Development and Validation of Automated Viewpoint Generation and Calibration Tools for Enhanced Robotic Surface Inspection in Aerospace Manufacturing

Arun Nandagopal

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2024

Reading Committee:

Xu Chen

Santosh Devasia

Krithika Manohar

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Development and Validation of Automated Viewpoint Generation and Calibration Tools for Enhanced Robotic Surface Inspection in Aerospace Manufacturing

Arun Nandagopal

Chair of the Supervisory Committee:

Quality control is key in the advanced manufacturing of complex parts. Modern aerospace manufacturing must identify and exclude parts with visual imperfections (e.g., scratches, discolorations, dents, tool marks, etc.) to ensure safe operation. This inspection process—often manual—is not only time-consuming, but burdensome, subjective, and requires months to years of training, particularly for high-volume production operations. A reliable robotic visual inspection solution has been hindered by the small defect size, intricate part characteristics, and demand for high inspection accuracy. This work proposes a novel framework for automated inspection path planning that addresses these core features through five innovations: camera-parameter-based mesh segmentation, ray-tracing viewpoint placement, robot-agnostic viewpoint planning, Bayesian Optimization (BO) for efficient segmentation and Region Growth (RG) algorithm for adaptability to highly complex parts. The development of a viewpoint generation tool is followed by the exploration and implementation of systematic camera calibration techniques. Finally, the effectiveness of the proposed workflow is tested with simulation and experimentation on a robotic inspection of heterogeneous complex geometries.

TABLE OF CONTENTS

Page

List of Figures iii

List of Tables v

Chapter 1: Introduction 1
1.1 Background and Motivation 1
1.2 Contribution of the Thesis 1

Chapter 2: Viewpoint Generation 6
2.1 Overall Workflow 6
2.2 Mesh Segmentation Algorithm 8
2.3 Optimal Search of Imageable Point Clusters With Bayesian Optimization . . 12
2.4 Automated Pointcloud Preprocessing for Segmenation 20
2.5 Occlusion Avoidance 24
2.6 Viewpoint Clustering and Traversal 26

Chapter 3: Robot Setup and Calibration 28
3.1 Robot Hardware Setup 28
3.2 Camera Calibration 29

Chapter 4: Results 36
4.1 Mesh Segmentation Algorithm for Different Topologies 36
4.2 Efficiency of Clustering 38
4.3 Bayesian Optimization to Find K for Different Planar Segments 39
4.4 Optimal K Value Search Method Time Comparison 40
4.5 Segmentation with Pointcloud Preprocessing 41
4.6 Imaging a Part 42

Chapter 5: Conclusion 45
5.1 Conclusion 45

5.2 Limitation and Future Work 45

Bibliography 47

Appendix A: Code 52

LIST OF FIGURES

Figure Number Page

1.1 The proposed workflow for precision visual surface inspection of complex parts: left - input parameters of the task, including high-mix low-volume complex object geometries, defect characteristics, and camera specifications; middle - viewpoint generation detailing the factors influencing the generation of robot poses for imaging; right - resulting intelligent robotic imaging system comprising a robotic arm equipped with cameras and LED lights for adaptive lighting. 2

2.1 Illustration showing the different stages of automated viewpoint generation . 7

2.2 Flowchart of the proposed segmentation algorithm 9

2.3 Segmentation of planar segment: a) an individual cluster with all points within the field of view; b) all the clusters over an example decagon shape. . 13

2.4 Different elements taken into consideration while calculating the cost function: a) the diameter calculated for FOV; b) an i^{th} cluster where the clustering does not satisfy the constraint; c) the bounding boxes drawn to calculate the cost. 14

2.5 Performance of two cost functions for different values of K in K means segmentation of a triangle-shaped planar patch. The bounding-box-based cost function is represented as cost function 1 and point-based cost function is represented as cost function 2 15

2.6 Region growth algorithm flowchart 23

2.7 Flowchart of the segmentation algorithm combined with region growth algorithm for pointcloud pre-processing 25

2.8 Illustration of occlusion avoidance feature 26

3.1 UR5e robot setup with mounting LED panel and camera at the end-effector . 28

3.2 System architecture of the robotics inspection system. 30

3.3 The illustration depicts the extrinsic calibration for a robot by imaging a calibration board at various locations. Figure source: [1] 32

3.4 The illustration depicts the test setup to quantify the offset. The green dot represents the center of the image and the center point found through the intersection of the lines show the known location on the calibrated test bed. . 33

3.5 Calibration performed at 1:3 magnification with partial view of the board. The left image shows the robot pose to image the board, center image shows intrinsic calibration and the right image shows extrinsic calibration. 34

4.1 Illustration of the estimated surrogate model after 6 observations against true cost (objective) function 40

4.2 Time comparison between Binary search and Bayesian Optimization for viewpoint generation of a sphere 42

4.3 Images of the hollow cylinder taken using the inspection plan generated from the proposed algorithm 44

LIST OF TABLES

Table Number Page

3.1 Calibration expirements 35

4.1 Segmentation efficiency on benchmark topology 37
4.2 Bayesian Optimization performance evaluation 40
4.3 Bayesian Optimization comparison with exponential search 41
4.4 Segmentation of complex manufacturing part topology 43

ACKNOWLEDGMENTS

I extend my sincere appreciation to the Mechanical Engineering Department, the Mechatronics, Automation, and Control Systems Lab, and the Advanced Robotics for Manufacturing (ARM) program, as well as General Electric (GE) Global Research, whose support was instrumental in realizing this project.

I am deeply grateful to Dr. Xu Chen for providing me with the opportunity to contribute to the inspection project and for his unwavering support throughout.

Special thanks are also due to my committee members, Dr. Santosh Devasia and Dr. Krithika Manohar, for their invaluable guidance.

I would like to acknowledge the significant contributions of my lab members: Colin Acton, for his major role in developing the graphical user interface and occlusion avoidance; Sangyoon Back, for his support during experimentation; and Jonas Beachy, for his contributions to the development of Bayesian Optimization for viewpoint generation.

Furthermore, I extend my thanks to Shatil Sinha, Matt Hockemeyer, and Anirban Sinha at GE Aerospace for their insights into manufacturing processes.

Lastly, I express my gratitude to all my lab colleagues at MACS Lab for their unwavering support throughout this endeavor.

DEDICATION

to my parents.

Nandagopal Sengoden & Indra Nandagopal.

Chapter 1

INTRODUCTION

1.1 Background and Motivation

In the aerospace industry, meticulous visual inspection is crucial in ensuring flight safety. Even minor flaws such as scratches and dents in critical components like stator vanes can have catastrophic consequences, including airflow imbalances, premature fatigue failure, and potential loss of life [2, 3].

As the demand for quality-assured aerospace components skyrockets, inspection becomes a bottleneck hindering industrial throughput. Manual inspection processes are not only costly and time-consuming, but also laborious, subjective, and reliant on extensive training [4]. In fact, human inspectors exhibit an accuracy rate of only around 75% in inspecting precision parts [5].

Several sub-millimeter defect classification methods [6, 7, 8] and robotic image-capturing systems exist, but high-level automation remains largely absent in the aerospace industry due to its high-mix, low-volume (HMLV) manufacturing nature [9]. This necessitates expert reprogramming of inspection systems for each new part geometry and camera, hindering automation.

1.2 Contribution of the Thesis

To reach rapid manufacturing and quality assurance in aerospace, an agile robotic inspection framework adaptable to diverse part specifications encompassing shape complexity, material variations, and defect detection criteria, is critical. Key features of such a framework include:

- complete surface coverage for diverse geometries,
- occlusion-free image capture,
- optimal number of imaging locations (viewpoints),

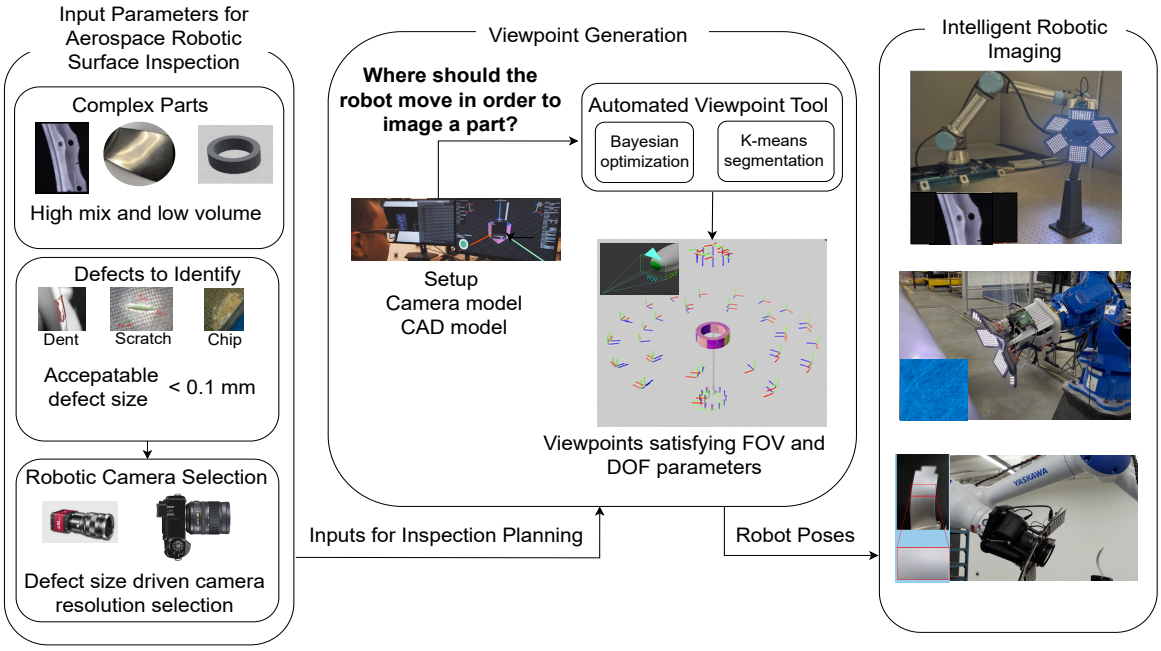


Figure 1.1: The proposed workflow for precision visual surface inspection of complex parts: left - input parameters of the task, including high-mix low-volume complex object geometries, defect characteristics, and camera specifications; middle - viewpoint generation detailing the factors influencing the generation of robot poses for imaging; right - resulting intelligent robotic imaging system comprising a robotic arm equipped with cameras and LED lights for adaptive lighting.

- in-focus image acquisition,
- generalizability to different robots, and
- efficient scalability to large parts.

This thesis proposes a novel framework for automated inspection path planning that addresses these core features through five innovations: camera-parameter-based mesh segmentation, ray-tracing viewpoint placement, robot-agnostic viewpoint planning, Bayesian Optimization (BO) for efficient segmentation and Region Growth (RG) algorithm for adaptability to highly complex parts (Figure 1.1). The thesis takes an additional step to ensure the gap between the image from viewpoint generated in simulation to the image

taken in real world settings for high-resolution cameras.

Considering input parameters including part geometries, defect characteristics, and camera specifications, we develop a viewpoint generation detailing the factors influencing the generation of robot poses for imaging. Then, the proposed intelligent robotic imaging system executes the generated viewpoints and paths with advanced lighting and camera controls. The adaptive lighting method developed to generate optimal and even illuminance on complex shapes by the authors in [10] is utilized for advanced lighting control Such a generic approach applies to different industrial robot platforms [11]. In this work, experimentation results are demonstrated using a Universal Robot UR5e collaborative robot manipulator equipped with a customized end-of-arm tool housing a 61MP DSLR camera and 384 independently controlled LEDs, enabling sharp, repeatable, and safe image capture of part surfaces.

The proposed framework capitalizes on three algorithmic contributions of the work. The *first* algorithmic contribution relates to the challenge of capturing a part's entire surface from optimal viewpoints in inspection. Manual selection of the viewpoints to collect data is time-consuming and imprecise. Existing automated approaches such as identification of flat surfaces using normal vector data of triangles present in CAD model (Sheng et al., 2009) [12] and geometric feature analysis (Mosbach et al., 2021) [13] do not address self-occlusion, meaning certain areas of the part cannot be imaged due to obstruction by its own surface. Reinforcement learning methods by Landgraf et al. [14] (2021) and Wang et al. (2023)[15] achieve promising results at the cost of an elevated computation cost. In this work, a novel, efficient, and flexible solution using unsupervised machine learning for viewpoint generation and 3D model segmentation is proposed. Surface-based mesh segmentation, which is increasingly gaining traction in inspection [16, 17], forms the basis of the approach. Established methods like hierarchical clustering and region growth [18, 19] face challenges due to their neglect of camera constraints during segmentation.This requires a specialized segmentation algorithm that incorporates camera constraints for quantitative assessment. In the work [20] the utilization of K-mean for the clustering based on Camera parameters is explored, but it is mostly applicable to simple convex shapes. The proposed methodology integrates unsupervised machine learning us-

ing K-means clustering and the camera intrinsic model in two steps to guide cluster size determination for concave parts found in manufacturing settings. This ensures an efficient formation of segments within the camera's Depth of Field (DOF) and Field of View (FOV), which is crucial for high-resolution imaging. Inspired by Kaljaca et al. (2020) [21], the proposed approach leverages surface point locations and normal vectors to generate viewpoints for inspection.

