Robotic Coverage Path Planning for Ultrasonic Inspection

Kastor Felsner *, Klaus Schlachter and Sebastian Zambal

PROFACTOR GmbH, Im Stadtgut D1, 4407 Steyr, Austria; klaus.schlachter@profactor.at (K.S.); sebastian.zambal@profactor.at (S.Z.)
* Correspondence: kastor.felsner@profactor.at

Abstract: Automatic robotic inspection of arbitrary free-form shapes is relevant for many quality control applications in different industries. We propose a method for planning the motion of an industrial robot to perform ultrasonic inspection of varying 3D shapes. Our method starts with the calculation of a set of sub-paths. These sub-paths are derived from streamlines. The underlying vector field is deduced from local curvature of the inspected geometry. Intermediate robot motions are planned to connect individual sub-paths to obtain a single complete inspection path. Coverage is calculated via ray tracing to simulate the propagation of ultrasound signals. This simulation enables the algorithm to proceed adaptively and to find a good trade-off between path length and coverage. We report experiments for four different geometries. The results indicate that shorter paths are achieved by using ray tracing for adaptive adjustment of streamline density. Our algorithm is tailored to ultrasonic inspection. However, the main concept of exploiting local surface curvature and streamlines for coverage path planning generalizes to other robotic inspection problems.

Keywords: robotic inspection; coverage path planning; ultrasonic inspection; streamline; principal curvature

1. Introduction

Coverage path planning (CPP) deals with the problem of finding a path for a robot to cover some region of interest. With the growing number of applications related to mobile robotics, CPP is a highly relevant problem to study. This is especially true as robotic applications cover more and more areas of everyday human life. A few applications to mention are vacuum cleaning robots, windmill inspection drones, and robotic quality control in industry.

Depending on the application, coverage may relate to visiting, inspecting, or treating (mowing, painting, etc.) the complete region. Hence, the definition of what region is covered by which robot configuration or position strongly depends on the application. For example, the region covered by a lawn mower robot at a specific position is a disk-shaped area around the center point of the rotating blade. For a robot performing optical inspection, the covered area is derived from the camera position, the field of view, and obstacles within the field of view. In any case, the definition of coverage is an important topic for all coverage planning algorithms.

We introduce an off-line method for the coverage path planning that is built upon a heuristic based on streamlines. Our main application is robotic inspection via ultrasonic testing (UT), for which a highly automated workflow was described in [1]. In order to calculate coverage, we rely on a ray tracing approach, using a 3D model of the specimen to be inspected. Via ray tracing, it is possible to generate a volumetric representation of covered regions inside the inspected part. While the ray tracing approach is very specific to UT, the heuristic of using streamlines is applicable to other coverage planning problems.

This article is structured in the following way. In Section 2, we review related work on coverage path planning and the coverage calculation. In Section 3, we provide an overview about streamlines in general. To derive streamlines for our CPP algorithm,
we build upon a vector field defined by local surface curvature. Surface curvature is discussed in Section 4. Another building block for our method is ray tracing to simulate the propagation of ultrasound waves—outlined in Section 5. Section 6 explains how we put everything together for the proposed coverage path planning algorithm. An evaluation of the algorithm is provided in Section 7, and conclusions are drawn in Section 8.

2. Related Work

Most existing work on coverage path planning relates to problems where a robot moves in 2D space. This is interesting for applications such as lawn mowers, vacuum robots, and harvesting machines. In the literature, recent surveys on coverage path planning have limited focus on 3D coverage [2,3]. Many classical approaches, such as cellular decomposition [4], and grid-based [5] and graph-based [6] methods, are difficult to extend to 3D cases. In general, the computational complexity tends to increase massively with the transition from 2D to 3D.

