FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

6DoF tool path generator from CAD model for visual inspection of part surfaces

Luís Rodrigues de Castro

Mestrado em Visão por Computador

Supervisor: Luís Filipe Pinto de Almeida Teixeira

Second Supervisor: Sacha Trevelyan Mould

April 5, 2022

6DoF tool path generator from CAD model for visual inspection of part surfaces

Luís Rodrigues de Castro

Mestrado em Visão por Computador

April 5, 2022

Abstract

Nowadays, the inspection of metallic parts for the aerospace industry (in particular, the Fluorescent Penetrant Inspection) is mainly performed by trained and expert operators. These people are subject to long periods of high cognitive load, often in poor postures. This could be avoided with the automation of the inspection procedure, even if under the control of these operators. This is, indeed, the target of the project that integrates this thesis.

The project is comprised of two parts. The first is the topic of this thesis, which proposes a generic off-line framework to generate, select and sort a sequence of viewpoints (with 6 DoF, representing a camera pose in the 3D space) from a CAD model of a component. The second (which is out of the thesis scope) will use the framework here created to guide a robotic manipulator (i.e., following the sorted sequence of viewpoints) in the inspection of such a component. In short, this thesis aims at finding the minimum set of viewpoints that scans the maximum surface area of a component, and at the same time, minimize the traveling cost required to go through all the viewpoints.

Today's literature includes different and effective algorithms to generate and select viewpoints for a surface inspection, but not to sort these viewpoints, leaving a gap for an integral framework.

Hence, this thesis proposes a more complete framework that starts by generating a high number of viewpoints from a tessellated mesh and a set of camera and scanning parameters (camera's Field of View, scanning distance and others). Those viewpoints are, then, validated and filtered through a set of measurability criteria, that constrain the visibility of the viewpoints. As the generated viewpoints are numerous and redundant, they are, then, pruned using three different methods: Method 1 selects the viewpoints that maximize an object function, thus having a greedy behavior; Method 2, is a generalization of Method 1, and operates using the same greedy principals, however the objective function uses 6 distinct features, whose weights are optimized by the Particle Swarm Optimization algorithm; finally, Method 3 uses the simulated annealing optimizer to find a subset of viewpoints. The resultant subset from the best method was sorted using the Ant Colony Optimization algorithm – a step over the state of the art.

The framework was validated using a representative CAD model, from which 4552 viewpoints were extracted. Method 3 proved to be the best, with a 91.6% reduction of the number of viewpoints, however, at the expenditure of computational cost, when compared to Method 1, which was faster. Method 2 produced the worst results, as the optimizer failed to converge.

Even if only one mesh was studied, its characteristics are quite representative of variable meshes and it also served to exemplify the well functioning of the proposed framework. In conclusion, it was possible to determine that Method 3 is the most suitable to integrate this framework, and this is in agreement with results of past literature.

Keywords: Tessellated mesh, Pointclouds, Dijkstra's shortest path, Particle Swarm Optimization, Ant Colony Optimization, Travelling Salesman Problem, Camera viewpoints

Resumo

Atualmente, a inspeção de peças metálicas para a indústria aeroespacial (em particular, o método de Fluidos Penetrantes) é realizada principalmente por operadores treinados e especializados. Estas pessoas estão sujeitas a longos períodos de intensa concentração, muitas vezes numa má postura corporal. Isto poderia ser evitado com a automatização do procedimento de inspeção, mesmo que sob o controlo desses operadores. Este é, aliás, o alvo do projeto que integra esta tese.

O projeto é composto por duas partes. A primeira é onde se enquadra o tema da presente tese, que propõe uma *framework* genérica e off-line para gerar, selecionar e ordenar uma sequência de pontos de vista (com 6 graus de liberdade, em que cada um representa a pose de uma câmara no espaço tridimensional) a partir de um modelo CAD de um componente. A segunda (fora do âmbito desta tese) usará a *framework* aqui criada para guiar um manipulador robótico (i.e., seguindo a sequência ordenada de pontos de vista) na inspeção de um tal componente. Em sumário, esta tese procura encontrar um conjunto mínimo de pontos de vista, capaz de inspecionar o máximo de área superficial do componente, e ao mesmo tempo minimizar o custo de transporte necessário para percorrer todos os pontos de vista.

A literatura corrente inclui vários algoritmos eficazes para gerar e selecionar pontos de vista para a inspeção de uma superfície, mas não para ordenar esses pontos de vista, deixando em falta uma *framework* que integre todos os passos.

Assim, esta tese propõe uma *framework* integral que começa por gerar um grande número de pontos de vista a partir de uma malha tesselada e um conjunto de parâmetros relacionados com a câmera e o scanning (campo de visão da câmera, distância de scan e outros parâmetros). Esses pontos de vista são validados e filtrados através de um conjunto de critérios de mensurabilidade, que restringem a visibilidade dos pontos de vista. Visto que muitos pontos de vista são gerados de forma redundante, três métodos diferentes foram usados para reduzir o número de pontos de vista: o Método 1 seleciona os pontos de vista que maximizam uma função objetivo, e portanto é dotado de um comportamento ganancioso; o Método 2 é uma generalização do Método 1, e opera usando o mesmo principio ganancioso, no entanto a função objetivo utiliza 6 *features* distintas, cujos pesos associados são otimizados pelo algoritmo *Particle Swarm Optimization*; finalmente, o Método 3 utiliza o otimizador *simulated annealing* para encontrar um subconjunto de pontos de vista. O subconjunto resultante do melhor método foi ordenado usando o Algoritmo de Otimização por Colónia de Formigas – um passo introduzido além do estado da arte.

A *framework* foi validada usando um modelo CAD representativo, do qual foram extraídos 4552 pontos de vista. O Método 3 mostrou-se o melhor, com uma redução do número de pontos de vista em 91.6%. Porém, este método acarreta um maior custo computacional, quando comparado com o Método 1, que foi mais rápido. O Método 2 obteve os piores resultados, uma vez que o otimizador não convergiu.

Mesmo que apenas uma malha tenha sido estudada, as suas características são bastante representativas de várias malhas e serviu também para exemplificar o bom funcionamento da *framework*

proposta. Em conclusão, foi possível determinar que o Método 3 é o mais adequado para integrar esta *framework*, o que está de acordo com resultados observados na literatura.

Keywords: malha tesselada, nuvem de pontos, Caminho minimo de Dijkstra, Otimização por enxame de particulas, Algoritmo de Otimização por Colónia de Formigas, Problema do Vendedor Viajante, Pontos de vista de uma câmara

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and goals

The scope of this thesis is associated with a project hosted by the company *Digital Transformation Colab* [2]. The aim of the project, named *Coruja*, is to automatize the Fluorescent Penetrant Inspection (FPI) on metallic parts for the aerospace industry. The visual inspection of the metallic surface is to be done by a robotic platform, comprised of an image acquisition system, that takes several images of the object surface. Those images are then fed to a deep Neural Network that detects and classifies surface defects.

Nowadays these inspections are mainly done by trained and expert operators that are often in bad postures leading to muscular-skeleton damages. Furthermore, the inspection can be compromised by human factors such as drowsiness or physical/mental fatigue [16].

Automating these activities can reduce inspection times, and allow the reallocation of workers into less physically demanding posts.

The goal of the thesis is to maximize the scanned surface area of a component, while minimizing the amount of viewpoints to do so, and, at the same time, minimize the traveling cost required to go through all the viewpoints. For that, a framework was developed, able to generate, select and sort a set of viewpoints, capable of scanning most of the surface area of a part. The architecture is to be run off-line, uses a pre-existing CAD model (mesh file) of the part to be inspected and has to be generic to any type of closed mesh (mesh with no holes). Moving outside of the scope for this thesis, the robotic platform should then visit each computed viewpoint, in order to visually scan the part. At each viewpoint, an image is taken and fed to the defect detection and classification model.

1.2 Background

To better understand the methods developed, some prior background is first given:

- **Traveling Salesman Problem (TSP)**: It is a problem that aims at finding a path that visits all nodes once while minimizing the traveling cost [13].

- **Particle Swarm Optimization (PSO)**: This optimization uses a population of particles, that have a position, velocity and momentum component and each particle stores the best position it has seen so far. The momentum allows a particle to accumulate speed in a favorable direction independent of the local perturbations. The particles are accelerated at each iteration towards both the best position that particle has encountered and the best position found among all particles. The acceleration contains two random weights that give some stochasticity to the optimization process [8, pp.158-159].

- **Ant Colony Optimization (ACO)**: It is a stochastic method used for optimizing paths through graphs, and in this thesis was used to produce a near-optimal solution for the TSP. This algorithm was inspired in how ants search for food, leaving pheromone trails behind them. Those pheromones will be captured by other ants that will follow the same trail. As pheromones evaporate the unused trails will fade and the shorter paths, with stronger pheromones, will be traveled more often attracting more ants [8, pp.354-359].

- **Dijkstra's shortest path algorithm**: This algorithm finds the shortest path between the source node of a graph and all other nodes producing a shortest path tree. It is used in GPS, to find the shortest path between the current location and the destination, and modelling networks [11].

- **Simulated Annealing (SA)**: This is a stochastic optimization process inspired in the metallurgy. The temperature of the process controls the level of stochasticity during the random search. Temperature starts high allowing to explore the search space freely, in the hope that in this phase the process will find a good regions with the best local minimum. As the temperature is slowly reduced so does the stochasticity, forcing the search to converge to a minimum. SA is used on functions with many local minimum due to its ability to escape those areas [8, pp.128-133].

1.3 Thesis structure

The thesis is organized in the following manner: Chapter 2 delves into the state of the art of viewpoint generation and selection/pruning; Chapter 3 explains all the steps that comprise the developed framework and the methodology used to generate, prune and sort the viewpoints; Chapter 4 presents the results of the several steps of the framework; in Chapter 5 we discuss the results obtained; finally Chapter 6 concludes the thesis and presents some future works.

Chapter 2

State of the art

The state of the art regarding the viewpoint generation and pruning tasks, can be divided into two main approaches, non-voxel and voxel approaches.

2.1 Non-Voxel approaches

One of the first works in the viewpoint generation for visual scanning of parts was done by Tarbox and Gottschlich [17]. Here they developed and tested three different algorithms that generate and select a set of viewpoints to scan an object. Their setup is comprised of a camera and a light source (see figure 2.1) that can only move within the *viewing sphere* (contains the object to be scanned), and points to its center. Then they discretize the *viewing sphere* surface by fitting a icosahedron and subdivide its faces recursively and project the vertices into the sphere, until an acceptable amount of points has been reached (figure 2.2). Each of the generated points represents a valid viewpoint. The CAD model of the object to be scanned is converted to a point cloud by sampling points on the surface in a uniform manner. They developed three different algorithm, A, B (derivation from A) and C that try to compute the smallest number of viewpoints that allow to measure the highest amount of sampled points from the surface. All algorithm rely on the concept of measurably. To consider a point to be measurable, they defined two criteria:

1. occlusion criterion: A surface point is not viewable if there is a solid volume between the surface point and the viewpoint. It is checked using ray tracing techniques;

2. surface normal criterion: It is defined by the glance angle, angle between the normal vector of a surface point and the vector going from the viewpoint origin to that point. As the glance angle approaches $\pi/2$ small object surface deviations can cause regions to become occluded. A limit of 80° was used in their work.

Besides these criteria, they define two data-structures, the measurability matrix, $C(i,k)$, and measuring difficulty vector, W_i. The measurability matrix is a two-dimensional binary array, whose columns represent a viewpoint and the rows represent the sampled points of the surface. A cell (i,k) has a value of 1 if the surface point i is not occluded in a given k viewpoint and the glance

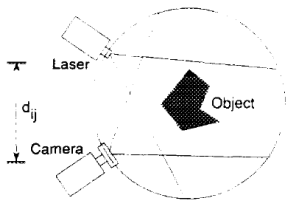

Figure 2.1: Light, camera and object placement around the *viewing sphere* [17].

angle between that same surface point and viewpoint is smaller than 80°, otherwise it has a zero value. The measuring difficulty vector has size equal to the number of sampled points and it is equal to the inverse of the sum of the measurability matrix over its rows. A high value indicates that a surface point is difficult to measure.

