense D435) noise model and presents real-world experiments conducted using the same camera model as in the simulation.
- **Chapter 5: Multi-Objective Optimization for selecting optimal path**: This chapter discusses the challenges associated with basic sampling-based path planners and how multiple objectives are employed to address these challenges. It further includes simulations of a robot arm using the GAZEBO simulator.
- **Chapter 6: Conclusion**: This chapter evaluates the overall effectiveness of the accurate normal vector extraction and multi-objective optimization for optimal path planning. It also discusses the limitations of the current work and outlines directions for future research.

Through this structured approach, the paper aims to provide a clear and comprehensive understanding of the advancements in normal vector extraction and path planning, highlighting their interdependencies and their collective contribution to enhancing automated NVH testing systems.

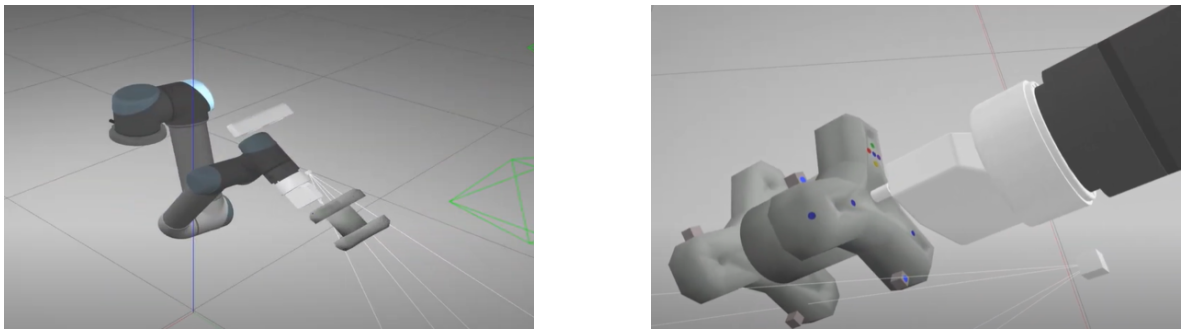


Figure 1-2: Simple automated robotic system with blue target points in simulation

Chapter 2

Relevant Work

The fields of 3D point cloud analysis and autonomous navigation have grown tremendously in recent years. With this growth, two challenges stand out: accurately extracting surface normals and designing efficient path planning algorithms. Surface normals play a vital role in tasks like object recognition and robotic manipulation, while effective path planning is essential for safe and efficient movement. In this chapter, we take you on a journey through the latest approaches in these areas—starting with traditional geometric methods and cutting-edge deep learning techniques for normal extraction, and then moving on to various sampling-based and multi-objective strategies for path planning.

2-1 Point cloud processing with Normal Vector Extraction

Extracting surface normals accurately is one of the first and most important steps in processing 3D data. Over time, methods have evolved from classical statistical approaches to modern deep learning techniques, and researchers are even incorporating sensor-specific details to boost accuracy.

Robust Geometric Processing Frameworks

Early work in this area relied on traditional geometric methods. For example, [24] introduced a technique based on selective normal space exploration that smartly samples local neighborhoods to better handle irregular data. Similarly, [29] proposed an iterative principal component analysis (PCA) method that continually refines local surface estimates by updating neighborhood statistics—helping to reduce noise as illustrated in Figure.2-1.

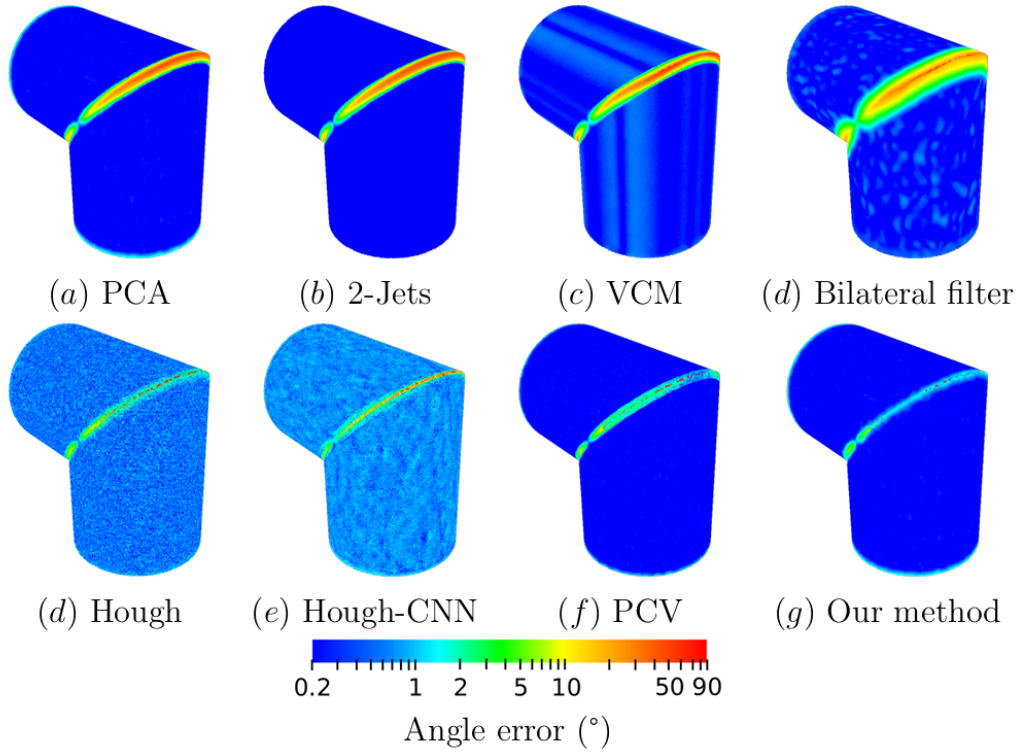


Figure 2-1: Iterative PCA given in [29] shows the accurate edge detection in the figure (g)

For outdoor settings, where LiDAR data is often affected by environmental noise, [8] developed a PCA-based denoising algorithm that suppresses outliers while preserving the true geometry. In scenarios that demand high precision, such as robotic manipulation, [14] crafted a method specifically designed to provide reliable normal estimates for guiding robotic arms. And to address real-time challenges, [5] introduced a fast algorithm for computing surface normals directly from range images. Together, these works illustrate how our methods have evolved from basic statistical techniques to more robust and adaptive approaches.

Deep Learning Architectures: From Local Features to Global Context

The emergence of deep learning has fundamentally transformed the processing of 3D point clouds. Modern data-driven architectures, particularly convolutional neural networks (CNNs), are engineered to learn hierarchical representations that capture both fine-grained local features and broader global contexts. For example, [10] provides a comprehensive review of deep learning techniques applied to point cloud completion and analysis, detailing the modifications necessary to address the irregular nature of 3D data. This study compares six different methods, categorizing 34 techniques in detail, while also discussing limitations such as noise, surface smoothness issues, and uneven point distribution.

Additionally, the survey in [19] outlines the evolution from the pioneering PointNet model to recent variants that integrate local feature extraction with global feature aggregation. This

combined approach enhances model robustness against common challenges like occlusions and noise, making deep networks a powerful tool for tasks such as normal vector estimation.

2-2 Path Planning optimization

Path planning is the heartbeat of autonomous systems, ensuring they can navigate safely and efficiently. It involves creating trajectories that avoid collisions while meeting constraints like time, energy, and safety. Researchers have explored a range of methods—from sampling-based approaches to sophisticated multi-objective optimization strategies—to tackle this challenge.

Sampling-Based Approaches

Sampling-based motion planning methods have become very popular due to their scalability and robustness in high-dimensional spaces. [26] provides a detailed look at techniques such as Probabilistic Roadmaps (PRM) and Rapidly-exploring Random Trees (RRT). These methods work by randomly sampling the configuration space to build a network of feasible paths—especially useful in environments where obstacles aren't easily modeled.

Taking this a step further, [27] introduced an experience-driven approach that leverages past experiences to guide the sampling process toward more promising areas, making planning more efficient in repetitive or structured tasks. Additionally, [28] developed asymptotically optimal algorithms like RRT*, which not only find a feasible path quickly but also improve the solution over time until it reaches an optimal state based on a defined cost function.

Applications in Mobile Robotics and UAV

Sampling-based path planning is widely applied across various domains. In industrial automation, for example, [30] demonstrated an intelligent robotic system that integrates target detection with advanced path planning algorithms for automated part placement in precast concrete manufacturing, achieving high levels of precision and efficiency.

Mobile robotics also greatly benefits from robust path planning frameworks. Reviews in [12] and [31] discuss trajectory planning algorithms that address dynamic environments, real-time processing requirements, and uncertainties. These works illustrate how traditional planning techniques are evolving to meet the demands of modern mobile robotic platforms. In the context of unmanned aerial vehicles (UAVs), unique operational constraints and mission-critical requirements call for specialized approaches. As described in [4], optimization-based methods for UAV path planning focus on ensuring safety, reducing energy consumption, and adhering to strict mission duration limits.

Multi-Objective Optimization Algorithms

When planning must balance multiple, often conflicting objectives, multi-objective optimization (MOO) becomes essential. For instance, [25] proposed a framework for path planning in multi-robot systems using an enhanced genetic algorithm that concurrently minimizes travel

time and collision risk. In a similar vein, [2] applied elitist non-dominated sorting genetic algorithms to derive Pareto-optimal solutions, while [20] investigated particle swarm optimization (PSO) variants for addressing multi-objective challenges. These studies underscore the importance of considering multiple performance metrics in complex, dynamic environments.

An overall comparison of various MOO algorithms is presented in [1]. In this study, five MOO algorithms and one random algorithm were evaluated using four performance metrics, as shown in Figure 2-2. It is important to note that several hybrid algorithms were excluded from this comparison. Ultimately, selecting the most appropriate algorithm depends on the specific scenario and the objectives at hand.

	BiMADS	MOPSO	MOGA	NSGA-III	MOGB	Random
Computational time (seconds)	26.16	10.10	5.59	44.86	113.11	4.28
Number of outperformances	22	0	0	3	0	0
Average normalized performance	0.984	0.978	0.951	0.842	0.961	0.960
Standard deviation of normalized performance	0.0492	0.0505	0.0527	0.1018	0.0555	0.0558

Figure 2-2: Comparison of Multi-Objective Optimization (MOO) algorithms based on four performance metrics

Summary

In summary, the literature on normal vector extraction and path planning reveals a continuous evolution of ideas. In 3D point cloud analysis, methods have advanced from classical geometric processing to sophisticated deep learning and sensor fusion techniques that offer greater accuracy and reliability. Meanwhile, path planning has grown from traditional sampling-based methods to advanced multi-objective frameworks with wide-ranging applications—from industrial automation to mobile robotics and UAV operations. These developments demonstrate the exciting potential of hybrid systems that blend classical and modern approaches, paving the way for more robust and efficient autonomous systems.

Chapter 3

Preliminary

3-1 Concept of Point Cloud

A *point cloud* is a collection of data points defined in a three-dimensional coordinate system. Each point in the cloud represents a precise location in space and, together, these points capture the external surface of objects or entire scenes. Point clouds are widely used in various fields such as robotics, computer vision, photogrammetry, and 3D scanning.

- **Acquisition:** Point clouds are generated using different sensor technologies, including:
  - **LiDAR (Light Detection and Ranging):** Uses laser pulses to measure distances.
  - **Stereo Cameras:** Capture depth information by comparing images from multiple viewpoints.
  - **Structured Light:** Projects known patterns on surfaces and measures the deformation to determine depth.
  - **Time-of-Flight Sensors:** Estimate distances based on the time taken for a light signal to return to the sensor.

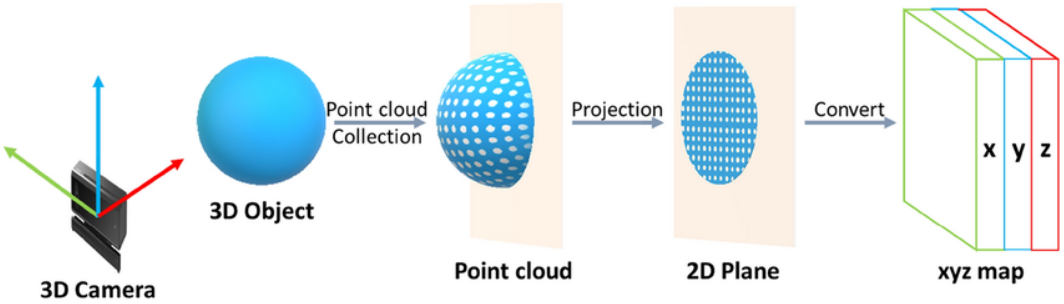


Figure 3-1: Data representation in (x,y,z) map from depth camera [6]

- **Data Representation:** Typically, each point is represented by its spatial coordinates (x, y, z) through the process illustrated in Fig.3-1. In addition to spatial data, points may also include attributes such as color (RGB values), intensity, or even surface normals, which provide further information about the scanned scene or object.
- **Applications:** Point clouds serve as the foundation for many advanced applications, including:
  - **3D Reconstruction:** Building detailed three-dimensional models of objects or environments.
  - **Robotics:** Assisting in navigation, obstacle detection, and manipulation tasks by providing a spatial representation of the surroundings.
  - **Geospatial Analysis:** Creating digital terrain models and mapping large-scale landscapes in remote sensing.
  - **Computer Graphics and Animation:** Enhancing realism in virtual scenes and enabling accurate simulations.
- **Challenges:** Despite their versatility, point clouds pose several challenges:
  - **Noise and Outliers:** Sensor inaccuracies and environmental factors can introduce noise and erroneous points that need to be filtered out.
  - **Varying Density:** The density of points may vary significantly within the cloud, complicating processes such as surface reconstruction and feature extraction.
  - **Computational Complexity:** Managing and processing large-scale point clouds (often containing millions of points) require efficient algorithms and high computational power.

In summary, point clouds provide a flexible and detailed representation of three-dimensional environments, enabling a wide range of applications from 3D modeling to autonomous navigation. Their inherent challenges drive ongoing research in data processing, analysis, and efficient algorithm design.

3-2 Normal Vector Extraction using PCA

Point cloud data, which consists of a set of 3D coordinates, often requires the estimation of local surface orientations for applications in computer vision, robotics, and surface analysis. One robust method for obtaining these orientations is by extracting normal vectors via Principal Component Analysis (PCA). In this section, we describe the step-by-step process to extract a local normal vector from a point cloud, emphasizing the flow of ideas from data preparation to the final normal extraction.

To extract a reliable normal vector, the procedure is divided into several stages: defining the problem, centering the data, constructing the covariance matrix, performing eigenvalue decomposition, and finally extracting the normal vector. The following elaborates on each of these stages.

Problem Setup:
Consider a set of points in $\mathbb{R}^3$,

$$\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\},$$

which represents our entire point cloud. For a specific point $\mathbf{p}$, the goal is to compute a normal vector that best represents the orientation of the local surface. To do so, we first identify its k nearest neighbors:

$$\mathcal{N} = \{\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_k\}.$$

These neighbors are assumed to lie on or close to the same surface. A plane fitted to these points serves as a local approximation of the surface, as illustrated in Fig. 3-2.

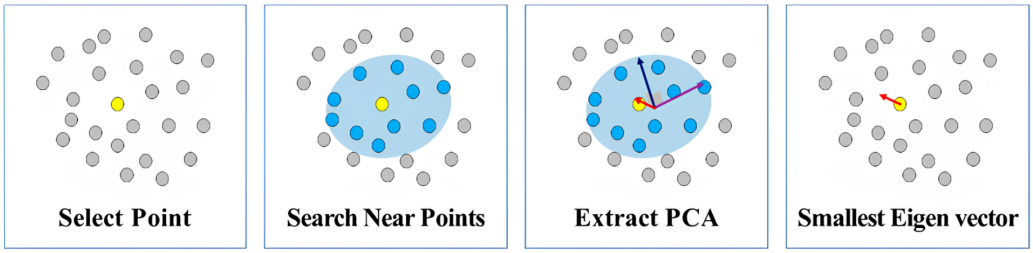


Figure 3-2: Simple steps for normal vector extraction using PCA [17]

Data Centering:
Before performing PCA, it is essential to center the neighborhood data to ensure that the analysis is invariant to translation. We compute the centroid of the neighborhood as

$$\mathbf{c} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{q}_i.$$

Subsequently, each neighbor is recentered by subtracting the centroid:

$$\mathbf{r}_i = \mathbf{q}_i - \mathbf{c}, \quad i = 1, 2, \ldots, k.$$

Centering the data in this way guarantees that the covariance matrix, which follows, accurately captures the true spread and orientation of the neighborhood.

Covariance Matrix Construction:
The covariance matrix is a critical component in PCA as it encapsulates the variance and covariance among the data dimensions. It is computed as

$$\mathbf{C} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{r}_i \, \mathbf{r}_i^T.$$

This 3×3 symmetric matrix describes how the points in the neighborhood are dispersed along the x, y, and z axes. A well-constructed covariance matrix reflects the underlying geometric structure of the surface.

Eigenvalue Decomposition:
Next, we perform eigenvalue decomposition on the covariance matrix:

$$\mathbf{C}\,\mathbf{v}_i = \lambda_i\,\mathbf{v}_i, \quad i = 1, 2, 3.$$

Here, λ_1, λ_2, and λ_3 are the eigenvalues, and $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ are the corresponding eigenvectors. We order the eigenvalues such that

$$\lambda_1 \leq \lambda_2 \leq \lambda_3.$$

In this framework, the eigenvector corresponding to the smallest eigenvalue, $\mathbf{v}_1$, points in the direction where the data exhibits the least variation. This direction is orthogonal to the plane that best fits the neighborhood, since the plane captures the directions with the greatest variance (i.e., along $\mathbf{v}_2$ and $\mathbf{v}_3$).

Normal Vector Extraction:
The final step is to extract the normal vector from the eigenvalue decomposition. The normal vector $\mathbf{n}$ of the best-fit plane is given by:

$$\mathbf{n} = \mathbf{v}_1.$$

If a particular orientation is required (for instance, ensuring the normal points outward from the surface), the sign of $\mathbf{n}$ can be adjusted using a reference direction. Fig. 3-3 provides a visual representation where the third eigenvector (corresponding to the smallest eigenvalue) represents the normal to the surface.

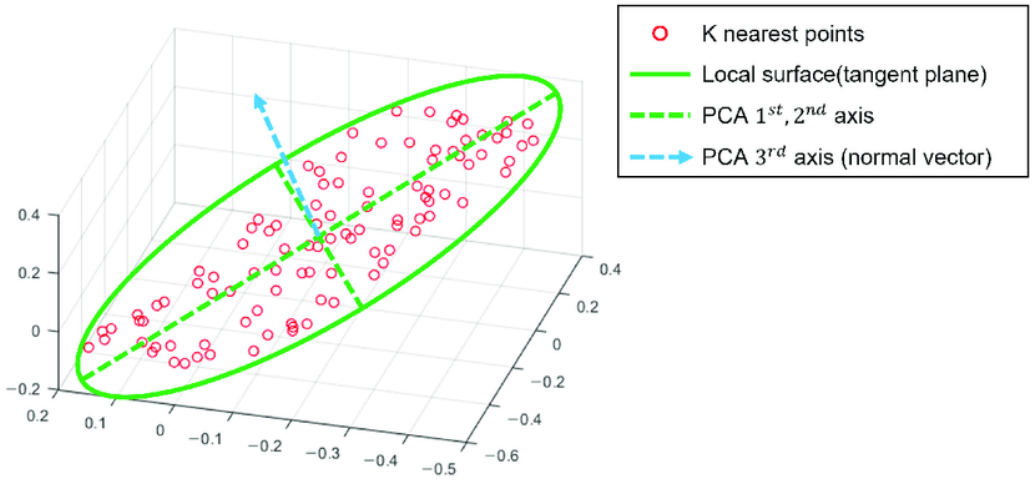


Figure 3-3: The third eigenvector represents the normal vector of the surface [13]

In summary, by following these systematic steps—starting from data centering, through covariance matrix computation and eigenvalue analysis—we obtain a robust estimate of the local surface normal. This method not only ensures an accurate representation of the surface orientation but also facilitates further processing tasks in various practical applications.

3-3 Inverse Kinematics

Inverse kinematics (IK) is a fundamental problem in robotics and computer graphics, involving the determination of joint parameters required to achieve a desired end-effector pose. In contrast to forward kinematics—where the end-effector's position and orientation are computed from known joint values—IK addresses the reverse problem.

Problem Definition: Given a desired end-effector transformation $\mathbf{T}_d$, the task is to compute the set of joint variables

$$\mathbf{q} = [q_1, q_2, \ldots, q_n],$$

such that

$$f(\mathbf{q}) = \mathbf{T}_d,$$

where f denotes the forward kinematics function mapping the joint space to the end-effector's pose. Typically, the transformation $\mathbf{T}_d$ is represented as a homogeneous transformation matrix that encodes both position and orientation. Figure 3-4 illustrates the process of calculating joint angles from the end-effector pose.

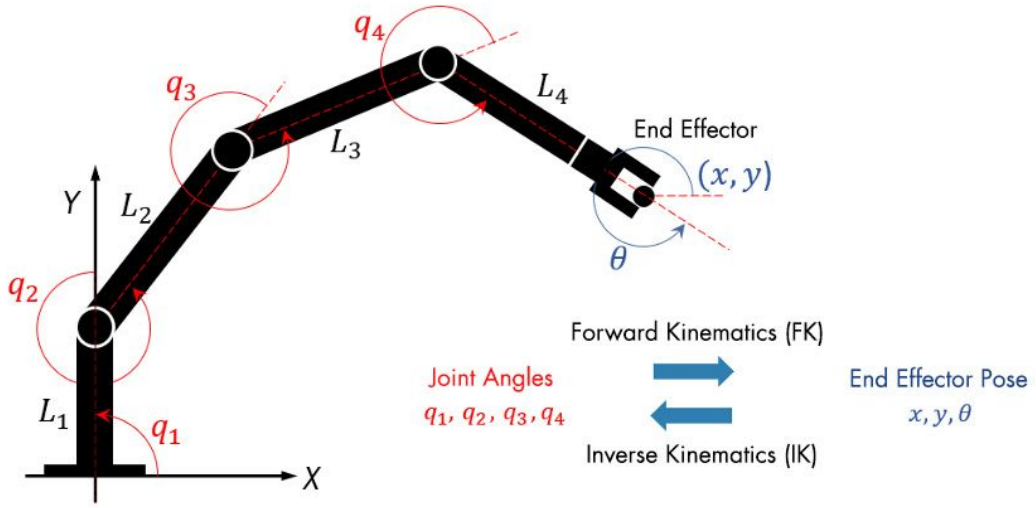


Figure 3-4: Inverse Kinematics: calculation of joint angles from end effector pose [21]

Challenges: The IK problem is complicated by several factors. First, the non-linearity of the forward kinematics equations makes it difficult to derive closed-form solutions. Second, redundant manipulators—with more degrees of freedom than strictly necessary—may have multiple or even infinite joint configurations yielding the same end-effector pose. Third, singularities, where the manipulator loses one or more degrees of freedom, can lead to numerical instability and control difficulties. Finally, physical constraints such as joint limits and collision avoidance further restrict the feasible solution space.

Solution Methods: A range of methods has been developed to tackle the IK problem. For simpler robotic systems, analytical methods can sometimes provide closed-form solutions that yield explicit formulas for each joint variable; these methods are computationally efficient but are generally applicable only to systems with few degrees of freedom and specific geometries

[11]. For more complex systems, iterative numerical techniques are often used. One common approach involves Jacobian inverse or pseudoinverse methods, which linearize the kinematics around the current configuration and iteratively update the joint variables using the inverse (or pseudoinverse) of the Jacobian matrix [9]. Alternatively, optimization-based methods reformulate the IK problem as an optimization task that minimizes the error between the current and desired end-effector poses, while simultaneously handling constraints such as joint limits [32].

