Journal of Intelligent Manufacturing
https://doi.org/10.1007/s10845-023-02306-x

Optimization of image acquisition by automated white-light interferometers during the inspection of object surfaces

Björn Schwarze[1] · **Stefan Edelkamp**[1,2]

Received: 24 July 2023 / Accepted: 6 December 2023

Abstract

This paper considers the efficient quality assurance of diverse geometric objects through the use of a white-light interferometer, with a primary focus on minimizing the number of required image captures. The motivation behind such an algorithm stems from the extended recording times associated with various free-form sheet metal parts. Given that capturing images with a microscope typically consumes 30–40 s, maintaining high-quality assurance is imperative. A reduction in the number of images not only expedites part throughput but also enhances the economic efficiency. A unique aspect in this context is the requirement for focus points to consistently align with the part's surface. We formulate this challenge in a mathematical framework, necessitating a comprehensive literature review to identify potential solutions, and introduce an algorithm designed to optimize the image acquisition process for inspecting object surfaces. The proposed algorithm enables efficient coverage of large surfaces on objects of various sizes and shapes using a minimal number of images. The primary objective is to create the most concise list of points that comprehensively encompass the entire object surface. Subsequently, the paper conducts a comparative analysis of various strategies to identify the most effective approach.

Keywords Free-form surface · Automated inspection · Quality control of parts · View planning

Introduction

The quality assurance system is a key component of various companies' plans (Leopold et al., 2003). Components with freeform surfaces are used in a variety of industries, including aerospace, automotive manufacturing, mold making, and more. The flawless functionality of these products is significantly influenced by the geometric accuracy of the freeform surfaces (Zahmati et al., 2018). Therefore, quantitative measurement of surface topography is essential for precise surface processing. Yi et al. (2021) In this process, robot solutions are often used to cover the surface to be inspected. Automated inspection of freeform surfaces helps significantly reduce the mean time to error detection (Glorieux et al., 2020). Geometric testing on free-form surfaces is carried out using either contact or non-contact measurement methods (Zahmati et al., 2018). This work deals with the planning of a measurement using non-contact methods with an automated microscope. The planning of the measurement for capturing the images plays a crucial role, as without it, the images cannot be automatically acquired.

The coverage path problem involves determining the viewpoints and sequence from which the surface of the part should be measured. When planning the coverage path, multiple criteria need to be considered, including the complete coverage of target areas, as well as the resulting cycle time for the inspection task (Glorieux et al., 2020). Other boundary conditions may also arise, such as compliance with so-called focus points or something similar. In this case, numerous sheet metal components should guarantee a surface that is free from defects. Given the reflective properties inherent to these specialized metal components, ensuring damage-free production becomes highly imperative. If fingerprints,

✉ Björn Schwarze
schwabjo@fel.cvut.cz

Stefan Edelkamp
edelkste@fel.cvut.cz; edelkamp@ktiml.mff.cuni.cz

1 Faculty of Electrical Engineering, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague, Dejvice, Czech Republic

2 Department of Theoretical Computer Science and Mathematical Logic, Faculty of Mathematics and Physics, Charles University in Prague, Ke Karlovu 3, 121 16 Praha 2, Czech Republic

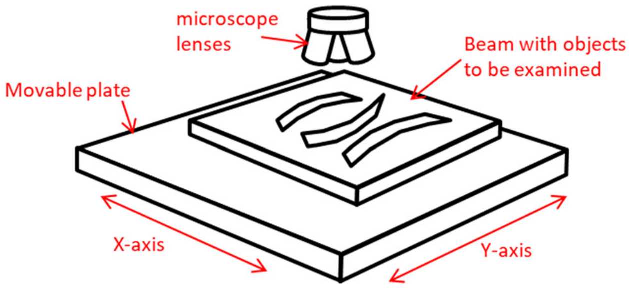


Fig. 1 Schematic illustration of microscope and carrier. The plate moves in the X and Y axes to move the carrier. This allows the microscope to take different images of the objects on the carrier

scratches or particles are found on the surface of the object, it can no longer be delivered. In this case it has to go into post-processing. To lower the costs of this quality assurance measure, an automated white light interferometer (WLI) is used for this application. There is also software logic for controlling the axes and receiving the image. In general, the metal objects are too large to be captured with just a single image. The image frame is always very small, as the resolution is very high to detect damage or particles in the micrometer range. That's why the metal object is displayed in partial images. The partial images are then combined into one image and evaluated. But before the automatic WLI can get started picking up the images, scheduling is needed. This involves strategizing the positions at which partial images will be captured. Since taking the images takes a comparatively long time in this scenario, it is necessary to keep the number of images required as low as possible. Therefore, a solution is required that calculates the minimum number of image positions from the geometry of the metal part and at the same time can cover the entire part.

An automated WLI that can move on three axes is employed to streamline the procedure. Figure 1 shows a sketch of the machine. A carrier is used to secure several objects so they do not need to be placed separately under the microscope by the operator. Sub-images are then created from the object at a specific resolution, which are used to analyze the object.

The objects must be recorded at a high resolution in order to capture even the smallest particles. In this case, each image has a resolution of 1336×1020 pixels. At this resolution, objects are too large to be captured in a single image. As a consequence, in order to capture an object in its entirety, a series of partial images must be created, as shown in Fig. 2. Then, partial images are merged to form a single large one. Since that the sheet metal object can have its surface curved in various ways, and due to the focussing process, creating an image using WLI may take up to 30 or 40 s. With 50–60 images required, capturing an entire object can take up to 25 min. These partial products are frequently requested in a variety of forms. Reduced image counts can be a valuable

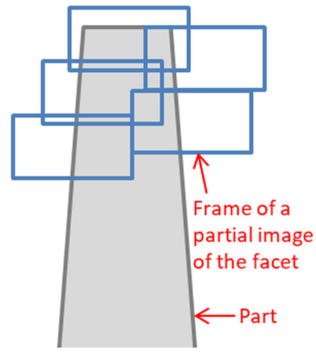


Fig. 2 Division of the surface of the facet into partal images. Here for example a surface of an object is divided into multiple images to cover the surface

tool for process optimization. As a corollary, the total time required can be decreased, improving production efficiency.

Therefore, an optimal division of the object into a minimal amount of images is advantageous. Because each object has a unique shape with different focus points, the minimum number of images required to inspect the part must be constantly calculated. However, the target geometry of the object is known and can be called up to plan the measurement.

An essential step in the manufacturing process that significantly affects the quality of industrial products is damage detection on their surface (Zhou et al., 2019). Defect detection is also a crucial part of the inspection process to accept or reject a part manufactured in a process or delivered by a supplier. In addition, it can also enable rework and repair of parts, thereby reducing material waste. In the past, error detection was performed by human experts who had experience with the process (Bhatt et al., 2021). The manual detection approach costs a lot of time and is easily influenced by the subjectivity, vigor, and experience of the inspector (Zhou et al., 2019). As a result, there is a rapidly growing market for automated inspection in numerous industries, including aviation. Automated inspection additionally quickens quality control. Intelligent visual inspection systems are increasingly in demand to guarantee excellent quality in industrial operations (Ben Abdallah et al., 2019). With this in mind, it is crucial to create several algorithms for automated quality control. The approach provided here can minimize the number of required photos by WLI, in the interests of lowering quality assurance expenses.

The Zero Defect Manufacturing (ZDM) describes a disruptive concept that contributes to the realization of the "First-Time-Right" quality strategy. Powell et al. (2022) Since quality control is carried out by a WLI after the production processes, it can be classified as 'physical detection' according to Psarommatis et al. (2019) and (2022). Detection of errors and possible repairs are not new strategies (Psarommatis et al., 2022) and are not exclusively reserved for the ZDM paradigm. According to Powell et al. (2022), strategies that focus on detecting and fixing errors should not be considered zero-defect strategies. Depending on which philosopher you follow, the algorithms can be considered part of ZDM due to the domain of this problem. In the authors' opinion, recognizing errors should be part of classic quality assurance. However, this does not make the algorithms presented in this work unimportant because they solve a real problem in practice. In addition, the solution presented here can also contribute to other sub-problems in which image positions are to be determined based on geometries.

This paper presents a new variant of the inspection problem in Chapter "Mathematical problem description", emphasizing the critical role of focus point positioning. To address this problem, the literature is searched for solutions. The results were presented in Chapter "Related work". In addition, the approaches were adapted to the inspection problem (Chapter "Selection and implementation of the approaches"), with a novel approximation approach also being introduced (Chapter "QuadPos-approximation"). In Chapter "Experiments", all approaches are compared and evaluated in different test scenarios. Chapter "Discussion" shows that the QuadPos approximation delivers significantly better results when covering free-form sheet metal parts than conventional approaches. In Chapter "Conclusion and outlook", the results are summarized again and an outlook for further research is given.

Related work

Literature review

Due to its enormous potential, quality assurance in manufacturing sectors has been the subject of numerous studies. Recent advancements in pattern recognition field have widened the scope of research being conducted on automated image recognition, as in Zhou et al. (2019), Ben Abdallah et al. (2019), Konrad et al. (2019) or Huang et al. (2020). The inspection problem for three-dimensional objects is formalized as a hitting set problem in Edelkamp et al. (2017) and solved with a Monte Carlo-based hitting set solver. For leather quality assurance, a process for detecting faults was enhanced in Bong et al. (2019). The camera's high resolution, moreover, enables a full capture of the entire object. It is comparable to the "watchman route" from Danner and Kavraki (2000). To gather more information on this issue, a literature review on surface inspection planning was conducted at the beginning of the work. We decided to use a search engine to classify the first 70 articles sorted as relevant in order to gain an overview of the literature. The search term used was 'surface inspection,' and only articles published after the year 2000 were included. The articles were then categorized into different groups based on their titles and abstracts.

In Fig. 3, the result of this literature classification is presented. It is quite noticeable that the 'Surface detection' group is very prominent. This group includes articles primarily focused on defect detection on a surface, with 39 out of 70 articles classified within it. The second strongest category is 'Inspection Planning,' which is also the focus of this paper. This category encompasses all articles related to the planning of quality assurance systems in some form. The 'Full inspection system' category includes papers that introduce comprehensive systems for addressing specific inspection problems. Furthermore, the 'Localization methods' category comprises articles discussing coordinate transformations or position determinations. Articles summarizing other works were labeled as 'Literature summary.' Detailed classifications and articles can be found in the Appendix.

The articles (Andreas Bircher et al., 2018; Phung et al., 2017) and Hoang et al. (2020) deal with the exploration of

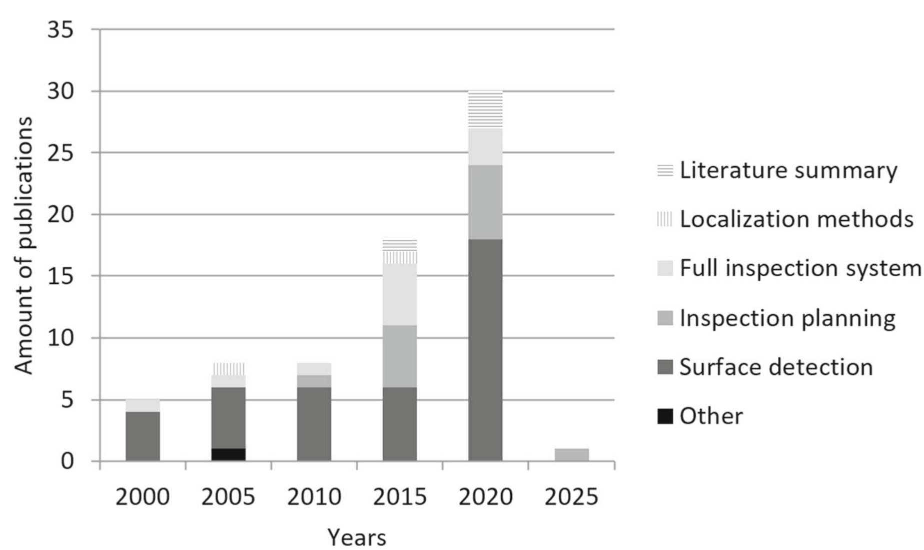


Fig. 3 Representation of a column chart of literature classes grouped by year

autonomous robots or unmanned aerial vehicles (UAV) in 3D environments. In Phung et al. (2017) a discrete particle swarm optimization (DPSO) is used to determine the path. Yi et al. (2021) and Zhou et al. (2016) deal with contact measuring devices and coordinate measuring machine (CMM) respectively. The article (Wu et al., 2015) describes the pathfinding problem with laser scanners. Even if the domain is a little different here and the algorithm was primarily defined for 3D environments, parts of the solution can be transferred to the problem of this paper. Here, path planning is made possible by determining the minimal enclosing rectangle (MER) of the freeform geometry. The area of the MER was then rasterized into segments with the size of the field of view (FOV). Since this is a 3D problem, the corresponding angles for the curvature of the 3D object were calculated here. However, the solution cannot be transferred to the current problem without adjustments because the alignment of the focus points was not taken into account here. The same applies to the reference articles (Zhou et al., 2011; Fernandez et al., 2008; Lee & Park, 2000; Son et al., 2003). In terms of domain, Pernkopf and O'Leary (2003) is similar to the domain of this paper. Three image capture techniques are presented in the paper. Unfortunately, this paper does not delve further into the pathfinding of freeform surfaces. In Gronle and Osten (2016), various algorithms are proposed for path planning of a microscope for quality assurance of a gear in three dimensions. A greedy algorithm is used to make a selection between different points. This requires a list of possible camera points. However, how the actual camera positions are calculated remains open; only a determination is made using sensor information and the model of the object.

In Glorieux et al. (2020) several methods for 3D path planning are summarized and compared. 3D models of sheet metal parts of a car door were used to compare them with each other. The methods cannot be completely transferred to the problem of this paper. The focus point is also not taken into account here, which is an important secondary service of this paper. However, these approaches offer initial clues. Many random-based strategies are listed; which, according to Glorieux et al. (2020), do not guarantee complete coverage. Selected methods of Glorieux et al. (2020) are presented in "Suggested methods for planning the coverage path" section. More approaches are to be found in Glorieux et al. (2020). However, since the domain differs so much, we do not elaborate on them. The literature search shows that there are initial approaches that address parts of the problem, but none of the approaches are currently able to solve the problem completely. There is, therefore, a need for further research.

Suggested methods for planning the coverage path

As shown in the previous section, there are several approaches to solving similar problems. In this section, the approaches are broken down and summarized again. This is intended to provide a structured overview of relevant approaches to solving the problem in order to be able to conduct a comprehensible discussion of the approaches afterwards ("Selection and implementation of the approaches" section).