Algorithm A, iteratively selects the viewpoint that maximizes the following objective function:

$$f(k) = \sum_{i=1}^{|S|} C(i,k)W_i \tag{2.1}$$

where $f(k)$ is the objective value for the k_{th} viewpoint. After computing this function to all viewpoints, the one with the highest objective value is selected, k_{select}. After its selection, the measurability matrix has to be updated, and the rows, i_{select} where the column vector $C(i,k_{select})$ is 1 are set to zeros. This means that the already measured surface points cannot be measured again, so, their measurability is zero across all possible viewpoint. The selection process proceeds always followed by the update of the measurability matrix, until this matrix is all zeros, meaning all surface points have been measured. The designed objective function is hoped to be biased towards viewpoints that can view difficult regions (high W_i) and also acquire large portions of the surrounding surface areas for these regions.

Algorithm B is similar to A, with a difference on the objective function which takes the form.

$$f(k) = \sum_{i=1}^{|S|} C(i,k)W_i \frac{\Theta_g(s_i)}{\theta_g(s_i,k)} \tag{2.2}$$

$\Theta_g(s_i)$ is the minimum glance angle (across all viewpoints) for the surface point i and $\theta_g(s_i)$ is the glance angle, for viewpoint k and surface point i. So, the glance angle is now included on the objective function and allows to choose "better" viewpoints to measure certain regions of the object difficult to see.

Both Algorithms A and B start by selecting regions with an high area, which can create small holes between selected viewpoints. Algorithm C tries to solve such problem by locally adjusting viewpoints, while using the simulated annealing process to find a suitable solution. They start by randomly picking n_0 viewpoints and at each iteration of the simulated annealing process, they perturb a chosen viewpoint by replacing it for one of its neighbour viewpoints. At each iteration, the energy is recalculated and, depending on the temperature, the current viewpoint solution can be saved and reused for the next iteration. If n_0 viewpoints are not enough to cover the surface

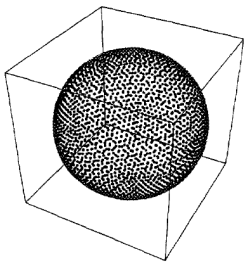

Figure 2.2: Discretized surface of the *viewing sphere* [17].

area, then this number is increased and the simulated annealing process starts again. This approach behaves in a very random way initially and as the temperature cools the search becomes more like a standard gradient descent. They have used the results of Algorithm A and B to set a value for the initial number of viewpoints, n_0.

Regarding the performance of the algorithms, they have concluded that algorithm A is ideal if a quick plan, but not optimal, is desired. Algorithm B, generally produces plans that are longer than algorithms A or C but are more suitable to inspect deep concavities in objects. Lastly algorithm C can generate smaller plans than A and B however the computational time is far superior.

2.2 Voxel approaches

In other related work, Lartigue *et al.* [10] developed a generic approach for scan path planning using different type of digitizing sensors. Similarly to Tarbox and Gottschlich [17], their goal is to find a minimum set of viewpoints that digitize the part. They have considered the sensor FoV as a cone and discretized the mesh model into voxels, whose size is related to the sensor FoV. For each voxel, the normal vectors of the faces contained on it are averaged to obtain the voxel normal vector. They also define two criteria for the voxels: consistency and qualification. The consistency checks how many faces inside the voxel have an angle between their normal vector and the voxel's normal vector higher than a threshold. The qualification step, classifies each voxel as well-seen, poorly-seen or not-seen, based on the angle between the voxel's normal vector and the viewpoint direction and in which range this angle fits. The viewpoint finding algorithm is composed of 4 main steps:

1. Create a voxel-map and define an initial set of viewpoints- To define the initial viewpoints, a parallelipiped is fit to the voxel-map along its coordinate system, (X, Y, Z), then it is enlarged by the optimal digitizing distance (d_{opt}). The center of the faces of each voxel are projected into the nearest face of the enlarged parallelipiped, defining the initial viewpoints positions. The viewpoints' orientation is equal to the normal of the intersected face;

2. Adaptive refinement of the voxel-map based on the consistency of the normal vectors- If the voxel consistency is not verified the voxel is divided and the consistency of the newly divided voxels is evaluated again. The process is performed iteratively until the consistency

criterion is reached or a minimum size for the voxel is reached. The new voxels' faces are projected to the parallelipiped creating more valid viewpoints;

3. Qualification of the voxel- A direction among the 6 available from the voxel-map coordinate system (see figure 2.3) is chosen and only the well-seen and poorly-seen voxels, visible from that direction are selected. Each of the selected voxels generate a viewpoint whose origin is the voxel's center offsetted by the optimal digitizing distance in the selected direction. For each new created viewpoint, there is a collision checking between a sphere, centered in this new viewpoint's origin and with the dimensions of the sensor, and the voxel-map. The whole process is repeated for the other 5 directions;

4. Determination of additional points of view for not-seen voxels- After the whole process, some voxels may remain unseen, so, for each, an additional viewpoint is created, offsetting the not-seen voxels' center along their normal vectors.

Figure 2.4 depicts an example of the generated voxel-map and viewpoints for a test surface.

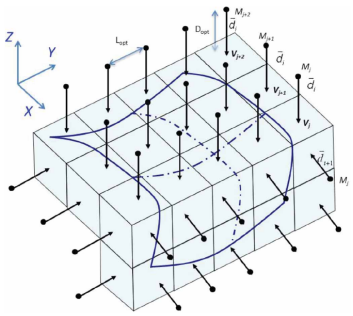

Figure 2.3: Simplified scheme with variables and their meaning [10]. (X, Y, Z): Voxel-map coordinate system; d_i, d_{i+1}: viewpoint directions, aligned with the coordinate system; M_i, M_{i+1}: positions of the viewpoints; L_{opt}: width of the FoV of the sensor; D_{opt}: optimal digitizing distance.

This voxelization approach is also used by Martins *et al.* [12], where they classify each voxel as *Surface*, *Empty* or *Inside*. For the *Surface* voxels, their normal vector was computed using a weighted average of the normal vectors of the faces inside that voxel, using the face area as the weighting term. Each voxel gives rise to a surface point, from where a viewpoint is defined using the voxel's normal vector as the offset direction. To measure a surface point they defined: the viewable criterion, no occlusion between a viewpoint j and a surface point i; accessible criterion, no intersection between sensor model (simulated as sphere), centered at viewpoint j, and the voxel-map; confidence criterion, angle between viewpoint direction and surface normal must be less than a threshold. Like Tarbox and Gottschlich [17], they have defined a measurability matrix, $M(i, j)$, where the rows represent the surface points and the columns the viewpoints, and likewise, each cell is equal to one if all the criteria for the surface point and viewpoint pair are respected. From the measurability matrix, the objective function is calculated for each viewpoint j in the following manner:

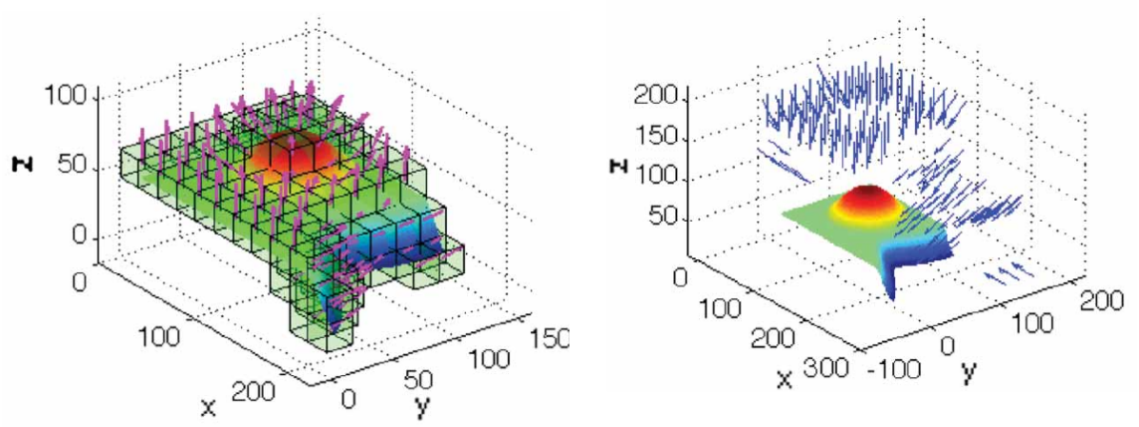

Figure 2.4: Results of the method developed by Lartigue *et al.* [10]. Left: Generated voxel-map and possible viewpoints; Right: Subset of viewpoints generated after applying the selection algorithm.

$$G(j) = \omega_s G_s(j) + \omega_{sc} G_{sc}(j) \tag{2.3}$$

where ω_s and ω_{sc} are weights that sum up to 1. $G_s(j)$ represents the amount of surface coverage (sum of the M matrix along column j) and G_{sc} is related to the inverse of the surface scanning cost. The viewpoint that maximizes the objective function is selected and the measurability matrix is updated by masking off the measured surface points. The selection process is done until all surface points are measured. The solution is not optimal but it "guarantees that the viewpoint set is complete in the sense that all possible measurable surface voxels will be covered by the viewpoint set" [12].

Chapter 3

Materials and methods

The framework is divided into three parts:

1. first, a set of valid viewpoints were generated, respecting several constraints such as, Field of View (FoV), Depth of View (DoV), glance angle, collision with part and occlusion;

2. then, a subset of viewpoints were chosen, and for that three different methods were developed and tested, involving non gradient optimization algorithms such as the Particle Swarm Optimization (PSO) and Simulated Annealing (SA);

3. lastly, the set of chosen viewpoints was ordered. To organize the viewpoints, Dijkstra's Shortest Path algorithm was used to get a free collision path between connected viewpoints and the Ant Colony Optimization (ACO), was the responsible to sort the viewpoints based on their distances.

3.1 Tested mesh

The methods developed in this thesis should be general and applicable to closed meshes, being them convex or concave and be robust to occlusion and collision problems that might arise. So, in order to test the robustness of the developed algorithms, the mesh downloaded from a CAD repository [4] and depicted in figure 3.1 a) was selected. The original mesh was too refined so, to save memory and computational time, a remeshing procedure was applied according to the algorithm found in GitHub at [5], resulting in a mesh length resolution of 1.6mm, see figure 3.1 b). The selected mesh has planar, concave and convex regions as well as hard-to-see regions, the inside part of the ears of the bunny, so it is as a complete and challenging example to test the algorithms. The mesh file used was in the STL format, which contains the vertex coordinates of the faces and which vertices compose each face.

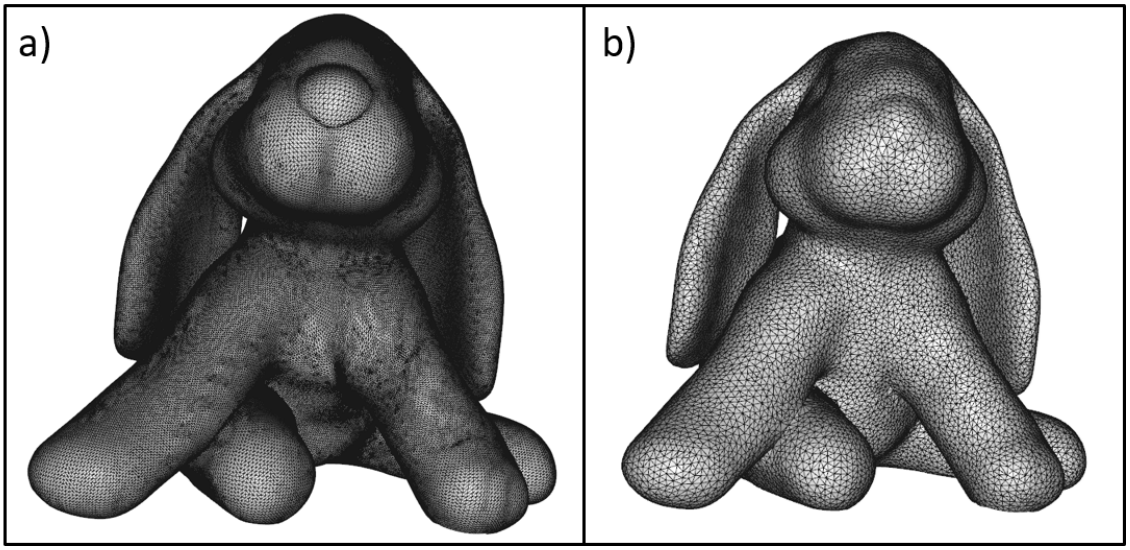

Figure 3.1: Mesh used to test the framework. a) Original mesh; b) Pre-processed mesh with the algorithm found in [5].