Applications and Importance: IK is essential in both robotics and computer graphics. In robotics, it enables tasks such as reaching, grasping, and manipulation by converting high-level motion commands into precise joint-level movements. In computer graphics and animation, IK is crucial for generating natural, realistic movements in animated characters, ensuring that limb motions comply with physical constraints.

In summary, inverse kinematics is a pivotal technique for converting desired end-effector positions and orientations into feasible joint configurations. Its analysis involves addressing issues of non-linearity, redundancy, singularities, and constraints through both analytical and numerical methods, making it indispensable for advanced robotic control, motion planning, and realistic animation.

Chapter 4

Normal vector extraction

4-1 Motivation

Accurate extraction of surface normal vectors is crucial in robotics and automotive applications, as these normals often determine the orientation of end effectors or sensors. They play a vital role in tasks such as manipulation, obstacle avoidance, and scene understanding. However, traditional methods—particularly principal component analysis (PCA)—face several challenges:

- **Dependence on Point Density:** PCA requires a sufficiently large number of local points for reliable normal estimation. In environments with sparse or non-planar (e.g., highly curved) surfaces, its performance can degrade.
- **Computational Constraints:** Many real-world applications demand rapid computation. When processing limited point cloud data, PCA may not achieve the necessary speed or stability.

These issues motivate the development of an improved approach that can extract normal vectors accurately and efficiently, even in challenging, low-density scenarios.

4-2 Method

This study introduces a systematic processing pipeline designed to extract and analyze local surface characteristics from 3D point cloud data. Building upon a previously established PCA-based normal estimation method, the pipeline is enhanced through the sequential application of two filtering techniques—Gaussian smoothing and voxel grid filtering. These filtering steps are essential for reducing noise and computational complexity, thereby ensuring that the subsequent analysis accurately reflects the true geometric structure of the scene.

The overall pipeline is organized into five principal stages:

1. **Target Region Selection:** Identify and extract the local region surrounding a target point.
2. **Gaussian Smoothing:** Apply a Gaussian filter to smooth the data and suppress noise.
3. **Voxel Grid Filtering:** Downsample the point cloud by aggregating points within voxels while preserving essential structural features.
4. **PCA-based Normal Estimation:** Compute local surface normals via PCA to characterize the orientation of the underlying surfaces.
5. **Evaluation:** Assess the accuracy of the estimated normals by measuring the average angular difference between the computed normals and reference normals.

In the sections that follow, each stage of the pipeline is detailed, starting with the selection of the target region.

1. Target Region Selection

Objective: To isolate the region of interest around a specific target point, thereby reducing the computational load and enabling a more focused analysis of the local structure.

Description:

- A specific target point is identified within the overall point cloud.
- A fixed-radius window (or neighborhood) is established centered on this target point.
- Only the points falling within this predefined window are selected for further processing such as smoothing and data down sampling.

Selecting a sufficiently small window that still contains an adequate number of points is crucial in many point cloud processing tasks. The key reason is that the distribution of the point cloud near a target can be easily skewed by sensor or measurement noise, especially when defining which points fall within a certain window radius.

For example, consider a point that is physically located 0.03 m away from the target point. Due to noise in the measurement, this point might falsely appear at 0.005 m away from the target, causing it to be incorrectly included in a window of radius 0.005 m. Conversely, a point that is actually very close to the target might appear farther away due to noise and get excluded.

When such misplacements happen, the local neighborhood around the target point may no longer represent the true local geometry. This ultimately can degrade the quality of subsequent calculations, such as PCA-based normal estimation. As a result:

- A smaller window is desirable to maintain locality (i.e., it naturally restricts the neighborhood to points that are truly near the target in the underlying geometry).
- However, the window should not be so small that too few points remain, as this may cause further processing and PCA to become unstable or overly sensitive to random noise.

In short, the window size should be kept as small as possible while ensuring there are still enough points to reliably estimate local properties. Balancing these two aspects, noise robustness and local sampling density, helps achieve more accurate results when selecting nearest neighbors and performing local PCA.

2. Smoothing via Gaussian Filtering

Objective: Reduce noise in the point cloud data to achieve a smoother and more accurate representation of local point positions.

Description:

The Gaussian filter is a linear operator that smooths data by performing a weighted average of points, where the weights are determined by the Gaussian function. This function assigns greater significance to points near the center and diminishes the influence of those farther away. Widely used in image processing and point cloud smoothing, the Gaussian filter effectively suppresses high-frequency noise. Its suitability is particularly pronounced in applications involving depth cameras, as the noise typically follows a Gaussian (bell-curve) distribution, making it more effective than alternative filtering methods.

Mathematical approach:

The Gaussian function, or Gaussian kernel, is central to many smoothing and noise reduction techniques. It is defined as

$$G(d) = \exp\left(-\frac{d^2}{2\sigma^2}\right),$$

where d represents the distance between a given point and one of its neighbors, and σ is the standard deviation that determines the width of the Gaussian bell curve.

For a data point $\mathbf{p}$ with a set of neighboring points $\{\mathbf{p}_i\}_{i=1}^{k}$, each neighbor is assigned a weight based on its distance from $\mathbf{p}$. The weight for the i-th neighbor is computed by

$$w_i = \exp\left(-\frac{\|\mathbf{p} - \mathbf{p}_i\|^2}{2\sigma^2}\right).$$

To ensure that the influence of all neighbors is properly balanced, these weights are normalized so that their sum equals 1. This normalization is achieved through

$$\tilde{w}_i = \frac{w_i}{\sum_{j=1}^{k} w_j} = \frac{\exp\left(-\frac{\|\mathbf{p}-\mathbf{p}_i\|^2}{2\sigma^2}\right)}{\sum_{j=1}^{k} \exp\left(-\frac{\|\mathbf{p}-\mathbf{p}_j\|^2}{2\sigma^2}\right)}.$$

With the normalized weights $\tilde{w}_i$ computed, the next step is to calculate the new, filtered position $\mathbf{p}'$ as a weighted sum of the neighboring points:

$$\mathbf{p}' = \sum_{i=1}^{k} \tilde{w}_i \, \mathbf{p}_i.$$

This formulation inherently emphasizes the contributions of points that are closer to $\mathbf{p}$, leading to effective noise reduction and a smoother representation of the data.

This flow—from distance measurement to weight computation, normalization, and averaging—ensures that the smoothing process accurately reflects the local structure of the data.

3. Data Simplification via Voxel Grid Filtering

Objective: To reduce the number of points in the region while preserving the overall geometric structure, thus lowering the computational complexity for subsequent analyses.

Description:

The voxel grid filter is a down-sampling technique designed to reduce the number of points in a point cloud while preserving its overall structure. It works by dividing the three-dimensional space into a uniform grid of cubic cells, known as voxels (see Figure.4-1(b)). Within each voxel, a representative point is computed—typically the centroid of all the points contained in that voxel (as illustrated in Figure.4-1(c)).

The collection of these centroids forms the down-sampled point cloud, which retains the essential geometric features of the original scene. This method is widely used in 3D reconstruction, robotics, and computer vision, as it reduces computational load and processing time, thereby facilitating applications such as multiple target detection.

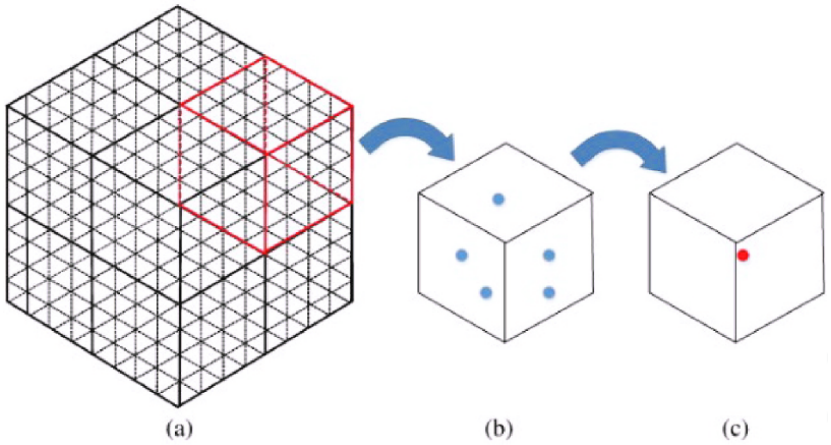


Figure 4-1: Process of voxel grid filtering [7]

Mathematical approach:

The voxel grid filtering process begins with the voxelization of the point cloud. Consider a point cloud

$$\{\mathbf{p}_i\}_{i=1}^{N}, \quad \mathbf{p}_i \in \mathbb{R}^3,$$

and let l be the user-defined voxel size representing the edge length of each voxel. Each point $\mathbf{p}_i$ is assigned to a voxel by computing its voxel index as

$$\mathbf{v}_i = \left\lfloor \frac{\mathbf{p}_i}{l} \right\rfloor \in \mathbb{Z}^3,$$

where the floor function $\lfloor \cdot \rfloor$ is applied element-wise.

After voxelization, the next step is to compute a representative point for each voxel. For each unique voxel index $\mathbf{v}$, define the set of points within that voxel as

$$V_{\mathbf{v}} = \{\mathbf{p}_i \mid \left\lfloor \frac{\mathbf{p}_i}{l} \right\rfloor = \mathbf{v}\}.$$

The centroid of these points, which serves as the representative point for voxel $\mathbf{v}$, is given by

$$\mathbf{p}'_{\mathbf{v}} = \frac{1}{|V_{\mathbf{v}}|} \sum_{\mathbf{p}_i \in V_{\mathbf{v}}} \mathbf{p}_i,$$

with $|V_{\mathbf{v}}|$ denoting the number of points in the voxel.

The final down-sampled point cloud is composed of the centroids of all voxels that contain at least one point:

$$\{\mathbf{p}'_{\mathbf{v}} \mid V_{\mathbf{v}} \neq \emptyset\}.$$

This collection of centroids preserves the overall spatial structure of the original point cloud while significantly reducing its density.

4. PCA-based Normal vector Extraction

The details of the PCA-based normal vector extraction process are explained in Section 3-2. While that section covers the traditional PCA-based normal extraction in depth, this section (Section 4-2) focuses primarily on the filtering method, with a brief introduction to the PCA-based normal vector extraction.

Objective: To determine the local surface orientation by estimating normal vectors in the downsampled region.

Description:

- For each point in the simplified point cloud, nearby neighbors are identified.
- The spatial distribution of these neighbors is analyzed by computing their covariance matrix.
- Principal Component Analysis (PCA) is applied to determine the eigenvectors and eigenvalues of the covariance matrix.
- The eigenvector corresponding to the smallest eigenvalue is chosen as the local normal vector, representing the direction with the least variance.

5. Evaluation

Objective: Define the reference normal vector accurately and establish a clear evaluation metric.

Description:

- In Fig.4-2, the blue dots represent the point cloud on the sphere's surface.
- For local processing, only the red points within the sphere window are considered, with the center of the window designated as the target point.
- The reference normal vector is defined as the vector passing through both the sphere's center and the target point.

- The accuracy of the PCA-estimated normal vector is evaluated by measuring its angular difference from the reference normal vector.
- An angular deviation below a specified threshold (e.g., 1°) is considered acceptable. The primary objective is to meet this threshold, with any further reductions evaluated for potential accuracy enhancements.

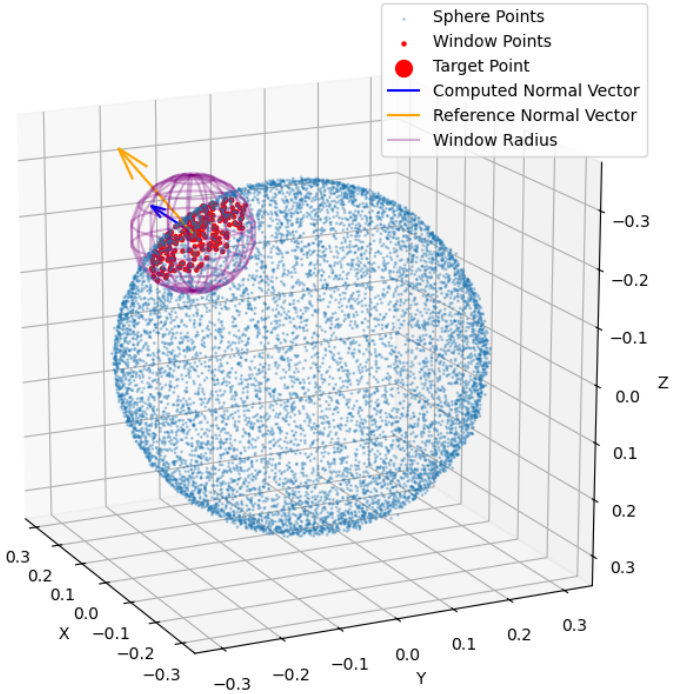


Figure 4-2: Evaluation model : Sphere with noisy point cloud

4-3 Simulation

4-3-1 Description for simulation

The distance between the target point and the camera is fixed at 0.2 m. Two cases are simulated to represent low- and high-curvature surfaces. The low-curvature case, with a curvature of 0.1 m^{-1}, which effectively approximates a flat surface (i.e., curvature = 0) while avoiding potential computational singularities and accounting for the minor imperfections that are typically present in real-world "flat" surfaces. Simulating a high-curvature scenario with a curvature of 100 m^{-1} is necessary because the test components vary—for example,

compressor components and tire wheels. To enhance the adaptability and accessibility for various shapes of the test component, the target point on the curved surface must also be simulated.

To reflect real-world data, the number of point clouds within a circle of radius 0.005 m was estimated using a RealSense D435 depth camera and used as the K-nearest neighborhood (KNN). The results indicated that approximately 800 points were present within the circle when the distance between the target point and the camera was 20 cm. Here, the circle of radius 0.005 m serves as the circular window in the target region selection step, with the target point defined as the centroid of this window. Assuming a uniform distribution, the total number of points on the sphere's surface was then calculated.

To ensure the reliability of the normal estimation process, we performed 100 iterations for each nearest neighbor parameter and analyzed the mean and standard deviation of the angular differences between the reference and estimated normal vectors for both low- and high-curvature surfaces. An angular difference of less than 1° was deemed acceptable, as this threshold accounts for calibration accuracy, sensor precision, and environmental disturbances.

This choice is further supported by repeatability studies. Previous research shows that while human operators targeting five points can incur an error of approximately 4 cm, industrial robotic systems achieve positional repeatability errors of around 0.3 mm [33] and orientation errors close to 0.83° (0.014 rad) [18]. This significant difference in precision underscores that even small directional errors exceeding 1° may accumulate and adversely affect robotic system performance, thereby justifying the strict 1° threshold.

By adhering to this threshold, we ensure that the angular error remains within the high-precision bounds of robotic systems. This minimizes the propagation of angular errors in subsequent measurements and manipulations, ultimately enhancing the overall reliability and accuracy of the system in practical applications.

To reflect real-world conditions in the simulation, a noise model—based on the characteristics of a real depth camera—is incorporated into the spherical point cloud. The simplified noise model is designed to mimic the radial distribution of measurement noise encountered in depth sensors. The process is as follows:

Gaussian noise model

The noise model for the Intel RealSense D435 camera has been previously established, showing that its noise characteristics follow a Gaussian distribution. This simplifies our approach compared to dealing with random noise. In the evaluation model, Gaussian noise is applied to each point in the point cloud that lies on the surface of the sphere. The procedure for adding noise to the sphere is described as follows:

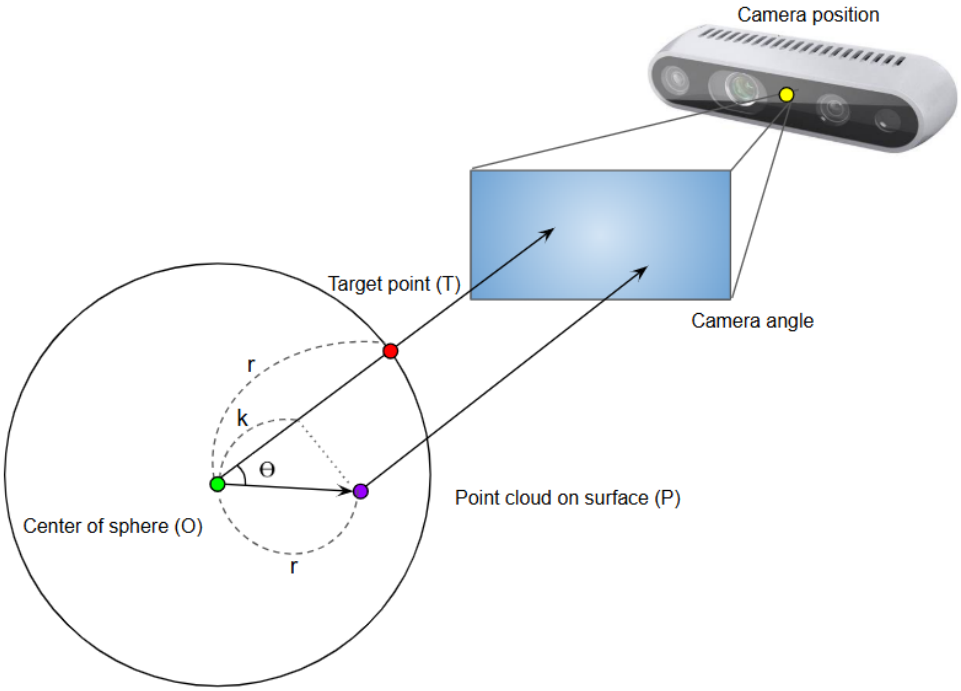


Figure 4-3: Noise model

The following equations describe the relationship between vectors and the calculation of the corrected z coordinate based on vector projections. :

- **Vector Definition:** Let r be the vector from the origin point O to the target point T, which is equivalent to the vector $\vec{OP}$:

$$r = |\vec{OT}| = |\vec{OP}|$$

- **Angle Calculation:** The angle θ between the vectors $\vec{OP}$ and $\vec{OT}$ can be calculated using the dot product:

$$\theta = \arccos\left(\frac{\vec{OP} \cdot \vec{OT}}{|\vec{OP}| \cdot |\vec{OT}|}\right)$$

- **Projection Calculation:** The scalar value k represents the projection of the vector r onto the direction of $\vec{OT}$:

$$k = r \cdot \cos(\theta)$$

- **Corrected Distance:**
  The corrected distance between the camera and each point in the point cloud is given by:

$$z_{\mathrm{p}} = r + d - k,$$

  where d is the distance from the camera to the target point.

- **Radial distribution of noise:**

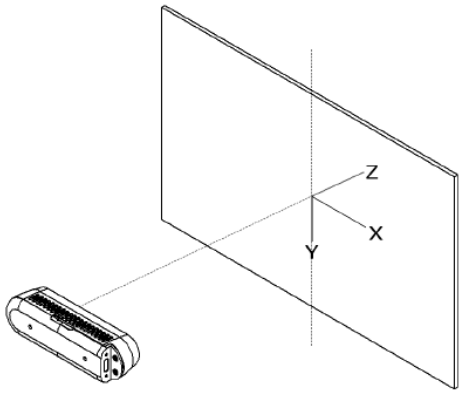


Figure 4-4: The flat white target is positioned a known z distance rotated θ_y degrees. The coordinates follow the common camera convention.[3]

The noise standard deviation model proposed in [3] is given by:

$$\sigma_z(z, \theta_y) = 0.001063 + 0.0007278\, z + 0.003949\, z^2 + 0.022\, z^{\frac{3}{2}} \frac{\theta_y}{\left(\frac{\pi}{2} - \theta_y\right)^2},$$

where z is the distance from the camera to the target point and θ_y is the rotation angle around the y-axis as shown as Fig.4-4. Since the noise distribution is found to follow a Gaussian shape, we apply Gaussian smoothing in subsequent steps.

Assuming the reference normal vector is perfectly perpendicular to the camera angle ($\theta_y = 0$), the formula simplifies to:

$$\sigma_z(z) = 0.001063 + 0.0007278\, z + 0.003949\, z^2.$$

Using the corrected distance z_{point}, standard deviation from the noise model can then be applied to each point on the surface to reflect realistic, depth-dependent measurement noise.

- **Noise application:**

Let $\mathbf{p} \in \mathbb{R}^3$ be a point on the ideal, noise-free spherical point cloud, and let $z_p = \|\mathbf{p}\|$ denote its depth (distance from the sensor). An effective noise standard deviation $\sigma(z_p)$ is computed as a function of d, typically in the form

$$\sigma(z_p) = \alpha + \beta\, z_p + \gamma\, z_p^2,$$

where α, β, and γ are parameters determined from calibration with the real depth camera.

The noise is applied radially. For each point $\mathbf{p}$, a noise vector $\boldsymbol{\eta}$ is sampled from a multivariate Gaussian distribution,

$$\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma(z_p)^2\, \mathbf{I}),$$

where $\mathbf{I}$ is the 3×3 identity matrix. The noisy point $\mathbf{p}_{\text{noisy}}$ is then given by

$$\mathbf{p}_{\text{noisy}} = \mathbf{p} + \boldsymbol{\eta}.$$

This formulation ensures that the perturbation is applied in all directions, with a magnitude consistent with the depth-dependent noise model, effectively simulating the noise characteristics of a real depth sensor.

Simulation Setup: Gaussian Filtering and Point Cloud Simplification

The simulation setup is meticulously designed to explore a comprehensive range of parameters to identify the optimal combination that minimizes the angle error while maintaining computational efficiency.

Gaussian Filtering Parameters

Gaussian filtering is employed to smooth the point cloud data, mitigating the effects of noise and outliers. The degree of smoothing is controlled by the standard deviation parameter, σ. A broader range of σ values allows us to assess the trade-off between noise reduction and the preservation of geometric details.

$$\sigma = [0.05, 0.1, 0.2, 0.3, 0.5, 1, 3, 5, 10, 50, 100]$$

These σ values were selected based on previous research [14], which demonstrated their effectiveness in representing a wide spectrum of filtering intensities. Smaller σ values correspond to subtle smoothing, preserving intricate features, whereas larger σ values induce more aggressive smoothing, potentially obscuring fine geometric structures.

Point Cloud Simplification

Point cloud simplification is achieved using voxel grid filtering, which reduces the number of points by computing the centroid (or average) of all points within each voxel. The simplification rate (SR) is defined as the fraction of the original point cloud retained after filtering. For clarity, an SR of 0.3 (equivalently, 30%) means that 30% of the original points remain post-simplification. The following SR values are used to evaluate the impact of simplification:

$$\text{SR} = [10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%]$$

These values are selected to systematically study the trade-off between data reduction and the retention of critical geometric information, affecting both the accuracy of normal estimation and the computational performance.

The relationship between the simplification rate and the voxel size is defined as:

$$\text{voxel\_size} = \frac{\text{base\_voxel\_size}}{\text{SR}}, \tag{4-1}$$

where `base_voxel_size` is a predefined constant that sets the overall scale of the voxel grid. In this formulation, a lower SR (i.e., fewer points retained) leads to a larger voxel size, resulting in more aggressive downsampling. The value of `base_voxel_size` is adjusted based on the observed characteristics of the filtered point cloud to ensure consistent performance across different simplification levels.

Optimization Objective

The primary objective is to identify the optimal combination of σ and SR that achieves a predefined **threshold angle** with the minimal number of neighborhood points, denoted by k. The threshold angle represents the maximum allowable deviation between the estimated normal vector and the ground truth. Achieving this threshold with the smallest k enhances computational efficiency without compromising accuracy.