Randomized algorithms

In González-Banos (2001) an algorithm based on a random sampling strategy transforms the art gallery problem into an instance of the set cover problem. The Greedy algorithm will then be used to determine a route for driving through the points (González-Banos, 2001). A similar sampling strategy from the perspective is adopted in the methodology of Bircher et al. (2018) and is intended to be integrated into the proposed rapid exploration path planning algorithm "Random Tree of Trees". The probability of achieving complete coverage increases with the number of randomly selected permitted viewpoints. Unfortunately, these algorithms do not guarantee complete coverage (Glorieux et al., 2020).

Grid viewpoints

In Raffaeli et al. (2013) is a strategy proposed that first clusters the primitives based on distance and surface normal direction in order to group primitives that can be covered from the same viewpoint (Glorieux et al., 2020). This is called "surface sampling" (Raffaeli et al., 2013). Analogous to this approach, the MER of the free form is determined in Wu et al. (2015). The MER is then divided into segments that are as large as the FOV. This principle is illustrated in Fig. 4.

In Raffaeli et al. (2013), for each segment, a viewpoint is randomly selected that covers all primitives in the group and is included in the coverage path. This significantly reduces the number of viewpoints, but makes it difficult to ensure complete coverage. Glorieux et al. (2020) If the approach of

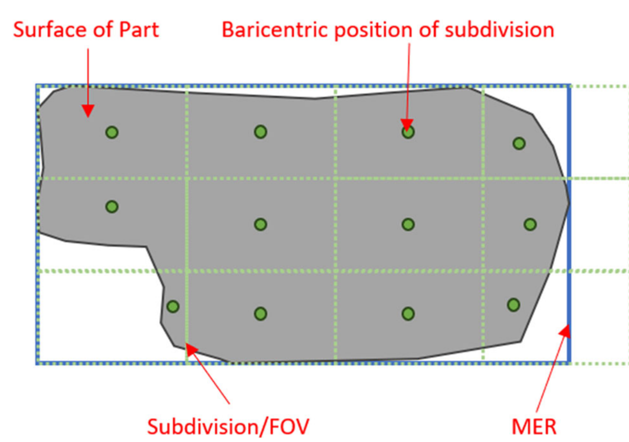


Fig. 4 Surface sampling described by Raffaeli et al. (2013), Wu et al. (2015) and Glorieux et al. (2020)

Wu et al. (2015) is transferred to this 2D scenario, then the center of the section is simply used as the viewpoint.

Coverage of basic geometric elements

In Englot and Hover (2017) an algorithm for path planning on damaged ships is presented. The algorithm uses sphere tessellation and cube grids to create view positions and evaluate their visibility (Glorieux et al., 2020). New viewpoints are added until each geometric primitive has been observed the required number of times. Start by selecting a geometric primitive that has not been observed in the required number. Different viewpoints are generated uniformly and randomly in the local neighborhood of this primitive. After a viewpoint is added to the roadmap, another primitive is selected and the process repeats until the redundancy requirement is met (Englot & Hover, 2017).

Methods to consider model of a hitting set problem

As already mentioned, there are similar approaches to solve the problem under the consideration as a hitting set. According to Karp (1972), hitting set is an NP-hard optimization problem and is defined as follows: Given is a bipartite graph $G = (V, E)$ with $V = V_1 \cup V_2, V_1 \cap V_2 = \emptyset$ and $E \subseteq (V_1 \times V_2)$, find a set V' of V_1 of minimal cardinality, so that all nodes in V_2 are convered, implying that there is a $v_1 \in V_1$ for every $v_2 \in V_2$, so that $(v_1, v_2) \in E$ (Edelkamp et al., 2017). Since hitting set is NP-hard, brute force does not appear to be a viable alternative. Hitting set is equivalent to set cover and algorithm Greedy is one of the most natural and effective heuristic for set cover (Skiena, 2008). Thus, the Monte-Carlo approach of Edelkamp et al. (2017) as well as a classical Greedy approach to the solution were considered in more detail.

In single-agent games, nested Monte Carlo search has consistently delivered impressive performance. The efficient condensation of information occurs through a recursive process, wherein the algorithm adeptly orchestrates each step. This methodology is underpinned by a rollout concept, with successive actions being generated as long as the game remains ongoing. The generation of these actions is governed by a reinforcement strategy that leverages insights from prior outcomes (Edelkamp et al., 2017). Greedy algorithms are often used to solve optimization problems by maximizing or minimizing a set. A greedy algorithm typically seeks a local optimum, studying only a small portion of the problem, thereby making a more efficient decision (Alsuwaiyel, 2016). Hence, greedy algorithms always prefer the option that seems most advantageous at the time. The work of Cormen et al. (2013) demonstrates various greedy approaches. In Algorithm 1, the approach of Alsuwaiyel (2016) will be explained and implemented as a classical greedy algorithm.

Algorithm 1 Example of a Greedy-Algorithm following Alsuwaiyel (2016)

```
1: Add 1 to List X
2: Add List V without entry 1 to List Y
3: For each vertex v ∈ Y if there is an edge from 1 to v then let δ[v]
   (the label of v) be the length of that edge; otherwise let δ[v] = ∞
4: Let δ[v] = 0
5: while Y ≠ do
6:     Let y ∈ Y be such that δ[v] is minimum
7:     move y from Y to X
8:     update the labels of those vertices in Y that are adjacent to y
9: end while
```

Mathematical problem description

Object and grid model

An object o is defined by a set of points p. Since the problem is two-dimensional, each point p has an x- and a y-value.

$$o = \{p_1, p_2, p_3, \ldots, p_n\} \tag{1}$$

$$p = (x, y) \tag{2}$$

The set $\{p_1, p_2, p_3, \ldots, p_n\}$ symbolizes the shape of the object o. The surface of the object is described by a point raster. The columns in a row are shifted by half of the distance employing a three-cornered rasterization. This will enable better coverage of the object's intermediate spaces. The calculation of the triangular rasterization is outlined by the following equations (3), (4) and (5). The row function, which creates a series of points, is demonstrated by the Eq. (5). For the x sequence, the points set is generated in a region between y_{start} and y_{end}. After that, the function $tritacticRow$ in Eq. (4) moves the odd rows by 0.5 in a range of 0 to r_{end}. Equation (3) defines the merging of the different rows of the function $tritacticRow$ from rows 0 to c_{end}.

$$raster(r_{end}, c_{end}) = \bigcup_{x=0}^{c_{end}} tritacticRow(x, r_{end}) \tag{3}$$

$$tritacticRow(x, r_{end}) = \begin{cases} row(x, 0, r_{end}), & \text{if } x \bmod 2 = 0 \\ row(x, 0.5, r_{end}), & \text{if } x \bmod 2 = 1 \end{cases} \tag{4}$$

$$row(x, y_{start}, y_{end}) = \bigcup_{y=y_{start}}^{y_{end}} (x, y) \tag{5}$$

The selection of the resolution size in relation to the object's edge and the raster image size is highly essential. If the resolution is too high, uncovered areas that are not detectable may arise automatically. If the resolution is too low, the calculation's performance will inevitably degrade.

However, this is an empirical value that can vary depending on the resolution of the camera and the size of the part to be examined. In the scenarios of this paper, the camera's field of view has a width of 4.2 and a height of 2.8 (of any dimensional unit). After a bit of experimentation, good results were achieved with a grid resolution of 0.5 and a border resolution of 0.25 (of any dimensional unit). These values may have be adjusted for other use cases.

The surface shape is defined by Eq. (6). The variables r_F and w_F, therefore, describe the object's radius and width, while the variables h_{minF} and h_{maxF} characterize the object's beginning and ending points.

$$contur(x, r_F, w_F, h_{maxF}, h_{minF}) = \begin{cases} \sqrt{r_F^2 - x^2} - r_F \pm \frac{w_F}{2}, & \text{if } h_{minF} \leq x \leq h_{maxF} \\ \pm\frac{w_F}{2}, & \text{if } h_{maxF} \leq x \text{ or } x \geq h_{minF} \end{cases} \tag{6}$$

The circle function serves as the foundation for the equation $\sqrt{r_F^2 - x^2} - r_F$, while the part $\pm\frac{w_F}{2}$ describes the division and translation of the function by $-\frac{w_F}{2}$ and $+\frac{w_F}{2}$. The facet length between h_{minF} and h_{maxF} is constrained by the equation $h_{minF} \leq x \leq h_{maxF}$.

Microscop model

A microscope camera C is also available, and it has advanced features including adjustable field of view width and height (w_c, h_c). A point drawn from the values x_c and y_c signifies the current position of the camera center.

Two focus points are shown by positive and negative distances on the y-axis f_c from the image's center in Fig. 5. The variables x_c and y_c represent the current position of the image center. The function $isOnSurface$ takes the object o and a position defined by x and y as input parameters. The function indicates that the position lies within the object's contour o by returning the value $true$. If the position is outside the contour, a result $false$ is obtained. Applying to the focus points we obtain

$$isOnSurface(o, x_c, y_c \pm f_c) = true \tag{7}$$

so that the focus points must lie always be within the contour of the object o. Unless this is done, no image can be captured. Based on two corner points of a rectangle, Eq. (8) defines the field of view.

$$fov(x, y) = \left\{\left(x - \frac{w_c}{2}, y - \frac{h_c}{2}\right), \left(x + \frac{w_c}{2}, y + \frac{h_c}{2}\right)\right\} \tag{8}$$

The details regarding what a partial image can capture are represented in the Eq. (9). Here the function $vis(o, p)$ is defined inside a square area and the function $areaMatchesObject$ returns the contour points. This is identified by the function $fov(x, y)$.

$$vis(o, p) = \Big\{areaMatchesObject(fov(x, y), o), \quad x \in p \text{ and } y \in p \tag{9}$$

$$vis(o, p_1, p_2, \ldots, p_n) = \bigcup_{i=0}^{n} vis(o, p_i) \tag{10}$$

The function vis of Eq. (10) is overloaded with images. This function takes as parameters the object and the positions of the partial images $p_1, p_2, \ldots, p_n$, whereas n describes the positions number to return. A quantity is returned for all points when the union is applied.

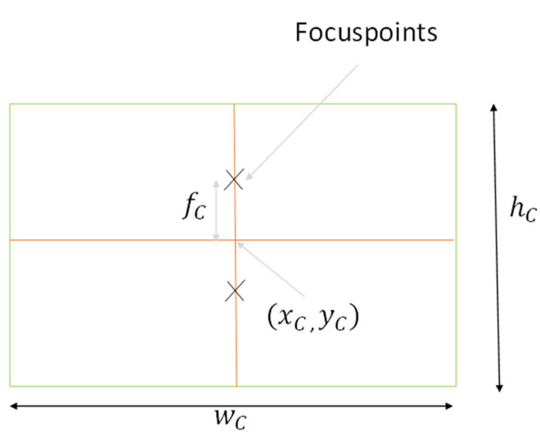


Fig. 5 Mathematical description of the microscope camera. It describes the width and height of the image. The focus points are also displayed at a distance of f_c

Method

For the use case presented in detail in Chapter 3, the best possible algorithm for the solution should be found. To do this, different algorithms are compared against each other and solve the problem for different parts. This solution is then evaluated using the model for evaluating the algorithms presented in the first part of this chapter. Next, this chapter tests the various algorithms from the "Suggested methods for planning the coverage path" section for their applicability to this scenario and presents the implementation details. Then, the QuadPos approximation is presented. The aim of this chapter is to prepare for experimental testing of the algorithms.

Scoring and implementation of the approach interface

This section presents the calculation of the solution quality of an algorithm and the implementation of the approach interface.

Scoring

Two factors must be considered when assessing the validity of a solution. On the one hand, the entire surface must be covered. On the other hand, the number of image positions generated by an effecient solution should be kept small, so that the surface measurement time is as short as possible. In general, the number of covered grid points may be applied to determine the extent of surface coverage. If all points are covered by at least one image, it is assumed that the surface was covered using the partial images. Such a procedure necessitates an adequate rasterization resolution. If the resolution is too big, the part may not be completely covered. On the contrary, if the resolution is too low, it'll result in major performance decline. The author created the raster using a factor of 0.5, as described in "Other conditions" section. The ratio of the object's surface to the surface consumed by the part-images is chosen for quality assessment.

$$cut(x) = \begin{cases} x, & \text{if } x \leq 1 \\ 1, & \text{if } x > 1 \end{cases} \tag{11}$$

$$quality(o, p_1, \ldots, p_n) = \frac{count(vis(p_1, \ldots, p_n, o))}{count(o)} \cdot cut\left(\frac{area(o)}{n \cdot w_c \cdot h_c}\right) \tag{12}$$

Equation (12) together with Eq. (11) represent an optimization method based on complete coverage and image number minimization. The maximum number of images is denoted by n. The function $count(...)$ simply returns the number of elements. Term $\frac{count(vis(p_1, p_n, o))}{count(o)}$ indicates the number of grid points covered by images and located on the facet divided by the total number of grid points on the facet by the $count(o)$. When the entire facet is covered, an optimal result is 1. As a ratio of the object's surface to the cumulative surface of the partial images, the term $\frac{area(o)}{n \cdot w_c \cdot h_c}$ is used. Hence, a result above 1 indicates that there are few images below the theoretical minimum. By default, function $cut(x)$ treats any result over 1 as 1 since it does not match the optimized solution.

In the scoring algorithm new solutions are iteratively generated. These solutions should be assessed, and the best one should be chosen and developed further. To fullfill this, a higher-level evaluation algorithm that follows a simple maximization logic is used. This is represented by Algorithm 2. A new solution is only deemed effective if it covers more than the previous solution or covers the same portion with fewer images. The $CalculateCoverRate$ method describes the determination of the coverage percentage of the current solution. A new solution with always a new image is generated until the calculated coverage corresponds to one. A new solution is generated using the $DoCalculationStep$ function. This function has to be implemented by each approach. An implementation of QuadPos is presented in "Initiation" section.