The *second* algorithmic contribution addresses the challenge of systematically determining the number of inspection segments. Methods such as Elbow, Gap statistic, Silhouette Coefficient, and Canopy [22, 23] aid in determining K in K-means clustering through iterative evaluation of various metrics. Recent advancements, such as the binning-based silhouette approach and U-K-means algorithm [24, 25], enhance search efficiency and eliminate initialization requirements. However, these techniques are tailored for general segmentation problems where the inter-cluster distance is the primary focus. This work investigates two methods, exponential search, and BO, to find the optimal K value for segmentation based on a quality evaluation index using camera model metrics. BO, in particular, allows for understanding the K-means clustering output and helps predict the right K value with minimal clustering attempts. We present optimal image segmentation with both methods and compare their time performance.

Third *third* algorithmic contribution addresses the challenge of applying the developed segmentation technique for highly complex manufacturing parts which have large number of concave surfaces. For example, parts with small edges, holes and high curvature variation regions. To deal with such parts, point cloud manipulation software like Meshlab could be utilized, but to utilize these software the operators need substantial training. Therefore we implement RG algorithm in python based on Point Cloud Library (PCL) [26] in C++. This algorithm helps in eliminating the high curvature variation region, using curvature and normal metric.

Collectively, the proposed automated inspection framework offers an intelligent algorithm for creating planar patches and generating viewpoints, considering camera constraints for diverse set of parts representative of the ones seen in manufacturing settings. This innovative approach streamlines the inspection process by minimizing human subjectivity, re-

ducing setup time for robotic inspections, and lowering computational costs compared to current methods.

The remainder of the work is organized as follows. Chapter 2 provides the methodology of inspection planning from the input of CAD models and camera models to the generation of robot poses. Following that, Chapter 3 explains about the system setup and camera calibration to mitage the simulation to real gap. Subsequently, Chapter 4 evaluates various geometries of aerospace-relevant parts based on segmentation efficiency metrics. Section 5 concludes the work and also discusses limits of performance and scope for future work.

Chapter 2
VIEWPOINT GENERATION

This chapter elaborates on the proposed framework for agile inspection. An overarching depiction of the entire workflow is provided, followed by detailed explanations of individual innovations such as camera-parameter-based mesh segmentation, BO for efficient segmentation, RG for automated pointcloud pre-processing, viewpoint generation with ray-tracing occlusion avoidance, and robot-agnostic viewpoint planning in subsequent sections. Finally, a metric is defined to evaluate the overall process on a set of benchmark parts.

2.1 Overall Workflow

The proposed process for calculating viewpoints for imaging a part is shown in Figure 2.1. In Step 1, the pipeline takes in a surface model in mesh file formats such as STL and OBJ, and a camera model specifying the camera's FOV and depth of field (DOF). In Step 2, two procedures are performed. The Pointcloud Resampling procedure transforms the STL model to a pointcloud via Poisson-disk sampling. Then, the algorithm described in Section 2.2 is employed to obtain segments of the generated pointcloud that fall within the camera's FOV and DOF, referred to as Camera-based K-means Segmentation in Figure 2.1. Here, resampling the surface of a mesh file is crucial for two main reasons. Firstly, it helps eliminate symmetry in points for symmetrical objects like spheres or cubes, which can lead to singular points causing issues with surface subdivision. Secondly, resampling is necessary due to the high density of triangular surfaces around vertices, which can affect segmentation by indicating differences in point density. Additionally, flat surfaces in STL file formats often have sparse points, requiring resampling to ensure a uniform density of points for an accurate representation of the CAD model's surface. In highly complex shapes having more number of concave surfaces the automatic pointcloud pre-

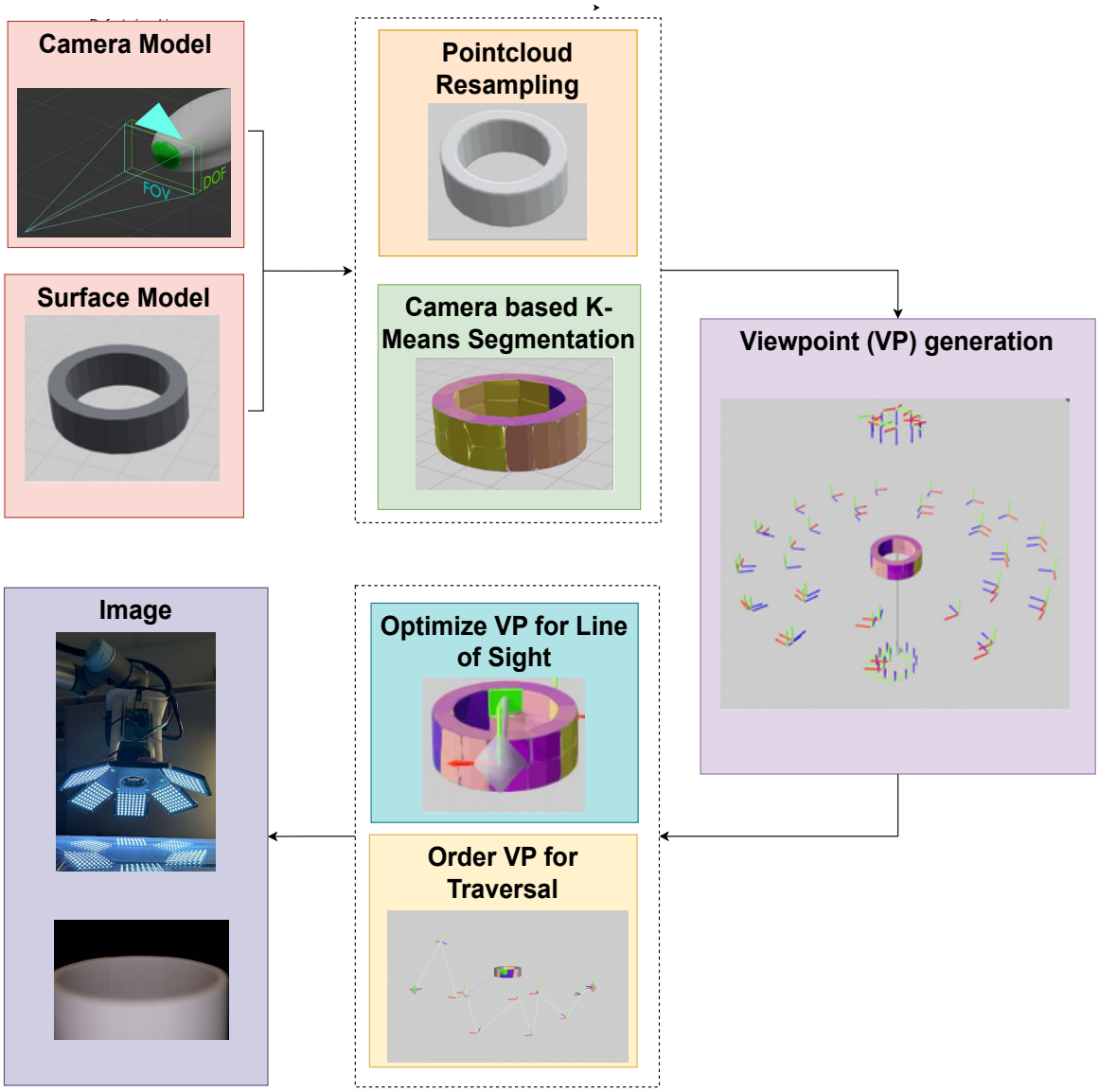


Figure 2.1: Illustration showing the different stages of automated viewpoint generation

processing as defined in Section 2.4 is done as a precursor to the given segmentation algorithm.

Once the part is divided into segments suitable for imaging, the local geometric properties of each point in the segment are utilized to calculate the position and orientation of the viewpoint in Step 3. This involves computing the average of the normal vectors for each point to obtain the mean normal vector. In addition, the spatial coordinates of each point are averaged to derive the center point for the segments. Subsequently, the camera position is determined by displacing the center point in the direction of the mean normal by a predefined offset determined by the camera model. The orientation of the principal axis of the camera is then calculated using the opposite direction of the mean normal vector.

Step 4 involves optimizing the viewpoint position and orientation to avoid occlusion by regions of the part itself, as explained in Section 2.5. The results of occlusion avoidance are shown in the final stage of Figure 2.1. After occlusion avoidance, these viewpoints are ordered and categorized as explained in Section 2.6 for inspection using a robotic imaging system.

Finally, the robot is commanded to move to these viewpoints to capture in-focus images covering the entire surface of the part.

2.2 Mesh Segmentation Algorithm

In this work the segmentation technique is built on Lloyd's algorithm (commonly referred to as K-means Clustering) and is expanded to integrate constraints on part geometry, exponential search, and BO. Additionally, we incorporate camera parameters including the FOV and DOF to ensure that the resulting clusters produce well-focused images. Figure 2.2 shows the proposed workflow, designed to consist of two segmentation stages. The initial stage involves segmenting the part into planar segments that fall within the camera's DOF. The subsequent stage consists of segmenting these planar segments into regions falling within the FOV.

To perform mesh segmentation using the K-means algorithm, a collection of data points and a method to find an optimal K value are essential. The data points are obtained from the geometric characteristics of the individual points within the point cloud produced

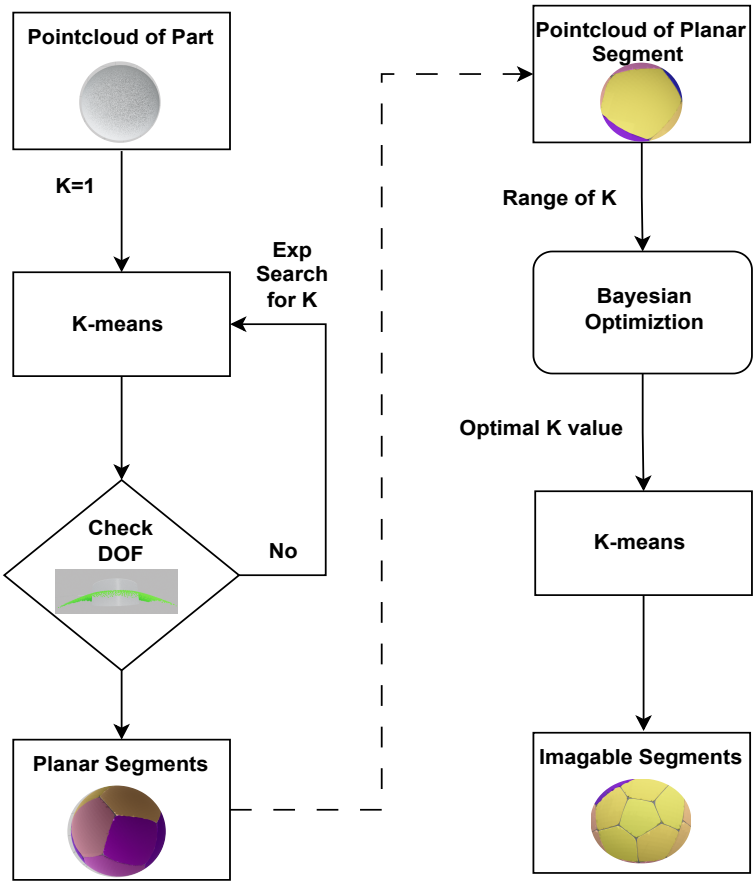


Figure 2.2: Flowchart of the proposed segmentation algorithm

during the resampling phase. Specifically, the data points or feature matrix for the algorithm are derived from the coordinates x_i, y_i, z_i, and the unit normals u_i', v_i', w_i' at each point in the point cloud.

Regarding the quantity of segments, determining K for any given point cloud segmentation is challenging. We propose exponential search when the bounds of K are unknown, and BO when such bounds are known.

To derive the planar segments, a feature matrix is constructed by normalizing the coordinates x_i, y_i, and z_i to x_i', y_i', and z_i' such that each coordinate falls within the closed interval [-1, 1], a process known as feature normalization [27]. Equation 2.1 shows the process

of transformation for the x coordinate. Similar procedure applies to other coordinates.

$$x'_i = -1 + 2\left(\frac{x_i - x_{\min}}{x_{\max} - x_{\min}}\right) \tag{2.1}$$

Subsequently, a vector $[x'_i\ y'_i,\ z'_i,\ u'_i,\ v'_i,\ w'_i]$ is constructed for each i^{th} point in the point cloud, and these vectors f_i are stacked to form a $p{\times}6$ feature matrix ($\mathbf{F_{DOF}}$) as shown in Equation 2.2, where p represents the total number of points:

$$\mathbf{F_{DOF}} = \begin{bmatrix} x'_1 & y'_1 & z'_1 & u'_1 & v'_1 & w'_1 \\ x'_2 & y'_2 & z'_2 & u'_2 & v'_2 & w'_2 \\ x'_3 & y'_3 & z'_3 & u'_3 & v'_3 & w'_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_p & y'_p & z'_p & u'_p & v'_p & w'_p \end{bmatrix} \tag{2.2}$$

The next step involves identifying planar segments by applying the K-means Clustering algorithm [28] to the feature matrix, resulting in clusters of points. Briefly, the algorithm applied to the feature matrix ($\mathbf{F_{DOF}}$) has the following process:

1. **Initialization of K Centroids:**

   $$\mathbf{C} = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$$

   where $\mathbf{c}_l$ represents the l-th centroid which is initialized by sampling from f_i .