Optimization Procedure

The optimization process involves a systematic exploration of the parameter space defined by the combinations of σ and SR. The procedure is outlined as follows:

1. **Parameter Combination Evaluation**: For each combination of σ and SR, we assess whether the resulting filtered and simplified point cloud satisfies the threshold angle criterion. This assessment begins with the smallest feasible value of k to ensure computational efficiency.
2. **Recording Satisfactory Combinations**: If a particular combination meets the threshold angle at a specific k value, this combination is recorded. The focus is on capturing the earliest occurrence where the threshold is satisfied, corresponding to the minimal k required.
3. **Selection of Optimal Combination**: Among all recorded combinations, the one associated with the smallest k value is selected as the optimal parameter set. This selection ensures that the normal estimation achieves the desired accuracy with the least computational overhead.

This methodical approach guarantees that the optimal parameters not only meet the accuracy requirements but also enhance the efficiency of the normal estimation process by minimizing the neighborhood size.

Implementation Details

The simulation is conducted using a synthetic point cloud generated to mimic real-world scenarios. Gaussian noise is introduced to simulate measurement inaccuracies. The combination of Gaussian filtering and voxel grid simplification is applied iteratively across all parameter combinations. For each iteration, the normal vectors are estimated, and the angle errors are computed to evaluate compliance with the threshold angle.

The optimization leverages parallel processing to expedite the evaluation of numerous parameter combinations. This computational strategy ensures that the extensive search across the parameter space is performed efficiently, enabling the identification of the optimal combination within a reasonable timeframe.

Evaluation Metrics

The performance of each parameter combination is quantified using the following metrics:

- **Mean Angle Error**: The average deviation between the estimated normal vectors and the ground truth across all k number of neighbors.
- **Standard Deviation of Angle Error**: Measures the variability in the angle errors, providing insights into the consistency of the normal estimation. Standard deviation is provided in figures such as Figure.4-5.
- **Computation Time**: The time required to perform normal estimation for a given parameter set, reflecting the computational efficiency.

These metrics collectively inform the selection of the optimal parameter combination, ensuring that both accuracy and efficiency are adequately addressed.

4-3-2 Low curvature surface case

The target points are often placed on a flat surface of the test component. To simulate this scenario, a case with very low curvature is analyzed to represent flat surface conditions. The simulation setup is summarized in the following table:

Parameter	**Value**
Curvature (m^{-1})	0.1
Radius of Sphere (m)	10.0
Radius of Window Sphere (m)	0.005
Number of Points in the Window	800

Table 4-1: Sphere Parameters for low curvature surface case

Mean Angle Error

Figure 4-5 shows that classic Principal Component Analysis (PCA) cannot achieve an average angle error below 1° when analyzing a highly localized region (with k ranging from 3 to 100) from 800 available point clouds. To bring the error within an acceptable range, classic PCA must consider almost the entire point cloud within a 0.005-meter radius, which corresponds to a much larger k value of 722 (see Figure 4-6).

In contrast, the proposed method maintains an acceptable average angle error using only a very localized region. For instance, at $k = 25$, the method achieves an average angle error of

0.998° (Figure 4-5). Increasing the number of neighbors to $k = 96$ further reduces the error to 0.832° within 100 point clouds, representing the minimum average angle difference observed across various combinations of sigma and simplification rate, as summarized in Table 4-2. It shows that it can possibly reduce the angle error further. These results demonstrate that the proposed method can achieve superior accuracy with minimal local information compared to classic PCA.

Computation Time

The proposed method also offers significant improvements in computation time. As noted, classic PCA requires a large neighborhood size ($k = 722$) to achieve an average angle error below 1°, resulting in a computation time of 0.2170 seconds.

In contrast, the proposed method reaches similar accuracy with far fewer neighbors. At $k = 25$, it reduces the average angle error to approximately 0.998° in only 0.0025 seconds. These findings underscore the superior computational efficiency of the proposed method, which attains the desired threshold angle with fewer points and significantly reduced processing times compared to classic PCA.

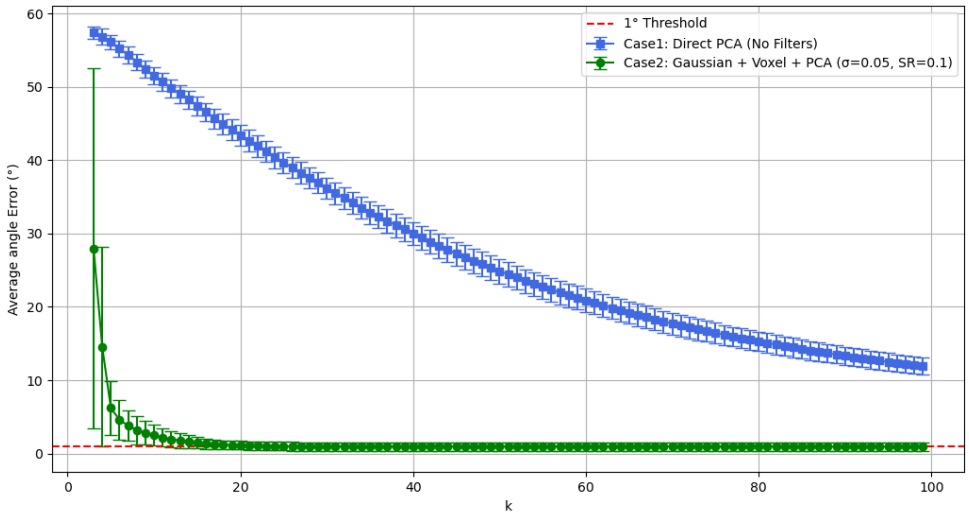


Figure 4-5: Comparison of two approaches : Mean and standard deviation of angle error for 100 times simulation

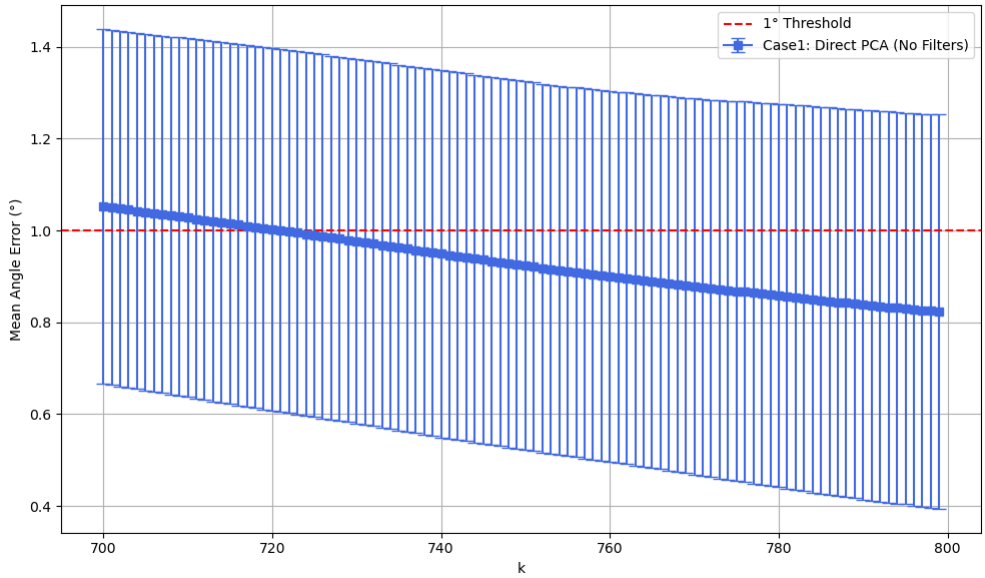


Figure 4-6: Classic PCA using entire point clouds in the window

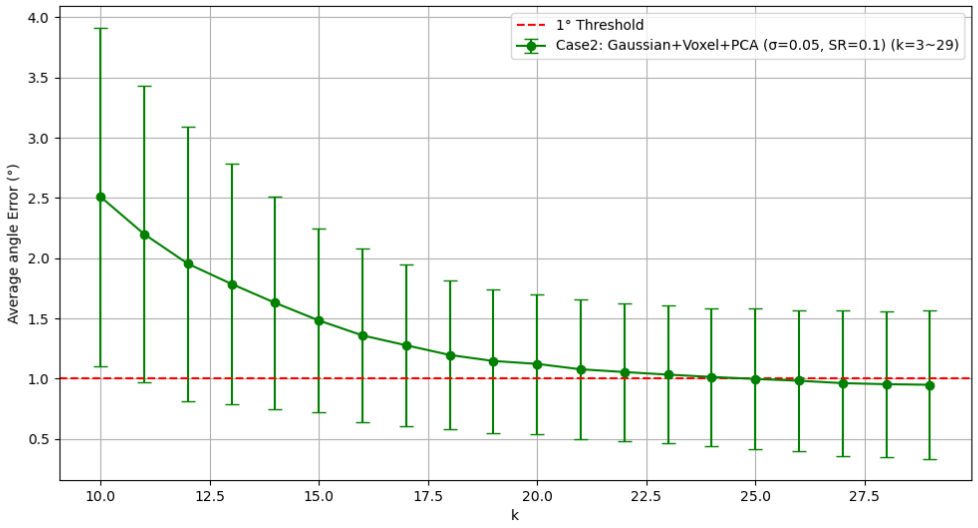


Figure 4-7: Proposed method with threshold angle error

Parameter	Threshold angle case	Minimum angle case
k (Number of Points used for PCA)	25	96
Sigma	0.05	0.2
Simplification Rate	0.1	0.9
Average Angle Error (°)	0.998	0.832
Average Computation Time (s)	0.002570	0.012373

Table 4-2: Optimal combination of parameters for threshold angle and minimum angle in simulation up to 100 point clouds

4-3-3 High curvature surface case

The high-curvature simulation addresses scenarios where the target point is situated on rounded surface regions of the test components. Various test components include such rounded shapes, and the target points are often located on these curved sections. In this simulation, the curvature of the surface is set to 100 m^{-1}, representing a significant deviation from a planar or low-curvature condition. To ensure a fair comparison with the low-curvature case, the same number of point cloud data points are considered around the target point.

Parameter	Value
Curvature (m^{-1})	100
Radius of Sphere (m)	0.01
Radius of Window Sphere (m)	0.005
Number of Points in the Window	800

Table 4-3: Sphere Parameters for high curvature surface case

Mean Angle Error

Figure 4-9 demonstrates that the classic Principal Component Analysis (PCA) method fails to achieve the acceptable average angle error threshold of below 1° in scenarios with increased curvature. This limitation arises because higher curvature causes the point clouds to become more dispersed, making them less representative of the local surface geometry. Consequently, classic PCA struggles to accurately capture the underlying surface orientation in such high-curvature environments.

In contrast, the proposed method successfully meets the threshold, achieving a mean angle difference of 0.968° at $k = 55$. This performance is attained with a sigma of 0.05 and by retaining only 40% of the original point cloud through simplification using a voxel grid filter. Furthermore, across the tested range of k values (up to 100), the proposed method consistently maintains low angle errors, with the minimum angle difference recorded at 0.889° occurring at $k = 62$. These results underscore the proposed method's ability to effectively handle high-curvature cases by leveraging a more representative subset of the point cloud, thereby enhancing accuracy without necessitating extensive computational resources.

Computation Time

In terms of computation time, the classic PCA method's inability to meet the threshold angle error of below 1° renders its computational performance less relevant for practical applications, where accuracy is paramount. However, for verification purposes, it is noteworthy that classic PCA achieves its minimum angle error of 1.1824° at $k = 799$, with a corresponding computation time of 0.1619 seconds. This significant k value highlights classic PCA's inefficiency in high-curvature scenarios, as it requires a large neighborhood to approach acceptable accuracy, leading to prolonged computation times.

Conversely, the proposed method exhibits remarkable computational efficiency alongside superior accuracy. It achieves the threshold angle error of below 1° with a substantially smaller neighborhood size of $k = 55$, incurring a computation time of approximately 0.007 seconds. Moreover, when targeting the minimum angle error of 0.889° at $k = 62$, the computation time remains exceptionally low at 0.007 seconds as summarized in Table.4-4. These findings illustrate that the proposed method not only surpasses classic PCA in accuracy under high-curvature conditions but also does so with significantly reduced computational overhead, making it highly suitable for real-time and resource-constrained applications.

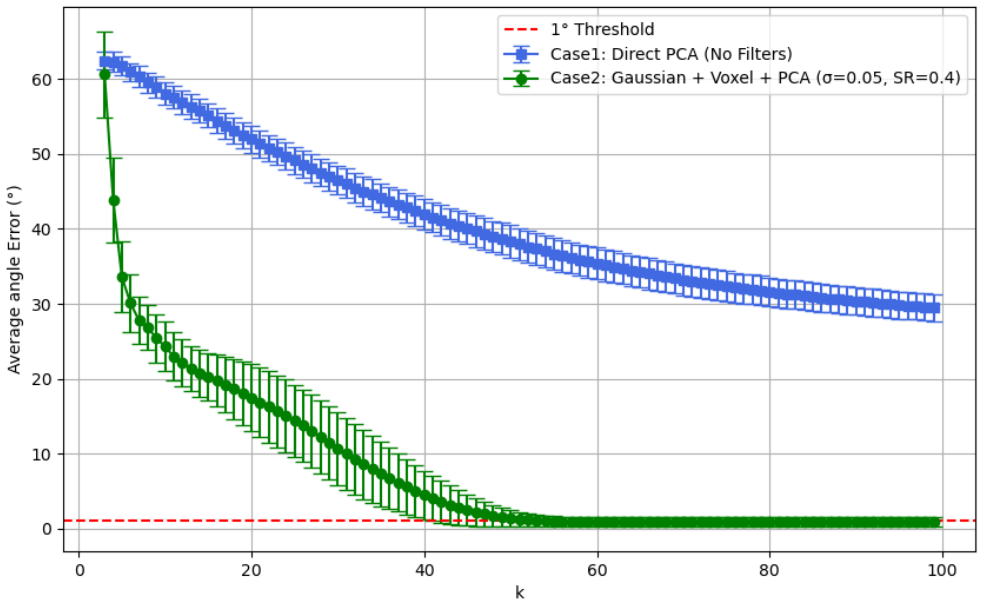


Figure 4-8: Comparison of two approaches : Mean and standard deviation of angle error for 100 times simulation

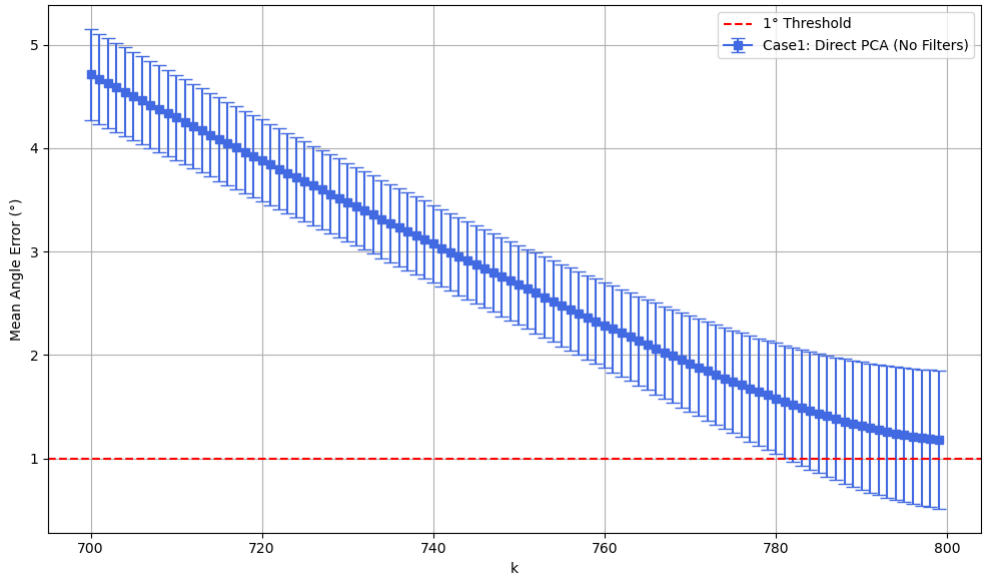


Figure 4-9: Classic PCA using entire point clouds in the window

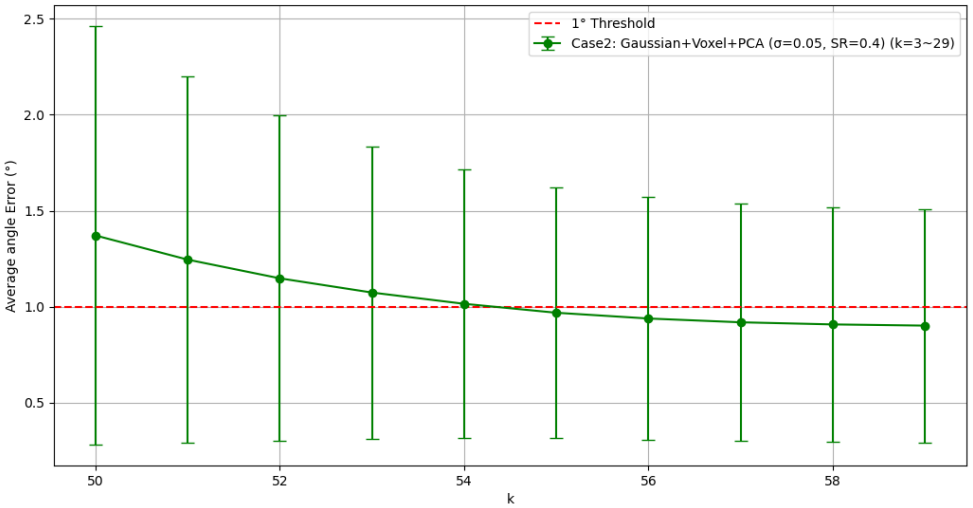


Figure 4-10: Proposed method with threshold angle error

Parameter	Threshold angle case	Minimum angle case
k (Number of Points used for PCA)	55	62
Sigma	0.05	0.1
Simplification Rate	0.4	0.4
Average Angle Error (°)	0.968	0.889
Average Computation Time (s)	0.007223	0.007065

Table 4-4: Optimal combination of parameters for threshold angle and minimum angle in simulation up to 100 point clouds

4-3-4 Importance of Filter Order

The proposed method consists of applying a Gaussian filter followed by a voxel grid filter, and finally executing Principal Component Analysis (PCA) in the same manner as in the classical approach. The order in which these filters are applied is critical to achieving an optimal combination of parameters that minimizes the angle error relative to a predefined threshold.

To assess the influence of filter order, we simulated an alternative case where the voxel grid filter is applied before the Gaussian filter. In the subsequent discussion, the proposed method (Gaussian filter $\rightarrow$ voxel grid filter) is referred to as **Case 2**, while the reversed order (voxel grid filter $\rightarrow$ Gaussian filter) is labeled as **Case 3**.

The simulations were conducted over a sequence of point cloud sets, with the analysis continuing until $k = 100$ point clouds. If the processing pipeline in any case fails to reach the threshold angle by $k = 100$, then the optimal parameter combination is selected based on the smallest achievable angle error within this range.

Figure 4-11 illustrates the results for a low curvature scenario, whereas Figure 4-12 shows the high curvature case. In both curvature scenarios, the reversed filter order (Case 3) did not achieve the threshold angle; the best results obtained were an angle error of approximately 1.678° in the low curvature case and 2.131° in the high curvature case. Although Case 3 exhibited a slightly lower angle error for the high curvature case over an initial sequence of point clouds (roughly from $k = 5$ to $k = 43$), this improvement is ultimately irrelevant given that the threshold angle was not reached.

This outcome aligns with our initial expectations. When noise, particularly Gaussian noise, is present around the target point, it is advantageous to first apply a Gaussian filter to effectively attenuate this noise before simplifying the point cloud using a voxel grid filter. The rationale behind this sequence is that the noise originating from the camera typically follows a Gaussian distribution. If the voxel grid filter is applied first, the noise may be unevenly distributed across voxels, potentially distorting the Gaussian profile and leading to the emergence of non-Gaussian noise characteristics.

In summary, our results underscore the necessity of applying the Gaussian filter prior to the voxel grid filter to maintain the integrity of the noise characteristics and to achieve the best possible performance in reducing the angle error.

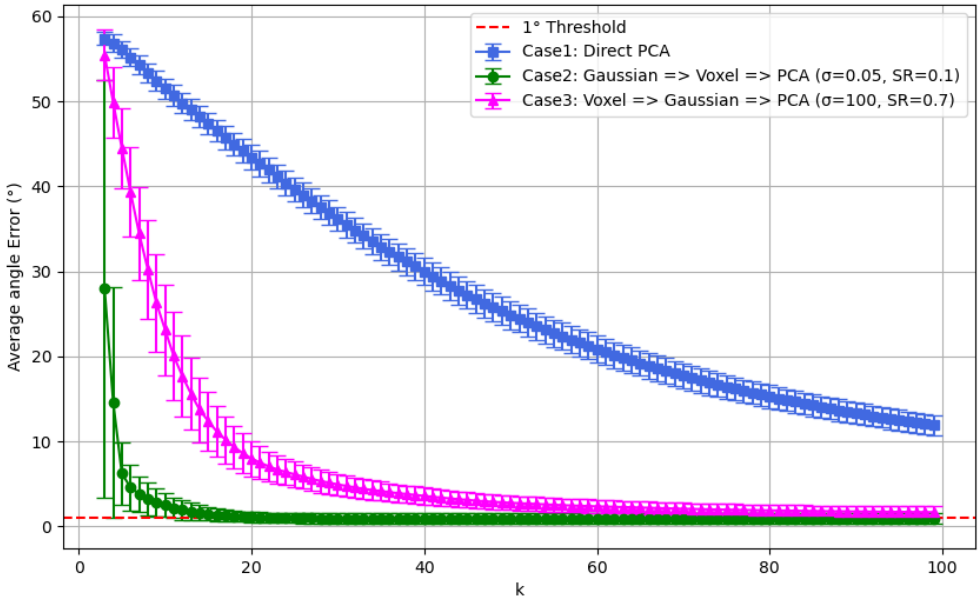


Figure 4-11: Comparison in low curvature case

Parameter	Case2 (Proposed method)	Case3
Threshold angle reached	Reached	Not reached
k (Number of Points used for PCA)	25	99
Sigma	0.05	100
Simplification Rate	0.1	0.7
Average Angle Error (°)	0.998	1.678
Average Computation Time (s)	0.002570	0.028319

Table 4-5: Optimal combination of parameters for threshold angle and minimum angle in simulation up to 100 point clouds

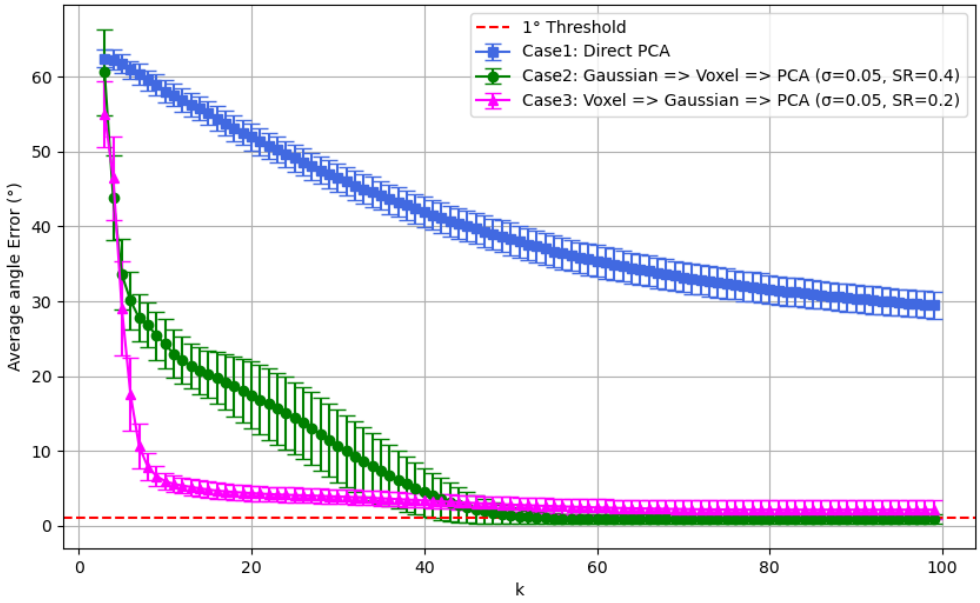


Figure 4-12: Comparison in high curvature case

Parameter	Case2 (Proposed method)	Case3
Threshold angle reached	Reached	Not reached
k (Number of Points used for PCA)	55	62
Sigma	0.05	0.05
Simplification Rate	0.4	0.2
Average Angle Error (°)	0.968	2.131
Average Computation Time (s)	0.007223	0.012768

Table 4-6: Optimal combination of parameters for threshold angle and minimum angle in simulation up to 100 point clouds

4-3-5 Evaluation of Simulation

In the simulation, two scenarios—low curvature and high curvature—are considered to evaluate the precision of normal vector estimation at a target point.