Algorithm 2 Algorithm to generate the positions of the partal images

procedure IMAGERASTER($object, image$)
    $currentIndex = 0$
    $coverRate = 0$
    $bestScore = 0$
    $bestImageSize = 0$
    $bestSolution = null$
    while $coverRate \neq 1$ **do**
        $solution = DoCalculationStep(object, image, currentIndex)$
        $coverRate = CalculateCoverRate(solution)$
        if $isBetter(coverRate, bestScore, bestImageSize, solution)$ **then**
            $coverRate = bestScore$
            $bestSolution = solution$
            $bestImageSize = solution.imageSize$
        end if
        $currentIndex = currentIndex + 1$
    end while
    return $bestSolution$
end procedure

procedure ISBETTER($coverRate, bestScore, bestImageSize, solution$)
    return $coverRate > bestScore$ or ($coverRate \geq bestScore$ and $bestImageSize > solution.imageSize$)
end procedure

Selection and implementation of the approaches

"Suggested methods for planning the coverage path" section presented various approaches to generate different viewpoints to analyse a sheet metal part. In the first part of the "Randomized algorithms" section, random-based algorithms were introduced. However, according to Glorieux et al. (2020), this does not guarantee complete coverage of the plate part. Since complete coverage is always required, these approaches are no longer considered. The approaches of Wu et al. (2015) and Raffaeli et al. (2013) define a grid in sections scaled like the FOV. They then attempt to set a view position in that local section if the center point in the section does not meet the two conditions. Otherwise, the center of the section is simply chosen, as it most likely covers all points. On first glance, this procedure described in "Grid viewpoints" section seems efficient. Unfortunately, the focus points that are absolutely necessary for the solution are not taken into account. However, the approaches of Raffaeli et al. (2013) could be extended to include this function by a condition that checks the position of the focus points. The method has also been adjusted so that only 90% of the width and height of the image is used to generate the grid. This procedure means that there is more luck when generating the image points, since only 90% of the geometric points have to be covered. This means that the specifications with the focus points and the coverage of the image points can be met. Initial tests with this approach show valid results. In the following, this approach is called "Gridded Randomization". The approach of Englot and Hover (2017) is based on trying to cover basic geometric elements. These do not exist in our problem. The approach, therefore, is no longer be pursued.

In single-agent games, nested Monte Carlo search has produced well-performing results. Information should be condensed exponentially by nesting the search. The algorithm controls this through recursive method calls. A rollout concept underpins the procedure. Until the game is over, successors are generated based on the current state. Through a reinforcement strategy based on previous results, a successor is randomly generated (Edelkamp et al., 2017). In this way, several different solutions can be generated. When implementing the algorithm, we decided to generate 50 different solutions. An updated current solution is generated by selecting the solution with the largest coverage or the same coverage with fewer images. The process is repeated until the entire surface is covered. This algorithm is known as "Rollout Monte Carlo". Another idea is to create a Monte Carlo algorithm that changes the hole sequence of the current solution to produce faster solutions. New solutions replace old ones if they cover a larger or equal area but contain fewer images. This is repeated again until the surface is completely covered. "Sequence Monte Carlo" is the name for this algorithm. A greedy algorithm is also implemented that solves the problem as a hitting set.

Another approach is to view the issue from a geometric standpoint. Since representing it as a Hitting Set problem raises it to an additional level and vastly abstracts the problem, some facts might not be prominently featured. Therefore, a more direct view of the problem may be more convenient. The implementation of the QuadPos approximation is presented in detail in "QuadPos-approximation" section. All algorithms discussed in this chapter are explained in text or pseudo code. The complete implementation of all algorithms can be found in the repository https://github.com/bschw4rz3/OptimizationOfImageAcquisition.

QuadPos-approximation

In this section, the QuadPos approximation method is introduced. The fundamental idea here is that a quadratic surface can be efficiently divided using uniformly sliced rectangles. This is referred to as an image grid in the following.

As shown in Fig. 6, this uniform grid division is not suitable for shapes that are not squares. In the left example, using the uniform image distribution requires more images to cover the polygon. In the right example, adjusting the grid results in the need for fewer image positions. The focus points must be located on the surface of the object, as described in Eq. (7). In summary, the following aspects have to be considered:

- Find start of the image grid (the first image determines how the grid continues)
- Adjustment of the image grid to the contour to reduce the number of images required
- The focus points must be located on the surface of the object

This type of image decomposition is accomplished iteratively. An image must always be placed on the surface of the

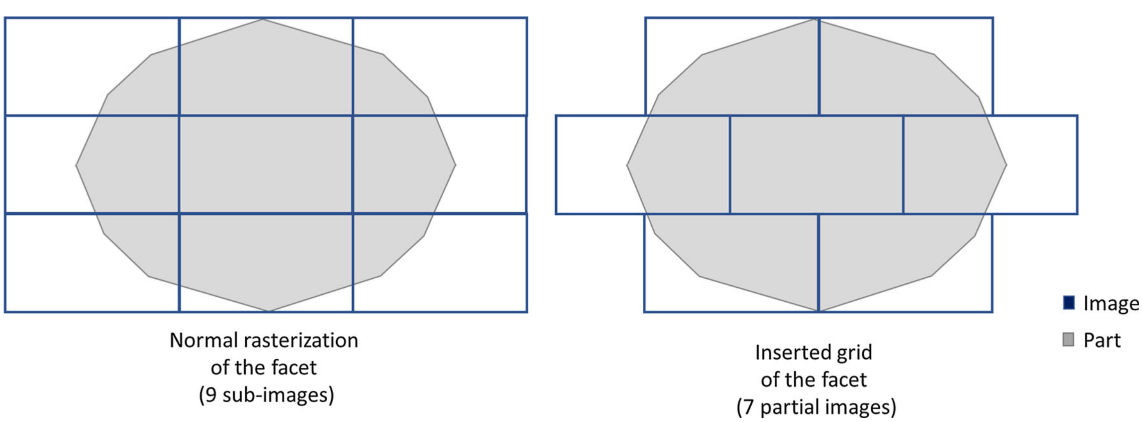


Fig. 6 Various grids of a geometry. This shows how ineffective normal rasterization is compared to a indented grid

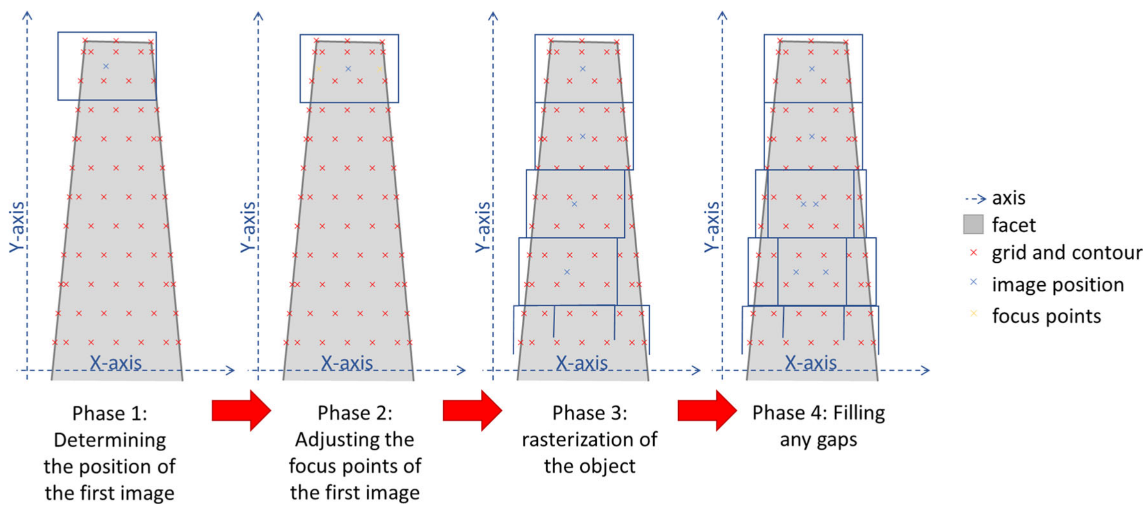


Fig. 7 Different phases of the algorithm. Phase 1 searches for the position of the first image. Phase 2 Adjustment of this position by focus points. Phase 3 and 4 covers the surface with images by rasterization

object in this case. Following that, it is determined whether the grid points are adequately covered. This logic is discussed in "Scoring" section. The algorithmic process is divided into four phases, as shown in Fig. 7. In the first phase, iteratively finding an optimal position for the first image is carried out. The first image must fit well to the contour of the surface, since all other images are based on it. Keeping the total number of pictures low requires a good position for the first image. As starting points, the grid and contour positions are checked.

The second phase evaluates whether the found position takes into account that the focus points are on the object's surface. If this is not the case, an attempt is made to detect a starting point based on the knowledge gained during Phase 1. It may happen that no position with all focus points on the surface is found. In such a situation, the algorithm is terminates. The screening is performed in the third phase, based on the first image. This always occurs with the focus points on the surface. As a result, images can be shifted back and forth or not set at all. Therefore, uncovered areas may also form on the surface. When this occurs, Phase 4 is launched. This phase's primary function is to fill in any gaps that may have developed so that the focus points are visible on the surface. It is also possible to deviate from the remaining grid of image positions.

The conditions for executing the phases are also depicted in Fig. 8. For completeness, Algorithm 2 from "Scoring" section illustrates the overhead. The QuadPos implementation initiates with the execution of the $DoCalculationStep$ function. Initially, it checks for potential starting positions, generating a list of possible starting points, which are then validated for being suitable as focus points. Each validation and score calculation constitutes an iteration and contributes to the solution.If the index exceeds the number of possible starting points, it verifies the existence of a valid starting point. If none is found, the algorithm attempts to approximate a starting point. When a starting point is successfully identified, the algorithm endeavors to complete the adjusted row or add a new row near the left contour of the geometry. This process results in the creation of an adapted grid of image positions. Once the grid generation is complete, if the coverage is not equal to 1, the algorithm searches for uncovered positions on the surface and approximates image positions to fill the gaps.

Initiation

The variables $contourMarker$, $topLeftCorner$, and $possibleStartPoints$ are initialized prior to the initial execution of $DoCalculationStep$. As a result, the surface's contour is pre-analyzed. The variable $contourMarker$ first marks a maximum on the Y axis with the smallest X value possible. If multiple points on the Y axis have the same value, the point with the smallest X value is chosen. This value is constantly adjusted.

Algorithm 3 Algorithm for initiation

```
procedure PREANALYSIS(possibleStartPoints, object)
    Analyse the conture of the object for corners
    Filter for corner with maximal Y-value and X-value as small
    Set topLeftCorner to the result of the search
    Filter for position with maximal Y-value and X-value as small
    Set it to contourMarker to the result of the search
    append contourMarker to possibleStartPoints
end procedure
```

Just iteratively improving the $contourMarker$ does not suffice to find the corner of the surface's contour. This necessitates additional corner detection. The work of Karim and

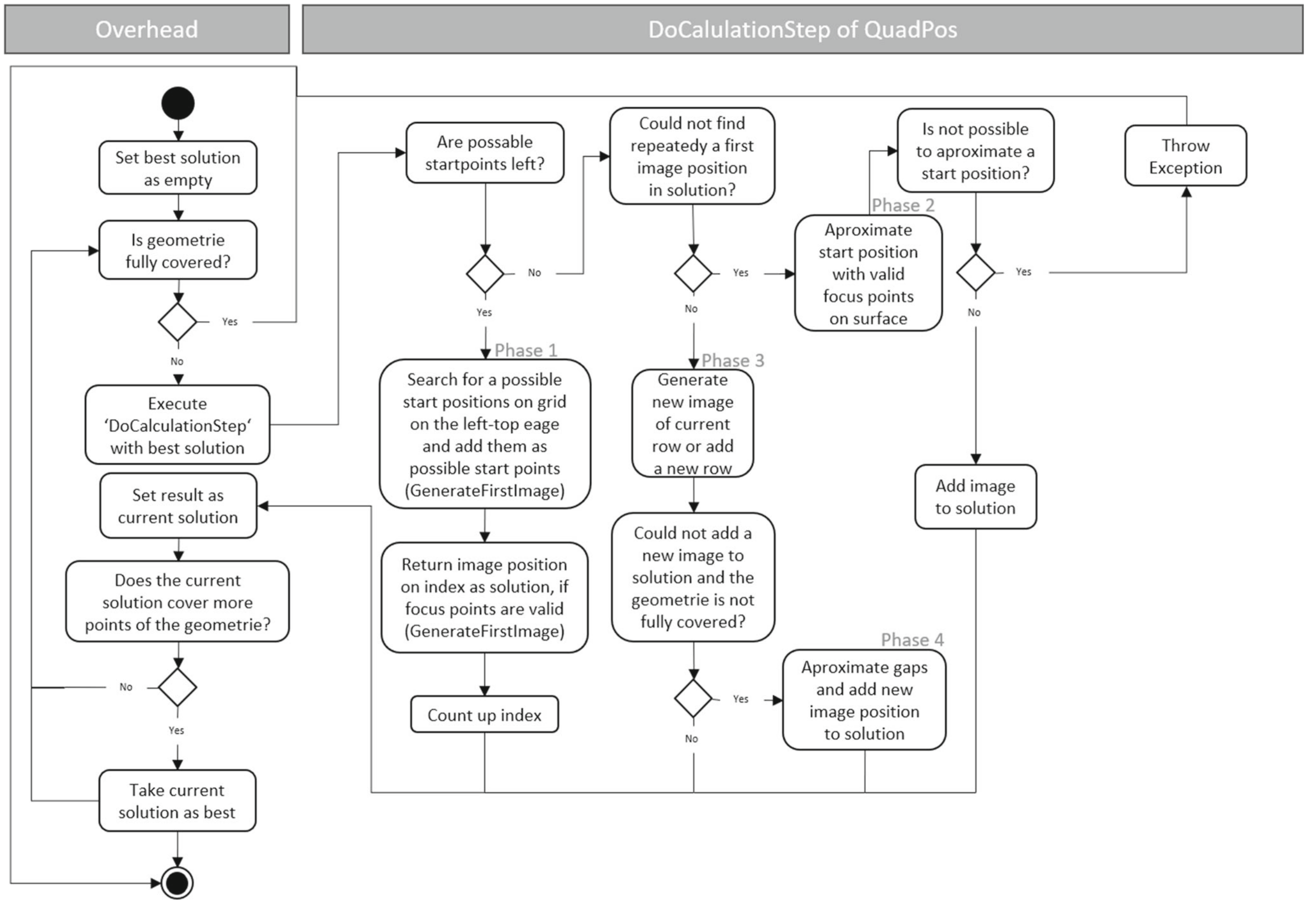


Fig. 8 The flowchart illustrates all the phases of QuadPos and the conditions under which they are executed

Nasser (2017) asserts that the algorithms SUSAN, Harris, and FAST recognize pixel matrices using gradients and neighboring points. These algorithms cannot be directly applied because they only know individual points, not pixels. However, corner detection can be achieved by using the concepts of these attachments. Corner detection was carried out similarly to the SUSAN method by calculating an angle between the first and last point within a radius. An angle that is outside the tolerance is considered a corner. In this application, 60° and 300° tolerance values were effective for detecting corners. The $topLeftCorner$ variable defines then the corner with the highest Y value and the lowest X value.

