D I S S E R T A T I O N

submitted

to the

Combined Faculty of the Natural Sciences and Mathematics

of

H E I D E L B E R G U N I V E R S I T Y, G E R M A N Y

for the degree of

Doctor of Natural Sciences

Put forward by

M.Sc. Silvan Lindner

Born in:

Meyrin, Switzerland

Date of oral examination:

.....

Optimization Based Coverage Path Planning for Autonomous 3D Data Acquisition

Supervisors

Prof. Dr. Katja Mombaur

Priv.-Doz. Dr. Christoph Garbe

Abstract

The demand for 3D models that represent real-world objects such as structures and buildings has increased in recent years. It is becoming increasingly important that the reconstructions are not only visually convincing but also feature high geometric accuracy. This includes, for example, the fields of civil engineering, terrestrial surveying and archeology, where precise measurements are made in the models for documentation and analysis purposes. There are different approaches to create such a reconstruction. The photogrammetric method Structure from Motion and laser scanning are among the most widely used methods here, as they do not require a complicated setup and can be used for scenarios at small to large scale. Recent developments are enabling unmanned robotic systems, especially sensor mounted UAVs, to assist in the recording of areas which are otherwise difficult to observe. The demand for a high geometric accuracy, however, comes at the expense of high computational complexity of up to several days. Hence, especially real-time reconstructions are unfeasible, such that recording and reconstruction procedure must be executed consecutively. The resulting model quality, i.e. completeness and accuracy, is only assessable afterwards. Since it is often difficult or even impossible to improve these models with additional measurements afterwards, methods that ensure a reliable acquisition of sufficient data is required.

In this thesis we develop new methods and theory that address this problem for the mentioned sensor types. For both, a probabilistic description of the expected surface reconstruction error is maintained cost-efficiently as an estimate for the model quality during the recording procedure. For image sensors this is realized by incrementally constructing confidence ellipsoids that describe the information obtained from all views. With depth sensors the surface quality is described by the variance of a Gaussian process implicit surface regression fit to point cloud data using polyharmonic kernel functions. Sensor poses are then assessed by the information they add to the subsequent reconstruction up to a desired geometric accuracy using a formulation that is motivated from Optimal Experimental Design. This quantity is further used in an iterative next-best-view selection framework as a subproblem of a coverage path planning problem.

The general formulations presented in this thesis enables a wide range of applications, such as offline and online view planning or various autonomous robot systems under consideration of dynamic and geometric constraints. We present the first multi-view coverage path planning approach, specifically targeted at autonomous Structure from Motion data acquisition. Its correctness is validated in simulation using the physics simulator Gazebo. Furthermore, we lay a foundation for similar applications with depth sensors. All presented algorithms were developed with scalability in mind and show promising results regarding real-time usability.

Zusammenfassung

Der Bedarf an 3D-Modellen, die reale Objekte wie Strukturen und Gebäude darstellen, hat in den letzten Jahren stark zugenommen. Gleichzeitig wird es immer wichtiger, dass diese Rekonstruktionen nicht nur visuell überzeugend sind, sondern auch eine hohe geometrische Genauigkeit aufweisen. Diese werden beispielsweise in den Bereichen Bauingenieurwesen, terrestrische Vermessung und Archäologie benötigt, wo präzise Messungen in den Modellen für Dokumentations- und Analysezwecke vorgenommen werden. Zu den am weitesten verbreiteten Rekonstruktionsmethoden gehören das photogrammetrische Structure from Motion und das Laserscanning, da sie keinen komplizierten Aufbau benötigen und für Anwendungen in kleinem bis großem Maßstab eingesetzt werden können. Eine neuere Entwicklung in diesem Bereich besteht darin, dass immer mehr unbemannte Robotersysteme, insbesondere sensorgestützte UAVs, unterstützend bei der Aufnahme von Bereichen eingesetzt werden, die sonst nur schwer zu beobachten sind. Eine hohe geometrische Genauigkeit zu erreichen, benötigt jedoch eine lange Rechenlaufzeit von bis zu mehreren Tagen. Daher sind insbesondere Rekonstruktionen in Echtzeit nicht möglich und die Datenerfassungs- und Rekonstruktionsverfahren müssen nacheinander ausgeführt werden. Die resultierende Modellqualität, d.h. Vollständigkeit und Genauigkeit, wird erst anschließend ersichtlich. Da es oft schwierig oder gar unmöglich ist, diese Modelle anschließend mit zusätzlichen Messungen zu verbessern, sind Methoden erforderlich, die eine zuverlässige Erfassung ausreichender Daten gewährleisten.

In dieser Arbeit entwickeln wir neue Methoden und Theorien, die dieses Problem behandeln. Dafür werden probabilistische Größen eingeführt und während des Aufnahmevorgans aktualisiert, die den zu erwartenden Rekonstruktionsfehler schätzen. Bei einem kamerabasierten Verfahren wird diese aus Konfidenzellipsoiden konstruiert. Im Falle von Tiefensensoren wird die Objektoberfläche implizit durch einen Gauß-Prozess aus Punktwolkendaten unter Verwendung polyharmonischer Kovarianzfunktionen modelliert. Die Qualität lässt sich dann anhand der zugehörigen Varianz beschreiben. Damit lassen sich nun die Informationen, die eine Sensorposition zur Rekonstruktion beitragen, bewerten. Dies ist aus der optimalen Versuchsplanung motiviert. Diese Größe wird weiter zur Planung neuer Sensorpositionen, als Unterproblem von Coverage Path Planning, benutzt.

Die in dieser Arbeit vorgestellten Lösungsansätze sind allgemein formuliert. Dadurch ergibt sich ein breites Anwendungsspektrum, wie zum Beispiel für Offline- und Online- Pfadplanungen, sowie die Unterstützung verschiedener autonomer Robotersysteme. Wir stellen den ersten Multi-View Coverage Path Planning Ansatz vor, der speziell auf die autonome Datenerfassung für Structure from Motion ausgerichtet ist. Dessen Korrektheit wird in der Simulation mit dem Physiksimulator Gazebo validiert. Darüber hinaus schaffen wir die Grundlage für ähnliche Anwendungen mit Tiefensensoren. Alle vorgestellten Algorithmen wurden mit Blick auf die Skalierbarkeit entwickelt und zeigen vielversprechende Ergebnisse hinsichtlich ihrer Echtzeit-Nutzbarkeit.

Contents

Acknowledgments

First and foremost I would like to express my gratitude to my supervisors Prof. Dr. Katja Mombaur and Priv.-Doz. Dr. Christoph Garbe for believing in me and giving me the opportunity for this research project. Without their guidance, support, and motivation this thesis would not have been possible.

I am deeply grateful to the Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences (HGS MathComp) for supporting me financially with a scholarship. The courses they offered for Ph.D. fellows and the possibility of connecting with scientists from different fields from all over the world was immensely helpful and enabled for interesting interdisciplinary discussions.

I would also like to thank all staff at Heidelberg University that handled bureaucracy and had a huge impact in making the last few years a pleasant experience. This especially includes Sarah Englert from our research group, as well as people from the Faculty of Mathematics and Computer Science, the HGS and the IWR.

Very special gratitude goes to my colleagues of the research group Optimization Robotics & Biomechanics, especially all current and previous office members, Benjamin Reh, Christian Seitz, Monika Harant, Felix Aller and Alexander Stepanov, as well as Marina Horn, I had the pleasure working with. They taught me a lot on a scientific and personal level. Their friendship, support, feedback, the interesting discussions and the fun we had are the most valuable experiences I gathered throughout my Ph.D. studies. Thank you for the amazing time. Furthermore, I want to thank Jonas Große Sundrup for helpful discussions, especially about functional analysis, and Dr. Matthew Millard for motivational support and mentoring regarding scientific practices.

Last but not least, I would like to express my deepest gratitude to my family and friends who unconditionally supported and encouraged me throughout these years in the pursuit of this project. This especially includes my parents, which I am very proud of and who made me who I am. Most importantly, I wish to thank my loving and supporting wife Fata, who always gave me strength and inspiration over the last 10 years. Without you, I would not have made it.

Acronyms & Abbreviations

ALS	-	**A**irborne **L**aser **S**canning
COLLADA	-	**COLLA**borative **D**esign **A**ctivity
CPP	-	**C**overage **P**ath **P**lanning
CPU	-	**C**entral **P**rocessing **U**nit
CUDA	-	**C**ompute **U**nified **D**evice **A**rchitecture
DDA	-	**D**igital **D**ifferential **A**nalyzer
FOV	-	**F**ield **O**f **V**iew
GNSS	-	**G**lobal **N**avigation **S**atellite **S**ystem
GPIS	-	**G**aussian **P**rocess **I**mplicit **S**urface
GPS	-	**G**lobal **P**ositioning **S**ystem
GPU	-	**G**raphics **P**rocessing **U**nit
HOG	-	**H**istogram of **O**riented **G**radients
ICP	-	**I**terative **C**losest **P**oint
k-NN	-	**k-N**earest **N**eighbors
LiDAR	-	**Li**ght **D**etection **A**nd **R**anging
NBV	-	**N**ext **B**est **V**iew
ODE	-	**O**rdinary **D**ifferential **E**quation
OED	-	**O**ptimal **E**xperimental **D**esign
RANSAC	-	**RAN**dom **SA**mple **C**onsensus
RBF	-	**R**adial **B**asis **F**unction
RKHS	-	**R**eproducing **K**ernel **H**ilbert **S**pace
ROS	-	**R**obot **O**perating **S**ystem
RRT	-	**R**apidly-exploring **R**andom **T**ree
RTK	-	**R**eal-**T**ime **K**inematic
SfM	-	**S**tructure **f**rom **M**otion
SIFT	-	**S**cale-**I**nvariant **F**eature **T**ransform
SLAM	-	**S**imultaneous **L**ocalization **A**nd **M**apping
SMART	-	**S**patial **M**easure for **A**ccelerated **R**ay **T**racing
SQP	-	**S**equential **Q**uadratic **P**rogramming
SURF	-	**S**peeded **U**p **R**obust **F**eatures
TIFF	-	**T**agged **I**mage **F**ile **F**ormat
TLS	-	**T**errestrial **L**aser **S**canning
TVP	-	**T**arget **V**isitation **P**roblem
UAV	-	**U**nmanned **A**erial **V**ehicle
UGV	-	**U**nmanned **G**round **V**ehicle
URDF	-	**U**nified **R**obot **D**escription **F**ormat
UUV	-	**U**nmanned **U**nderwater **V**ehicle
XACRO	-	**X**ML M**acros**

1 Introduction

1.1 Motivation

In recent years, the demand for high-resolution large scale 3D reconstructions has increased rapidly. For many scientific fields, it is not sufficient to simply create a visually appealing model. Rather, high geometric accuracy is required to carry out measurements and analyses to scale. Examples of this can be found in many areas.

In civil engineering, highly detailed models are used to assess structural stability as a cost and time-efficient alternative to conventional visual investigations. This is important to ensure the safety and usability of those structures. Popular applications include maintenance and inspection tasks of civil infrastructure, such as bridges ([Lattanzi and Miller, 2014], [Chen et al., 2019]) and tunnels ([Attard et al., 2018]). The required data acquisition is usually assisted by robotic systems, such as unmanned aerial vehicles (UAVs). The 3D representation allows the automatic detection of material defects such as cracks ([Jahanshahi et al., 2013]), deformations ([Hallermann et al., 2014], [Bhadrakom and Chaiyasarn, 2016]) and the analysis of surface structure and properties.

Another application of these scale models is in the field of terrestrial surveying, where they are used for analysis and evaluation of terrain properties. Besides small scale micro-topography analysis [Lucieer et al., 2014b], large scale surveys of landslides [Lucieer et al., 2014a], coastal environments [Mancini et al., 2013] and river flood-plains [Bakker and Lane, 2017] are performed. Data acquisition is sped up and simplified with UAVs. There are also applications for other types of robots. For example, unmanned underwater vehicles (UUVs) are used in [Pizarro et al., 2004] and [Storlazzi et al., 2016] to survey the ocean floor.

Three dimensional reconstructions are already extensively used in archaeology. There, very detailed models are created for the documentation of cultural heritages and excavation sites. These are subsequently used for an extensive analysis, e.g. using orthographic projections, cuts and contour profiles. Especially the photogrammetric method Structure from Motion (SfM) has become an important tool for archaeological documentation (e.g. [Green et al., 2014], [Lo Brutto and Meli, 2012], [López et al., 2016]). Besides a highly accurate 3D model, it additionally yields high-resolution surface textures crucial for those documentation tasks. SfM proves to be cost-efficient, easy to use, versatile in rough environments and allows for fast data acquisition. The latter in particular is very important, as the recording time is often limited. As for the beforehand mentioned examples, the aid of unmanned robotic systems for the data acquisition gained popularity in the last few years, especially for large scale objects and structures. In [Chiabrando et al., 2015], [Lo Brutto et al., 2014] and [Aicardi et al., 2016], UAVs are used to capture and

document cultural heritage sites using SfM. There are also examples where different reconstruction methods are employed simultaneously, e.g. [Xu et al., 2014] and [Valenti and Paternò, 2019] where SfM is used in combination with terrestrial laser scanning (TLS).

The biggest drawback of SfM is its high runtime requirement. Depending on factors like the number of acquired images and their resolution, a high detail reconstruction may even take several days to fully compute. This introduces further problems regarding the recording processes. When manually taking the required pictures, the distribution of their poses is often very uneven, over-observing certain areas of the geometries surface while others are captured poorly (see figure 1.1). Hence, the main difficulty lies in determining a set of images that provides good ob-

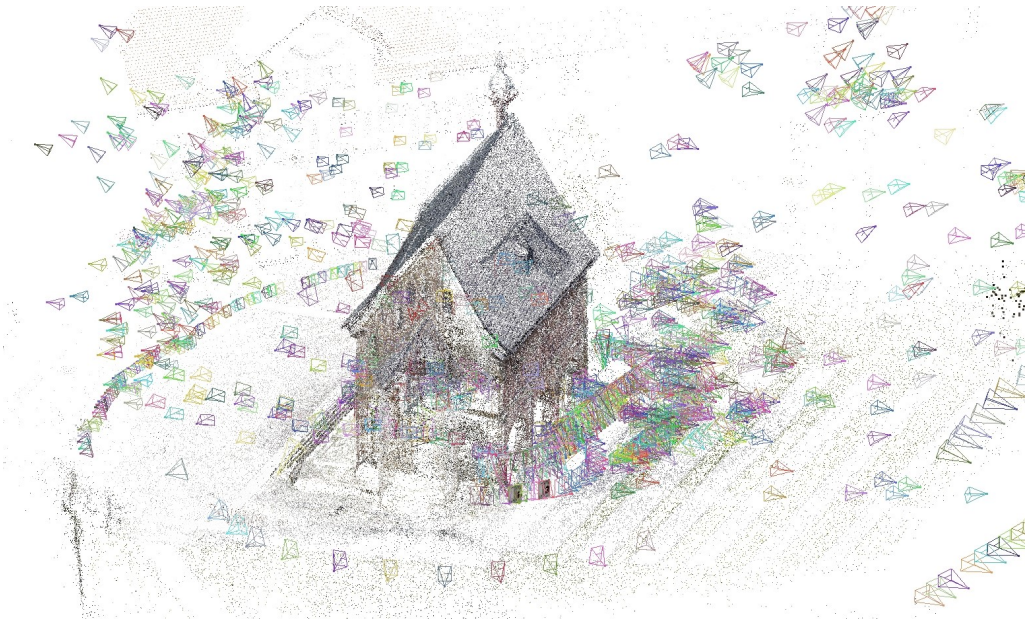


Figure 1.1: Choice of camera positions for an SfM reconstruction of the King's Hall of Lorsch Abbey (UNESCO World Heritage Site) by hand. A total amount of 1300 images were taken from a manually controlled UAV with strongly varying recording pose densities.
Source: ©2016 Christian Seitz

servability of all points on the object's surface. If the distribution of recording poses is too sparse, few or no feature correspondences can be detected between disjoint subsets of images. This leads to erroneous feature matching, a subsequent bad or wrong camera pose estimation and ultimately in a bad reconstruction quality (i.e. high reconstruction error and missing data), which is only visible after concluding all computations. The density of the SfM dense point cloud may then be too sparse or noisy in certain areas that have not been sufficiently observed, or, in the worst case, introduces holes (figure 1.2) or misaligned object subsets (figure 1.3). Those problems are unfortunately common in the SfM community and require subsequent improvements by increasing the image set size. However, those improvements introduce further problems due to different light and weather conditions, possibly changed object geometry or texture. In the worst case, they may even be impossible, which is common in archeology, where excavations are often time-limited or already too far advanced.

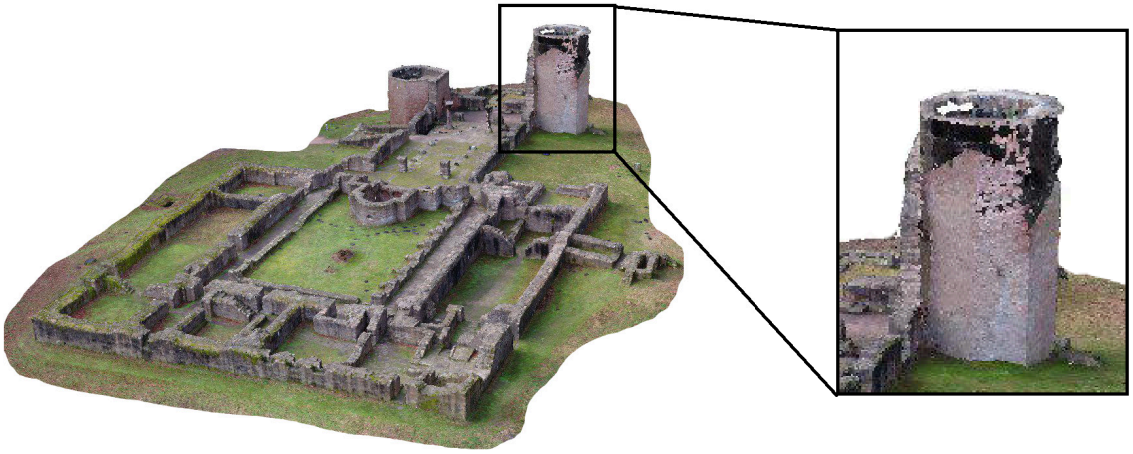


Figure 1.2: Faulty SfM reconstruction of the Monastery of St. Michael (Heidelberg). The right tower was not sufficiently observed leading to holes in its SfM reconstruction.
Source: ©2018 Christian Seitz

Figure 1.3: Faulty SfM feature matching of a meerschaum pipe. Two sets of images recorded on both sides of the pipe do not have sufficient overlap resulting in wrong camera pose estimations and wrong feature matching.

Acknowledgement: The meerschaum pipe was generously provided by the Reiss-Engelhorn-Museen Mannheim for the ArchEye Automatics project ([Seitz and Altenbach, 2011], [Seitz, 2012]).

Similar problems exist for data acquisition with a laser scanner. A reconstruction is built from a set of erroneous point measurements. In order to obtain a model representation with high geometric accuracy, all individual measurements are registered in a global point cloud first. A surface mesh is then usually created from a regression fit. Examples of surface reconstruction methods can be found in [Berger et al., 2014]. Computing these meshes is computationally expensive and results in long runtimes, such that the reconstruction algorithms must be run offline in post-processing. Only then can the resulting model quality be evaluated and deficits be identified. These, again, correspond to holes in the reconstruction or uncertainties about the surface shape, i.e. a locally insufficient level of detail. They are caused by occlusion in the individual measurement observations or insufficient surface point densities in areas with high geometric fidelity. While it is possible to maintain an approximate voxel discretization of measurement data during runtime to identify unobserved areas, the voxel sizes pose a limit to the detectable model resolution.

These problems are addressed in this thesis. Instead of selecting sensor poses manually, we choose them based on the information they will contribute to a subse-

quent reconstruction. This class of problems is called next-best-view (NBV) planning. If, in addition, a collision-free recording path that provides full coverage of the structure is to be achieved, it belongs to the class of coverage path planning (CPP) problems. We will present the first NBV planning approach that solves a multi-view CPP problem in the SfM case. The term "multi-view" here refers to the fact that depth can not be extracted from a single image and each surface point must be observed sufficiently often to determine its spatial position correctly. Furthermore, a foundation is laid to extend similar techniques we develop in this context to laser scanning. More specifically, we propose a method that allows maintaining a surface model of a point cloud that scales well with the number of points and similarly allows for efficient NBV assessments. We will now give a short introduction and an overview of related work regarding the different recording methods, NBV planning and CPP.

1.2 Related Work

1.2.1 Structure from Motion and Laserscanning

Most of the application examples mentioned in the previous section utilize Structure from Motion (SfM) (e.g. [Agarwal et al., 2011]) to create the 3D models, while some employ light detection and ranging (LiDAR) sensors, e.g. terrestrial laser scanning (TLS) or airborne laser scanning (ALS). When it comes to creating such highly detailed models on a medium to large scale, these are the state-of-the-art methods. On a smaller scale, structured light scanning systems are used alternatively. Over the years, SfM has proven itself as a cost-effective alternative to especially laser scanning. SfM requires a set of images captured by (consumer-grade, high resolution) cameras to calculate 3D reconstructions using methods from computer vision[1]. Regarding the reconstruction accuracy, SfM was considered inferior to TLS for a long time. However, recent advances in technology and algorithms for computer vision have leveled the odds. Recent studies show that both methods can compete, as they result in comparative model qualities ([Zhang et al., 2016], [Carraro et al., 2019]). This statement proofs to be true even for large scale structures and high distance recordings as shown in [Bolognesi et al., 2014]. They compared reconstructions from SfM using a set of images taken from a distance of 50 m with a TLS scan and observed that discrepancies between both point clouds did not exceed 3 cm. [Skarlatos and Kiparissi, 2012] even suggests that modern image-based methods may replace TLS standard solutions for small and medium size objects. While SfM can already achieve sub-millimeter precisions in certain scenarios, the sensor accuracy of laser scanners is often still higher than the accuracy of individual points of an SfM reconstruction. However, the latter usually produces a higher point cloud density which benefits a subsequent mesh generation that smooths out these errors. Further differences exist in the accuracy of sensor pose estimations. Laser scanning approaches are usually either ground-based and require manual, precise pose calibration or use the iterative closest point (ICP) algorithm [Besl and McKay, 1992] to merge multiple measurements. The accuracy of the latter highly depends on the point cloud overlap and densities. SfM utilizes thousands of features extracted from

[1]For a more detailed explanation of SfM consider section 2.

each image to obtain a robust and highly accurate pose estimation, leading to high flexibility in application. This allows for easy integration into robotic systems such as UAVs and UUVs that support the user to record areas that would otherwise be difficult to observe (e.g. roofs). Such integrations are already very widespread as seen in the beforehand mentioned example applications.

The high geometric accuracy of SfM comes at the cost of computational complexity, such that the construction of models in real-time becomes infeasible. While other impressive photogrammetric approaches exist that are capable of performing incremental reconstruction in real-time using consumer-grade phone cameras (for example [Newcombe et al., 2011] [Nießner et al., 2013] [Klingensmith et al., 2015]), those suffer from compromises between image resolution, frame rate, and reconstruction accuracy. In comparison, SfM being an offline method does not require sacrificing accuracy at the cost of run-time. Especially all recorded data is already available before executing the algorithm. Hence, joint bundle-adjustment is performed over all poses simultaneously, increasing the overall camera pose estimation accuracy and mitigating drift. More expensive and more robust Histogram-of-Oriented-Gradient (HOG) features (SIFT [Lowe, 1999] / SURF [Bay et al., 2006]) can be extracted, matched and tracked on the higher resolution images. The higher resolution images additionally allow for a more photo-realistic texture mapping.

1.2.2 Next-Best-View Planning

NBV planning has been a very active field of research over the last 40 years. A good overview of different types of NBV algorithms is given in [Scott et al., 2003]. They describe the classic model building cycle in four steps:

- **Planning:** A new view is planned based on a utility or gain function that assesses a selection of admissible sensor poses. The viewpoint space is often constraint, e.g. to lie on the surface of a cylinder [Pito, 1999] or sphere [Banta et al., 2000] to improve planning performance.
- **Scanning:** A scan is performed at the beforehand planned pose.
- **Registration:** Acquired data is registered with previous measurements, e.g. transformed into a global map using the ICP method [Besl and McKay, 1992].
- **Integration:** The new information is integrated into a globally consistent model. The model is usually maintained either as a surface based (e.g. mesh) or a volumetric (e.g. voxel occupancy map) representation.

The class of problems where no prior information is available is called non-model-based. Methods, where a priori knowledge of the object model is available at some fidelity, are classified as model-based. Those allow for a simulation of the model building cycle, such that all views can be planned offline. Another more recent overview of common NBV approaches is given in [Chen et al., 2011]. They performed a broad survey on the development of NBV planning, focusing on various applications in autonomous robotics. Those fields include autonomous navigation and exploration, inspection, modeling and grasp planning. Each task involves different requirements on the model to be captured. For example, while a rough voxel representation may be sufficient for autonomous navigation, the main purpose of

model perception is to obtain a highly detailed geometric representation. These different application purposes have resulted in a wide range of task-specific NBV approaches that address their respective requirements. However, the underlying methods are always similar and especially the basic model building workflow mentioned above applies. Some recent application examples are stated below.

Consider the field of non-model-based 3D surface reconstruction using depth sensors. While pure voxel-based space carving approaches, which use the number of unobserved voxels in their measure for the NBV selection (e.g. [Banta et al., 2000]), were most popular several years ago, additional quality criteria are now included in the utility function. In [Vasquez-Gomez et al., 2014] the authors include terms related to scan overlay, the path cost, the angle between view direction and surface normals, as well as a factor describing the amount of occlusion plane voxels (i.e. voxels bordering free and occluded space) in their utility function while still operating in a voxel map. Another popular approach is given in [Kriegel et al., 2015]. Besides a voxel map, they additionally maintain a surface mesh with fixed minimal resolution. Their utility function also depends on an exploration term related to a voxel map, but further includes quantities computed from a boundary detection and surface trend estimation on the mesh. Those include the angle of incidence, the relative point density and the border edge percentage (i.e. the amount of all visible border edges divided by the amount of all visible edges).