The classic PCA method requires a circular region with a radius of 0.005 m around the target point, even in high curvature cases, to approximate a locally flat surface. In practice, this necessitates the inclusion of approximately 800 points from the point cloud to reach the required threshold angle, thereby significantly increasing the computational time. Moreover, in the high curvature case, the classic PCA approach fails to achieve the threshold angle due to insufficient variation within the region.

In contrast, the proposed method—which combines a Gaussian filter, a voxel grid filter, and PCA—demonstrates successful performance in both high and low curvature scenarios. In high

curvature regions, the point cloud surrounding the target point is more affected by noise and deviation compared to that in low curvature regions. As anticipated, achieving the threshold angle in such cases requires a larger number of points; our results show that the low curvature case requires only 25 points, while the high curvature case requires 55 points. The lower point requirement in the low curvature case results in faster computation times.

Furthermore, the proposed method not only reaches the threshold angle but also offers the potential for higher accuracy in normal vector estimation for both curvature conditions. In simulations using up to 100 points, the normal vector estimation error improved from 0.998° to 0.832° in the low curvature case and from 0.968° to 0.889° in the high curvature case within 100 point clouds, while maintaining acceptable computational times compared to the classic PCA method.

This simulation is a crucial step in assessing the feasibility of the proposed method before its application in real-world scenarios. Since the simulation model—a sphere—is mathematically reliable and closely resembles the expected conditions in practical applications, the successful estimation of accurate and precise normal vectors across different curvature cases validates the proposed approach without the need for immediate physical testing.

4-4 Real-world experiment

For the real-world experiment, similarly to the simulation, the angle difference on a flat surface representing low curvature and on a curved surface representing high curvature will be compared with the already known angle difference using real test components.

Before estimating the angle difference of the normal vector at the target point, environment control and target point detection are essential processes because normal vector estimation is a post-process that relies on accurately detecting the target point under a controlled environment. To compare the methods under objective conditions, the same recording from the depth camera is used for both methods.

The following parts provide a brief explanation of the environment control and stable target detection process:

- **Environment control**
  Basic Realsense filters, such as spatial and temporal filtering functions, are used to stabilize the depth frame and generate sparse point clouds for the two methods: the classic PCA and the proposed method.

  In addition, for both methods, device parameters are adjusted for high and low curvature cases to improve depth detection and minimize disturbances, such as light reflections.

- **Target point detection**
  Accurate target point detection is critical for computing the normal vector and its corresponding angle difference. The following steps outline the target detection process:

  1. **HSV Thresholding:** The input RGB image is converted to the HSV color space, and dynamic thresholds for hue, saturation, and value are set using trackbars. Two HSV ranges are defined to detect red regions, accounting for the boundary spanning red hues.

2. **Mask Creation:** Binary masks are generated based on the HSV thresholds and combined. Additional constraints are applied using a region of interest (ROI) mask to focus on specific areas of the image and a depth mask to filter regions beyond a specified depth.
3. **Contour Detection:** Contours of the red regions are extracted, and the largest contours are selected based on predefined area constraints to remove noise. Centroids of these selected contours are calculated using image moments.
4. **Centroid Stabilization:** Centroid positions are smoothed across frames by maintaining a buffer of recent centroid locations and averaging them. This reduces jitter and improves the accuracy of subsequent computations.
5. **3D Point Cloud Extraction:** Depth information from the camera is used to generate a local 3D point cloud around the stabilized centroids. Camera intrinsics are applied to map pixel coordinates to 3D space.

This detection pipeline ensures robust and accurate identification of the target points, enabling reliable comparison of the classic PCA and proposed methods under consistent environmental conditions. Note that this pipeline does not affect the noise property of the depth camera.

Evaluation Metrics

In real-world experiments, the performance of both the proposed method and the classic PCA (without filters) is evaluated using the following metrics:

- k **(Number of Nearest Neighbors):** This parameter represents the number of nearest points used in the PCA computation for normal estimation. The choice of k is critical: a small k may not provide enough data for accurate estimation, while a large k may include points from adjacent surfaces, leading to degraded performance. Evaluations are conducted for different k values (e.g., $k = 10$, $k = 100$, and $k = 300$) to analyze the impact on estimation accuracy and stability.
- **Average Angle Difference:** This metric is defined as the mean absolute difference between the estimated normal vector and the corresponding reference normal vector across a series of frames. In the low curvature experiment, for example, the theoretical angle difference between the two surfaces is 45°. Deviations from this value quantify the estimation error; thus, a lower average angle difference indicates a more accurate normal estimation.
- **Stabilization Ratio:** This ratio represents the percentage of frames in which the difference between the two estimated normal vectors remains within an acceptable error range. In our experiments, each normal vector is required to have an error within 1°, meaning that the maximum acceptable deviation in the difference between two normals is 2°. The stabilization ratio is calculated as the fraction of frames (out of 1000 frames) where the estimated angle difference lies within $\pm 2°$ of the theoretical angle difference (for example, within 43° to 47° when the reference angle difference is 45°). A higher stabilization ratio indicates more consistent and reliable performance of the normal estimation method.

4-4-1 Low curvature surface case

In a controlled environment, a test component with target points on flat surfaces is used to represent the low curvature scenario as shown in Figure 4-13. Unlike pose estimation, where errors are minimized through object pose adjustments, the estimation of normal vectors is highly dependent on the camera angle, potentially leading to errors. To account for inaccuracies due to both camera placement and the component's position, the difference between two known normal vectors is used as a reference.

The angle between the two surfaces containing the target points is 135°, which implies that the expected difference between their normal vectors is 45°. In this experiment, the target points (labeled as centroid 1 and centroid 2 in Figure 4-13) are located at distances of 0.215 m and 0.230 m from the camera, respectively. As in the simulation, a circular window with a 5 mm radius is centered around each target point. The point cloud within each window is then processed either by applying a filter or by performing PCA directly when no filter is used.

In the proposed method, the parameters for the Gaussian filter and the voxel grid filter are set to $\sigma = 100$ and voxel lengths of 0.001 m and 0.0004 m, respectively. These values are chosen to highlight the pronounced effects of filtering. Specifically, increasing σ to 100 maximizes the smoothing effect, while adjusting the voxel length reduces the average number of points in the cloud from approximately 500 to 60 for a voxel length of 0.001 m, and from approximately 500 to 200 for a voxel length of 0.004 m, corresponding to reductions of 88% and 53%, respectively. These parameter values are deliberately selected to test the significant impact of both filters, although further research is needed to determine the optimal combination for real-world experiments. In Table 4-7, stabilization refers to the percentage of frames in which the estimated angle difference remains within 2° of the expected 45°.

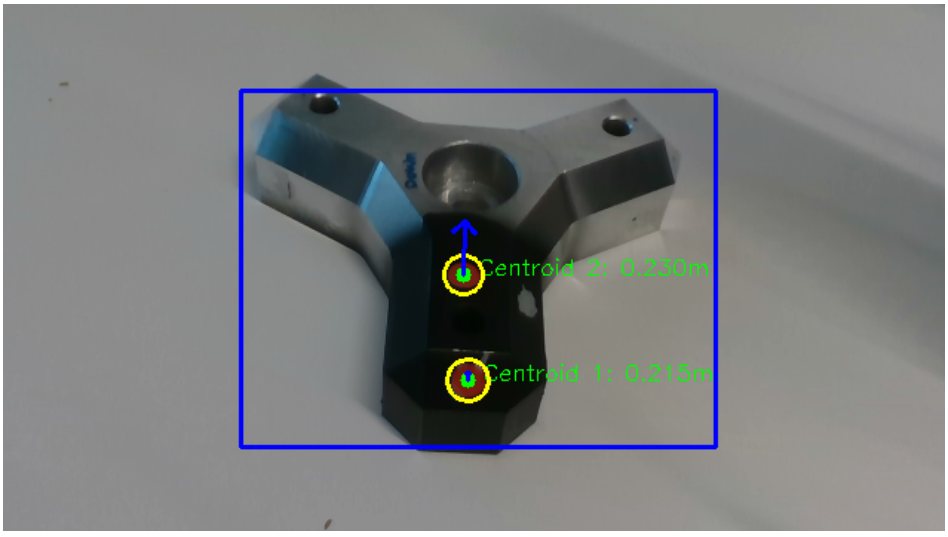


Figure 4-13: Test component used for real-time normal vector extraction in a low-curvature scenario

Results

In the following figures, green dots indicate stable points—those for which the angle difference between the two normal vectors falls within the acceptable range of 43° to 47°. Points with an angle difference outside this range are colored red. Thus, the dot colors directly represent the deviation of the measured angle difference from the desired value. Figures 4-14 and 4-15 show the results obtained with the proposed method over 1000 frames, whereas Figures 4-16, 4-17, and 4-18 display the results from the classic PCA method without filters. Table 4-7 summarizes the experimental outcomes illustrated in these figures.

As summarized in Table 4-7, the proposed method consistently achieves average angle differences near the theoretical value of 45° and delivers significantly higher stabilization, particularly when fewer points are used. In contrast, the classic PCA method without filters requires a larger number of points to approach the expected performance, yet it still exhibits lower stability.

Both simulation and real-world experiments reveal that the performance of classic PCA without filtering is highly dependent on the number of nearest neighbors used for normal estimation. For instance, when only 10 points ($k = 10$) are utilized, the classic PCA method yields an average angle difference of just 9.59°, indicating that the frames do not capture a sufficient number of points near the targets. As k increases to 100 and then to 300—thus effectively employing nearly all available points within the window—the average angle difference improves to 41.01° and 43.77°, respectively, and the stabilization ratio increases, reaching 55.30% at $k = 300$ which is similar stabilization as the proposed method at $k = 10$.

Notably, the proposed method outperforms classic PCA even when dramatically fewer points are used. With $k = 10$, it achieves an average angle difference of 43.93° and a stabilization ratio of 54%, values that are much closer to the theoretical 45° and approximately twice as stable as the best performance observed with classic PCA. These results demonstrate that, in low curvature scenarios, reliable normal extraction for multiple target points is achievable even when the targets are farther from the camera and only a limited number of points are detected.

Method	**Proposed Method**		**Classic PCA without Filters**		
k (Number of points used for PCA)	$k = 10$	$k = 100$	$k = 10$	$k = 100$	$k = 300$
Average Angle Difference	43.93°	43.65°	9.59°	41.01°	43.40°
Stabilization Ratio	54%	61.5%	0.00%	24.2%	55.30%

Table 4-7: Comparison of the proposed method and classic PCA without filters in high curvature scenario

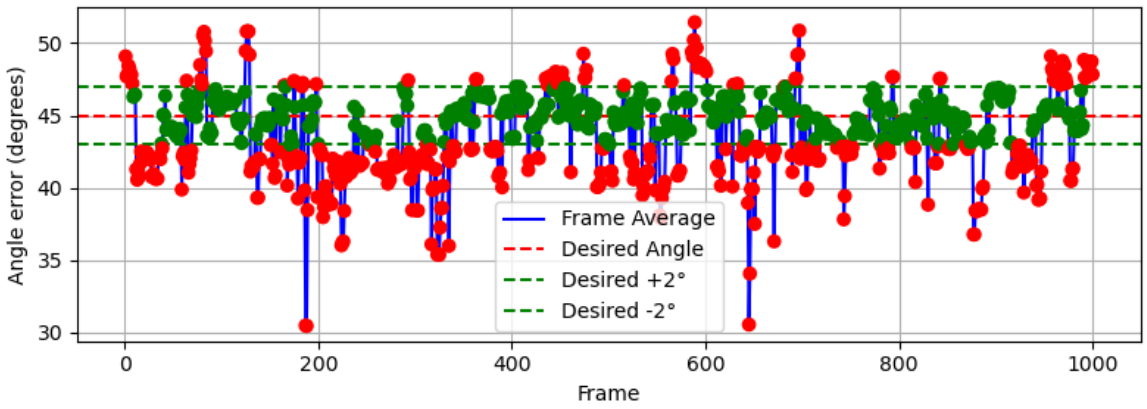


Figure 4-14: Proposed method ($\sigma = 100$, voxel length = 0.001m) with $k = 10$

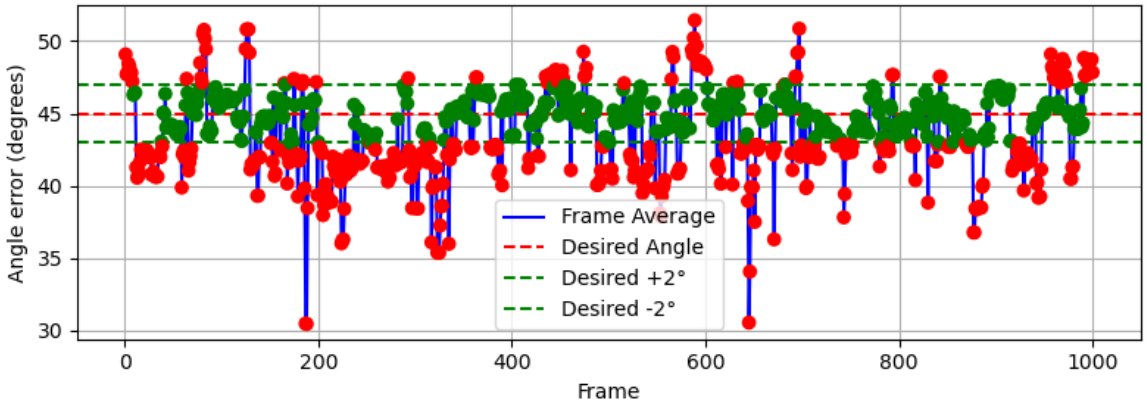


Figure 4-15: Proposed method ($\sigma = 100$, voxel length = 0.0004m) with $k = 100$

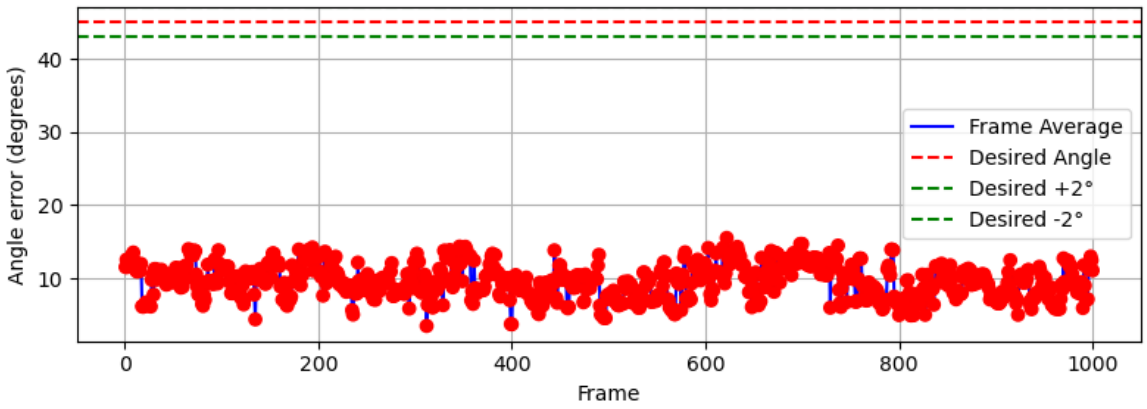


Figure 4-16: classic PCA without filter with $k = 10$

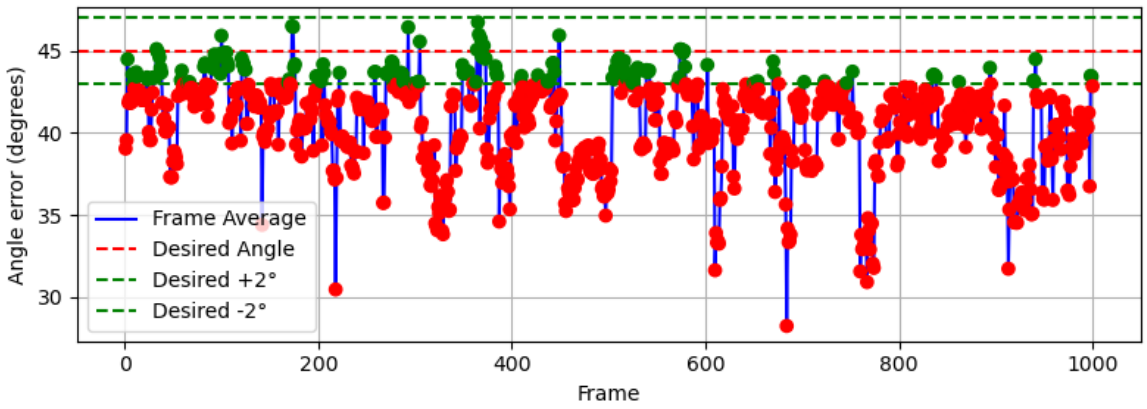


Figure 4-17: classic PCA without filter with $k = 100$

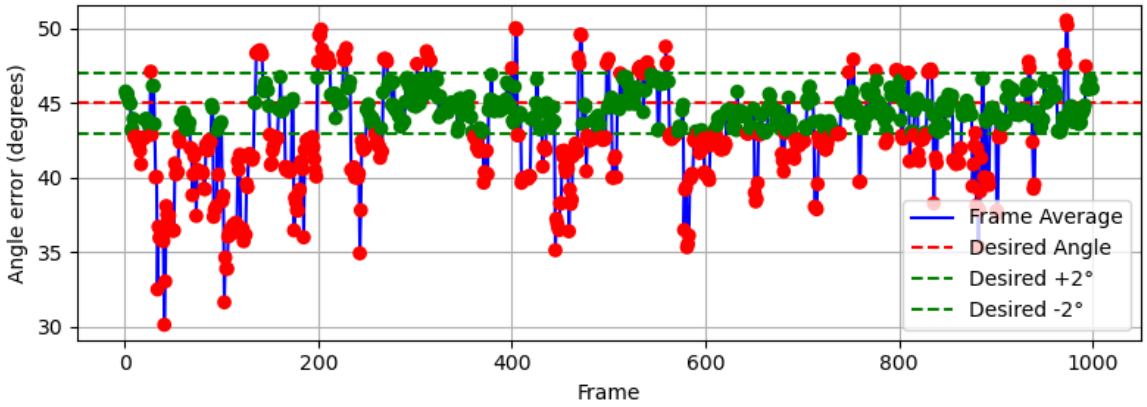


Figure 4-18: classic PCA without filter with $k = 300$

4-4-2 High curvature surface case

The high curvature case was evaluated using the same metrics as those applied in the low curvature case. The test component, shown in Figure 4-19, features a rounded shape with target points located on its curved surface. The true angle between the two reference normal vectors is 66°, and the distances from the depth camera to the two target points are approximately 0.285 m and 0.287 m, respectively.

For the proposed method, the Gaussian filter parameters were set to $\sigma = 10$ and $\sigma = 5$ for different numbers of k-nearest neighbors used in the PCA computation, while a fixed voxel length of 0.0004 m was maintained. With this voxel length, the average number of points in the local window around each target point was reduced from approximately 270 to 115, corresponding to an approximate 60% reduction in point cloud density.

As shown in Table 4-8, for an estimate to be included in the stabilization ratio, the angle difference between the two estimated normal vectors must lie within the range of 64° to 68°, given that the desired angle difference is 66°.

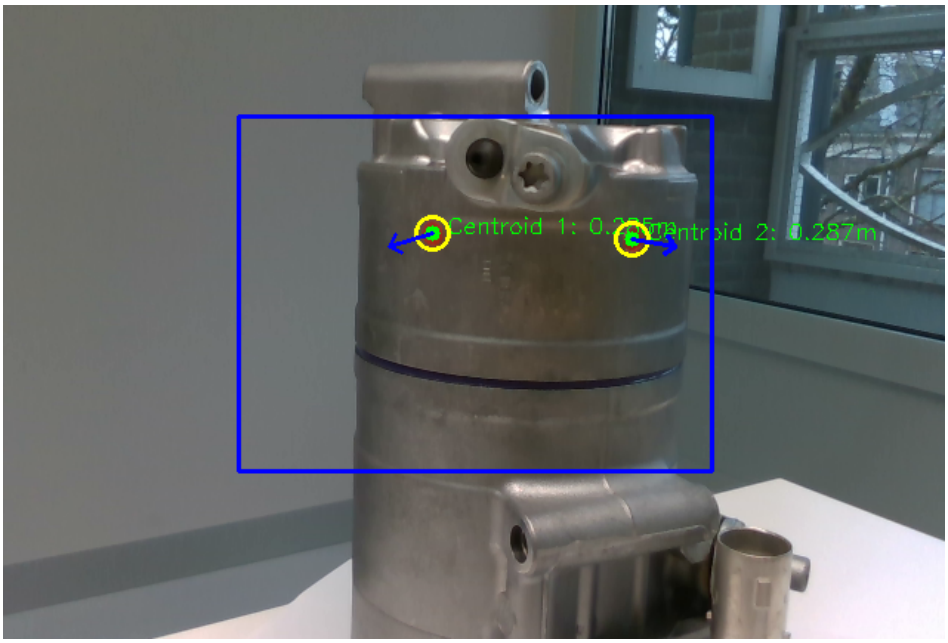


Figure 4-19: Test component used for real-time normal vector extraction in a high-curvature scenario

Results

The proposed method demonstrated significantly better performance in high-curvature scenarios compared to low-curvature ones. As shown in Table 4-8, it achieved an acceptable average angle difference ranging from 64 to 68 degrees and maintained high stabilization even when using a small number of nearest neighbors ($k = 10$). In contrast, the classic PCA method without filtering failed to achieve stable results with only 10 points. Although its performance may be acceptable under certain conditions, it was consistently outperformed by the proposed method. Even when all available points in the window were used for classic PCA, it did not reach the performance level of the proposed approach, which achieved superior results with just 10 point clouds.

In classic PCA without filtering, it is notable that the stabilization ratio decreased from around 86% to 78% when more point clouds were incorporated. This degradation is especially evident when estimating the plane of a target point on a curved surface. When PCA is applied to a point cloud on a curved surface, using only a few nearest neighbors tends to capture the local tangent plane accurately. However, as more points are added, they increasingly include data from regions where the surface deviates from the local plane. These additional points introduce geometric inconsistencies, causing the PCA to fit a plane that poorly represents the true local surface orientation. Consequently, the performance deteriorates, as reflected by the lower stabilization ratio.