For generating the first image, $contourMarker$ and $topLeftCorner$ are considered important reference points (see "Phase 1: Generation of the first image position" section). This $contourMarker$ serves as a starting point on the $possibleStartPoints$ list. Also, the initialization is shown in Algorithm 3. Following that, the routine $DoCalculationStep$ is executed. The pseudocode of Algorithm 4 describes which one of the four phases is selected, based on the logic depicted in Fig. 7. Initially, the algorithm would seek an image in the upper left corner of the object. This is accomplished through the $generateFirstImage()$ call, explained in "Phase 1: Generation of the first image position" section. As Phase 1 fails to find a position with focus points on the surface after several iterations, Phase 2 begins to approximate that position. $contourMarker$ is used here, which has been repeatedly adjusted in Phase 1. When the first image is located, Phase 3, which adds new columns or images to an existing row at each iteration, can initiat. Images are added until coverage reaches 100% or adding new images no longer affects coverage positively. This is followed by Phase 4, which closes the gaps outside the image grid.

Phase 1: Generation of the first image position

Basically, to determine the position of the first image, different positions are selected. The starting point here is the position of the $contourMarker$, from which the neighboring points are checked as starting points. The determination of the $contourMarker$ for the first iteration takes place in the initiation of the algorithm. This point is read from the $possibleStartPoints$ list at index 0 and is denoted as $current$ in Algorithm 5. Within the $current$ posi-

Algorithm 4 Algorithm to generate the positions of the partal images

```
procedure DoCalculationStep(object, image, currentIndex)
    try
        if there is no starting image in solution then
            image = GenerateFirstImage(...)                ▷ Phase 1
            if focus points of new image are on the surface then
                add image to solution
            end if
        else if can't find a first image in solution then
            aproximate focus points on surface             ▷ Phase 2
            if can aproximate a focus point on surface then
                throw exception
            end if
            add image to solution
        else if can detect row in solution then
            generate new images in the grid                ▷ Phase 3
            if new image raise score then
                add image to solution
            else
                add a new row to solution
            end if
        end if
    catch unable to add images and coverage rate is not equal to 1
        aproximate Holes                                   ▷ Phase 4
        add a new row to solution
        if cover rate of solution equals 1.0 then
            Search for unnessary images
            Remove unnessary images from solution
        end if
    end try
    return solution
end procedure
```

Algorithm 5 Determines the first starting point

```
procedure GenerateFirstImage(object, picture, currentIndex, possibleStartPoints)
    Set current to possibleStartPoints at currentIndex
    Set coveredPoints to the covered points of the solution including current
    Get point current from possibleStartPoints at currentIndex
    Search for point NLBP on surface between current.Y − picture.Height·0.5 and current.Y + picture.Height·0.5 with lowest X-value
    contourIsPicture = pointIsInPicture(current, picture, contourMarker)
    cornerInPicture = topLeftCorner is invalid or pointIsInPicture(current, picture, topLeftCorner)
    if contourIsPicture and cornerInPicture then
        Add coveredPoints to possibleStartPoints that are not included
        if contourMarker.x > NLBP.x and isOnSurface(NLBP, object) then
            contourMarker = NLBP
        end if
        if isFocusValid(current, picture, object) then
            Add current to solution
            return solution
        end if
        return fail
    end if
end procedure
```

tion, the point with the smallest x-value is searched for within an image size. This point is denoted as nearest left boarder point (NLBP) in Algorithm 5. With this search the $contourMarker$ should be improved iteratively. Using the maximization function from the Algorithm 2, the starting position is chosen that covers the most points on the contour.

The $possibleStartPoints$ list is updated to include all raster and contour points covered by the image, if the $contourMarker$ is inside the image drawn around the $current$ and the determined NLBP is on the object's surface. The question of whether a better beginning point can be generated with these points is then explored in the next cycles. The $contourMarker$ will be replaced with the calculated NLBP if its x-value is less than the existing $contourMarker$ value. The object's contour is meticulously tracked using the $contourMarker$ as it moves through the many iterations. This guarantees that the initial picture is always placed at the contour's edge, as the $contourMarker$ must always be within the image of a start position. In addition, the corner point with the smallest x and highest y value must always be within the image if this could be determined. Through the different iterations of the $possibleStartPoints$ list, it is possible to maximize the number of covered points of the surface. New points are added to the $possibleStartPoints$ list if they have been covered within the image and are not already in the list.

Phase 2: Approximate the first position

If no start point could be established, Phase 2 is initiated. As the focus points of the image cannot also be on the surface, it is presumed there isn't any spot on the grid where such a scenario may occur. Hence, a position is roughly determined using the $contourMarker$'s previously iterated position. The search is conducted within a perimeter of $contourMarker$ plus half the width and height of the image, at intervals of 0.05.

This assumes that the $contourMarker$ is positioned in the top left corner. The Algorithm 6 calls the $getRelativeFocusPoints()$ function, which provides the relative position of the focus points. They are then added to the estimated positions in order to establish the absolute ones. Thus, it is possible to check each focus point for the relevant area. If a location is discovered where all focal points are on the surface, it is added to the $possibleStartPoints$ list. The best starting position can then be revealed by repeating Phase 1.

Algorithm 6 Approximation of the first starting point

```
procedure GenerateFirstImage(picture, possibleStartPoints, contourMarker)
  relativeFocusPoints = picture.getRelativeFocusPoints();
  x = contourMarker.x
  while x < contourMarker.x + picture.width · 0, 5 do
    y = contourMarker.y
    while y < contourMarker.y + picture.height · 0, 5 do
      isValidPosition = true
      i = 0
      while i < relativeFocusPoints.size() do
        Set relative to relativeFocusPoints on i
        Set absoluteFocusPoint to (x + relative.x, y + relative.y)
        if isOnSurface(absoluteFocusPoint) then
          isValidPosition = false
        end if
      end while
      if isValidPosition then
        Append position (x,y) to possibleStartPoints
      end if
      y = y + 0.05
    end while
    x = x + 0.05
  end while
end procedure
```

Algorithm 7 Algorithm for generating the image grid

```
1:  procedure CompleteGrid(solution, object, picture, contourMarker)
2:    processRows = true
3:    if last image of solution is first image in row then    ▷ Try to move the image to get a better first position in row
4:      set currentPoint to last image of solution
5:      newImagePoint = getNextLeftXValueInRow(solution, currentPoint, picture)
6:      if isFocusValid(newImagePoint, picture, object) then
7:        Caluclate covered surceface points with newImagePoint
8:        if contourMarker.x + (picture.width/2) >= newImagePoint.x then
9:          if New cover rate is bigger than old then
10:           Overwrite last item from solution with newImagePoint
11:           processRows = false
12:         end if
13:       end if
14:     end if
15:   end if
16:   if processRows equals true then    ▷ Add a new image to row
17:     Set lastImagePoint to the last position of solution
18:     Set newCalculatedImage to lastImagePoint
19:     newCalculatedImage = moveOnXAxeForValidFocus(newCalculatedImage, lastImagePoint, picture, object)
20:     Add newCalculatedImage to solution
21:     Calculate cover rate of solution
22:     if new cover rate is lower or equal the old rate then    ▷ Add new row
23:       Remove newCalculatedImage from solution
24:       Set nearestSurfacePoint to detected begin of current row
25:       Set halfHeight to half of height of picture
26:       Set rowStart to the result of nearestSurfacePoint + halfHeight
27:       Approximate rowStart to valid focus
28:       contourMarker = nearestSurfacePoint
29:       Append rowStart to solution
30:       Calculate cover rate of solution
31:       if new cover rate is lower or equal the old rate then
32:         throw exception
33:       end if
34:     end if
35:   end if
36:   return solution
37: end procedure
```

Phase 3: Continuation of the grid

Most of the image positions are generated in Phase 3. To accomplish this, a distance equal to the width and height of an image is added to the grid starting with the first image. Three steps are required to complete this process.

A suitable starting point is to find a good position. This is achieved by placing the left-hand contour of the object closest to the left edge of the image, without intersecting it. According to the Algorithm 7, this corresponds to rows three to ten. It creates a point that is to the left of the first row, yet still on the surface, by using the $getNextLeftXValueInRow$ function. In Algorithm 8, the logic used to find the minimum distance from the x-value of other points on the contour and grid is explained. In the case of a point with a minimum distance, it is checked whether the position has better coverage of the grid and contour points and whether the $contourMarker$ is present at this location. If so, the algorithm will repeat itself with the better solution.

We assume that no better position exists if the image could not be relocated and no better alternative could be found. In this instance, a new image is added to the column. Following that, the image will be continuously moved to the left until the focus points are on the object. This approximation is explained in the Algorithm 9. To determine whether the focus points are pointing at the object's surface in this case, the $isFocusValid$ function is utilized. Then, the position is inserted in the solution, and it is examined to see if it has resulted in a higher coverage. If a better solution is found, the solution is returned and the algorithm is repeated for a new image.

If the previous steps have not resulted in a better solution, a new row is added to the image grid. To do this, the image from the previous cut is deleted from the solution and the beginning of the current column is searched for. If found, this position is shifted by one image height on the y-axis. However, since it cannot be guaranteed that the contour of the object will also be in this position, the x-axis of the point is shifted to the smallest x-value of the contour in a range of $\pm(picture.height/2)$. This point is added to the solution and it is checked whether a higher coverage could be achieved.

Algorithm 8 Generated the next X position

```
procedure getNextLeftXValueInRow(surface, point, picture)
    minDistanceX = -99999
    Set halfHeight to the half height of picture
    i = 0
    while i < surfacePoints.Length do
        Set surfacePoint to value from surface on index i
        Set distanceY to absolute Y-distance of surfacePoint and point
        if distanceY is lower or equal than halfHeight then
            Set distanceX to relative X-distance of surfacePoint and point
            if distanceX > 0 and distanceX > minDistanceX then
                Set current to new point with X of surfacePoint and Y of point
                minDistanceX = distanceX
            end if
        end if
        i = i + 1
    end while
    return current
end procedure
```

Algorithm 9 Search for the next position in the image plate

```
procedure moveOnXAxeForValidFocus(newCalculatedImage, lastImagePoint, picture, object)
    rasterStep = picture.width/100
    while isFocusValid(newCalculatedImage, picture, object) do
        if newCalculatedImage.x < lastImagePoint.x + (picture.width/2) then
            newCalculatedImage = lastImagePoint
        end if
        newCalculatedImage.x = newCalculatedImage.x - rasterStep
    end while
    return newCalculatedImage
end procedure
```

If this is the case, the current solution is returned. If a better coverage cannot be obtained, the end of the object is reached and an exception is thrown. Throwing the exception starts Phase 4.

Phase 4: Fill in any uncovered areas

This step is only carried out if the rasterization of the image locations has been finished, but total coverage was not possible. As no spot could be located where the focus points were covered, it is presumed that there are still gaps in the covering of the object's surface. Different places are estimated and chosen in accordance with the maximum coverage rate in order to fill these gaps. Here, the variables *approximationFactor* and *maxIterations* must be utilized. The value *approximationFactor* indicates the delta's percentage step size. While the maximum number of iterations allowed by the search is specified by the *maxIterations* argument. These settings should be modified if required. The uncovered points are first identified via the Algorithm 10 method, and the findings are saved in the *restPoints* variable. The focus points on the list are then checked to verify that they are placed on the object's surface. Since this won't initially be the case, the focus points outside the surface are chosen. From this focus point and the present point, a delta is calculated. By multiplying *approximationFactor* by *delta*, the shift of *currentPoint* is calculated by subtraction.To the *restPoints* list, the result is encoded as a new point.This would be repeated until either a place is found where all focus points are on the surface or the number of iterations exceeds *maxIterations*. In the last scenario, no more approximations are made and no viable location could be identified.

Algorithm 10 Approximate the gaps in the image rasterization

```
procedure fillHoles(picture, object)
    approximationFactor = 0.1
    maxIterations = 1500
    Set restPoints to the list of uncovered positions
    maxScore = 0
    Define maxPoint as undefined
    i = 0
    while i < restPoints.size() do
        Set currentPoint to restPoints on index i
        if isFocusValid(currentPoint, picture, object) then
            Add currentPoint to solution
            set covered to cover rate of solution
            if maxScore < covered then
                maxScore = covered
                maxPoint = currentPoint
            end if
            Remove currentPoint from solution
        else if i < maxIterations then
            Set focusPointList to invalid absolut positions of focuspoint
            h = 0
            while h < focusPointList.size() do
                Set focusPoint to focusPointList on index h
                Set delta to the diference between focusPoint and currentPoint
                Set delta to the product of approximationFactor times delta
                Set newPoint to the Subtraction of currentPoint and delta
                Add newPoint to restPoints
                h = h + 1
            end while
        end if
        i = i + 1
    end while
    Add maxPoint to solution
    return solution
end procedure
```

In the case that a valid position is detected, the coverage percentage is examined by including the position in the solution. This position is momentarily saved as a maximum if a more accurate calculation can be made with it than with the previous one. After that, the item is eliminated from the solution and other items are examined. The greatest approximation is provided once this procedure has been run $maxIterations$ times. In order to attain a coverage of 100%, the Algorithm 10 is repeated until there are no more holes on the object's surface. When the surface has been completely covered, it is crucial to inspect each image of the solution once to determine if it enhances the coverage. Algorithm 10 is repeated until there are no more gaps on the surface of the object and thus a coverage of 100% could be achieved. It's conceivable that the Phase 3 image positions overlap with those in Phase 4 in such a way that they no longer serve to increase coverage. In order to maintain a minimal number of image positions, these images should then be extracted from the solution.