Autonomous mapping and exploration tasks also count to non-model-based NBV planning problems. Here, probabilistic voxel maps are still the most widespread volumetric model representation due to their excellent performance on robotic platforms. In this context, a selection of various volumetric information gain metrics is introduced and evaluated in [Delmerico et al., 2018]. Those metrics include terms related to visibility and occlusion, relative geometric relations to neighboring voxels, and voxel entropies. An example application of such a metric for autonomous exploration of an unknown volume or surface manifolds is given in [Bircher et al., 2018]. They create random viewpoint candidates using a rapidly-exploring random tree (RRT) [Karaman and Frazzoli, 2011], which are evaluated online on a UAV. A fundamentally different approach is given by [Jadidi et al., 2014]. They estimate continuous occupancy maps using a Gaussian process without the need for grid discretization (see [Ramos and Ott, 2016], [O'callaghan and Ramos, 2012]). Frontier maps (boundaries between known-free and unknown areas) are extracted from the Gaussian process and used in an information gain formulation to determine NBV poses.

Similar approaches for NBV computations also exist for surface inspection tasks, e.g. [Hollinger et al., 2012]. They use probabilistic regression to incrementally train a Gaussian process that implicitly describes the geometry for the purpose of model-based inspection planning. This surface is called Gaussian process implicit surface (GPIS) [Williams and Fitzgibbon, 2007]. NBVs are then picked as sensor poses that reduce the uncertainty (i.e. covariance function) on the surface mesh. Their method, however, requires an a priori mesh of the model. Other large scale terrain modeling GPIS approaches [Vasudevan et al., 2009] [Hadsell et al., 2010] exist that could also be equipped with a similar NBV selection strategy.

GPIS are already very popular in modeling small objects from measurements of tactile sensors [Dragiev et al., 2011], [Ottenhaus et al., 2016]. There, only a small amount of measurements are integrated into the model in each iteration, such that

the computational complexity of the Gaussian process stays reasonable. This allows for online haptic exploration, commonly used in grasp planning, which is another task associated with non-model-based NBV planning. Examples are given in [Caccamo et al., 2016] and [Huang and Hermans, 2019] where the next best measurement pose is again determined from the surface model uncertainty given by covariance information.

1.2.3 Coverage-Path-Planning

The field of coverage path planning (CPP) addresses the task of determining a collision-free path that allows the observation of all points of a volume or area of interest. An extensive survey of state-of-the-art CPP approaches is given in [Galceran and Carreras, 2013] and includes cellular decompositions, graph-, or grid-based schemes. These allow for a wide range of applications. Some examples of paths that follow simple geometric patterns are given below.

In [Yakoubi and Laskri, 2016] the authors present an evolutionary approach using a genetic algorithm to compute a coverage trajectory of a confined 2D region with obstacles for a cleaning robot. [Torres et al., 2016] consider polygonal shapes in the context of UAV terrain coverage for image-based 3D reconstructions. They use a line sweep algorithm to obtain piecewise linear path segments in a plane that guarantee a specified image overlap. An application for underwater inspection is given in [Galceran et al., 2014]. They cover complex structures on the ocean floor using paths obtained from intersecting horizontal planes with the target region at uniformly spaced depths. A similar strategy is employed in [Peng Cheng et al., 2008]. A time-optimal UAV flight path is computed from horizontally sliced cylinders, providing complete 3-dimensional coverage of 2.5-dimensional features in urban environments.

These pattern-based algorithms, however, do not take the quality of the observed area into account. CPP approaches that do consider this quantity for the view selection are closely related to NBV planning. More precisely, by iteratively performing NBV planning and interconnecting the resulting sensor poses, full coverage is achieved. For example, the beforehand mentioned NBV 3D surface reconstruction algorithms [Vasquez-Gomez et al., 2014] and [Kriegel et al., 2015], as well as the autonomous exploration approach by [Bircher et al., 2018] fall into this category. The main task of extending NBV planning to CPP is to connect individual sensor poses. For many years, view planning and path planning were treated as separate problems. All sensor poses were computed first and were subsequently connected as a solution to a target-visitation problem (TVP), yielding the shortest connecting path. However, this two-step optimization scheme suffered from certain limitations. Depending on the optimality criteria, the resulting trajectory may not be optimal since path costs could not be considered in the NBV planning. Furthermore, even if the computed poses provide full coverage, there is no guarantee that a solution to the TVP exists that is capable of reaching all views, due to robotic specific dynamic constraints or cluttered environments. In recent years, fundamentally different approaches that rely on rapidly-exploring random trees (RRT) [Karaman and Frazzoli, 2011] have gained popularity. Here, the planning step of the NBV selection is adjusted slightly. A random tree from the robot's current pose is grown, which especially also considers the robot's dynamics and geometric con-

straints. The utility function is then evaluated for all computed trajectory endpoints and the branch with the highest score is chosen as NBV. Executing this NBV planning iteratively ultimately results in an admissible coverage path. Note that this procedure does not yield optimal trajectories, i.e. neither a minimum set of views nor the shortest coverage path. Optimality can be achieved nonetheless with model-based offline planning. There, a refinement procedure can be used to iteratively resample the random tree. Popular examples employing RRT* to solve 3D CPP problems include the rapidly-exploring random tree of trees (RRTOT) method [Bircher et al., 2017], the random inspection tree algorithm (RITA) [Papadopoulos et al., 2013] and the method introduced in [Englot and Hover, 2013] employing the sampling-based subroutine $\text{RRT}^*_{||}$. An application of another resampling approach for autonomous 3D structural inspection using aerial robots is given in [Bircher et al., 2016].

In order to obtain an SfM reconstruction, each point on the geometries surface must be observed a sufficient number of times in the recorded images to allow for proper triangulation. Depending on the distance of the recording pose to the surface and spatial relations to other viewpoints, multiple views of the same area may be necessary to achieve a desired reconstruction quality. Since current CPP approaches only guarantee to observe a surface once, they are not suitable for this task. While some of them are targeted at 3D reconstructions from photogrammetry (e.g. [Torres et al., 2016]), these only consider image overlap for simple geometric primitives. This is not sufficient for our application, because we consider complex 3D structures where the geometry may not be known in advance.

1.3 Contribution & Outline

This thesis covers two main topics.

In part II, we present the first full 3D multi-view CPP algorithm that aims at collecting images, such that a subsequent SfM reconstruction is complete and satisfies certain quality constraints. This section has already been partially published by us in [Lindner et al., 2019]. An estimator is constructed that estimates the expected reconstruction quality (i.e. reconstruction error) of surface points during the recording procedure. Besides its quality estimation properties, it also indicates whether a point was observed and whether it can be reconstructed at all. Based on the covariance information obtained from our estimator, a utility function is developed that measures information gain for arbitrary viewpoints. For that purpose, we utilize methods from optimal experimental design (OED) to measure the gain in entropy a new view provides, i.e. the change in relevant information for the reconstruction. The theory developed for quality estimation and NBV planning is given in chapter 6. Afterward, we follow the standard NBV - RRT cycle explained above to solve the CPP problem: multiple viewpoint candidates are obtained from RRT samples that also consider the utilized robotic system's dynamics. The highest score trajectory is executed and quality estimates inside the camera field of view (FOV) are updated. We present different strategies for performing these updates with various model representations, for use in both model-based (offline) and non-model-based (online) planning. The corresponding algorithm for practical application is given in chapter 7. Because of the general formulation given there, a high degree of flexibility is achieved. This allows for the adaption of our algorithms for a wide variety of robots and specialized RRT sampling methods. Especially RRT resampling schemes could

be utilized for offline planning to obtain optimal trajectories, which is a topic for future research. The evaluation of the utility function proofs to be the bottleneck in the computational complexity of our algorithm. Hence, we give implementation details on crucial performance culprits (e.g. ray-casting in voxel maps) in chapter 8, such that a fast runtime is achieved. This is especially important for use cases for online planning. A simulation environment is described in chapter 9 that allows for a comparison between obtained SfM reconstruction with a ground-truth reference mesh, such that the estimate on the expected reconstruction quality can be analyzed. We verify and analyze the derived algorithm in this simulation environment in chapter 10. While our main application is in archaeology for the documentation of structures, the general formulation allows for a wide range of other use cases. Possible further application scenarios include UAV aided reconstructions of large scale areas, small scale photogrammetric based surface scanning using robotic arms, or to provide a guide for manual photography, each for real-time non-model-based planning or model-based offline planning, where optimal coverage trajectories can be precomputed.

In part III we lay the foundation to extend similar methods we developed in part II to an application with depth sensors. As in [Vasquez-Gomez et al., 2014] and [Kriegel et al., 2015], we want to assess the quality of the observed surface. Instead of geometric measures, however, a Bayesian regression approach is employed. A local GPIS [Williams and Fitzgibbon, 2007] is used to represent the model surface similar to [Vasudevan et al., 2009]. The main benefit over mesh-based representations is that – in theory – arbitrary small geometric details can be described by the implicit surface using a probabilistic model. The covariance information then reflects the surface quality as in [Hollinger et al., 2012]. The kernel function determines the surface properties, i.e. inter- and extrapolation behavior. In order to maintain surface orientations or curvature when extrapolating to unmapped areas, we use polyharmonic kernel functions [Wahba, 1990]. They are augmented as in [Solak et al., 2003] [Wu et al., 2017] to include observations of surface normals to improve the model representation. The primary difficulty in utilizing a similar NBV - RRT approach as in part II lies in computing ray-model intersections for arbitrary viewpoints to evaluate a utility function.

Our main contributions in this second part are twofold. First, we give a clean derivation of GPIS with polyharmonic kernels in the context of surface modeling in chapter 11. Especially the connection to polyharmonic spline interpolation is highlighted. We believe that such an approach to this topic does not yet exist to this extent. Chapter 12 addresses the problem of computing GPIS-ray intersections, required for a NBV gain function. Two methods are presented that efficiently solve this problem and especially allow for large scale environments. One approach calculates these intersections using iterative ray-marching directly from a point cloud without the need for any explicit surface representation. The other approach maintains a tetrahedral mesh of the implicit surface, such that intersections can efficiently be extracted from the depth buffer of a GPU. This mesh is updated for each new NBV. In addition, we provide an algorithm that requires almost constant runtime for local mesh approximations, independent of the size of the observed point cloud.

This thesis brings together different topics from computer vision, statistics, numerical optimization, and autonomous robotics. Therefore, a short but detailed insight into various topics is given in part I. These include structure from mo-

tion (chapter 2), optimal experimental design (chapter 3), occupancy grid mapping (chapter 4) and Gaussian processes (chapter 5). The level of detail of the individual components of each topic depends on the extent to which they are later used.

For the remainder of this thesis, we assume that the locations of all robot and camera poses are known. We believe that this assumption can be made since simultaneous localization and mapping (SLAM) is a very active field of research that is already achieving impressive results in localization accuracy. Furthermore, additional hardware exists that achieves millimeter-precise localization accuracies with external positioning systems like the real-time kinematic (RTK) navigation technique.

Part I

Preliminaries

Structure from Motion 2

Structure from motion (SfM) is the process of estimating three-dimensional structures from a set of two-dimensional images. This photogrammetric technique is versatile since consumer-grade digital cameras suffice to obtain a highly accurate representation of the recorded objects on small to large scale. The resulting model quality is even comparable to those obtained from other methods such as LiDAR or structured light scans. As one camera is sufficient as a recording device, SfM comes at a comparatively low cost and does not require extensive specialized training. This also enables applications in special environments where other methods are unsuitable. Furthermore, the images obtained can additionally be used for texture mapping allowing for photo-realistic representations with high geometric and visual accuracy.

The main drawback of SfM is its high computational complexity. For example, creating models from thousands of images may take days to weeks to process on consumer hardware. As stated in the introduction, this is the main motivation for the NBV planning derived in part II. There, we strive to collect a small set of images that sufficiently captures the full scene in order to reduce the image set size and consequently computational time, while still obtaining high accuracy and complete models.

Roughly speaking, SfM triangulates each point of an object through several observations in overlapping images, as shown in figure 2.1. A more detailed outline

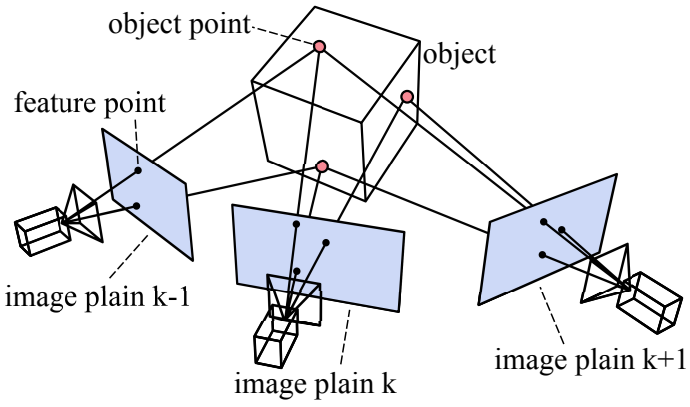
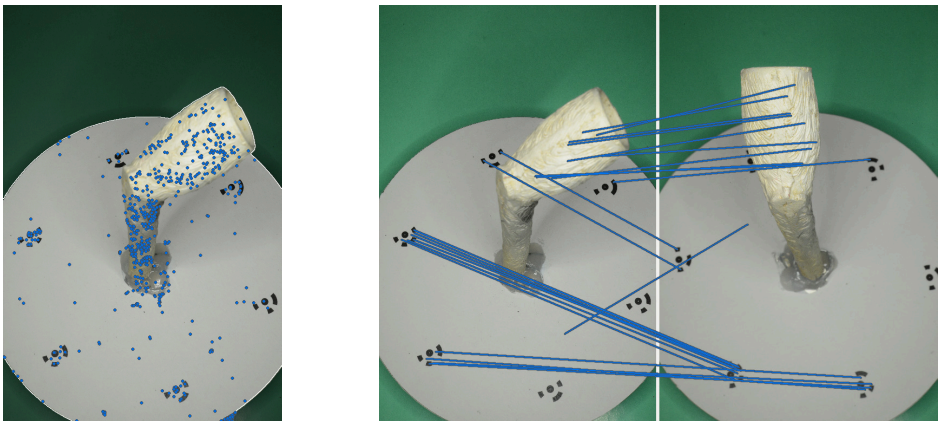


Figure 2.1: Simplified visualization of the SfM method. Feature points are observed in multiple images, allowing for camera pose estimation and object point triangulation.

of the main steps of the SfM workflow is given below in chronological order. It is worth noting that in the SfM pipeline, algorithms with higher accuracy are usually preferred to faster approaches with less accurate output. The individual steps are illustrated with an example.

Feature extraction

Given a set of images, features are extracted in each of them. A feature point is a salient point that usually corresponds to a corner obtained by an edge detector. More elaborate features can also range from edges over blobs to complex objects. Those points are further converted to a descriptive formulation that uniquely characterizes each feature and allows the identification of equivalent features across the set of images. Therefore, it is also important that the extractor selects points that are stable under viewpoint and lighting differences. Usually, a descriptor computes a unique identifier from the intensities of a region around the feature point. *Scale Invariant Feature Transformation* (SIFT, [Lowe, 1999]) and *Speeded-Up Robust Features* (SURF [Bay et al., 2006]) are popular descriptors that are commonly used for SfM. There, a feature is described by histograms over pixel intensity gradients. This yields descriptors with high robustness to small viewpoint changes, local affine distortions, changes in illumination, noise and partial occlusion at the cost of comparatively high computational time. An example of feature extraction is shown in figure 2.2a.



(a) Feature Extraction. **(b)** Feature Matching.

Figure 2.2: Feature extraction and matching using Agisoft Photoscan [Agisoft, 2017] on the example of a meerschaum pipe. Blue points correspond to extracted features, while blue lines indicate feature correspondences between two images.

Acknowledgement: The meerschaum pipe was generously provided by the Reiss-Engelhorn-Museen Mannheim for the ArchEye Automatics project ([Seitz and Altenbach, 2011], [Seitz, 2012]).

Feature matching

Feature matching identifies feature correspondences between images. While this step depends on the descriptor used, this is usually merely a comparison of all identified features. The feature matching step is visualized on an example in figure 2.2b. To receive many feature correspondences, an adequate image overlap among all images is necessary.

Camera pose estimation

Given matching features in multiple overlapping images, the camera poses can be estimated. Each feature corresponds to an ideally unique object point. The camera

pose is obtained by minimizing the reprojection error given a camera model. Usually, the simple pinhole camera model is used. Besides the extrinsic matrices, the intrinsic camera parameters can be optimized at the same time. Due to imperfections from the feature description and matching steps, not all feature points are considered for optimization, but a subset obtained from a RANSAC[2] selection. Further graph optimization methods like bundle-adjustment or additional sensor data (e.g. GNSS[3]) are commonly employed to improve the pose estimation accuracy, overall consistency and mitigate camera pose drift over long distances. The result of a camera pose optimization step is illustrated in figure 2.3.

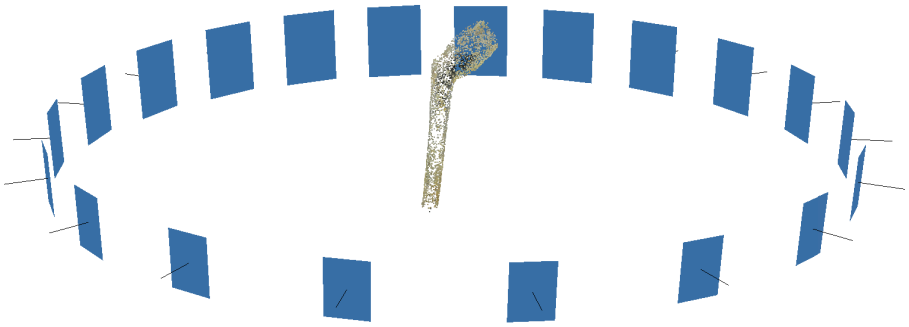


Figure 2.3: Camera pose estimation using the align step of Agisoft Photoscan [Agisoft, 2017] with the highest accuracy settings on the example of a meerschaum pipe. All of the 20 circular arranged camera poses could successfully be estimated. The sparse object feature point cloud consists of 5116 points.

Acknowledgement: The meerschaum pipe was generously provided by the Reiss-Engelhorn-Museen Mannheim for the ArchEye Automatics project ([Seitz and Altenbach, 2011], [Seitz, 2012]).

Dense Point Cloud Computation

Given the estimated camera poses, a dense point cloud is computed, e.g. by estimating depth for all individual pixels of all images with epipolar geometry. This step is visualized in figure 2.4a.

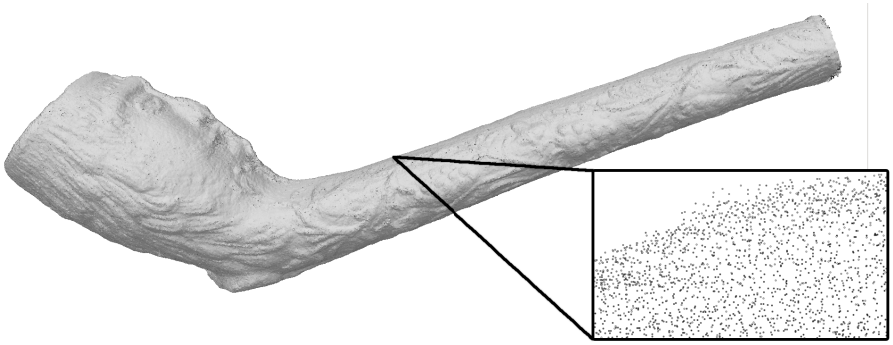
Surface reconstruction

Surface meshes are estimated from the dense point cloud. Popular methods to achieve this are Poisson surface reconstruction or radial basis function approaches. An overview of common surface reconstruction methods from point cloud data is given in [Berger et al., 2014]. This step is visualized in figure 2.4b.

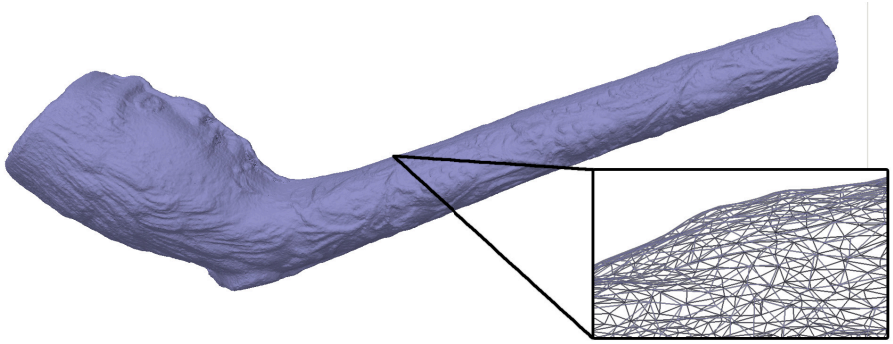
Texture mapping

After computing the 3D mesh, the texture can be added by projecting all images from their respective viewpoint onto the mesh. A 2D texture is obtained by parameterizing the surfaces and blending all projected photos. An example of texture mapping for SfM is given in figure 2.4c.

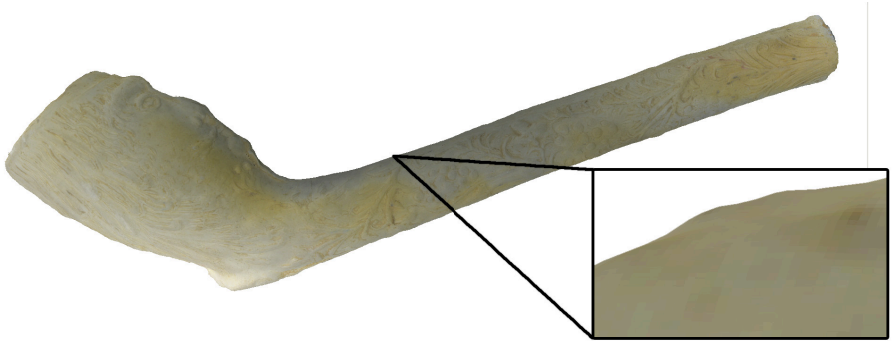
[2]Random Sample Consensus.
[3]Global Navigation Satellite System.



(a) Dense point cloud.



(b) Surface mesh.



(c) Textured mesh.

Figure 2.4: Full reconstruction of the meerschaum pipe using the entire Agisoft Photoscan [Agisoft, 2017] SfM pipeline on the highest settings. A total amount of 202 images with a resolution of 4912×7360 pixels taken at grid-points of two geodesic domes with different radii were considered for the reconstruction. The dense point cloud comprises 6 048 155 points, while the mesh contains 3 486 117 faces and 1 930 199 vertices. Despite the relatively small scale of the pipe (length of 8.3 cm), the object could be reconstructed with very high geometric and visual fidelity. The zoomed-in section describes a part of the model magnified by a factor of 160.

Acknowledgement: The meerschaum pipe was generously provided by the Reiss-Engelhorn-Museen Mannheim for the ArchEye Automatics project ([Seitz and Altenbach, 2011], [Seitz, 2012]).

Optimal Experimental Design 3

Optimal experimental design (OED, e.g. [Körkel, 2002]) describes the process of planning experiments, such that subsequent measurements are estimated to be optimal in some sense. In sections 6.3 and 12, the NBV planning is designed to result in sensor poses that contribute the most information to a subsequent reconstruction. They can therefore be interpreted as OED problems, which will be briefly explained in this section.

Consider an experiment that is used to estimate an unknown model parameter. The obtained measurements are subject to errors and are therefore modeled as random variables. Then an optimal parameter is usually chosen such that the likelihood of the observed data is maximized. Hence, the parameters also allow for a description as random variables. OED addresses the reverse question of how experiments must be conducted to optimally estimate model parameters, i.e. how their uncertainty can be minimized. This uncertainty is usually expressed in terms of covariance matrices that describe the spread of the random variable. Some common ways to formulate this quantity, together with their respective geometric interpretation, are given in section 3.2. As all of them depend on covariance information, an intuitive understanding of confidence regions, i.e. the interpretation of covariance matrices as ellipses and ellipsoids is required first, which is given in section 3.1. We stress the importance of this interpretation as it is later used to construct covariance updates for the NBV planning problem in part II of this thesis.

3.1 Confidence Regions

A confidence region quantifies the likeliness of a random variable being contained in a given area. Usually, the inverse question is more interesting: to determine a region which contains a random variable with a given probability. In the case of N-dimensional multivariate normal random variables, these regions often come in the shape of ellipsoids[4] centered on the mean. In order to motivate those shapes, we first need to understand the relation between the covariance matrix and ellipsoids.

Consider the N-dimensional multivariate random variable $\mathcal{X} \sim \mathcal{N}(\mu, \Sigma)$, with mean $\mu \in \mathbb{R}^N$ and symmetric, positive definite covariance matrix $\Sigma \in \mathbb{R}^{N \times N}$. Then

$$E = \left\{ x \in \mathbb{R}^N \;\middle|\; \frac{1}{\sqrt{(2\pi)^N \det \Sigma}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu) \right) = l \right\} \tag{3.1}$$

[4]More precisely, for $N = 1$ the shape is a line segment, for $N = 2$ an ellipse, for $N = 3$ an ellipsoid and for $N > 3$ a hyper-ellipsoid. For simplicity we will refer to all of those shapes as N-dimensional ellipsoids.

describes a level-set of the probability density function for fixed level $l \in \mathbb{R}$. This is equivalent to

$$E = \left\{ x \in \mathbb{R}^N \;\middle|\; (x-\mu)^T \Sigma^{-1} (x-\mu) = \underbrace{-2 \log\left(l \cdot \sqrt{(2\pi)^N \det \Sigma} \right)}_{=:\tilde{l}} \right\}, \tag{3.2}$$

which is the level set of an affine transformation of an N-sphere centered at μ, at level $\tilde{l} \in \mathbb{R}$, i.e. an N-dimensional ellipsoid. This can be realized by identifying the quadric in equation (3.2) together with the symmetry and positive definiteness of Σ^{-1}. A visualization of this quadric in the two-dimensional case is given in figure 3.1.