Method	Proposed Method		Classic PCA without Filters		
k (Number of points used for PCA)	$k = 10$	$k = 50$	$k = 10$	$k = 50$	$k = 250$
Average Angle Difference	66.45°	66.72°	14.52°	65.27°	67.07°
Stabilization Ratio	96.5%	95%	0.00%	86.4%	78.25%

Table 4-8: Comparison of the proposed method and classic PCA without filters in high curvature scenario

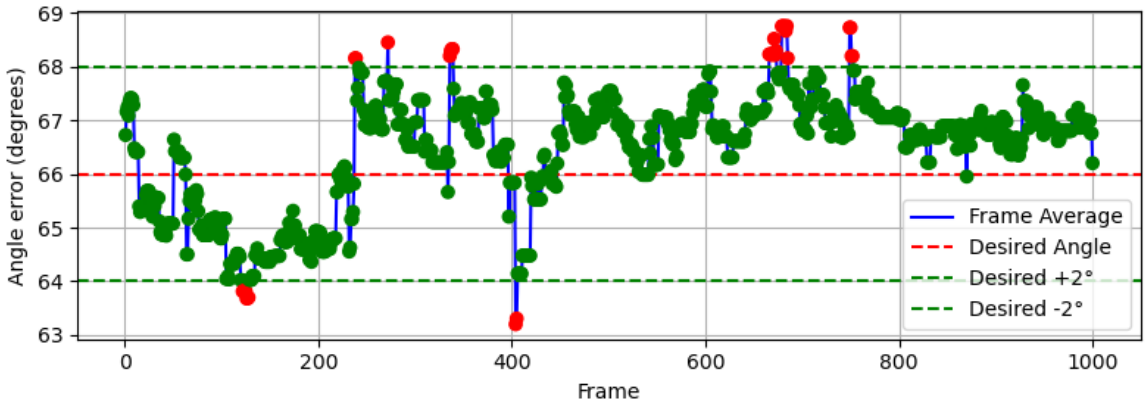


Figure 4-20: Proposed method ($\sigma = 10$, voxel length = 0.0004m) with $k = 10$

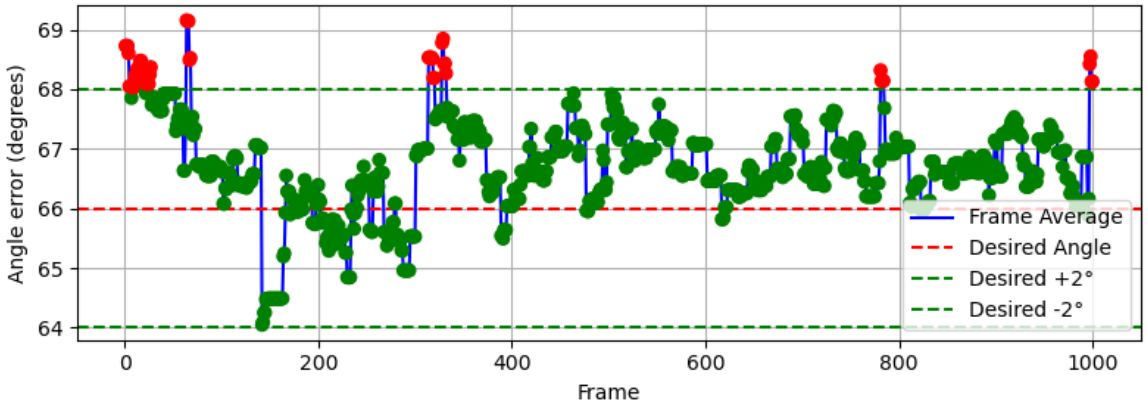


Figure 4-21: Proposed method ($\sigma = 5$, voxel length = 0.0004m) with $k = 50$

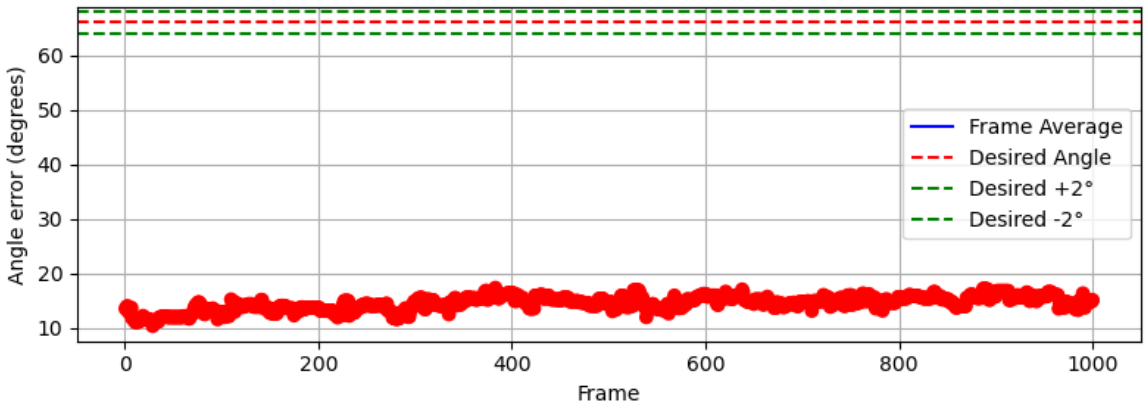


Figure 4-22: classic PCA without filter with $k = 10$

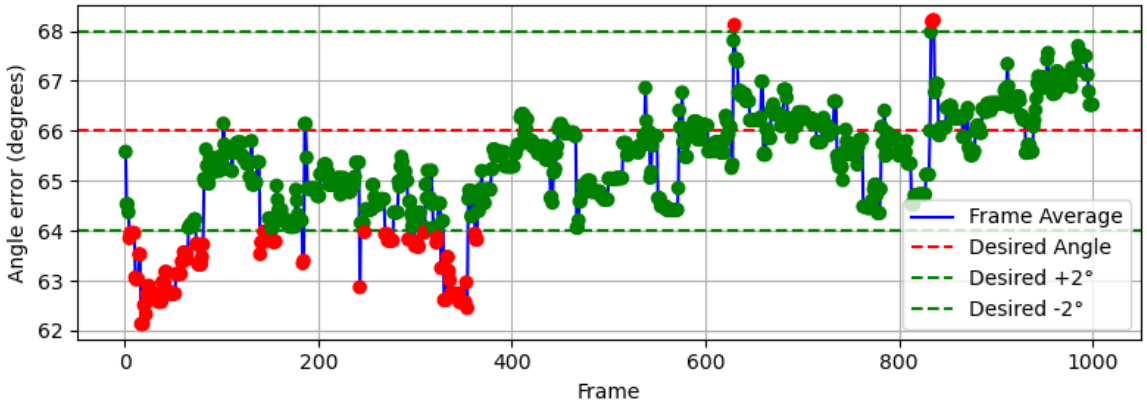


Figure 4-23: classic PCA without filter with $k = 50$

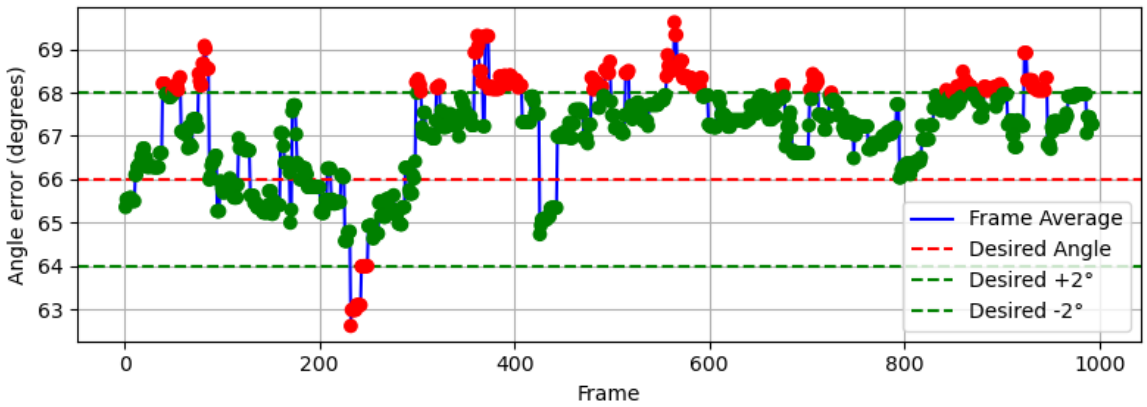


Figure 4-24: classic PCA without filter with $k = 250$

4-4-3 Evaluation of Real-world Experiment

The real-world experiment was conducted to compare the performance of the proposed method against the classic PCA method. Overall, the proposed method outperformed classic PCA in both low-curvature and high-curvature scenarios. Real-time normal vector estimation was performed using target points of the same size (indicated in red) as in the simulation. While the simulation assumed a single target point at the center of the camera's field of view, the real-world experiment involved the simultaneous detection of two target points, demonstrating the potential for multiple target detections in real time.

Interestingly, the experimental results revealed that the low-curvature case exhibited poorer stabilization compared to the high-curvature case. Intuitively, one would expect a low-curvature (nearly flat) surface to yield more stable depth measurements than a highly curved surface, as observed in the simulation. This discrepancy is likely attributable to environmental factors—such as variations in lighting and glare—that affected the stability of the depth frames.

It is hypothesized that the lower brightness and reduced glare associated with the metallic test component in the high-curvature scenario contributed to more stable depth measurements. To investigate this further, the brightness of the region of interest was evaluated by computing the average pixel intensity (ranging from 0 for black to 255 for white) over 500 frames. The low-curvature case exhibited an average brightness of 95.51, whereas the high-curvature case had a lower average brightness of 75.34. These findings suggest that obtaining stable depth frames under well-controlled lighting conditions could further improve overall performance.

Despite the unexpected poorer stabilization in the low-curvature case, the comparative analysis between the proposed method and classic PCA remains valid.

Furthermore, the experiment revealed a key disadvantage of the classic PCA method in high-curvature scenarios. As more nearest neighbors are incorporated, the performance of classic PCA degrades because additional points from regions where the surface deviates from a local plane are included. This degradation is particularly evident when estimating the plane at a target point on a curved surface. In contrast, the proposed method consistently outperformed classic PCA even when using only a few nearest neighbors.

A significant advantage of the proposed method is its ability to accurately extract the normal vector from a sparse point cloud. When the camera is positioned farther from the target, only a limited number of points are available. In such cases, the proposed method's robustness in estimating the normal vector using fewer points ensures reliable performance. This capability is critical for real-world applications, where data sparsity can otherwise compromise the accuracy of surface orientation estimation.

Chapter 5

Multi-objective optimization for path planning

5-1 Motivation

In automated systems utilizing robotic arms, ensuring safety and determining an accurate path are essential requirements. Numerous studies have explored optimal path planning and safety measures, offering technical solutions such as real-time obstacle avoidance and safe navigation within specific environments. However, this raises an important question: what truly defines the "best" path? While much of the existing research demonstrates technical approaches, it often focuses on a single aspect, neglecting comprehensive definitions of an optimal path that balance safety and adaptability.

This research aims to define the objectives of an optimal path while emphasizing safety considerations, such as avoiding collisions with obstacles like test components and sensor cables. The proposed method is based on a classic sampling-based path planner, making it accessible and user-friendly for those with a foundational understanding of optimization techniques.

Although path planning using multi-objective optimization has been extensively studied for mobile robots, similar research for robotic arms remains limited. One likely reason is that robotic arms are predominantly used in fixed environments, such as factory conveyor belts, where they repetitively handle identical tasks, such as moving the same product along a predefined path without encountering obstacles. In such cases, speed and efficiency are prioritized over adaptability.

In contrast, the scenario presented in this research requires the robotic arm to adapt frequently to new test components and dynamic environments where obstacles may vary. This variability necessitates an approach that not only ensures safety but also defines an optimal path adaptable to changing conditions, addressing a gap in current research.

5-2 Method

This section outlines our approach for optimal path planning for the robotic arm, which integrates inverse kinematics, concurrent processing, multi-objective optimization with constraints (filtering), and weighted path selection.

1. **Data Collection via Inverse Kinematics**
    - Compute joint configurations for desired end effector positions and orientations.
    - Generate waypoints and employ adaptive sampling to enhance end effector accessibility.
2. **Concurrent Path Planning**
    - Implement a thread pool to execute inverse kinematics, waypoint generation, and trajectory refinement in parallel.
    - Use synchronization techniques to manage shared resources and ensure consistency.
3. **Trajectory Optimization and Filtering**
    - Evaluate candidate trajectories with multiple criteria (e.g., minimum distance from obstacles, Joint space path length, smoothness).
    - Filter out trajectories that do not meet predefined feasibility thresholds.
4. **Optimal Candidate Selection**
    - Apply a weighted scoring function to rank candidates.
    - Select and execute the trajectory with the highest score.
5. **Implementation and Validation**
    - Develop the methodology using ROS, MoveIt, and Gazebo for modular integration and realistic simulation.
    - Validate on a UR5 robotic arm (6-DOF) using a system with a 24-core CPU and NVIDIA 4080 GPU.
    - Compare the optimized path against non-optimized, randomly generated paths.
6. **Algorithm Workflow**
    (a) Compute inverse kinematics and generate waypoints.
    (b) Perform concurrent path planning.
    (c) Apply multi-objective optimization with filtering constraints to obtain Pareto-optimal trajectories.
    (d) Use weighted sum selection to choose the optimal path.
    (e) Execute the selected path to ensure safe and efficient movement.

Multi-Objective Optimization Problem for Trajectory Selection

In our application, we are not generating trajectories on a per-waypoint basis; rather, we are selecting the best candidate trajectory from a precomputed set. Each candidate trajectory is evaluated according to three objective functions:

$$\begin{aligned} f_1(x): &\quad \text{Minimum distance from obstacles,} \\ f_2(x): &\quad \text{Joint space path length,} \\ f_3(x): &\quad \text{Smoothness.} \end{aligned}$$

The multi-objective optimization problem for selecting a candidate trajectory can be formulated as follows:

$$\begin{aligned} &\text{Find a candidate trajectory } x, \\ &\text{to maximize } f_1(x) \quad \text{and minimize } \big(f_2(x), f_3(x)\big), \\ &\text{subject to:} \\ &\qquad f_1(x) \geq \epsilon, \\ &\qquad f_2(x) \leq \tau, \\ &\qquad f_3(x) \leq \sigma, \end{aligned}$$

where:

- $f_1(x)$ measures the safety of the trajectory by considering the minimum distance to obstacles (to be maximized). Here, ϵ represents the minimum acceptable obstacle distance (e.g., 0.001 m).
- $f_2(x)$ quantifies the joint space path length, reflecting the total movement of the robot's joints (to be minimized). The upper bound τ represents the maximum allowable joint length (e.g., 20.0).
- $f_3(x)$ assesses the smoothness of the trajectory, typically approximated by a jerk or acceleration measure (to be minimized). The threshold σ represents the maximum allowed smoothness value (e.g., 0.5).

In this formulation, the constraints ensure that candidate trajectories not only provide a good trade-off between the objectives but also meet feasibility requirements by filtering out trajectories that do not satisfy safety, joint length, and smoothness thresholds.

In this selection-based framework, each candidate trajectory x is associated with a feature vector $\mathbf{f}(x) = \big(f_1(x), f_2(x), f_3(x)\big)$. Since the objectives may conflict—for example, increasing safety might lead to a longer path—our MOP requires balancing these criteria. The final selection can be based on a weighted scoring function such as:

$$\text{score}(x) = \alpha\, f_1(x) - \beta\, f_2(x) - \gamma\, f_3(x), \tag{5-1}$$

where α, β, and γ are weights that reflect the relative importance of each objective.

The candidate trajectory with the highest score is then chosen as the optimal trajectory for execution.

Dominance Relation and Pareto Frontier

Let $\mathbf{f}(x) = (f_1(x), f_2(x), f_3(x))$ be the objective vector for a candidate trajectory x, where

- $f_1(x)$ is to be maximized (e.g., safety, represented by the minimum distance from obstacles),
- $f_2(x)$ is to be minimized (e.g., joint space path length),
- $f_3(x)$ is to be minimized (e.g., smoothness).

For two candidate trajectories with objective vectors $\mathbf{y} = \mathbf{f}(x_1)$ and $\mathbf{z} = \mathbf{f}(x_2)$, we say that

$$\mathbf{y} \preceq \mathbf{z} \quad (\text{“}x_1 \text{ dominates } x_2\text{”})$$

if and only if:

1. $y_1 \geq z_1$ (since a higher f_1 is preferable),
2. $y_i \leq z_i$ for $i = 2, 3$ (since lower values of f_2 and f_3 are preferable),
3. and at least one of these inequalities is strict.

In other words, $\mathbf{y} \preceq \mathbf{z}$ indicates that the candidate trajectory x_1 is at least as good as x_2 in every objective and strictly better in at least one objective.

A candidate trajectory x^* is said to be *Pareto optimal* if there is no other candidate x' such that

$$\mathbf{f}(x') \preceq \mathbf{f}(x^*),$$

i.e., no other trajectory dominates x^*. The set of all Pareto optimal trajectories is called the *Pareto set*, and the corresponding set of objective vectors in the objective space is known as the *Pareto frontier*:

$$\text{Pareto Frontier} = \{\, \mathbf{f}(x) \mid x \text{ is Pareto optimal} \}.$$

5-3 Objectives in optimization

In the context of a multi-objective optimization problem, three primary objectives are considered: **minimum distance from obstacles**, **joint space path length**, and **smoothness**. These objectives are crucial for ensuring both the safety and efficiency of robotic arm path planning.

1. **Minimum Distance from Obstacles**

   The minimum distance from obstacles is a critical factor in path planning. This objective measures the shortest distance between the robot and any obstacles along the waypoints of the planned path, ensuring that an acceptable safety distance is maintained to avoid collisions.

Using this metric as an objective rather than a hard constraint offers several advantages. First, it allows the optimization process to balance safety with other critical factors, such as the efficiency of the path. A strict constraint could force the robot to take overly detoured routes, which might result in longer trajectories, higher energy consumption, and increased complexity in movement. In contrast, treating the minimum distance as an objective enables the algorithm to evaluate trade-offs more flexibly. It can choose a route that just meets the safety requirements while still optimizing for path length and smoothness.

Moreover, incorporating safety as an objective avoids prematurely excluding candidate paths that may only marginally violate a strict safety threshold but offer significant benefits in terms of overall efficiency. This approach ensures a more comprehensive evaluation of potential solutions by retaining borderline candidates in the optimization process, ultimately leading to a more balanced and practical trajectory.

To mitigate these potential inefficiencies, the second objective, joint space path length, is introduced as a complementary metric.

2. **Joint Space Path Length**

   Joint space path length is preferred over Cartesian path length because it accounts for the movement of all joints of the robotic arm. While Cartesian path length is useful for measuring the shortest distance between two points in space, it does not fully capture the complexities of the robot's joint movements.

   Joint space path length provides a more comprehensive measure by reflecting the overall energy consumption and mechanical strain on the robot arm. By considering the motion of each joint, this objective ensures that the robot operates efficiently while minimizing unnecessary stress on its mechanical components. This leads to more sustainable and reliable robot performance over time.

3. **Smoothness**

   Smoothness is the third critical objective, aimed at preventing overshooting and abrupt movements during the execution of the planned path. Abrupt motions can compromise the accuracy of tasks and increase the risk of mechanical failures or instability in the robot's operation.

   By prioritizing smoothness, the robot achieves controlled and precise movements, which contribute to both safety and operational efficiency. Smooth trajectories reduce the likelihood of unexpected behaviors, ensuring that the robot can perform tasks reliably and consistently.

These three objectives work in tandem to balance safety, efficiency, and reliability in robotic arm path planning. By optimizing for minimum distance from obstacles, joint space path length, and smoothness, the proposed method aims to achieve an optimal and practical path that is both safe and efficient for dynamic and variable environments.

Trading Off Relationship

In multi-objective optimization, achieving an optimal balance between conflicting objectives is crucial for effective path planning. The three selected objectives—**minimum distance**

from obstacles, **joint space path length**, and **smoothness**—often exhibit trade-offs that must be carefully managed to attain a feasible and efficient path for the robotic arm.

1. **Minimum Distance from Obstacles vs. Joint Space Path Length**

   Optimizing for the minimum distance from obstacles prioritizes safety by ensuring that the robot maintains a safe buffer around all obstacles. However, this objective can lead to longer paths as the robot may need to navigate around obstacles rather than taking the most direct route. Consequently, increasing the minimum obstacle distance can result in a higher joint space path length, thereby consuming more energy and potentially increasing wear and tear on the robotic arm. This trade-off necessitates a balanced approach where the safety margins are sufficiently large to prevent collisions without excessively elongating the path.

2. **Joint Space Path Length vs. Smoothness**

   Minimizing the joint space path length aims to reduce the energy consumption and mechanical strain on the robot by shortening the overall movement required. However, a focus on path length alone may compromise smoothness, leading to abrupt or jerky motions that can affect the precision and reliability of the robot's operations. Conversely, prioritizing smoothness may require more complex joint movements, potentially increasing the joint space path length. Therefore, it is essential to harmonize these objectives to achieve a path that is both energy-efficient and mechanically stable.

3. **Minimum Distance from Obstacles vs. Smoothness**

   Ensuring a minimum distance from obstacles can also influence the smoothness of the robot's path. Navigating around obstacles may require sudden changes in direction or speed, which can result in less smooth trajectories. To maintain smoothness while adhering to safety constraints, the path planner must generate trajectories that curve gently around obstacles rather than making sharp deviations. This balance ensures that the robot can move safely without introducing abrupt movements that could jeopardize task accuracy or mechanical integrity.

The interplay between these objectives highlights the complexity of path planning in dynamic and variable environments. To effectively manage these trade-offs, the proposed optimization method employs a weighted approach, assigning appropriate significance to each objective based on the specific requirements of the application. By doing so, it ensures that the robotic arm can navigate safely around obstacles, move efficiently with minimal energy consumption, and execute movements smoothly to maintain operational precision and reliability.

Additionally, the use of a sampling-based path planner facilitates the exploration of diverse solutions, enabling the identification of Pareto-optimal paths that offer the best possible compromises between the conflicting objectives. This approach allows for flexibility in adjusting the weights of each objective, providing a customizable framework that can adapt to varying operational demands and environmental conditions.

In summary, understanding and effectively managing the trading off relationships between minimum distance from obstacles, joint space path length, and smoothness is essential for developing a robust and efficient path planning strategy for robotic arms. The proposed method addresses these challenges by integrating a balanced optimization framework that prioritizes safety, efficiency, and reliability in path generation.

5-3-1 Minimum distance from obstacles

Ensuring the safety of robotic operations is paramount, particularly in environments where robots interact closely with humans or navigate complex terrains with numerous obstacles. This section delineates the safety protocols implemented in our motion planning algorithm, focusing on maintaining a minimum distance from obstacles to prevent collisions and ensure reliable performance.

The primary safety criterion in our motion planning framework is maintaining a **minimum obstacle distance** ($d_{\min}$) from all obstacles within the operational environment. This metric ensures that the robot's end effector and body do not inadvertently collide with surrounding objects, thereby safeguarding both the robot and its environment.

Definition and Importance

- **Minimum Obstacle Distance ($d_{\min}$):** The smallest distance between the robot and any obstacle at any given time. Maintaining a non-zero $d_{\min}$ is essential for collision avoidance and operational safety.
- **Distance Function ($d(w_i)$):** For each waypoint w_i, $d(w_i)$ denotes the distance to the nearest obstacle. This value is computed using the axis-aligned bounding box (AABB) method (see Figure 5-1) and is updated whenever the robot's state or the environment changes. Although the AABB approach is computationally efficient and conserves memory, its axis-aligned nature may not perfectly represent the true 3D shape of an object. Consequently, AABB checks serve as a coarse preliminary filter, with more detailed collision checks performed if a potential collision is detected. The MoveIt! framework handles these dynamic AABB computations and collision assessments. For further details, please refer to [23].