Many coverage planning algorithms address the problem of a mobile robot that is required to visit all locations in the region of interest. The region of interest is discretized so that it is represented by a finite number of points or grid elements [7,8]. Depending on the application, additionally, the orientation of inspection positions needs to be considered. This enlarges the space of possible solutions, leading to increased computational complexity. For on-machine inspection with a 5-axis system, a method was proposed that considers multiple orientations per inspected point [9]. Subsequently, the points with different orientations are connected by solving a traveling salesman problem.

Robotic inspection using ultrasound is relevant for multiple quality control applications in industry. An example for robotic UT is the inspection of components of a thermonuclear reactor [10]. Even in the medical field, ultrasonic scanning by a robot is an interesting topic. A recent method [11] is based on force-feedback during the scanning process.

In the context of robotic scanning, recent work involves in-line dimensional inspection via spatio-temporal adaptive sampling (STAS) [12]. This method estimates deviations of the whole part based on partial measurement of a free form surface. In addition, the method adaptively selects the next region to be measured in each step. An interesting extension of coverage path planning deals with multi-robot environments [13,14]. For most applications, this requires some kind of synchronization of robots. Although there could be some potential of using our method in a multi-robot environment, we do not further address this scenario.

As computational complexity is often high in the context of 3D surface inspection, some papers propose heuristics that iteratively grow the inspected region. One method uses a geometric random tree to sample possible future configurations [15]. This was shown to provide good scaling properties with respect to the complexity of scenarios. In general, the concept of iteratively growing the covered area with influence from control theory is referred to as receding horizon. A recent work [16] deployed this concept for online path planning to adaptively acquire information from the surface. Information already acquired is inserted into a manifold Gaussian process and influences the progressive calculation of the path.

A problem that is related to coverage path planning is classical point-to-point robot path planning. Robot path planning addresses the problem of finding a valid sequence of configurations or states by which a robot moves from some start position to a target position. Depending on the space of possible states, this can have different levels of complexity. There are well-known algorithms available, e.g., rapidly exploring random trees (RRT) [17]), that solve the point-to-point path planning problem. In the following, we deploy standard solutions for point-to-point robot motion planning. This is mainly relevant for situations where sub-paths that cover separate regions need to be connected. The main intention of such a connecting motion is not to increase coverage but to perform a transition from one region or robot configuration to another.
An adequate definition of coverage is an important topic for CPP algorithms. Definitions of coverage can be very diverse, ranging from point-wise coverage [13] to drones that spray disinfectants [18]. In the present work, we focus on volumetric ultrasonic inspection. The respective coverage relates to the propagation of ultrasound waves through the investigated materials. We deploy ray tracing to approximate the process of ultrasonic inspection. The goal is to predict the volumetric coverage for a single sensor position.

In order to model inspection via UT, it is necessary to have a closer look at the underlying physical process. For ultrasonic inspection, phased arrays are commonly used for signal generation and data acquisition. The total focusing method (TFM)—introduced in [19]—is therefore one of the most popular methods. The term refers to a post-processing of the full matrix capture, which accounts for all received signals of all receiver–sender pairs of the phased array. In order to calculate volumetric coverage for a single sensor position, we need to approximate how ultrasonic waves propagate from a specific sender to a specific receiver interacting with the specimen.

Generally, strategies for numerically evaluating sound propagation can be grouped into wave-based and geometric-based approaches: The underlying wave equation can be solved by classical methods, e.g., the finite element method and boundary element method [20,21]. This is especially useful to evaluate low frequency effects, such as diffraction. A geometric acoustic approach, on the other hand, is suitable for high-frequency waves, i.e., if the examined wave lengths are short in relation to the objects that the wave is interacting with. In order to additionally account for low frequency effects, geometric- and wave-based approaches can be combined [22].

For our purpose, we use a geometric approach, which already is a widely studied field, motivated by a variety of use cases, e.g., auralization for audio post processing and virtual environments, or simulation of classrooms helping the design process [20,23,24]. Recently, realistic ultrasound rendering was studied to enhance medical training [25–28].