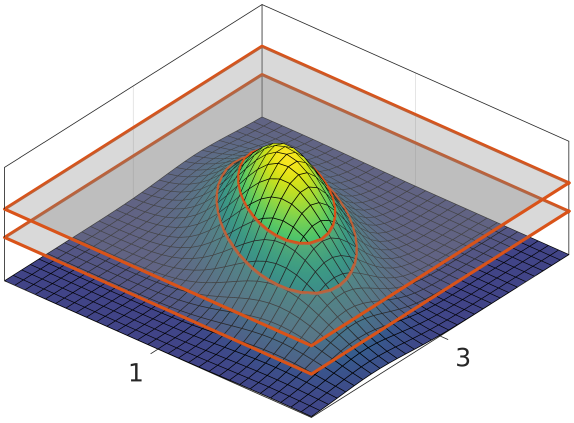


Figure 3.1: Visualization of two level-set ellipses for the bivariate normal distribution probability density function with mean $\mu = \begin{bmatrix} 1 & 3 \end{bmatrix}^T$ and covariance matrix $\Sigma = \begin{bmatrix} 6 & -3 \\ -3 & 4 \end{bmatrix}$.

To further analyze the properties of said shapes, consider the spectral decomposition $\Sigma = U \Lambda U^T$, where $\Lambda \in \mathbb{R}^{N \times N}$ is a diagonal matrix containing the eigenvalues $\{\lambda_i\}_{i=1}^N$ of Σ which correspond to the eigenvectors $\{v_i\}_{i=1}^N$ given by the columns of the unitary matrix $U \in \mathbb{R}^{N \times N}$. Then $\Sigma^{-1} = U \Lambda^{-1} U^T$ and

$$E = \left\{ x \in \mathbb{R}^N \;\middle|\; \left\| \Lambda^{-\frac{1}{2}} U^T (x-\mu) \right\|_2^2 = \tilde{l} \right\}. \tag{3.3}$$

Note that $\Lambda^{-\frac{1}{2}}$ is still a diagonal matrix containing the inverse square roots of the eigenvalues as its diagonal, which are still larger than zero due to the positive definiteness of Σ. Substituting $y = \Lambda^{-\frac{1}{2}} U^T (x-\mu)$ we obtain

$$E = \left\{ x = \mu + U \Lambda^{\frac{1}{2}} y \;\middle|\; \|y\|_2 = \sqrt{\tilde{l}},\ y \in \mathbb{R}^N \right\}. \tag{3.4}$$

Equation (3.4) reveals the geometric shape of E. The set

$$\left\{ y \in \mathbb{R}^N \;\middle|\; \|y\|_2 = \sqrt{\tilde{l}} \right\} \tag{3.5}$$

describes an N-sphere with radius $\sqrt{\tilde{l}}$. In (3.4) this N-sphere is scaled along the main axes (i.e. the i-th main axis direction is scaled by $\sqrt{\lambda_i}$), rotated by U and finally translated by μ. Hence, the half-axes of the resulting ellipsoid can be identified. The i-th half-axis has direction v_i and length $\sqrt{\lambda_i \tilde{l}}$.

By definition, the confidence region corresponding to a given confidence $p_{conf} \in [0,1]$ is not unique. Uniqueness can be achieved by choosing the region with highest

probability density, such that it contains all most-likely sub-regions. This property results in the smallest possible region size and corresponds to a set of points that are bound by the previously explained ellipsoidal level-sets. In order to determine a confidence region with this property, we need to find a level $Q_N(p_{conf})$ such that

$$P\left((\mathcal{X}-\mu)^T\Sigma^{-1}(\mathcal{X}-\mu) \leq Q_N(p_{conf})\right) = p_{conf} \tag{3.6}$$

holds. By substituting

$$\mathcal{Y} = \begin{bmatrix}\mathcal{Y}_1 & \dots & \mathcal{Y}_N\end{bmatrix}^T := \Lambda^{-\frac{1}{2}}U^T(\mathcal{X}-\mu) \sim \mathcal{N}(0, I_N), \tag{3.7}$$

equation (3.6) reduces to

$$P\left(\sum_{i=1}^{N}\mathcal{Y}_i^2 \leq Q_N(p_{conf})\right) = p_{conf}. \tag{3.8}$$

Since $\{\mathcal{Y}_i\}_{i=1}^N$ are independent standard normal random variables, $\sum_{i=1}^N \mathcal{Y}_i^2$ is chi-square distributed, i.e.

$$\sum_{i=1}^{N}\mathcal{Y}_i^2 \sim \chi^2(N). \tag{3.9}$$

Given this insight, $Q_N(p_{conf})$ can be identified as the quantile function[5] of the chi-square distribution. While no analytic formulation for this function is known, it is well studied and can be evaluated using lookup tables. Some values of $Q_N(p_{conf})$ for different N and p_{conf} are given in table 3.1. As explained earlier in this section, the half-axes of the ellipsoids can be stated explicitly. Here, the i-th half-axis with direction v_i has length $\sqrt{\lambda_i Q_N(p_{conf})}$.

Table 3.1: Values of the quantile function $Q_N(p_{conf})$ of the chi-square distribution for various N and p_{conf}.

N \ p_{conf}	**0.01**	**0.05**	**0.1**	**0.2**	**0.8**	**0.9**	**0.95**	**0.99**
1	0.0002	0.0039	0.0158	0.0642	1.6424	2.7055	3.8415	6.6349
2	0.0201	0.1026	0.2107	0.4463	3.2189	4.6052	5.9915	9.2103
3	0.1148	0.3518	0.5844	1.0052	4.6416	6.2514	7.8147	11.3449

We only analyzed confidence regions for multivariate random variables. For other distributions, the ellipsoidal shapes are still used to approximate maximum probability density confidence regions. Multivariate random variables are uniquely characterized by their mean and second central moment (covariance matrix). As the mean only acts as a shift for the confidence regions, the ellipsoid shape is determined solely by the covariance information. The central moments of any probability distribution function – if they exist – can be interpreted somewhat sloppily formulated as the Taylor expansion of the density function around the mean. Geometrically the

[5] Also known as *inverse cumulative distribution function*.

second central moment (covariance matrix) gives information about the "spread" of a random variable. The third central moment contains the skewness, and the fourth central moment relates to the kurtosis. Hence, confidence ellipses can be interpreted as a linearization of this expansion, giving us insight over the size and shape of the real confidence regions.

3.1.1 Degenerate Cases

In the previous section, we required Σ to be positive definite. However, also positive semi-definite covariance matrices can occur. In this case, Σ has eigenvalues equal to zero which prohibits its inversion. However, similar results can still be obtained by projecting the matrix into a lower-dimensional and strictly positive definite subspace. This is easily achieved by removing all eigenvalues with $\lambda_i = 0$ and their corresponding eigenvectors from the singular value decomposition as they correspond to zero variance directions in the distribution. The resulting matrix is again strictly positive definite, allowing us to apply results from the previous section. Those can now be generalized for positive semi-definite matrices. The length of the i-th half axis with direction v_i in N dimensions is then given by $\sqrt{\lambda_i Q_{N-N_0}(p_{conf})}$, where N_0 denotes the dimension of the null-space of Σ.

Another special case arises if the precision matrix Σ^{-1} has eigenvalues equal to zero. In terms of confidence ellipsoids, this corresponds to an infinite uncertainty in the direction of the corresponding eigenvector. This comes to no surprise given that $\{1/\lambda_i\}_{i=1}^{N}$ are the eigenvalues of Σ^{-1}. Precision matrices with zero eigenvalues will play an important role later on in chapter 6 and are elaborated in more detail in section 6.1.

3.1.2 Misconceptions

Confidence regions describe areas given the distribution of a random variable. If this random variable was observed a sufficient number of times, the observation would be inside the confidence region with a probability of p_{conf}. Now consider a parameter that is being measured a finite number of times given some measurement error. Then mean and covariance can be approximated using an estimator, hence, also yielding confidence ellipsoids. However, as the parameter is an unknown constant, it never necessarily lies inside any of those regions – which is the general misconception about confidence ellipsoids. In those cases, the confidence probability relates to the quality of the estimator and not the quality of the measurement.

3.2 Optimality Criteria

Consider an experiment that is conducted in order to measure design variables x (parameters, functions) that control the experimental setup. Given a distribution for the measurement error, let the covariance matrix of the design variable estimator be given by $\Sigma(x)$. Then the minimization problem

$$\min_x \Phi(\Sigma(x)), \tag{3.10}$$

where $\Phi : \Sigma(x) \to \mathbb{R}$ is a function corresponding to some statistical criterion, is considered an optimal experimental design problem.

For simplicity, we will drop the x from this notation. The criterion can often be related to the eigenvalues of Σ. From the previous sections we know that the half-axes of confidence ellipsoids are proportional to the square root of their corresponding eigenvalues, since $Q_N(p_{conf})$ is constant for fixed p_{conf}. Hence, we can give geometric interpretations for some commonly used optimality criteria:

- **A-Criterion:** $\Phi(\Sigma) = \frac{1}{N} \operatorname{tr} \Sigma$
  As the trace of a matrix is equal to the sum of its eigenvalues, minimizing the A-Criterion corresponds to minimizing the length of the diagonal of a hyperrectangle with side lengths given by the confidence ellipsoids half-axis length.

- **E-Criterion:** $\Phi(\Sigma) = \|\Sigma\|_2$
  The two-norm of a matrix is equivalent to its largest singular value. Since singular values are equivalent to eigenvalues for symmetric positive definite matrices, the E-Criterion corresponds to the largest half-axis length squared.

- **D-Criterion:** $\Phi(\Sigma) = (\det \Sigma)^{\frac{1}{N}}$
  The determinant of a matrix is equivalent to the product of its eigenvalues. As the volume of an ellipsoid is equivalent to the volume of an N-sphere scaled by the product of all half-axes lengths, minimizing the D-Criterion is equivalent to minimizing the volume of any confidence ellipsoid. In the degenerate case, the D-Criterion can be formulated as the product of all non-zero eigenvalues.

Occupancy Grid Maps

4

The NBV planning method proposed in part II of this thesis requires a discrete representation of the surface geometry (see section 7.1). This can be achieved with voxels that classify free and occupied space. These voxel maps allow for a memory and runtime efficient spatial representation, which is easily integrable into an autonomous exploration framework using occupancy mapping. A brief introduction to occupancy grid mapping using an inverse sensor model alongside a short discussion regarding shortcomings and alternative mapping methods is given in the first part of this chapter. Voxel grid maps are usually implemented using n-tree data structures, due to their good memory efficiency. In three dimensions these are also called octrees. We will make extensive use of this data structure in our proposed NBV planning approach using a voxel grid surface geometry representation. Details on efficient implementations of required voxel map operations, such as ray casting, are presented in section 8. An introduction to octrees is given in the second part of this chapter.

Occupancy grid maps have become one of the most prominent tools in mobile robotics for tasks such as path planning, navigation, and collision avoidance. They refer to a family of algorithms that give a spatial representation of a robotic environment. Space – usually two- or three-dimensional – is discretized into grid cells, each equipped with information on its occupancy (e.g. occupancy probability). In mobile robotics, creating those maps goes hand in hand with the additional task of localizing the robot. Hence, it is not only required to accommodate for sensor measurement errors, but additionally localization errors when integrating new measurements into the grid map. The task of simultaneous localizing the robot and building a map of the environment is often referred to as simultaneous localization and mapping (SLAM). However, as stated in the introduction, SLAM is beyond the scope of this thesis. Some examples of two- and three-dimensional occupancy grid maps are given in figure 4.1.

4.1 Occupancy Grid Mapping using Inverse Sensor Model

The theory of occupancy grid mapping was developed in the mid-80s and first introduced by [Moravec and Elfes, 1985], who mapped an environment with an autonomous robot using a wide-angle sonar. This approach is still one of the most popular ones for discrete spatial representations due to its simple implementation, fast measurement updates, and good scalability properties.

Consider a sensor that is able to measure depth. Examples include sonar, LiDAR

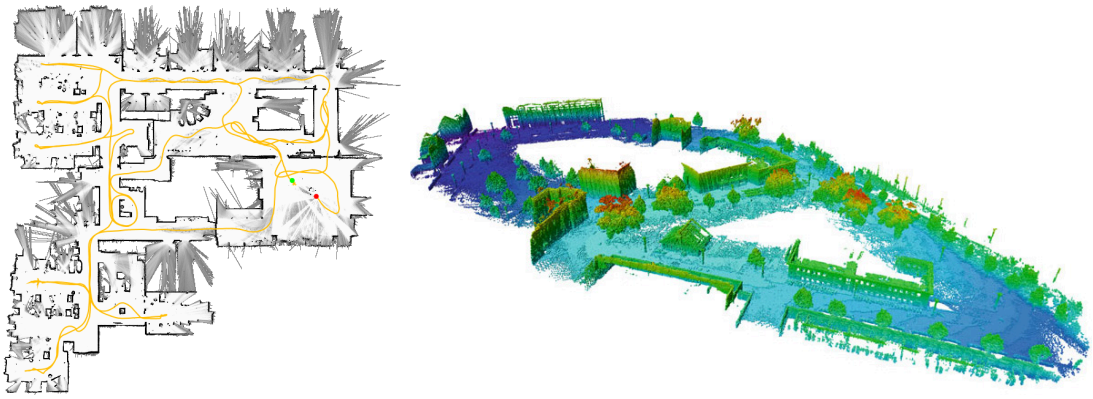


Figure 4.1: Left: 2D occupancy grid map generated using Revo LDS sensor data. Right: 3D occupancy grid map of the Freiburg campus dataset. The scene has a total size of $292 \times 167 \times 28$ m^3 at a voxel resolution of 0.2 m. Colors correspond to height.
Source: Left: [Hess et al., 2016] ©2016 IEEE, Right: Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer [Hornung et al., 2013] ©2013.

or stereo-camera setups. Let $z_{1:T} = \{z_1, \dots, z_T\}$ denote the set of measurements along with the corresponding sensor poses. The measurement poses are assumed to be known. We further assume that the area that is observed is not subject to change during the mapping process. Space is divided into equally sized cells, indexed by an index set I. Each grid cell corresponds to a binary random variable that models the occupancy of cell $i \in I$. We indicate the probability that cell i is occupied with $p(m_i)$, while $p(\neg m_i)$ denotes the probability of cell i being free. As the number of grid cells rapidly grows with the granularity of the discretization, estimating the posterior $p(\{m_i\}_{i \in I} \mid z_{1:T})$ for the joint map is infeasible due to the high dimensionality. Hence, by assuming independence between all cells, the problem is decomposed into many one-dimensional problems instead. Applying a binary Bayes filter for static state estimation, the occupancy probabilities is given by

$$\begin{aligned} p(m_i|z_{1:T}) &\overset{\text{Bayes' theorem}}{=} \frac{p(z_T|m_i, z_{1:T-1}) \cdot p(m_i|z_{1:T-1})}{p(z_T|z_{1:T-1})} \\ &\overset{\text{Markov assumption}}{=} \frac{p(z_T|m_i) \cdot p(m_i|z_{1:T-1})}{p(z_T|z_{1:T-1})} \\ &\overset{\text{Bayes' theorem}}{=} \frac{p(m_i|z_T) \cdot p(z_T)}{p(m_i)} \cdot \frac{p(m_i|z_{1:T-1})}{p(z_T|z_{1:T-1})}. \end{aligned} \tag{4.1}$$

Note that the Markov assumption $p(z_T|m_i, z_{1:T-1}) = p(z_T|m_i)$ is used for the second identity. It states that sensor readings are assumed to be conditionally independent given knowledge of the cell's occupancy. In order to obtain a representation that enables more computational efficient updates, the log-odds formulation is employed. It is given by

$$p(x) \mapsto \log\left(\frac{p(x)}{1 - p(x)}\right) =: l(x). \tag{4.2}$$

This mapping is bijective, hence the occupancy probability can also be calculated

from the log-odds ratio with

$$l(x) \mapsto 1 - \frac{1}{1 + \exp(l(x))} = p(x). \tag{4.3}$$

In analogy to (4.1), the posterior probability of a cell being free is obtained as

$$1 - p(m_i|z_{1:T}) = p(\neg m_i|z_{1:T}) = \frac{p(\neg m_i|z_T) \cdot p(z_T)}{p(\neg m_i)} \cdot \frac{p(\neg m_i|z_{1:T-1})}{p(z_T|z_{1:T-1})}. \tag{4.4}$$

Hence, the log-odds ratio is given as

$$\begin{array}{rcccccc} \dfrac{p(m_i|z_{1:T})}{p(\neg m_i|z_{1:T})} & = & \dfrac{p(m_i|z_T)}{1-p(m_i|z_T)} & \cdot & \dfrac{p(m_i|z_{1:T-1})}{1-p(m_i|z_{1:T-1})} & \cdot & \dfrac{1-p(m_i)}{p(m_i)} \\ \Longrightarrow l(m_i|z_{1:T}) & = & l(m_i|z_T) & + & l(m_i|z_{1:T-1}) & - & l(m_i) \\ & & \text{inverse sensor model} & & \text{recursive term} & & \text{prior} \end{array} \tag{4.5}$$

This ratio can therefore easily be calculated incrementally from previous occupancy estimates. The inverse sensor model describes the certainty of occupancy along a measurement ray. For example, consider a single depth measurement. Then all cells traversed by this ray are updated with some value $p(m_i|z_T) < 0.5$, while the grid cell containing the ray endpoint is updated with $p(m_i|z_T) > 0.5$. The prior is commonly chosen as a constant value $p(m_i) \in [0.2, 0.5]$.

This mapping approach, however, suffers from some shortcomings even with noise-free data. Assigning a single binary random variable to each cell implies that each cell is either completely free or occupied. Furthermore, the assumption of mutually independent grid cells, and in consequence the utilized Markov assumption, does not hold in reality. Consider a set of depth measurements. Then some rays pass through cells that may also contain ray-endpoints. This results in uncertainty for partially occupied cells. In practical application this approach nevertheless generated satisfactory spatial representations.

There are other methods that do not suffer from this shortcoming. For example, methods involving reproducing kernels (e.g. [O'callaghan and Ramos, 2012] [Ramos and Ott, 2016]) have attracted attention in the last few years. They predict occupancy values through a learned regression classifier, thus allowing for theoretically infinite detail without the need for discretization. For navigation purposes, these maps are usually also discretized afterward. These learning approaches, however, entail a higher computational complexity, which is undesirable for our purposes.

4.2 Efficient Octree Implementation

An octree is a hierarchical data structure, where each node is subdivided into eight equally sized child nodes. Although more abstract scenarios are also possible, they can intuitively be interpreted as a spatial subdivision of three-dimensional space. The 2D equivalent of an octree is called quadtree due to subdivision in four equally sized child nodes.

Two popular octree implementations that are available as open-source C++ libraries supporting occupancy mapping are given by OctoMap [Hornung et al., 2013]

and GPU-Voxels [Hermann et al., 2014]. The latter provides a fast CUDA implementation, requiring a NVIDIA graphics card, while OctoMap runs on the CPU. In mobile robotics, graphics cards are not always viable due to power consumption and weight restrictions. For this reason, OctoMap is more widely used and enjoys greater popularity in the scientific community. We will now give an overview of some common implementation details of octrees for occupancy grid mapping. Note that while these apply to other implementations as well, we focus on OctoMap, since this library is used and also improved upon in some of our algorithms (see section 8).

In OctoMap octrees have a fixed depth of $d = 16$, with occupancy information being stored in the grid defined by the lowest tree level in terms of log-odd values. Nodes at this level are commonly referred to as leaves. Given a leaf-voxel resolution of r meter, a cube with side length $r \cdot 2^d$ m can be covered by the octree. Due to the tree structure, a random leaf lookup has complexity $\mathcal{O}(d)$. Using a log-odds threshold, occupancy information can be classified into the two discrete states *free* and *occupied*. A voxel occupancy update is realized by

$$l(m_i|z_{1:T})) = \max\left(\min\left(l(m_i|z_{1:T-1}) + l(m_i|z_T), l_{\max}\right), l_{\min}\right), \tag{4.6}$$

with some constants $l_{\min}$ and $l_{\max}$, i.e. a clamped version of equation (4.5). This clamping ensures the adaptability to temporary or permanent changes in the map. Furthermore, if a voxel reaches one of those clamping parameters, it is assumed to be stable. This is an important property as it allows for pruning. Instead of maintaining eight stable children with identical information, their memory can be freed after passing this occupancy value to their common parent node, optimizing memory usage. This is especially important for voxels observed as free. Occupied voxels usually represent a two-dimensional manifold defined by surface observations, while free voxels are characterized by large voxel volumes that would dominate memory usage if left un-pruned. Details regarding the inverse sensor model and parameters used can be found in [Hornung et al., 2013].

By only initializing voxels once they have been observed at least once (free or occupied), memory usage is further improved. This has the additional benefit that not only two discrete states (free and occupied) can be defined, but additionally *unknown* if a voxel is never observed. A visualization of this OctoMap data structure is given in figure 4.2.

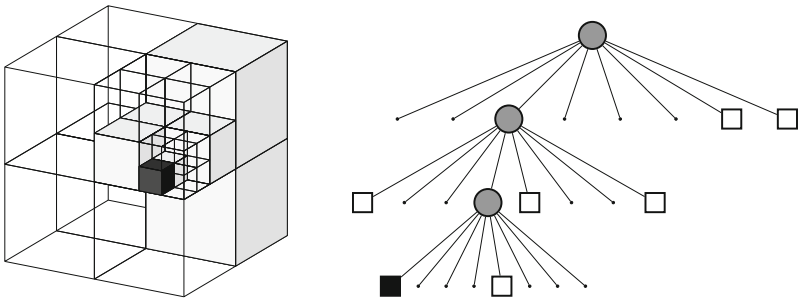


Figure 4.2: Visualization of the OctoMap octree data structure (left: volumetric, right: tree). Opaque white squares correspond to free voxels, the black one to an occupied voxel. All other voxels are unknown.

Source: Reprinted by permission from Springer Nature Customer Service Centre GmbH: Springer [Hornung et al., 2013] ©2013.

Furthermore, [Hornung et al., 2013] implemented methods to counteract the shortcomings described in the previous section, i.e. the occupancy uncertainty for voxels being observed both as free or occupied. Instead of inserting each measurement ray consecutively, a full sweep from a single sensor pose is processed simultaneously. Two disjoint sets of free and occupied voxels are created from these measurements. If a voxel is contained in both sets, it is discarded from the one containing the free voxels. Afterward, each voxel of these sets is updated exactly once accordingly. This way, cancellation errors are mitigated.

In OctoMap, occupancy information is propagated to all parent nodes as the maximum of all of its children's occupancy. Then a coarse approximation of the occupancy map is given on other levels of the octree, which can later be exploited in our NBV algorithms (see section 8). This especially includes distance computations and the iteration of voxel subsets (section 8.3.1), as well as ray traversal operations (section 8.3). In section 8.3.3 we will further take advantage of this property to allow for updates of free and occupied voxels on different octree levels.

Gaussian Process Fundamentals

5

In part III of this thesis, the surface of a model is estimated as an implicit function of the posterior mean of a Gaussian process. Therefore, we give a brief introduction to Gaussian processes as well as some of their properties in this chapter. For the most part, we will follow [Rasmussen and Williams, 2006, chapters 2,4], which is also recommended for further details on the topic. Besides a formal definition, especially their behavior under linear operator transformations is highlighted. We will return to this topic in section 11.3, where surface normal measurement observations are included in the Gaussian process.

Gaussian processes are used to mathematically model the behavior of nondeterministic systems from observations. This Bayesian machine learning approach is often described as a distribution over functions. Although widely used in the context of classification and signal analysis, we will be particularly interested in its application for interpolation, extrapolation, and smoothing of discrete measurement points in this work. In contrast to other machine learning methods, such as neural networks, Gaussian processes are derived from statistical quantities, using linear algebra and probability theory. Hence, the entire mathematical process remains very transparent and controllable. In addition, the method provides associated variance information for each output value describing its predictive uncertainty.

5.1 Formal Definition

Definition 5.1 (Gaussian process)**.** Let $\mathcal{X}$ be an arbitrary index set. A stochastic process $(f(x))_{x\in\mathcal{X}}$ is called *Gaussian process* if all of its finite subsets have a joint Gaussian distribution.

Since a Gaussian multivariate random variable is uniquely characterized by its mean and covariance matrix, a Gaussian process, written as $f \sim \mathcal{GP}(m, k)$, is uniquely defined by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a kernel function (also called covariance function) $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with

$$\begin{aligned} m(x) &:= \mathbb{E}[f(x)] \\ k(x, x') &:= \mathbb{E}\left[(f(x) - m(x))\,(f(x') - m(x'))^T\right] = \operatorname{cov}(f(x), f(x')). \end{aligned} \tag{5.1}$$

Covariance matrices must be positive definite, hence, the kernel function must also be positive definite, i.e.

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j k(x_i, x_j) \geq 0, \quad \forall n \in \mathbb{N}, x_i \in \mathcal{X}, c_i \in \mathbb{R}. \tag{5.2}$$

Two classes of special covariance functions can be defined:

Definition 5.2 (stationary, isotropy)**.** Let $\mathcal{X}$ be a vector space. If the kernel function is a function of $x - x'$ it is called *stationary*. If additionally $\mathcal{X}$ is a normed vector space and the kernel is a function of $\|x - x'\|$, it is called an *isotropic* kernel.

Geometrically speaking, for example, if $\mathcal{X}$ refers to an Euclidean space, stationary kernels are invariant to translations, while isotropic kernels are invariant to all rigid motions.

A Gaussian random field is the $d \in \mathbb{N}$ dimensional generalization of a Gaussian process, i.e. we allow for $m : \mathcal{X} \to \mathbb{R}^d$ and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{d \times d}$ with $d \in \mathbb{N}$. However, by simply augmenting $\tilde{X} := \mathcal{X} \cup \{1, \ldots, d\}$ and defining

$$\begin{aligned} \tilde{m} &: \quad \tilde{X} \to \mathbb{R}, \qquad \tilde{m}(\{x, i\}) := m(x)_i \\ \tilde{k} &: \tilde{X} \times \tilde{X} \to \mathbb{R}, \quad \tilde{k}(\{x, i\}, \{x', j\}) := k(x, x')_{i,j}, \end{aligned} \tag{5.3}$$

with the subscripts corresponding to rows (and columns), we can transform the Gaussian random field to the Gaussian process $\tilde{f} \sim \mathcal{GP}(\tilde{m}, \tilde{k})$. Hence, we will also allow for d-dimensional mean and $d \times d$ dimensional kernel functions, but due to the reasoning given above, refer to them as d-dimensional Gaussian processes.