3.2 The proposed framework

3.2.1 Viewpoints generation pipeline

A set of valid viewpoints were generated in a pipeline, comprised of the following steps:

1. **point cloud subsampling**- The first step sampled a uniform point cloud from the mesh. The amount of points were defined as a percentage of the number of faces;

2. **generate seed faces**- From the subsampled point cloud, the closest face to each point is searched and stored. In the end, repeated faces were removed so we are left with a unique set of faces from the mesh, the seed faces. Each seed face was the genesis for the creation of a viewpoint;

3. **compute viewpoint pose**- The viewpoint origin was computed by offsetting the centroid of the seed face by the scanning distance, d_{scan}, in the direction of the normal vector of this face (always points to the outside of the mesh). A viewpoint can be seen as a camera with an associated coordinate frame (X_{vp}, Y_{vp}, Z_{vp}). Z_{vp} is the direction the camera is pointing to and is equal to the opposite direction of the normal vector of the seed face. So the viewpoint is pointing to the centroid of the seed face. Initially the viewpoint's coordinate frame was aligned with the mesh coordinate frame, and one had to find the rotation matrix, $[M]$, that aligned the initial Z_{vp} axis, denoted as $\vec{q}$, with the goal Z_{vp}, denoted as $\vec{p}$, (pointing to the centroid of the seed face). $[M]$, was computed using the following method by Kundu *et al.* [9]:

$$[M] = I + [r]_x + \frac{[r]_x^2}{1 + \vec{p} \cdot \vec{q}}$$

$$r = \vec{p} \times \vec{q}$$

$$[r]_x = \begin{bmatrix} 0 & -r3 & r2 \\ r3 & 0 & -r1 \\ -r2 & r1 & 0 \end{bmatrix}$$

$$\vec{q} : Goal \quad Z \quad axis$$

$$\vec{p} : Initial \quad Z \quad axis$$

The viewpoint pose merges the rotation matrix and its origin, $[t]$, into a single matrix, $[T]$, the homogeneous transformation matrix.

$$[T] = \begin{bmatrix} [M] & [t] \\ [0] & 1 \end{bmatrix}$$

Figure 3.2 illustrates the viewpoint frame as well as the p and q vectors that have to be aligned.

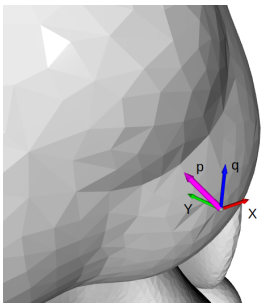

Figure 3.2: Frame located at the origin of a viewpoint. X, Y and q are the viewpoint's frame before aligning q with p.

4. **collision checking**- After defining the viewpoint pose, it was verified if a collision between the sensor model (to simplify, it was modeled by a sphere) and the mesh occurred. If so, the viewpoint in analysis was discarded;

5. **occlusion checking**- If no collision was detected, the next step evaluated if a ray, beginning at the viewpoint origin and ending at the centroid on the corresponding viewpoint's seed face, did not intersect any volume belonging to the mesh. If this criterion was not respected the viewpoint was discarded as well;

6. **compute visible triangles**- After validating the viewpoint, the faces that were visible and inside the camera's frustum were extracted using ray casting techniques. The frustum of

the camera was defined from the Angular Field of View (AFoV). The extracted faces were stored and filtered in the next stages;

7. **DoV criterion**- The selected faces were first filtered based on the Depth of View (DoV) defined for the camera. We may wish to capture the surface that is within a specific distance range from the camera because outside of it, the surface may appear out of focus affecting the result and conclusions one might extract from posterior image processing;

8. **glance angle criterion**- The selected faces were further filter by checking if the glance angle was within a threshold. This angle was measured between the normal vector of the face and the vector going from the face's centroid to the viewpoint origin. This criterion, reported in [17] and [10], avoids including faces which are very tilted w.r.t the camera thus having a poor visibility.

The points 3) to 8) were repeated for each seed face in order to create a set of valid viewpoints with their corresponding valid faces.

The parameters used by the pipeline were:

- **AFoV**- It is composed by the horizontal and vertical FoV and it defines the frustum of the camera so we know which faces are visible from a given viewpoint;
- **DoV**- Defines a distance range in the Z_{vp} axis, after its alignment, that the faces must be at, in order to be visible/valid;
- **Glance Angle threshold**- If the glance angle of a face is above this threshold, it is removed because of its poor visibility;
- **Camera model**- As mentioned, to simplify, this model was considered a sphere.

3.2.2 Viewpoints selection

The output of section 3.2.1 was a set of viewpoints that can be potentially reduced while keeping the same scanning surface area. To find a subset of viewpoints that respect the original scanning area three different methods were developed and tested.

3.2.2.1 Method 1 (Greedy area)

This method is similar to Algorithm A by Tarbox and Gottschlich [17]. Likewise, a measurability matrix, $C(i, j)$ (two-dimension binary array) is defined, where the rows correspond to the mesh faces and the columns to the viewpoints from the original set. Matrix C has a cell value of 1 if the face i, happens to fulfill the criteria 6) to 8) (defined in 3.2.1) of viewpoint j, otherwise has a value of 0. The objective function, evaluated at each viewpoint, is simply the sum of the area of the faces measurable from that viewpoint. The faces are measurable if $C(i, j) = 1$. The selected viewpoint, VP_{sel} is the one that maximizes the objective function. After selecting a viewpoint, the

measurability matrix has to be updated since there are faces that were measured. The update is done by simply zeroing the rows corresponding to the measured faces. This way all viewpoints will have those faces as not measurable anymore. The selection and update process continue until the measurability matrix is filled with zeros.

This method iteratively selects the viewpoints with highest measurable area, thus it has been named greedy area. See figure 3.3 for a schematic diagram with the steps for the algorithm.

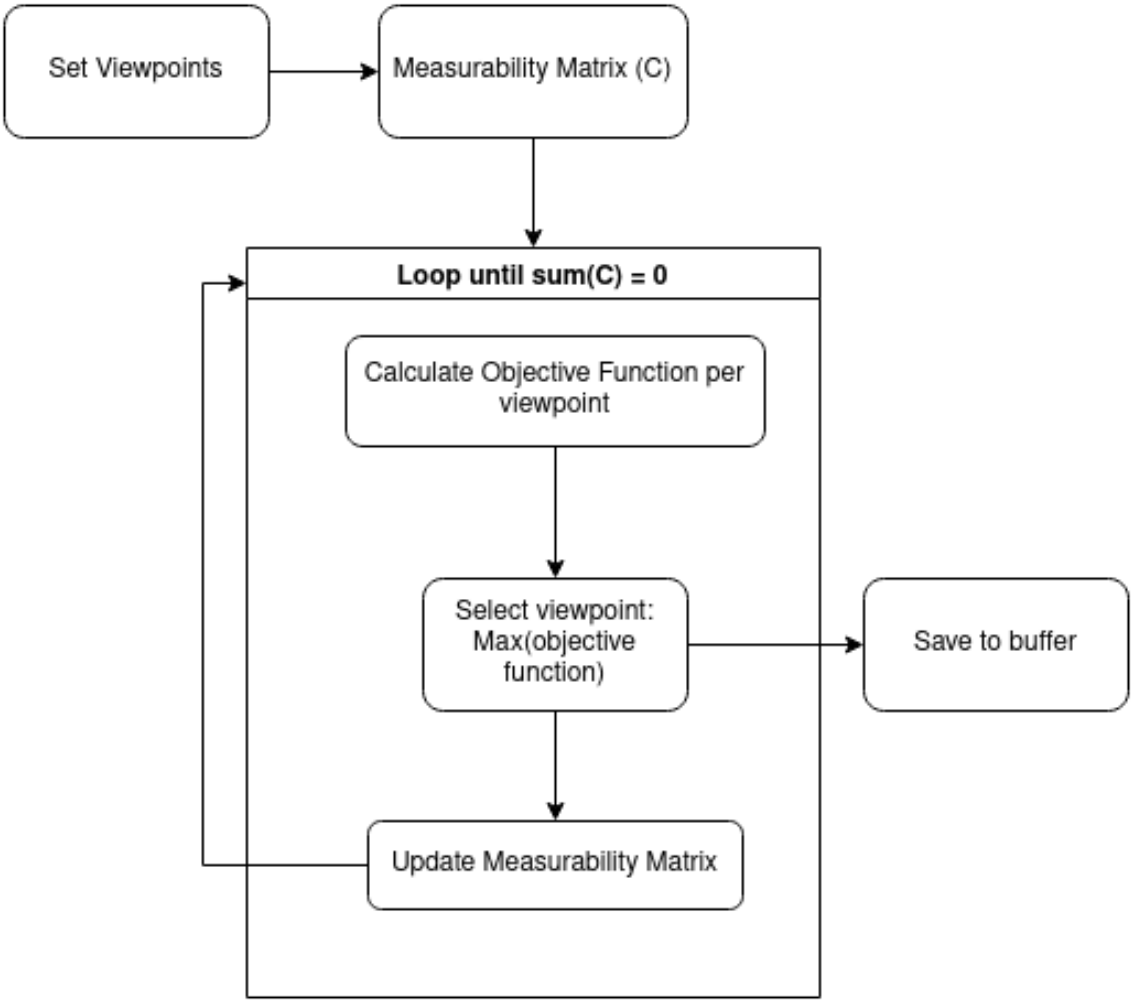

Figure 3.3: Algorithm for Method 1 (Greedy Area)

3.2.2.2 Method 2 (Weighted Greedy Features)

This second method shares many similarities with the previous one, the main difference is on the objective function it uses. On method 1 the area was the only feature used, but for this method others were added. Besides the area, the planarity, sphericity, ominivariance, anisotropy and change in curvature were introduced. These features were computed by sampling a uniform point cloud from a cropped mesh. This cropped mesh is composed by the set of measurable faces from a given

viewpoint. Principal Component Analysis (PCA) is applied to the sampled point cloud and from the eigenvalues, ($\lambda_1 > \lambda_2 > \lambda_3$), the mentioned features were computed as in [7]:

$$
\begin{aligned}
Planarity: &\quad P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1} \\
Sphericity: &\quad S_\lambda = \frac{\lambda_3}{\lambda_1} \\
Omnivariance: &\quad O_\lambda = \sqrt[3]{\lambda_1 \lambda_2 \lambda_3} \\
Anisotropy: &\quad A_\lambda = \frac{\lambda_1 - \lambda_3}{\lambda_1} \\
Change\ of\ curvature: &\quad C_\lambda = \frac{\lambda_3}{\lambda_1 + \lambda_2 + \lambda_3}
\end{aligned}
$$

The objective function for each viewpoint is simply a weighted sum of the features calculated from the measurable faces of that viewpoint, see equation 3.1.

$$Ob(j) = \vec{W} \cdot \vec{F}_j \tag{3.1}$$

where $\vec{W}$ is an array of coefficients, $\vec{F}_j$ is the feature vector (of size 6) from viewpoint j and $\cdot$ is the dot product. Since all features, except for the area, are in the same order of magnitude, only this one was normalized, dividing it by the scanning surface area measured from the set of viewpoints.

The same greedy approach was used, thus selecting the viewpoint that maximizes the objective function. The measurability matrix was updated as in Method 1, and for the next iteration the features had to be recomputed since some viewpoints had their measurable faces changed. The diagram in figure 3.3 also holds for Method 2.

The reasoning for this objective function is that the search for viewpoints can be optimized by preferring regions with some given features. The feature importance is controlled by the weighting vector $\vec{W}$ and its optimal values were searched by the PSO algorithm. This algorithm tries to minimize a cost function, which for this problem is the number of necessary viewpoints which assumes discrete values only.