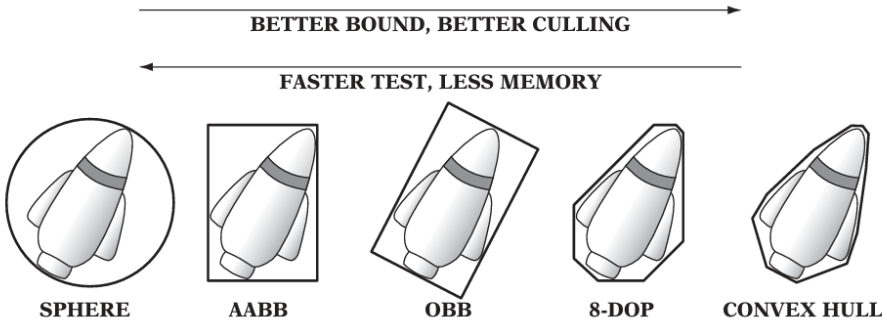


Figure 5-1: Scheme of the thread pool [22]

Mathematical Formulation

To formalize the above procedure, we define the following:

- Let $\Theta = \{w_1, w_2, \ldots, w_N\}$ represent the set of waypoints in the robot's trajectory.

- The function $C(w_i)$ evaluates whether waypoint w_i is in collision:

$$C(w_i) = \begin{cases} \text{True}, & \text{if } w_i \text{ is in collision,} \\ \text{False}, & \text{otherwise.} \end{cases}$$

- The distance to the nearest obstacle from waypoint w_i is denoted as $d(w_i)$.

The update rule for $d_{\min}$ can thus be expressed as:

$$d_{\min} = \min_{w_i \in \Theta} \left(\mathbb{I}_{\{C(w_i)=\text{True}\}} \cdot 0 + \mathbb{I}_{\{C(w_i)=\text{False}\}} \cdot d(w_i) \right),$$

where $\mathbb{I}$ is the indicator function.

Algorithmic Implementation

The algorithm for maintaining and updating the minimum obstacle distance is outlined below:

1. **Initialization:**

$$d_{\min} = \infty$$

   Initially, $d_{\min}$ is set to infinity to ensure that any subsequent computed distance will be smaller, allowing the algorithm to update $d_{\min}$ appropriately as waypoints are evaluated.

2. **Waypoint Evaluation:** For each waypoint w_i in the robot's planned trajectory, perform the following checks:

$$d_{\min} = \min_{i \in \{1,2,\ldots,N\}} \begin{cases} 0, & \text{if } w_i \text{ is in collision,} \\ d(w_i), & \text{otherwise.} \end{cases}$$

   Here, N denotes the total number of waypoints in the trajectory. The algorithm iterates through each waypoint to determine its proximity to the nearest obstacle.

3. **Collision Detection and Distance Update:**

   - If a waypoint w_i is found to be in collision with an obstacle, $d_{\min}$ is immediately set to 0, indicating a collision state.
   - If there is no collision at waypoint w_i, $d(w_i)$ is computed and compared against the current $d_{\min}$. The smallest of these distances is retained as the updated $d_{\min}$.

The following pseudocode outlines the implementation of the minimum obstacle distance calculation:

Algorithm 1 Minimum Obstacle Distance Calculation

Require: Set of waypoints $\Theta = \{w_1, w_2, \ldots, w_N\}$
Ensure: Minimum obstacle distance $d_{\min}$
Initialize $d_{\min} \leftarrow \infty$
for each waypoint $w_i \in \Theta$ **do**
  if $C(w_i) = \text{True}$ **then**
    $d_{\min} \leftarrow 0$
    break
  else
    Compute $d(w_i)$
    $d_{\min} \leftarrow \min(d_{\min}, d(w_i))$
  end if
end for
return $d_{\min}$

Implementing a minimum obstacle distance metric is a fundamental aspect of ensuring safety in robotic motion planning. By systematically evaluating each waypoint for potential collisions and maintaining a positive $d_{\min}$, the robot can navigate complex environments reliably and securely. Future work may explore adaptive thresholds for $d_{\min}$ based on environmental dynamics and robot speed, further enhancing the safety and efficiency of robotic operations.

5-3-2 Joint Space Path Length

Understanding the efficiency and precision of a robotic arm's movement is crucial in robotic path planning. The Joint Space Path Length provides a quantitative measure of the total movement undertaken by the robot's joints as it navigates through its trajectory. This metric is essential for optimizing motion, minimizing energy consumption, and ensuring smooth operation.

Joint Space Path Length Formula

The Joint Space Path Length implemented in the code can be mathematically expressed as:

$$\text{Path Length} = \sum_{i=1}^{N-1} \sqrt{\sum_{j=1}^{M} (q_{i,j} - q_{i-1,j})^2}$$

Where:

- N: Total number of trajectory points.
- M: Total number of joints in the robot.
- $q_{i,j}$: Position of the j-th joint at the i-th trajectory point.

Explanation:

1. **Inner Summation:**
$$\sum_{j=1}^{M} (q_{i,j} - q_{i-1,j})^2$$
This calculates the squared differences of all joint positions between two consecutive trajectory points (i-th and $(i-1)$-th) in Joint Space. The result represents the squared distance between these two points.

2. **Outer Summation:**
$$\sum_{i=1}^{N-1}$$
This sums up the Euclidean distances over all pairs of consecutive trajectory points. The total represents the entire path length in Joint Space.

Example: Joint Space Path Length Calculation

Let us consider a robotic arm with three joints: **Shoulder**, **Elbow**, and **Wrist**. The robot moves through three trajectory points A, B, and C. At each point, the joint positions (angles in degrees) are as follows:

$$\textbf{3 Joints} = [\text{Shoulder, Elbow, Wrist}]$$

$$A : [10^\circ, 20^\circ, 30^\circ] \quad B : [15^\circ, 25^\circ, 35^\circ] \quad C : [20^\circ, 30^\circ, 40^\circ]$$

1. **Distance Between Points**
   - **Distance between A and B:** The Euclidean distance between A and B in Joint Space is calculated as:
   $$\sqrt{(15-10)^2 + (25-20)^2 + (35-30)^2} = \sqrt{5^2 + 5^2 + 5^2} = \sqrt{75} \approx 8.66$$
   - **Distance between B and C:** Similarly, the Euclidean distance between B and C is:
   $$\sqrt{(20-15)^2 + (30-25)^2 + (40-35)^2} = \sqrt{5^2 + 5^2 + 5^2} = \sqrt{75} \approx 8.66$$

2. **Cumulative Path Length**
   - To calculate the total path length in Joint Space, sum the distances between consecutive trajectory points:
   $$\text{Path Length} = 8.66 + 8.66 = 17.32$$

3. **Interpretation of the Result**
   - The total path length of 17.32 ° indicates that the robot has moved a total of 17.32° as it traveled from point A to point C. This value represents the total change in the joint positions across the trajectory.

The Joint Space Path Length serves as a fundamental metric for evaluating and optimizing the motion of robotic arms. By quantifying the total joint movements, engineers can assess the efficiency of path planning algorithms, minimize unnecessary movements, and enhance the overall performance of robotic systems. This metric is instrumental in applications where precision and energy efficiency are paramount.

5-3-3 Motion Smoothness

Ensuring smooth and precise motion in robotic systems is critical for both operational efficiency and the longevity of mechanical components. Smoothness is typically quantified by evaluating the jerk, which represents the rate of change of acceleration. This section presents the derivation of a simplified jerk calculation method, followed by the calculations for velocity and acceleration. These formulations facilitate the assessment and optimization of motion trajectories to achieve minimal abrupt changes, thereby enhancing overall system performance.

Smoothness: Derivation of Simplified Jerk Calculation

Smoothness in robotic motion is quantified by minimizing the jerk, which is the derivative of acceleration with respect to time. Mathematically, the smoothness S can be expressed as the sum of the squared jerk values across the trajectory:

$$S = \sum_{i=2}^{N} J_i^2, \tag{5-2}$$

$$\text{where} \quad J_i \approx x_i - 2x_{i-1} + x_{i-2}. \tag{5-3}$$

Here, x_i represents the position of the robot's joint at the i-th discrete time step, and N is the total number of trajectory points. Squaring the jerk ensures non-negative contributions and emphasizes large deviations, promoting smoother motion by penalizing abrupt changes in acceleration.

Explanation:

To compute the smoothness, it is essential to first determine the velocity and acceleration at each trajectory point. These are approximated using finite difference methods as follows:

To characterize the motion, we first approximate the velocity at each trajectory point using a first-order finite difference. Specifically, the velocity v_i at the i-th point is given by

$$v_i \approx \frac{x_{i+1} - x_i}{\Delta t}, \tag{5-4}$$

where Δt denotes the time interval between consecutive points. This formulation captures the instantaneous rate of change of the position.

Building upon this, the acceleration a_i is derived as the change in velocity over time. By applying the finite difference method to the velocity values, we obtain

$$\begin{aligned} a_i &\approx \frac{v_{i+1} - v_i}{\Delta t} \\ &= \frac{\frac{x_{i+2}-x_{i+1}}{\Delta t} - \frac{x_{i+1}-x_i}{\Delta t}}{\Delta t} \\ &= \frac{x_{i+2} - 2x_{i+1} + x_i}{\Delta t^2}. \end{aligned} \tag{5-5}$$

This second-order finite difference approximation effectively measures the change in velocity, thereby providing a reliable estimate of the acceleration at each point.

Jerk J_i represents the rate of change of acceleration and is calculated as the finite difference of the accelerations. Starting with the definition, we have

$$\begin{aligned} J_i &= \frac{\partial a}{\partial t} \\ &\approx \frac{a_{i+1} - a_i}{\Delta t}, \end{aligned} \tag{5-6}$$

where the acceleration at point i is approximated by the second-order finite difference. By substituting the expressions

$$a_i \approx \frac{x_{i+2} - 2x_{i+1} + x_i}{\Delta t^2} \quad \text{and} \quad a_{i+1} \approx \frac{x_{i+3} - 2x_{i+2} + x_{i+1}}{\Delta t^2},$$

we obtain

$$\begin{aligned} J_i &= \frac{\frac{x_{i+3}-2x_{i+2}+x_{i+1}}{\Delta t^2} - \frac{x_{i+2}-2x_{i+1}+x_i}{\Delta t^2}}{\Delta t} \\ &= \frac{x_{i+3} - 3x_{i+2} + 3x_{i+1} - x_i}{\Delta t^3}. \end{aligned} \tag{5-7}$$

Equation (5-7) employs points up to x_{i+3}, which can be computationally intensive in real-time applications. One natural approach to simplify this is to approximate the distant points x_{i+3} and x_{i+2} in terms of nearer ones. Assuming a small time step Δt, we might write

$$x_{i+3} \approx x_{i+2} + (x_{i+2} - x_{i+1}), \tag{5-8}$$

$$x_{i+2} \approx x_{i+1} + (x_{i+1} - x_i), \tag{5-9}$$

and substitute these approximations into Equation (5-7). This leads to

$$\begin{aligned} J_i &\propto (x_{i+2} + (x_{i+2} - x_{i+1})) - 3x_{i+2} + 3x_{i+1} - x_i \\ &= 2x_{i+2} - x_{i+1} - 3x_{i+2} + 3x_{i+1} - x_i \\ &= -x_{i+2} + 2x_{i+1} - x_i, \end{aligned} \tag{5-10}$$

and further replacing x_{i+2} using Equation (5-9) yields

$$\begin{aligned} J_i &\propto -(x_{i+1} + (x_{i+1} - x_i)) + 2x_{i+1} - x_i \\ &= -2x_{i+1} + x_i + 2x_{i+1} - x_i \\ &= 0. \end{aligned} \tag{5-11}$$

The cancellation observed in Equation (5-11) indicates that this naive approximation is insufficient, prompting a reassessment that incorporates lower-order terms.

A more robust approach leads us to a final simplified jerk formula that uses only immediate trajectory points:

$$J_i \approx x_i - 2x_{i-1} + x_{i-2}. \tag{5-12}$$

This final expression relies solely on the neighboring points x_i, x_{i-1}, and x_{i-2}, thereby enhancing computational efficiency while still capturing the essential dynamics of smooth motion. Although $\tilde{J}_i$ is not the exact time derivative of acceleration, its variations along the trajectory effectively penalize abrupt changes, making it a practical proxy for evaluating and optimizing motion smoothness in robotic systems.

Implementation Considerations

- **Trajectory Point Handling**

  The first two trajectory points ($i = 0$ and $i = 1$) do not have sufficient preceding points to compute the jerk using Equation (5-12). Therefore, these points are typically excluded from the jerk calculation or handled using alternative methods, such as assuming zero jerk or utilizing boundary conditions.

- **Discrete Trajectory Representation**

  The trajectory consists of N discrete points, indexed from 0 to $N - 1$. At each index i, the position x_i of each joint is recorded. This discrete representation allows for the application of finite difference methods to approximate derivatives, facilitating the computation of velocity, acceleration, and jerk.

The derivation of a simplified jerk calculation method, as presented in this section, provides an efficient means to evaluate motion smoothness in robotic systems. By leveraging finite difference approximations and reducing the reliance on distant trajectory points, the proposed approach balances computational efficiency with the accuracy necessary for precise motion control. These formulations are integral to optimizing motion trajectories, thereby enhancing the overall performance and longevity of robotic systems.

5-4 Thread Pool

5-4-1 Structure

To mitigate computational time and enhance the efficiency of the simulation, a thread pool approach is employed. A thread pool manages a collection of threads that can execute tasks concurrently, thereby optimizing resource utilization and reducing the overhead associated with thread creation and destruction.

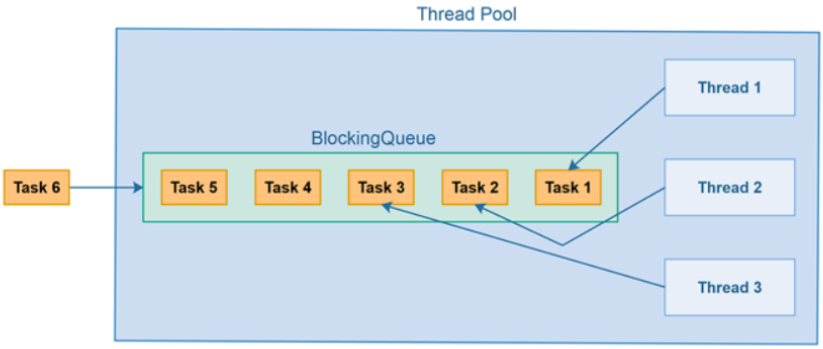


Figure 5-2: Scheme of the thread pool [15]

Overview of Thread Pool and Blocking Queue

A fundamental component of the thread pool mechanism is the *blocking queue*. The blocking queue serves as a task scheduler, where incoming tasks are enqueued and assigned to available threads. When a thread completes its current task, it retrieves the next task from the queue. If no tasks are available, the thread remains idle until new tasks are introduced. This ensures a balanced distribution of tasks and prevents thread starvation.

Implementation in Simulation

In the context of our simulation, a thread pool consisting of 10 threads is established. Each thread operates independently with its own set of resources to prevent conflicts and ensure thread safety. Specifically, the following components are assigned to each thread:

- **Planning Scene:** This refers to the environmental context and the detected obstacles within the simulation. By assigning a separate planning scene to each thread, we eliminate the risk of concurrent modifications that could lead to inconsistencies or collisions between threads.
- **Move Group Interface:** This interface manages the robot's state, objectives, and the selection of path planners. Isolating the Move Group Interface per thread ensures that each thread can independently manage its path planning strategies without interference from other threads.

Rationale for Independent Planning Scenes and Interfaces

The decision to prevent threads from sharing the planning scene and the Move Group Interface is driven by the need to avoid collisions and ensure the integrity of each thread's operations. By maintaining separate instances:

1. **Collision Avoidance:** Independent planning scenes prevent threads from inadvertently interacting with each other's environmental data, thereby avoiding potential collisions.

2. **Concurrent Strategy Execution:** Threads can concurrently execute diverse strategies, such as employing different path planners or targeting distinct target points. This parallelism enhances the simulation's robustness and efficiency.

Local Coordinate Application:

The target points are located on the non-planar surface and as the test component moves after impact, the position of the target points are going to be slightly changing. Using the estimated normal vector of each target point, setting the local coordinate system is crucial.

Assume that an object's surface is characterized by a normal vector:

$$\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}.$$

The first step is to normalize this vector to obtain a unit normal:

$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|}, \quad \|\mathbf{n}\| = \sqrt{n_x^2 + n_y^2 + n_z^2}.$$

In many applications, the robot must approach from the opposite direction of the surface, so the local z-axis is defined as

$$\mathbf{z}_{\text{local}} = -\hat{\mathbf{n}}.$$

To complete the coordinate system, two additional orthogonal axes are required. However, a direct choice may lead to degeneracies when the normal aligns with one of the standard basis vectors. To avoid this, an arbitrary vector $\mathbf{a}$ is selected as follows:

$$\mathbf{a} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{if } |n_z| < 0.99, \qquad \mathbf{a} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{otherwise.}$$

Using $\mathbf{a}$, the local x-axis is computed via the cross product:

$$\mathbf{x}_{\text{local}} = \frac{\mathbf{z}_{\text{local}} \times \mathbf{a}}{\|\mathbf{z}_{\text{local}} \times \mathbf{a}\|}.$$

Next, the local y-axis is determined by ensuring orthogonality:

$$\mathbf{y}_{\text{local}} = \frac{\mathbf{z}_{\text{local}} \times \mathbf{x}_{\text{local}}}{\|\mathbf{z}_{\text{local}} \times \mathbf{x}_{\text{local}}\|}.$$

Thus, the three axes $\{\mathbf{x}_{\text{local}}, \mathbf{y}_{\text{local}}, \mathbf{z}_{\text{local}}\}$ form an orthonormal basis that defines the local coordinate system.

This local frame can be embedded into a rotation matrix R that maps local coordinates to global coordinates by placing the axes as columns:

$$R = \begin{bmatrix} \mathbf{x}_{\text{local}} & \mathbf{y}_{\text{local}} & \mathbf{z}_{\text{local}} \end{bmatrix} = \begin{bmatrix} x_{\text{local},x} & y_{\text{local},x} & z_{\text{local},x} \\ x_{\text{local},y} & y_{\text{local},y} & z_{\text{local},y} \\ x_{\text{local},z} & y_{\text{local},z} & z_{\text{local},z} \end{bmatrix}.$$

In many cases, an additional rotation about the local z-axis is applied to optimize the robot's approach. Let θ be the desired rotation angle (in radians). The rotation about the z-axis is given by:

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The final rotation matrix combining the local frame and the additional rotation becomes:

$$R_{\text{final}} = R \cdot R_z(\theta).$$

Now, suppose we define a target point in the local coordinate system as:

$$\mathbf{L} = \begin{bmatrix} \text{offset}_x \\ \text{offset}_y \\ \text{approach\_distance} \end{bmatrix}$$

where $\text{offset}_x = 0$, $\text{offset}_y = 0.0275\,\text{m}$, and $\text{approach\_distance} = 0.115\,\text{m}$.

As shown in Figure 5-3, the predefined automatic hammer mounted on the robot arm is asymmetric. The tip of the hammer is 0.0275 m away from its center in y-axis direction, and the distance from the robot arm (not the hammer) to the target point is set to be 0.115 m. This configuration maintains a clearance of approximately 1 cm between the tip of the hammer and the target point. The value of the approach_distance is adjustable as needed in different scenarios.

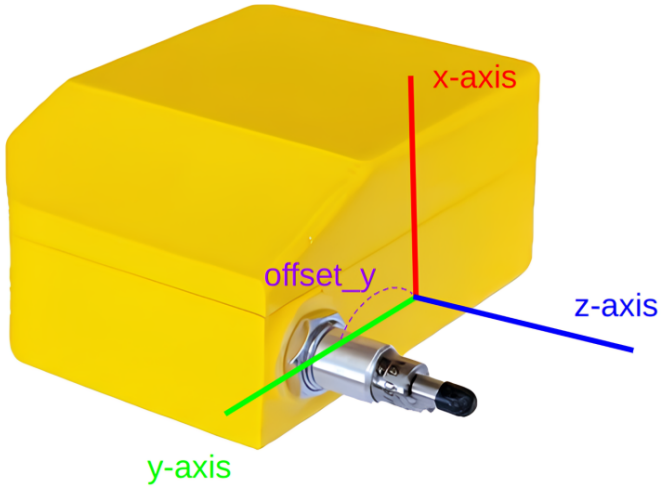


Figure 5-3: Asymmetric automatic hammer

To find the corresponding global position, the transformation is performed by rotating the local target and then translating it by the global centroid $\mathbf{C}$ of the target points:

$$\mathbf{p}_{\text{global}} = R_{\text{final}} \cdot \mathbf{L} + \mathbf{C},$$

where

$$\mathbf{C} = \begin{bmatrix} \text{centroid.x} \\ \text{centroid.y} \\ \text{centroid.z} \end{bmatrix}.$$

The establishment of this local coordinate system has profound benefits. First, it provides a natural and intuitive frame for defining the robot's approach relative to the target points. The computation of the target pose, given by

$$\mathbf{p}_{\text{global}} = R_{\text{final}} \cdot \mathbf{L} + \mathbf{C},$$

becomes more manageable as the offsets and rotations are defined in an object-centric manner rather than in the often more complex global frame. Second, this approach enhances precision and safety. By aligning the robot's end-effector with the inherent geometry of the object, the risk of collisions is minimized, and the robot can position itself accurately for tasks such as grasping or assembly.

5-4-2 Adaptive Sampling for Accessible Angles

After applying the coordinate transformation, each task proceeds with an **adaptive sampling process** to identify the optimal accessible angles for the end effector. Unlike a fixed-offset approach—which can limit the robot's ability to reach certain target points—adaptive sampling dynamically adjusts the end effector's orientation to enhance accessibility. This adjustment is especially important when dealing with **asymmetric end effectors** and **irregularly shaped components**, where precise tip placement is critical.

In practice, many test components, such as compressor units, feature target points that are not readily accessible. As shown in Figure 5-5, the regions around these target points are often partially obstructed by the component's structural features. Initially, the robot reaches the target point using a fixed orientation, as illustrated in Figure 5-4. While this fixed orientation may suffice for some components, in cases like the one depicted in Figure 5-5, the red-circled regions frequently obstruct the end effector. In such scenarios, it is necessary for the robot arm to rotate the end effector to an appropriate direction and angle to achieve proper access.