For ease of notation we introduce ordered sets of variables, i.e.

$$\begin{aligned} X &:= \{x_i \in \mathcal{X}\}_{i=1}^{N}, \quad \text{training points} \\ X^* &:= \{x_i^* \in \mathcal{X}\}_{i=1}^{M}, \quad \text{test points.} \end{aligned} \tag{5.4}$$

From now on the asterisk superscript will indicate that a quantity is related to test input or – in absence – training input. We further allow functions to be evaluated with those sets of variables as arguments, yielding (block) matrix and vector expressions, e.g.

$$k(X, X^*) = \begin{bmatrix} k(x_1, x_1^*) & \cdots & k(x_1, x_M^*) \\ \vdots & \ddots & \vdots \\ k(x_N, x_1^*) & \cdots & k(x_N, x_M^*) \end{bmatrix}, \qquad m(X) = \begin{bmatrix} m(x_1) \\ \vdots \\ m(x_N) \end{bmatrix}. \tag{5.5}$$

With this new notation we can give a definition for d-dimensional Gaussian process that is equivalent to definition 5.1 in the one-dimensional case:

Definition 5.3 (d-dimensional Gaussian process)**.** Let $m : \mathcal{X} \to \mathbb{R}^d$ and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{d \times d}$ with $d \in \mathbb{N}$. Furthermore, let $k(X, X)$ be positive semi-definite for all finite subsets $X \subset \mathcal{X}$. Then $f \sim \mathcal{GP}(m, k)$ is a d-dimensional Gaussian process and $f(X) \sim \mathcal{N}(m(X), k(X, X))$ holds.

Due to the relation of Gaussian processes to normal distributions given in definition 5.3, posterior distributions for Gaussian processes can be derived. Let

$$f_{1:N} := f(X) + \epsilon = [f(x_1), \ldots, f(x_N)]^T + \epsilon, \quad \epsilon \sim \mathcal{N}(0, I_N \cdot \sigma^2) \tag{5.6}$$

be a set of observations disturbed by measurement errors ϵ, observed at the training points X. Assume the Gaussian process is to be evaluated at all test points X^*, i.e. we are interested in

$$f^*_{1:M} := f(X^*) = [f(x^*_1), \ldots, f(x^*_M)]^T . \tag{5.7}$$

The joint prior distribution of training and test data is given by

$$\begin{bmatrix} f_{1:N} \\ f^*_{1:M} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(X) \\ m(X^*) \end{bmatrix}, \begin{bmatrix} k(X,X) + \sigma^2 I & k(X,X^*) \\ K(X^*,X) & K(X^*,X^*) \end{bmatrix} \right). \tag{5.8}$$

Using the multivariate conditional distribution, the posterior distribution is given as

$$f^*_{1:M} | X, f_{1:N}, X^* \sim \mathcal{N}\left(m_{\text{post}}(X^*), k_{\text{post}}(X^*)\right) \quad \text{with} \tag{5.9}$$

$$m_{\text{post}}(X^*) = m(X^*) + k(X^*,X)\left[k(X,X) + \sigma^2 I\right]^{-1}(f_{1:N} - m(X)) \tag{5.10}$$

$$k_{\text{post}}(X^*,X^*) = k(X^*,X^*) - k(X^*,X)\left[k(X,X) + \sigma^2 I\right]^{-1} k(X,X^*). \tag{5.11}$$

The posterior kernel consists of a prior minus a term representing the information gained by the observations at X. As posterior mean (5.10) and covariance functions (5.11) are functions in X^* and k_{post} is positive semi-definite for all possible inputs if $\sigma > 0$, they define the posterior Gaussian process

$$f^* \sim \mathcal{GP}\left(m_{\text{post}}, k_{\text{post}}\right). \tag{5.12}$$

Note that the requirement $\sigma > 0$ is only necessary for special cases of positive semi-definite kernels, as $[k(X,X) + \sigma^2 I]$ may not be invertible otherwise. Also note that large portions of (5.10) and (5.11) can be precomputed given measurement data X and $f_{1:N}$. Then the evaluation of the posterior mean at a single training point is a simple vector matrix multiplication together with an addition of total computational complexity $\mathcal{O}(N)$. The posterior variance of a single point is then similarly obtainable in $\mathcal{O}(N^2)$.

5.2 Closure under Evaluation of Linear Operators

Gaussian processes are closed under the evaluation of linear functionals:

Theorem 5.1. Let $f \sim \mathcal{GP}(m,k)$ be a Gaussian process and $\mathcal{L}$ be a bounded linear operator operating on f. If $\mathcal{L}m$ and $\mathcal{L}^1(\mathcal{L}^2 k)^T$ exist and are well defined, then $\mathcal{L}f \sim \mathcal{GP}\left(\mathcal{L}m, \mathcal{L}^1(\mathcal{L}^2 k)^T\right)$. The notation $\mathcal{L}^i$ here means that the operator $\mathcal{L}$ acts on a function, where all but the i-th argument are fixed, i.e. $\mathcal{L}^1 k = \mathcal{L}k(\cdot, x')$ and $\mathcal{L}^2 k = \mathcal{L}k(x, \cdot)$.

Proof. From (5.1) we obtain $\mathcal{L}f \sim \mathcal{GP}\left(m_{\mathcal{L}}, k_{\mathcal{L}}\right)$ with

$$\begin{aligned} m_{\mathcal{L}}(x) &= \mathbb{E}\left[\mathcal{L}f(x)\right] \\ k_{\mathcal{L}}(x,x') &= \mathbb{E}\left[\left(\mathcal{L}f(x) - \mathbb{E}\left[\mathcal{L}f(x)\right]\right)\left(\mathcal{L}f(x') - \mathbb{E}\left[\mathcal{L}f(x')\right]\right)^T\right] \end{aligned}$$

Hence, in order to conclude the proof it suffices to show that $\mathbb{E}\left[\mathcal{L}f(x)\right] = \mathcal{L}\mathbb{E}\left[f(x)\right]$. This is shown in [Papoulis et al., 2002, chapter 10]. ■

As differentiation is a linear operation, theorem 5.1 implies that the Gaussian process $\partial f/\partial x_i$ exists if $\partial m(x)/\partial x_i$ and $\partial^2 k(x,x')/\partial x_i \partial x_i'$ exist and are well defined. Similar statements can be formulated for higher-dimensional functions, higher order derivatives or integration operators. Theorem 5.1 is extremely powerful, as it allows for some interesting applications:

- Let $\mathcal{L} = \begin{bmatrix}\mathcal{L}_1 & \cdots & \mathcal{L}_{n_\mathcal{L}}\end{bmatrix}^T$ be a linear operator with $n_\mathcal{L} \in \mathbb{N}$. Theorem 5.1 allows us to include observations $\mathcal{L}_j f$ with $j = 1, \ldots, n_\mathcal{L}$ in the joint prior distribution (5.8). This results in a different posterior from the conditional distribution (5.9), that also considers these new types of measurements. Note that especially the number of observations, as well as the observation points, may be different for each $\mathcal{L}_j$. One example application is given in section 11.3 where we will include derivative observations to a Gaussian process. Another example using zeroth, first and second order derivative observations to compute Bayesian quadratures is given in [Wu et al., 2017].

- Similarly, the posterior distribution can be modified with $\mathcal{L}f^*$ to obtain only relevant informations for the respective problem.

- A Gaussian process can be constrained to linear operators. For example, if we find an operator $\bar{\mathcal{L}}$ s.t. $\mathcal{L}\bar{\mathcal{L}} \equiv 0$ and define $f = \bar{\mathcal{L}}g \sim \mathcal{GP}(\bar{\mathcal{L}}m, \bar{\mathcal{L}}k)$ then $\mathcal{L}f \equiv 0$ holds. This constraint can – for example – describe a differential equation. For further details on Gaussian processes constrained by linear operators, with equality and inequality constraint, see [Agrell, 2019], [Jidling et al., 2017].

5.3 Log Marginal Likelihood

In this section we will briefly elaborate the log marginal likelihood function for a Gaussian process and its use in the context of hyperparameter optimization. From (5.6) follows

$$f_{1:N} \sim \mathcal{N}\left(m(X), k(X,X) + \sigma^2 I\right). \tag{5.13}$$

The log likelihood function, marginalizing over the function values f of the Gaussian process, is then given - analogous to the multivariate Gaussian distribution - as

$$\begin{aligned}\log p(f_{1:N}|X) = &-\frac{1}{2}\operatorname{tr}\left[(f_{1:N} - m(X))^T \left(k(X,X) + \sigma^2 I\right)^{-1} (f_{1:N} - m(X))\right] \\ &-\frac{1}{2}\log\det\left(k(X,X) + \sigma^2 I\right) - \frac{n}{2}\log(2\pi).\end{aligned} \tag{5.14}$$

Assume the mean function m and the kernel function k depend on some unknown parameters p. The optimal parameters $\hat{p}$ that give the highest probability for the observed data are obtained as:

$$\hat{p} = \arg\min_p l(f_{1:N}|X). \tag{5.15}$$

A solution to (5.15) can easily be computed using numerical methods (e.g. SQP method).

5.4 Mean Square Continuity

Later on in chapter 11 we will model implicit surfaces using the posterior mean of a Gaussian process. Since smoothness of those surfaces is a desired property the continuity of the mean is now analyzed.

Definition 5.4 (Mean-Square Continuity)**.** A Gaussian process $f \sim \mathcal{GP}(m, k)$ is called *continuous in mean square* at $\hat{x} \in \mathcal{X}$ if for all sequences $x_1, x_2, \ldots$ in $\mathcal{X}$ with $\|x_i - \hat{x}\|_2 \overset{i\to\infty}{\longrightarrow} 0$ the limit $\lim_{i\to\infty} \mathbb{E}\left[|f(x_i) - f(\hat{x})|^2\right]$ converges to zero.

Mean-Square continuity of a Gaussian process corresponds to pointwise convergence in real analysis. Note that, while it does not imply sample function continuity, it is a necessary condition for continuous sample paths (see [Adler, 2010, chapter 2]). Fortunately, there is a simple criterion to check for mean-square continuity given below:

Theorem 5.2. Let $f \sim \mathcal{GP}(m, k)$ be a Gaussian process with continuous mean m. Then f is mean square continuous at $\hat{x} \in \mathcal{X}$ if and only if $k(x, x')$ is continuous at $x = x' = \hat{x} \in \mathcal{X}$.

Proof. Consider an arbitrary series $x_i \overset{i\to\infty}{\longrightarrow} \hat{x}$ and the expansion

$$\begin{aligned}\mathbb{E}\left[|f(x_i) - f(\hat{x})|^2\right] &= \mathbb{E}\left[f(x_i) - f(\hat{x})\right]^2 + k(x_i, x_i) - 2k(x_i, \hat{x}) + k(\hat{x}, \hat{x}) \\ &= \left(m(x_i) - m(\hat{x})\right)^2 + k(x_i, x_i) - 2k(x_i, \hat{x}) + k(\hat{x}, \hat{x}).\end{aligned} \tag{5.16}$$

I. Let $k(x, x')$ be continuous at $x = x' = \hat{x}$. Then the right part of (5.16) vanishes for $i \to \infty$, hence, proofing mean square continuity of f in $\hat{x}$.

II. Assume mean square continuity of f in $\hat{x}$. Then the limit $i \to \infty$ of (5.16) is given by

$$\begin{aligned} 0 &= \lim_{i\to\infty} k(x_i, x_i) - \lim_{i\to\infty} 2k(x_i, \hat{x}) + k(\hat{x}, \hat{x}) \\ \iff \quad \lim_{i\to\infty} 2k(x_i, \hat{x}) &= \lim_{i\to\infty} k(x_i, x_i) + k(\hat{x}, \hat{x}).\end{aligned} \tag{5.17}$$

Squaring both sides we can apply the Cauchy-Schwarz inequality given by $k(x_i, x_i)k(\hat{x}, \hat{x}) \geq k(x_i, \hat{x})^2$, i.e.

$$\begin{aligned} 4 \lim_{i\to\infty} k(x_i, x_i)k(\hat{x}, \hat{x}) &\geq \left(\lim_{i\to\infty} k(x_i, x_i) + k(\hat{x}, \hat{x})\right)^2 \\ \iff \quad 0 &\geq \left(\lim_{i\to\infty} k(x_i, x_i) - k(\hat{x}, \hat{x})\right)^2.\end{aligned} \tag{5.18}$$

This inequality is satisfied if and only if $\lim_{i\to\infty} k(x_i, x_i) = k(\hat{x}, \hat{x})$, proofing the continuity of $k(x, x')$ in $x = x' = \hat{x}$.

■

Theorem 5.2 further simplifies in the case of stationary or isotropic kernels, as continuity only has to hold for a single point, i.e. $x - x' = 0$.

5.5 Interpretation as Linear Regression Model

A Gaussian process can be transformed into other regression models, each of which highlighting different aspects and properties of the Gaussian process. In this section we will briefly relate them to linear regression models. Let $f \sim \mathcal{GP}(m, k)$ be a one dimensional Gaussian process. Consider the eigenfunction expansion of the kernel given by $k(x, x') = \sum_i \lambda_i \phi_i(x)\phi_i(x')$. Then the Gaussian process is equivalent to the linear regression model $f(x) = \phi(x)^T w + m(x)$ with prior on the weights $w_i \sim \mathcal{N}(0, \lambda_i)$, because

$$\begin{aligned} \mathbb{E}[f(x)] &= \phi(x)^T \mathbb{E}[w] + m(x) = m(x), \\ \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] &= \phi(x)^T \mathbb{E}[ww^T]\phi(x') = k(x, x'). \end{aligned} \tag{5.19}$$

Hence, the eigenfunctions of the kernel project the inputs into a feature space, where linear regression is applied. This connection between non-linearity of the Gaussian process and the corresponding linear regression model is also refereed to as the *kernel trick* and is extensively used in machine learning.

Part II

Structure from Motion
Next-Best-View Planning

Theoretical Derivation

6

In this chapter, we derive the theory for SfM-NBV planning. We introduce an estimator for the expected SfM reconstruction quality. This estimator is subsequently employed in a gain formulation that assesses arbitrary recording poses. The gain term is then used in the objective function of an OED problem to obtain full, admissible NBV trajectories. Mathematical formulations are kept very general and especially allow for arbitrary robotic systems acting as recording platforms and multi-camera setups. The resulting OED problem will prove too hard to be solved analytically. Obtaining approximate solutions will be the subject of chapter 7.

We will now give a geometric motivation on how the estimate on the expected reconstruction quality will be modeled. Consider a single surface point and a set of camera poses that observe it. For the sake of simplicity, let the cardinality of this set of viewpoints be two for now. Then the reconstruction of the surface point can be thought of as a triangulation, i.e. computing intersections between the rays from the camera centers through the observations of the point on the image plane. If the viewpoints are identical or collinear with the surface point, such unique intersection points do not exist and reconstruction becomes impossible. Also, small baselines between the rays lead to an erroneous point reconstruction. Then the angle between the rays becomes very narrow and small errors in the image plane observations are greatly amplified during triangulation. Hence, it is insufficient to only consider the number of viewpoints, as the spatial relation between the recording poses is also important. A naive approach would associate the point with each camera pose that observed it, e.g. by arrows pointing in the respective directions. However, then the computational complexity required for evaluating a quality measure, as well as the memory consumption for each point, would grow indefinitely with the number of images. Instead, we describe the area that is estimated to contain the true position of the point using simple geometric shapes. Since depth can not be extracted from a single image, such a shape would be an infinite height elliptic cylinder, oriented such that its main axis coincides with the viewing direction, for each recorded image. The intersection of all elliptic cylinders would then describe our assumption on the position of the point. However, this would lead to complex geometric shapes, the description of which would also require an increasing number of parameters. Fortunately, these elliptic cylinders can be related to Bayesian theory as confidence areas of Gaussian distributions. Each image can be interpreted as a measurement and the quality estimate of the point is simply the joint distribution of all measurements. Then a single symmetric 3×3 matrix, that describes the covariance of the joint distribution, is required for the quality estimate of each point. The associated confidence region's shape is an ellipsoid.

6.1 Special Treatment of Singular Precision Matrices

Precision matrices of the form $\Sigma^{-1} \in \mathbb{R}^{3\times3}$ appear frequently in the following sections. While – by definition – those are symmetric positive definite, we will also allow for positive semi-definite matrices. In the special case of a singular precision matrix, the inverse (covariance matrix) does not exist. However, we will introduce a way to overcome this issue and define some properties of an associated covariance matrix. Consider the symmetric positive semi-definite precision matrix $\Sigma^{-1} \in \mathbb{R}^{3\times3}$. Its singular value decomposition is given by

$$\Sigma^{-1} = U \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_3 \end{bmatrix} U^T, \tag{6.1}$$

where $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}_{\geq 0}$ are the eigenvalues and $U \in SO(3)$ contain the eigenvectors of Σ^{-1}. In the case of positive definiteness, i.e. strictly positive eigenvalues, this matrix is invertible with its inverse given by

$$\Sigma = U \begin{bmatrix} 1/\lambda_1 & & \\ & 1/\lambda_2 & \\ & & 1/\lambda_3 \end{bmatrix} U^T. \tag{6.2}$$

If one of those eigenvalues λ_i goes to zero, all entries of Σ go to infinity. Hence, it makes sense that this matrix expression does not exist, since in this case $\Sigma \cdot \Sigma^{-1} \neq I_3$. Although no analytic expression of its inverse is possible, we can characterize it using the singular value decomposition (6.2) and consider the limits $\lambda_i \searrow 0$.

An interpretation of covariance matrices as confidence ellipsoids is given in chapter 3.1. The eigenvalues $1/\lambda_i$ of Σ define the lengths of these ellipsoids half-axes, while the eigenvector matrix U rotates it from its axis-aligned position. Without loss of generality, let λ_1 be the single eigenvalue that goes to zero. Then the confidence ellipsoid scales in the direction of the associated eigenvalue u_1 and ultimately becomes an elliptic cylinder. This makes sense, as a precision of zero results in maximum uncertainty (i.e. "infinite" variance) in this direction. The associated axis-aligned multivariate normal distribution is given by

$$\begin{bmatrix} \mathcal{Y}_1 \\ \mathcal{Y}_2 \\ \mathcal{Y}_3 \end{bmatrix} \sim U^T(\mathcal{N}(\mu, \Sigma) - \mu) = \mathcal{N}\left(0, \begin{bmatrix} 1/\lambda_1 & & \\ & 1/\lambda_2 & \\ & & 1/\lambda_3 \end{bmatrix}\right), \tag{6.3}$$

with a mean vector $\mu \in \mathbb{R}^3$. Note that $\{\mathcal{Y}_i\}_{i\in\{1,2,3\}}$ are mutually independent distributed. While not mathematically rigorous, we now allow for infinite variance distributions, i.e.

$$\lim_{\lambda_1 \searrow 0} \mathcal{Y}_1 \sim \mathcal{N}(0, \infty). \tag{6.4}$$

In this artificial distribution each value in $\mathbb{R}$ has the same probability. These observations allow us to justify the following notation:

$$\mathcal{N}(\mu, \Sigma). \tag{6.5}$$

Although the matrix Σ cannot be represented analytically for $\lambda_1 \searrow 0$, the corresponding distribution can be determined unambiguously in the limiting case by the singular value decomposition of its precision matrix.

We will also require a notation for the eigenvalues of Σ for possibly singular precision matrices. Again, from the singular value decomposition (6.2), we can define them using the eigenvalues of the precision matrix with

$$\lambda_i[\Sigma] := \frac{1}{\lambda_i[\Sigma^{-1}]}, \quad i \in \{1,2,3\}. \tag{6.6}$$

Note that those eigenvalues are always in $\mathbb{R}_{>0} \cup \infty$ and especially are never zero. This further allows for the definition of the determinant as

$$\det(\Sigma) := \lambda_1[\Sigma] \cdot \lambda_2[\Sigma] \cdot \lambda_3[\Sigma] = \frac{1}{\det(\Sigma^{-1})} = \frac{1}{\lambda_1[\Sigma^{-1}]} \cdot \frac{1}{\lambda_2[\Sigma^{-1}]} \cdot \frac{1}{\lambda_3[\Sigma^{-1}]}, \tag{6.7}$$

which is defined for all possible semi-definite precision matrices Σ^{-1} and also maps to $\mathbb{R}_{>0} \cup \infty$.

Ultimately, this extension to singular precision matrices is only required to simplify the notation for the following sections. In the end, all quantities will be formulated in terms of precision matrices that do not require this unconventional use of inverses of singular matrices. Identical results can be achieved by consideration of the respective limits for the problematic eigenvalues. Alternatively, arbitrarily small constant values can be used instead of zero eigenvalues in the precision matrices.

6.2 Estimating Expected Reconstruction Quality

In this section, we develop an estimator for the expected SfM reconstruction error for a single surface point. While this surface point does not necessarily correspond to a SfM feature in any image, it is assumed to be reconstructable during depth map estimation in the SfM workflow. Points that do not satisfy this assumption correspond to reflective or (semi-) transparent surfaces, or areas that result in low image gradients (e.g. due to lack of texture, monochromatic surfaces). Even if observed in many overlapping images, those areas can not be correctly reconstructed, hence, all contained points are neglected by the estimator.

We treat each image that observes the point as an additional measurement that improves the estimate of the true surface point. Combining all measurements yields a joint posterior distribution that quantifies the point quality (i.e. the reconstruction error) in terms of the estimated covariance matrix and confidence volume. We will now give the mathematical background and construct a suitable measurement covariance matrix.

6.2.1 Point Observations

Let $M \subset \mathbb{R}^3$ be a set of two-dimensional manifolds that describe the surface of the object to be observed. Furthermore, let $p_w = (x_w, y_w, z_w)^T \in M$ be a point on this surface in world-space that is observed in the image of a camera located at $\{R_c, t_c\} \in SO(3) \times \mathbb{R}^3$. Then point p_w in camera coordinate frame is given as

$$p_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = R_c^T \left(\begin{bmatrix} x_w \\ y_w \\ c_w \end{bmatrix} - t_c \right). \tag{6.8}$$

The world point is projected to the image plane using a pinhole camera model

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} f_u & s \\ 0 & f_v \end{bmatrix} \begin{bmatrix} x_c/z_c \\ y_c/z_c \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix}, \tag{6.9}$$

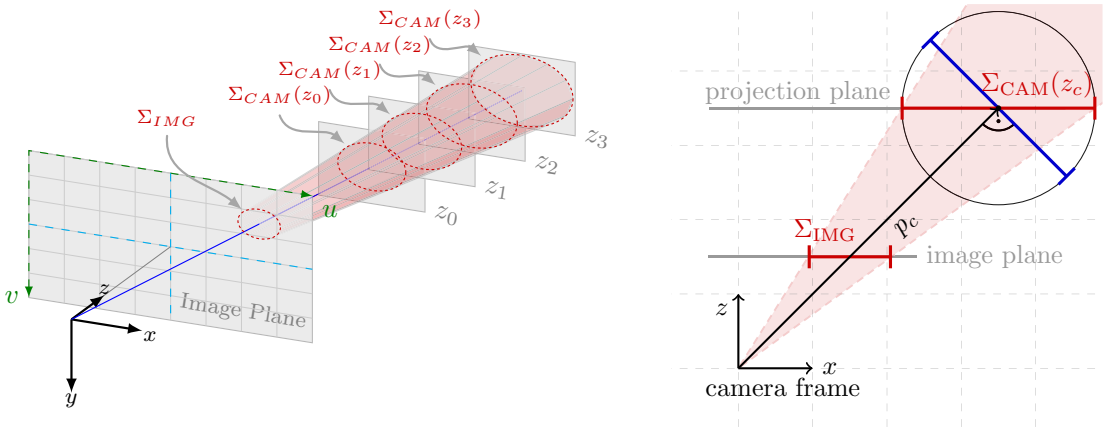
where f_u, f_v are the cameras focal lengths, s the axis skew and $(c_x, c_y)^T$ the principal point, all expressed in pixel units. Since pixels are discrete quantities and due to other imprecision introduced in the recording procedure, the exact value of $(u_0, v_0)^T$ can not be measured exactly in an image. This error is modeled by a normal distributed random variable

$$\begin{bmatrix} \mathcal{U} \\ \mathcal{V} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} u_0 \\ v_0 \end{bmatrix}, \Sigma_{\text{IMG}} \right), \quad \Sigma_{\text{IMG}} := \begin{bmatrix} \sigma_u^2 & 0 \\ 0 & \sigma_v^2 \end{bmatrix}, \tag{6.10}$$

with some constants $\sigma_u, \sigma_v \in \mathbb{R}_{>0}$. By reprojecting $(\mathcal{U}, \mathcal{V})^T$ into camera space with known depth z_c, we obtain a random variable in camera space:

$$\begin{bmatrix} \mathcal{X} \\ \mathcal{Y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} x_c \\ y_c \end{bmatrix}, \underbrace{z_c^2 \cdot \begin{bmatrix} \frac{f_v^2\sigma_u^2 + s^2\sigma_v^2}{f_u^2 f_v^2} & -\frac{s\sigma_v}{f_u f_v^2} \\ -\frac{s\sigma_v}{f_u f_v^2} & \frac{\sigma_v^2}{f_v^2} \end{bmatrix}}_{=:\Sigma_{\text{CAM}}(z_c)} \right). \tag{6.11}$$

This covariance matrix is visualized in figure 6.1a in terms of confidence ellipses for different z_c.



(a) Confidence cone. **(b)** Projected confidence cone.

Figure 6.1: Reprojection of a confidence ellipse on the image plane into camera coordinates for different values of z_c. The depth z_c parameterizes an oblique confidence cone. Each Σ_{CAM} lies on a projected plane parallel to the image plane.
Source: Left: [Lindner et al., 2019] ©2019 IEEE.

