Structural Inspection Planning for Mobile Robots

Raphael Hagmanns

Vision and Fusion Laboratory
Institute for Anthropomatics
Karlsruhe Institute of Technology (KIT), Germany
raphael.hagmanns@kit.edu

Abstract

In this report we present a pipeline for static coverage planning of known objects, which is an important task in the field of mobile robot based inspection. We analyse the main components of the Structural Inspection Planner [1] and embed an improved implementation into a autonomous flight pipeline for UAVs. Triangle mesh models serve as input for an initial viewpoint sampling. Inspection quality and path length are optimized by formulating the viewpoint sampling as constraint QP. We thoroughly evaluate the ROS-based inspection pipeline on synthetic and real models using a Gazebo simulation. Our experimental evaluation shows that while an efficient inspection trajectory could be generated for most of the tested models, the result is very dependent on regular and well formed input models.

1 Introduction

Automated structural inspection tasks have become increasingly important in the last couple of years. As facilities grow larger, it is difficult to guarantee a continuous and smooth operation without generating a high manual workload. The field of automating those operations is called *Non-Destructive Inspection* (NDI) [9]. Utilizing mobile robots to perform NDIs can keep workers out of

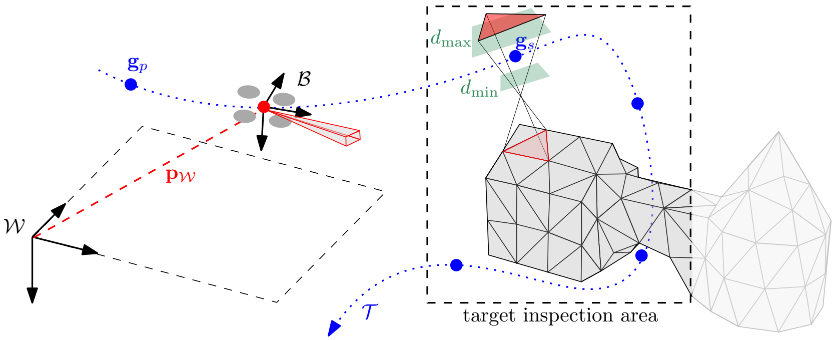


Figure 1.1: Outline of a UAV inspection flight. The drone with a fixed camera is following the inspection trajectory (blue), while tracking its exact position in the world coordinate frame $\mathcal{W}$. Blue points on the trajectory mark viewpoints for specific triangles. The position of the viewpoint g_s is dependent on a set of constraints, for instance the one constructed by the parameters d_{min} and d_{max} given in green.

dangerous situations and save a lot of time and resources. Especially with the availability of UAVs (Unmanned Aerial Vehicles), high risk operations such as inspection of buildings, bridges or oversea structures become feasible. Nooralishahi et al. recently provided an in-depth review on the current state of UAVs in NDI [9]. Oftentimes, it is desirable to create an inspection on the basis of an existing model of the respective structure. This allows to quantify the structural damage on the surface compared to the existing model. However, such an approach requires the mobile inspection robot to approach specific viewpoints with a high accuracy in order to inspect the correct target area. This is very difficult to achieve by a manual UAV operator and leads to artifacts in the imagery generated during the inspection flight due to the accumulation of positioning errors. These issues can be dealt with by automatically generating suitable viewpoints and a corresponding inspection trajectory. This way, consistent inspection imagery can be generated in an automated fashion through a number of repetitions.

Therefore, we propose a framework which allows for inspection planning of structures using different kinds of mobile robots, even though we focus on UAVs here. A schematic visualization of the inspection process is given in Figure 1.1. More detailed building blocks of the proposed framework are depicted in Figure 3.1. The inspection planning approach conceptually bases heavily on the Structural Inspection Planner (SIP) [1], which has been developed to calculate inspection trajectories for fixed wing drones as well as UAVs for existing triangle meshes. Our work identifies weaknesses of the SIP and re-implements it with certain modernisations and adoptions. We also abstract the planner component in a way that it can be used inside a larger simulation framework. This allows us to easily perform tests and simulation flights for reconstruction purposes using the framework. Section 2 gives a more detailed overview on the structural inspection planner and other related work, before we present the main structure of our inspection pipeline in Section 3. We apply the planner to different artificial and real models in a Gazebo simulation. This allows us to easily test different UAV, scenario and sensor setups and also account for measurement uncertainties. The qualitative and quantitative evaluation of these experiments is presented in Section 4. Finally, we conclude our work, identify drawbacks and comprehensively describe possible future adoptions in Section 4.