3. Streamlines

Physical flow, for example, fluid flow in a liquid, is described by a vector field. If the vector field does not change over time, we call it a static vector field. The trajectory of a very light particle (with zero-mass) through the flow follows a streamline. Given a start point and a vector field, it is possible to calculate the respective streamline. The position of the particle evolves through the vector field such that the velocity of the particle in each point corresponds to the value of the vector field at that position. For a vector field \( \mathbf{v} \), a streamline \( s_t = s_t(t) \) is defined as follows:

\[
\frac{ds_t}{dt} \times \mathbf{v}(s_t) = 0
\]  

(1)

where \( t \) is the parameter of the streamline. In general, at any point along the streamline \( s_t \), the vector \( \mathbf{v}(s_t) \) is parallel or anti-parallel to the derivative \( \frac{ds}{dt} \). Therefore, both vectors differ at most in their absolute size:

\[
\frac{ds_x}{dv_x} = \frac{ds_y}{dv_y} = \frac{ds_z}{dv_z}
\]  

(2)

where \( s_t = (s_x, s_y, s_z)^T \) and \( \mathbf{v} = (v_x, v_y, v_z)^T \). The streamline is not defined where the vector field is zero. Figure 1 shows an example for a streamline in 2D. The first step for calculation of a streamline is to select a seed point. At this point, the respective vector from the vector field is investigated. Via numerical integration, the streamline is expanded point by point, following the parallel and anti-parallel direction of the vector field in each point. In the following sections, we outline how streamlines are used as a basis for the trajectory of a sensor and the corresponding robot path.
4. Using Curvature as Vector Field

Any twice differentiable parametric surface $S$ in $\mathbb{R}^3$ has two principal curvatures, $\kappa_1$ and $\kappa_2$, at each surface point. These principal curvatures correspond to two perpendicular vector $t_1$ and $t_2$ in the tangent plane. Given a parametric surface description, the respective tangent vectors of principal curvatures can be derived analytically. In the case of discretized surface representations, e.g., triangle meshes, principal curvature tangent vectors need to be approximated numerically.

Different methods were proposed to calculate curvature for arbitrary surfaces. For example, one approach builds upon principal component analysis (PCA) of points in the local neighborhood [29]. We follow a similar approach. However, instead of an analysis of points, we estimate curvature based on surface normals. A singular value decomposition (SVD) of surface normals delivers curvatures $\kappa_i$ and tangent vectors $t_i$.

The goal of our CPP algorithm is to keep a suitable alignment of a line-shaped linear phased array ultrasound sensor relative to the surface of the inspected part, see Figure 2. We assume that suitable poses along the path can be derived from local surface properties, i.e., orientations of the principal curvatures. For ultrasonic inspection with a linear phased array, it is important that a large fraction of the complete energy of the emitted signal is also reflected back to the transducer head. Ideally, the surface normals of all points at which the reflection occurs point toward the phased array. Small variations (e.g., due to calibration errors or positioning inaccuracies) within the plane $\epsilon$ spanned by the surface normal $n$ and $t_1$ lead to only moderate loss of the reflected signal intensity. If the variation is small enough, there is still a chance that the reflection hits an element from the phased array. Surface normals that are tilted out of this plane result in high loss of signal intensity. The reason for this is that reflections that exit the plane are not captured by any of the phased array transducer elements.
5. Ray-Tracing Based Process Model

We propose to estimate the coverage of a linear phased array ultrasound sensor via ray tracing. This provides a good trade-off between physical accuracy and computational complexity. The main goal is to predict at which locations defects are detectable.

Common defects in carbon fiber reinforced plastic (CFRP) parts are delaminations and porosities \[30\]. These fault types cause two different effects on the captured ultrasound data, which are utilized for fault detection: (a) delaminations cause reflections of the sound, and (b) porosities scatter the signal and, therefore, cause an attenuation of the back wall echo. We use a ray-tracing approach to judge whether both fault types can be detected in a volume element from a given sensor pose (position and orientation). For each sensor element of the phased array, we simulate the propagation of the wave vector. If a ray passes a specific volume element (of the discretized specimen), we assume that delaminations can be detected. If the same ray is reflected at the back wall of the specimen and afterwards detected by the sensor, we assume that an existing porosity in that volume element can be detected by detecting a back wall attenuation. Volume elements that are passed by rays that meet these criteria are marked as captured; see Figure 3.