We now want to extend Σ_{CAM} to three dimensions, such that the information added by the camera is modeled. From a single image, depth can not be measured. Hence, the desired covariance matrix should have infinite variance in the ray-direction $\vec{p_c}$. Expressed in terms of confidence ellipsoids, this corresponds to a precision matrix with one eigenvalue equal to zero (see section 3.1). The remaining two half-axes are now to be determined from $\Sigma_{\text{CAM}}(z_c)$ using upper bounds.

Definition 6.1 (Covariance upper bound)**.** Let $\Sigma \in \mathbb{R}^{3\times3}$ be a covariance matrix. Then the covariance matrix $\widetilde{\Sigma} \in \mathbb{R}^{3\times3}$ is called an upper bound of Σ if for any fixed confidence percentage the confidence ellipsoid of Σ are completely contained in the confidence ellipsoid of $\widetilde{\Sigma}$.

The half-axes of ellipsoids are always orthogonal, hence, an upper bound on conic sections of the confidence cone (see figure 6.1a) orthogonal to the view directions is required. For geometric reasons Σ_{CAM} is always an upper bound to this orthogonal projection (see figure 6.1b, blue line) and is chosen to define the remaining two confidence ellipsoid half-axes. To further simplify calculations, we introduce another upper bound. Since the lengths of the confidence ellipsoid half-axes correspond to the eigenvalues of their covariance matrices,

$$\Sigma^*_{\text{CAM}}(z_c) := z_c^2\gamma^2 I_2, \quad \gamma = \sqrt{\lambda_{\max}\left(\begin{bmatrix} \frac{f_v^2\sigma_u^2+s^2\sigma_v^2}{f_u^2f_v^2} & -\frac{s\sigma_v}{f_uf_v^2} \\ -\frac{s\sigma_v}{f_uf_v^2} & \frac{\sigma_v^2}{f_v^2} \end{bmatrix}\right)} \tag{6.12}$$

is an upper bound of the reprojected covariance matrix Σ_{CAM}.

With all axes of the desired confidence ellipsoid determined, it needs to be rotated into the world frame. Let

$$R_{p_c} = \left[\; r_1 \;\middle|\; r_2 \;\middle|\; \tfrac{p_c}{\|p_c\|_2} \;\right] \in \text{SO}(3) \tag{6.13}$$

be a rotation matrix, rotating the image plane normal (z-axis in camera frame) onto p_c. Then a measure for the information an image taken at pose $\{R_c, t_c\}$ contributes to the SfM reconstruction of a surface point p_w is given in terms of the precision matrix (inverse covariance matrix)

$$\begin{aligned} \Theta^{-1}(p_w, R_c, t_c) &= R_c R_{p_c} \begin{bmatrix} \Sigma^{*-1}_{\text{CAM}}(z_c) & \\ & 0 \end{bmatrix} R_{p_c}^T R_c^T \\ &= R_c \left[\frac{1}{\gamma^2 z_c^2}\left(I_3 - \frac{p_c}{\|p_c\|_2}\frac{p_c^T}{\|p_c\|_2}\right)\right] R_c^T \\ &= \frac{1}{\gamma^2\langle p_w - t_c, R_c^z\rangle^2}\left(I_3 - \frac{(p_w-t_c)}{\|p_w-t_c\|_2}\frac{(p_w-t_c)^T}{\|p_w-t_c\|_2}\right), \end{aligned} \tag{6.14}$$

where R_c^z is the image plane normal in world coordinates. Note that the scaling factor of $1/\gamma^2\|p_c\|_2^2$ can alternatively be used in equation (6.14), as this results yet again in a coarser upper bound estimate. Doing so would make the precision matrix independent of the camera orientation.

Since the true value of p_w is unknown, we model it using the multivariate normal distribution

$$\mathcal{P}_i \sim \mathcal{N}(p_i, \Sigma_i), \tag{6.15}$$

where the mean $p_i \in \mathbb{R}^3$ and covariance matrix $\Sigma_i \in \mathbb{R}^{3\times3}$ correspond to our prior knowledge of the true value p_w after observing it in i images. As we will see later, p_i is of no importance for our estimator. An image observing the point p_w from a camera pose $\{R_c, t_c\}$ can be interpreted as a measurement $m_i \in \mathbb{R}^3$ of p_w, which is a sample from a distribution

$$\mathcal{M}_i|\mathcal{P}_i \sim \mathcal{N}(\mathcal{P}_i, \Theta(p_w, R_c, t_c)). \tag{6.16}$$

For simplicity of notation we omit the arguments of $\Theta(p_w, R_c, t_c)$ and only write Θ. We are now interested in the distribution of $(\mathcal{P}_i|\mathcal{M}_i = m_i)$, to obtain a more precise estimate of the true value p_w. From Bayes's rule the density function of this posterior distribution is given by

$$f_{\mathcal{P}_i|\mathcal{M}_i=m_i}(p) = \frac{f_{\mathcal{M}_i|\mathcal{P}_i=p}(m_i) f_{\mathcal{P}_i}(p)}{f_{\mathcal{M}_i}(m_i)}, \quad \forall p \in \mathbb{R}^3. \tag{6.17}$$

Gaussian distributions are self-conjugate. Hence, by straight forward multiplication of the densities in equation (6.17), we obtain

$$\mathcal{P}_{i+1} := (\mathcal{P}_i|\mathcal{M}_i = m_i) \sim \mathcal{N}(p_{i+1}, \Sigma_{i+1}), \tag{6.18}$$

with

$$\begin{aligned} p_{i+1} &= \Sigma_{i+1}^{-1} \left(\Sigma_i^{-1} p_i + \Theta^{-1} m_i\right) \\ \Sigma_{i+1} &= \Sigma_i \left(\Sigma_i + \Theta\right)^{-1} \Theta = \left(\Sigma_i^{-1} + \Theta^{-1}\right)^{-1}. \end{aligned} \tag{6.19}$$

Note that, strictly speaking, the matrix Θ does not exist, since one eigenvalue of Θ^{-1} is equal to zero by construction. Similarly, Σ_i^{-1} may also have zero eigenvalues. Those zero eigenvalues correspond to "infinite variance" in the direction of their corresponding eigenvectors. Ultimately, however, we are only interested in the posterior covariance information, which can be written entirely in terms of precision matrices without the need of matrix inversion:

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + \Theta^{-1}(p_w, R_c, t_c). \tag{6.20}$$

This result is still valid for those zero eigenvalue precision matrices, which is easily checked by following the same argumentation as above and looking at the limits for the problematic eigenvalues. This is explained in more detail in section 6.1. Further consideration of the eigenvalues of all quantities in equation (6.20) provides a geometrical interpretation. Weyl's inequality (see appendix A) guarantees that the eigenvalues of a sum of symmetric positive definite matrices are always larger or equal to the eigenvalues of each individual matrix. This means that the confidence ellipsoid of the precision matrix Σ_{i+1}^{-1} is guaranteed to be a subset of both confidence ellipsoids Σ_i^{-1} and Θ^{-1} (see section 3.1). Hence, those prior confidence ellipsoids are guaranteed to shrink, improving our knowledge about the precision of our estimate. This is illustrated in figure 6.2.

The recursive update formulation in equation (6.20) is computationally efficient. All of the involved matrices are symmetric, which means that only an upper triangular matrix has to be stored in memory. Due to the structure of the update (6.14), equation (6.20) is nearly as simple as a rank 1 update. Furthermore, multi-camera systems with varying intrinsic parameters are easily supported. This is realized by simply adjusting γ accordingly, as it is the only quantity that depends on the intrinsic parameters.

Note that while p_w is assumed to be unknown Θ^{-1} still depends on it. This is further discussed in section 7.1.

6.2.2 Choice of Estimator Parameters

Before the point p_w is observed for the first time, it is considered unknown. Therefore, we choose $\Sigma_0 = 0_{3\times3}$ as initial value. The constant γ is computed from each set

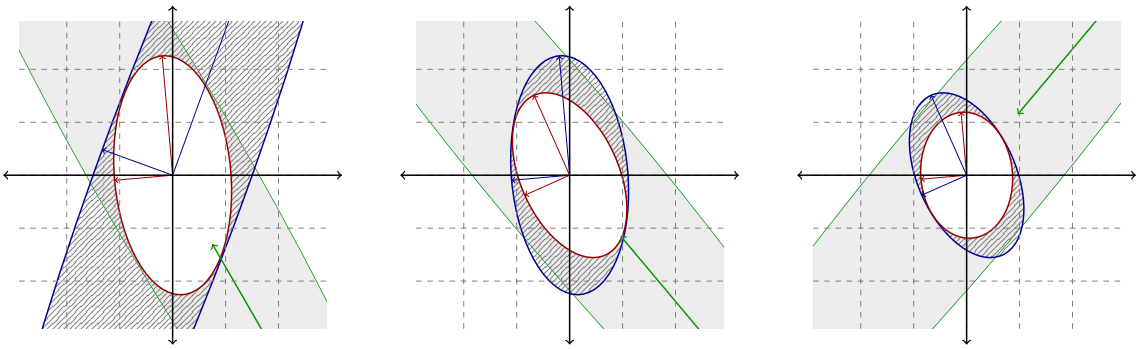


Figure 6.2: SfM covariance quality estimate example for three consecutive updates. The green elliptic cylinder (ellipse with infinite half axis length in one direction) corresponds to confidence area of the update Θ_i, i.e. the gained information. The blue ellipse is the confidence area of the prior Σ_i and the red one the confidence ellipse of the posterior Σ_{i+1}.

of intrinsic camera parameters and the assumption on the image plane error σ_u and σ_v. Note that usually the axis skew s is zero, such that equation (6.12) simplifies to $\gamma = \max\{\sigma_u/f_u, \sigma_v/f_v\}$.

The variances σ_u^2 and σ_v^2 describe the spread of the pixel error in the image plane. They can be determined from a desired confidence interval sizes. Following the notation of chapter 3.1, if a confidence of p_{conf} is desired for an interval of $\pm n_{\text{pix}}$ pixel, the identity

$$\begin{aligned} |n_{\text{pix}}| &\stackrel{!}{=} \sqrt{\sigma_u^2 Q_2(p_{conf})} \\ |n_{\text{pix}}| &\stackrel{!}{=} \sqrt{\sigma_v^2 Q_2(p_{conf})} \end{aligned} \tag{6.21}$$

must hold, which is equivalent to

$$\begin{aligned} \sigma_u &= \frac{|n_{\text{pix}}|}{\sqrt{Q_2(p_{conf})}} \\ \sigma_v &= \frac{|n_{\text{pix}}|}{\sqrt{Q_2(p_{conf})}}. \end{aligned} \tag{6.22}$$

Then γ can be written as

$$\gamma = \frac{|n_{\text{pix}}|}{\sqrt{Q_2(p_{conf})}} \max\{1/f_u, 1/f_v\}. \tag{6.23}$$

By construction, γ has only a scaling effect on Θ and consequently on the precision matrices that describe the estimate of the expected point reconstruction quality. More precisely, if we scale γ by a factor $c \in \mathbb{R}_{>0}$, then all Σ_i^{-1} are scaled by a factor of $1/c^2$. From (6.23), similar statements are obtainable for n_{pix} and p_{conf}. Appropriate values for n_{pix} and p_{conf} will be specified in chapter 10.

6.2.3 Image Observations

In this section, the estimator for the expected point reconstruction quality is extended to the whole set of manifolds M. This is as simple as defining the quality of

each point in M, while accounting for observability. The main purpose of this chapter is to introduce new notations. We are interested in the quality of an arbitrary point on that manifold after taking n images at locations $\xi_i := \{R_i, t_i\} \in SO(3) \times \mathbb{R}^3$. The set of all n previous camera recording poses is denoted by $\Xi_n := \{\xi_1, \dots, \xi_n\}$. Furthermore, let

$$\mathrm{FOV} : \xi \in SO(3) \times \mathbb{R}^3 \mapsto \mathrm{FOV}(\xi) \subset M \tag{6.24}$$

denote the set of all surface points that are visible from configuration ξ. The observability function classifying surface points as visible or occupied, is given by

$$\mathrm{vis}_\xi(p) := \begin{cases} 1 & \text{if } p \in \mathrm{FOV}(\xi), \text{ i.e. } p \text{ is visible from pose } \xi \\ 0 & \text{if } p \notin \mathrm{FOV}(\xi), \text{ i.e. } p \text{ is not visible from pose } \xi. \end{cases} \tag{6.25}$$

Then the point reconstruction quality estimator from equation (6.20) can be written as

$$\Sigma^{-1}(p, \Xi_n) := \sum_{\xi \in \Xi_n} \mathrm{vis}_\xi(p) \cdot \Theta^{-1}(p, \xi), \tag{6.26}$$

with Θ^{-1} as in (6.14). Equivalently, the following recursive formulation holds:

$$\Sigma^{-1}(p, \Xi_{n+1}) = \begin{cases} \Sigma^{-1}(p, \Xi_n) + \Theta^{-1}(p, \xi_{n+1}) & \text{if } p \in \mathrm{FOV}(\xi_{n+1}) \\ \Sigma^{-1}(p, \Xi_n) & \text{if } p \notin \mathrm{FOV}(\xi_{n+1}). \end{cases} \tag{6.27}$$

As explained in section 6.2.2, this recursion is initialized with $\Sigma^{-1}(p, \{\}) = 0_{3\times3}$. The associated multivariate random variable describing a point's quality is denoted by

$$\mathcal{X}_p(\Xi_n) \sim \mathcal{N}(0, \Sigma(p, \Xi_n)). \tag{6.28}$$

6.3 Optimal View Planning

In the previous sections we derived an estimator for the expected SfM reconstruction quality, given a set of camera poses Ξ_n. Now the question arises how these recording poses can best be chosen to achieve a uniform coverage of the manifolds. An optimal set must contain as few images as possible, such that the SfM reconstruction runtime is minimal. At the same time it must be possible to reconstruct the geometry sufficiently, i.e. without holes and with a certain reconstruction quality. The latter requirement can be formulated as

$$\lambda_i \left[\Sigma(p, \Xi_n)\right] \leq \lambda_{\min}, \quad i \in \{1, 2, 3\}, \, \forall p \in M, \tag{6.29}$$

for some constant threshold $\lambda_{\min} \in \mathbb{R}_{>0}$. This means that all half-axes of the associated confidence ellipsoids (see section 3.1) must be sufficiently small. In order to obtain a small image set, each individual photograph, again interpreted as measurements, must significantly improve the estimate of the reconstruction quality. Putting this into a mathematical framework, we are interested in the solution to

$$\min_{n \in \mathbb{N}} n \tag{6.30a}$$

$$\text{s.t.} \quad \Xi_n^* = \operatorname*{argmin}_{\Xi_n} \int_M \Phi(\Sigma(p, \Xi_n)) dp, \tag{6.30b}$$

$$\lambda_i \left[\Sigma(p, \Xi_n^*)\right] \leq \lambda_{\min}, \quad i \in \{1, 2, 3\}, \, \forall p \in M, \tag{6.30c}$$

i.e. the minimum number of pictures n, such that the camera poses obtained by (6.30b) still satisfy constraints (6.29). The function Φ quantifies the size of the confidence ellipsoid associated with a covariance matrix. It may correspond to an OED optimality criteria (see section 3.2), e.g.

$$\Phi(\Sigma(p, \Xi_n)) = \log\det[\Sigma(p, \Xi_n)], \tag{6.31}$$

which is equivalent to the D-optimality criteria in the minimization context. This choice seems intuitive, due to the equivalence to the minimization of

$$\Phi(\Sigma(p, \Xi_n)) = H(\mathcal{X}_p(\Xi_n)), \tag{6.32}$$

where

$$H[\mathcal{X}_p(\Xi_n)] = \frac{3}{2}\left(\log[2\pi] + 1\right) + \frac{1}{2}\log\det\left[\Sigma(p, \Xi_n)\right] \tag{6.33}$$

is the differential entropy[6] of a surface point. Entropy plays a key role in information theory and describes the average information content or the minimum number of bits necessary to encode a message. This means that a low entropy gives higher certainty about the observed data.

The two-stage mixed-integer minimization problem in equation (6.30) is NP-hard. For practical applications, we further require additional geometric constraints to guarantee feasibility of the optimized camera poses and an admissible path connecting them. However, even without these additional constraints, problem (6.30) is infeasible to solve numerically. The sub-problem (6.30b) is high dimensional and the function Φ is non-smooth at the borders of each viewing frustum. To overcome these issues, we will now propose an alternative approach that does not optimize all camera poses at once, but greedily searches only for the NBV.

6.3.1 Next-Best-View: Gain Formulation

By only searching for a NBV, we are not interested in the total quality obtained from all views combined, but the improvement this next view provides. Assume n images are already taken at poses Ξ_n. In analogy to (6.30b), we are therefore interested in the solution to

$$\xi^*_{n+1} = \underset{\xi_{n+1}}{\mathbf{argmax}} \int_M \text{gain}(p, \Xi_n, \xi_{n+1})\ dp. \tag{6.34}$$

Note that this gain-function plays a similar role as Φ in (6.30b). However, instead of quantifying the absolute quality, it describes the information gained from view ξ_{n+1}.

The choice of this gain function determines the behavior of the NBV planning. By choosing it similar to Φ, e.g. as one of the OED criteria given in section 3.2, the NBV would prefer observing the currently worst covered parts of the surface geometry. In the greedy NBV scheme, this results in a uniform reduction of the half-axes of all confidence ellipsoids of the expected reconstruction quality of points on the whole surface. However, consecutive views will also be far apart as they tend to result in poses with little image overlap with previous recordings. Ultimately, we

[6]Note that differential entropy is not the continuous analogue of discrete entropy. However, it can be interpreted similarly.

are not only interested in a set of NBVs, but also a trajectory connecting all views. Therefore, the gain function should "motivate" the NBV selection to first observe local portions of the surface sufficiently before continuing to explore the surface. This motivates our choice of the gain function as

$$\begin{aligned} \text{gain}(p, \Xi_n, \xi_{n+1}) &:= H[\mathcal{X}_p(\Xi_n)] - H[\mathcal{X}_p(\Xi_{n+1})] \\ &= -\frac{1}{2} \log \left[\frac{\det \Sigma(p, \Xi_{n+1})}{\det \Sigma(p, \Xi_n)} \right]. \end{aligned} \tag{6.35}$$

This formulation is inspired by the mutual information approach for optimal sensor placements given in [Krause et al., 2008]. It describes a relative quantity and is therefore only observing relative improvements of the estimated surface quality. Note that we still require a termination criteria that stops observing surface points once a certain quality threshold is reached. Again, all the precision matrices associated with covariance matrices in (6.35) may be singular, leading to undefined gain expressions. Both of these issues will be discussed in the next section. For now we will assume that these matrices are regular.

Similar to OED optimality criteria, the gain function (6.35) also allows for a geometric interpretation. Due to (6.27) and Weyl's inequality (appendix A), the eigenvalues of $\Sigma(p, \Xi_{n+1})$ must always be smaller or equal to the ones of $\Sigma(p, \Xi_n)$. The fraction

$$\frac{\det \Sigma(p, \Xi_{n+1})}{\det \Sigma(p, \Xi_n)}. \tag{6.36}$$

is always in $(0, 1]$, but apart from the initial observation closer to 1. Then a good approximation of the gain function (6.35) is given by

$$\text{gain}(p, \Xi_n, \xi_{n+1}) \overset{\substack{\text{First order} \\ \text{Taylor expansion} \\ \text{of log at 1}}}{\approx} \underbrace{1 - \sqrt{\frac{\det \Sigma(p, \Xi_n, \xi_{n+1})}{\det \Sigma(p, \Xi_n)}}}_{\in [0,1]}. \tag{6.37}$$

As we have seen in section 3.2, the square root of the determinant of a covariance matrix is proportional to the volume of its confidence ellipsoid. Therefore, equation (6.37) describes the percentage decrease of confidence volume given a new observation ξ_{n+1}.

The gain function (6.35) gives us a computational advantage over an absolute measure like (6.31). Considering the recursive definition of $\Sigma^{-1}(p, \Xi_{n+1})$ given in (6.27), we realize

$$\text{gain}(p, \Xi_n, \xi_{n+1}) = 0, \quad \forall p \notin \text{FOV}(\xi_{n+1}). \tag{6.38}$$

This allows us to simplify (6.34) to obtain

$$\xi^*_{n+1} = \underset{\xi_{n+1}}{\textbf{argmax}} \int_{FOV[\xi_{n+1}]} \text{gain}(p, \Xi_n, \xi_{n+1}) \, dp. \tag{6.39}$$

Now, the objective function only considers points which are inside the viewing frustum of ξ_{n+1}. This insight is extremely important, as it significantly accelerates computations when evaluating this function later on in section 7.2. From now on we will only consider the non-trivial case of $\text{gain}(p, \Xi_n, \xi_{n+1}) \neq 0$, or equivalently $p \in \text{FOV}(\xi_{n+1})$, when talking about the gain function.

6.3.2 Gain Clamping

In section 6.1 we established how to treat singular precision matrices. The matrix $\Sigma^{-1}(p, \Xi_n)$ may be singular for some values of p and Ξ_n, resulting in $\det[\Sigma(p, \Xi_n)] = \infty$. This poses a problem as the gain function (6.35) can yield undefined expressions (e.g. $\log(\infty/\infty)$) or a small surface area may dominate the whole objective function integral (6.39) (e.g. $\log(0)$). This is quite common since the precision matrix $\Sigma^{-1}(p, \{\})$ is initialized with $0_{3\times3}$, and $\Sigma^{-1}(p, \{\xi_1\})$ always has one eigenvalue equal to zero. To overcome these issues, we introduce an upper bound for the eigenvalues of Σ, such that a modified determinant is computed as

$$\prod_{i=1}^{3} \min\left(\lambda_i[\Sigma(p, \Xi_n)], \lambda_{\max}\right), \tag{6.40}$$

with some constant $\lambda_{\max} \in \mathbb{R}_{>0}$. Similarly, a lower bound $\lambda_{\min} \in \mathbb{R}_{>0}$ can be introduced that acts as a termination criterion. The modified gain formulation is then given as

$$\begin{aligned} \overline{\text{gain}}(p, \Xi_n, \xi_{n+1}) &:= -\frac{1}{2}\log\left[\frac{\overline{\det}\,\Sigma(p, \Xi_{n+1})}{\overline{\det}\,\Sigma(p, \Xi_n)}\right], \\ \text{with} \qquad \overline{\det}[A] &:= \overline{\lambda_1}[A] \cdot \overline{\lambda_2}[A] \cdot \overline{\lambda_3}[A], \\ \overline{\lambda_i}[A] &:= \text{clamp}(\lambda_{\min} \leq \lambda_i[A] \leq \lambda_{\max}). \end{aligned} \tag{6.41}$$

The constants $\lambda_{\min}$ and $\lambda_{\max}$ ensure that only relevant changes to the covariance matrix of point p are observable. The upper bound $\lambda_{\max}$ now allows us to sense measurements even in the case of singular precision matrices. Furthermore, the lower bound $\lambda_{\min}$ has the same functionality as the constraint (6.30c), because points p with

$$\lambda_i\left[\Sigma(p, \Xi_n)\right] \leq \lambda_{\min}, \quad \forall i \in \{1, 2, 3\} \tag{6.42}$$

yield a gain equal to zero. It is therefore used to avoid over-observing surface points. Note that the clamping parameters also introduce bounds on the gain function itself, given by

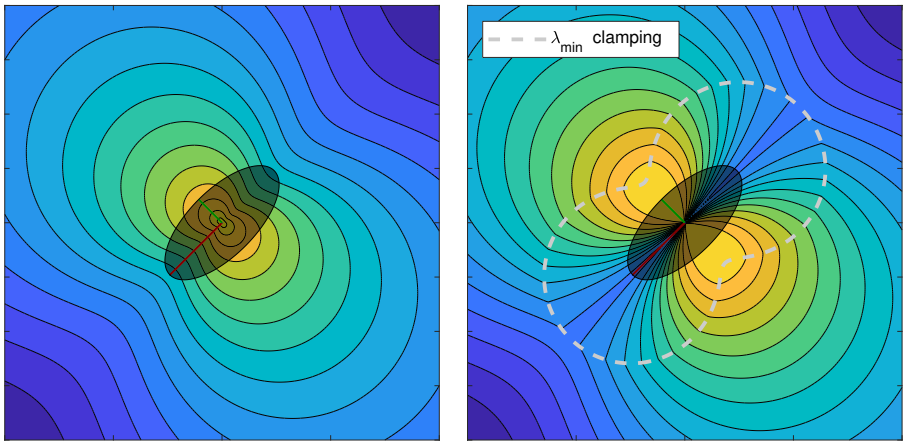
$$\text{gain}(p, \Xi_n, \xi_{n+1}) \in \left[0, -\frac{3}{2}\log\left(\frac{\lambda_{\min}}{\lambda_{\max}}\right)\right]. \tag{6.43}$$

The effect this clamping formulation has on the gain is visualized in figure 6.3.

Finally, the gain formulation can also be formulated in terms of the precision matrices:

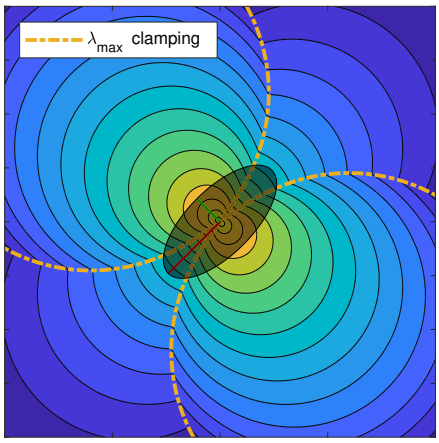
$$\begin{aligned} \overline{\text{gain}}(p, \Xi_n, \xi_{n+1}) &= \frac{1}{2}\log\left[\frac{\overline{\det}\,\Sigma^{-1}(p, \Xi_{n+1})}{\overline{\det}\,\Sigma^{-1}(p, \Xi_n)}\right] \\ \overline{\det}\,\Sigma^{-1}(p, \Xi_i) &= \prod_{i=1}^{3} \text{clamp}\left(\frac{1}{\lambda_{\max}} \leq \lambda_i\left[\Sigma^{-1}(p, \Xi_i)\right] \leq \frac{1}{\lambda_{\min}}\right) \end{aligned} \tag{6.44}$$

Since the gain update (6.27) also only requires precision matrices, covariance matrices never need to be calculated explicitly. This way we avoid unnecessary matrix inversions and no longer require the properties of inverses of singular matrices defined in section 6.1.