Experiments

Analysis

The approaches outlined in "Method" section are cross-compared in this section, and predictions are made regarding how the problem will behave based on "Mathematical problem description" section. No research could be found that fully addresses the issue. From the literature search, the algorithm Gridded Randomize was discovered after modification. However, it is assumed that this generates inefficient solutions because it always looks for new points in a given grid. The autors of Edelkamp and Stommel (2012) formulates the inspection problem in three dimensions. Since this technique cannot be utilized to two-dimensional problem formulations, the transformed Monte-Carlo solutions are highlighted. The method is randomized, meaning the sole potential outcome is an efficient one. By viewing the problem as a hitting set, the formulation became compatible with brute force and greedy solutions. The brute-force method is ineffective due to the multitude of possible options. Greedy, on the other hand, can definitely provide a solution. However, the Greedy algorithm (see Algorithm 1) essentially computes local maxima. Additionally, since geometrical aspects are lost, reducing the issue to a hitting set problem has both beneficial and adverse consequences. By contrast, QuadPos works directly on the geometric aspect and is, therefore, capable of calculating a solution immediately. As a result, compared to the other algorithms q_0 mentioned, the QuadPos algorithm covers a wider spectrum of efficient solutions. This hypothesis is expressed in Eq. (13).

$$Hq_0 = q \leq q_0 \text{ or } Hq_a = q > q_0. \tag{13}$$

As already described above, we assume that the problem cannot be solved efficiently, due to the reduction to a hitting set problem. The hypothesis, therefore, applies to the velocity as follows

$$Ht_0 = t \leq t_0 \text{ or } Ht_a = t > t_0. \tag{14}$$

Description of geometric shapes

To be able to carry out the experiment, the geometries listed in Table 1 are selected. These are used in many different dimensions for measurement. According to Kabacoff (2011),

Table 1 Overview of the geometries used

Geometry	Description 1	Description 2	Description 3	Description 4
Square	8 × 16	16 × 8	5 × 5	18 × 18
	A: 128	A: 128	A: 25	A: 324
Isosceles	16 × 8	7.8 × 16	3 × 5	10 × 10
triangle	A: 64	A: 62.399	A: 7.5	A: 50
Trapezoids	7.8 × 16; A: 78	8 × 10; A: 50	3 × 5; A: 7.5	4 x 5; A: 12.5
Hexagon	16 × 14	10 x 8	4 x 4	6 × 4
	Angle: 113.63°/132.74°	Angle: 113.63°/132.74°	Angle: 109.28°/141.42°	Angle: 117.7°/124.6°
	A: 184.8	A: 66	A: 13.199	A: 19.8
Ellipse	Radius: 4	Radius: 7	Radius: 5	Radius: 3.5; Center distance: 8
	Length: 24	Length: 30	Length: 26	Length: 22
	Center distance: 8	Center distance: 8	Center distance: 8	Center distance: 8
	A: 23.0006	A: 194.74233	A: 43.9335	A: 43.9335
Car door without hole	4 × 5	8 × 6	12 × 10	20 × 18
	A: 14.5	A: 34.8	A: 87	A: 261

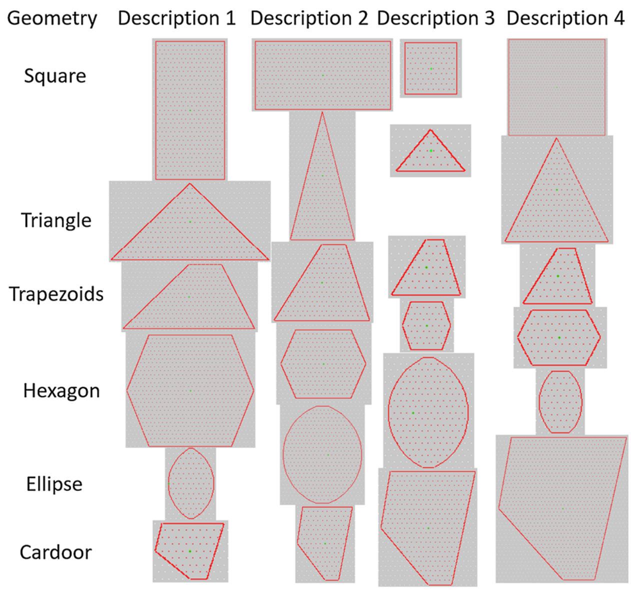


Fig. 9 Geometries used in the experiment

20–30 random samples are usually sufficient to carry out a hypothesis test using bootstrapping.

Table 1 shows the geometric properties of the different geometries. The first two numbers are the dimensions of the geometry. The area is noted in the property A (Area) and was calculated for all geometries using the Gaussian trapezoidal formula. To facilitate a better understanding of the geometries used in this test, they have been depicted in Fig. 9. This concerns the variant described as "Description 1" in Table 1.

Other conditions

A Fujitsu PC with an Intel(R) Core(TM) i7-8565U processor, 8 MB of cache, a 4.6GHz maximum clock speed as well as 32GB RAM was used for this measurement. Algorithms like brute force, sequence, and rollout Monte Carlo can be run with a maximum of 8 threads. Every other algorithm is run by a single thread. Each algorithm applies an interface through which the most effective solution is constantly inserted and modified every frame, ensuring execution with the same data. In the "Scoring" section, see also Algorithm 4. As already introduced in the "Object and grid model" section, the grid is resolved with a resolution of 0.5 and the contours of the geometries with a factor of 0.25.

Operationalization and evaluation of results

Quality measures

In "Scoring" section, the resulting solution is assessed using the Eq. (12). It is calculated by taking the ratio of points covered to total points. The number of solution images is also required. The surface of the geometries can be extracted using Eq. (1).

Duration of the calculations

The running time of the algorithms is measured in milliseconds. Prior to the first running of the related algorithm, the begining point is recorded. The end point is determined immediately after the operation has fully completed or just after the algorithm's maximum allowed ten seconds have passed. The time required for the calculation in milliseconds is the difference between the beginning and ending points.

Results

The forms mentioned in "Description of geometric shapes" section were used for the measurements. The results are shown in the Appendix. As can be observed, the brute force algorithm reaches a maximum coverage of around 98% in ten seconds. Thus, the outcome is insufficient. The approach based on Sequential Monte Carlo can achieve 100% coverage in many scenarios, but in each scenario the number of images required is more than twice as large as the number of other approaches. It is additionally apparent that all algorithms except QuadPos and Sequential Monte Carlo failed to solve the isosceles triangle 1. The "Discussion" section offers substantial information upon that. The Appendix contains a table of measurement findings. The evaluation in R can be found in the repository https://github.com/bschw4rz3/OptimizationOfImageAcquisition in the "Data" folder.

Evaluation of the quality

The solutions must initially be evaluated for quality using the Eq. (12) from "Scoring" section, so that the hypotheses could be verified with a simulation-based hypothesis test. Table 2 supplies an overview of the results obtained.

The quality values for the algorithms Greedy, Brute Force, Sequential Monte Carlo, Rollout Monte Carlo and Gridded Randomize were simulated a total of 1000 times. It was intended to show that all simulations had a normal distribution using the Kolmogorov–Smirnov and Shapiro–Wilk tests. This has not been achieved in any simulation. The validation of the normal distribution of Greedy using the Shapiro–Wilk test failed and the distribution check of Brute Force, Gridded Randomize, Rollout and Sequence Montecarlo could not be confirmed by both tests because there are deviations at the tail of the distribution. However, the authors are of the opinion that these are indeed normally distributed simulations and that the deviations at the edge of the distribution should not play a significant role in the hypothesis test. In order to show the deviations at the edge of the distribution, the histograms

Table 2 Overview of the evaluated quality of the generated solutions

Geometry	QuadPos	Greedy	Brute force	Sequential Monte-Carlo	Rollout Monte-Carlo	Gridded random-ization
Elipse 1	**1.000**	**1.000**	0.424	0.540	**1.000**	**1.000**
Elipse 2	**0.837**	0.454	0.116	0.249	0.473	0.640
Elipse 3	**0.172**	0.120	0.106	0.070	0.110	0.150
Elipse 4	**0.531**	0.354	0.293	0.106	0.354	**0.531**
Hexagon 1	**0.231**	0.147	0.186	0.048	0.143	0.143
Hexagon 2	**0.058**	0.046	0.069	0.009	0.043	0.043
Hexagon 3	**0.561**	**0.561**	0.538	0.070	0.374	0.374
Hexagon4	**0.531**	0.354	0.517	0.101	0.354	0.087
Isosceles triangle 1	**1.000**	0.968	0.390	0.350	0.968	0.107
Isosceles triangle 2	**0.663**	0.415	0.344	0.084	0.415	0.415
Isosceles triangle 3	**0.531**	0.417	0.493	0.125	0.348	0.104
Isosceles triangle 4	**1.000**	**1.000**	0.984	0.432	**1.000**	**1.000**
Square 1	**0.907**	0.605	0.199	0.089	0.454	0.454
Square 2	**0.557**	0.424	0.208	0.071	0.371	0.371
Square 3	**1.000**	**1.000**	0.480	0.787	**1.000**	**1.000**
Square 4	0.107	**0.121**	0.108	0.016	0.062	0.062
Trapezoids 1	**0.196**	0.130	0.190	0.025	0.122	0.122
Trapezoids 2	0.425	0.472	**0.477**	0.079	0.387	0.387
Trapezoids 3	0.354	0.266	**0.380**	0.048	0.266	0.266
Trapezoids 4	0.561	0.421	**0.602**	0.062	0.421	0.421
Car Door 1	**0.411**	0.308	0.217	0.068	0.247	0.247
Car Door 2	**0.493**	0.423	0.324	0.087	0.296	0.296
Car Door 3	**0.616**	0.462	0.182	0.114	0.493	0.493
Car Door 4	**0.793**	0.528	0.096	0.137	0.444	0.528

The best rating was marked in bold

of the simulations were added to the Appendix. A p-value below 0.033 was determined for all hypothesis tests.

The overall simulations of the algorithms are shown in Fig. 10. The confidence interval (acceptance region) is represented by the black section, while the rejection region (critical region) is depicted in light gray. The p value is far right of center, even beyond the rejection range. Furthermore, to ensure the independence of the data, a Friedman test was executed. The test resulted in a p-value very close to zero, indicating significant differences between the groups. Additionally, a Kendall correlation of 0.581 was observed, indicating a strong positive correlation between the algorithm and quality assessment. Thus, the Hq_0 value may be ignored.

Evaluation of the runtime

The speed was also assessed using a simulation-based mortgage test. As a normal distribution was disproved by both the Kolmogorov-Smirnov test and the Shapiro–Wilk test, it turned out that the simulated speeds of the algorithms were typically not normally distributed. These are fairly left-skewed distributions, as shown by a graphical examination of the simulation's histogram. As a result, no simulation-based hypothesis test can be carried out. A significant difference between QuadPos and the other algorithms' speeds was, therefore, identified using the Mann–Whitney U test. The test showed that there are significant differences between the speeds of Greedy, Gridded Randomized and QuadPos. Greedy's average time is 2.2 s, while QuadPos needs an average of 24.13 milliseconds to generate a solution. This is only exceeded by Gridded Random's average time of 6.8 milliseconds. This means Ht_0 cannot be rejected.

The speeds of QuadPos, Greedy, Gridded Randomize, and Rollout Monte-Carlo are shown in Fig. 11. Due to the fact that the brute force and sequence Monte Carlo rates were almost always above the ten-second limit, they were not included in this analysis. In order to better compare the relevant times of QuadPos, Gridded Randomize and Greedy, the scale is only visualized up to 1 s.

Discussion

Especially in comparison to the algorithms Greedy, Brute Force, Rollout, and Sequence Monte-Carlo, QuadPos approach delivers solutions that possess a significantly higher

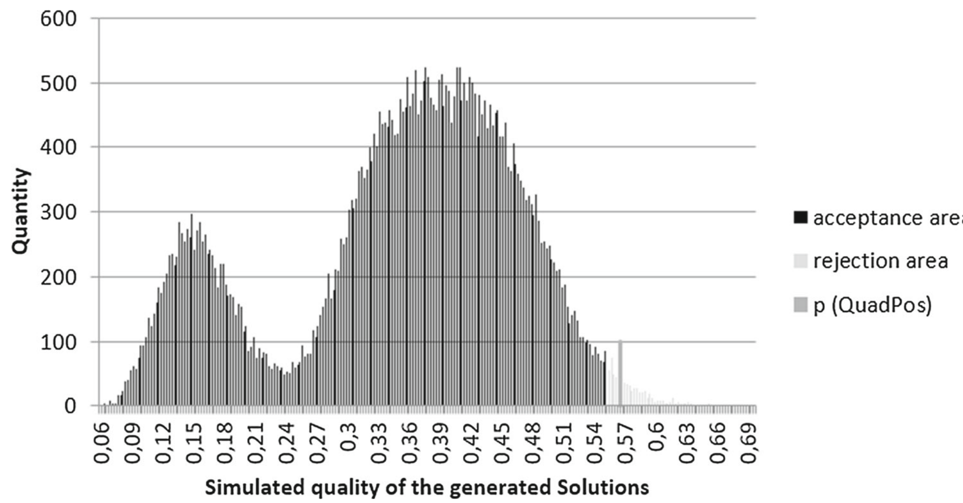


Fig. 10 Histogram of the simulations including the rejection range with p. The simulations of the algorithms Greedy, Brute Force, Monte-Carlos and Gridded Random are shown in black. The rejection area is shown in light gray. The p of QuadPos is shown in dark gray

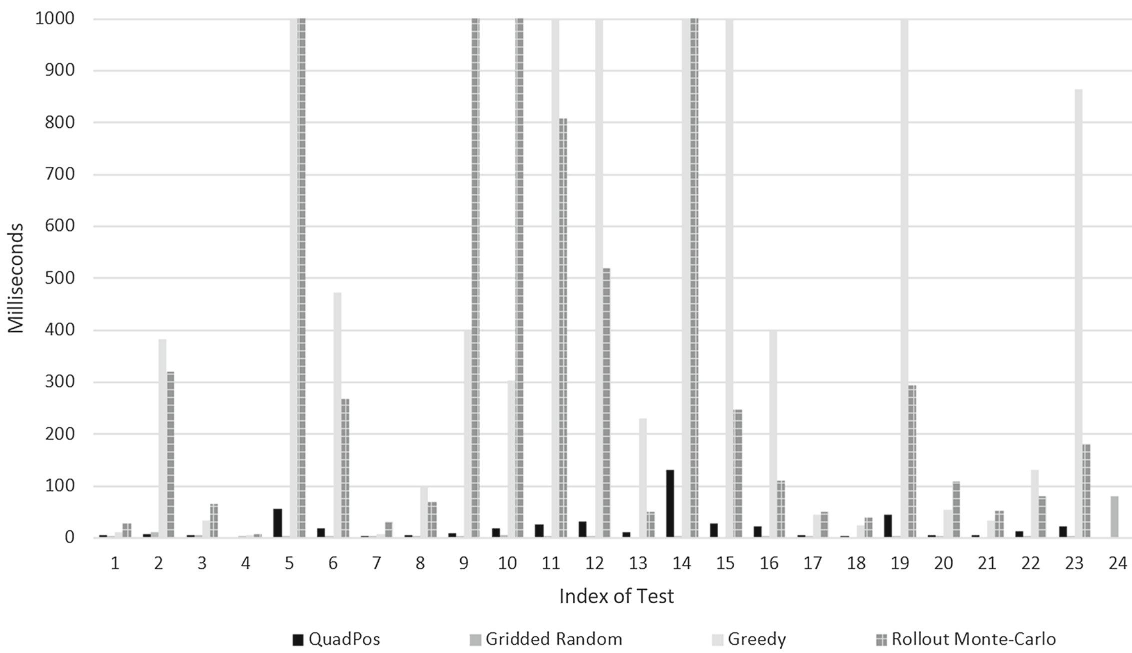


Fig. 11 Speeds of QuadPos, Greedy and Rollout Monte-Carlo clustered by passes ordered by speed of greedy

quality. When contrasted to Greedy and Gridded Randomize, QuadPos was capable of minimizing the number of images by an average of 3 and 4. Nonetheless, several factors must be addressed while interpreting the results. For instance, the geometry of the isosceles triangles 1 and 4 was exclusively solved by the QuadPos and Sequence Montecarlo approach. Furthermore, the other algorithms were unable to locate a point in the tip's grid, ensuring that the focus points were also covered. No point can be located if no valid parameter exists, as the algorithms only search for points on the grid. This problem could be solved using QuadPos or a random based approach, that is additionally capable of creating new points outside the grid. It might be argued that these geometries deplete the database. Since this problem was three-dimensional, the algorithm from Edelkamp and Stommel (2012) could not be highlighted accurately. The Monte Carlo algorithms are just simplified derivations. Also, each geometry was only solved once with each Monte Carlo approach because the algorithms are based on random variables. It's highly questionable that a second attempt would provide different outcomes. This, though, remains certainly an option. In addition, a single "classic" greedy algorithm was selected as a sample of the greedy algorithms for such research. This greedy algorithm might have produced different results if it had been modified. The Gridded Random approach generates good results much faster than QuadPos. This is because in most cases the center of the grid can simply be used as the image position. On average, 20 milliseconds