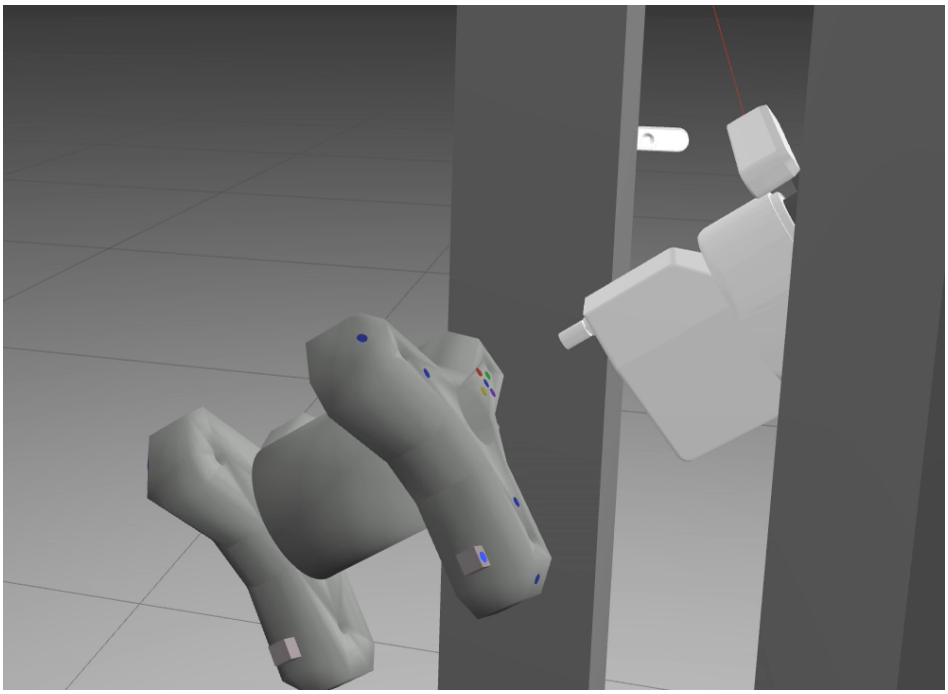


Figure 5-4: Fixed orientation of the local coordinate system before adaptive sampling

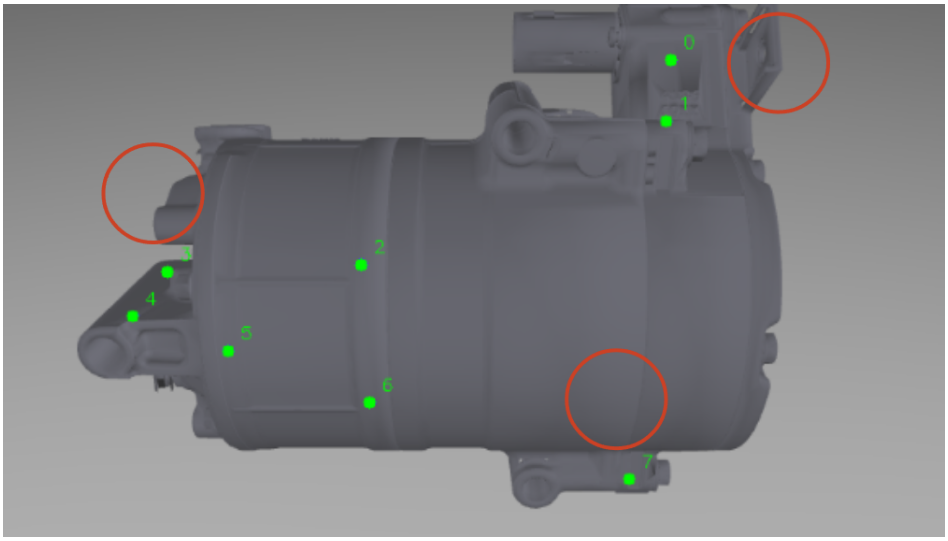


Figure 5-5: Compressor (test component) with 8 green target points

The adaptive sampling process evaluates multiple potential angles within a defined range to determine the most suitable orientation for the end effector. This evaluation enables the robot to navigate around obstacles and align the end effector parallel to the target surface, ensuring a more accurate and efficient approach.

This method significantly enhances the system's flexibility when handling real-world objects with complex geometries, particularly those that are inaccessible with fixed angles or paths.

By dynamically adjusting the end effector's orientation, the robot maximizes contact precision while minimizing collision risks, even in constrained environments.

An additional advantage of this approach is that the robot arm does not need to search for accessible angles in real-time, as is common with visual servoing techniques. By performing these calculations in advance, the system ensures safe motion without collision risk, thereby reducing the computational burden on the robot controller. Moreover, this method avoids complications associated with the robot arm's Jacobian matrix—such as singularities—that can arise during motion planning in visual servoing methods.

Adaptive Sampling Process

The adaptive sampling process is composed of three primary steps: coarse sampling, fine sampling, and optimal angle selection. This structured approach reduces computational time compared to exhaustively evaluating all angles at 1-degree increments from 0 to 360 degrees.

1. **Coarse Sampling:**
    - The angle is incremented in larger steps, for example, 30 degrees, to quickly identify a rough range of accessible angles.
    - This results in evaluating 12 different angle cases within the 0 to 360-degree range.
2. **Fine Sampling:**
    - Once a continuous range of accessible angles is identified from coarse sampling, the sampling resolution is increased, for instance, to 5-degree increments within the identified range.
    - This provides a more precise determination of accessible angles.
3. **Optimal Angle Selection:**
    - From the refined accessible range obtained from fine sampling, the widest continuous range is selected.
    - The midpoint of this widest range is chosen as the optimal angle for the end effector.

General Formulation of Adaptive Sampling

To formalize the adaptive sampling process, we define the following components:

- Let $\Theta = [0^\circ, 360^\circ)$ denote the full range of possible angles.
- Let $\Delta\theta_c$ be the coarse sampling step (e.g., 30°).
- Let $\Delta\theta_f$ be the fine sampling step (e.g., 5°).
- Define a function $C(\theta)$ that returns `true` if the end effector can access the target point at angle θ without collision, and `false` otherwise.

The adaptive sampling algorithm can be described as follows:

Algorithm 2 Adaptive Sampling for Accessible Angle

Require: Θ, $\Delta\theta_c$, $\Delta\theta_f$, $C(\theta)$
Ensure: θ_{opt}
Initialize an empty list of accessible angles A
Coarse Sampling:
for $\theta_c \in 0^\circ, \Delta\theta_c, 2\Delta\theta_c, \ldots, 360^\circ - \Delta\theta_c$ **do**
  if $C(\theta_c)$ **then**
    Add θ_c to A
  end if
end for
Identify continuous ranges in A and denote them as $R_1, R_2, \ldots, R_n$
Fine Sampling:
for each continuous range R_i in A **do**
  for $\theta_f \in R_i.\text{start}, R_i.\text{start} + \Delta\theta_f, \ldots, R_i.\text{end}$ **do**
    if $C(\theta_f)$ **then**
      Add θ_f to A_f
    end if
  end for
end for
Identify the widest continuous range in A_f and denote it as R_{widest}
Optimal Angle Selection:
$\theta_{\text{opt}} =$ midpoint of R_{widest}
return θ_{opt}

Alternatively, the process can be expressed mathematically as:

$$\theta_{\text{opt}} = \arg\max_{\theta \in \Theta} \left(\text{Accessibility}(\theta)\right)$$

where the accessibility function Accessibility(θ) evaluates the feasibility of the end effector reaching the target point at angle θ without collisions and aligns it optimally with the target surface.

Mathematical Representation

The adaptive sampling algorithm can be formalized using the following mathematical expressions:

1. **Coarse Sampling:**

$$A = \{\theta_c \in \Theta \mid \theta_c = k \cdot \Delta\theta_c,\ k \in \mathbb{N},\ C(\theta_c) = \texttt{true}\}$$

2. **Fine Sampling:**

For each continuous range $R_i = [\theta_{\text{start}}, \theta_{\text{end}}]$ in A,

$$A_f = \bigcup_{R_i} \{\theta_f \in R_i \mid \theta_f = \theta_{\text{start}} + m \cdot \Delta\theta_f,\ m \in \mathbb{N},\ C(\theta_f) = \texttt{true}\}$$

3. **Optimal Angle Selection:**

Identify the widest continuous range R_{widest} in A_f and compute the optimal angle as:

$$\theta_{\text{opt}} = \frac{\theta_{\text{start}} + \theta_{\text{end}}}{2}$$

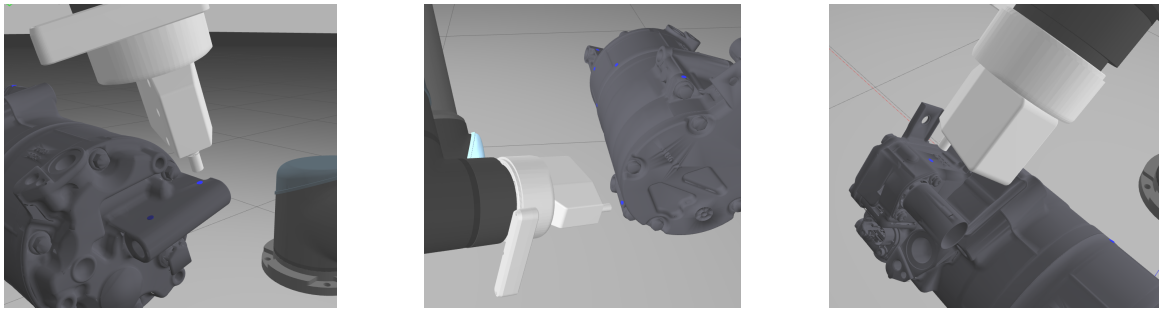
where $R_{\text{widest}} = [\theta_{\text{start}}, \theta_{\text{end}}]$.

Advantages of Adaptive Sampling

The adaptive sampling strategy offers several benefits:

- **Computational Efficiency:** By initially sampling at a coarse resolution, the algorithm quickly narrows down potential accessible regions, reducing the number of evaluations required during fine sampling.
- **Enhanced Accessibility:** Dynamically adjusting the end effector's orientation allows the robot to reach target points that are otherwise inaccessible with fixed angles, especially in environments with obstacles.
- **Collision Avoidance:** Pre-calculating accessible angles ensures that the robot arm moves safely without real-time collision checking, minimizing the risk of accidents.
- **Controller Load Reduction:** Offloading the angle determination process from the robot controller to the adaptive sampling algorithm decreases the computational burden on the controller, leading to smoother and more reliable robot operations.
- **Singularity Avoidance:** By not relying on the Jacobian matrix, the method avoids complications arising from singular configurations, which can hinder motion planning in visual servoing approaches.

Adaptive sampling results



(a) Target Point 1 **(b)** Target Point 2 **(c)** Target Point 3

Figure 5-6: Applying adaptive sampling to the target points which have potential obstruction.

When using a fixed orientation (see Figure 5-4), some target points may be obstructed. In particular, the red-circled areas on the compressor (Figure 5-5) prevent the robot from reaching certain target points. By employing adaptive sampling, the optimal orientation for the end effector's tip was successfully determined, enabling the robot to avoid the irregularly shaped obstructions on the compressor and accurately reach the target points (Figure 5-6).

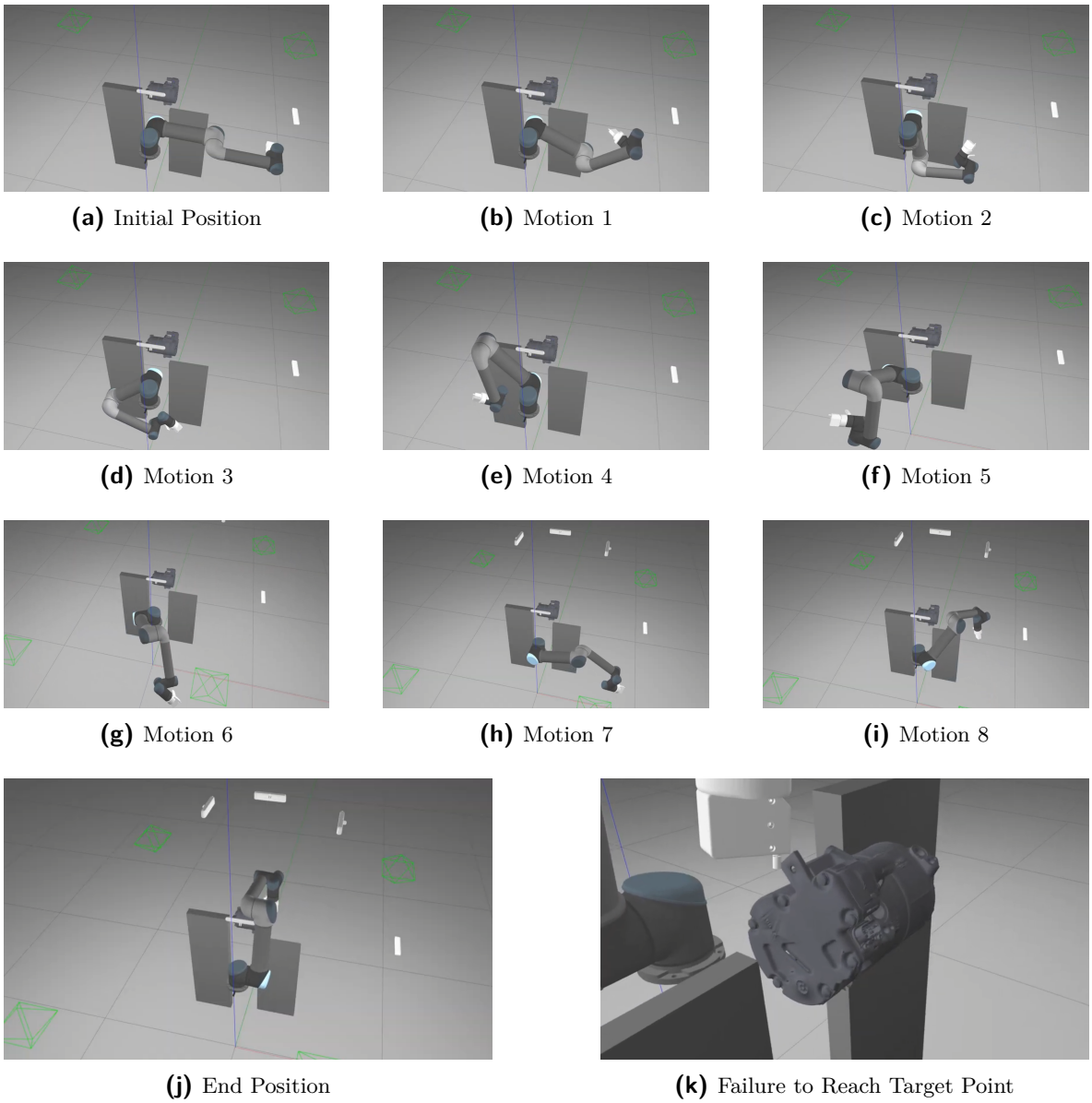
5-5 Simulation

5-5-1 Comparison of the Optimal Path with the Sampling-Based Path

It is true that sampling-based path planners are effective and easy to use. However, because they rely on random searches, the generated paths can vary even when starting and ending at the same positions. In obstacle-constrained environments, these randomly generated paths tend to be longer than those in obstacle-free environments [16]. Figure 5-7 illustrates a path generated by a sampling-based planner without any optimization—notice that the path is significantly longer than the optimal path obtained by the proposed method.

Moreover, the classic sampling-based planner often fails to reliably reach the target point. For example, in Figure 5-7k, the end-effector does not reach the target due to obstructions caused by the irregular shapes of the test component. In contrast, although the proposed method is also based on sampling-based planning, it produces a reasonably short path with a guaranteed success rate by accounting for the minimum distance from obstacles, as shown in Figure 5-8 and Figure 5-9.

To clarify, the target point is indicated as point 1 in Figure 5-5 for both the classic sampling-based path and the optimal path.



(a) Initial Position **(b)** Motion 1 **(c)** Motion 2

(d) Motion 3 **(e)** Motion 4 **(f)** Motion 5

(g) Motion 6 **(h)** Motion 7 **(i)** Motion 8

(j) End Position **(k)** Failure to Reach Target Point

Figure 5-7: Consecutive motion images showing the classic sampling- based path without optimization in obstacle-constrained environments.

In an obstacle-constrained environment, 441 candidate paths were generated using two popular sampling-based planners—PRM and RRT*—to mitigate bias from relying on a single method. As shown in Figure 5-10, these paths were evaluated using multi-objective optimization. Out of the 441 paths, 155 (represented by blue dots) were rejected due to constraint violations, leaving 281 feasible candidate paths (marked by green triangles). From these, five optimal paths—indicated by the yellow square—emerged as the undominated solutions on the Pareto frontier.

However, generating over 400 trajectories required approximately one hour of computation. To address this drawback, a reduced test was performed with around 30 trajectories, resulting in one optimal solution on the Pareto frontier. A comparison between the executed mo-

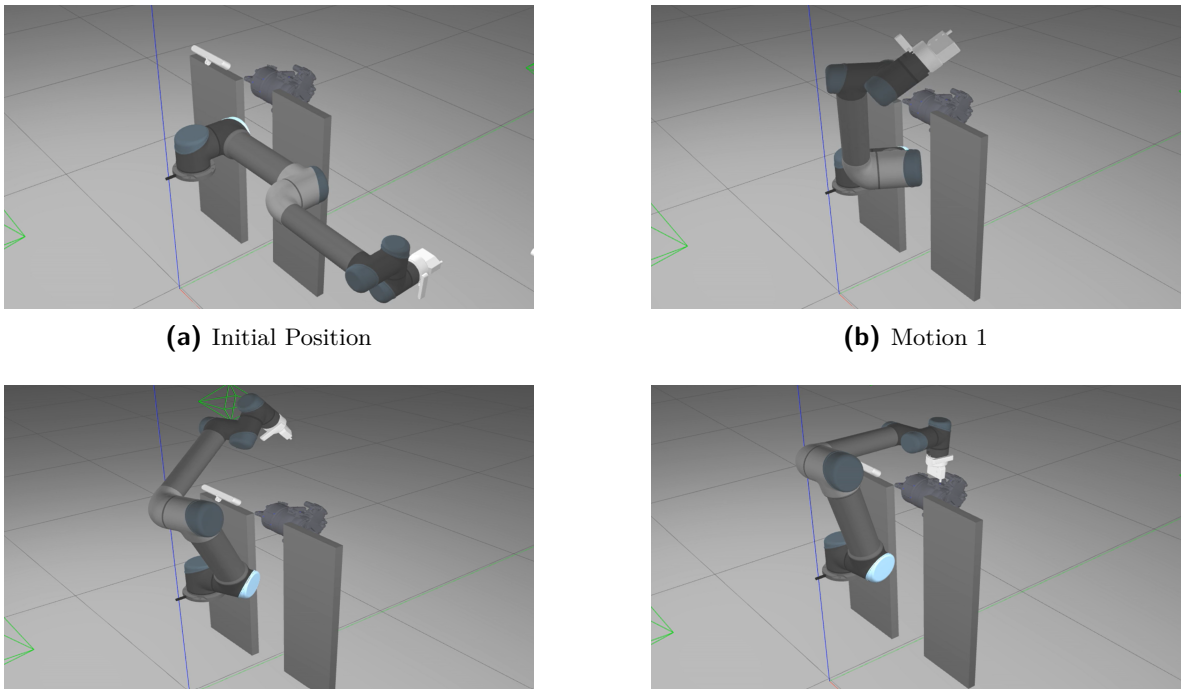
tions—441 trajectories as shown in Figure 5-8 versus 26 trajectories in Figure 5-9—indicates that the motion performance remains comparable, while the computation time is drastically reduced from one hour to five minutes.

The details of the Pareto solutions are as follows:

Case	Candidate	Minimum distance from obstacles	Joint Length	Smoothness
441 trajectories	1	0.002856	4.61036	5.92223e-31
	2	0.002856	4.34904	1.77763e-01
	3	0.002856	4.41885	1.14964e-01
	4	0.002856	4.48083	9.97054e-31
	5	0.002856	4.47990	1.71909e-30
26 trajectories	1	0.002856	3.90478	1.41286e-30

Table 5-1: Minimum distance from obstacles, Joint Length, and Smoothness Metrics

Figure 5-13, 5-12 further verify the Pareto solution by illustrating the relationships among three objectives. While this simulation in 26 trajectories case resulted in a single Pareto-optimal solution, in general, multiple Pareto solutions like 441 trajectories case can exist. In such cases, the weighted sum method, as shown in Equation 5-1, is used to select a single optimal solution from the set of Pareto solutions.



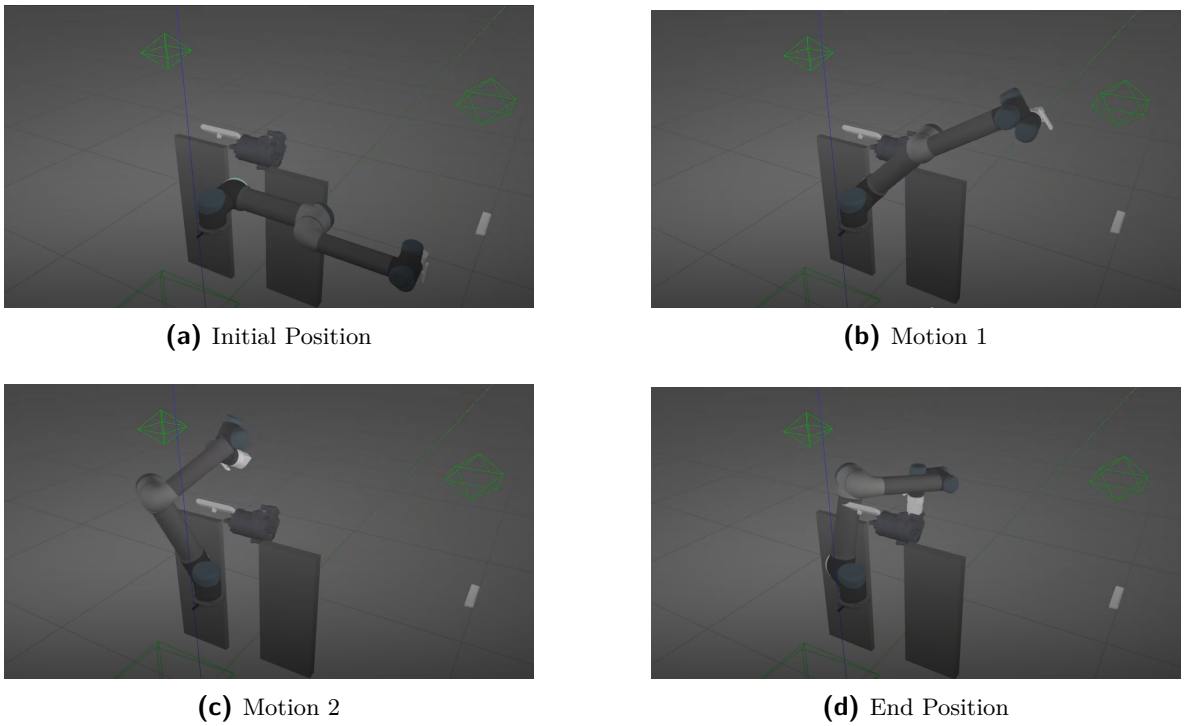
(a) Initial Position

(b) Motion 1

(c) Motion 2

(d) End Position

Figure 5-8: Consecutive motion images showing the optimal path from optimization in obstacle-constrained environments within 441 trajectories.



(a) Initial Position **(b)** Motion 1

(c) Motion 2 **(d)** End Position

Figure 5-9: Consecutive motion images showing the optimal path from optimization in obstacle-constrained environments within 26 trajectories.