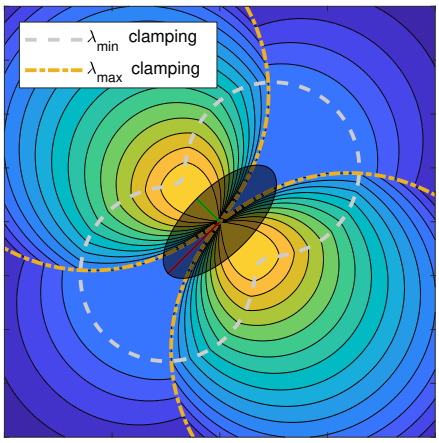


(a) No clamping.

(b) Clamping to $\lambda_{\min}$.



(c) Clamping to $\lambda_{\max}$.



(d) Clamping to $\lambda_{\min}$ and $\lambda_{\max}$.

Figure 6.3: Contour lines of the $\overline{\text{gain}}$ function. The confidence ellipse of $\Sigma(p, \Xi_n)$ is visualized in gray (transparent) with the point p being in its center. The position of each new view ξ_{n+1} is sampled in the 2D plane, with the viewing direction from the camera to the point being orthogonal to each image plane. Clamping to $\lambda_{\min}$ is active inside the area enclosed by the gray line in figure 6.3b and 6.3d. Clamping to $\lambda_{\max}$ is active in the lower left and upper right corners in figure 6.3c and 6.3d, bordered by the yellow line.

The values of $\lambda_{\min}$ and $\lambda_{\max}$ can be chosen from the ground sampling distance of the camera[7]. Assume a sufficient ground sampling distance is achieved from pictures that are as close as $d_{\min}$ meters from the surface point p. This means that a single recording ξ from a distance of equal or less than $d_{\min}$ meters is sufficient to reach the desired precision $\lambda_{\min}$ for two out of three eigenvalues of $\Sigma(p, \{\xi\})$. By construction (see equation 6.14), these two finite eigenvalues can be identified as the eigenvalues of $\Sigma^*_{\text{CAM}}(d_{\min})$. Analogous we can define a maximum distance $d_{\max}$ after which the information contributed by the observations become irrelevant. Hence, the clamping parameters can be chosen as

$$\begin{aligned} \lambda_{\min} &:= d_{\min}^2 \gamma^2 \\ \lambda_{\max} &:= d_{\max}^2 \gamma^2. \end{aligned} \tag{6.45}$$

[7] If several different cameras are used, these clamping parameters can be selected differently for each camera.

Note that by choosing the clamping parameters as in (6.45), the gain function (6.44) becomes invariant to scaling of γ. This especially implies that the actual assumption on the image plane error σ_u, σ_v – and in consequence n_{pix} and its associated confidence percentage (see section 6.2.2) – do not affect the gain function or the NBV computation. Therefore, the sole purpose of these parameters is to approximate the magnitude of the expected reconstruction error (see section 9.2.2).

6.3.3 Next-Best-View Trajectory

We only considered discrete camera poses so far. In order to take a new measurement at camera pose ξ_{n+1}, it is also important to find a short, collision-free trajectory $\xi(t) : \mathbb{R} \to SO(3) \times \mathbb{R}^3$ that describes the cameras pose at time t. Hence, we are interested in an admissible path from $\xi(t_n) := \xi_n$ to $\xi(t_{n+1}) := \xi_{n+1}$ for a greedy NBV selection. As the camera may be attached to a robotic system, such as a robotic arm or a UAV, we write the camera trajectory as a dynamic system for the sake of generality. The full OED problem, which calculates the NBV, is given by

$$\max_{\xi(\cdot),u(\cdot),t_{n+1}} \int_{FOV[\xi(t_{n+1})]} \overline{\text{gain}}(p, \Xi_n, \xi(t_{n+1}))\, dp \tag{6.46a}$$

$$- \int_{t_n}^{t_{n+1}} \text{cost}(\xi(\tau))\, d\tau \tag{6.46b}$$

$$\textbf{s.t.} \quad \ddot{\xi}(t) = f(t, \xi(t), u(t)), \quad \xi(t_n) = \xi_n, \quad \dot{\xi}(t_n) = \dot{\xi}_n \tag{6.46c}$$

$$g(t, \xi(t), u(t)) \leq 0 \tag{6.46d}$$

$$o_{\min} \leq g_o(\xi(t_{n+1})) \tag{6.46e}$$

$$d^{\text{o}}_{\min} \leq d_o(\xi(t)) \leq d^{\text{o}}_{\max} \tag{6.46f}$$

$$d^{\text{u}}_{\min} \leq d_u(\xi(t)). \tag{6.46g}$$

The objective function consists of the gain term (6.46a) (see equation 6.44), as well as a penalty term (6.46b) describing the cost of state trajectory segment $\xi_{[t_n,t_{n+1}]}$. This cost can, for example, be directly related to the segment length, the time to reach the goal pose $\xi(t_{n+1})$, or the energy required to execute the trajectory. The dynamic system is described by the initial value problem (6.46c), together with dynamic constraints (6.46d), and a control function $u(t) : \mathbb{R} \to \mathbb{R}^{n_u}$. A constraint on the minimum overlap of images with previous recordings is given by (6.46e). This is only active after taking the first image and further encourages the NBV selection to first observe a surface sufficiently locally. We further introduced additional geometric constraints on the minimum/maximum distance to the geometries surface (6.46f), as well as a minimum distance to potentially unknown space (6.46g).

Problem (6.46) is now solved iteratively for $n = \{0, 1, \ldots\}$. After each iteration, the quality estimate of all points in FOV$[\xi_{n+1}]$ is updated according to (6.27). The algorithm terminates once $\overline{\text{gain}}(p, \Xi_n, \xi(t_{n+1}))$ is zero for all points p and admissible path segments $\xi_{[t_n,t_{n+1}]}$. Then full coverage is achieved up to geometric and dynamic limitations. Note that the total number of images obtained from this greedy procedure is not optimal. As mentioned in section 1.2.3 such optimality can be achieved using an RRT resampling scheme (e.g. [Bircher et al., 2017], [Papadopoulos et al., 2013], [Englot and Hover, 2013]). Obtaining approximate solutions to (6.46) numerically is the subject of the next chapter.

SfM-NBV Algorithm

7

In the previous chapter we derived an estimator for the expected SfM reconstruction quality (6.27). It is subsequently used in an OED problem (6.46) in order to obtain optimal NBV trajectories. While theoretically solving the task of computing satisfying recording poses for a SfM reconstruction, the OED problem is still too hard to solve for various reasons:

- The gain function (6.44) used inside the objective function (6.46a) of the OED problem requires information on the current estimate of the reconstruction quality $\Sigma^{-1}(p, \Xi_n)$, which is computed from (6.27). Even though the true position of a point on the surface of a geometry is unknown, the measure for the information Θ^{-1} (see equation (6.14)) gained from an observation still depends on it. By construction, the axis of the elliptic confidence cylinder described by Θ^{-1} must point in the direction of said point. This makes sense as depth can not be extracted from single images. The ellipses obtained from right sections of this cylinder also depend on the orthogonal distance of the point to the image plane. This is a direct result of the reprojection of a fixed size confidence ellipse on the image plane into world space in the derivation of Θ^{-1}.

- Furthermore, prior knowledge of the geometry is also required to determine observability. The integral in the objective function of the OED problem (6.46a) depends on the set of all surface points that are observable from a recording pose $\mathrm{FOV}(\xi_{n+1})$. To compute this set, we need to be able to differentiate between visible and occluded points. Therefore, a representation of the set of surface manifolds is required as prior information.

- Additionally, $\mathrm{FOV}[\xi(t_{n+1})]$ is an infinite set containing continuous subsets of points on the surface geometry. Since the set of manifolds that describes the geometries surface can usually not be parameterized, the integral (6.46a) can only be evaluated numerically. This means we need to determine a finite discrete set of points that allows us to approximate this integral with a sum.

- Most of the constraints in the OED problem have infinite dimensionality. Hence, they must hold for an infinite number of points on the whole time interval $[t_n, t_{n+1}]$. The path cost term (6.46b) depends on the continuous cost function and can be obtained from analytic integration in some cases, otherwise numeric integration must be used. The differential constraints (6.46c) and (6.46d), as well as the geometric constraints (6.46f) and (6.46g), are also infinite dimensional. Solving similar systems with different objective functions

is already well studied in the context of optimal control[8]. Since our focus lies on the evaluation of the quality estimate and NBV planning, this point will not receive much attention. In chapter 10, we will use a directly positional and orientational controlled camera without differential constraints. The path costs will then relate to the length of line segments between successive views.

In order to obtain an approximate solution for the OED problem (6.46) nonetheless, we will use the standard direct approach commonly employed in numerics: first discretize, then optimize in order to reduce the infinite to a finite dimensional problem.

We will first discuss some types of required prior knowledge of the surface geometry and their respective discretization in section 7.1. The full, implementation-ready algorithm is given in section 7.2. Finally, an additional correction step is introduced to further enhance the NBV solution.

7.1 Surface Geometry Discretization

For the reasons stated above, a dense representation of the geometries' surface is required as prior knowledge. Since this representation comes from previous measurements or geometric approximations, it is always subject to some error. As a result, a discrepancy between true and estimated observability is introduced. Surface points that may be visible from a camera configuration in reality, may be occluded in the geometric approximation and vice-versa. We call this error the *observability error*.

The surface integral (6.46a) will now be approximated from a finite set of points obtained from this surface approximation. For performance reasons, it is desirable that this set is uniquely determined from a collection of points that does not change during the runtime of the algorithm. This way, we can associate each point with a quality estimate that can simply be updated by (6.27) when inserting new images. Otherwise, the quality estimate would have to be recomputed from all previous recording poses each time a point is observed for every evaluation of the gain function (6.44). We will call this discrete set of points *control points*. The finite set of control points which are visible from a pose in the geometric approximation is denoted by

$$\overline{\mathrm{FOV}} : \xi \in SO(3) \times \mathbb{R}^3 \mapsto \overline{\mathrm{FOV}}(\xi) \quad \text{(finite set)}. \tag{7.1}$$

Since we are only able to use a geometric approximation, control points are usually not exactly on the surface of the real geometry. However, this poses no problem, as the quality of theoretical points in space can also be estimated by (6.27). We then assume that the estimate for the expected SfM reconstruction quality does not change significantly in a small, local area around those control points. The discretization into control points introduces another source of error, which we will call the *discretization error*. It summarizes the effects of the deviation of control points from the real surface and errors introduced due to their local approximations.

[8] A common approach to solve optimal control problems (for example, see [Betts, 2010]) is to discretize the control functions (e.g. piecewise linear), use numerical integration (e.g. RKF45) to solve the initial value problem and then apply an SQP method to find optimal control functions. There, constraints are usually discretised and only evaluated at grid points. Alternatively, a control function sampling RRT (e.g. [Kuwata et al., 2009]) can be used.

By assuming that control points sample the surface evenly, the gain term (6.46a) in the objective function of the OED problem can be replaced by the sum

$$w \cdot \sum_{p \in \overline{\mathrm{FOV}}[\xi(t_{n+1})]} \overline{\mathrm{gain}}\left(p, \Xi_n, \xi(t_{n+1})\right), \tag{7.2}$$

where $w \in \mathbb{R}_{>0}$ is a constant relating to the average mutual distance of control points. Demanding an even distribution is important here, as otherwise areas with higher point densities would be weighted more. This would result in higher objective function values for views that observe high point density areas. According to the integral (6.46a), this should instead be the case when observing large surface areas. Hence, an uneven control point density would manipulate the NBV selection in the OED problem. If control points are chosen arbitrarily, weights corresponding to local point densities would be required for each term in (7.2).

The combination of discretization and observability errors poses another problem. Consider the case where a control point is assumed to be observable, but in reality the associated surface geometry is actually occluded. When inserting a new image at this position, the estimate of the expected reconstruction quality of that control point may get updated, although this image does not provide information on the real geometry in that area. There, the true reconstruction error will then be higher than the estimated one. This effect will be called *overestimation*.

Now we have reached a paradoxical point. In order to be able to collect data that suffices to build a 3D reconstruction, a 3D representation is already required to be known in advance. However, as we will see later, this is only a minor inconvenience. The prior knowledge can be a very coarse approximation, while the resulting model will have a very high resolution and accuracy. We will also present an approach in section 7.1.3 that does not require any prior information on the geometry, by combining our NBV planning with an autonomous exploration approach in an occupancy map. On the other hand, there are advantages if the complete geometry is already known in advance. Then the full NBV trajectory can be precomputed and planned offline. Such a path can further be refined using RRT resampling strategies (see section 1.2.3) to obtain truly optimal trajectories.

We will now give a few examples of how the surface geometry can be approximated. For each of them, we will also propose a possible choice of control points and explain how the set $\overline{\mathrm{FOV}}(\xi)$ is efficiently obtained.

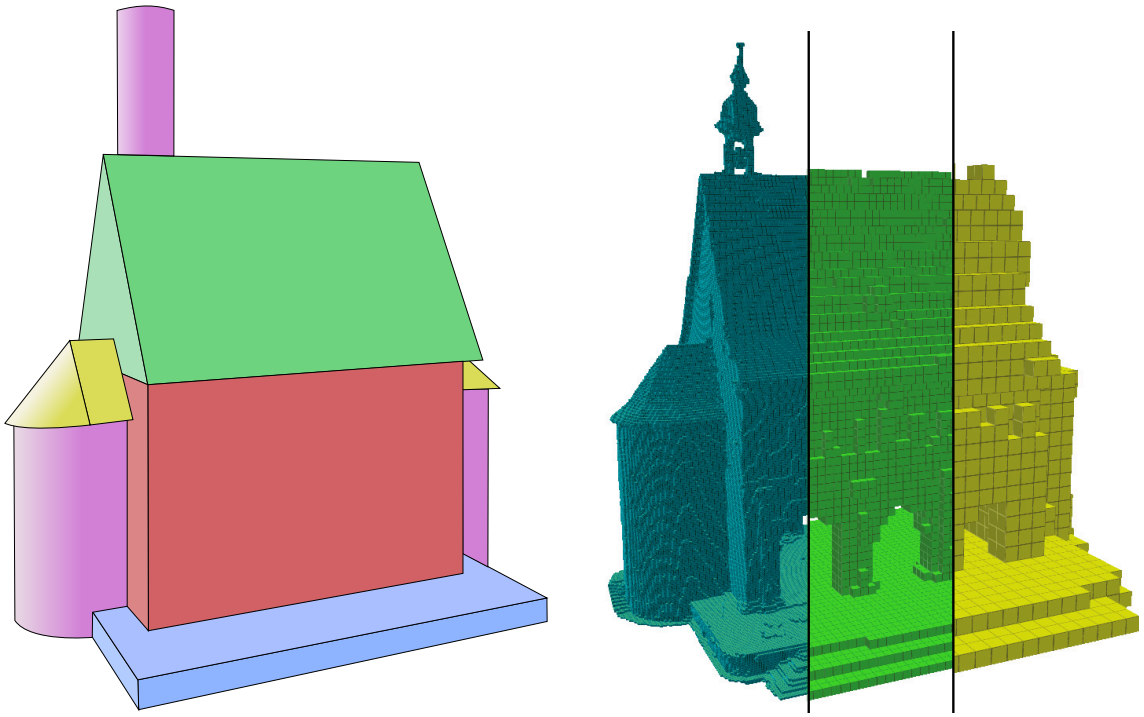
7.1.1 3D Model

Sometimes a full 3D mesh is already available. This may be the case, for example, in construction research or for inspection tasks, where the same structure is recorded at different points in time. Depending on the accuracy of the mesh, the observability error can be negligible. One way to obtain evenly distributed control points is to resample the mesh with equilateral triangular faces of arbitrary, but similar size as in [Pietroni et al., 2010]. Then the set of control points is chosen as vertices of the new mesh. Alternatively, the entire surface area can be sampled uniformly. By selecting many sample points, the discretization error becomes smaller but leads to higher computational costs when evaluating the gain function 7.2. The set $\overline{\mathrm{FOV}}(\xi)$ is computed according to standard ray tracing approaches, which can also be heavily parallelized (e.g. with GPU acceleration).

7.1.2 Primitive Hulls

The object's surface can be approximated by (hand crafted) geometric primitives. For instance, a building can be represented by cubes, cylinders, cones and prisms. An example of primitive hull approximation is given in figure 7.1a. As for the 3D model approach, the set of control points can be obtained by resampling the primitive mesh or by drawing uniform samples. Observability is also computed in the same way through ray tracing.

Compared to the other methods, the observability and discretization errors are usually the largest here, since they depend on the accuracy and fidelity of the primitive hull approximation. Constructing these hulls is often cumbersome, since scale and proportions need to be known in advance and require additional measurements.



(a) Primitive hull. **(b)** Voxels (multiple resolutions).

Figure 7.1: Visualization of different surface geometry approximations using the example of the King's Hall of Lorsch Abbey (UNESCO World Heritage Site).

7.1.3 Voxel Discretization

The surface geometry can also be approximated using a voxel representation. Space is tesselated into cubes of equal size. All voxels that presumably contain segments of the surface geometry are marked as occupied, while others are labeled as free space. The control points are then selected as the centers of all occupied voxels. Due to the grid structure, those voxel representations can be efficiently stored in octrees (see section 4.2).

The set $\overline{\mathrm{FOV}}(\xi)$ is obtained by collecting all occupied voxels that are visible from configuration ξ, i.e. not occluded by other occupied voxel volumes. A voxel is classified as visible if it lies inside the camera frustum and the intersection point of the ray from the camera center to the voxel center with the voxel surface is

observable. This makes sense because it is equivalent to the control point being visible in the grid map if we count the corresponding voxel as transparent. Details on a cost-efficient implementation of the required ray casting task inside octree structures are given in section 8.3.2. Since voxels describe cubic volumes, the real surface geometry within occupied voxels can be arbitrarily complex. Approximating it with a cubic volume therefore introduces an observability error, because parts of the true geometry may be occluded even if the voxel is visible.

For a voxel discretization, overestimation is a noticeable issue. Consider the case where a voxel is placed on a sharp edge of an object. Then it is visible from various sides around the edge, even though the associated camera poses each only contain information about a single side that connects to that edge. This effect is also observed later in our simulations in chapter 10. The discretization error, observability error and effects of overestimation are all proportional to the voxel size.

The voxel surface approximation is illustrated in figure 7.1b. A visualization of the discrete gain term (7.2) for NBV selection using voxel discretization is given in figure 7.2.

Combination with Autonomous Exploration

The main benefit of the voxel discretization is that it can be used in an autonomous exploration framework. This is significant because then no prior information about the surface geometry is required. Here, an additional sensor is necessary, that greedily builds a 3D occupancy map (see chapter 4), which is also used as voxel discretization for NBV planning. The OED problem (6.46) can easily be converted to an OCP[9] that is used for autonomous exploration. For that purpose, the image overlap constraint (6.46e) is removed and an exploration objective function term is introduced, replacing the gain integral (6.46a). This new exploration term could, for example, count the number of unknown voxels inside the field of view of a camera pose. Other possible autonomous exploration objective functions are given in [Delmerico et al., 2018]. An example application of this autonomous exploration framework using UAVs is given in [Bircher et al., 2018]. The exploration and NBV objective function are then evaluated simultaneously and the task (exploration or recording an image) corresponding to the higher-value objective function is executed. As the map becomes more explored, the quality estimates of newly discovered, occupied voxels need to be updated according to previous recording poses.

The additional advantage of using an occupancy map is that free, occupied and unknown space can be distinguished. Together with the octree structure, this enables for cost-efficient collision checking and, in consequence, fast collision-free path planning. If our NBV planning is run simultaneously with an autonomous exploration framework, no global optimal coverage trajectories can be computed, because the full geometry is not known during runtime. Here, however, it is possible to run the entire approach online without the need for an excessive amount of preparatory work in order to obtain prior information on the geometry.

[9] By removing the gain term we do not have an OED objective function anymore.

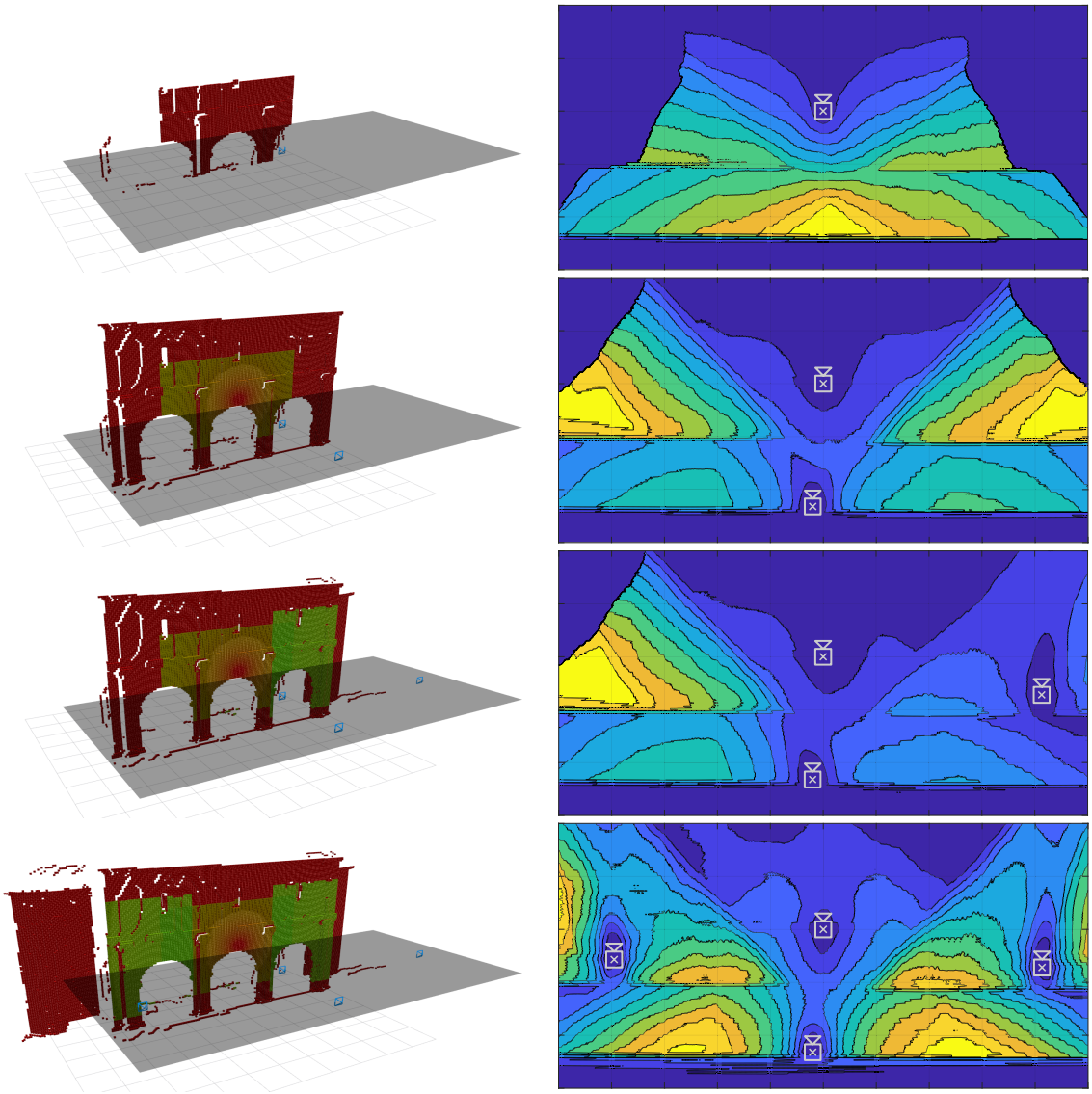


Figure 7.2: Iterative gain selection (top to bottom) using a voxel discretization at the example of the King's Hall of Lorsch Abbey (UNESCO World Heritage Site). The camera position is constrained to the gray plane with fixed orientation and camera far plane at 5 m. The color in the left column correspond to the largest eigenvalue of the estimator covariance matrices for each voxel (red = large, green = small). For each iteration, the objective function (7.2) is sampled in the gray plane and visualized as contour plot in the right column (with different colormap scalings). The next viewpoint is inserted at the maximum of the objective function.
Source: [Lindner et al., 2019] ©2019 IEEE.

7.2 Pseudo-Code

Now the complete iterative NBV planning algorithm can be formulated. The pseudo-code of the main loop is given in algorithm 7.1. Here, approximate solutions to the OED problem (6.46) are generated by the `getNextSegment` function (see algorithm 7.2). These NBV paths are then executed. If the algorithm is run online, an image is recorded at the end of each trajectory. This step is skipped if the NBV trajectories are to be precomputed. Finally, the estimator updates the estimate of the expected

Algorithm 7.1: SfM-CPP pseudo-code.

```
1  Ξ_0 ← {}; n ← 0 // initialization
2  do // main loop
3
4     // execute NBV trajectory
5     ξ|_[t_n,t_{n+1}], gain ← getNextSegment(Ξ_n)
6     execute trajectory ξ|_[t_n,t_{n+1}]
7     insert image at ξ_{n+1}
8
9     // update precision matrices (6.27)
10    for p ∈ FOV[ξ_{n+1}]
11       Σ^{-1}(p, Ξ_{n+1}) ← Σ^{-1}(p, Ξ_n) + Θ^{-1}(p, ξ_{n+1})
12    end for
13    Ξ_{n+1} ← Ξ_n ∪ {ξ_{n+1}}; n ← n + 1
14
15 while gain > threshold
```

reconstruction quality for each control point that is visible from the new camera location. The algorithm terminates, once the gain term obtained from (7.2) is below a threshold value. This means that no relevant improvement to a subsequent SfM reconstruction can be achieved from additional measurements. Then gain clamping to $\lambda_{\min}$ is active for almost all observable control points, i.e. a desired estimator quality is reached. This threshold is rather important. In some special cases, a control point may be very hard to observe, such that a huge amount of images would be required to sufficiently capture that single point, each of which contributing little new information to the reconstruction. Therefore, the threshold acts as a safeguard that ignores these points and keeps the number of images reasonable.