2. **Cluster Assignment:** For each vector $\mathbf{f}_i$ in the feature matrix $\mathbf{F_{DOF}}$, assign it to the nearest centroid:

   $$\mathbf{r}_{ij} = \begin{cases} 1 & \text{if } j = \arg\min_l \|\mathbf{f}_i - \mathbf{c}_l\|^2 \\ 0 & \text{otherwise} \end{cases}$$

   where $\mathbf{r}_{ij}$ is an indicator variable representing the assignment of point $\mathbf{x}_i$ to cluster j.

3. **Centroid Update:** Update each centroid by computing the mean of all points assigned to that centroid:

   $$\mathbf{c}_j = \frac{\sum_{i=1}^{n} \mathbf{r}_{ij}\mathbf{f}_i}{\sum_{i=1}^{n} \mathbf{r}_{ij}}$$

   where $\mathbf{c}_j$ is the new centroid of cluster j.

4. **Convergence Check:** Repeat the steps 2 and 3 until the centroids no longer change significantly or a maximum number of iterations is reached.

At the end of this algorithm the points derived are assigned to each of these centroids $\mathbf{c}_j$. These cluster of points represents a collection of points that exhibit geometric similarity, i.e., they are closely positioned and share the same local normal.
In the case of planar segmentation, the bounds of K are uncertain. Therefore, exponential search [29] is preferred. It start with a conservative guess for K, often beginning with 1, and then iteratively adjusting it based on segment evaluation:

1. **Initialization:** Set the initial upper bound as $K = 1$.

2. **Exponential Range Expansion:** Double the upper bound until the segmentation at that value of K produces clusters which satisfy the evaluation function:

$$K = 2^i \quad \text{for } i = 0, 1, 2, \ldots$$

3. **Binary Search in the Found Range:** Perform a binary search in the subarray defined by the range $[K/2, K]$.

The principle of exponential search allows for a systematic exploration of a wide range of K values. Here, the evaluation function involves verifying whether each point lies within the depth of field of the camera. This involves utilizing the height of bounding box (h_i) of these points in cluster i and comparing with DOF ($h_i < DOF$). Through this iterative process, the optimal value of K is determined to segment the point cloud into planar segments (see Figure 2.2: left section).
The next step involves subdividing the planar regions into image-able segments (Figure 2.2: right section). This subdivision is achieved by employing the K-means clustering al-

gorithm on the feature matrix consisting of non-normalized coordinate points $[x_i\ y_i\ z_i]$:

$$\mathbf{F_{FOV}} = \begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \\ x_p & y_p & z_p \end{bmatrix} \tag{2.3}$$

This p by 3 matrix is used to obtain segments of points, where each segment represents a collection of points that are closest in terms of Euclidean distance only. The normal components of individual points are ignored in this feature matrix as the planar segments derived contain points with similar normals.

For the image-able region segmentation, a conservative guess of optimal K value's bounds can be determined. Therefore, BO is preferred, which will be detailed in Section 2.3. The optimal K value determined through this method ensures that all the points are within a radius equal to the least dimension of the camera FOV. As a result, every planar segment is subdivided into image-able sub-regions.

The output of the overall procedure is a set of image-able point clusters. The checks ensure that the cluster is within the bounds of the depth of field and that its surface area projected onto the imaging plane lies within the camera's FOV.

2.3 Optimal Search of Imageable Point Clusters With Bayesian Optimization

To efficiently segment a part into the optimal K number of partitions that satisfy the FOV and DOF requirements of the camera, one must minimize the number of evaluations of different K values. This is because it becomes extremely expensive to assess the FOV requirements at large K values. While the exponential search is on the modest time order of $\mathcal{O}[(\log n)]$, the real expense comes from generating the partitions using the K-means algorithm. At a high number of partitions, the computations needed to segment the part are quite intense, and an additional time load will arise from evaluating each partition against the FOV requirements. Thus, at large K values, many partitions, each containing many individual points, get evaluated for each iteration of K. The calculation time scales

quickly, making the segmentation of large parts extremely slow.

One way to minimize the number of calls to the K-means algorithm and subsequent FOV evaluations is to quantify how close the segmentation for each K-value is to an optimal segmentation, and then use this quantification to inform future K selections. This can be accomplished by a cost function that can be evaluated at all K values, designed such that the minimum value of the cost is at the optimal K. If the shape of the cost could be predicted from prior K-value evaluations of the cost function, then future evaluations of the cost function could be focused around the estimated minima. BO is well-suited to optimizing such costly to evaluate, black box cost function. Next, we elaborate the cost function design in Section 2.3.1, and then the setup of BO for segmentation is explained in Section 2.3.2.

2.3.1 Cost Function Selection

The optimal segmentation happens when all the points in a partition lie within the camera's FOV while at the same time having the least K value. Figure 2.3a shows an example of the FOV (the white circle) and the green represents the points. The goal of the cost function is to form a convex function with the minimum exactly at the minimum viable (optimal) K value. If the optimal K is at our cost function minimum, then BO can be used to reach the minimum while only sparsely sampling different K partitions.

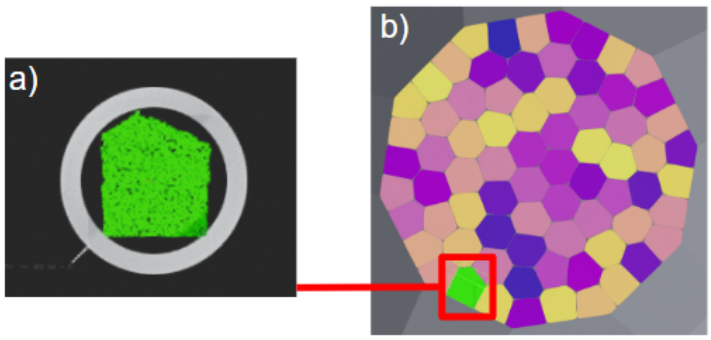


Figure 2.3: Segmentation of planar segment: a) an individual cluster with all points within the field of view; b) all the clusters over an example decagon shape.

Bounding-box-based cost function Consider first a simple-to-evaluate convex cost function using the concept of a bounding box (BB), where the constructed BB around the points provides the length and width of the area occupied by the points. For instance, consider a given FOV with diameter and area ($area_{FOV}$), as shown in Figure 2.4a. An i^{th} cluster is depicted in Figure 2.4b. The area of the green dotted rectangle BB in Figure 2.4c for the points that satisfy the FOV condition (green points) is referred to as $area(i)_{in}$. Additionally, the area of the black rectangle BB for all points (red and green points) in Figure 2.4c is referred to as $area(i)_{out}$. Integrating all the above, the proposed bounding-box cost function is:

$$\text{Cost} = \sum_{i=1}^{n} [area(i)_{out} + area_{FOV} - 2 \times area(i)_{in}]$$

which penalizes both the area of the region not satisfying the condition and the unoccupied region.

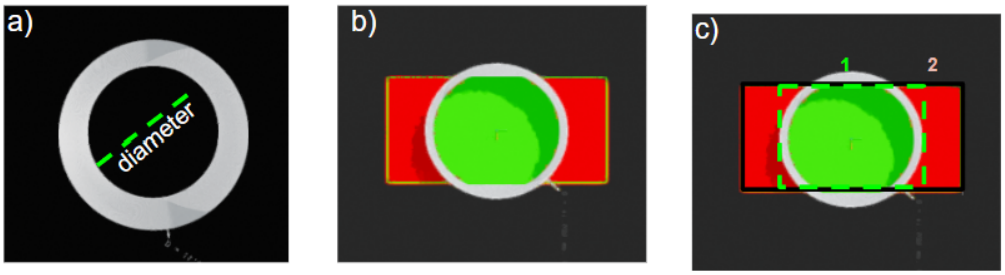


Figure 2.4: Different elements taken into consideration while calculating the cost function: a) the diameter calculated for FOV; b) an i^{th} cluster where the clustering does not satisfy the constraint; c) the bounding boxes drawn to calculate the cost.

This cost function above performed very well with certain shapes and provides a good starting performance. To evaluate the performance of the cost function, the cost function's minima and the percentage of points not meeting the FOV, referred to as the constraint violation (CV) are computed and analyzed. The cost function for segmentation was tested on common planar shapes with a ratio of the shape's area to the FOV's area

equal to 40. The FOV's shape was taken as a circle due to the camera placement requirement. The outcomes were that on planar shapes like hexagon and decagon the CV was 1% and 3% at the predicted minimum point, both fairly good. Whereas for a square and triangle the CV was 4% and 12% at the predicted minimum point, which was not acceptable. The variability is due to certain shapes producing highly angular segments which negatively affect our rectangular cost assignment's accuracy, as shown in Figure 2.5, Another problem analyzed with this cost function is that the minimum value for none of the shapes reached a CV of 0%, which was because the area of the unoccupied region in the FOV and area of the region outside the FOV were weighted equally.

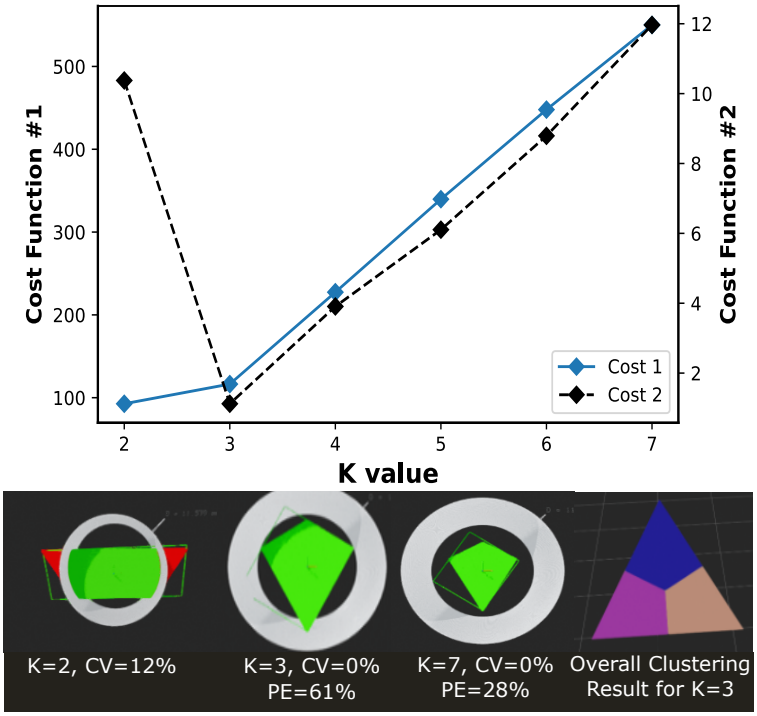


Figure 2.5: Performance of two cost functions for different values of K in K means segmentation of a triangle-shaped planar patch. The bounding-box-based cost function is represented as cost function 1 and point-based cost function is represented as cost function 2

Point-based cost function With the developed understanding of the relationship between shapes and clustering, we now design the full cost function $f(x)$ for implementation. Firstly, we enhanced the cost calculation for points outside the FOV by incorporating three parameters: the number of points inside the FOV for the i^{th} cluster ($\#points(i)_{in}$), the number of points not meeting the condition ($\#points(i)_{out}$), and n, representing the total number of clusters. The proposed constraint violation percentage (CV) is:

$$CV = \frac{1}{n}\sum_{i=1}^{n}\frac{\#points(i)_{out}}{\#points(i)_{in} + \#points_{out}} \tag{2.4}$$

The efficiency of points being packed inside FOV is derived from using $Max\#points_{in}$, the max number of points that can be fit within it (calculated using the point density). This term is defined as the packing efficiency (PE) and is calculated via:

$$PE = \frac{1}{n}\sum_{i=1}^{n}\frac{\#points(i)_{in}}{Max\#points_{in}} \tag{2.5}$$

The second design consideration is to change the equal weighting between CV and PE. This can give more importance to the constraint (CV) in a form such as:

$$f(K) = \lambda \times CV + \phi \times (\frac{1}{PE})^{\beta} \tag{2.6}$$

where λ, β, and ϕ are positive numbers.

To not have the packing efficiency influence the calculation until the constraint is met, the ϕ term can be set as 0, which makes the cost purely proportional to the constraint violation until the constraint is met. Once the constraint is satisfied, the CV term automatically becomes zero and the packing efficiency comes into effect. To initialize the first point which attains the CV as 0 % we check if any of the points of the cluster lies on the FOV limits and set a value of ϕ zero for that case alone:

$$\phi = \begin{cases} 0, & \text{if CV} \geq 0.001 \\ 1, & \text{if CV} < 0.001 \end{cases} \tag{2.7}$$

For the specification of circular FOV and polygonal planar shape, we identified λ equal to 100, and the β equal to 2 generated good results. For a triangle, this cost function produced 0% CV at the minimum value and also the highest packing efficiency when satis-

fying the constraint. Moreover, this cost function worked with similar accuracy on the square, hexagon, and decagon planar segments as well.