The proposed framework is built within the Robot Operating System (ROS) [11] connecting the different components as visualized in Figure 3.1. It is widespread in the robotic community as it comes with sensor drivers, state of the art simulation frameworks (Gazebo [7]) and lots of prebuilt algorithms for perception and navigation. [1,2] This allows us to abstract the inspection planner into a single node as modular component in a greater UAV stack developed in a previous work [5]. This stack is supported by a Gazebo simulation of a UAV platform running the px4 software stack.[3] Px4 [8] is an open source autopilot running on various drones. It comes with Software-in-the-loop *SiL* and Hardware-in-the-loop *HiL* features which allows ours scenarios to be simulated in a realistic way.

[1] `https://ros.org/`
[2] `https://gazebosim.org/`
[3] `https://px4.io/`

2 Related Work

UAVs are a natural choice for structural inspection as they allow for agile movements in complicated and cluttered environments. This led to extensive research for flight planners in the last couple of years. Oftentimes, the desired goal is to create a reconstruction of a previously unknown environment. In this work, we focus on *model-based inspection*, where we require an accurate mesh of the target. Previous works with these conditions are rare.

The work by Yan et al. uses a multistage approach to generate a coarse reconstruction in a first step and then samples viewpoints for a high quality reconstruction in a second step [15]. Such an approach targets large scale reconstruction as prior knowledge of the target shape is not utilized. Instead, the skeleton is being build with a costly Structure-from-Motion technique. Schmid et al. provide an online informative path planner, where only one inspection flight is required. It uses an RRT*-inspired exploration scheme with object coverage as optimization target. They showed a TSDF-based reconstruction of previously unknown target areas [12, 4].

The Structural Inspection Planner (SIP) [1] is one of the few frameworks which explicitly uses triangle meshes as input to sample a viewpoint trajectory. It samples viewpoints for each triangle in the mesh. Viewpoints have geometrically derived constraints, which are solved as global optimization problem. In a next step, all sampled viewpoints are connected in an efficient way by interpreting the trajectory generation as Traveling Salesman Problem. The steps of viewpoints sampling and trajectory generation via TSP are combined in an iterative fashion until a minimal-length path is found. In practical application however, we found the planner to sample not admissible viewpoints or not converging at all for difficult meshes. The initial viewpoint sampling is highly dependant on the structure of the triangle mesh. Jing et al. [6] also uses an explicit model as input representation. However, they require a voxelized version of the model to first sample a suitable inspection area (via-points) using voxel dilation of the target. Suitable path primitives are then randomly sampled and verified by estimating the target visibility at each point. In a final step, a graph based method is used to generate the final UAV trajectory from the path primitives.

For larger objects, the operation on single voxels can become costly, so that the visibility calculation is not feasible for all admissible via points.

A recent work by Debus and Rodehorst [2] on the inspection of buildings provides evaluation metrics for path planning approaches. The corresponding *Bauhaus Path Planning Challenge* comes with a framework implementing these metrics for a set of models, which will also be targeted in this report. Despite typical evaluation metrics such as *path length* and *runtime*, they also focus on measurable reconstruction quality and surface resolution.

3 Planning Pipeline

In the following, we briefly describe our pipeline architecture before we discuss the inspection planner design in greater detail. The main building blocks of the pipeline are depicted in Figure 3.1. We embed the planner into our UAV framework presented in a previous work [5]. We design the main building blocks “Viewpoint Sampling” and “Trajectory Generation” to be components of a Planner Manager, as this manager also takes care of the sequential control for replanning and avoiding obstacles during the mission. In the original work [1], both blocks of viewpoint sampling and trajectory generation were supposed to run multiple times in an alternating scheme. However, in the experimental evaluation we show that the viewpoint arrangement resulting from the initial sampling iteration is already an intuitive result providing full coverage. Therefore, we mostly apply only one step of sampling and trajectory planning routines. This reduces the overall planning time at the cost of longer trajectories.