The ultrasound sensor used throughout this paper is a linear phased array (PA) that consists of 64 transducers and has a width of 114 mm. This is a simplification of a real sensor with 114 transducers at 114 mm width. The discrepancy allows faster ray tracing computations. The sensor is modeled as a rectangle where the single transducers are spaced equidistantly along the longest edge of the sensor. Rays are emitted at the center of each transducer in a range of different angles with a pre-configured maximal aperture. The number of rays per transducer is fixed. Ray tracing is performed with a recursion depth of four. All rays that are reflected by the geometry in such a way that they hit a phased array element are marked as captured by the sensor. As illustrated in Figure 4, the modeled PA sensor is placed at a given working distance relative to a streamline that is calculated for the mesh surface. Emitted rays and reflections/transmissions are shown. Only rays emitted near the center of the PA sensor are cast back to the sensor, due to the challenging position of the sensor.
Figure 3. Illustration of coverage calculation via ray tracing. Two rays that are emitted by the sensor are shown. At any surface, transmission and reflection occur. Hence, each ray spans a tree. Voxels that are traversed by rays are shown in color. Red voxels correspond to a tree of rays that do not return to the sensor—at least not within the first four levels of the tree. Blue voxels are covered by rays that return back to the sensor. In contrast to red voxels, we consider blue voxels to be covered.

Figure 4. Visualization of reflected rays on the geometry of a complex beam. The PA sensor is positioned at a working distance to a streamline that follows the edge of the part. To reduce clutter in the visualization, the depicted phased array sensor consists of only 21 transducers. Only 5 co-planar rays are emitted by each transducer. The rays highlighted in orange are emitted by the same transducer near the center of the ultrasound sensor. Some rays are reflected at the front wall and some at the back wall of the specimen.

6. Coverage Path Planning

A conventional approach for coverage path planning is to move the sensor over the surface along a set of evenly spaced parallel lines. Those lines are often traversed in such a way that a meander-shaped path is formed. However, by more accurate modeling of the inspection process, it is possible to generate shorter inspection paths as the distance between scan lines can be chosen according to the local specimen’s geometry (e.g., flat regions allow larger gaps between scan lines because the sensor is able to capture a larger region at once). To achieve a more reasonable placement of scan lines, we formulate a process model which addresses both the problem of (a) mapping surface points to sensor orientations and positions and (b) determining the inspected volume of the specimen for a certain sensor pose.
When determining the sensor pose for a given point on the surface, the following constraints need to be considered:

1. To maximize the signal strength, the sensor has to be placed such that the main direction of emission (the sensor’s z-axis) is parallel to the surface normal, and the center of the sensor is positioned along that normal.
2. The inspection distance should be equal to the optimal working distance of the sensor.
3. To maximize coverage, the long edge of the sensor (the sensors y-axis) has to be perpendicular to the scanning direction. That is, the sensor needs to be positioned as show in Figure 4, where the phased array travels along the streamline, which is depicted as a black line on the mesh surface.
4. For every point of a curved surface, the sensor should be orientated such that the direction of the largest curvature is parallel to the sensor’s y-axis. This tends to maximize the strength of the captured signals.

Note that the PA sensor is invariant under a 180° rotation around its z-axis. Thus, for every point on a curved surface, we can consider the sensor pose as fully defined by the aforementioned constraints.

Our coverage path planning approach can be divided into two separate phases: (a) a Cartesian coverage path planning phase (see Algorithm 1) that uses the described process model, and (b) a greedy search to derive a robot path that scans each point on the Cartesian path exactly once while considering the limitations of the robot kinematics. The Cartesian coverage path planning algorithm uses the specimen’s CAD data to plan a path on the outer surface of the specimen. To keep track of the inspected volume, the algorithm simulates the ultrasonic data acquisition. The volumetric information is then projected back onto the surface. This allows to generate a coverage path that includes information about the actual covered volume.