2.3.2 Bayesian optimization

With the developed cost function, we now apply BO (see, e.g., [30, 31]), a surrogate-model-based approach to find the optima of the cost $f(K)$. BO is particularly suitable here as our cost function does not have a closed-form solution and is expensive to evaluate. Briefly, BO is an iterative process, using samples of the cost function to calculate a surrogate model, and then selecting the next sample point as the location with the highest probability of finding a new minimum as given by an acquisition function. To perform a BO for the segmentation problem, the surrogate model and type of acquisition function need to be selected from a wide range of options [32]. Additionally, we need to determine parameters that affect the quality of the output, such as the width of the search space and the maximum number of iterations.

The surrogate model utilized for this problem was the Gaussian Process (GP). It was selected due to its popularity to represent non-linear relationships and its success with various use cases [30]. A GP, denoted as:

$$f^*(K) \sim \mathrm{GP}(\mu(K), Cov(K, K')) \tag{2.8}$$

is a function of distributions. For every input value K, it gives the mean and variance of a normal distribution. The input for the GP is prior mean $\mu(K)$ and covariance $Cov(K, K')$. For the segmentation problem, the prior mean at the start is taken as zero as the cost function doesn't have an analytic solution. Furthermore, the covariance function is taken as the standard squared exponential function as the designed cost function is smooth. The equation for the Covariance function for given two arbitrary K values (K_i, K_j) in search bounds and hyper-parameter ℓ is:

$$Cov(K_i, K_j) = \exp(-\frac{\|K_i - K_j\|^2}{2\ell^2}) \tag{2.9}$$

Here the hyper-parameter ℓ controls the width of the covariance function, which in turn determines the smoothness of the surrogate model. The ℓ value is tuned based on the

magnitude of the cost function search bounds to ensure appropriate smoothness.

As BO explores different values of K for the cost function, the GP surrogate model $f^*(K)$ is updated using Gaussian Process Regression (GPR). When a new K value K_n is sampled and the corresponding cost value $f(K_n)$ is found, the entire GP is updated by taking into account all the K values evaluated such as $\mathbf{K_{1:n}} = \{K_1, K_2, ..., K_n\}$ and corresponding cost function observations $\mathbf{f_{1:n}} = \{f(K_1), f(K_2), ..., f(K_n)\}$. The posterior mean and variance which defines the updated GP $f^*_{n+1}(K)$ for an arbitrary point K in search bounds are found using Equations 2.10 and 2.11, respectively:

$$\mu(\mathbf{K}|\mathbf{K_{1:n}}, \mathbf{f_{1:n}}) = \mathbf{Cov_*}^T \mathbf{Cov}^{-1} \mathbf{f_{1:n}} \tag{2.10}$$

$$\sigma^2(\mathbf{K}|\mathbf{K_{1:n}}, \mathbf{f_{1:n}}) = \mathbf{Cov}(\mathbf{K}, \mathbf{K}) - \mathbf{Cov_*}^T \mathbf{Cov}^{-1} \mathbf{Cov_*}^T \tag{2.11}$$

The $\mathbf{Cov_*} = \mathbf{Cov}(\mathbf{K}, \mathbf{K_{1:n}})$ is the covariance between the test point and the sampled points, and $\mathbf{Cov}$ is the covariance of the sampled points ($\mathbf{K_{1:n}}$). In practice, $\mathbf{K}$ is a vector of evenly spaced points across the whole domain such that the acquisition function has a GP surrogate of the entire cost function. For the derivation and further discussion, readers are referred to [33].

Because the cost function is expensive to evaluate, there is a need to carefully select the next observation point, K_{n+1}, such that the potential information gain is maximized. This is the purpose of an acquisition function, $\alpha(K)$. Crucially, $\alpha(K)$ is calculated using the GP $f^*_{n+1}(K)$. There are many acquisition functions, but they all balance the trade-off between exploration and exploitation. The exploration refers to evaluating areas with high uncertainty and the exploitation for evaluating areas near previously sampled low-cost values. In this work, the celebrated Expected Improvement (EI) [34] function is used. EI at an arbitrary K value is obtained by comparing the minimum value of all the cost functions evaluated in the previous n iterations $f(K_{min})$ and the GP evaluated at K value ($f^*_{n+1}(K)$). The formulation of the EI is:

$$\alpha_n^{EI}(K_i) = \mathbb{E}[\max\{0, f(K_{min}) - f^*_{n+1}(K))\}] \tag{2.12}$$

The EI accounts for both the likelihood that sampling at K_{n+1} will return a new minimum and the potential magnitude of that improvement in the surrogate model, making

it a balanced acquisition function and one well-suited for our needs. After evaluating, the next point is:

$$K_{n+1} = \text{argmax}_K(\alpha_n^{EI}(K_n)) \tag{2.13}$$

In the formulation of the GP, it is assumed that the cost function can be evaluated at any rational K value in the search bounds. However, the cost function for segmentation is only defined for integer values as the K-means algorithm only works for positive integer inputs. To overcome this, there are three primary methods of adapting the BO to a discrete space as identified in [32]. The naive approach simply rounds the maximum of the acquisition function to the nearest integer before evaluating the cost. This often leads to repeatedly sampling the same point depending on the size of the domain, though research has shown that careful selection of hyperparameters can limit resampling [35]. A second approach is to round inside the covariance matrix. This creates a step-function-like surrogate and acquisition function, which can be difficult to maximize depending on the method. The third method rounds only in the wrapper evaluating the cost (i.e., sample at an integer, but record observation with non-integer value from acquisition). This method is applied in our work. While a simple method, rounding inside the cost wrapper yields impressive results that regularly outperform other discrete surrogate methods [36]. Indeed, while not intended for discrete domains, BO has been applied to a wide variety of discrete situations with minimal effect on performance [37, 38]. The complete BO algorithm is shown by Algorithm 1.

The total number of iterations, N, can be set in a variety of ways based on time or computational resources available. In our case, N was determined based on the size of the cost function search bounds.

For the bounds of K, the start point is set as K_{min} as shown in Equation 2.14:

$$K_{\text{min}} = \frac{\text{Total area of planar region}}{\text{Area of the FOV}} \tag{2.14}$$

The rationale behind selecting this K_{min} value is that it is impossible to have the value of K for the planar segment subdivision less than this number. For the upper bound the K value is set as 3 times the K_{min} value. Some exceptions to the utilized bounds were

Algorithm 1 BO for N total cost function evaluations

1: Take n_0 initial samples of $f(K)$
2: Calculate $f_n^*(K)$ using initial samples
3: Set $n = n_0$
4: **while** $n \leq N$ **do**
5: Compute EI acquisition function, $\alpha_n^{EI}(K)$ using $f_n^*(K)$
6: Determine K value using $\text{argmax}_K(\alpha_n^{EI}(K))$
7: Round K value to nearest integer $K_{n_{round}}$ and evaluate $f(K_{n_{round}})$
8: Store K value and $f(K_{n_{round}})$ to all evaluated samples
9: With GPR, calculate $f_{n+1}^*(K)$
10: Set $n = n + 1$
11: **end while**
12: return location of minimum value of all cost function evaluation.

when the "length to width" ratio of the bounding box of the shape to be segmented was extremely large.

2.3.3 Usage

The derived cost function is utilized as the function to be minimized using the BO function [39] to find the minimum point location. The bounds for the K values are passed to the BO process and the optimal value of K is derived.

2.4 Automated Pointcloud Preprocessing for Segmenation

For complex shapes which are highly concave, i.e. those shapes which have a large number of corners, holes for fasteners and thin edges with high length to width ratio, the K-means segmentation doesn't do well at planar patch subdivision stage. This is due to the fact that the features which represent these points are sparse and they turn to be a inhibitor for individual clusters to form in those areas. Therefore for the segmentation algorithm discussed in Section 2.2 to work for shapes with feature of the aforementioned

properties, we utilize tool like Meshlab [40] to perform pointcloud pre-processing. This stage removes the points representing these features to allow for smooth segmentation. As this step is manual and requires technical expertise, a more automated and robust process for doing the same was explored. Initially a RANSAC algorithm was explored for planar patch formation, it was not suitable as it was not well extendable to different shapes. Therefore RG algorithm was explored due to its capability to identify features using the normal and curvature values and extend quite well to most of the shapes. This leads to utilization of region growth to replace the manual pointcloud pre-processing.

2.4.1 Region growth

To utilize RG for a given pointcloud the normal and curvature values for each point must be determined. The normal vector at a point in a point cloud is typically estimated using the covariance matrix of the neighborhood of the point. Given a point cloud $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\} \subset \mathbb{R}^3$, and a query point $\mathbf{p}_i$, we first identify the m-nearest neighbors of $\mathbf{p}_i$. Let $\{\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \ldots, \mathbf{p}_{i_m}\}$ be the set of m-nearest neighbors. The covariance matrix C is given by:

$$C = \frac{1}{m} \sum_{j=1}^{m} (\mathbf{p}_{i_j} - \bar{\mathbf{p}})(\mathbf{p}_{i_j} - \bar{\mathbf{p}})^T \tag{2.15}$$

where $\bar{\mathbf{p}}$ is the centroid of the neighborhood, calculated as:

$$\bar{\mathbf{p}} = \frac{1}{m} \sum_{j=1}^{m} \mathbf{p}_{i_j} \tag{2.16}$$

The eigenvalues and eigenvectors of the covariance matrix C are computed. As the covariance matrix C is a 3×3 matrix, we derive 3 eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$ and eigenvectors (n_1, n_2, n_3) arranged in ascending order. The normal vector $\mathbf{n}$ at a point $\mathbf{p}_i$ is the eigenvector (n_1) corresponding to the smallest eigenvalue (λ_1).

$$C\mathbf{n} = \lambda\mathbf{n} \tag{2.17}$$

To estimate curvature we utilize the eigenvalues of the covariance matrix. The curvature κ at point $\mathbf{p}_i$ can be estimated as:

$$\kappa = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \tag{2.18}$$

The curvature provides a measure of how much a surface deviates from being flat in the vicinity of a point. This is an important metric to determine features in a 3d geometry. Following the normal and curvature values calculation for each points, the region growth algorithm is initiated. The algorithm begins by selecting a seed point as shown in Step 2 of Figure 2.6. Here the point with least curvature is taken as the seed point. Based on the nearest neighbor search parameter the points are identified and evaluated for normal variation. Utilizing the specified angle threshold between their normal vector, a point is added to the region containing the seed point as shown in Step 4 of Figure 2.6. After which the added point is assessed against a curvature threshold, and if it meets the criteria, it becomes a new seed point as shown in Step 6 of Figure 2.6. This process repeats, continually growing regions from these seed points. If no new seed points are generated, the algorithm includes all points that have satisfied the normal threshold in a region as shown in Step 8 of Figure 2.6, and then finds a new seed point to initiate growth for the next region. This continues until all points in the pointcloud are added to regions which are generated as depicted in Step 9 of Figure 2.6

The regions consisting of larger set of points are relatively planar surfaces. For example, when region growth is applied to a pointcloud representing a sphere, all the points are fall into one region as the angle variation between the neighboring points are relatively low and the curvature is within the threshold. The regions which are consisting lesser number of points are those with higher normal variation. These regions usually represent edges, holes, and corners. Therefore by applying a minimum number of points threshold for a cluster we can identify and remove those regions which present difficulty to the segmentation algorithm.

2.4.2 Usecase

In order to utilize region growth for the mesh segmentation certain parameters need to be set. Based on the constraint of forming regions at all locations except at small edges, holes, and corners the parameters are determined. We set the normal threshold to 4 degrees, the curvature threshold to a value equal to the 98^{th} percentile of the curvature

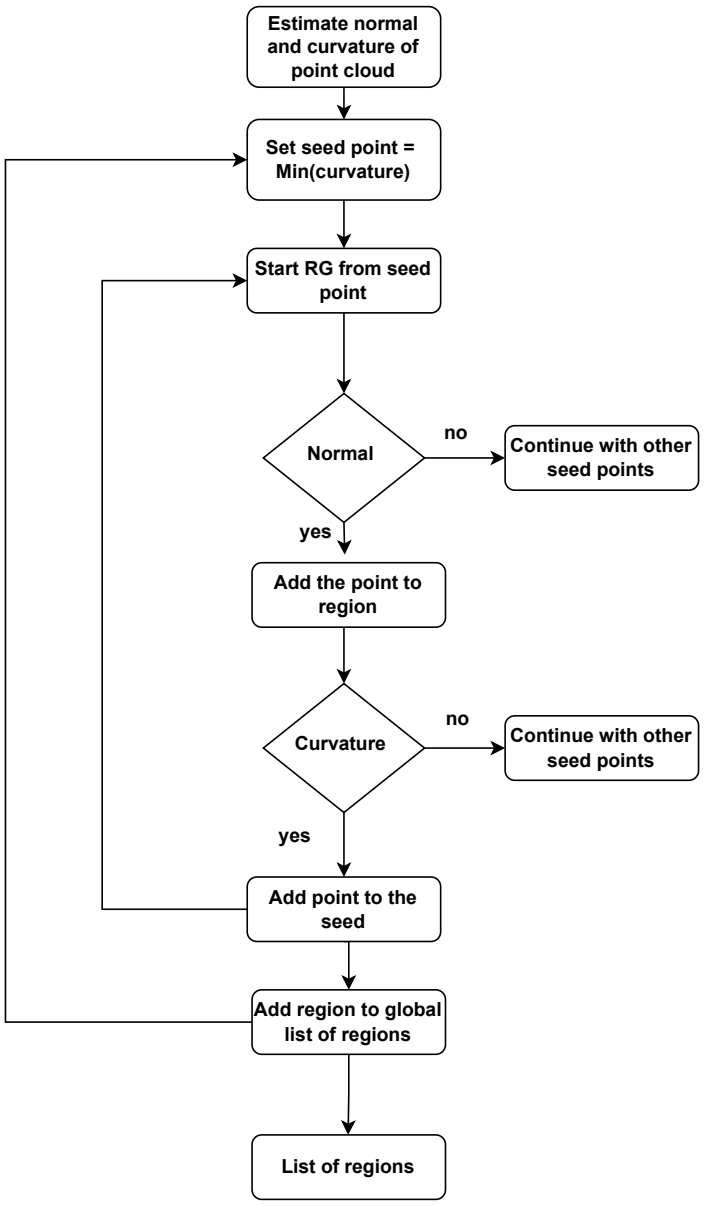


Figure 2.6: Region growth algorithm flowchart

value found for all points, and the number of neighboring points to 30. We also specify the minimum number of points to consider as a region as 5 percent of the total number of points. All the aforementioned parameters help get regions which make the segmentation work for deriving the viewpoints.