3.1 Viewpoint Sampling

We use the same optimization scheme as Bircher et al. in [1], as we iterate through all triangles in the mesh to generate one viewpoint each. A viewpoint needs to fulfill all *intrinsic* and *extrinsic* constraints. The first refers to the visibility of the targeted triangle, while the latter refers to boundary constraints given by the user. Opposing to [1], we increase the flexibility of the optimization problem by allowing an arbitrary number of constraints in the solver. This also

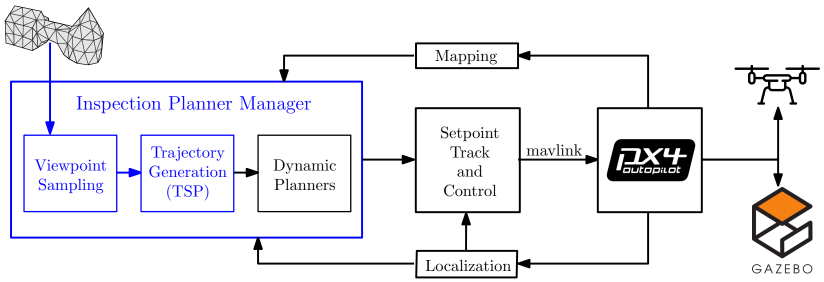

Mapping
Inspection Planner Manager
Viewpoint Sampling
Trajectory Generation (TSP)
Dynamic Planners
Setpoint Track and Control
mavlink
PX4 autopilot
GAZEBO
Localization

Figure 3.1: Structure of the whole simulation pipeline. The relevant blocks for inspection planning are highlighted in blue. The system consists of individual components for localization, mapping and control, which enable to fly the inspection trajectory within a simulated or real environment. The pipeline works with system based on the px4 [8] firmware, which implements the mavlink protocol for communication.

allows to build more generic constraints for polygons instead of triangles. We first calculate the normal a_N of each of the polygons with vertices $\{x_1, \ldots, x_n\}$, as well as all edge normals $n_1, \ldots, n_n$. In addition, we need to specify the camera parameters horizontal and vertical field of view as well as the pitch. Using these values as input, we build the following set of constraints.

$$\begin{aligned}
\min_{g^k} \quad & (g^k - g_p^{k-1})^T (g^k - g_p^{k-1}) + (g^k - g_s^{k-1})^T (g^k - g_s^{k-1}) \\
& + \alpha_w (g^k - g^{k-1})^T (g^k - g^{k-1}) \\
\text{s.t.} \quad & \underbrace{\begin{bmatrix} \infty \\ \infty \\ \vdots \\ \infty \\ d_{\max} \\ \infty \\ \infty \\ \infty \\ \infty \end{bmatrix}}_{\text{ubA}} \geq \underbrace{\begin{bmatrix} n_1^T \\ n_2^T \\ \vdots \\ n_n^T \\ a_N{}^T \\ n_{\text{right}}^T \\ n_{\text{left}}^T \\ n_{\text{lower}}^T \\ n_{\text{upper}}^T \end{bmatrix}}_{\text{A}} \cdot g^k \geq \underbrace{\begin{bmatrix} n_1^T \cdot x_1 \\ n_2^T \cdot x_2 \\ \vdots \\ n_n^T \cdot x_n \\ d_{\min} + a_N{}^T \cdot m \\ n_{\text{right}}^T \cdot m \\ n_{\text{left}}^T \cdot m \\ n_{\text{lower}}^T \cdot x_{\text{lower}}^{\text{cam}} \\ n_{\text{upper}}^T \cdot x_{\text{upper}}^{\text{cam}} \end{bmatrix}}_{\text{lbA}}
\end{aligned} \tag{3.1}$$

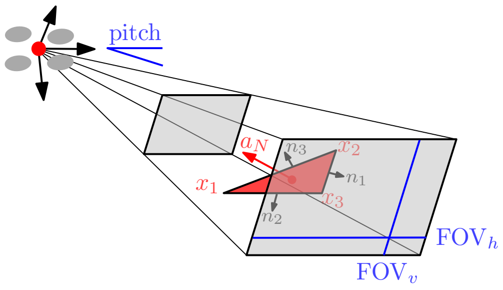


Figure 3.2: Visualization of the camera parameters FOV and pitch as well as the notation for triangle vertices and normals.