3.2.2.3 Method 3 (Simulated Annealing)

The third method uses the SA, a stochastic optimization algorithm, to find a subset of viewpoints, B, that covers the same surface area as the original set, A. The search begins by considering an initial number of viewpoints, n_0, which are randomly chosen over the entire set of viewpoints, A. The random subset of viewpoints defines the current solution, $Curr_{sol}$, which is updates to generate a new solution that will be evaluated. This new solution, New_{sol}, is generated by swapping a certain number of viewpoints, n_{swap}, from the current solution with the unused viewpoints, Δ, obtained by the difference between A and B(see equation 3.2).

$$\Delta = A \backslash B \tag{3.2}$$

The new solution is evaluated through a cost function which computes its missing area (equation 3.3).

$$Cost(New_{sol}) = 1 - \frac{Area(New_{sol})}{Area(A)} \tag{3.3}$$

where $Area()$ is a function that computes the measurable area of a set of viewpoints by summing the area of the non repeated measurable faces from all viewpoint in the given set. Since New_{sol} is a subset of A, its measurable area cannot be greater than the measurable area of Set A, so the cost function is bounded between 0 and 1. After evaluating the new solution, we can either replace the current solution with it or just discard it. If this new solution is better than the best solution, $Best_{sol}$, (during the optimization the best solution is tracked and stored) we make the replacement, otherwise we must calculate the metropolis acceptance criterion (MAC) as in equations 3.4 and 3.5.

$$MAC = exp^{\frac{diff}{T_i}} \tag{3.4}$$

$$diff = Cost(New_{sol}) - Cost(Curr_{sol}) \tag{3.5}$$

where T_i is the temperature at iteration i.

The MAC number is compared to a random number sampled from a uniform distribution from 0 to 1. If the random number is lower than MAC then the new solution becomes the current solution and a new iteration starts again, otherwise, the current solution remains unchanged. The temperature, used in MAC, follows an exponential decay profile, so, initially the temperature is high and the MAC numbers are also high, close to one. This causes the new solutions to be likely accepted and become the current solution even though the cost function worsened. The algorithm runs through a defined number of iterations after which the best solution (lowest cost/missing area) is returned. One could only accept a given solution if its cost was zero, meaning the subset of viewpoints covers the exact same area as the original set. However the constrains were relaxed, and a valid solution had to have a missing area equal to or less than the last viewpoint added in Method 1. So, Method 1 has to be run before Method 3, not only to define this missing area for the validation criterion, but also to estimate a good initial number of viewpoints, n_0 which have to be less than the result of Method 1. n_0 can be reduced until the optimization process fails to find a valid subset of viewpoints. Regarding the number of viewpoints to swap per iteration, n_{swap}, this starts at a higher number than 1 and decreases exponentially but it is never smaller than 1. This strategy allows for a higher exploration at the beginning.

Both the temperature and the number of viewpoints to swap follow an exponential decay as in figure 3.6.

$$InitialValue * exp^{(-decay * iteration)} \tag{3.6}$$

The steps of the algorithm are summarized in the diagram shown in figure 3.4.

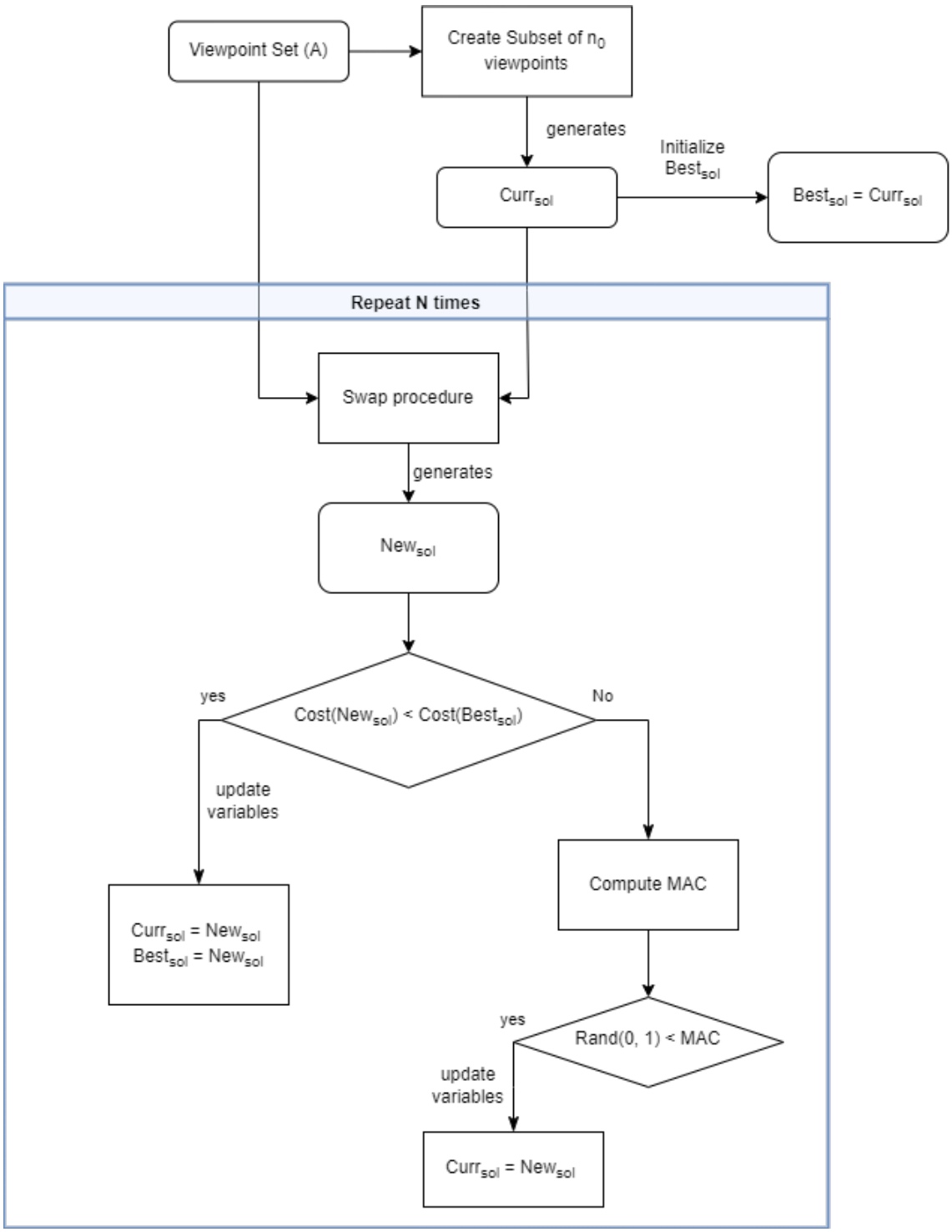

Figure 3.4: Algorithm for Method 3 (Simulated Annealing).

3.2.3 Viewpoints sorting

After having a subset of viewpoints, these had to be placed in a sequence that the robot's end effector should follow. To find an optimal viewpoint sequence that minimizes traveling distances, the Ant Colony Optimization (ACO) was used. This algorithm is one of the many solutions for the Travelling Salesman Problem (TSP) which, according to Kochenderfer and Wheeler [8, pp.354-359], holds satisfactory results when compared to other methods.

3.2.3.1 Compute Distance matrix

The input for the ACO is a distance matrix, $D(i, j)$, which, in this problem, is a symmetric matrix with a zero diagonal, and registers the distance/cost that it takes to move from viewpoint i to j. This distance metric could be viewed as the euclidean distance of a straight line connecting the two viewpoints. However it would produce misleading distances if the straight path intersects the mesh. So, the distance considered was the euclidean distance of a path that does not intersect the mesh. Such path was found using the Dijkstra's Shortest Path Algorithm. To use this algorithm, an occupancy grid was computed, classifying a cell as free or occupied (if it was inside the mesh), and the occupied cells would not be used by Dijkstra's algorithm while creating the shortest path. The algorithm only uses the cells in the grid as possible points for the path, so the more compact the grid is, the smoother the generated path will become, but its computation time will also increase. The grid is built upon a parallelepiped, that fits the mesh and uses its coordinate frame, and then is dilated in all faces by the distance d_{scan}. Afterwards, several points are sampled along the X, Y and Z direction of the parallelepiped, and it is checked whether each is outside (free cell) or inside (occupied cell) the mesh. When evaluating a path from two viewpoints it is very unlikely that their origin will coincide with a cell in the grid, so the closest cell to the viewpoint's origin is considered instead. This consideration will result in paths that do not start and end in the viewpoint's origin but with an enough dense grid such differences can be considered negligible.

Due to the properties of the distance matrix above mentioned, only $n(n-1)/2$ distance values need to be computed. Each computation needs to set a start and goal viewpoint. However, the start and goal coordinates are not the viewpoints' origins but their nearest cell in the grid. Then, Dijkstra's algorithm determines the optimal grid points that compose the path that connects the start and goal points. The integrated euclidean distance over the calculated path is the result that goes into the distance matrix.

3.2.3.2 ACO

The distance matrix is the input to the ACO which requires several hyperparameters that control the algorithm's performance [1]:

- Ants- Number of ants used for the optimization;
- Iterations- Number of iterations;

- α- Parameter related to the weight of the pheromone concentration in the probability function;
- β- Parameter weighting the relative importance of heuristic information in the probability function;
- ρ- Pheromone evaporation rate, where $\rho \in]0,1]$, measures the information that is to be transported to the next iteration;
- q- Parameter weighting the quantity of pheromone to be deposited in each component of the solution.

The output of the ACO algorithm is the sequence of viewpoints as well as the total distance of the sequence.

3.2.4 Mask generation

The camera mounted on the robotic platform will capture parts of the object that respect the criteria established in 3.2.1 and other parts that will not. To avoid processing unnecessary parts (during the classification and detection of surface defects) a binary mask was created and when multiplied by the original image we are left with the part of the image that corresponds to the measurable faces associated to that viewpoint.

The mask is created by shooting rays from the viewpoint origin and within the camera's frustum to a cropped mesh that only contains the measurable faces of that viewpoint. Each ray has a correspondent pixel so, if that ray collides with the mesh that pixel is marked as 1, otherwise as 0. The generated mask was then smoothed using a Gaussian filter, followed by an Otsu threshold to binarize the mask again.

3.3 Software libraries

The software developed in this thesis was programmed in *Python* with the usage of several open-source libraries among which the most relevant are:

- *Open3D* [18]- This open-source library was used to handle the visualization of point clouds and meshes. Some utility functions were also used like, sampling a uniform point cloud from a mesh object, ray-casting and ray-tracing techniques;
- *PyMesh* [6]- This open-source library was used to remesh the original mesh file, to compute the mesh normals and areas and intersect meshes;
- *dijkstra3d* [3]- This library was used to compute Dijkstra's shortest path algorithm;
- *ACO* [1]- This open-source library was used to solve the Ant Colony Optimization algorithm given a distance matrix;

- Nevergrad [15]- This open-source library was used to run the Particle Swarm Optimization.

Chapter 4

Results

This chapter presents the results obtained, using the mesh illustrated in 3.1 b), for each step explained above; Viewpoints generation pipeline 3.2.1, Viewpoints selection 3.2.2, Viewpoints sorting 3.2.3 and Mask generation 3.2.4.

4.1 Generated viewpoints

To generate a set of viewpoints, the camera and scanning parameters, were defined as:

- scanning distance, d_{scan} = 20mm;
- horizontal angular field of view (HAFoV) = 120°;
- vertical angular field of view (VAFoV) = 120°;
- depth of view range (DoV) = [18mm, 22mm];
- glance angle threshold = 60°;
- sphere radius of camera model = 4 mm;

The camera parameters, HAFoV, VAFoV, DoV, sphere radius of camera model, were not taken from a real camera, instead were made up values (although some realistic), enough to draw conclusions from the developed methods.

Figure 4.1 illustrates, through color, the evolution of the valid faces through steps 6) to 8) in 3.2.1, for a single viewpoint. The subfigure a) illustrates the camera model (spherical mesh) in grey and the viewpoint direction can be seen in b) and c) by the purple arrow. Step 6 results in a considerable amount of valid faces (painted as green) that are drastically reduced using the DoV criterion (step 7). The impact of the glance angle criterion (step 8), turns out to be minor for this viewpoint. Besides the green (valid faces) and red colors (invalid faces) in figure 4.1, some faces

are painted yellow and exist in the greed-red interface. This occurs due to the way *Open3D* library colors mesh objects. Instead of giving a color to a face, in *Open3D* we must attribute a color to the vertices, so, yellow comes from an interpolation between red and greed vertices of the same face.