Once the regions are derived from the region growth, each of the region goes through the segmentation algorithm defined in the Section 2.2. This is done because it helps us to generate segments which satisfy the DOF and FOV condition even in instances when the region derived as the output is a pointcloud representing the entire surface of the sphere. Also the individual regions are passed to make K-means segmentation algorithm always converges. The updated workflow for complex shapes is depicted in the Figure 2.7.

2.5 Occlusion Avoidance

As mentioned in Section 2.1, a viewpoint is created for each cluster of points across the surface by projecting a point from the center of the cluster by the focal distance of the camera in the direction of the average normal of the set of points. For convex surfaces, the resulting viewpoint will properly capture the desired surface area within the camera's FOV and DOF; however, projecting viewpoints from concave surfaces can often lead to perspectives occluded by other facets of the part. To bypass these occlusions, a novel method of viewpoint adjustment using ray casting was developed.

Each projected viewpoint is tested for occlusion by casting a ray from the viewpoint to each surface point in its corresponding cluster. If the distance traveled by any of the rays does not match the distance from the viewpoint to the surface point within a certain tolerance ϵ, the viewpoint is considered to be occluded, and a new viewpoint must be selected.

As shown in Figure 2.8, the origin of the cluster o_c is considered as a pivot point for the original viewpoint, v. New candidate viewpoints are progressively generated and tested within the spherical cap defined by angle $\theta_{\max}$ from the z-axis of o_c. To optimize computational efficiency and bias the choice of optimal viewpoint $v*$ towards v, the search domain is swept by progressively increasing θ from the apex of the cap and testing points sampled uniformly along and in proportion to the circumference of the circle to maintain

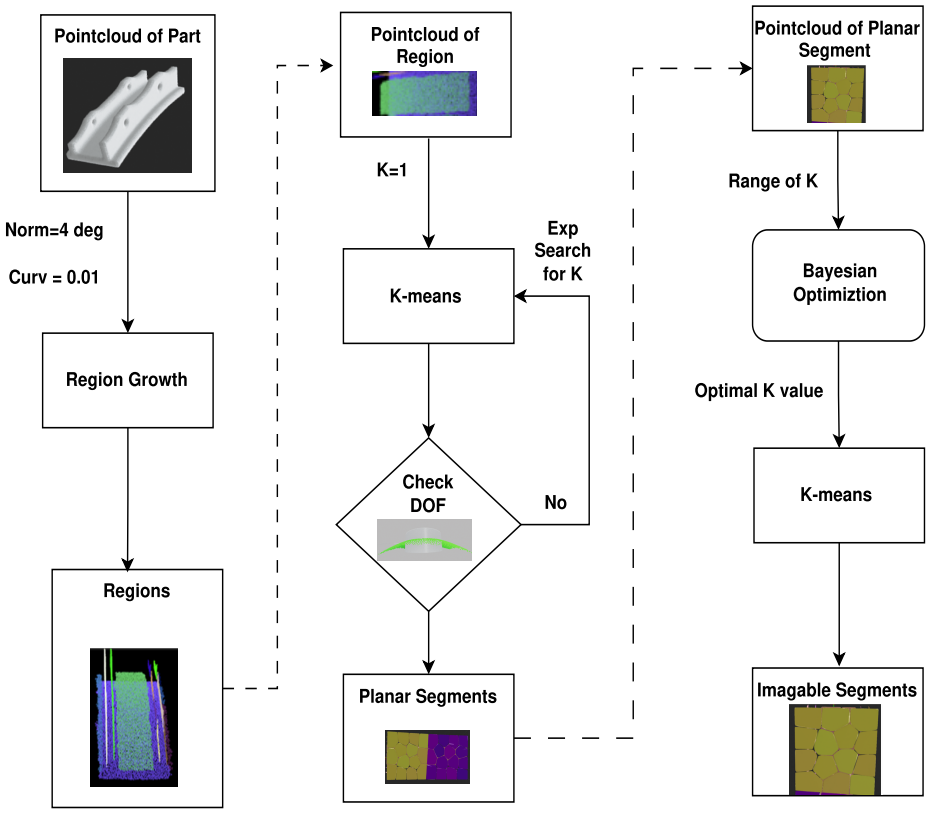


Figure 2.7: Flowchart of the segmentation algorithm combined with region growth algorithm for pointcloud pre-processing

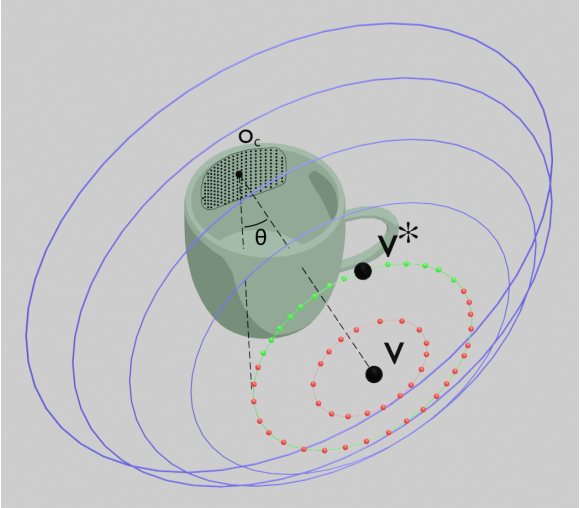


Figure 2.8: Illustration of occlusion avoidance feature

a uniform distribution of points across the search domain. Once a search of the circle defined by theta results in a set of viable viewpoints, $v*$ is selected from the group to minimize the variance of distances traveled by the rays cast. This ensures that all points in the cluster are as equidistant to the camera as possible, maximizing the likelihood of a well-focused image.

2.6 Viewpoint Clustering and Traversal

The viewpoints generated for any given part can be categorized into six regions: top, bottom, left, right, front, and back. However, challenges may arise with larger parts, particularly in accessing certain regions, such as the bottom, especially when the part is positioned on a fixture. In such cases, accessing the viewpoints in the bottom region becomes impractical. To overcome this limitation and ensure comprehensive coverage of all regions, a solution involves implementing a rotatable fixture. This fixture allows inspection of one region at a time, for example, imaging the front region initially, followed by rotating the fixture to enable sequential imaging of other facets.

To efficiently facilitate the automatic clustering of viewpoints, this work utilizes another layer of K-means clustering. With the precise number of regions known (i.e., six regions),

the value for K is specified as 6. The input data points for the K-means algorithm comprise the position and normal vectors of the generated viewpoints for the part. This approach enables effective clustering of viewpoints based on their respective regions. Moreover, if interest lies solely in imaging four regions, the value of K can be adjusted accordingly to 4. This methodology enhances the adaptability of imaging these viewpoints without constraints on the robot being used.

For traversing the viewpoints within a particular region, a systematic approach is adopted. The points are sorted from the bottom to the top, ensuring a structured progression. In cases where points possess similar heights from the bottom, the leftmost point is prioritized. This selection criterion facilitates smoother transitions for the robot as it moves from one viewpoint to another within the region. By prioritizing the leftmost point in such scenarios, the traversal process becomes more organized and streamlined, contributing to efficient robotic inspection planning.

Chapter 3

ROBOT SETUP AND CALIBRATION

This chapter talks about the robotic software and hardware setup done to validate the viewpoint generation tool developed in Chapter 2. At first, Section 3.1 explains the system setup. In Section 3.2.1 the details of the camera calibration is explained.

3.1 Robot Hardware Setup

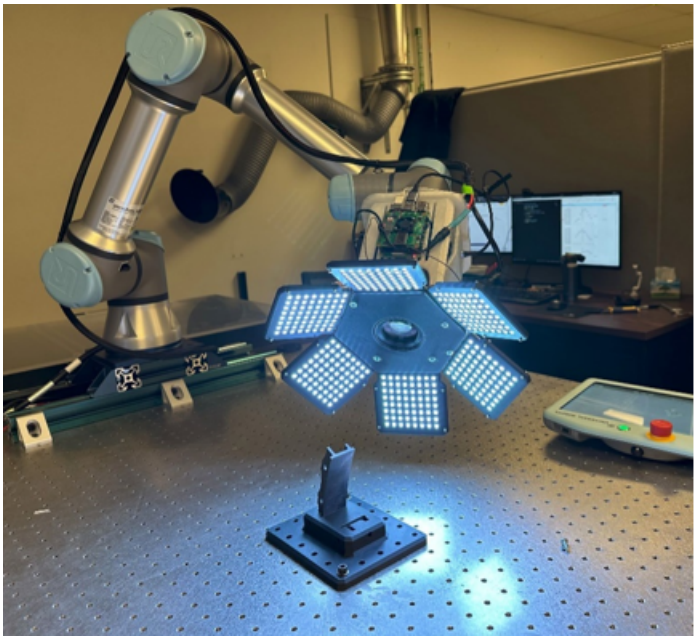


Figure 3.1: UR5e robot setup with mounting LED panel and camera at the end-effector

The developed robotic system employs an End-of-Arm Tool (EOAT) equipped with a 61MP Sony mirrorless camera paired with a macro lens, chosen for its ability to capture intricate details necessary for detecting small defects as tiny as 0.1 mm in aerospace components. Surrounding the camera are 384 individually addressable LEDs that provide pre-

cise lighting control, ensuring optimal illumination to highlight defects and produce high-quality, clear images. This lighting system features adaptive lighting technology developed by Gerges et al. [10].

The EOAT hardware is controlled using a Raspberry Pi 4 Model B with a Power over Ethernet (PoE) Hat, which simplifies the system by integrating power and network connectivity. Users can easily connect the Raspberry Pi to the network and utilize the provided API calls, showcasing an innovative and readily adoptable inspection technology.

The system's core functionality relies on the Robot Operating System (ROS), which is crucial for implementing the calculated viewpoints discussed in Chapter 2. As depicted in Figure 3.2, various ROS nodes, each assigned specific tasks such as camera control, lighting, and robot manipulation, are activated to initiate the inspection process.

The inspection process begins with the viewpoint generation application. The generated viewpoint poses are passed to the robot service node as a pose array. The robot service node then uses the Moveit Python API to generate the joint trajectory, which is sent to the UR5e controllers via a service-type ROS message. Upon reaching the requested pose, the robot service node receives an update indicating the success of the movement.

Once the robot reaches the pose, the robot service node sends the camera and lighting settings to the inspection action planning node using ROS service and action messages. This inspection planning node manages both the imaging system and the defect detection system. It sends a message to the camera control node to adjust the lighting and capture the image, which is then stored on the computer. Simultaneously, it coordinates the defect detection process using the stored images.

Overall, this is a high-level representation of all the nodes working together to perform the inspection process.