The left and right parts of the above equation ubA $= \{x_{min}, y_{min}, z_{min}\}$ and lbA $= \{x_{max}, y_{max}, z_{max}\}$ quantify the admissible viewpoint sampling space. The first n constraints force the viewpoint g^k to be sampled "in front" of the triangle, as the projection of the hyperplane normal $n_i^\top$ of the respective edge onto the vector spanned by the viewpoint $(g^k - x_i)^\top$ is required to be positive. The next constraint forces the distance of the viewpoint to be in $[d_{\text{min}}, d_{\text{max}}]$ by restraining projected distance of the triangle normal a_N. Finally, the last four constraints are exactly the *field of view* constraints from [1]. They guarantee the viewpoint to lie inside the horizontal and vertical FOV of a camera with a specific pitch. To accomplish this, four hyperplane normals $n_{\text{upper}}, n_{\text{lower}}, n_{\text{left}}, n_{\text{right}}$ with respective anchor points $x_{<\cdot>}$ are sampled using the camera pitch and FOV. A more detailed derivation of these constrains can be found in the original work of SIP [1].

The optimization objective is to minimize the distance between two consecutive viewpoints g_p and g_s, which optimizes the total path length. In the original formulation from [1], the viewpoint sampling was meant to run multiple iterations, such that the squared distance between g_p^k, g^k and g_s^k and their previous iteration $k - 1$ is minimized. The quadratic problem is then solved using qpOASES [3], resulting in a optimal position g_{opt} for the current iteration k. In a next step, the rotation is sampled by performing an explicit visibility analysis. In a simple UAV scenario, pitch and roll of the rotorcraft are always fixed, while the yaw angle γ is subject to change. For the sampled position and all possible yaw angles in $[0, 2\pi]$ with a step size of 0.2rad we check if (a) all distance constraints are met, (b) all vertices lie withing the FOV of the camera, and (c) there are no collisions on the way from camera to the polygon. The collision check can be invoked by performing a simple raycast which allows it to consider other static obstacles in the scene. The first position and orientation pair which passes all requirements, is selected as viewpoint.

3.2 Trajectory Planning

Given the set of N viewpoints $\{g_1, g_2, \ldots, g_N\}$, one for each triangle, we now connect them into a single shortest path trajectory $\mathcal{T}_{\text{opt}}$. Connecting such as set of “must visit”-points is a typical application for the *Traveling Salesman Problem* (TSP). Each viewpoint must be visited exactly once while optimizing the overall path length. Even though TSP it is a NP-hard problem, efficient solvers exist for the comparatively small number of viewpoints. We use a TSP-solver developed as part of the *Google OR-tools* [10]. It allows to use a custom distance matrix between all viewpoints as input. This allows it to implicitly embed more metrics such as the change of yaw or inspection angle into the optimization. However, in the current version we simply use an Euclidean distance matrix in order to optimize for path length as primary objective. The TSP is then solved using the guided local search heuristic which is considered one of the most efficient sampling heuristics for routing problems.

4 Experimental Evaluation

The utilized planning framework allows to use the px4 *Software-In-The-Loop* component to run experiments with Gazebo as simulator. We tested the planning procedure on a number of different models. Figure 4.1 shows some of them, either taken from the Bauhaus Challenge [2], coming with the SIP-implementation [1] or created from real-world objects on our premises. Even if Gazebo is not capable of rendering the environment in a photorealistic way, it allows to test

	Parameter Name	Units	Default	Explanation
1	**Planner**			
	$d_{\text{min}}, d_{\text{max}}$	meter	$[6.0, 10.0]$	distance constraints
	Min. incidence angle	degree	$10°$	$\angle(a_n, n_{<\cdot>})$ c.f. Fig. 3.2
2	**Rotorcraft**			
	Max. velocity	m/s	2	
	Max. angular velocity	rad/s	0.5	
3	**Space boundary**			
	Max. space size	meter	$[200, 200, 50]$	x, y and z size
	Space center	meter	$[0, 0, 0]$	3D coordinate [x, y, z]
6	**Camera**			
	FOV	degree	$[120°, 120°]$	[horizontal, vertical] field of view
	Pitch	degree	$30°$	Pitch angle of the camera

Table 4.1: List of the most important parameters and their respective default values within the inspection framework.

results of the view-point sampling using different sensor setups and environment data.