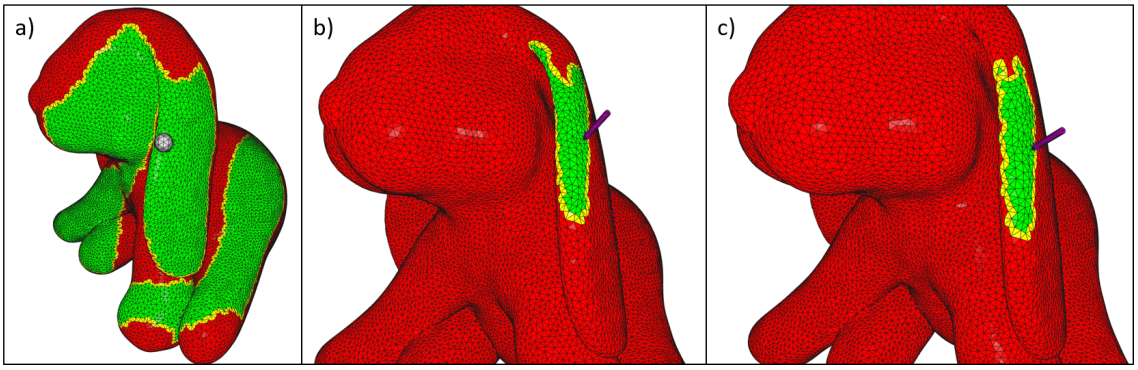

Figure 4.1: Impact of the criteria in 3.2.1 on the number of valid faces. a) step 6 – Compute visible triangles; b) step 7 – DoV criteria; c) step 8 – glance angle criterion; Green: valid faces; Red: invalid faces

4.2 Impact of number of seed faces

The subset of viewpoints found by Method 1, 2 and 3 should be influenced by the amount of viewpoints in the original set, which in turn depends on the number of seed faces used.

To evaluate the impact of the number of seed faces on the subset of viewpoints, we used Method 1, the simplest and fastest of the three methods. The number of seed faces was gradually increased, and the resulting set of viewpoints was reduced by Method 1, whose subset was evaluated. At the same time, the measurable surface area was also evaluated while increasing the seed faces.

Figure 4.2 illustrates graphically the relation between the amount of seed faces and, 1) the number of viewpoints on the subset computed using Method 1, 2) the measurable area of the subset. The array of numbers of seed faces (normalized by the total number of faces in the mesh) used to create the plot in 4.2 was $[0.1\%, 0.2\%, 0.3\%, 0.5\%, 1\%, 5\%, 10\%, 15\%, 30\%, 50\%]$. The array is comprised of mostly small numbers because the measurable area quickly plateaus. This can be seen in figure 4.2 where for $x >= 5\%$ the measurable area is practically the same. So, there is no point going beyond this number of seed faces, as one only increases the computation time without gaining information (surface area).

The following subsections will present the results for the several developed methods using a normalized number of seed faces of 10% which correspond to 4552 faces, able to measure 92.22% of the total surface area. Figure 4.3 depicts how this set of viewpoints (represented only by an arrow) is distributed over the space, and in figure 4.4 we can see only the visible mesh from the chosen set of viewpoints.

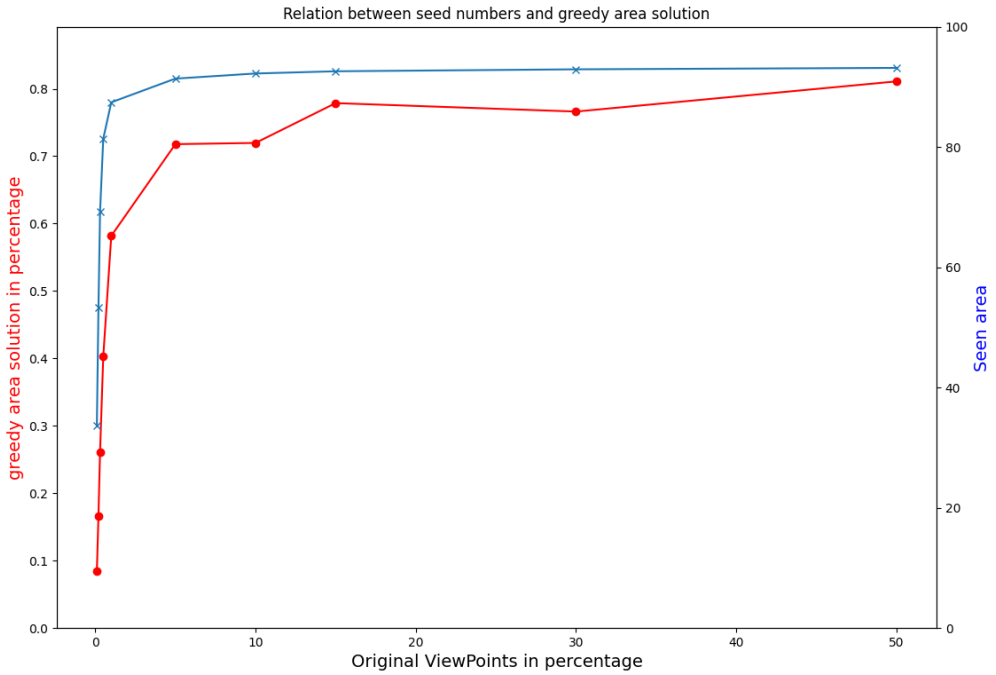

Figure 4.2: Relation between the seed numbers and the solution of Method 1. X axis: Percentage of number of seed faces normalized by the number of total faces of the mesh; Y axis left: Percentage of number of viewpoints computed using Method 1, normalized by the number of total faces of the mesh; Y axis right: Percentage of measurable area from method's 1 solution, normalized using the total surface mesh area.

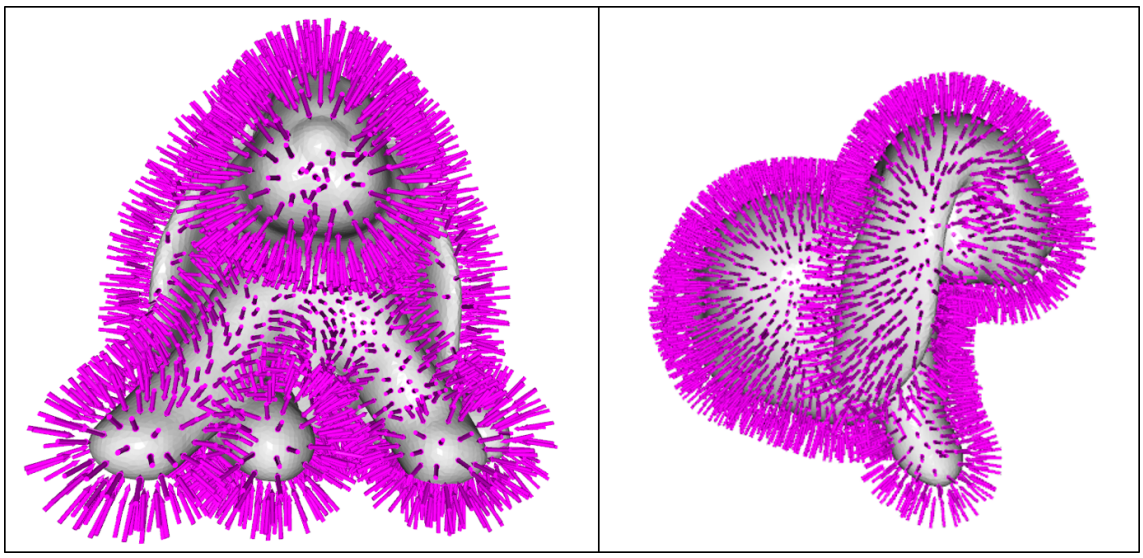

Figure 4.3: Two different views of the mesh along with the 4552 viewpoints represented by purple arrows.

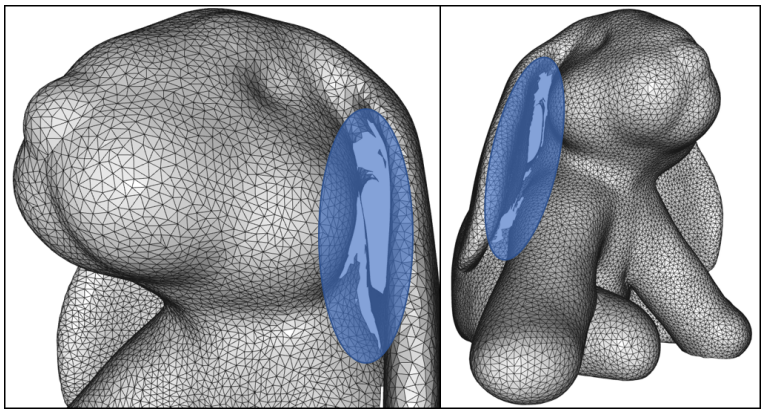

Figure 4.4: Visible mesh from a set of viewpoints. The blue regions delimit unseen faces of the mesh.

4.3 Viewpoints selection

4.3.1 Method 1: Greedy area

From the set of 4552 viewpoints, the greedy area approach only needed 402 to view the same surface area, a 91.17% reduction. Figure 4.5 allows to evaluate the method's decisions at each iteration. The top plot illustrates the normalized area added per iteration, the middle one is the normalized missing area to scan and the bottom plot shows how many times faces have been seen using an histogram.

Since the criteria to choose a viewpoint is based on its measurable area (choose the highest), it is normal to see a monotonic decrease in the top plot. The solution found presents a significant redundancy since there are faces seen up to 14 times and approximately $\frac{1}{3}$ of the visible surface area is scanned twice. As the algorithm progresses, the remaining area to scan slowly converges to zero as one can see from the middle plot.

A more visual interpretation of the bottom plot in figure 4.5 is found in figure 4.6, where each face has a color that corresponds to a certain number of times the face has been scanned. In the colormap d) there is a color missing, corresponding to when a face was seen zero times, which is the black, and only appears under the ears of the bunny, so it is never seen in figure 4.6.

4.3.2 Method 2: Weighted Greedy Features

This method relies on the PSO algorithm to find an optimal array of coefficients $\vec{W}$ that minimizes the number of viewpoints on the found subset. The elements of this array were bounded between 0 and 1 and the algorithm was run for 500 epochs. The cost value (number of viewpoints of the subset) for each iteration can be seen in figure 4.7. PSO is unable to produce solutions that decrease the cost along the optimization process. Instead, it generates solutions with a constant cost, around 500 viewpoints, with some outliers ranging from 1500 and 3500 viewpoints. The best subset is composed of 428 viewpoints and the decisions of the greedy selection process for this solution can be seen at figure 4.8. In the second plot, counting from the top, the missing area still

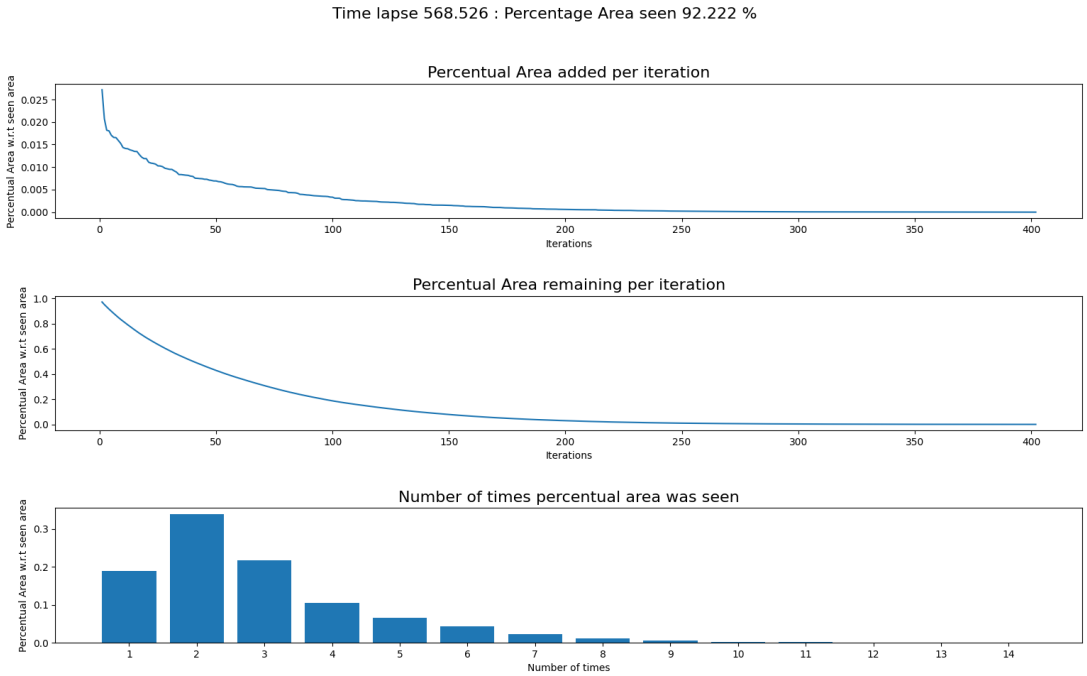

Figure 4.5: Graphical representation of the results of Method 1. Top: normalized area added per iteration; Middle: normalized missing area to scan; Bottom: histogram of how many times areas have been seen.