Algorithm 1 Cartesian coverage path planning algorithm

```plaintext
1: procedure CCPP(MESH)
2:    voxelgrid ← voxelize mesh ▷ mark all voxels as not inspected
3:    surface ← discretize outer surface
4:    seeds ← calculate curvature vector field for surface
5:    sort seeds using the curvature value in descending order
6:    for s in seeds do
7:      if s is too close to an existing streamline then
8:        skip s
9:      if curvature of s > threshold then
10:         streamline ← expand streamline from s
11:     else  ▷ Note: specimen is now subdivided into flat areas
12:        A ← connected flat area containing s
13:        c ← center of A, re-projected onto the mesh
14:        v ← vector pointing along the principal component of A
15:        streamline ← expand streamline from c with initial direction v
16:    for p in streamline do
17:        inspected voxels ← run ray-tracing algorithm for pose p
18:        update voxelgrid using inspected voxel
19:    for s in seeds do  ▷ Only the seed points underneath the sensor are considered
20:       n ← #voxels along the surface-normal marked as not inspected
21:       if n = 0 then
22:          remove s
```

The complete algorithm proceeds as follows. Initially, the input mesh is voxelized, and the outer surface is discretized. For the discretized surface, the vector field based on curvature is calculated, and all surface points are initially marked as seed points. Afterwards, the algorithm iterates over all seed points in descending order of their curvature.
value. For each seed point, a streamline that follows the second principal curvature is expanded. Once the streamline expansion is terminated, the volumetric inspection is updated by running the ray tracing algorithm for every point on the streamline. Seed points are skipped if all voxels along the surface-normal (inward) are already inspected or if another streamline is in close proximity.

If a seed point’s curvature value is below a given threshold, the corresponding area is considered flat. Note that once a seed point meets this criterion, the not inspected area of the mesh is composed of unconnected flat areas. On flat areas, streamlines are expanded in a straight line from the areas center along its principal component.

Once the Cartesian paths are created, the second phase of the algorithm is executed. We deploy a greedy search algorithm that generates the corresponding robot path. In a first step, the algorithm calculates multiple valid robot paths per streamline. In a second step, the separate streamline-paths are connected, yielding the final robot path that covers all points on the Cartesian paths exactly once.

For a 6-axis robot, each Cartesian pose may have multiple inverse kinematics (IK) solutions, i.e., multiple joint configurations yield the same sensor pose. To generate valid robot paths for a given streamline, first, all possible joint configurations are calculated for each point on the streamline. Since the sensor is invariant under a 180° rotation around its z-axis, also the IK solutions for the rotated pose are calculated. In a next step, connectable IK solutions for adjacent points on the streamline are connected, forming robot path segments that the robot can traverse without considerable re-configuration. More precisely, two points $a$ and $b$ (in joint space) are connectable if the Cartesian midpoint $m = (FK(b) - FK(a)) / 2$ and the point obtained from the forward kinematics (FK) of the midpoint in $m' = FK((b - a) / 2)$ are sufficiently close in joint space. The path segments are constructed in such a way that multiple streamline paths are generated. A streamline path consists of multiple path segments, which together, cover all points of the streamline exactly once. Note that a path segment may be part of multiple streamline paths.

In the final stage of the algorithm, the generated path segments are connected. Starting from a randomly selected path segment, the final complete robot path is created. In each iteration, the path is expanded at one of its endpoints by the path segment which is closest in joint-space. Once a path segment is used, all other path segments covering the same streamline that do not share a streamline path are deleted. This guarantees that all points of the Cartesian paths are inspected exactly once.