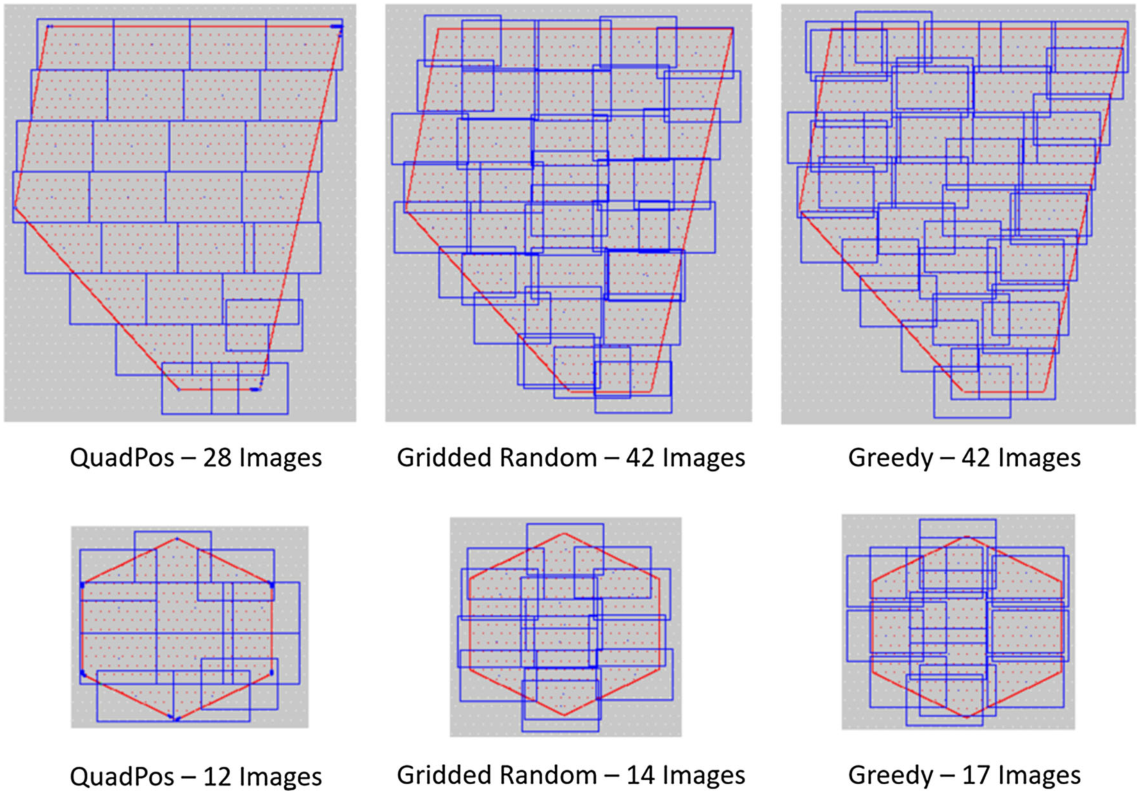


Fig. 12 Presentation of various solution approaches from QuadPos, Gridded Randomize and Greedy

can be saved compared to QuadPos. However, the generated solution is also significantly worse than that of QuadPos.

In Fig. 12, the different solutions from QuadPos, Gridded Randomize and Greedy are shown in two parts for comparison. This shows that the reduction of images can be achieved by reducing the overlapping images. While there is little overlap in QuadPos, the overlaps of the images in the Gridded Randomize and Greedy solutions are greatly increased due to the reckless positioning of the images. With Gridded Random, this comes about through the division into sections and the random setting. In Greedy, this is achieved by locally maximizing the image setting. Through the compromise of a loose grid and taking the beginning into account, QuadPos manages to very elegantly reduce the number of images that are required for full coverage.

Conclusion and outlook

This paper explores the efficient quality assurance of various geometric objects using a WLI by minimizing the number of required image captures. The need for such an algorithm arises from the extended recording times associated with various free-form sheet metal parts. Reducing the image count enhances part throughput, thereby increasing the economic efficiency of a WLI. A distinctive aspect of the WLI employed in this context is the requirement for focus points to consistently reside on the part's surface. This paper formulates this challenge mathematically, necessitating a comprehensive literature review to identify potential solutions. Many of the solution strategies had to undergo modifications to address the problem adequately.

In response to these requirements, the novel QuadPos approximation allgorithm, was developed to accommodate the idiosyncrasies of automatic WLI focusing. Two hypotheses were formulated: QuadPos generates the best solution and is the fastest algorithm. To assess the quality of a solution, an evaluation formula was developed. Following extensive testing with various geometric parts, a simulation-based hypothesis test was employed to demonstrate that the QuadPos approximation method significantly outperforms brute force, greedy, and gridded random, along with other Monte Carlo algorithms, all while considering the specificities of WLI. It is important to note, that QuadPos exhibits slightly slower performance compared to gridded random.

The QuadPos approach holds significant promise and may pique further research interest within this field. It's plausible that performance and efficiency could be further enhanced. Moreover, the realm of free-form parts presents additional challenges, such as components with holes within their contours, which were not addressed in this study. Furthermore, the need for an algorithm that efficiently covers a part extends beyond quality assurance in sheet metal using WLI. Additional domains, like quality control in natural products such as leather or the examination of ship hulls, have been identified in the literature as potential areas of application. The algorithm requirements may vary, necessitating similar attention to domain-specific nuances as with the focus points in WLI.

Acknowledgements This research was partly funded by the Czech Science Foundation Grant Number 22-30043S.

Funding Open access publishing supported by the National Technical Library in Prague.

Data availability An implementation of all algorithms including the QuadPos algorithm and the evaluation of the data can be found here https://github.com/bschw4rz3/OptimizationOfImageAcquisition. The evaluation of the data was stored in the "Data" folder of the repository.

Appendix

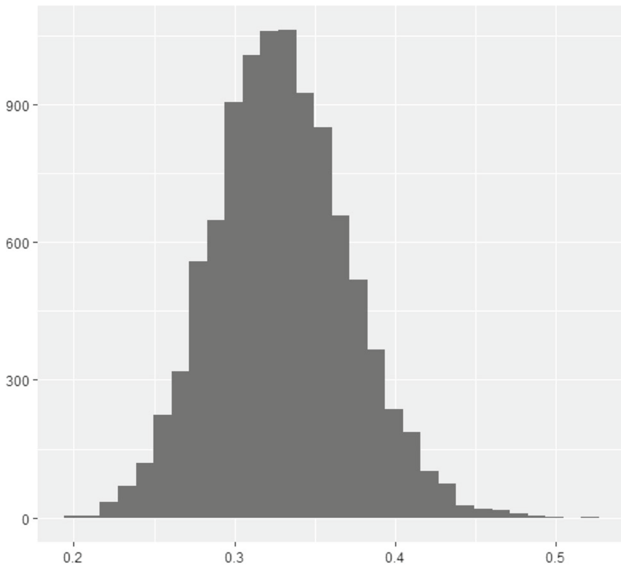
Raw data of the experiment

Algorithm	Geometry	Time (ms)	Number of images	Coverage rate
QuadPos	Elipse 1	6	5	1.000
Greedy	Elipse 1	11	4	1.000
Brute Force	Elipse 1	121,198	3	0.424
Sequence Monte-Carlo	Elipse 1	111,134	10	0.992
Rollout Monte-Carlo	Elipse 1	28	5	1.000
Gridded Random	Elipse 1	3	5	1.000
QuadPos	Isosceles triangle 1	10	9	1.000
Greedy	Isosceles triangle 1	398	12	0.968
Brute Force	Isosceles triangle 1	11,1218	3	0.390
Sequence Monte-Carlo	Isosceles triangle 1	62,690	45	1.000
Rollout Monte-Carlo	Isosceles triangle 1	20,332	14	0.968
Gridded Random	Isosceles triangle 1	3	1	0.107
QuadPos	Hexagon 1	57	23	1.000
Greedy	Hexagon 1	6593	36	1.000
Brute Force	Hexagon 1	115,698	3	0.186
Sequence Monte-Carlo	Hexagon 1	46,530	109	0.994
Rollout Monte-Carlo	Hexagon 1	1303	37	1.000
Gridded Random	Hexagon 1	3	37	1.000
QuadPos	Isosceles triangle 2	46	10	1.000
Greedy	Isosceles triangle 2	1992	16	1.000
Brute Force	Isosceles triangle 2	112,016	3	0.344
Sequence Monte-Carlo	Isosceles triangle 2	46,036	79	1.000
Rollout Monte-Carlo	Isosceles triangle 2	294	16	1.000
Gridded Random	Isosceles triangle 2	3	16	1.000
QuadPos	Trapezoids 1	28	10	1.000
Greedy	Trapezoids 1	1334	15	1.000
Brute Force	Trapezoids 1	112,643	3	0.292
Sequence Monte-Carlo	Trapezoids 1	39,642	79	1.000
Rollout Monte-Carlo	Trapezoids 1	248	16	1.000
Gridded Random	Trapezoids 1	2	16	1.000
QuadPos	Square 1	26	12	1.000
Greedy	Square 1	3619	18	1.000
Brute Force	Square 1	107,064	3	0.199
Sequence Monte-Carlo	Square 1	44,264	122	1.000
Rollout Monte-Carlo	Square 1	809	24	1.000
Gridded Random	Square 1	3	24	1.000

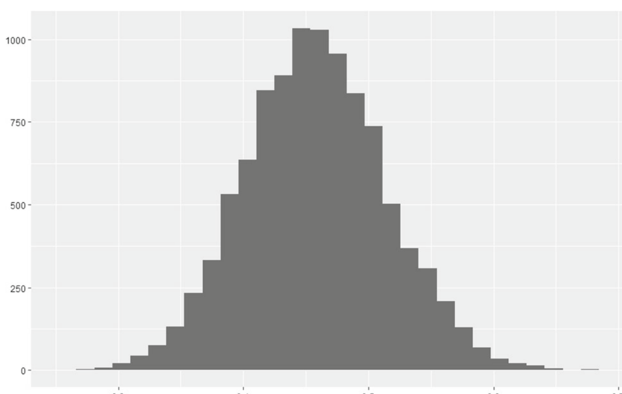
Algorithm	Geometry	Time (ms)	Number of images	Coverage rate
QuadPos	Car Door 1	5	3	1.000
Greedy	Car Door 1	34	4	1.000
Brute Force	Car Door 1	121,265	3	0.528
Sequence Monte-Carlo	Car Door 1	96,094	18	1.000
Rollout Monte-Carlo	Car Door 1	52	5	1.000
Gridded Random	Car Door 1	2	5	1.000
QuadPos	Elipse 2	8	13	1.000
Greedy	Elipse 2	383	24	1.000
Brute Force	Elipse 2	127,073	3	0.116
Sequence Monte-Carlo	Elipse 2	72,529	43	0.981
Rollout Monte-Carlo	Elipse 2	320	23	1.000
Gridded Random	Elipse 2	12	17	1.000
QuadPos	Isosceles triangle 3	12	8	1.000
Greedy	Isosceles triangle 3	303	10	0.982
Brute Force	Isosceles triangle 3	116,670	3	0.493
Sequence Monte-Carlo	Isosceles triangle 3	68,813	34	1.000
Rollout Monte-Carlo	Isosceles triangle 3	19,683	12	0.982
Gridded Random	Isosceles triangle 3	5	1	0.104
QuadPos	Hexagon 2	19	11	1.000
Greedy	Hexagon 2	472	14	1.000
Brute Force	Hexagon 2	119,287	3	0.324
Sequence Monte-Carlo	Hexagon 2	50,751	69	1.000
Rollout Monte-Carlo	Hexagon 2	268	15	1.000
Gridded Random	Hexagon 2	3	15	1.000
QuadPos	Isosceles triangle 4	6	3	1.000
Greedy	Isosceles triangle 4	55	5	1.000
Brute Force	Isosceles triangle 4	113,166	3	0.984
Sequence Monte-Carlo	Isosceles triangle 4	99,188	13	1.000
Rollout Monte-Carlo	Isosceles triangle 4	108	5	1.000
Gridded Random	Isosceles triangle 4	4	5	1.000
QuadPos	Trapezoids 2	23	10	1.000
Greedy	Trapezoids 2	401	9	1.000
Brute Force	Trapezoids 2	109,928	3	0.477
Sequence Monte-Carlo	Trapezoids 2	52,239	54	1.000
Rollout Monte-Carlo	Trapezoids 2	110	11	1.000
Gridded Random	Trapezoids 2	3	11	1.000
QuadPos	Square 2	32	16	1.000
Greedy	Square 2	4697	21	1.000
Brute Force	Square 2	105,714	3	0.208
Sequence Monte-Carlo	Square 2	31,094	126	1.000
Rollout Monte-Carlo	Square 2	520	24	1.000
Gridded Random	Square 2	4	24	1.000
QuadPos	Car Door 2	13	6	1.000
Greedy	Car Door 2	132	7	1.000
Brute Force	Car Door 2	113,000	3	0.328
Sequence Monte-Carlo	Car Door 2	71,783	34	1.000
Rollout Monte-Carlo	Car Door 2	80	10	1.000
Gridded Random	Car Door 2	3	10	1.000
QuadPos	Elipse 3	6	7	1.000
Greedy	Elipse 3	33	10	1.000
Brute Force	Elipse 3	126,278	3	0.264
Sequence Monte-Carlo	Elipse 3	105,365	17	0.995
Rollout Monte-Carlo	Elipse 3	66	11	1.000
Gridded Random	Elipse 3	5	8	1.000
QuadPos	Hexagon 3	3	2	1.000
Greedy	Hexagon 3	7	2	1.000