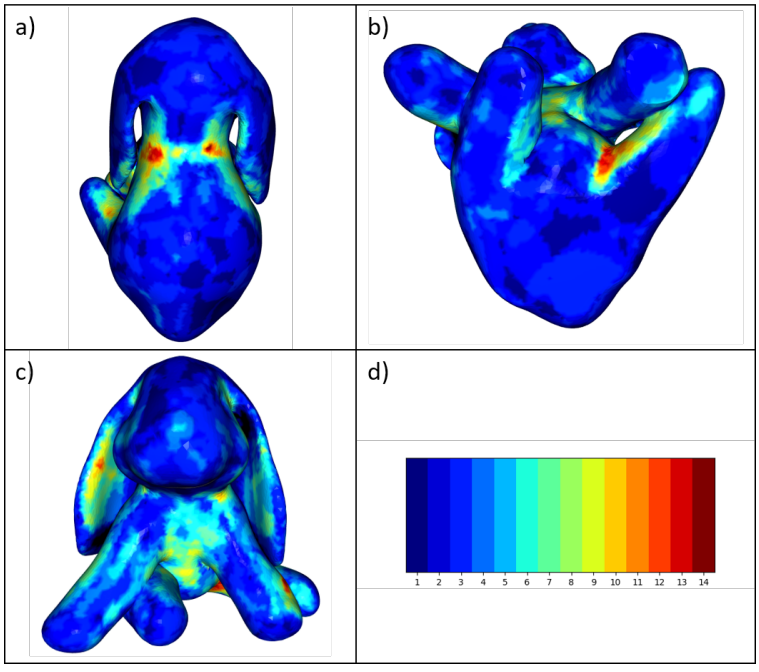

Figure 4.6: Colored representation of the number of times a face has been seen, produced by the solution of Method 1; a) Top view; b) Bottom view; c) Front view; d) mapping between color and the number of times a face has been scanned.

decreases slowly converging to zero, like in Method 1. In the bottom most plot, only the area and omnivariance features decrease as the greedy selection process continues.

The $\vec{W}$ vector responsible to produce the subset of 428 viewpoints was:

$$W_{area} = 0.0; W_{plan} = 0.552; W_{sphe} = 0.722$$
$$W_{omni} = 0.488; W_{ani} = 0.457; W_{curv} = 1.$$

The redundancy of the best solution can be better understood through figure 4.9 that depicts in a colors, the number of times each face was seen. The same information is presented in the third plot from figure 4.8 using an histogram.

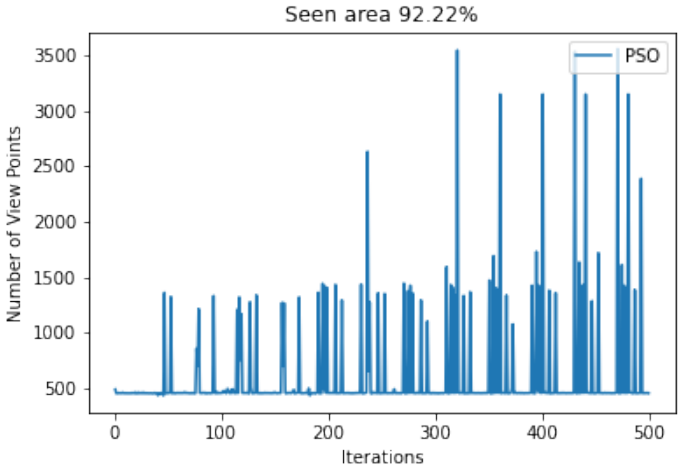

Figure 4.7: PSO cost evolution.

4.3.3 Method 3: Simulated Annealing

The success of the SA optimization process is highly dependent on both the schedule for the temperature and the number of viewpoints to swap. Through trial and error, the *decay* parameter (see equation 3.6), for both schedule profiles (temperature and number of viewpoints to swap) was found to be $5e-05$ (see figure 4.10) in order to have a converging optimization. As for the initial temperature and initial number of viewpoints to swap, those were set to 10 and 4 respectively. Regarding the number of iterations, a fixed number of 1 million was used in all experiments.

Figure 4.11 illustrates how the cost function evolves throughout the optimization for four different initial numbers of viewpoints (n_0), 401, 392, 382 and 302. All these values are less than 402, which was the amount of viewpoints found by Method 1. Except for the bottom right plot in figure 4.11, the other optimizations were able to find a subset of viewpoints with a zero missing area (event marked as a black circle) using the correspondent n_0. From the trials done, it was possible to reduce 20 viewpoints (solution with 382 viewpoints) from the 402 found by Method 1. Since the optimizer failed with 302 viewpoints, through more trials one could probably find a

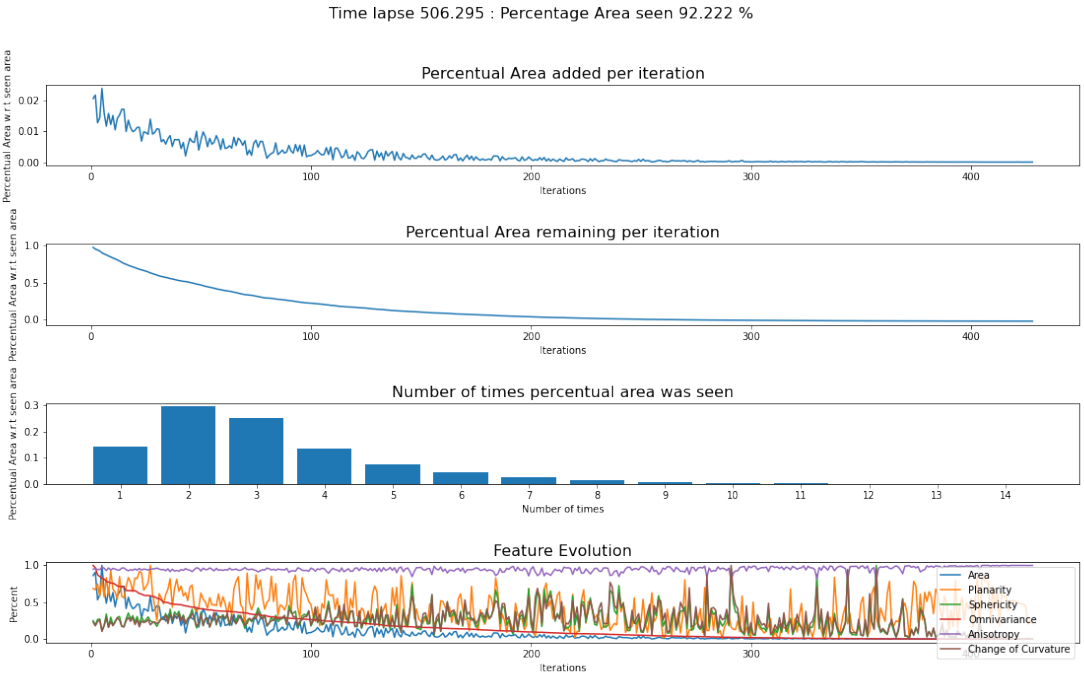

Figure 4.8: Graphical representation of the results of Method 2. First plot: normalized area added per iteration; Second plot: normalized missing area to scan; Third plot: histogram of how many times areas have been seen; Fourth plot: Max Normalized Features added per iteration.

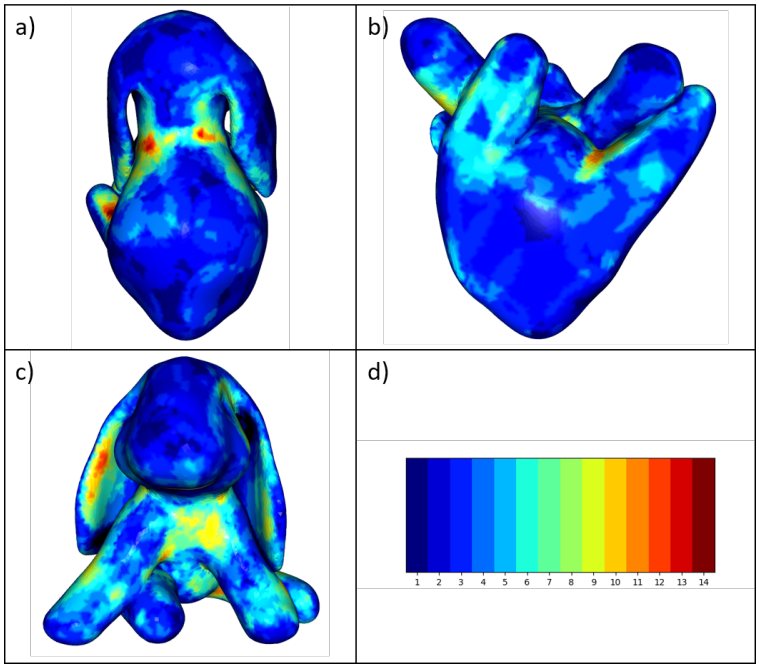

Figure 4.9: Colored representation of the number of times a face has been seen, produced by the best solution found by PSO. a) Top view; b) Bottom view; c) Front view; d) mapping between color and the number of times a face has been scanned.

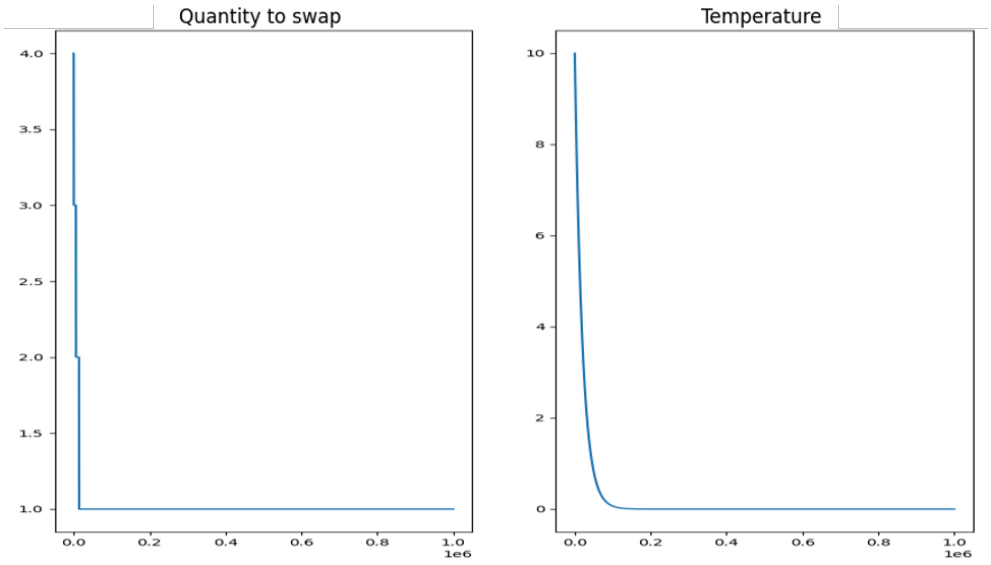

Figure 4.10: Exponential decay profile for the temperature and the number of viewpoints to swap (n_{swap}), using a *decay* of $5e-05$. Left: Number of viewpoints to swap (since this quantity is an integer they had to be quantized); Right: Temperature profile.