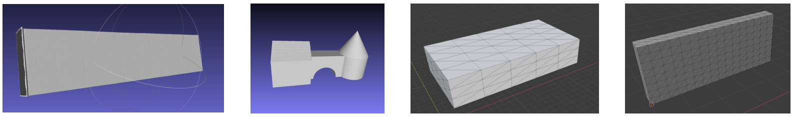

(a) Bridge Pier (b) House (c) Hall (d) Wall Model

Figure 4.1: Exemplary triangle mesh models used for experimental evaluation. The first two were taken from the Bauhaus Path Planning Challenge [2], while the last two have been created for simple ablation studies.

The framework has various parameters, which heavily influence the experimental results. We tried to use similar defaults as in [1]. Table 4.1 gives an overview over the most important parameters and their default values.

We quantitatively evaluate the planners in different configurations on a set of standard metrics. We leverage the *planning time*, *path length* and *mean yaw*

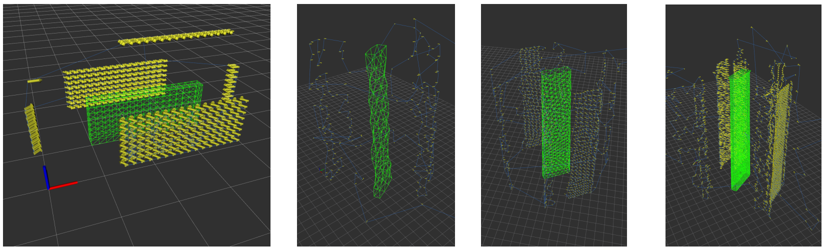
Model	Mesh Size facets	Path Length $[m]$	Mean Yaw Rate $[°]$	Planning Duration $[s]$
Hall	148	216	14.36	2.60
Artificial House	154	152	14.23	2.51
	186	408	21.17	2.63
Bridge Pier	965	441	11.09	8.85
	3930	2011	0.28	262.02

Table 4.2: Quantitative evaluation on the three main models hall, house and bridge pier. Mesh size denotes the number of triangles and planning duration the total planning time for all steps.

rate as main metrics. We also verify the number of *rejected* triangles. The path length is specified as $\mathcal{L} = \sum_{i=0}^{N-1} d_{i \to i+1}$ where N is the total number of viewpoints and $d_{i \to i+1}$ is the distance between the i^{th} viewpoint and the subsequent one. The mean yaw rate $\overline{\Delta\psi} = \frac{1}{N} \sum_{i=0}^{N-1} \|\Delta\psi_{i \to i+1}\|$ specifies the mean change in yaw angle over time, with $\Delta\psi_{i \to i+1}$ being the change in yaw between two consecutive viewpoints.

Table 4.2 shows the results for some of the models in different resolutions. All metrics are dependant on the number of triangles in the mesh. The main cause for an increasing path length are outliers in the viewpoint sampling, while the increased planning time results mostly from the exponential increase in TSP solving time. The decrease in yaw rate simply follows from the fact that the many viewpoints are interpolations of viewpoints from the lower resoluted mesh and thus not contributing to any turns of the UAV.

We qualitatively inspect the generated inspection trajectory on some of the models in Figure 4.2. For the simplest wall model in Figure 4.2(a) the viewpoint generation works as expected when running one iteration of viewpoint sampling. The remaining images in Figure 4.2 show the bridge pier in different resolutions. Even though the generated flight plan looks regular in general, more outlier viewpoints are generated for the higher resoluted meshes. The reason for this is typically the result from non optimal QP outputs for some difficultly placed

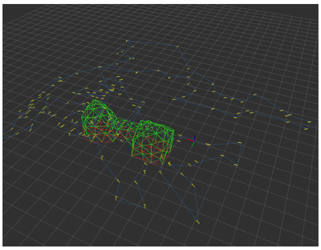


(a) Wall (b) Bridge Pier 186 facets (c) Bridge Pier 965 facets (d) Bridge Pier 3930 facets

Figure 4.2: Generated paths for the wall model and for different resolutions of the bridge model. The light green lines are the input mesh, blue the trajectory and the yellow arrows denote the viewpoints and their direction.