Algorithm	Geometry	Time (ms)	Number of images	Coverage rate
Brute Force	Hexagon 3	125,514	1	0.538
Sequence Monte-Carlo	Hexagon 3	102,507	16	1.000
Rollout Monte-Carlo	Hexagon 3	32	3	1.000
Gridded Random	Hexagon 3	4	3	1.000
QuadPos	Trapezoids 3	6	3	1.000
Greedy	Trapezoids 3	46	4	1.000
Brute Force	Trapezoids 3	121,445	2	0.716
Sequence Monte-Carlo	Trapezoids 3	81,637	22	1.000
Rollout Monte-Carlo	Trapezoids 3	51	4	1.000
Gridded Random	Trapezoids 3	3	4	1.000
QuadPos	Square 3	12	4	1.000
Greedy	Square 3	230	6	1.000
Brute Force	Square 3	114,129	2	0.480
Sequence Monte-Carlo	Square 3	64,310	35	1.000
Rollout Monte-Carlo	Square 3	51	6	1.000
Gridded Random	Square 3	2	6	1.000
QuadPos	Car Door 3	23	12	1.000
Greedy	Car Door 3	864	16	1.000
Brute Force	Car Door 3	112,869	3	0.182
Sequence Monte-Carlo	Car Door 3	46,225	65	1.000
Rollout Monte-Carlo	Car Door 3	181	15	1.000
Gridded Random	Car Door 3	4	15	1.000
QuadPos	Elipse 4	2	2	1.000
Greedy	Elipse 4	6	3	1.000
Brute Force	Elipse 4	129,854	2	0.551
Sequence Monte-Carlo	Elipse 4	98,654	10	1.000
Rollout Monte-Carlo	Elipse 4	8	3	1.000
Gridded Random	Elipse 4	3	2	1.000
QuadPos	Hexagon 4	6	4	1.000
Greedy	Hexagon 4	99	6	1.000
Brute Force	Hexagon 4	103,823	2	0.517
Sequence Monte-Carlo	Hexagon 4	83,993	21	1.000
Rollout Monte-Carlo	Hexagon 4	70	6	1.000
Gridded Random	Hexagon 4	4	4	0.163
QuadPos	Trapezoids 4	3	3	1.000
Greedy	Trapezoids 4	25	4	1.000
Brute Force	Trapezoids 4	103,874	2	0.716
Sequence Monte-Carlo	Trapezoids 4	78,910	27	1.000
Rollout Monte-Carlo	Trapezoids 4	39	4	1.000
Gridded Random	Trapezoids 4	2	4	1.000
QuadPos	Square 4	132	35	1.000
Greedy	Square 4	19,870	27	0.877
Brute Force	Square 4	92,300	3	0.108
Sequence Monte-Carlo	Square 4	33,565	238	0.999
Rollout Monte-Carlo	Square 4	2818	60	1.000
Gridded Random	Square 4	3	60	1.000
QuadPos	Car Door 4	88	28	1.000
Greedy	Car Door 4	11,169	42	1.000
Brute Force	Car Door 4	94,277	3	0.096
Sequence Monte-Carlo	Car Door 4	32,940	161	0.996
Rollout Monte-Carlo	Car Door 4	2384	50	1.000
Gridded Random	Car Door 4	80	42	1.000

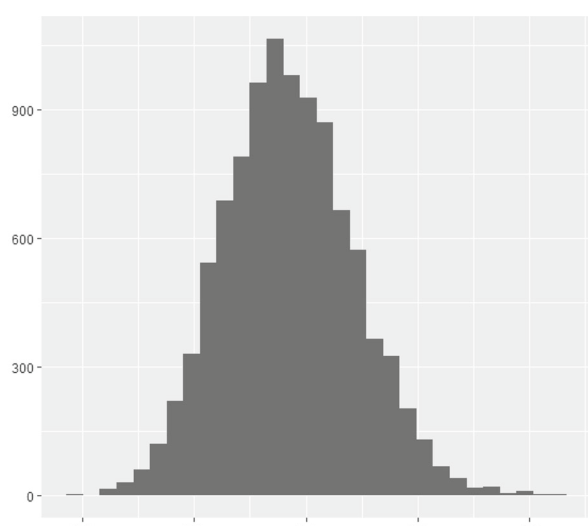
Normal distribution of simulations



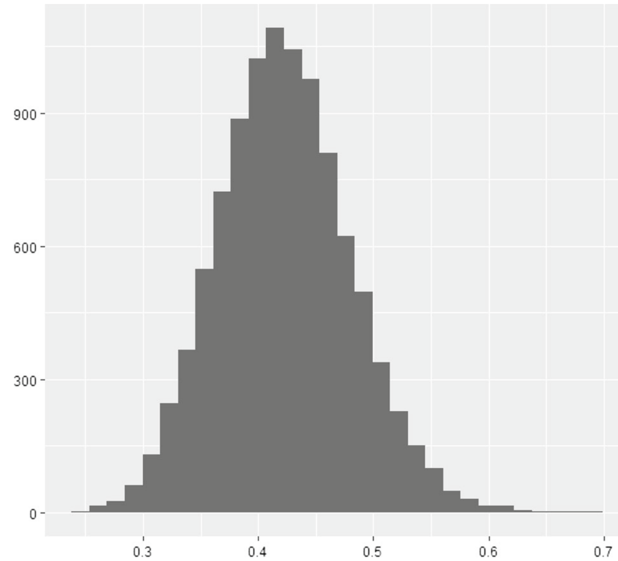
Normal distribution of Brute Force simulation



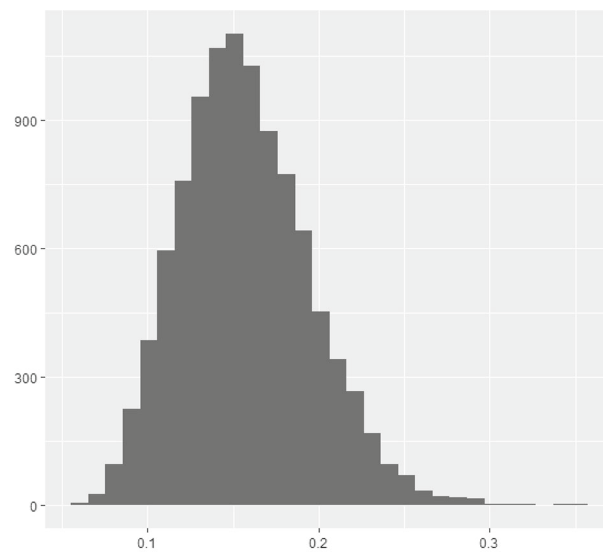
Normal distribution of Greedy simulation



Normal distribution of Gridded Randomization simulation



Normal distribution of Rollout Montecarlo simulation



Normal distribution of Sequentail Montecarlo simulation

Overview of the search results for "surface inspection"

ID	Title	Author	Year	Category
1	Free-form surface inspection techniques state of the art review	Yadong Li and Peihua Gu	2004	Literature summary
2	Review of vision-based steel surface inspection systems	Nirbhar Neogi, Dusmanta K Mohanta & Pranab K Dutta	2014	Literature summary
3	Automated surface inspection for statistical textures	Du-Ming Tsai and Tse-Yun Huang	2003	Surface detection
4	Image-Based Surface Defect Detection Using Deep Learning: A Review	Bhatt, Prahar and Malhan, Rishi and Rajendran, Pradeep and Shah, Brual and Thakar, Shantanu and Yoon, Yeo Jung and Gupta, Satyandra	2021	Literature summary
5	Surface Defect Detection Methods for Industrial Products: A Review	Chen, Yajun and Ding, Yuanyuan and Fan, Zhao and Zhang, Erhu and Wu, Zhangnan and Shao, Linhao	2021	Literature summary
6	Surface Defect Detection Methods for Industrial Products: A Review	Melanthota, S.K., Gopal, D., Chakrabarti, S.	2022	Literature summary
7	Automated surface inspection of cold-formed micro-parts	Bernd Scholz-Reiter and Daniel Weimer and Hendrik Thamer	2012	Surface detection
8	A Generic Deep-Learning-Based Approach for Automated Surface Inspection	Ren, Ruoxu and Hung, Terence and Tan, Kay Chen	2018	Surface detection
9	Automated Surface Inspection Using Gabor Filters	Tsa, D.-M. and Wu, S.-K.	2000	Surface detection
10	Automatic surface inspection using wavelet reconstruction	Du-Ming Tsai and Bo Hsiao	2001	Surface detection
11	Anomaly detection with convolutional neural networks for industrial surface inspection	Benjamin Staar and Michael Lütjen and Michael Freitag	2019	Surface detection
12	Receding horizon path planning for 3D exploration and surface inspection	Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova and Roland Siegwart	2018	Inspection planning
13	Real-time surface inspection by texture	Topi Mäenpää and Markus Turtinen and Matti Pietikäinen	2003	Surface detection

ID	Title	Author	Year	Category
14	A Hierarchical Extractor-Based Visual Rail Surface Inspection System	Gan, Jinrui and Li, Qingyong and Wang, Jianzhu and Yu, Haomin	2017	Full inspection system
15	A Simplified Computer Vision System for Road Surface Inspection and Maintenance	Quintana, Marcos and Torres, Juan and Menéndez, José Manuel	2016	Full inspection system
16	A smart surface inspection system using faster R-CNN in cloud-edge computing environment	Yuanbin Wang and Minggao Liu and Pai Zheng and Huayong Yang and Jun Zou	2020	Surface detection
17	Enhanced discrete particle swarm optimization path planning for UAV vision-based surface inspection	Manh Duong Phung and Cong Hoang Quach and Tran Hiep Dinh and Quang Ha	2017	Inspection planning
18	Convolutional networks for voting-based anomaly classification in metal surface inspection	Natarajan, Vidhya and Hung, Tzu-Yi and Vaikundam, Sriram and Chia, Liang-Tien	2017	Surface detection
19	Adaptive surface inspection via interactive evolution	P. Caleb-Solly and J.E. Smith	2007	Other
20	Automatic localization and comparison for free-form surface inspection	Yadong Li and Peihua Gu	2006	Localization methods
21	Semi-supervised anomaly detection with dual prototypes autoencoder for industrial surface inspection	Jie Liu and Kechen Song and Mingzheng Feng and Yunhui Yan and Zhibiao Tu and Liu Zhu	2021	Surface detection
22	Detecting Change for Multi-View, Long-Term Surface Inspection	Stent, Simon and Gherardi, Riccardo and Stenger, Bjorn and Cipolla, Roberto	2015	Surface detection
23	A Generic Semi-Supervised Deep Learning-Based Approach for Automated Surface Inspection	Zheng, Xiaoqing and Wang, Hongcheng and Chen, Jie and Kong, Yaguang and Zheng, Song	2020	Surface detection
24	System Architecture for Real-Time Surface Inspection Using Multiple UAVs	Hoang, Van Truong and Phung, Manh Duong and Dinh, Tran Hiep and Ha, Quang P.	2020	Inspection planning
25	Surface Inspection System of Steel Strip Based on Machine Vision	Tang, Bo and Kong, Jian-yi and Wang, Xingdong and Chen, Li	2009	Surface detection

ID	Title	Author	Year	Category
26	Design of online surface inspection system of hot rolled strips	Guifang Wu and Hoonsung Kwak and Seyoung Jang and Ke Xu and Jinwu Xu	2008	Full inspection system
27	Coverage path planning with targetted viewpoint sampling for robotic free-form surface inspection	Emile Glorieux and Pasquale Franciosa and Dariusz Ceglarek	2020	Inspection planning
28	A Method for Automatic Surface Inspection Using a Model-Based 3D Descriptor	Madrigal, Carlos A. and Branch, John W. and Restrepo, Alejandro and Mery, Domingo	2017	Full inspection system
29	Automated surface inspection for steel products using computer vision approach	Jiaqi Xi and Lifeng Shentu and Jikang Hu and Mian Li	2017	Full inspection system
30	Automatic inspection of metallic surface defects using genetic algorithms	H Zheng and L.X Kong and S Nahavandi	2002	Full inspection system
31	Robust localization to align measured points on the manufactured surface with design surface for freeform surface inspection	Vahid Mehrad and Deyi Xue and Peihua Gu	2014	Localization methods
32	Automatic inspection data collection of building surface based on BIM and UAV	Yi Tan and Silin Li and Hailong Liu and Penglu Chen and Zhixiang Zhou	2021	Full inspection system
33	Wind turbine blade surface inspection based on deep learning and UAV-taken images	Xu, Donghua and Wen, Chuanbo and Liu, Jihui	2019	Full inspection system
34	A DEFECT DETECTION SCHEME FOR WEB SURFACE INSPECTION	IIVARINEN, JUKKA and HEIKKINEN, KATRIINA and RAUHAMAA, JUHANI and VUORIMAA, PETRI and VISA, ARI	2020	Surface detection
35	An automated shearography system for cylindrical surface inspection	Yimin Ye and Ke Ma and Hui Zhou and Dwayne Arola and Dongsheng Zhang	2019	Shearography
36	Vision-Based Surface Inspection System for Bearing Rollers Using Convolutional Neural Networks	Wen, Shengping and Chen, Zhihong and Li, Chaoxian	2018	Surface detection