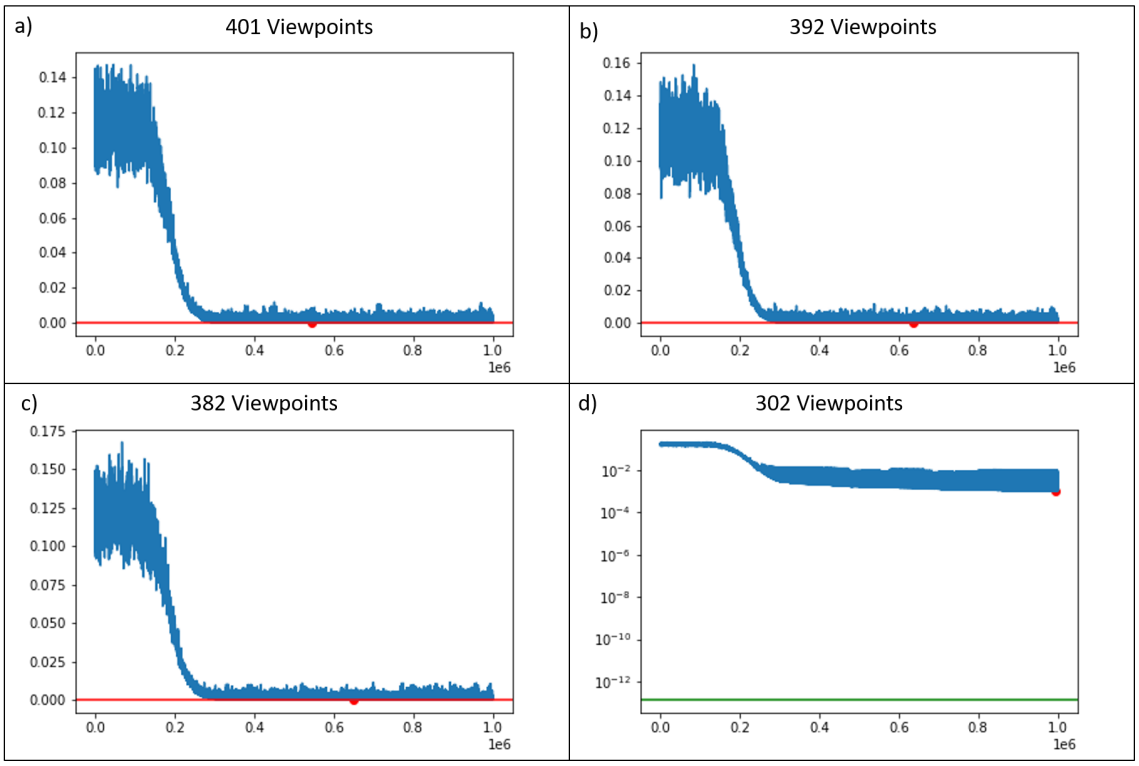

Figure 4.11: Evolution of the cost function (missing area) for four different initial number of viewpoints, n_0. Blue line: cost function of the New_{sol} generated at each iteration; Red dot: Minimum of the cost function; Red line; Zero cost function (Zero missing area); Green line; Acceptability criterion (added area by the last viewpoint of method 1).

valid subset that used a n_0 between 382 and 302. Still regarding figure 4.11, the blue line represent the cost function of the new solution generated at each iteration thus why it is very noisy. The plot for n_0 equal to 302 is presented at a log scale in the y axis to have a more clear visualization of the trend. For this case, the optimizer failed to generate a valid subset of viewpoints since the green line (missing area criteria) was not reached.

To analyse the redundancy of the best solution found (382 viewpoints) the histogram with the number of times the area has been seen was computed and it can be viewed in figure 4.12. The visual distributions of these areas are depicted in figure 4.13.

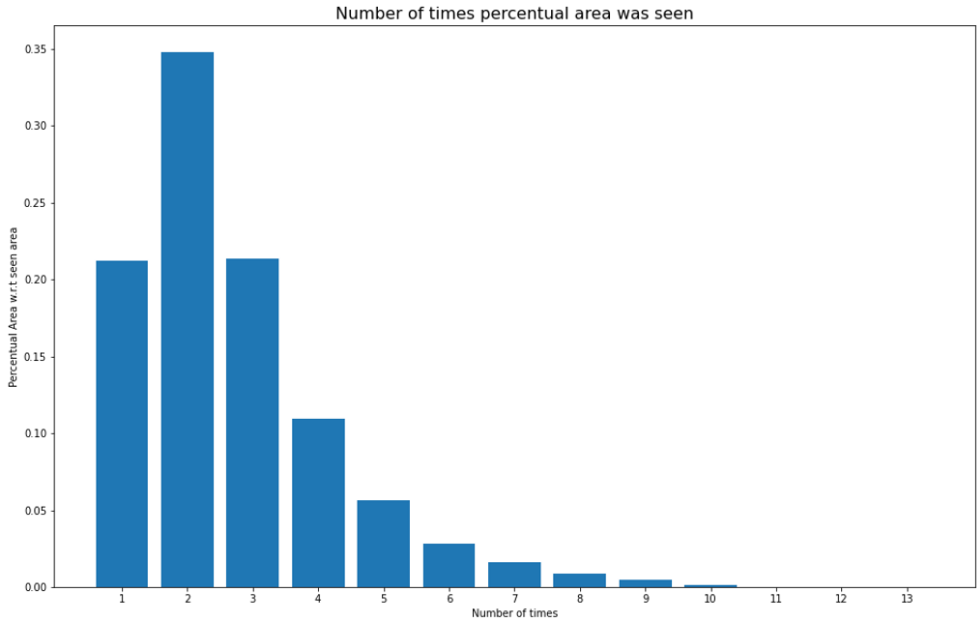

Figure 4.12: Histogram of how many times areas have been seen. Computed for the solution with 382 viewpoints

4.4 Viewpoints sorting

The best subset achieved, found by Method 3, was used to test the viewpoint sequencing functionality. First, to obtain the occupancy map, the volume fitting the mesh is discretized into a cube of $200 \times 200 \times 200$ points (see red points in figure 4.14). Figure 4.14 a) and b) depicts a minimal path generated by Dijkstra's Shortest Path algorithm, between two viewpoints. As one can see, the generated path does not intersect the occupied points, and finds a path that passes through a hole that exists between the bunny's paws. In subfigure c) it is evident the approximation error one is doing by considering the nearest grid point from the viewpoint's origin as the start or goal points of the path. Since the used subset has 382 viewpoints, the distance matrix, depicted in figure 4.15 as a heatmap, needs 72771 distinct distances. To calculate the viewpoint sequence, the ACO algorithm was used with the following parameters, as recommended in the book [8, pp.354-359] and by Pettersson and Johansson in [14]:

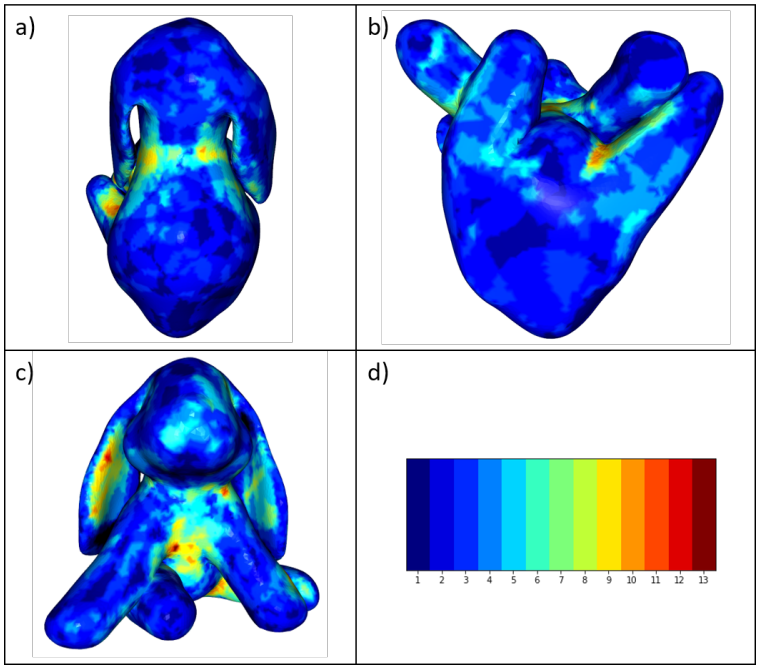

Figure 4.13: Colored representation of the number of times a face has been seen, produced the best solution found by SA; a) Top view; b) Bottom view; c) Front view; d) mapping between color and the number of times a face has been scanned.

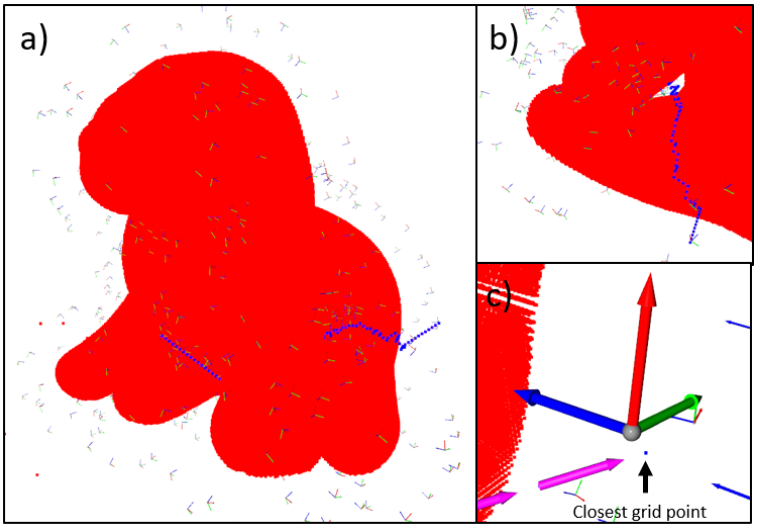

Figure 4.14: Example of a path generated by Dijkstra's Shortest Path. a) and b): Example of a path generated by the Dijkstra's Shortest Path algorithm using different perspectives; c): Difference between a viewpoint's origin and its closest grid point; Red points: Represent occupied points; Blue points: Grid points that make up the generated path; Frames: Represent the viewpoints' frames from the subset generated by Method 3.

- Ants = 40 ($\approx$10% of the number of viewpoints);

- Iterations = 200;
- $\alpha = 1$;
- $\beta = 10$;
- $\rho = 0.5$;
- $q = 10$.

The final sequence of viewpoints, had a total path distance of 9045.5mm, and the generated path can be seen in figure 4.17. The evolution of the best cost during the optimization is depicted in figure 4.16.

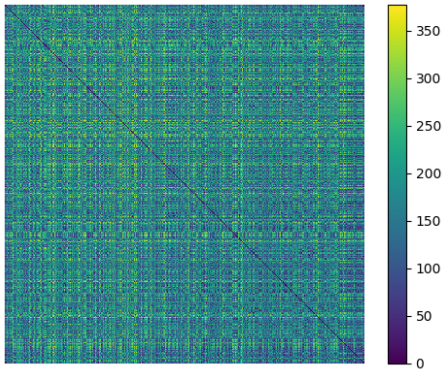

Figure 4.15: Heatmap representation of the distance matrix. The distance is measured in mm

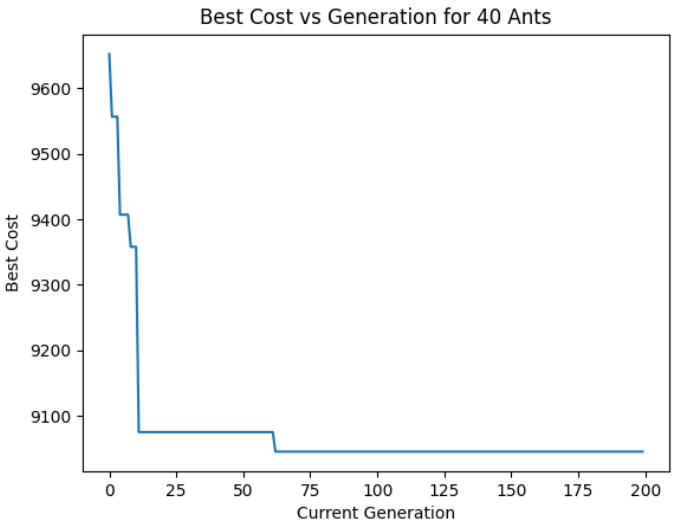

Figure 4.16: Evolution of the best cost achieved during the ACO algorithm iterations.

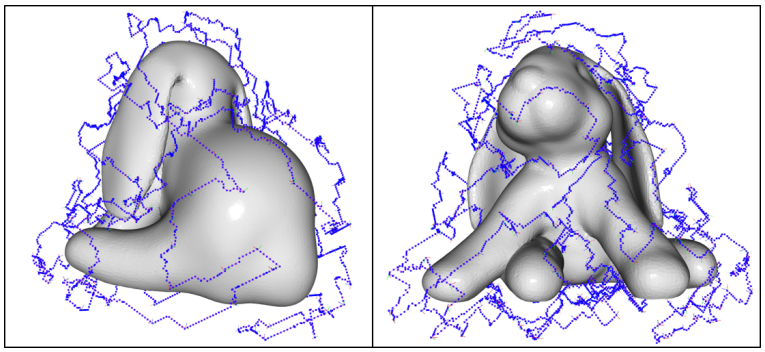

Figure 4.17: Generated sequence by ACO algorithm, united through the shortest path created by Dijkstra's.