7. Results

To determine the effectiveness of the proposed method, we evaluate it on different geometries. Our evaluation focuses on the Cartesian path planning phase, or more precisely, on the method used to mark a part of the surface as inspected. The algorithm proposed in Section 6 (later called papp-adaptive) iteratively generates streamlines and deploys a ray-tracing algorithm after each iteration. The obtained volumetric inspection is then projected back onto the surface, marking surface points that are above a column of fully inspected voxels as inspected. This approach is compared to a baseline algorithm (papp-base-d) that does not consider the volumetric inspection, but simply considers surface points in a certain proximity $d$ to the streamline as inspected. Four different values for $d$ are tested: 30 mm, 40 mm, 50 mm and 60 mm. The phased array sensor has a width of 114 mm, and thus, an inspection radius of 60 mm still yields good results on slightly curved surfaces.

The algorithm variants are compared using the Cartesian path length, i.e., the total length of all streamlines. The quality of the path is measured as the percentage of the (voxelized) mesh-volume that is covered by the given path. For the baseline algorithm, the ray tracing algorithm is used to derive the volumetric inspection once all streamlines are calculated.

The evaluation is conducted, using four mesh objects shown in Figure 5. These objects are the following: WAVE, a moderately wavy plate with increasing wall thickness; EDGE1,
an edge which is bent at its center with thinner walls towards the ends; and EDGE2, an edge that bends twice in two different directions and BEAM: a complex U-shaped beam. The surface curvature has a big impact on the size of the inspected volume. A large fraction of energy is lost when the surface normal is not parallel to the sensors’ z-axis. Challenging areas are also those with thick walls and/or non-parallel back walls. The wall thickness for all objects is in a range between 15 mm to 55 mm. All mesh objects are available online (https://zenodo.org/record/5173521#.YRIIJECxVPY). Streamlines calculated in the first phase of the coverage planning algorithm papp-adaptive are illustrated in Figure 5.

Figure 5. Test objects used to evaluate the proposed algorithm. The streamlines (in red) are generated with the following variants of the algorithm (from top-left to bottom-right): papp-base-60, papp-adaptive, papp-base-40, papp-adaptive.

Table 1 shows the path length and achieved coverage for the different algorithms. For flat areas, such as the object depicted in Figure 5a, the baseline algorithms achieve full coverage with short paths. However, for more challenging areas, e.g., for the highly curved center area on the mesh EDGE1, the baseline algorithms require an even smaller inspection radius to achieve full coverage. Figure 6 visualizes the voxels that are missed by algorithm papp-base-60.

Table 1 summarizes path length and coverage for different versions of our algorithm on the four different test geometries. The papp-adaptive version of the algorithm performs quite well in the sense that it delivers full coverage for all test geometries. For WAVE, there is little advantage of using the adaptive method as the respective geometry contains only moderate curvature. For EDGE1 and EDGE2, the adaptive method outperforms all other variants of the algorithm. For these geometries, papp-adaptive delivers full coverage.
with a shorter path length. For BEAM, the two algorithm variants papp-adaptive and papp-base-30 both achieve full coverage. However, in this case, the papp-adaptive version requires a slightly larger path length than papp-base-30.

For this evaluation, we only considered parts for which the complete volume can be inspected. For regions where ultrasonic inspection fails, e.g., due to the objects thickness or due to the angle of the backwall, the described algorithm places streamlines with a minimal distance to each other, yielding a unnecessary long path. For such objects, the user may manually select the area that should be inspected.

**Table 1.** Comparison of different Cartesian path planning algorithms for four different mesh objects. The path length is the combined Cartesian path length in millimeters. The coverage indicates the percentage of voxels that are inspected using the generated path. The last column states the total number of missed voxels.