3.2 Camera Calibration

When a camera is affixed to a robotic arm, the precise location of its optical center becomes paramount. This is crucial because, to capture an image of a part from a specific viewpoint, it's imperative to align the camera's optical center with that generated viewpoint. To determine the optical center of the camera relative to the robot's end effector

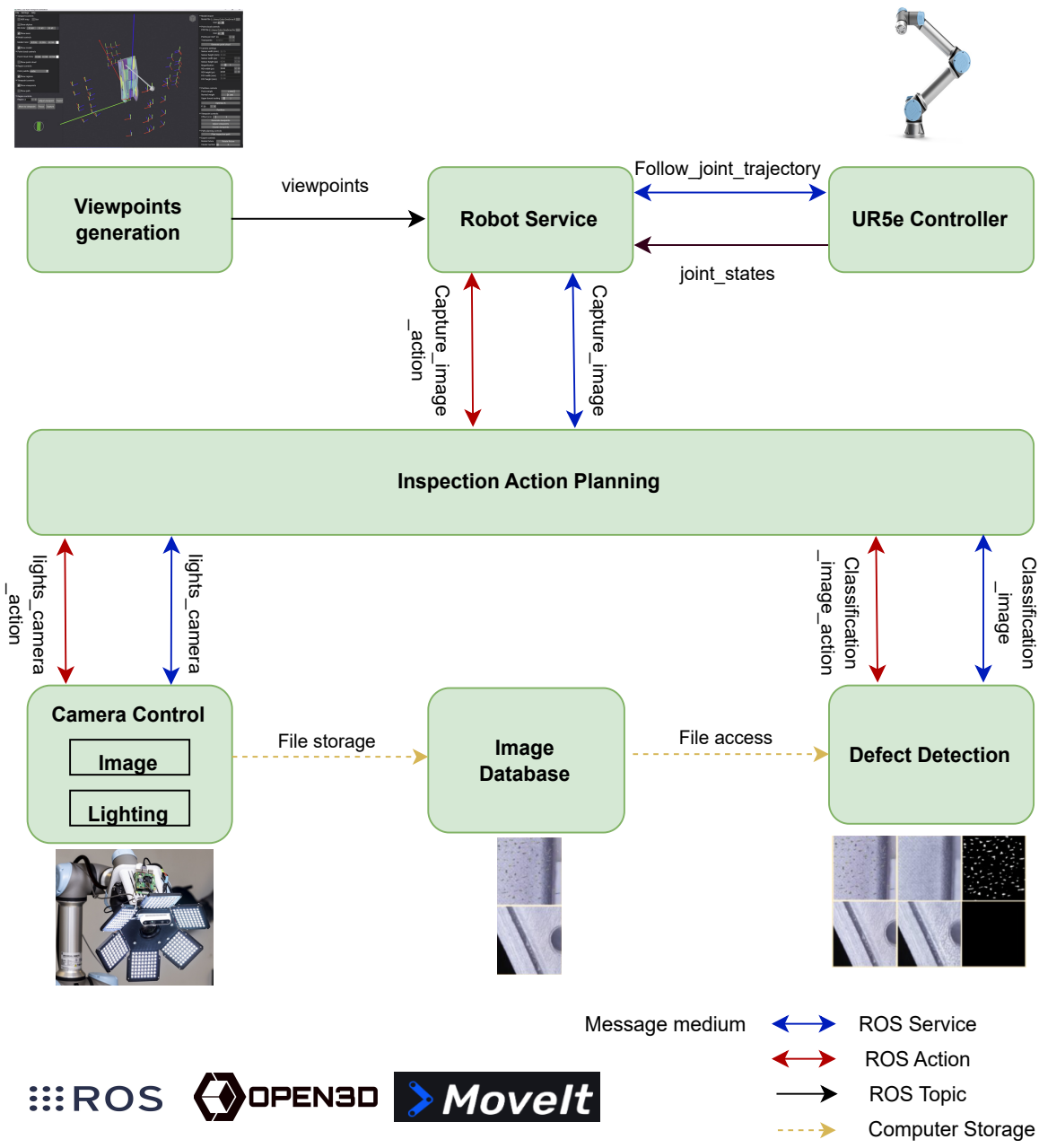


Figure 3.2: System architecture of the robotics inspection system.

frame, we conduct eye-in-hand calibration.

The eye-in-hand calibration process consists of two main steps: intrinsic calibration and extrinsic calibration. Initially, intrinsic calibration is performed to ascertain the camera's

projection matrix and distortion coefficients [41]. These parameters are essential for understanding the relationship between spatial distances and pixel distances in the camera's image. For intrinsic calibration, the well-established Zhang's Homography technique can be employed [42].

Following the completion of intrinsic calibration, we proceed with extrinsic calibration [43] using the camera's projection matrix and distortion coefficients. To achieve this, the robot is moved to various locations, and images of a calibration board with known dimensions are captured, as shown in the Figure 3.3 [1]. By imaging the calibration board from different locations, a set of corresponding image points and their respective world coordinates of the calibration board corners are collected. With this data, we can compute the extrinsic parameters, including the rotation matrix and translation vector, which describe the transformation (represented as X in the Figure 3.3) from the camera coordinate system to the world coordinate system.

As the calibration involves imaging the board at several different positions. This step of moving the robot to image a stationary board was automated by using the principles in Chapter 2. That is by using the location of the board and the camera model the poses are generated to view the board at different orientations.

3.2.1 Camera Calibration of our System

To assess the effectiveness of the calibration, a specific setup was implemented. A known location on a precisely machined, calibrated test bed was selected. The optical center location was then moved to an offset from this known point. The center of the generated image was used to determine the resulting offset. This setup is illustrated in Figure 3.4. In terms of implementing eye-in-hand calibration, the following methods were tested:

- Intrinsic calibration using the ROS camera calibration package [44] and a checkerboard, followed by MoveIt Handeye calibration [45] with an Aruco marker board.

- Intrinsic calibration using the Matlab calibration toolbox [46] and a checkerboard, followed by MoveIt Handeye calibration with a Charuco marker board.

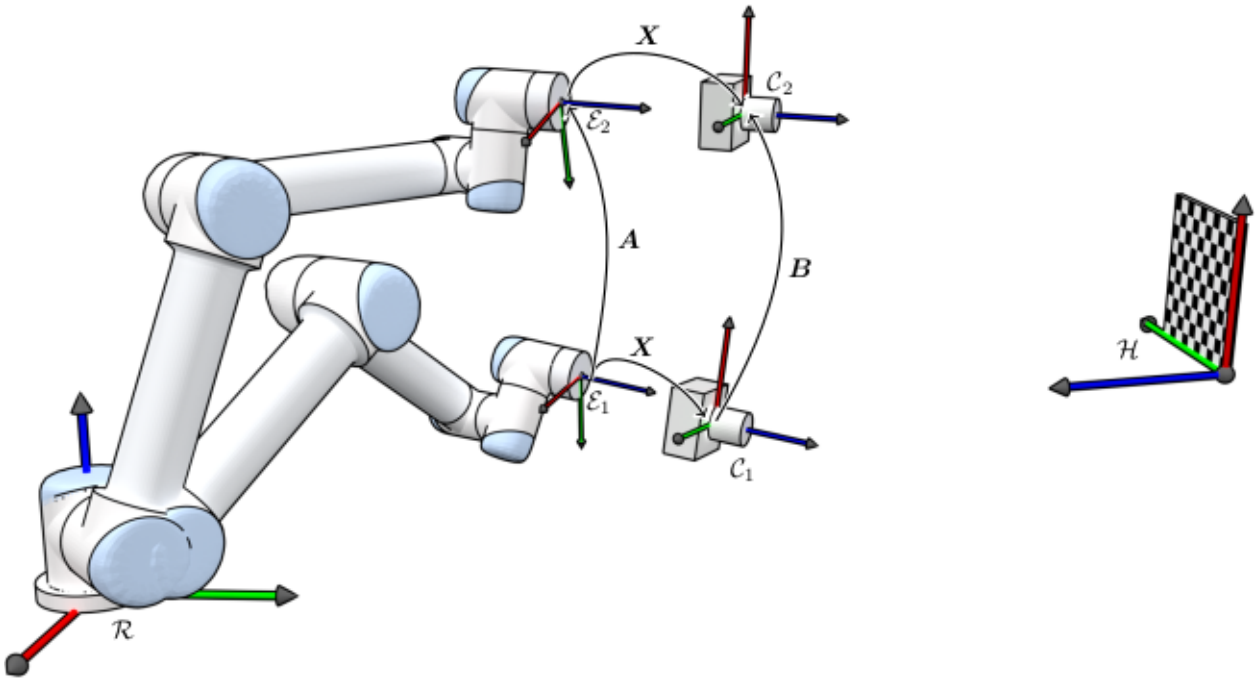


Figure 3.3: The illustration depicts the extrinsic calibration for a robot by imaging a calibration board at various locations. Figure source: [1]

Among the methods evaluated, the combination of intrinsic calibration using the Matlab calibration toolbox with a checkerboard, followed by Moveit Handeye calibration with a Charuco marker board, yielded the least offset error of 1.5 cm. This improved accuracy was attributed to the Charuco board's ability to detect a larger number of features in a single frame compared to the Aruco marker. Additionally, the Matlab calibration proved to be more reliable than the ROS package for intrinsic calibration of high-resolution cameras.

However, neither software could handle a 61-megapixel image for calibration, so a lower resolution image of 1024×680 was used instead. For the camera settings, the F-number, which indirectly corresponds to the depth of field (DOF), was set to 0.1, providing a large DOF. The magnification was set to $1:10$ to allow for a larger field of view (FOV).

To minimize this error, the calibration board was transitioned from inkjet-printed normal paper sheets to Ultra Violet (UV) printed metallic boards. This change aimed to enhance

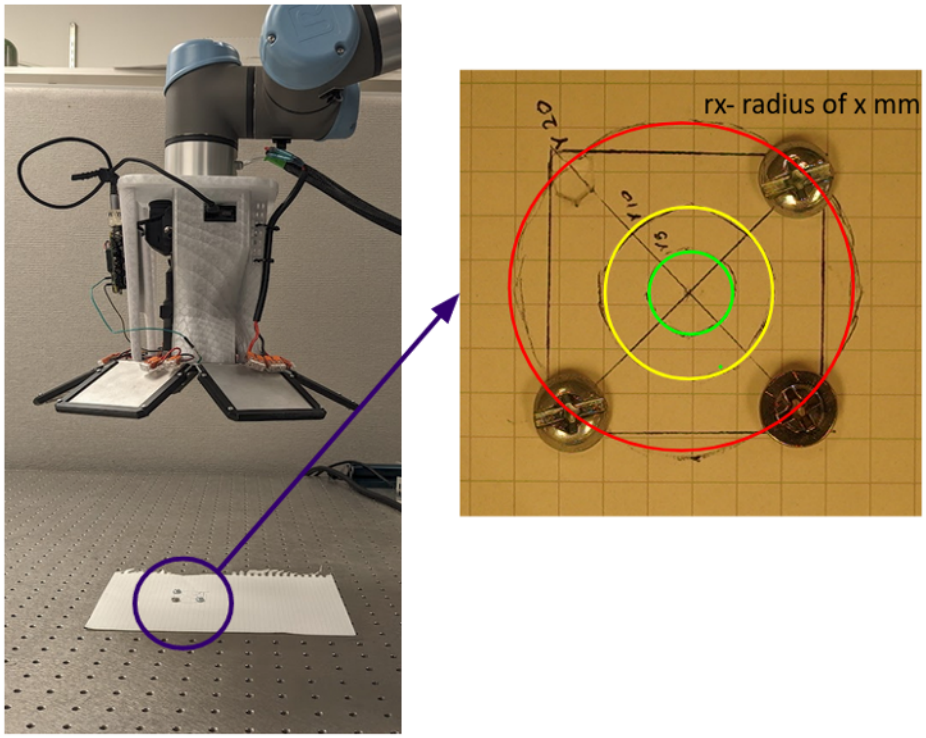


Figure 3.4: The illustration depicts the test setup to quantify the offset. The green dot represents the center of the image and the center point found through the intersection of the lines show the known location on the calibrated test bed.

the feature-to-feature dimensional accuracy to 2 microns and improve the flatness of the calibration targets. Charuco markers were used on the metallic board because they are suitable for scenarios with a smaller field of view (FOV), such as in cases involving higher magnification, where imaging the entire board at once is challenging.

For intrinsic calibration, the Camcalib software was employed to compute the projection and distortion matrices. This software was chosen because it supports Charuco markers for intrinsic calibration. For extrinsic calibration, the MoveIt HandEye Calibration plugin was utilized. However, this software could not handle a 61-megapixel image for calibration, so a lower resolution image of 2168 $\times$ 1433 was used while maintaining the aspect ratio.

In terms of camera settings, the F-number was set to 0.03, resulting in a shallow depth of field (DOF). The magnification was set to 1 : 3, providing a narrow field of view (FOV). This was done to simulate the camera setting representative of the final process requirement. The images of the calibration board, along with the detected features, are shown in Figure 3.5.

The results of this experiment showed an offset value of 3 cm, which did not improve upon the previous performance. Similar experiments were conducted with various other camera magnifications keeping the other parameters constant, as shown in Table 3.1, but the offset values remained at 3 cm in the best case.

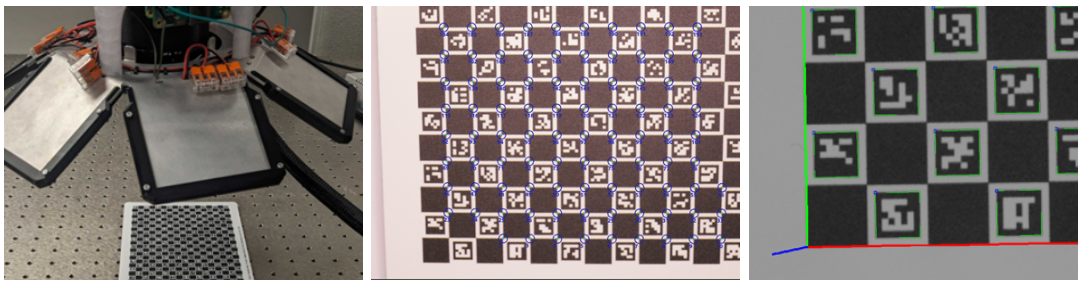


Figure 3.5: Calibration performed at 1:3 magnification with partial view of the board. The left image shows the robot pose to image the board, center image shows intrinsic calibration and the right image shows extrinsic calibration.