The computation of a single NBV segment is realized in algorithm 7.2, which is explained below. First, we will state the final discretized version of the OED problem (6.46):

$$\max_{\xi(\cdot),u(\cdot),t_{n+1}} \quad \begin{cases} w \cdot \sum_{p \in \overline{\mathrm{FOV}}[\xi(t_{n+1})]} \overline{\mathrm{gain}}\,(p, \Xi_n, \xi(t_{n+1})) & \text{if } o_{\min} \leq g_o(\xi(t_{n+1})) \\ 0 & \text{if } o_{\min} > g_o(\xi(t_{n+1})) \end{cases} \tag{7.3a}$$

$$- \int_{t_n}^{t_{n+1}} \mathrm{cost}(\xi(\tau))\, d\tau \tag{7.3b}$$

$$\textbf{s.t.} \quad \ddot{\xi}(t) = f(t, \xi(t), u(t)), \quad \xi(t_n) = \xi_n, \quad \dot{\xi}(t_n) = \dot{\xi}_n \tag{7.3c}$$

$$g(t, \xi(t), u(t)) \leq 0 \tag{7.3d}$$

$$d^{\mathrm{o}}_{\min} \leq d_o(\xi(t)) \leq d^{\mathrm{o}}_{\max} \tag{7.3e}$$

$$d^{\mathrm{u}}_{\min} \leq d_u(\xi(t)). \tag{7.3f}$$

Here, we use the control point discretization from section 7.1 that leads to the

Algorithm 7.2: getNextSegment(Ξ_n) pseudo-code.

```
function ξ|_[t_n,t_{n+1}], gain = getNextSegment(Ξ_n)

    // compute L RRT candidates
    for i = 1 : L
        ξ^i|_[t_n,t_{n+1}] ← random RRT path-segment satisfying
                             dynamic constraints (7.3c, 7.3d) and
                             geometric constraints (7.3e, 7.3f)
        cost^i = ‖ξ^i|_[t_n,t_{n+1}]‖    // path cost (7.3b)

        // gain computation (7.3a)
        gain^i = 0
        for p ∈ FOV[ξ^i_{n+1}]
            gain^i += w · gain(p, Ξ_n, ξ^i_{n+1})
        end for

        // check image overlap (7.3a)
        if (n > 0) and (o_min > g_o(ξ^i_{n+1}))
            gain^i = 0
        end if
    end for

    // return segment with biggest objective function
    k = arg max_i  gain^i − cost^i
    return ξ^k|_[t_n,t_{n+1}],  gain^k
end function
```

discretized gain term (7.2). Note that the image overlap constraint (6.46e) is additionally incorporated into the objective function. The overlap function $g_o(\xi(t_{n+1}))$ is now defined as the fraction of points from the set $\overline{\text{FOV}}[\xi(t_{n+1})]$ that have been observed before, i.e. with precision estimates $\Sigma^{-1}(p, \Xi_n)$ not equal to zero. The case distinction in (7.3a) is not made for the first recording pose ($n = 0$), since it cannot overlap with other measurements. Although this modification of the objective function worsens its smoothness properties, it will ultimately not be an issue.

As we have seen in figure 7.2, the gain objective function term (7.2) has discontinuities and many local minima, despite it being a very basic example. This already makes the OED problem difficult to solve using gradient based numerical methods. In the end, however, we are not interested in optimal NBV solutions – as these will not result in global optimality anyways – but in good recording poses that yield much information for the reconstruction process. In addition, computing these exact

solutions would require a large amount of runtime. Therefore, we employ an RRT sampling-based approach that approximates good NBV trajectories that satisfy all constraints and yield high objective functions values. This is realized in line 5 of algorithm 7.2, where in total $L \in \mathbb{N}$ valid samples are drawn. For dynamic systems, trajectories are obtained by control space sampling (e.g. [Kuwata et al., 2009]), while standard RRT and RRT* (e.g. [Karaman and Frazzoli, 2011]) approaches can be utilized otherwise. Samples are considered valid and are accepted if they satisfy constraints (7.3c) - (7.3f). Continuing with algorithm 7.2, the gain and cost terms are evaluated for each admissible RRT trajectory. The path corresponding to the highest value objective function is returned.

In optimization problem (7.3), dynamic and geometric constraints (7.3c - 7.3f) are completely separated from the task, which is now solely defined by the objective function. This allows us to simultaneously evaluate multiple objective functions with identical constraints, each relating to a different task, with the same RRT samples. Depending on the values of each objective function the corresponding action (NBV recording, exploration, ...) can be executed. As elaborated in section 7.1.3, such a task can be, for example, related to autonomous exploration. Then exploration and NBV planning can run in parallel and online on an autonomous robot.

While we stated that gradient based approaches are not suitable to obtain solutions to the OED problem 7.3, a combined approach could be realized. For that purpose good NBV trajectories are computed first, using the RRT sampling scheme. Subsequent Newton correction steps can then be used to improve the estimate and ultimately converge to a local minimum of the OED problem. Among other things, this requires derivatives of the objective function (7.3a), thus also derivatives of the gain function (6.44). These can be derived analytically (see appendix B) and depend on the eigensystems of the precision matrix quality estimates. However, this approach was not further analyzed.

Implementation Details

8

The pseudo code for our greedy NBV algorithm is given in algorithms 7.1 and 7.2. These still depend on the choice of the type of voxel discretization (see section 7.1). Furthermore, the actual implementation of the $\overline{\text{FOV}}(\xi)$ function (see equation (7.1)) proves to be difficult because it involves visibility checks. It is used both in the main loop of the RRT sample generation as well as in the update step for the precision matrices and will turn out to be the most expensive sub-task, which dominates runtime and therefore requires special attention and efficient implementation. In order to achieve high performance, C++ is used as the programming language of choice. We will now cover some implementation details.

Due to its versatility, we have adopted a voxel discretization strategy (section 7.1.3) using version 1.9 of the OctoMap library [Hornung et al., 2013]. The octree structure used for storing the voxel map gives us many advantages over other discretization methods. As will be shown, it allows for computational fast visibility checks such that the set $\overline{\text{FOV}}(\xi)$ can be determined efficiently. In addition, collision checks can be implemented cost-effectively as part of collision-free path planning. Furthermore, the voxel discretization allows for easy integration into a combined autonomous exploration framework, as described in section 7.2, and different discretization accuracies.

This chapter is structured as follows. We give a brief overview of the octree implementation in the OctoMap library in section 8.1. Based on this library, a custom gain-octree structure is built in section 8.2 that contains information on the estimate of the expected SfM reconstruction quality in each occupied voxel. Extensions to the OctoMap library that result in better runtime performance are explained in section 8.3. Finally, in section 8.4, the runtime of the algorithm is discussed. Note that the supplied C++ snippets may have been simplified for better understanding.

8.1 OctoMap Overview

The OctoMap library (see [Hornung et al., 2013]) provides an octree structure for a 3D occupancy grid mapping approach. Some aspects of this library have already been described in section 4.2. There, our focus lied on its occupancy mapping properties. Now we will give some details about the implementation of the data structure and its use as an efficient voxel map:

- **Memory Layout**
  The octree is stored as a singly linked tree of maximum depth 16, with the root node at depth 0. Each tree node contains a single pointer to an array of

pointers to 8 potential child nodes. If the node is a leaf then the pointer to the array is a null pointer. Similarly, any of the pointers in the array of pointers can be a null pointer, meaning that the respective child node does not exist. Hence, the memory usage for a single node is either 8 bytes if it is a leaf, or $8 + 8 \cdot 8$ bytes if at least one child is available[10]. Furthermore, each node can hold additional data.

- **Spatial Relation**
  Each node represents a space described by a cube. Subdividing this cube into eight equally sized child cubes (see figure 4.2) then defines the cubes corresponding to all potential child nodes. The position of the cube corresponding to a child node relative to its parent is uniquely determined by the position of the child node pointer in the array of pointers. Hence, its position within the parent node can be described by 3 bits (x, y, z positions), and the position of each cube at level 16 relative to the root cube is uniquely determined by a $3 \cdot 16$ bit key. The side length of a voxel at level 16 is denoted as resolution r. Since the root node defines a bounding box, the entire space described by the octree is a cube with side length $r \cdot 2^{16}$.

- **Tree Traversal**
  A node itself has no information about its spatial position. Consider the case where the position of a given node inside the octree has to be computed. Since nodes do not contain pointers to their parents, the entire tree must be searched recursively from the root node. This is very inefficient. Hence, the $3 \cdot 16$ bit key is used instead, as it describes the full path from the root node through the entire tree. Note that although child nodes do not have to exist, each node traversed on the path defined by the key also contains the target cube at level 16.

  A common task is to find a node that contains a specific Euclidean point. Such nodes can be obtained at any octree level by first converting the point's position to a key, which is then used to traverse the tree.

- **Tree Pruning**
  The OctoMap octrees can be pruned. This means that all children of a node can be deleted if a certain requirement is met. Usually this is the case when all children contain identical or similar data. Then this data is summarized inside their parent node, all child nodes are destroyed, and the node's pointer to the array of pointers is set to a null pointer. Pruning is extremely important in the context of voxel maps. In this spatial representation, the majority of space is usually free and can be heavily pruned. Therefore, it is possible to represent large areas with relatively little memory.

One example of nodes inside the OctoMap library is given by the class `OcTreeNode`. Besides pointers to potential child nodes, it also holds a float representing an occupancy value. This value is clamped to a minimum and maximum value. Based on thresholds for the occupancy value, nodes are classified as free or occupied space. Moreover, if a node has at least one child, then all other child null pointers correspond to unknown space. Nodes can be pruned if all 8 of their children exist and

[10] On a 64 bit system.

have identical occupancy value. The occupancy of a cube can only be modified at tree level 16. Then either new child nodes need to be created or occupancy values of old ones are updated. Occupancy values propagate through the whole tree. Starting from the leaf nodes, all parents receive the maximum of their children's occupancy values. This especially implies that if a node at any tree level is classified as free, all of its children must also correspond to free or unknown space.

8.2 Gain-Octree

We introduce a new octree type called `GainOcTree`. The nodes in this tree inherit from the class `OcTreeNode` (described above) and is given by

```
1 class GainOcTreeNode : public OcTreeNode {
2     ...
3 private:
4     std::array<float, 6>* prec_mat;
5 };
```
. (8.1)

The corresponding tree `GainOcTree` is used for voxel discretization as described in section 7.1.3. We associate the voxel discretization points with the centers of each occupied node at level 16. The expected reconstruction quality of these control points is encoded in a precision matrix (see section 6.2), which is stored in the tree nodes. Since precision matrices are symmetrical, only 6 additional floats are necessary to encode this information. However, we can not distinguish directly between node types when allocating the required memory. This would lead to a large memory overhead, since nodes classified as free or at tree levels other than 16 would also allocate 6 floats, despite them not corresponding to control points. Hence, the pointer `prec_mat` (see (8.1)) is added to each node instead. Then new memory is allocated for the pointer if and only if its node corresponds to a control point (occupied, tree level 16) and is a null pointer otherwise. All floats in the precision matrices are initialized with 0. The total memory usage of a single `GainOcTreeNode` is given in table 8.1.

Table 8.1: Memory usage of a single `GainOcTreeNode`. The occupancy value (float, 4 bytes) is present in all nodes. The pointer (8 bytes) to the array of child nodes is a null pointer if no children exist. The pointer (8 bytes) to the precision matrix (6 floats, 24 bytes) is a null pointer if the node is no control point.

has child nodes	is control point	memory usage
false	false	20 bytes
false	true	44 bytes
true	false	84 bytes
true	true	does not exist

Control points are associated with occupied nodes at level 16. Therefore, those nodes must not be pruned. This behavior is realized in the `GainOcTreeNode` class: pruning is prohibited for all occupied nodes, while free and unknown ones are pruned identical as in the parent class `OcTreeNode`. Theoretically, the largest possible memory consumption of a `GainOcTree` is 15 763 terabytes and is reached in a fully expanded tree where all nodes at level 16 are occupied. In reality, free space dominates

the octree and can efficiently pruned. Thus, usually only up to several hundred megabytes of memory are used.

In the clamped gain formulation (6.44), the eigenvalues of the precision matrices need to be computed. For general square matrices, eigenvalues are usually obtained from iterative methods or matrix decompositions. Depending on the discretization of the geometries' surface and the camera pose, each RRT sample in algorithm 7.2 may require the evaluation of thousands of clamped gain function terms. Hence, the beforehand mentioned eigenvalue computation approaches becomes computationally too expensive. Fortunately, our precision matrices are only of size 3×3, symmetrical and positive definite. This special structure can be exploited to efficiently obtain an analytic solution to the eigenvalue problem (see [Kopp, 2008]). For more details on our implementation of the eigenvalue computation, please refer to appendix C.

The `GainOcTree` class is given as

```
1 class GainOcTree : public OccupancyOcTreeLeveledBase<GainOcTreeNode>
2     {...};
```
. (8.2)

The `OccupancyOcTreeLeveledBase` class is the subject of the following section.

8.3 Leveled Octree

The templated class `OccupancyOcTreeLeveledBase<NODE>` is an extension to the OctoMap class `OccupancyOcTreeBase<NODE>`. There, the octree structure is further exploited to increase runtime performance. This is achieved through extensive use of different tree levels, wherefrom the class's name originates. Compared to the original OctoMap implementation, computationally demanding functions such as ray-casting, generic filtering, and mapping tasks receive a considerable speedup.

This section is divided into three parts. The first two cover recursive depth-first tree traversal approaches. An important application of this will be the computation of the set $\overline{\mathrm{FOV}}(\xi)$ (see equation (7.1)). First, all occupied voxels that are inside the camera frustum are collected using a depth-first filter. Their visibility from the camera pose is subsequently checked using a depth-first ray casting method. An extension to occupancy grid mapping using our new tree class is given in the final section. In contrast to the corresponding OctoMap implementation, we allow updates of occupied and free nodes at different levels. This may be used in combination with an autonomous exploration framework in the future.

8.3.1 Filter Function

We introduce a `filterNodes` function that iterates all nodes of the octree in a depth-first manner. It is given by

```
1 template<class NODE>
2 template<typename FCN, typename... Args>
3 void OccupancyOcTreeLeveledBase<NODE>::filterNodes
      (const FCN& filter, Args && ... args);
```
. (8.3)

The `filter` input must be a function that matches the signature

```
1 bool filter(NODE* node, const OcTreeKey& key,
      unsigned int depth, Args && ... args)
```
(8.4)

and defines the operations that are performed for each node in the recursion. The inputs `node`, `key` and `depth` correspond to each voxel that is iterated through by the `filterNodes` function. The Boolean return value controls the recursion. If true is returned, the depth-first iteration continues. Otherwise, if false is returned, the current tree branch is skipped. Skipping branches early results in a fast termination of the recursion. The parameter pack `args` describes an arbitrary amount of additional input arguments of arbitrary type. Those additional parameters are not used inside the `filterNodes` function (8.3) directly, but are only forwarded to the `filter` function (8.4) using perfect forwarding[11]

```
1 std::forward<Args>(args)...
```
. (8.5)

Note that the parameter pack can also contain reference types that may be used to obtain additional outputs. It can also be empty. Then the `filter` function (8.4) is utilized to modify the iterated nodes directly. By using a template for the type of the function argument and a variadic template for the parameter pack in (8.3), better runtime performance is achieved. Each call to `filterNodes` with different `filter` functions results in a new template instance. Since this happens during compile time, the compiler can heavily optimize the code for the specific filter functions (e.g. by inlining).

We will now give a few examples of how the `filterNodes` function (8.4) is used in our implementation.

Get Voxel in Field of View

All occupied nodes on tree level 16 with centers inside a given viewing frustum are to be determined. For an efficient implementation as a filter using the `filterNodes` function (8.4), branches that do not contain relevant voxels need to be discarded early in the depth-first recursion. Consider the case from figure 8.1, where a voxel center is barely inside the camera frustum. In order to have potential child nodes

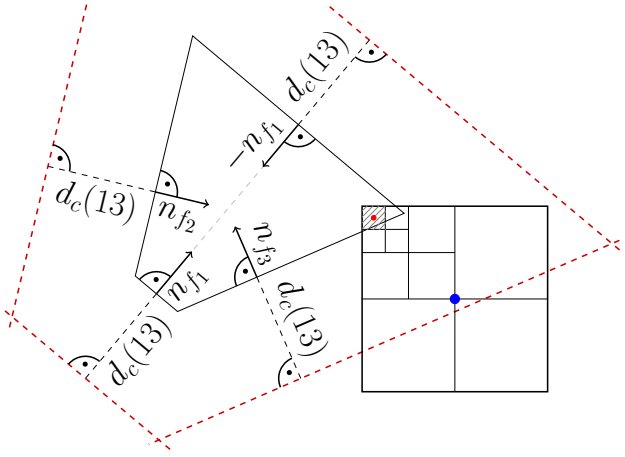


Figure 8.1: Maximum allowed distance of voxel centers to the camera frustum. In order for the red dot (voxel center of node on lowest tree level) to be inside the frustum, all parent node centers (e.g. blue) must be within a certain distance $d_c(13)$ (see equation (8.6)) to all frustum planes. The vectors $n_{f_1}, n_{f_2}, n_{f_3}$ denote the frustum normals.

inside the FOV, all parent nodes' centers can only have a maximum signed distance

[11] This means that all arguments are passed to a function as if it had been called directly (e.g. not within a wrapper function) with those arguments. This allows us to preserve the arguments value category (lvalue, rvalue, ...) and modifiers (const, volatile, ...). Furthermore, possible intermediate copy operations are omitted.

to the frustum planes. Here, the direction of the frustum plane normals, which are pointing inwards, define the side with positive sign. The maximum absolute value of this distance can be identified to be the length of a cube diagonal, from the center of any parent node at arbitrary depth to the center of one of its corner voxel on tree level 16. This length can be calculated as

$$d_c(\text{depth}) = \left(2^{16-\text{depth}} - 1\right) \frac{\sqrt{3}}{2} r, \quad \text{depth} \in \{0, 1, \ldots, 16\}. \tag{8.6}$$

Since d_c only depends on the depth and the voxel resolution r, all 17 possible values can be precomputed. In our implementation they are stored inside the `OccupancyOcTreeLeveledBase<NODE>` class as

```
1 std::array<float, 17> range;
```
. (8.7)

An additional function is required that determines if a point is within a range of the frustum. We introduce a camera class

```
1  class Camera {
2  public:
3     bool isInRangeOfFOV
4        (const octomap::point3d& point, float range) const;
5     ...
6  private:
7     octomap::point3d center;       // camera center
8     octomath::Vector3 view_dir;    // camera view direction
9     float near, far;               // distance to near and far plane
10    std::array<octomath::Vector3, 4> normals; // frustum side normals
11 };
```
(8.8)

that contains information on its frustum's geometry. The desired function, called `isInRangeOfFOV`, is given in algorithm 8.1. There, the distance to each frustum plane is computed separately. The dot product is utilized to obtain the length of the orthogonal projection of a point onto the frustum normals.

Algorithm 8.1: Code of the `isInRangeOfFOV` function that checks if a point is within a distance to a given viewing frustum.

```
1  bool Camera::isInRangeOfFOV(const octomap::point3d& point, float range) const {
2
3     const octomath::Vector3 camToPoint = point - this->center;
4
5     // check near / far planes
6     float viewDirProjection = static_cast<float>(camToPoint.dot(this->view_dir));
7     if ( !(viewDirProjection >= - range + this->near) ||
8          !(viewDirProjection <= + range + this->far) ) {
9        return false;
10    }
11
12    // check frustum side planes
13    for (const octomath::Vector3& normal : this->normals) {
14       if (!(static_cast<float>(camToPoint.dot(normal)) >= -range)) {
15          return false;
16       }
17    }
18
19    return true;
20 }
```

Finally, the full procedure can be assembled using the `filterNodes` function (8.3) and a lambda expression. The function that determines all keys of occupied voxels at tree level 16 that are within a camera frustum is given in algorithm 8.2. Note that especially all free nodes can be skipped since all of their children must

Algorithm 8.2: Code of the `getVoxelFOV` function which determines all occupied voxels at tree level 16 inside a viewing frustum given by `camera`.

```
1  float GainOcTree::getVoxelFOV(const Camera& camera,
2     std::vector<octomap::OcTreeKey>& keyVec) const {
3
4     // initialize output
5     keyVec.resize(0);
6
7     // create filter-lambda
8     auto filter = [this, &camera](octomap::GainOcTreeNode* node,
9        const octomap::OcTreeKey& key, unsigned int depth,
10       std::vector<octomap::OcTreeKey>& out)
11    {
12
13       if (node && this->isNodeOccupied(node)) {
14          octomap::point3d point = this->keyToCoord(key, depth);
15
16          if (camera.isInRangeOfFOV(point, this->range[depth])) { // see alg. 8.1
17             if (depth == 16) {
18                out.emplace_back(key);
19                return false;
20             }
21             return true;
22          } else {
23             return false;
24          }
25       }
26
27       return false;
28    };
29
30    // apply filter
31    this->filterNodes(filter, keyVec);
32 }
```

also be unknown or classified as free. The result is returned in the call by reference parameter `keyVec` as a vector of octree keys (see section 8.1). Depending on the tree resolution, the size of this vector can get large. Dynamic arrays, such as vectors, can become a performance hit if used incorrectly. If they grow above a certain size, these containers require memory reallocation and a copy operation. To avoid this, memory for the `keyVec` vector should be allocated once. It can then be reused for subsequent calls to `getVoxelFOV`. The initial size can be set to the volume of the camera frustum divided by the volume of a cube at tree level 16, as an approximation of the maximum number of cubes that fit into the frustum.

Compute Distance

In order to compute the distance from a point to the next occupied node center at tree level 16, we proceed analogously to the previous chapter. Through similar geometrical considerations, an upper bound for this distance can be formulated using different tree levels. Consider a node at arbitrary tree level l that is currently being processed in the depth-first recursion of the `filterNodes` function (8.4). The distance of the point to this node's center can be computed easily. Then any of its child-nodes centers at tree depth 16 may have a maximum additional distance

of $d_c(l)$ (see equation (8.6)). Hence, we can reuse the `range` variable introduced in (8.7) and state the corresponding C++ implementation of this distance function in algorithm 8.3. The function returns the distance to the closest node center at level 16, as well as the corresponding voxel key in the call by reference argument `outkey`.

Algorithm 8.3: Code of the `distance` function that returns the distance from a point to the closest occupied voxel center at tree level 16.

```
template<class NODE>
float OccupancyOcTreeLeveledBase<NODE>::distance(const octomap::point3d& point,
     octomap::OcTreeKey& outkey) const {

   // upper bound on distance
   float distance_max = std::numeric_limits<float>::max();

   auto filter = [this, &distance_max, &point]
   (NODE* node, const octomap::OcTreeKey& key, unsigned int depth, octomap::
       OcTreeKey& outkey)
   {
      // fast return, only check occupied nodes
      if (!node || !this->isNodeOccupied(node)) {
         return false;
      }

      float distance = (point - this->keyToCoord(key, depth)).norm();

      if (distance > distance_max + this->range[depth]) {
         return false;
      } else {
         outkey = key;
         distance_max = std::min(distance_max, distance + this->range[depth]);
         return true;
      }
   };

   // apply filter
   this->filterNodes(filter, outkey);

   // substract radius -> distance to voxel-center
   return distance_max;
}
```

8.3.2 Ray Traversal

In the previous section, we discussed how to determine a set of nodes on tree level 16, where each node center lies inside a camera frustum defined by a pose $\xi \in SO(3) \times \mathbb{R}$. To derive its subset $\overline{\mathrm{FOV}}(\xi)$, the visibility of each control point (i.e. node center) inside the voxel map must be tested. A voxel at level 16 is called visible if a ray from the camera center to the voxel's center only passes space (at level 16) classified as free until it reaches the target voxel. In the OctoMap library the *digital differential analyzer* (DDA) algorithm [Amanatides and Woo, 1987] is implemented[12] to collect the keys of all voxels on level 16 that are passed by a ray iteratively. This is realized by parameterizing the ray as a line. Beginning at the starting point, it is iteratively traversed by computation of the next ray-voxel intersection (see figure 8.2). Depending on the voxel exit point, the neighboring voxel key is determined.

[12] See `octomap::OcTreeBaseImpl::computeRayKeys(...)` in the OctoMap library.

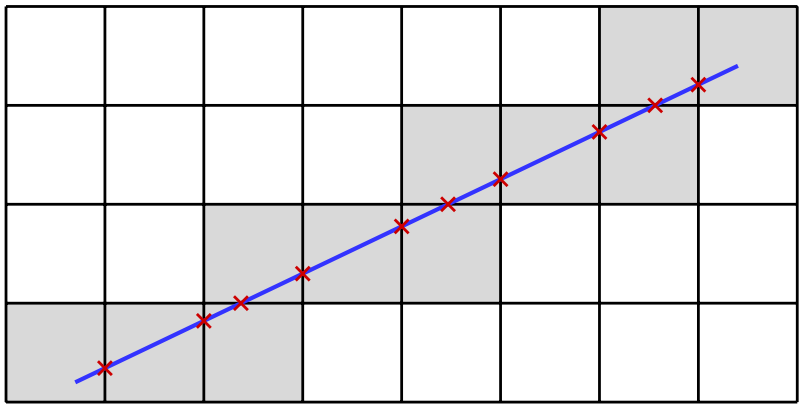


Figure 8.2: Visualization of the DDA ray iteration approach. The gray cells represent voxels traversed by a ray (blue line). The red crosses are intermediate intersections with voxels that are computed during the iterative process.