ID	Title	Author	Year	Category
37	An Optical Surface Inspection and Automatic Classification Technique Using the Rotated Wavelet Transform	Borwankar, Raunak and Ludwig, Reinhold	2018	Surface detection
38	A Deep Extractor for Visual Rail Surface Inspection	Zhang, Ziwen and Liang, Mangui and Wang, Zhe	2021	Surface detection
39	Recent advances in surface defect inspection of industrial products using deep learning techniques	Zheng, X., Zheng, S., Kong, Y. et al.	2021	Literature summary
40	An improved adaptive sampling strategy for freeform surface inspection on CMM	He, G., Sang, Y., Pang, K. et al.	2018	Literature summary
41	Online Rail Surface Inspection Utilizing Spatial Consistency and Continuity	Gan, Jinrui and Wang, Jianzhu and Yu, Haomin and Li, Qingyong and Shi, Zhiping	2018	Surface detection
42	A real-time surface inspection system for precision steel balls based on machine vision	Yi-Ji Chen and Jhy-Cherng Tsai and Ya-Chen Hsu	2016	Full inspection system
43	Convolutional Neural Network Based Surface Inspection System for Non-patterned Welding Defects	Park, JK., An, WH. & Kang, DJ	2019	Surface detection
44	A computer vision system for automatic steel surface inspection	Yung-Chun Liu and Yu-Lu Hsu and Yung-Nien Sun and Song-Jan Tsai and Chiu-Yi Ho and Chung-Mei Chen	2010	Surface detection
45	Feature-Driven Viewpoint Placement for Model-Based Surface Inspection	Mosbach, D., Gospodnetić, P., Rauhut, M. et al	2021	Inspection planning
46	Machine learning-based imaging system for surface defect inspection	Park, JK., Kwon, BK., Park, JH. et al	2016	Surface detection
47	Comparison of dimensionality reduction methods for wood surface inspection	Matti Niskanen and Olli Silven	2003	Dimensions reduktion
48	Flexible Surface Inspection Planning Pipeline	Gospodnetic, Petra and Mosbach, Dennis and Rauhut, Markus and Hagen, Hans	2020	Inspection planning

ID	Title	Author	Year	Category
49	Assessment of the influence of adaptive components in trainable surface inspection systems	Eitzinger, C., Heidl, W., Lughofer, E. et al.	2009	Surface detection
50	View and sensor planning for multi-sensor surface inspection	Marc Gronle and Wolfgang Osten	2016	Inspection planning
51	Image acquisition techniques for automatic visual inspection of metallic surfaces	Franz Pernkopf and Paul O'Leary	2016	Inspection planning
52	Machine vision system for curved surface inspection	Lee, MF., de Silva, C., Croft, E. et al.	2000	Surface detection
53	High-efficient view planning for surface inspection based on parallel deep reinforcement learning	Yuanbin Wang and Tao Peng and Wenhu Wang and Ming Luo	2023	Inspection planning
54	Development of Defect Classification Algorithm for POSCO Rolling Strip Surface Inspection System	Choi, Keesug and Koo, Kyungmo and Lee, Jin S.	2006	Surface detection
55	Path planning for surface inspection on a robot-based scanning system	Wu, Qian and Lu, Jinyan and Zou, Wei and Xu, De	2015	Inspection planning
56	Surface defects inspection of cold rolled strips based on neural network	Ge-Wen Kang and Hong-Bing Liu	2005	Surface detection
57	Automatic Optical Surface Inspection of Wind Turbine Rotor Blades using Convolutional Neural Networks	Dimitri Denhof and Benjamin Staar and Michael Lütjen and Michael Freitag	2019	Surface detection
58	On-line evolving image classifiers and their application to surface inspection	Edwin Lughofer	2010	Surface detection
59	Sweep scan path planning for efficient freeform surface inspection on five-axis CMM	Zi Zhou and Yang Zhang and Kai Tang	2016	Inspection planning

ID	Title	Author	Year	Category
60	A Generative Adversarial Network Based Framework for Unsupervised Visual Surface Inspection	Zhai, Wei and Zhu, Jiang and Cao, Yang and Wang, Zengfu	2018	Surface detection
61	Design of multi-scale receptive field convolutional neural network for surface inspection of hot rolled steels	Di He and Ke Xu and Dadong Wang	2019	Surface detection
62	On improving performance of surface inspection systems by online active learning and flexible classifier updates	Weigl, E., Heidl, W., Lughofer, E. et al.	2016	Surface detection
63	Supervised Machine Learning Based Surface Inspection by Synthetizing Artificial Defects	Haselmann, Matthias and Gruber, Dieter	2017	Surface detection
64	A deep learning-based approach for the automated surface inspection of copper clad laminate images	Zheng, X., Chen, J., Wang, H. et al.	2021	Surface detection

ID	Title	Author	Year	Category
65	Mechanisms of Autonomous Pipe-Surface Inspection Robot with Magnetic Elements	Suzuki, Masayuki and Yukawa, Toshihiro and Satoh, Yuichi and Okano, Hideharu	2006	Full inspection system
66	Adaptive sampling point planning for free-form surface inspection under multi-geometric constraints	Bowen Yi and Fan Qiao and Nuodi Huang and Xiaosun Wang and Shijing Wu and Dirk Biermann	2021	Inspection planning
67	Feature extraction based on contourlet transform and its application to surface inspection of metals	Yonghao Ai and Ke Xu	2012	Surface detection
68	Abnormality detection strategies for surface inspection using robot mounted laser scanners	Sara Sharifzadeh and Istvan Biro and Niels Lohse and Peter Kinnell	2018	Surface detection
69	A Compact Convolutional Neural Network for Surface Defect Inspection	Huang, Yibin and Qiu, Congying and Wang, Xiaonan and Wang, Shijun and Yuan, Kui	2020	Surface detection

ID	Title	Author	Year	Category
70	Three-Dimensional Inner Surface Inspection System Based on Circle-Structured Light	Ye, Zhu and Lianpo, Wang and Yong-gang, Gu and Songlin, Bi and Chao, Zhai and Jiang, Baoyang and Ni, Jun	2018	Full inspection system
71	Surface inspection problems in thermoelectric testing	Abouellail, Ahmed and Obach, Igor and Soldatov, Andrey and Soldatov, Alexey	2017	
72	Automated metal surface inspection through machine vision	W-Y Wu and C-C Hou	2003	Surface detection
73	Defective samples simulation through adversarial training for automatic surface inspection	Lizhe Liu and Danhua Cao and Yubin Wu and Taoran Wei	2019	Surface detection
74	Real-time aspects of SOM-based visual surface inspection	Matti Niskanen and Hannu Kauppinen and Olli Silven	2002	Surface detection

References

Alsuwaiyel, M. H. (2016). The Greedy approach. In *Algorithms: Design techniques and analysis band 7 Von Lecture Notes Series on Computing, Chap. 7* 2nd edn. (pp. 201–207). World Scientific.

Andreas Bircher, K. A. H. O., Mina, Kamel, & Siegwart, R. (2018). Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots, 42*, 1573–7527. https://doi.org/10.1007/s10514-016-9610-0

Ben Abdallah, H., Jovančević, I., Orteu, J. J., & Brèthes, L. (2019). Automatic inspection of aeronautical mechanical assemblies by matching the 3D CAD model and real 2D images. *Journal of Imaging, 5*(10), 81. https://doi.org/10.3390/jimaging5100081

Bhatt, P., Malhan, R., Rajendran, P., Shah, B., Thakar, S., Yoon, Y. J., & Gupta, S. (2021). Image-based surface defect detection using deep learning: A review. *Journal of Computing and Information Science in Engineering, 21*, 1–23. https://doi.org/10.1115/1.4049535

Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., & Siegwart, R. (2018). Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots, 42*, 291–306. https://doi.org/10.1007/s10514-016-9610-0

Bong, H. Q., Truong, Q. B., Nguyen, H. C., & Nguyen, M. T. (2019). Vision-based inspection system for leather surface defect detection and classification. *NICS 2018—proceedings of 2018 5th NAFOSTED conference on information and computer science* (pp. 300–304). https://doi.org/10.1109/NICS.2018.8606836

Cormen, T. H., Leiserson, C. E., Rivest, R., & Stein, C. (2013). Greedy-algorithmen. In *Algorithmen—Eine Einführung*. De Gruyter Oldenbourg. https://doi.org/10.1515/9783110522013-021

Danner, T., & Kavraki, L. E. (2000). Randomized planning for short inspection paths. In *Proceedings—IEEE international conference on robotics and automation* (Vol. 2, pp. 971–976). https://doi.org/10.1109/robot.2000.844726

Edelkamp, S., & Stommel, M. (2012). The Bitvector machine: A fast and robust machine learning algorithm for non-linear problems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 7523 LNAI(PART 1) (pp. 175–190). https://doi.org/10.1007/978-3-642-33460-3_17

Edelkamp, S., Secim, B. C., & Plaku, E. (2017). Surface inspection via hitting sets and multi-goal motion. *Planning, 10454*, 134.

Englot, B., & Hover, F. (2017). In Christensen, H. I., & Khatib, O. (Eds.), Planning complex inspection tasks using redundant roadmaps (pp. 327–343). Springer. https://doi.org/10.1007/978-3-319-29363-9_19

Fernandez, P., Rico, J. C., Alvarez, B. J., Valino, G., & Mateos, S. (2008). Laser scan planning based on visibility analysis and space partitioning techniques. *The International Journal of Advanced Manufacturing Technology, 39*, 699–715. https://doi.org/10.1007/s00170-007-1248-9

Glorieux, E., Franciosa, P., & Ceglarek, D. (2020). Coverage path planning with targetted viewpoint sampling for robotic free-form surface inspection. *Robotics and Computer-Integrated Manufacturing, 61*, 101843. https://doi.org/10.1016/j.rcim.2019.101843

González-Banos, H. (2001). A randomized art-gallery algorithm for sensor placement. In *Proceedings of the seventeenth annual symposium on computational geometry. SCG '01* (pp. 232–240). Association for Computing Machinery. https://doi.org/10.1145/378583.378674

Gronle, M., & Osten, W. (2016). View and sensor planning for multi-sensor surface inspection. *Surface Topography: Metrology and Properties, 4*(2), 024009. https://doi.org/10.1088/2051-672X/4/2/024009

Hoang, V. T., Phung, M. D., Dinh, T. H., & Ha, Q. P. (2020). System architecture for real-time surface inspection using multiple UAVS. *IEEE Systems Journal, 14*(2), 2925–2936. https://doi.org/10.1109/JSYST.2019.2922290

Huang, Y., Qiu, C., Wang, X., Wang, S., & Yuan, K. (2020). A compact convolutional neural network for surface defect inspection. *Sensors, 20*(7), 1–19. https://doi.org/10.3390/s20071974

Kabacoff, R. I. (2011). *R in action: Data analysis and graphics with R* (p. 309). Manning Publications. http://m.friendfeed-media.com/36d8ab666d485a984e441fd9d0f606c8c8553061

Karim, A. A., & Nasser, E. F. (2017). Improvement of corner detection algorithms (Harris, FAST and SUSAN) improvement of cor-

ner detection algorithms (Harris, FAST and SUSAN) based on reduction of features space and complexity time. *Engineering and Technology Journal, 35*(2), 112–118.
Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E., & Thatcher, J. W. (Eds.), *Complexity of computer computations plenum pre* (pp. 85–103).
Konrad, T., Lohmann, L., & Abell, D. (2019). Surface defect detection for automated inspection systems using convolutional neural networks. In *27th mediterranean conference on control and automation, MED 2019—proceedings* (pp. 75–80). https://doi.org/10.1109/MED.2019.8798497
Lee, K. H., & Park, H.-p. (2000). Automated inspection planning of free-form shape parts by laser scanning. *Robotics and Computer-Integrated Manufacturing, 16*(4), 201–210. https://doi.org/10.1016/S0736-5845(99)00060-5
Leopold, J., Günther, H., & Leopold, R. (2003). New developments in fast 3D-surface quality control. *Measurement: Journal of the International Measurement Confederation, 33*(2), 179–187. https://doi.org/10.1016/S0263-2241(02)00056-8
Pernkopf, F., & O'Leary, P. (2003). Image acquisition techniques for automatic visual inspection of metallic surfaces. *NDT and E International, 36*(8), 609–617. https://doi.org/10.1016/S0963-8695(03)00081-1
Phung, M. D., Quach, C. H., Dinh, T. H., & Ha, Q. (2017). Enhanced discrete particle swarm optimization path planning for UAV vision-based surface inspection. *Automation in Construction, 81*, 25–33. https://doi.org/10.1016/j.autcon.2017.04.013
Powell, D., Magnanini, M. C., Colledani, M., & Myklebust, O. (2022). Advancing zero defect manufacturing: A state-of-the-art perspective and future research directions. *Computers in Industry, 136*, 103596. https://doi.org/10.1016/j.compind.2021.103596
Psarommatis, F., MAY, G., Dreyfus, P.-A., & Kiritsis, D. (2019). Zero defect manufacturing: State-of-the-art review, shortcomings and future directions in research. *International Journal of Production Research, 58*, 1–17. https://doi.org/10.1080/00207543.2019.1605228
Psarommatis, F., Sousa, J., Mendonça, J. P., & Kiritsis, D. (2022). Zero-defect manufacturing the approach for higher manufacturing sustainability in the era of industry 4.0: A position paper. *International Journal of Production Research, 60*(1), 73–91. https://doi.org/10.1080/00207543.2021.1987551
Raffaeli, R., Mengoni, M., Germani, M., & Mandorli, F. (2013). Off-line view planning for the inspection of mechanical parts. *International Journal on Interactive Design and Manufacturing (IJIDeM), 7*, 1–12. https://doi.org/10.1007/s12008-012-0160-1
Skiena, S. S. (2008). Set and string problems. In *The algorithm design manual, Chap. 18* (pp. 622–623). Springer.
Son, S., Kim, S., & Lee, K. H. (2003). Path planning of multi-patched freeform surfaces for laser scanning. *The International Journal of Advanced Manufacturing Technology, 22*, 424–435. https://doi.org/10.1007/s00170-002-1502-0
Wu, Q., Lu, J., Zou, W., & Xu, D. (2015). Path planning for surface inspection on a robot-based scanning system. In *2015 IEEE international conference on mechatronics and automation (ICMA)* (pp. 2284–2289). https://doi.org/10.1109/ICMA.2015.7237842
Yi, B., Qiao, F., Huang, N., Wang, X., Wu, S., & Biermann, D. (2021). Adaptive sampling point planning for free-form surface inspection under multi-geometric constraints. *Precision Engineering, 72*, 95–101. https://doi.org/10.1016/j.precisioneng.2021.04.009
Zahmati, J., Amirabadi, H., & Mehrad, V. (2018). A hybrid measurement sampling method for accurate inspection of geometric errors on freeform surfaces. *Measurement, 122*, 155–167. https://doi.org/10.1016/j.measurement.2018.03.013
Zhou, A., Guo, J., & Shao, W. (2011). Automated inspection planning of freeform surfaces for manufacturing applications. In *2011 IEEE international conference on mechatronics and automation*. https://doi.org/10.1109/ICMA.2011.5986292
Zhou, F., Liu, G., Xu, F., & Deng, H. (2019). A generic automated surface defect detection based on a bilinear model. *Applied Sciences, 9*(15), 3159. https://doi.org/10.3390/app9153159
Zhou, Z., Zhang, Y., & Tang, K. (2016). Sweep scan path planning for efficient freeform surface inspection on five-axis CMM. *Computer-Aided Design, 77*, 1–17. https://doi.org/10.1016/j.cad.2016.03.003