This led to the conclusion that the existing camera calibration techniques used by the computer vision community were not well-suited for high-precision calibration of a 61-megapixel mirrorless camera. This inadequacy could be attributed to the existing software's inability to process high-resolution images, potentially leading to a loss of information. Consequently, the methodology of utilizing Matlab for intrinsic calibration and MoveIt for extrinsic calibration was adopted. This approach consistently allowed for the determination of the optical center with an accuracy of approximately 2 cm.

To achieve finer accuracy in the submillimeter range, the following technique was employed. Initially, the optical center location was set to the value obtained through the

Table 3.1: Calibration expirements

Index	Magnification	Offset value (cm)
a	1:7	8
b	1:5	4
c	1:3	3

aforementioned calibration technique. Then, the offset was determined using the setup shown in Figure 3.4. Using the teach pendant of the UR5e robot, the robot was moved in fine steps of 0.1 mm in the x and y directions until the image's center corresponded to the target location. The distance found through this process was used as an offset to the derived calibration result. By implementing this method, submillimeter-level accuracy was achieved.

To summarize, we utilize a systematic camera calibration technique to find the optical center with an accuracy of 2 cm and then leverage the precise motion capability of the robot to determine the offset, achieving submillimeter-level accuracy.

Chapter 4

RESULTS

This chapter evaluates the applicability of viewpoint generation tools across diverse shapes, considering various metrics.

Firstly, in Section 4.1, a qualitative analysis of the mesh segmentation algorithm is conducted on common aerospace parts. Subsequently, the efficiency of the clustering process is discussed in Section 4.2.

Next, Bayesian optimization is tested on a range of shapes, with a focus on its advantages over exponential search, as explored in Section 4.3, and 4.4.

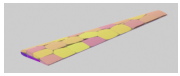
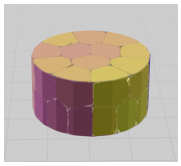
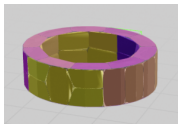
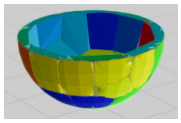
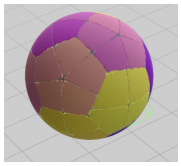
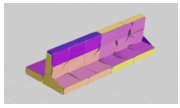
Following this, the region growth algorithm is applied as a preprocessing step for complex parts typically encountered in manufacturing settings, detailed in Section 4.5.

Lastly, the developed viewpoint tool is utilized for imaging an actual part, with outcomes presented in Section 4.6.

4.1 Mesh Segmentation Algorithm for Different Topologies

A qualitative analysis of the segmentation algorithm was conducted to evaluate its performance across various geometries. Different part geometries were sourced from GrabCAD for viewpoint generation. The machine used to compute these results is equipped with an AMD Ryzen 9 5900X 12-Core Processor @ 2.80GHz, a Graphics Card of Nvidia RTX3060Ti, and uses 32 gigabytes of RAM. To maintain consistency for comparison of the derived results between geometries, the surface area of all these meshes was kept within 10% of 2400 square millimeters. Additionally, the area of the image captured was set to 133 square millimeters. These parameters were chosen to minimize runtime while ensuring comprehensive testing across diverse shapes. The approach used for the FOV search for these shapes was exponential search as the number of segments required to divide the planar region was small. The objective was to assess the algorithm's adaptability to dif-

Table 4.1: Segmentation efficiency on benchmark topology

Index	Shape	Name	PE	# Viewpoints	Time (secs)
a		Wing	0.556	44	11
b		Cylinder	0.54	49	11.5
c		Ring	0.48	55	11.7
d		Bulkhead	0.47	59	14.3
e		Sphere	0.41	60	17
f		T-stiffener	0.35	76	24

ferent geometries, with the understanding that it could be easily extended to larger parts by adjusting the image capture area accordingly.

The camera parameters remained constant for all parts, with the following values:

- Sensor width = 35.70 mm.
- Sensor height = 23.80 mm.
- Sensor width = 9504 px.
- Sensor height = 6336 px.

- Magnification = 1:3.
- ROI Pixels = 1200 px x 1200 px.
- ROI Area = 11.54 mm × 11.54 mm.
- Depth of field = 4 mm.
- Segment FOV circular diameter = 11.54 mm.

The results are categorized into two main groups: convex shapes and concave shapes, commonly encountered in the aerospace industry. Convex examples are detailed in Table 4.1b and Table 4.1e, while concave shapes are outlined in Table 4.1a, Table 4.1c, Table 4.1d, and Table 4.1f. Notably, Table 4.1f illustrates a T-stiffener, Table 4.1a depicts an aerofoil-shaped wing, and Table 4.1d resembles a bulkhead.
The segmentation process achieved full coverage for all shapes, ensuring every part was successfully segmented. Subsequently, viewpoints were generated from these segments, and occlusion avoidance was performed to adjust for any facets blocking the projected viewpoint. Table 4.1 presents the detailed results, including the number of generated viewpoints and the time taken.

4.2 Efficiency of Clustering

The packing efficiency metric in Table 4.1 offers insight into the segmentation effectiveness for each shape, i.e. it provides information on how much new information is derived from each image taken from the generated viewpoints. It is observed that parts with extensive flat surfaces, such as the wing and the cylinder, achieve efficiencies surpassing 0.5. Conversely, shapes with pronounced curvature, such as spheres, rings, and bulkheads, demonstrate packing efficiencies ranging from 0.4 to 0.5. Notably, the T-stiffener case experiences a notable drop in efficiency, primarily due to the formation of planar segments on thin faces that prevent efficient packing of surface points within the circular FOV. Efficiency serves as a dependable indicator not only of segmentation performance but also of the time required for segmentation and the number of viewpoints required for complete coverage. Parts with lower efficiency demand a greater number of viewpoints for thorough coverage, while those with higher efficiency require fewer viewpoints to achieve equivalent

coverage. This correlation highlights the practical importance of efficiency in benchmarking various methods developed in the future.

4.3 Bayesian Optimization to Find K for Different Planar Segments

Planar segments generally manifest in one of the many common polygons. The efficiency of the BO for the search for minimum viable K is initially tested on a triangle. The cost function generated from a triangle shape with an area of 12662 mm^2 and a camera with a FOV area of 290 mm^2 is shown in Figure 4.1. The range space for the BO was set from 1 to 120 (c.f. Section 2.3.2). The parameters of the number of iterations were set to 6. Figure 4.1 depicts how the Gaussian Process can predict the minimum K value which is 71 with 6 iterations. Here it uses 3 data points to construct the prior and then uses the acquisition function to predict the minimum value over 3 iterations. This method proved to be faster than an exponential search in which to predict the upper bound (124) would have taken 8 iterations and then a binary search between 64 and 124 to find a K value equal to 71 would have taken 6 iterations. Therefore, 8 iteration searches were reduced due to Bayesian optimization. We can also see that the mean from the Gaussian Process was able to track the real curve accurately.

This method was extended to different shapes of different sizes and it was found to predict the minimum point with a maximum of 9 number of iterations to reach the minimum. An exhaustive list of shapes was evaluated and the number of iterations taken to reach the minimum is shown in Table 4.2. The BO extended well to other shapes with a very small acceptable tolerance of less than 0.25% constraint violation, which is insignificant when compared to the area these violated points contributed.

To validate the performance of the BO search method with the exponential search the following shapes evaluated previously were re-evaluated using the exponential search method. The number of iterations is compared in Table 4.3.

From our testing, the BO consistently outperformed the exponential search for the fewest evaluations of the FOV check. The gap between BO and exponential search only widens for larger search ranges. For example, when the K bound is found in between $K = 256$ and $K = 512$ the number of iteration scales to $18(=10 + log(256))$. In the same case when

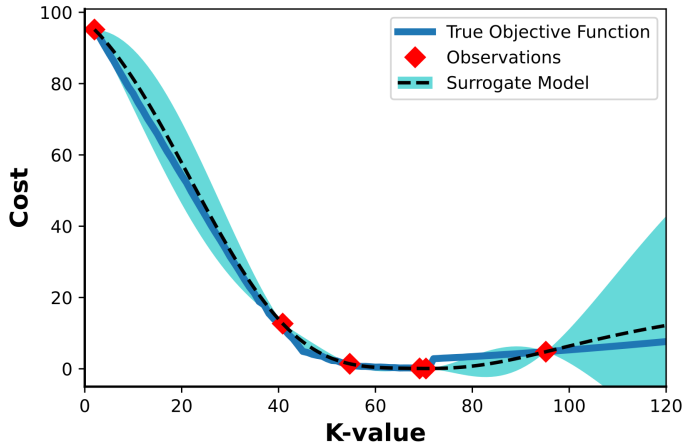


Figure 4.1: Illustration of the estimated surrogate model after 6 observations against true cost (objective) function

Table 4.2: Bayesian Optimization performance evaluation

Shape	Area of Region (mm^2)	FOV (mm^2)	BO Iterations	Constraint violation %
Triangle	53865	290	5	0.01
Hexagon	53865	1161	8	0.220
Decagon	11317	290	9	0.19
Square	400	10.7	8	0.22

evaluated using the BO, the number of iterations remained equivalent to the case where K was less, by having only 7 iterations to reach the minimum.

4.4 Optimal K Value Search Method Time Comparison

In the conducted study a sphere was utilized as the test subject to evaluate the performance of exponential search and BO methods in determining the optimal K value. The

Table 4.3: Bayesian Optimization comparison with exponential search

Shape	No. of iterations in BO	No. of iterations in exponential search
Triangle	5	18
Hexagon	8	14
Decagon	9	13
Square	8	12

sphere's area was maintained at 2400 square millimeters, while the area of the FOV varied between 3 and 290 square millimeters, corresponding to the number of clusters ranging from 195 to 1 for each flat region. BO was configured with a fixed number of iterations set to 4, employing a search range spanning from the minimum K value ($K_{\min}$) to three times $K_{\min}$. Conversely, the exponential search initiated its search from $K_{\min}$.
The data obtained from this study is shown in Figure 4.2. The analysis revealed a notable performance distinction between the two methods. Notably, the exponential search outperformed BO for K values within the range of 1 to 5, primarily due to consistent iteration counts across both approaches. However, beyond this range, BO demonstrated superior efficacy. Particularly, as the optimal K value increased to 195, the time disparity between the two methods approached approximately 100 seconds. This underscores the increasing efficiency of the BO algorithm with higher K values, rendering it particularly advantageous for identifying K values in larger surface area parts.

4.5 Segmentation with Pointcloud Preprocessing

In this section, more complex concave shapes found in the Manufacturing setting were tested with the RG algorithm as a pre-processing step. This pre-processing step removes edges and holes, enabling smooth segmentation. The uncolored regions in the CAD model

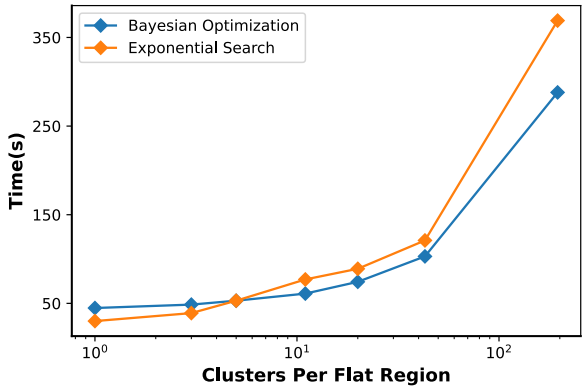


Figure 4.2: Time comparison between Binary search and Bayesian Optimization for viewpoint generation of a sphere

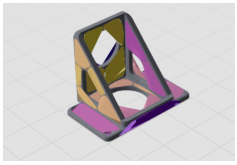
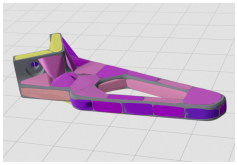
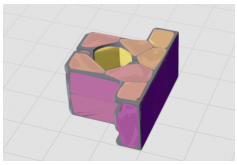
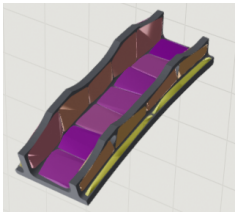
images in Table 4.4 represent these areas. Initially, all parts depicted in this section were tested without the pre-processing step and had difficulties converging to the solution. Notably, Table 4.4a and Table 4.4d illustrate a bracket and a shroud, both featuring multiple corners, small edges, and holes. Meanwhile, Table 4.4b and Table 4.4c display parts with hollow regions in addition to holes and corners. The algorithm effectively distinguishes hollow regions from holes, as evidenced by the segmentation occurring in these areas. As all parts having similar surface areas, the number of viewpoints is in the range of 40 to 45.