triangles. The solver is configured in a way, that some constraints may be relaxed, if a global optimum cannot be found. One approach to prevent such outliers from being sampled is to restrict the $d_{\max}$ parameter for the admissible sampling space. In addition, it could be considered to not sample one viewpoint per triangle but to combine multiple similar viewpoints in a later step to reduce the overall path length. This could also allow to filter sampled outlier viewpoints, which could lead to a smoother trajectory than in Figure 4.2(d). We observe a similar behaviour for the models in Figure 4.3. We use red to indicate triangles for which no viewpoints could be generated. As the camera pitch is fixed and the UAV cannot fly below the ground, all ground triangles are marked red in Figure 4.3(a) and therefore not participating in the trajectory planning. Figure 4.3(b) shows the inspection path on the simple hall model and additionally marks the UAV odometry from a simulated flight in red.

Despite of sampling some outlier viewpoints, we can observe a successful coverage for all tested models. This can also be verified by performing a reconstruction using the recorded images from the simulation. Figure 4.4 shows the result of this procedure on the Hall model. We post-processed the images with colmap [13] to obtain the sparse and dense reconstruction results. Note that the inspection planning itself does not target 3D-reconstruction applications in particular. We do not ensure within the viewpoint generation that a triangle


(a) House

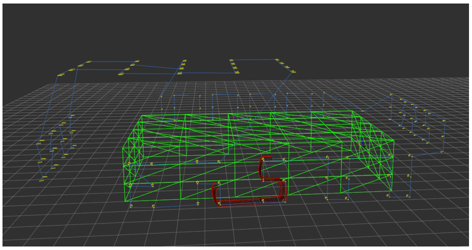
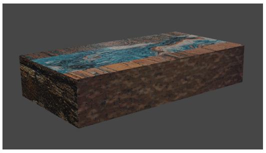
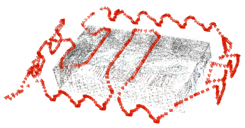

(b) Hall

Figure 4.3: Generated paths for the artificial house and the hall. In addition to the trajectory we mark rejected triangles in red.

must be observed from two distinct positions. Nevertheless, the generated paths often allow a dense reconstruction as two consecutive viewpoints oftentimes target neighboring triangles on the mesh, resulting in a stereo baseline for both triangles.


(a) Textured Hall Model


(b) Generated Inspection Plan

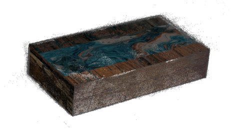

(c) Reconstructed Model

Figure 4.4: Application of the inspection planning for reconstruction purposes. The first image (a) shows a textured version of the model from Figure 4.1(c). We then use the generated inspection plan to fly it in simulation and perform a reconstruction using the saved images (b). Finally, (c) shows the dense reconstruction.

5 Conclusion and Future Work

In this work, we presented and evaluated a structural inspection pipeline for mobile robots using triangle meshes as input. We showed that intuitive inspection trajectories could be generated for a set of different models. In order to thoroughly test the inspection, we embedded it into a UAV autonomy pipeline with simulation capabilities. Using that simulation, we also applied the routine for the purpose of 3D-reconstruction. We identify the availability and variability of input data as main drawback in the presented approach. Given an arbitrary input mesh, it is a very error-prone pre-processing step to transform it into a regular triangle mesh with a desired amount of facelets. On the other side, the number of facelets is the only parameter to control the initial number of sampled viewpoints and thus the runtime required for the initial sampling step. Some approaches, such as ACVD [4] resulting from [14] exist to simplify existing meshes but as soon as the geometries get complex and non-convex, we were not able to produce a regular mesh (see 5.1(b)).

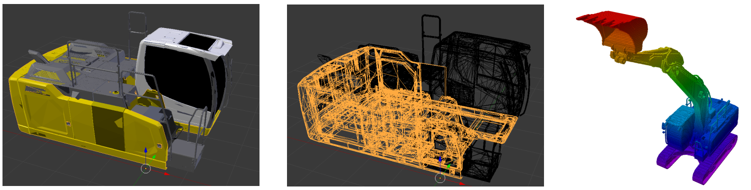

(a) Excavator (b) Upper Carriage Mesh (c) Voxelized Object

Figure 5.1: Example for an excavator model, which is difficult to tackle using a triangle mesh based inspection scheme. Using a dynamically generated voxelized version of the model such as in (c) might improve the viewpoint sampling and also allows resampling for different joint positions.