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Algorithm</th>
<th>Path Length (mm)</th>
<th>Coverage (%)</th>
<th>Missed Voxels</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAVE</td>
<td>papp-adaptive</td>
<td>2048.0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-30</td>
<td>3985.0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-40</td>
<td>2009.0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-50</td>
<td>2005.0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-60</td>
<td>2006.0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>EDGE1</td>
<td>papp-adaptive</td>
<td>3985.9</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-30</td>
<td>4816.8</td>
<td>99.99</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>papp-base-40</td>
<td>3226.9</td>
<td>99.98</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>papp-base-50</td>
<td>3142.9</td>
<td>99.76</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td>papp-base-60</td>
<td>2441.9</td>
<td>99.52</td>
<td>163</td>
</tr>
<tr>
<td>EDGE2</td>
<td>papp-adaptive</td>
<td>6906.4</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-30</td>
<td>8835.8</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-40</td>
<td>7147.8</td>
<td>99.96</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>papp-base-50</td>
<td>5628.5</td>
<td>99.96</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>papp-base-60</td>
<td>4723.8</td>
<td>99.92</td>
<td>71</td>
</tr>
<tr>
<td>BEAM</td>
<td>papp-adaptive</td>
<td>1753.8</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-30</td>
<td>1558.0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>papp-base-40</td>
<td>1133.0</td>
<td>98.13</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>papp-base-50</td>
<td>739.9</td>
<td>80.01</td>
<td>1329</td>
</tr>
</tbody>
</table>

8. Conclusions and Future Work

Coverage path planning for the inspection of complex 3D geometries is a challenging task. The exact solution of this problem is computationally not feasible. In this paper, we presented an approach that builds upon a heuristic using streamlines. Local curvature is taken as the measure to evolve individual streamlines. An important parameter of our algorithm is the distance at which streamlines are placed. This can either be a fixed value or can be adjusted adaptively. In the latter case, coverage is approximated via ray tracing in order to estimate the actual coverage for a path. While ray tracing is a relatively simple method to simulate ultrasound propagation, it provides a good trade-off between physical correctness and computational effort. An evaluation on four selected geometries shows that adaptive distances of streamlines can bring an improvement in the range of 17 to 22 percent shorter path length with the same or even better coverage on non-flat geometries. While the method does not provide an optimal solution to the computationally intractable coverage path planning problem, our method is well suited to many practical
robotic ultrasonic scanning problems. Moreover, the method of using streamlines lends itself well for adaptation of similar inspection tasks.

Clearly, our method is not able to overcome physical limitations of ultrasonic inspection systems. Limitations relate to possibilities in the design of ultrasonic transducers (limited size of transducers, wavelengths/resolutions, etc.). For very complex configurations, full coverage might not be possible, due to the physical limitations. In some situations, a different layout of the transducer array might help, for example, using a curved instead of a linear layout. However, we do not take this degree of freedom into account in our present work. In any case, with the calculation of coverage, our method provides valuable information about the possibility to inspect individual regions.

Future work will cover topics related to improvements of the ray tracing algorithm for better coverage calculation. Specifically, we plan to compare the calculated coverage with real ultrasound scans.

Author Contributions: Conceptualization and methodology, K.F., K.S. and S.Z.; software, validation and formal analysis, K.F. and K.S.; investigation, K.F.; writing—review and editing, K.F., K.S. and S.Z.; visualization, K.F. and S.Z.; supervision, project administration and funding acquisition, S.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This project has received funding from the Clean Sky 2 Joint Undertaking (JU) under grant agreement No. 831830. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and the Clean Sky 2 JU members other than the Union. Work presented in this paper has also received funding by the European Union in cooperation with the State of Upper Austria within the project “Investition in Wachstum und Beschäftigung” (IBW).

Data Availability Statement: The generated mesh objects (EDGE1, EDGE2 and WAVE) that have been used to evaluate the proposed CPP algorithm are available at https://zenodo.org/record/5173521#.YRIIJECxVPY.

Conflicts of Interest: The authors declare no conflict of interest.

References


20. Lakka, E.; Malamos, A.G.; Pavlakis, K.G.; Ware, J.A. Spatial Sound Rendering—A Survey. *IJIMA* 2018, 5, 33–45. [CrossRef]