Although the algorithm still misses some regions, such as holes and corners, it significantly improves segmentation, now covering 90 percent of the total surface area compared to non-converging segmentation.

4.6 Imaging a Part

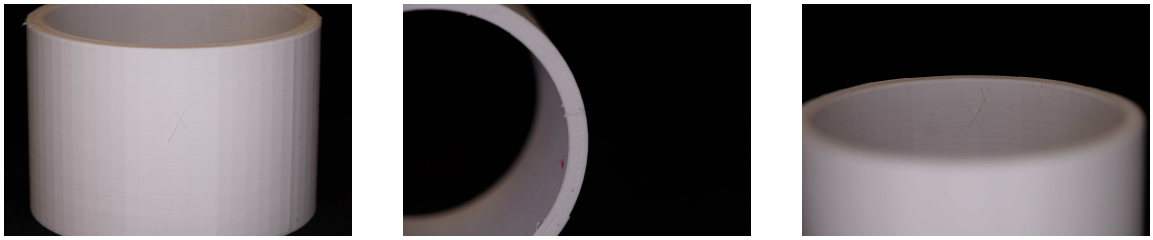
A 3D-printed hollow cylinder with a diameter of 75 mm and a height of 50 mm served as the test subject for actual viewpoint traversal and inspection. The viewpoints were generated using the same algorithm and then followed by using the proposed robotic imaging

Table 4.4: Segmentation of complex manufacturing part topology

Index	Shape	Name	# Viewpoints
a		Bracket	44
b		Leg Support	43
c		Bearing Housing	42
d		Shroud	41

system. The part was randomly marked to simulate defects, and after inspection, detailed images of these markings were identified, indicating complete coverage of the part's surface. Figure 4.3 showcases the quality and ability to capture defects at various regions of the part.

Figure 4.3a demonstrates the system's capability to capture defects on the outside surface, while Figure 4.3b illustrates defect capture on the edge. Figure 4.3c highlights the system's ability to detect defects on the inside surface. These results collectively affirm the efficacy of the proposed method and underscore its feasibility for deployment.



(a) Outside surface (b) Top edge (c) Inner surface

Figure 4.3: Images of the hollow cylinder taken using the inspection plan generated from the proposed algorithm

Chapter 5

CONCLUSION

5.1 Conclusion

Overall this work proposes an intelligent surface inspection framework using unsupervised machine learning for viewpoint generation and path planning for aerospace-grade parts. Validation was conducted on a robotic inspection cell along with various complex curved geometries. The results show applicability to a wide range of shapes and the ability to capture defects given a part's surface model and camera parameters efficiently.

5.2 Limitation and Future Work

Mesh segmentation offers customized solutions for specific applications, yet devising a universal solution for general surface inspection in the aerospace industry remains challenging due to the diverse surface geometries and unique defect characteristics. Overcoming these hurdles to create an algorithm capable of effectively handling such variations presents a significant challenge. Here, we explore recognized limitations and potential future work:

Initially, despite utilizing point cloud pre-processing for segmenting complex shapes, only 90% of the part is segmented, leaving room for improvement. Achieving full coverage may necessitate a post-processing step where discarded points are allocated to the nearest existing regions before segmentation.

Currently, generating viewpoints requires CAD models, camera parameters, and part locations, constrained by fixture setup requirements. However, employing a point cloud scanner to directly generate viewpoints from scan data could alleviate reliance on CAD models and specific part locations.

Furthermore, determining the optimal value of K for degrees of freedom (DOF) currently relies on a sub-optimal exponential search, impacting time consumption. Exploring var-

ious deep learning models for predicting K in 3D space holds promise for enhancing efficiency.

Camera calibration, currently reliant on manual correction for high precision, may increase initial setup time. Investigating advanced eye-in-hand calibration techniques, such as those discussed in [47], for DSLR cameras with macro lenses could offer efficiency improvements.

Additionally, K-means clustering's non-deterministic nature may result in different segments with repeated applications, potentially converging to a local minimum. Addressing this could involve iterating the segmentation process on new parts until satisfactory clustering is achieved, with resulting viewpoints reused for subsequent parts of the same geometry.

Lastly, the region growth algorithm's implementation in Python suffers from longer processing times compared to a C++ implementation. Transitioning the entire viewpoint generation tool to C++ and leveraging the PCL library could significantly enhance processing speed.

In conclusion, addressing these challenges through innovative solutions and methodological enhancements will drive the advancement of aerospace surface inspection methodologies, facilitating improved defect detection and quality control processes in the industry.

BIBLIOGRAPHY

[1] URVision — qipccc.github.io. https://qipccc.github.io/2019/12/01/URVision/. [Accessed 07-06-2024].

[2] P Brauny, M Hammerschmidt, and M Malik. Repair of air–cooled turbine vanes of high–performance aircraft engines–problems and experience. *Materials science and technology*, 1(9):719–727, 1985.

[3] Advisory Group For Aerospace Research And Development Neuilly-sur-seine (FRAN CE). Erosion, corrosion and foreign object damage effects in gas turbines (les consequences de l'endommagement des turbines a gaz par erosion, corrosion et objets etrangers). 1994.

[4] Jonas Aust and Dirk Pons. Comparative analysis of human operators and advanced technologies in the visual inspection of aero engine blades. *Applied Sciences*, 12(4):2250, 2022.

[5] Judi E. See. Visual inspection reliability for precision manufactured parts. *Human Factors*, 57(8):1427–1442, 2015. PMID: 26342002.

[6] Du-Ming Tsai and Po-Hao Jen. Autoencoder-based anomaly detection for surface defect inspection. *Advanced Engineering Informatics*, 48:101272, 4 2021.

[7] Yiping Gao, Xinyu Li, Xi Vincent Wang, Lihui Wang, and Liang Gao. A review on recent advances in vision-based defect recognition towards industrial intelligence. *Journal of Manufacturing Systems*, 62:753–766, 1 2022.

[8] Yuanbin Wang, Kangjie Hong, Jun Zou, Tao Peng, and Huayong Yang. A cnn-based visual sorting system with cloud-edge computing for flexible manufacturing systems. *IEEE Transactions on Industrial Informatics*, 16:4726–4735, 7 2020.

[9] Daniel Arey, James Gao, Mohammed Elsouri, and Chi Hieu Le. *An Investigation into the Adoption of Automation in the Aerospace Manufacturing Industry*, pages 87 – 92. 09 2019.

[10] Mark Gerges and Xu Chen. Adaptive lighting for curved and nonuniform objects in optomechanical inspection systems. *IEEE/ASME Transactions on Mechatronics*, 27(6):5792–5802, 2022.

[11] The ARM Institute. Project highlight: Automated defect inspection of complex metallic parts, 03 2021.

[12] Weihua Sheng, Ning Xi, Mumin Song, Yifan Chen, and J.S. Rankin. Automated cad-guided automobile part dimensional inspection. pages 1157–1162. IEEE.

[13] Dennis Mosbach, Petra Gospodnetić, Markus Rauhut, Bernd Hamann, and Hans Hagen. Feature-driven viewpoint placement for model-based surface inspection. *Machine Vision and Applications*, 32:8, 1 2021.

[14] Christian Landgraf, Bernd Meese, Michael Pabst, Georg Martius, and Marco F. Huber. A reinforcement learning approach to view planning for automated inspection tasks. *Sensors 2021, Vol. 21, Page 2030*, 21:2030, 3 2021.

[15] Yuanbin Wang, Tao Peng, Wenhu Wang, and Ming Luo. High-efficient view planning for surface inspection based on parallel deep reinforcement learning. *Advanced Engineering Informatics*, 55:101849, 1 2023.

[16] Rui S.V. Rodrigues, José F.M. Morgado, and Abel J.P. Gomes. Part-based mesh segmentation: A survey. *Computer Graphics Forum*, 37, 2018.

[17] Ariel Shamir. A survey on mesh segmentation techniques. *Computer Graphics Forum*, 27, 2008.

[18] M. Garland, A. Willmott, and P. S. Heckbert. Hierarchical face clustering on polygonal surfaces. 2001.

[19] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. 2004.

[20] Abhishek Arun Kulkarni. *Motion Planning and Image Capturing for Robotic Inspection of a Curved Surface Subject to Imaging Constraints*. University of Washington, 2020.

[21] Dejan Kaljaca, Bastiaan Vroegindeweij, and Eldert Henten. Coverage trajectory planning for a bush trimming robot arm. *Journal of Field Robotics*, 37:283–308, 3 2020.

[22] Chunhui Yuan and Haitao Yang. Research on k-value selection method of k-means clustering algorithm. *J*, 2(2):226–235, 2019.

[23] Arshad Muhammad Mehar, Kenan Matawie, and Anthony Maeder. Determining an optimal value of k in k-means clustering. In *2013 IEEE International Conference on Bioinformatics and Biomedicine*, pages 51–55, 2013.

[24] Akash Punhani, Neetu Faujdar, Krishna Kumar Mishra, and Manoharan Subramanian. Binning-based silhouette approach to find the optimal cluster using k-means. *IEEE Access*, 10:115025–115032, 2022.

[25] Kristina P. Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. *IEEE Access*, 8:80716–80727, 2020.

[26] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[27] M. Shanker, M.Y. Hu, and M.S. Hung. Effect of data standardization on neural network training. *Omega*, 24:385–397, 8 1996.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[29] Jon Louis Bentley and Andrew Chi-Chih Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.

[30] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning, 2010.

[31] Jie Wang. An intuitive tutorial to gaussian processes regression, 2022.

[32] Eduardo C. Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing*, 380:20–35, 2020.

[33] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[34] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2, 1978.

[35] Phuc Luong, Sunil Gupta, Dang Nguyen, Santu Rana, and Svetha Venkatesh. Bayesian optimization with discrete variables. In *AI 2019: Advances in Artificial Intelligence: 32nd Australasian Joint Conference, Adelaide, SA, Australia, December 2–5, 2019, Proceedings 32*, pages 473–484. Springer, 2019.

[36] Rickard Karlsson, Laurens Bliek, Sicco Verwer, and Mathijs de Weerdt. Continuous surrogate-based optimization algorithms are well-suited for expensive discrete problems. In *Artificial Intelligence and Machine Learning: 32nd Benelux Conference, Leiden The Netherlands, November 19–20, 2020, Revised Selected Papers*, page 48–63. Springer International Publishing, 2021.

[37] Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with gaussian process regression. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, IJCAI'07, page 944–949, 2007.

[38] G. Pirot, T. Krityakierne, D. Ginsbourger, and P. Renard. Contaminant source localization via bayesian global optimization. *Hydrology and Earth System Sciences*, 23(1):351–369, 2019.

[39] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[40] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *European Interdisciplinary Cybersecurity Conference*, 2008.

[41] Camera calibrator. MathWorks. (n.d.-a). https://www.mathworks.com/help/vision/ug/camera-calibration.html . [Accessed 06-06-2024].

[42] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[43] 3 - MoveIt - Easy Hand Eye Calibration with MoveIt — youtube.com. https://www.youtube.com/watch?v=xQ79ysnrzUk. [Accessed 07-06-2024].

[44] camera_calibration/Tutorials/MonocularCalibration - ROS Wiki — wiki.ros.org. https://wiki.ros.org/camera$_c alibration/Tutorials/MonocularCalibration.[Accessed 07-06-2024]$.

[45] GitHub - moveit/moveit_calibration: Hand-eye calibration tools for robot arms. — github.com. https://github.com/moveit/moveit$_c alibration.[Accessed 07-06-2024]$.

[46] Camera Calibration - MATLAB & Simulink — mathworks.com. https://www.mathworks.com/help/vision/camera-calibration.html. [Accessed 07-06-2024].

[47] Wei Li, Mingli Dong, Naiguang Lu, Xiaoping Lou, and Peng Sun. Simultaneous robot–world and hand–eye calibration without a calibration object. *Sensors*, 18(11), 2018.

[48] Akash Punhani, Neetu Faujdar, Krishna Kumar Mishra, and Manoharan Subramanian. Binning-based silhouette approach to find the optimal cluster using k-means. *IEEE Access*, 10:115025–115032, 2022.

[49] William R. Scott. Model-based view planning. *Machine Vision and Applications*, 20:47–69, 1 2009.

[50] Chuping Liang, Jie Yin, Jing Wu, Jun Wang, Mingqiang Wei, and Yanwen Guo. A survey of 3d mesh segmentation based on clustering analysis, 2020.

[51] Petra Gospodnetic, Dennis Mosbach, Markus Rauhut, and Hans Hagen. Flexible surface inspection planning pipeline. In *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*, pages 644–652, 2020.

[52] Heping Chen, Thomas Fuhlbrigge, and Xiongzi Li. Automated industrial robot path planning for spray painting process: a review. In *2008 IEEE International Conference on Automation Science and Engineering*, pages 522–527. IEEE, 2008.

[53] A. Kulkarni. Motion planning and image capturing for robotic inspection of a curved surface subject to imaging constraints. Master's thesis, University of Washington, 2020.

Appendix A

CODE

The work on segmentation with automated pointcloud processing along with Surface models utilized for this project is made available online at:
https://github.com/macs-lab/InspectionMLviewpointGen.git.