One way to overcome these limitations in the future is the usage of a different input modality. For instance, one could use a voxelized structure of the mesh such as visualized in 5.1(c), which is comparatively easy to generate even for dynamic joint positions. This approach would require a strategy to divide the voxel structure into different regions as the workload for sampling one viewpoint per voxel would be too high. This also raises the question, if the formulation as QP is even necessary if we only run one iteration of sampling and planning. We can influence the resulting inspection trajectory either by running multiple

[4] `https://github.com/valette/ACVD`

iterations or by adjusting the input modality in a way that less viewpoints are sampled in the first place.

Further extensions to the current approach are conceivable. It could be useful to explicitly encode inspection or reconstruction quality into the optimization. The first would require some dynamically generated distance constraints in order to achieve a user-definable ground sampling distance (GSD). The ladder requires that a triangle can be inspected from at least two viewpoints, so ideally each viewpoint must encode the constraints for multiple triangles.

6 Acknowledgments

A majority of this work is based on the implementation done by Shreyas Manjurnath during his Master Thesis, which has been carried out at Fraunhofer IOSB in 2021 and was supervised by Boitumelo Ruf and Raphael Hagmanns. Therefore, most ideas for the extensions presented in this work have been worked out in collaborative think sessions.

References

[1] A. Bircher et al. “Structural Inspection Path Planning via Iterative Viewpoint Resampling with Application to Aerial Robotics”. In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. May 2015, pp. 6423–6430.

[2] P. Debus and V. Rodehorst. “Evaluation of 3D UAS Flight Path Planning Algorithms”. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B1-2021 (2021), pp. 157–164. DOI: 10.5194/isprs-archives-XLIII-B1-2021-157-2021.

[3] Hans Joachim Ferreau et al. “qpOASES: a parametric active-set algorithm for quadratic programming”. In: *Math. Program. Comput.* 6.4 (2014), pp. 327–363. DOI: 10.1007/s12532-014-0071-1.

[4] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. "Informed RRT*: Optimal Incremental Path Planning Focused through an Admissible Ellipsoidal Heuristic". In: *CoRR* abs/1404.2334 (2014). arXiv: 1404.2334.

[5] Raphael Hagmanns. "Dynamic Planning Pipeline for Indoor Inspection Flights". In: *Proceedings of the 2021 Joint Workshop of Fraunhofer IOSB and Institute for Anthropomatics, Vision and Fusion Laboratory*. 2021, pp. 87–103. ISBN: 978-3-7315-1171-7.

[6] Wei Jing et al. "Coverage Path Planning using Path Primitive Sampling and Primitive Coverage Graph for Visual Inspection". In: Nov. 2019, pp. 1472–1479. DOI: 10.1109/IROS40897.2019.8967969.

[7] Nathan Koenig and Andrew Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. Sendai, Japan, Sept. 2004, pp. 2149–2154.

[8] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015 (June 2015), pp. 6235–6240. DOI: 10.1109/ICRA.2015.7140074.

[9] Parham Nooralishahi et al. "Drone-Based Non-Destructive Inspection of Industrial Sites: A Review and Case Studies". In: *Drones* 5.4 (2021). ISSN: 2504-446X. DOI: 10.3390/drones5040106.

[10] Laurent Perron and Vincent Furnon. *OR-Tools*. Version 7.2. Google, July 2019. URL: https://developers.google.com/optimization/.

[11] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: vol. 3. Jan. 2009.

[12] L. Schmid et al. "An Efficient Sampling-Based Method for Online Informative Path Planning in Unknown Environments". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 1500–1507. ISSN: 2377-3774. DOI: 10.1109/LRA.2020.2969191.

[13] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[14] Sebastien Valette, Jean-Marc Chassery, and Remy Prost. "Generic Remeshing of 3D Triangular Meshes with Metric-Dependent Discrete Voronoi Diagrams". In: *IEEE Trans. Vis. Comput. Graph.* 14 (Mar. 2008), pp. 369–381. DOI: 10.1109/TVCG.2007.70430.

[15] Feihu Yan et al. "Sampling-Based Path Planning for High-Quality Aerial 3D Reconstruction of Urban Scenes". In: *Remote Sensing* 13.5 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13050989.