4.5 Mask generation

Each viewpoint has a known position and orientation w.r.t the mesh so one can simulate what a camera would see if placed in the viewpoints' pose. However, only the valid part of the mesh (measurable faces) is of interest for a later image analysis stage. Figure 4.18 b) depicts an example of a mask containing only the valid faces of the mesh seen from the viewpoint marked in figure 4.18 a).

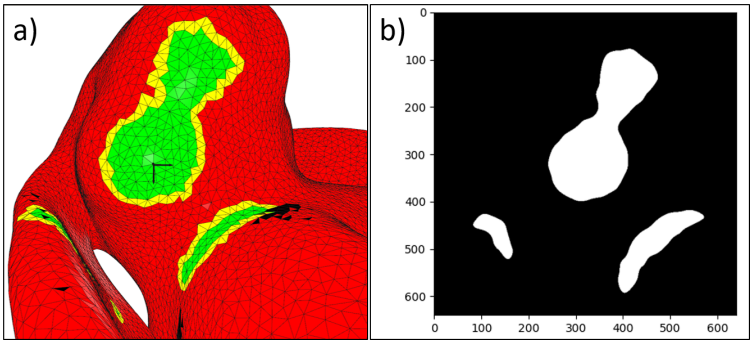

Figure 4.18: Example of a final mask from a given viewpoint. a): Viewpoint's valid faces in green; b) Mask created from a virtual camera placed in the viewpoints' pose with the camera properties specified in 4.1.

Chapter 5

Discussion

5.1 Generated viewpoints

The camera and scanning parameters chosen in 4.1 were used to generate sets of viewpoints. Intuitively, one can conclude that by increasing d_{scan} or the angular field of view there are more visible faces. Those are then filtered using the DoV and glance angle threshold criteria, being the first one the most impactful in the example shown in figure 4.1. The sphere radius has to be chosen in accordance to d_{scan}. A higher radius than d_{scan} will probably result in collisions with the mesh, whatever viewpoint is considered.

5.2 Impact of number of seed faces

Figure 4.2 summarizes the relation between the amount of seed faces and the resulting subset of viewpoints (chosen using Method 1) and the visible area. As the number of seed faces increases, the visible area also increases abruptly and quickly reaches a plateau, around $x = 5\%$, so, from this point further, little information of the mesh is gained. The number of viewpoints computed by Method 1 also increases quickly with the number of seeds however at $x = 5\%$ the increase decelerates substantially. After this point, there is a slight increase in the number of viewpoints from Method 1, because the increase in seed number allows to see difficult/low accessibility regions which have a small area. Method 1 prioritises regions with a large area and leaves small ones for last. In fact, these regions can only be captured by several viewpoints increasing its final number. The point $x = 5\%$ could be considered a suitable point to test the other methods since a plateau was reached for the measurable area and a small set of viewpoints is required, thus reducing the computational cost. However, the seed number chosen to test all the methods was 10%, because increasing seed faces will produce a higher set of viewpoints. This might help Method 3 to generate a better solution, since it has more viewpoints to chose from. The increase in seed numbers does not impact the solution of Method 1 since for $x = 10\%$ and $x = 5\%$ its results do not differ much both in terms of chosen viewpoints and visible area.

5.3 Comparison of the viewpoints selection methods

Method 1 resulted in 402 viewpoints (a 91.17% reduction) from the 4552 initial set of viewpoints. As this algorithm selects viewpoints, the remaining area decreases almost asymptotically to zero. This is a result from small gaps, that are created as the viewpoints are being selected, and require extra viewpoints to cover those small regions. This phenomenon is also reported by Tarbox and Gottschlich [17].

Method 2 failed to find a set of coefficients that improved the results of Method 1, reaching a minimum of 428 viewpoints (a 90.59% reduction from the initial set and a 6.5% increase from Method 1 solution). Since Method 1 is a particular case of Method 2, all the coefficients in $\vec{W}$ are zero except for the area's coefficient which is 1. So, one would expect the PSO to find at least this set coefficients. However, it was not able to do so. Probably the discrete cost function is not suitable for this kind of optimizers and a different one, more tailored for this kind problems, had to be used instead. Moreover, the search space can be too large so using less features could improve the algorithm.

As for Method 3, the results overcame Method 1, solving the problem with 382 viewpoints (a 91.61% reduction from the initial set and a 5% reduction from Method 1 solution). This number could probably be reduced even further if more experiments, with a lower number of viewpoints, were performed. In all the successful experiments a valid subset of viewpoints is found around the 600k iterations and the evolution of the cost function is very similar among experiments having a sudden drop around the 200k iteration, which coincides with the moment when the temperature reaches the plateau and converges to zero.

Table 5.1 summarizes the results obtained by each method.

Regarding the redundancy, all methods result in most of the area being seen twice, close to 35% of the total seen area, and some areas are seen up to 13 to 14 times depending on the method. Visualising the number of times faces are seen, using a colored representation from 4.6, 4.9 and 4.13 one can see that the most scanned regions are the inner side of the ears, the nape, the chest and the paws. Redundancy is thus fairly the same regardless of the method. Even though this event might lead to extra computation in the image processing stage, it can be useful to check the confidence in a detection since the same area can be analysed multiple times in different orientation resulting in different outputs, which can be then combined.

The code was developed in Python and execution time was not seen as a priority, since this is aimed to be an off-line tool. However, the number of experiments made were limited due to the long computation time. Using 4552 viewpoints as the initial set, Method 1 takes around 10 minutes to complete, the optimization of Method 2 takes around 3.5 days to complete its 500 iterations, and finally Method 3, takes up to 8 hours to complete 1 million iterations, see table 5.2. For reference, the computations were performed on an Intel Core i7-9700 CPU @ 3.00GHz.

5.4 Viewpoints sorting

Disktra's shortest path was used to provide a more realistic distance measurement between the viewpoints, taking into account the mesh as an obstacle to avoid. However, a more correct approach would be to constrain the generated paths so that they could be executed by a robotic arm. Figure 4.14 depicts an impossible path for a robot to follow because it passes through a small hole in the mesh. In the same figure one can also observe the error that exists between a viewpoint's origin and the nearest grid point, considered by the shortest path algorithm. This error can be reduced using a refined grid at the cost of more computation time. This discretization resulted in a rare event of two viewpoints having the same nearest grid point which led to a distance of 0 in a non-diagonal position in the distance matrix. This event caused the ACO to crash, so a small number ($1e-5$) was added to the distance matrix. The ACO algorithm reaches a best solution (9045.5mm), which is suboptimal, around 60 iterations and takes around 2 hours to complete the 200 iterations. A better solution could probably be found by tuning/optimizing the hyperparameters of the ACO algorithm.

Table 5.1: Summary of the results of the viewpoint selection methods.

	Method 1	Method 2	**Method 3**
Number of Viewpoints	402	428	382
Reduction w.r.t number of viewpoints	91.17%	90.59%	91.61%
Reduction w.r.t Method 1 solution	0.0%	-6.5%	5.0%

Table 5.2: Computational time for the several steps of the framework.

	Viewpoint Generation	Method 1	Method 2	Method 3	Distance Matrix	ACO
Time to compute (min)	5	10	5040	540	120	60

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The developed methodologies were proved to be functional and work for any closed mesh, being convex and/or concave, although unable to generate optimal results. It also generalizes to different camera and scanning parameters. Only one mesh was considered for the test, however it had enough complexity to test the full capabilities of the developed methods.

It was verified that the amount of seed faces necessary to create a set of viewpoints that is able to scan almost all possible visible area is rather small, only 5% of the total amount of faces of the mesh. These conclusions are drawn for the camera and scanning parameters used, different ones would change this number.

From the three methods developed to find a valid subset of viewpoints, Method 1 is the fastest due to its simplicity, and is able to reduce the amount of viewpoints by almost 92%. It should be used if time is a constraint and having the best subset is not mandatory. Regarding Method 2, even though it is a generalization of Method 1, using more features besides the area, the features weight optimization procedure is unable to find a smaller subset than Method 1. Method 3, that used the simulated annealing optimizer, was able to surpass Method 1, requiring 5% less viewpoints, being the best of the three methods. However the improvements in the results come with more computational cost when compared to Method 1. One could probably find a smaller subset if more tests were run, but the results computed already allowed to draw conclusions.

The ACO algorithm was able to solve the TSP generating a path of 9045mm which does not collide with the mesh. Although some portions of the trajectory cannot be executed by a robotic manipulator, the approach used to measure distance between pairs of viewpoints (Dijkstra's shortest path) is more realistic than considering the euclidean distance of a straight line between the viewpoints' origins.

6.2 Limitations and future work

One of the limitations of the proposed methods is the computational cost, so, increasing the performance of the code developed suites as a future development. Since only one mesh was used to test the algorithms, more extensive tests could be done, using different meshes with various geometries in order to draw more robust conclusions. As the optimization of Method 2 failed, more tests had to be done, with different optimizers and combinations of features. Regarding Method 3, other temperature profiles could be experimented, as they play an important role in the optimization stability and convergency point. Moreover, other stochastic optimization algorithms could be experimented and a parallelizable one should be preferred to reduce computational time. Another improvement that could be done is instead of using Dijkstra's shortest path to fill the distance matrix, one could integrate already the robotic manipulator in the loop, and use tailored path finding algorithms for manipulators that would compute a feasible and realistic path for the manipulator between pairs of viewpoints.

References

[1] Ant Colony Optimization. https://github.com/johnberroa/Ant-Colony-Optimization Accessed: 2021-10-01.

[2] Digital Transformation CoLab. https://www.dtx-colab.pt/ Accessed: 2022-1-05.

[3] dijkstra3d 1.12.0, Implementation of Dijkstra's Shortest Path algorithm on 3D images. https://pypi.org/project/dijkstra3d/ Accessed: 2021-10-01.

[4] Ducky The Lop Eared Bunny. https://www.thingiverse.com/thing:752379 Accessed: 2021-12-05.

[5] Pymesh, fixmesh.py. https://raw.githubusercontent.com/PyMesh/PyMesh/main/scripts/fix_mesh.py Accessed: 2021-10-05.

[6] PyMeshPyMesh — Geometry Processing Library for Python. https://pymesh.readthedocs.io/en/latest/ Accessed: 2021-10-01.

[7] R Blomley, M Weinmann, J Leitloff, and B Jutzi. Shape distribution features for point cloud analysis-a geometric histogram approach on multiple scales. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3):9, 2014.

[8] Mykel J Kochenderfer and Tim A Wheeler. *Algorithms for optimization*. Mit Press, 2019.

[9] Abhijit Kundu, Yin Li, and James M Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3559–3568, 2018.

[10] Claire Lartigue, Yann Quinsat, Charyar Mehdi-Souzani, Alexandre Zuquete-Guarato, and Shadan Tabibian. Voxel-based path planning for 3d scanning of mechanical parts. *Computer-Aided Design and Applications*, 11(2):220–227, 2014.

[11] Kairanbay Magzhan and Hajar Mat Jani. A review and evaluations of shortest path algorithms. *International Journal of Scientific & Technology Research*, 2(6):99–104, 2013.

[12] Fernando António Rodrigues Martins, Jaime Gómez García-Bermejo, Eduardo Zalama Casanova, and José R Perán González. Automated 3d surface scanning based on cad model. *Mechatronics*, 15(7):837–857, 2005.

[13] Eneko Osaba, Xin-She Yang, and Javier Del Ser. Traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics. *Nature-Inspired Computation and Swarm Intelligence*, pages 135–164, 2020.

[14] Lars Pettersson and Christoffer Lundell Johansson. Ant colony optimization-optimal number of ants, 2018. Dissertation, Available at https://www.diva-portal.org/smash/get/diva2:1214402/FULLTEXT01.pdf, Accessed on Jan-2022.

[15] J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. https://GitHub.com/FacebookResearch/Nevergrad, 2018.

[16] NJ Shipway, TJ Barden, P Huthwaite, and MJS Lowe. Automated defect detection for fluorescent penetrant inspection using random forest. *NDT & E International*, 101:113–123, 2019.

[17] Glenn H Tarbox and Susan N Gottschlich. Planning for complete sensor coverage in inspection. *Computer Vision and Image Understanding*, 61(1):84–111, 1995.

[18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.