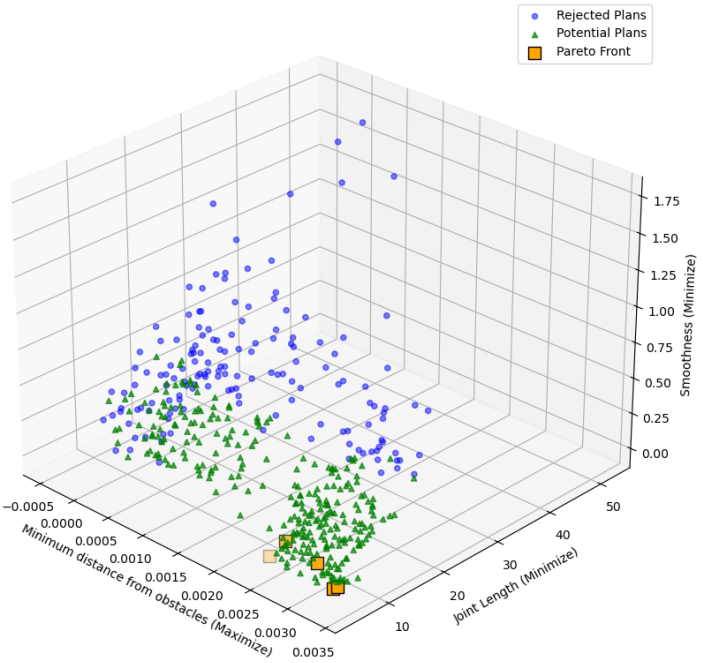


Figure 5-10: Relationships among three objectives in 3D with 441 trajectories

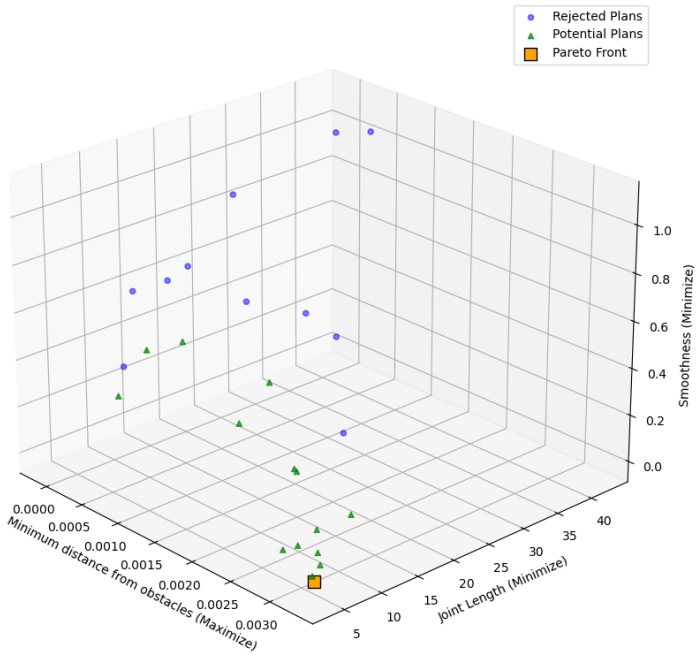


Figure 5-11: Relationships among three objectives in 3D with 26 trajectories

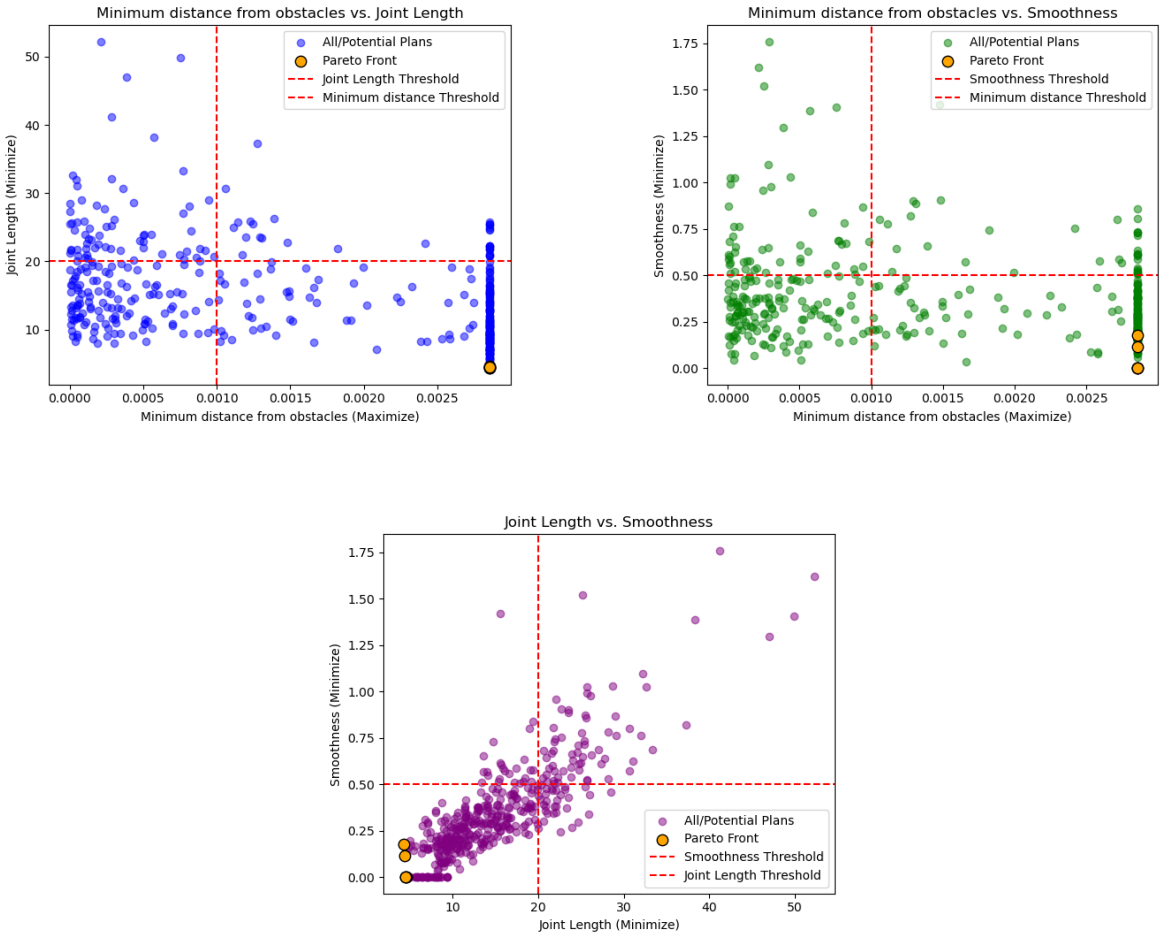


Figure 5-12: Relationships among three objectives in 2D

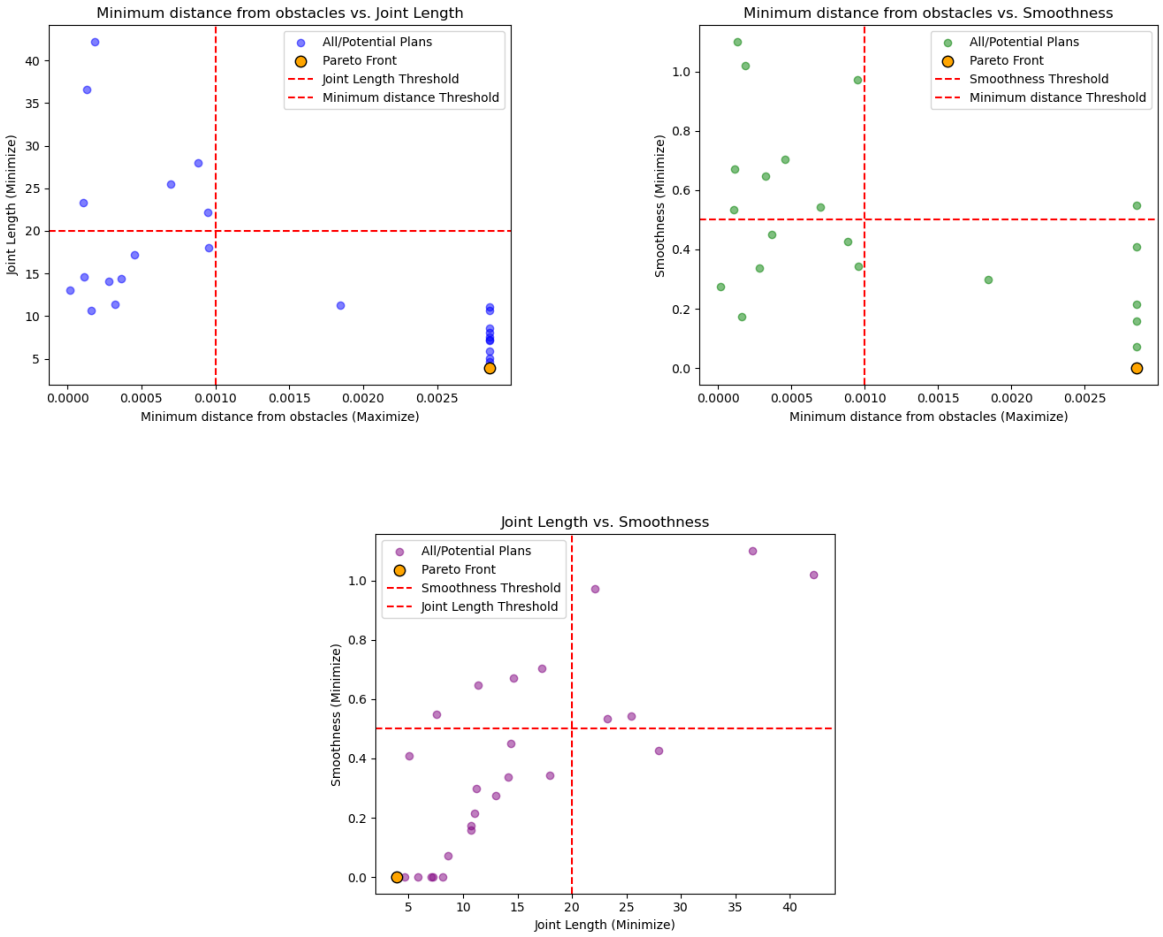


Figure 5-13: Relationships among three objectives in 2D

5-5-2 Evaluation

Traditional sampling-based path planners have limitations due to their varied search strategies. In the proposed method, over 20 candidate paths are generated and evaluated using multi-objective optimization, ensuring that the optimal path selected from the Pareto frontier satisfies three key objectives. This optimal path was successfully tested in the Gazebo simulator, which is recognized as one of the most reliable representations of real-world environments in robotics. Additionally, to efficiently generate the large number of candidate paths, a thread pool technique was employed. Without the thread pool, the planning process takes more than 40 minutes to generate around 30 trajectories and to execute; however, with the proposed method, the processing time is reduced to approximately 5 minutes—a significant improvement.

By leveraging multi-objective optimization rather than relying solely on a weighted sum selection process, several inherent shortcomings of the latter are overcome. First, by generating a Pareto frontier, the method explicitly captures the trade-offs between conflicting objectives such as minimum distance from obstacles, path length, and smoothness. This approach clearly

delineates the balance between these factors rather than compressing them into a single scalar value, which might obscure important nuances. Second, the weighted sum approach is highly sensitive to normalization and weight selection; improper scaling can lead to one objective dominating the others. The multi-objective framework, however, addresses each objective distinctly before combining them, ensuring a more balanced and representative evaluation of candidate trajectories.

Chapter 6

Conclusion

6-1 Contribution

The integration of accurate normal vector extraction with multi-objective path planning forms the cornerstone of the automated NVH testing approach presented in this thesis. By leveraging advanced 3D signal processing techniques—a foundational element in systems and control—for precise normal vector extraction, the system accurately captures surface orientations. These normals provide essential information about the geometry of target surfaces, allowing the robotic system to position the hammer precisely and maintain consistent perpendicular alignment during impacts. For example, by accurately determining a target point's orientation, the path planner—guided by multi-objective optimization, another core concept in systems and control—can devise routes that not only avoid collisions but also ensure that the end effector approaches the target from the optimal angle, thereby enhancing impact precision.

Moreover, the efficiency and robustness of normal vector extraction directly influence the responsiveness and adaptability of the path planning module. By reducing the computational load and improving the stability of normal estimation, the proposed method ensures that the path planner can operate in real time—even in environments with limited point cloud density or high sensor noise. This synergy between perception and planning results in a more cohesive and resilient automated NVH testing system, capable of handling complex, dynamic scenarios with enhanced precision and reliability.

This thesis makes the following key contributions:

- **Accurate Normal Vector Extraction:** A novel method is introduced that significantly reduces computation time and requires fewer data points for stable normal estimation, thereby enhancing the applicability of surface normal extraction in resource-constrained and dynamic NVH testing environments.
- **Multi-Objective Path Planning Framework:** A path planning strategy for selecting optimal path is developed that balances multiple objectives, ensuring both collision

avoidance and targeting accuracy for robotic arms operating in variable and obstacle-rich settings.

The significance of this work lies in its ability to bridge the gap between precise surface orientation detection and efficient, safe robotic motion planning. By addressing the limitations of existing methodologies and offering an integrated framework, this research paves the way for more intelligent, efficient, and adaptable automated NVH testing systems capable of performing complex tasks in a wide range of industrial environments.

6-2 Limitations

Accurate normal vector extraction in real-world experiments still faces several challenges that must be addressed to enhance overall system performance. One primary issue is the target point detection; many test components are metallic and exhibit strong light reflections, which significantly interfere with both target detection and depth measurement. In the simulations, the target point was assumed to be located at the center of the camera's field of view. However, during real-world experiments, multiple target points were detected, leading to ambiguities when calculating the angle difference between normal vectors. To improve accuracy, incorporating adaptive windowing strategies (e.g., an ellipsoidal window tailored for each target point) could help isolate the relevant point cloud data more effectively.

Additionally, current focus has been on processing point clouds rather than on optimizing detection techniques. As a result, the detection methods employed have not been fully refined. Another limitation is related to the robot arm's path optimization; the planned path derived from multi-object optimization has not yet been validated on an actual robot arm. Moreover, the current processing time of approximately 5 minutes is suboptimal for applications such as NVH testing, and would benefit from further reductions. Although leveraging better hardware resources can mitigate this issue to some extent, optimizing the algorithm for environments with limited computational power remains a critical challenge.

6-3 Future Work

Several avenues for future work have been identified to address the current limitations and further enhance the system's performance:

- **Enhanced Target Point Detection:**
  - Develop and integrate more robust target detection algorithms capable of handling adverse lighting conditions and reflective surfaces.
  - Investigate adaptive windowing techniques (e.g., ellipsoidal or dynamically sized regions of interest) for improved isolation of individual target points.
- **Multi-Target Point Processing:**
  - Extend the current approach to reliably handle multiple target points simultaneously, ensuring that the system accurately distinguishes and processes each target.

  - Explore advanced clustering or segmentation techniques to better manage overlapping or closely situated targets.

- **Robust Perception at Greater Distances:**
  - Enhance the system's ability to accurately extract normal vectors from sparse point clouds, particularly when the camera is positioned at a greater distance from the target.
  - Consider integrating multi-sensor data or depth enhancement methods to improve point cloud quality in low-density scenarios.

- **Real-Robot Validation:**
  - Validate the planned path and normal vector estimation techniques on an actual robot arm to assess real-world performance and identify practical constraints.
  - Collaborate with robotics experts to fine-tune the path optimization algorithms and ensure seamless integration with robotic control systems.

- **Processing Time Optimization:**
  - Explore algorithmic improvements and parallel processing techniques to significantly reduce the current processing time from approximately 5 minutes.
  - Investigate lightweight computational models and efficient coding practices to ensure that the system can operate effectively in resource-constrained environments.

- **Comprehensive System Integration:**
  - Work towards a fully integrated system that combines robust target detection, efficient point cloud processing, and validated robotic path optimization.
  - Conduct extensive field tests across diverse environments to ensure the system's robustness and adaptability in practical applications.

By pursuing these future work directions, it is aimed to overcome the current limitations and further develop a system that delivers reliable and efficient performance for both normal vector extraction and robotic path planning in complex real-world scenarios.

Appendix A

Robot arm model

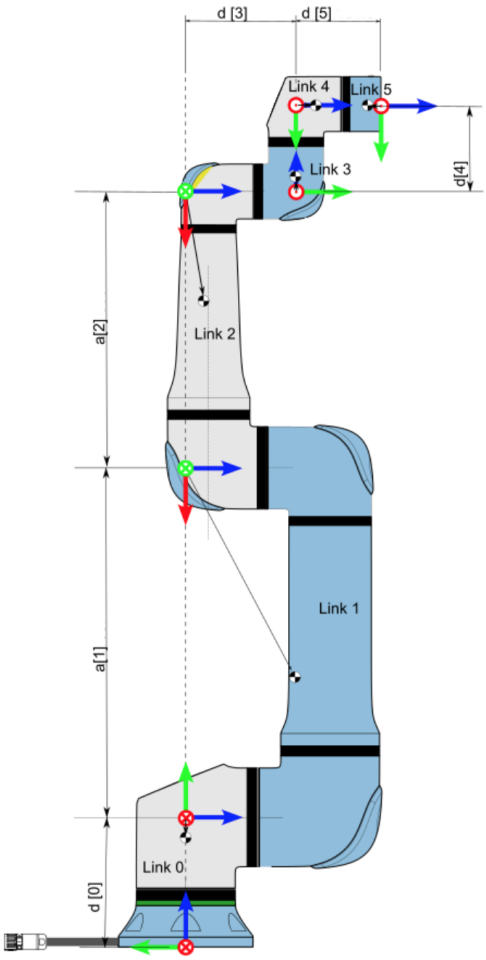


Figure A-1: Robot model : UR5

UR5							
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]	**Dynamics**	Mass [kg]	Center of Mass [m]
Joint 1	0	0	0.089159	π/2	Link 1	3.7	[0, -0.02561, 0.00193]
Joint 2	0	-0.425	0	0	Link 2	8.393	[0.2125, 0, 0.11336]
Joint 3	0	-0.39225	0	0	Link 3	2.33	[0.15, 0.0, 0.0265]
Joint 4	0	0	0.10915	π/2	Link 4	1.219	[0, -0.0018, 0.01634]
Joint 5	0	0	0.09465	-π/2	Link 5	1.219	[0, 0.0018,0.01634]
Joint 6	0	0	0.0823	0	Link 6	0.1879	[0, 0, -0.001159]

Figure A-2: Denavit–Hartenberg parameters (DH parameter) for UR5

Bibliography

[1] Hamid Afshari, Warren Hare, and Solomon Tesfamariam. Constrained multi-objective optimization algorithms: Review and comparison with application in reinforced concrete structures. *Applied Soft Computing*, 83:105631, 07 2019.

[2] Faez Ahmed and Kalyanmoy Deb. Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms. *Soft Computing*, 17:1283–1299, 2013.

[3] Min Sung Ahn, Hosik Chae, Donghun Noh, Hyunwoo Nam, and Dennis Hong. Analysis and noise modeling of the intel realsense d435 for mobile robots. pages 707–711, 06 2019.

[4] Amylia Ait Saadi, Assia Soukane, Yassine Meraihi, Asma Benmessaoud Gabis, Seyedali Mirjalili, and Amar Ramdane-Cherif. Uav path planning using optimization approaches: A survey. *Archives of Computational Methods in Engineering*, 29(6):4233–4284, 2022.

[5] H. Badino, D. Huber, Y. Park, and T. Kanade. Fast and accurate computation of surface normals from range images. In *2011 IEEE International Conference on Robotics and Automation*, pages 3084–3091, 2011.

[6] Gu Chaochen, Qi Feng, Changsheng Lu, Shuxin Zhao, and Rui Xu. Segmentation based 6d pose estimation using integrated shape pattern and rgb information. *Pattern Analysis and Applications*, 25:1–19, 06 2022.

[7] Liang-Chia Chen, Sheng-Hao Huang, and Bo-Han Huang. *Precise 6DOF Localization of Robot End Effectors Using 3D Vision and Registration without Referencing Targets.* 10 2022.

[8] Dongyang Cheng, Dang-Jun Zhao, Zhang Junchao, Caisheng Wei, and Di Tian. Pca-based denoising algorithm for outdoor lidar point cloud data. *Sensors*, 21:3703, 05 2021.

[9] Ignacy Dulęba and Michal Opalka. A comparison of jacobian-based methods of inverse kinematics for serial robot manipulators. *International Journal of Applied Mathematics and Computer Science*, 23, 06 2013.

[10] B. Fei, W. Yang, W.-M. Chen, Z. Li, Y. Li, T. Ma, X. Hu, and L. Ma. Comprehensive review of deep learning-based 3d point cloud completion processing and analysis. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):22862–22883, 2022.

[11] John Gan, Eimei Oyama, Eric Rosales, and Huosheng Hu. A complete analytical solution to the inverse kinematics of the pioneer 2 robotic arm. *Robotica*, 23:123–129, 01 2005.

[12] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. Path planning and trajectory planning algorithms: A general overview. *Motion and operation planning of robotic systems: Background and practical approaches*, pages 3–27, 2015.

[13] Kyoungmin Han, Kyujin Jung, Jaeho Yoon, and Minsik Lee. Point cloud resampling by simulating electric charges on metallic surfaces. *Sensors (Basel, Switzerland)*, 21, 11 2021.

[14] Ruoyu He, Yihang Li, Min Jia, Peng Zhang, and Bin Liu. A high accuracy extraction method of point cloud vectors for guiding robotic arm movement. pages 694–699, 07 2024.

[15] Jakob Jenkov. Thread pools in java, n.d. Accessed: 2025-01-28.

[16] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotic Research - IJRR*, 30:846–894, 06 2011.

[17] Soohyeon Kim, Sooahm Rhee, and Taejung Kim. Digital surface model interpolation based on 3d mesh models. *Remote Sensing*, 11:24, 12 2018.

[18] D. Lao, Y. Quan, F. Wang, and Y. Liu. Error modeling and parameter calibration method for industrial robots based on 6-dof position and orientation. *Applied Sciences*, 13(19):10901, 2023.

[19] W. Liu, J. Sun, W. Li, T. Hu, and P. Wang. Deep learning on point clouds and its application: A survey. *Sensors (Basel)*, 19(19):4188, Sep 2019.

[20] Ellips Masehian and Davoud Sedighizadeh. Multi-objective pso-and npso-based algorithms for robot path planning. *Advances in electrical and computer engineering*, 10(4):69–76, 2010.

[21] MathWorks. Inverse kinematics. https://nl.mathworks.com/discovery/inverse-kinematics.html, 2025. Accessed: 13 February 2025.

[22] Gang Mei. *RealModel-a system for modeling and visualizing sedimentary rocks*. PhD thesis, 07 2014.

[23] MoveIt! Developers. Developer concepts. https://moveit.ai/documentation/concepts/developer_concepts/. Accessed: March 4, 2025.

[24] Claudio Mura, Gregory Wyss, and Renato Pajarola. Robust normal estimation in unstructured 3d point clouds by selective normal space exploration. *The Visual Computer*, 34, 06 2018.

[25] Milad Nazarahari, Esmaeel Khanmirza, and Samira Doostie. Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm. *Expert Systems with Applications*, 115:106–120, 2019.

[26] Andreas Orthey, Constantinos Chamzas, and Lydia Kavraki. Sampling-based motion planning: A comparative review. *Annual Review of Control, Robotics, and Autonomous Systems*, 7, 11 2023.

[27] E. Pairet, C. Chamzas, Y. R. Petillot, and L. Kavraki. Path planning for manipulation using experience-driven random trees. *IEEE Robotics and Automation Letters*, 6(2):3295–3302, 2021.

[28] Alejandro Perez, Sertac Karaman, Alexander Shkolnik, Emilio Frazzoli, Seth Teller, and Matthew R Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 4307–4313. IEEE, 2011.

[29] Julia Sanchez, Florence Denis, David Coeurjolly, Florent Dupont, Laurent Trassoudaine, and Paul Checchin. Robust normal vector estimation in 3d point clouds through iterative principal component analysis. *ISPRS Journal of Photogrammetry and Remote Sensing*, 163:18–35, 05 2020.

[30] Huanyu Wu, Wei Zhang, Weisheng Lu, Junjie Chen, Jianqiu Bao, and Yongqi Liu. Automated part placement for precast concrete component manufacturing: An intelligent robotic system using target detection and path planning. *Journal of Computing in Civil Engineering*, 39(1):04024044, 2025.

[31] Mohd Nayab Zafar and JC Mohanta. Methodology for path planning and optimization of mobile robots: A review. *Procedia computer science*, 133:141–152, 2018.

[32] L. Zhang, H. Du, Z. Qin, et al. Real-time optimized inverse kinematics of redundant robots under inequality constraints. *Scientific Reports*, 14:29754, 2024.

[33] J. Zupančič and T. Bajd. Comparison of position repeatability of a human operator and an industrial manipulating robot. *Computers in Biology and Medicine*, 28(4):415–421, 1998.