For our use case, the DDA algorithm poses some problems. It only operates on a single, fixed octree level (in our case level 16). This creates the following dilemma. Small voxel resolutions are desirable for an accurate discretization of the surface geometry (see section 7.1.3), leading to low discretization and observability errors. However, by decreasing the voxel size, the amount of voxels traversed by a ray increases rapidly, which proportionally affects the runtime of the algorithm. Since intersections are computed using floating point precision, round off errors accumulate faster and lead to an offset from the original ray. Furthermore, the DDA algorithm only returns keys. To determine voxel visibility, the occupancy values of nodes corresponding to the voxel keys must be queried. Obtaining a single node from a key requires an additional tree traversal from the root node to a leaf node.

In order to speed up this procedure, the structure of the octree can again be exploited, such that ray traversal can be realized at different tree levels. For that purpose we implemented the Spatial Measure for Accelerated Ray Tracing (SMART) algorithm proposed by [Spackman and Willis, 1991]. Their algorithm can be described as depth-first recursion inside an octree along a ray. It is particularly numerically stable and may even be maintained in integer space. Simplified it can be summarized as follows:

1. **Initialization:** Consider the starting point of the ray.
2. **Depth iteration:** The tree is traversed down to the point until a certain criterion is met.
3. **Ray iteration:** If this criterion does not lead to the termination of the recursion, we iterate in the direction of the ray to the largest possible tree-node neighbor. A theoretical reference point is now set at the intersection of the ray with the new node.
4. **Recursion:** Go to step 2.

An illustration of the SMART algorithm is given in figure 8.3. The concept of the SMART algorithm is very similar to the exhaustive `filterNodes` octree recursion (8.3) from section 8.3.1. Both perform depth-first iterations, where only the branch

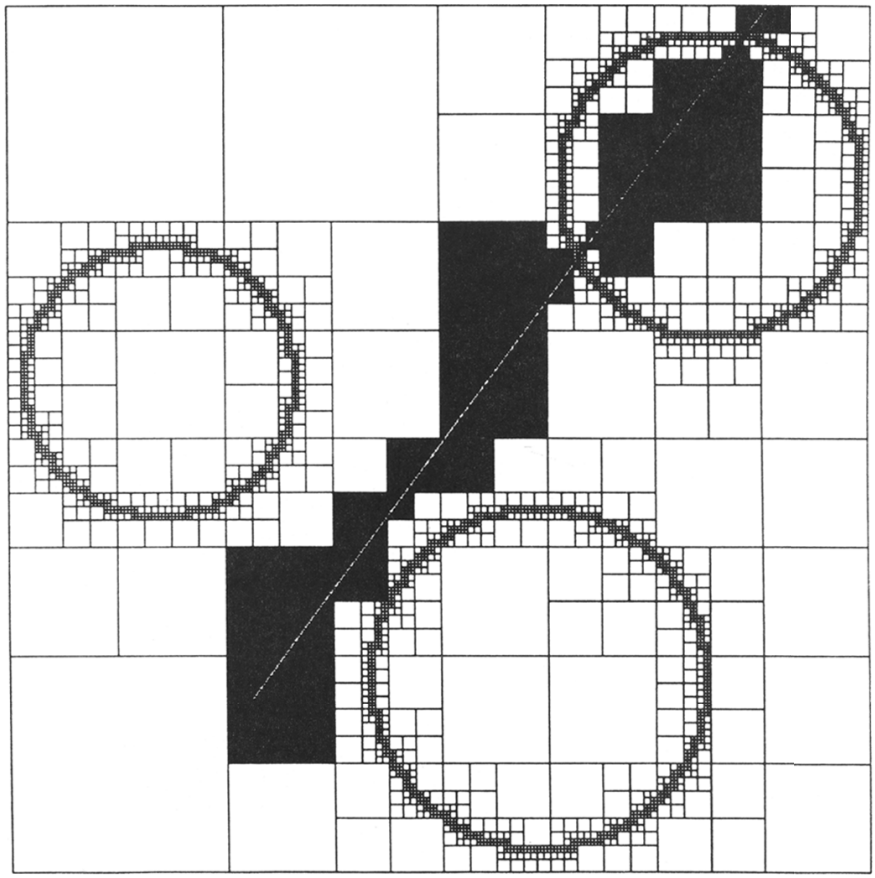


Figure 8.3: Visualization of the SMART algorithm by [Spackman and Willis, 1991] in a quadtree. The black squares correspond to the voxels that satisfy a criterion when iterating downwards, before continuing the iteration in the direction of the ray (white line).
Source: Reprinted from publication [Spackman and Willis, 1991], ©1991, with permission from Elsevier.

switching behavior differs. Hence, we implement the SMART algorithm with a similar function signature as (8.3) with

```
1 template<class NODE>
2 template<typename FCN, typename... Args>
3 bool OccupancyOcTreeLeveledBase<NODE>::rayTraverseLeveled(
      const point3d& origin, const point3d& end, const FCN&
      filter, Args && ... args);
```
. (8.9)

The points `origin` and `end` define the starting and endpoint of the ray. The `filter` must be a function with arguments identical to the `filter` function from (8.4), i.e.

```
1 RayTraverseAction filter(NODE* node, const OcTreeKey& key,
      unsigned int depth, Args && ... args);
```
. (8.10)

The parameter pack `args` is solely passed through to this `filter` function using perfect forwarding, identical to (8.5). These optional arguments can again be used to obtain additional output using call by reference. Unlike the filter function (8.4), this new one not only returns a boolean that defines the branching behavior. Two

additional return states are possible, which completely terminate the ray traversal and indicate success or failure. All possible values for `RayTraverseAction` are given in table 8.2. The `rayTraverseLeveled` function (8.9) returns true if the node

Table 8.2: Possible states of the `RayTraverseAction` type.

`RayTraverseAction`	**description**
`STOP_FAIL`	terminate recursion with failure
`STOP_SUCCESS`	terminate recursion with success
`EXIT_BRANCH`	continue ray iteration (next branch)
`CONTINUE_BRANCH`	continue depth iteration

corresponding to the endpoint at level 16 is reached during iteration, or if the filter function returns `RayTraverseAction::STOP_SUCCESS`. Otherwise, false is returned. Starting from the root node, all relevant keys are iterated exactly once, together with their respective node pointers. Then, in contrast to the DDA algorithm, no key-node lookups are required. Together with the recursive, templated formulation, this results in major performance benefits.

Ray Casting

The `rayTraverseLeveled` function (8.9) allows for arbitrary operations on nodes along a ray, which are defined by a `filter` function (8.10). We will now return to the initial problem of determining voxel visibility of voxel centers inside a camera frustum. The solution to this problem is as simple as defining a suitable filter and is given in algorithm (8.4). A substantial speedup compared to the DDA approach can be achieved, since the majority of the voxel map consists of empty space. If this space is pruned, the ray can be iterated at much higher octree levels. Note that if a voxel is classified as free, its children must not be occupied. They can only be unknown or free as well.

Algorithm 8.4: Code of the `isPointVisibleLeveled` function. It determines if a point is occluded or visible for a given camera pose.

```
1  template<class NODE>
2  bool OccupancyOcTreeLeveledBase<NODE>::isPointVisibleLeveled
3     (const octomap::point3d& camera_center, const octomap::point3d& point) const {
4
5     auto filter = [this](const NODE* node, const octomap::OcTreeKey& key, unsigned
          int depth) {
6        if (!node) {
7           return RayTraverseAction::STOP_FAIL;       // unknown  -> exit
8        } else if (!this->isNodeOccupied(node) &&
9           !this->nodeHasChildren(node)) {
10          return RayTraverseAction::EXIT_BRANCH;     // free     -> end branch
11       } else if (depth < 16) {
12          return RayTraverseAction::CONTINUE_BRANCH; // occupied -> continue branch
13       } else {
14          return RayTraverseAction::STOP_FAIL;       // occupied -> exit
15       }
16    };
17
18    return this->rayTraverseLeveled(camera_center, point, filter);
19 }
```

8.3.3 Inserting Measurements

As previously stated in section 7, the NBV planning can easily be integrated into an autonomous exploration framework. In this case, an additional sensor is required which estimates an occupancy map in the same octree that stores the quality estimate. The map is updated with each depth measurement ray to reflect the corresponding occupancy changes (see section 4). Here, we encounter a similar problem as in the previous section. While a dense discretization of the surface geometry (i.e. low voxel size at level 16) is desirable, this is also detrimental to the task of autonomous navigation. All voxels traversed by the measurement ray require an update, which is computationally expensive for small voxel sizes. On the other hand, those small voxel sizes are not necessary for the task of autonomous navigation, where the map resolution is usually much lower. Hence, we allow measurement updates on higher octree levels. This means, while the endpoints are still updated as occupied at tree level 16, only voxels traversed on a higher level (e.g. 14) are updated as free. We refer to the tree level on which autonomous navigation takes place as the *navigation layer*. Note that the node on the navigation layer containing the endpoint requires special attention, since it would otherwise be updated as free (navigation layer) and occupied (level 16). Here, the tree is simply only updated on level 16 for free nodes as well, which we will refer to as *filler nodes*. The basic idea of this leveled measurement insertion is illustrated in figure 8.4.

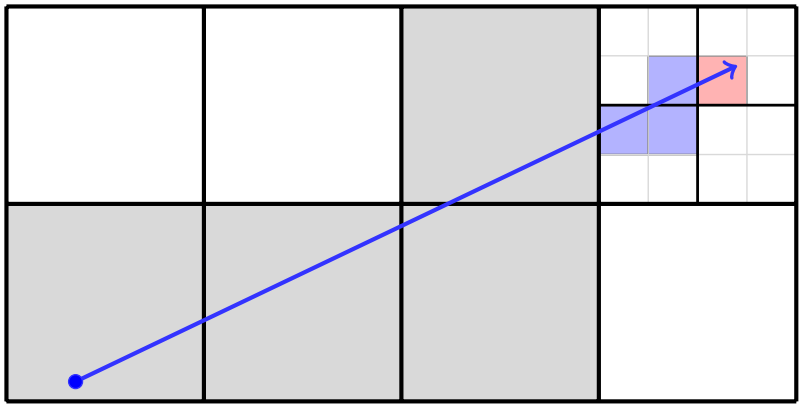


Figure 8.4: Relevant voxels for depth measurement ray insertion. The blue line represents a measurement ray. Gray cells correspond to voxels traversed by the ray that are updated as free space at the navigation layer. Blue cells represent filler voxels that are also updated as free, but this time at tree level 16. The red cell corresponds to the ray endpoint and is updated as occupied.

In the OctoMap library, the insertion of measurement rays on level 16 is realized inside the `OccupancyOcTreeBase<NODE>::insertPointCloud` function. The pseudo code for our new leveled implementation of `insertPointCloud` is given in algorithm 8.5. The individual components of this pseudo code are explained below:

- `free_depth`, `occ_depth`
  Tree depth of free voxel updates (navigation layer) and occupied voxel updates (level 16).

- `discretize_scan`
  Flag, indicating if the scan (i.e. a collection of depth measurements, corre-

sponding to measurement rays) should be preprocessed. In the case of a high resolution scan, this can lead to a massive increase in runtime performance.

- `discretize(scan, depth)`
  Function that discretizes a `scan`. All voxels at tree level `depth` that contain a point are collected. Their centers are returned as a new scan. This way the number of scan points can significantly be reduced.

- `computeKeysFree(scan, sensor_origin, free_depth)`
  This function computes all OctoMap voxel keys on the navigation layer that are traversed by any measurement ray from the `sensor_origin` to endpoints given by `scan`. For that purpose, each ray is traversed individually using the DDA algorithm (see section 8.3.2). Keys corresponding to each traversed voxel are collected in an `unordered_set`. This method can easily be parallelized for all individual rays.

- `computeKeysFiller(scan, sensor_origin)`
  This function is similar to `computeKeysFree`. Only voxels on the navigation layer that contain an endpoint are considered and used as bounding boxes. Voxels on a measurement ray inside these bounding boxes at level 16 are added to the `unordered_set` of filler voxels.

- `computeKeysOccupied(scan)`
  All keys at tree level 16 that contain a scan endpoint are collected and returned as `unordered_set`.

- `cleanFiller(keys_filler, keys_occ)`
  This function makes the sets `keys_filler` and `keys_occ` disjoint. In order to counteract cancellation errors (i.e. updating voxels as free, despite them also being measured as occupied), keys corresponding to occupied updates get a higher priority. Hence, keys that are present in both sets are removed from `keys_filler`.

- `cleanFree(keys_free, keys_occ)`
  For the same reasoning as for the previous function, the set of free keys `keys_free` is modified. If a parent node of any voxel corresponding to a key in `keys_occ` is present in `keys_free`, it is removed from that set.

- `updateNode(key, type, depth)`
  This function applies the occupancy update to a node that corresponds to the `key` value at a given `depth`. The `type` can either be `OCCUPIED` or `FREE`. The update is a simple addition to the log-odds value that is stored inside each node (see equation (4.5)). As for the `OcTree` class implemented in OctoMap, this occupancy value is propagated to all parents as the maximum of their children. If the node corresponding to the key does not exist, it is created at the specified depth first.

  On the other hand, if this node has children, then a different update is applied. This is only the case for `FREE` updates (e.g. at navigation layer), since `OCCUPIED` updates only happen at tree level 16. If such a node has children, then it has especially not been pruned. Then child nodes exist that are classified as occupied or correspond to unknown space. Instead of updating the node

itself, all child nodes are then updated with a damped log-odds value. This further counteract erasure errors. For example, if the parent node contains occupied children that were only classified from previous scans but have not been observed in the current scan, then it should be more difficult to change that classification to free.

Ultimately, we introduced an algorithm that allows for efficient autonomous navigation in an occupancy map, while maintaining a high detailed representation of surface geometries.

Algorithm 8.5: Pseudo-code of the `insertPointCloud` function. A more detailed description of all components is given in the text.

```
function insertPointCloud(scan, sensor_origin)

   if (discretize_scan)
      scan_discrete ← discretize(scan, free_depth)
      keys_free     ← computeKeysFree(scan_discrete, sensor_origin)

      scan_discrete ← discretize(scan, occupied_depth)
      keys_filler   ← computeKeysFiller(scan_discrete, sensor_origin)
      keys_occ      ← computeKeysOccupied(scan_discrete)
   else
      keys_free     ← computeKeysFree(scan, sensor_origin)
      keys_filler   ← computeKeysFiller(scan, sensor_origin)
      keys_occ      ← computeKeysOccupied(scan)
   end if

   // remove filler keys that are occupied keys
   keys_filler ← cleanFiller(keys_filler, keys_occ)
   // remove free keys that are parents of occupied keys
   keys_free   ← cleanFree(keys_free, keys_occ)

   // insert data into tree
   for (key in keys_free)
      updateNode(key, FREE, free_depth);
   end for
   for (key in keys_filler)
      updateNode(key, FREE, occupied_depth);
   end for
   for (key in keys_occ)
      updateNode(key, OCCUPIED, occupied_depth);
   end for

end function
```

8.4 Parallelization

The functions `isPointVisibleLeveled` (algorithm 8.4) and `getVoxelFOV` (algorithm 8.2) are used in combination to compute the set $p \in \overline{\text{FOV}}$ (see equation (7.1), algorithms 7.1 and 7.2). The reader may have realized that the check for voxel visibility `isPointVisibleLeveled` could also have been realized in the filter function of `getVoxelFOV` to directly discard obstructed voxel centers. This way some additional conversion between key, node and center point representations of voxels could be omitted. However, by keeping both algorithms separate, the ray casting from `isPointVisibleLeveled` can easily be parallelized over each key inside the frustum. In our simulations in chapter 10, we use 4 threads for this purpose, as it

resulted in the best average runtime using an I7-4790k processor. Similarly, the loop computing the L RRT samples (see algorithm 7.2) is parallelized on 4 threads.

Taking into account all implementation and parallelization details, the calls to `getVoxelFOV` still accumulate to an average of 7.3 % of the total runtime of the SfM-CPP algorithm 7.1, while calls to `isPointVisibleLeveled` account for 83.9 % of the total runtime. We believe that this can be significantly improved in the future by using a GPU acceleration, as it is predestined for ray casting tasks. Libraries such as GPU-Voxels [Hermann et al., 2014] can then be utilized to maintain the octree structure inside the GPU memory.

Simulation and Evaluation Setup

9

In this chapter, we describe the simulation setup and evaluation pipeline required for chapter 10, where the SfM-NBV algorithm 7.1 is analyzed. We use a voxel discretization (section 7.1.3) as model representation, with implementation details given in section 8. The focus of our analysis lies on the comparison of the estimate of the expected SfM reconstruction quality with the actual reconstruction error and its application in the gain formulation of the NBV planning (7.3a). For that purpose, we ignore the path cost term (7.3b) and allow for arbitrary long trajectory segments between consecutive views. Then the objective function of the OED problem 7.3 is only depending on the final pose of this path segment and especially independent of the traversed trajectory. Now, only the RRT sampling from algorithm 7.2 depends on the differential constraints (7.3c) and (7.3d). RRT based path planning is already a very well studied topic, both for direct pose control (e.g. [Karaman and Frazzoli, 2011], [Cover et al., 2013]), as well as in combination with kinodynamic constraints (e.g. [Kuwata et al., 2009], [Webb and van den Berg, 2013]). Hence, it is easy to include robot specific RRT algorithms into our SfM-NBV approach. Since those differential constraints only further constrain the admissible space $SO(3) \times \mathbb{R}^3$ for the NBV selection, it is sufficient for our analysis to directly control position and orientation of a camera with a simple RRT using piecewise linear path segments between consecutive views.

For our purpose, a simulation environment has several advantages over a real-world example. Exact reference data can be used as ground truth, i.e. the model obtained from a SfM reconstruction using the simulated images can be compared to an exact reference mesh. Furthermore, additional photogrammetric effects that otherwise contribute to the reconstruction error and distort the analysis can be mitigated. This is discussed along with the simulation setup in section 9.1. Afterward, an overview over our analytic methods is given in section 9.2. The simulation results for four distinct scenarios are presented in chapter 10, each highlighting a different aspect of the SfM-NBV planning.

9.1 Simulation Setup

A schematic overview over the full simulation setup is given in figure 9.1. We are using the simulation environment Gazebo [Koenig and Howard, 2004] to simulate a robot that is able to take images of a scenery. Our SfM-CPP algorithm communicates with Gazebo using the Robot Operating System (ROS) [Quigley et al., 2009] as interface. We send control commands to the simulation and receive images through a ROS topic.

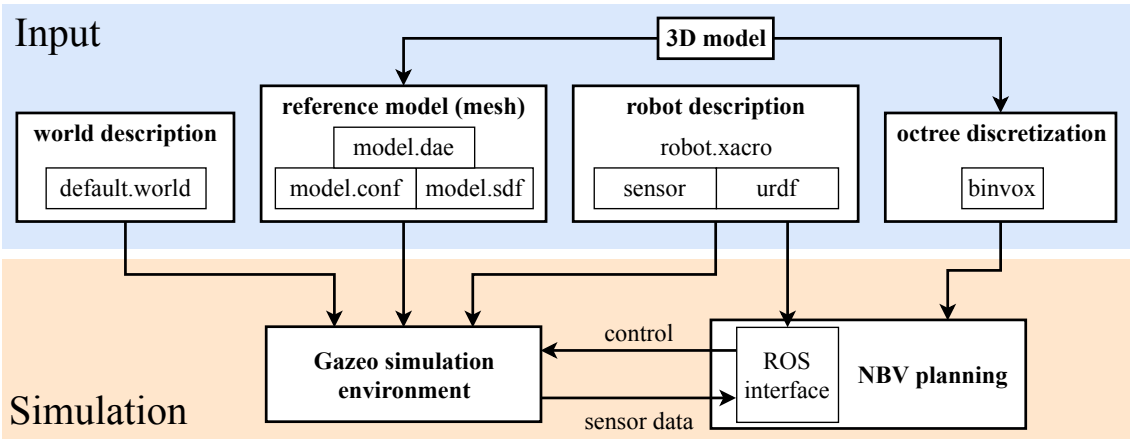


Figure 9.1: Schematic illustration of the different components in the simulation setup.

The world description file 9.2 contains general settings for the simulated environment. In order to eliminate SfM reconstruction effects caused by poor illumination, shadows are disabled and the ambient color is set to white. Due to the absence of lights in the scene, all textures now only render with their ambient color. This means that especially diffuse, specular, and emissive effects of the Blinn-Phong shading model [Blinn, 1977] are not visible and thus will have no effect on the images recorded in the simulation. By picking a monochrome background color (e.g. white), we prevent it from interfering with the SfM pipeline.

The 3D mesh of a structure is used as a reference model in the simulation. The necessary files required to import a COLLADA (.dae) model in Gazebo are given in files 9.1 and 9.3. Note that in order for the texture to be rendered properly, all material's ambient colors need to be set to `<color>1 1 1 1</color>` in the COLLADA file. Then all textures are rendered exactly with the colors specified in the texture files inside the simulation. The same 3D model is rasterized into a binary 3D voxel grid using the open source tool binvox ([Nooruddin and Turk, 2003] [Min, 2019]). This voxel representation is used as surface discretization, where all voxel centers of occupied nodes correspond to control points (see section 7.1.3). Here it is important to ensure that both, the reference mesh and its voxel representation, are properly aligned, i.e. scale, position and orientation must match.

The robot description is provided as an XML Macros (xacro) [Glaser et al., 2009] file. A minimal example of a free-floating camera is given in file 9.4. The xacro file is later expanded into the unified robot description (URDF) format. The `<gazebo>...</gazebo>` tag is an extension to the URDF format, used to pass additional information to gazebo. In this example, we specify a camera sensor with an image resolution of 800×800 and horizontal field of view of $80°$ (≈ 1.396 radians) that is to be simulated. All camera intrinsic parameters as well as lens distortion parameters and image noise can be specified. Their exact values can also be passed to the SfM reconstruction tool to avoid parameter estimation errors. For our use case, we chose to disable all distortion and noise. The observed scenery is published as a video stream at 30 frames per second through the ROS topic `/robot/camera/image_raw`. In order to capture an image, we simply extract a single frame from this video stream. The utilized image format must be lossless, as compression artifacts also have effects on the SfM reconstruction (see figure 9.2). Hence, we store all of our captured images in the tagged image file format (tiff).

File 9.1: Minimal model.config file.

```
<?xml version="1.0"?>

<model>
  <name>model name</name>
  <version>1.0</version>
  <sdf version="1.5">model.sdf</sdf>

  <author>
    <name>author name</name>
    <email>author email</email>
  </author>

  <description>
    description
  </description>
</model>
```

File 9.2: Minimal default.world file.

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <world name="default">
    <scene>
      <ambient>
        1.0 1.0 1.0 1.0
      </ambient>
      <background>
        1.0 1.0 1.0 1.0
      </background>
      <shadows>0</shadows>
    </scene>
  </world>
</sdf>
```

File 9.3: Minimal model.sdf file.

```
<?xml version="1.0"?>
<sdf version="1.5">
  <model name="model name">
    <pose>0 0 0 0 0 0</pose>
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <mesh>
          <uri>model://model.dae</uri>
          </mesh>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <mesh>
          <uri>model://model.dae</uri>
          </mesh>
        </geometry>
      </visual>
    </link>
  </model>
</sdf>
```

In file 9.5, we specify a minimal ROS launch file that starts the Gazebo server and client, loads the robot into the `robot_description` ROS parameter, and spawns it in the simulation. While the robot is directly spawned using the launch file, the 3D ground-truth model is imported into the simulation with a ROS service call to `/gazebo/spawn_sdf_model` using a `gazebo_msgs::SpawnModel` message. The robot can be controlled using one of the ROS services provided by gazebo, e.g. `/gazebo/set_link_state` and `/gazebo/set_model_state`, or by publishing messages to the eponymous topics.

File 9.4: Minimal robot.xacro file. The robot consist of a single box to which a simulated camera is attached.

```
<?xml version="1.0"?>
<robot name="robot" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- robot geometry -->
  <link name="camera_link">
    <inertial>
      <mass value="1.0" />
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1" />
    </inertial>
    <visual>
      <geometry>
        <box size="0.1 0.05 0.05"/>
      </geometry>
      <material name="orange">
        <color rgba="1.0 0.5 0.0 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <box size="0.1 0.05 0.05"/>
      </geometry>
    </collision>
  </link>

  <!-- gazebo specific settings -->
  <gazebo reference="camera_link">
    <turnGravityOff>true</turnGravityOff>
    <material>Gazebo/Orange</material>
    <!-- camera simulation -->
    <sensor type="camera" name="camera1">
      <update_rate>30.0</update_rate>
      <camera name="head">
        <horizontal_fov>1.3962634</horizontal_fov>
        <image>
          <width>800</width>
          <height>800</height>
          <format>R8G8B8</format>
        </image>
        <clip>
          <near>0.02</near>
          <far>300</far>
        </clip>
      </camera>
      <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
        <alwaysOn>true</alwaysOn>
        <updateRate>0.0</updateRate>
        <cameraName>robot/camera</cameraName>
        <imageTopicName>image_raw</imageTopicName>
        <cameraInfoTopicName>camera_info</cameraInfoTopicName>
        <frameName>camera_link</frameName>
        <hackBaseline>0.0</hackBaseline>
        <distortionK1>0.0</distortionK1>
        <distortionK2>0.0</distortionK2>
        <distortionK3>0.0</distortionK3>
        <distortionT1>0.0</distortionT1>
        <distortionT2>0.0</distortionT2>
      </plugin>
    </sensor>
  </gazebo>

</robot>
```

Figure 9.2: Effect of jpg compression artifacts on an SfM reconstruction using the example of the King's Hall of Lorsch Abbey (UNESCO World Heritage Site). Left: Example image of a set of pictures used for the SfM reconstruction. For visualization purposes, the color [255, 255, 255] has been replaced with green, such that compression artifacts become visible at the model outline. Right: SfM reconstruction from the jpg images. The background color is chosen as green. Notice the white noise around the model.

File 9.5: Minimal ROS launch file. The robot description is passed to ROS. Afterward, the gazebo simulation is started, wherein the robot is spawned.

```
<launch>

<!-- start gazebo server-->
<node name="gazebo" pkg="gazebo_ros" type="gzserver" respawn="false" output="screen
    " required="true" args="default.world" />

<!-- start gazebo client -->
<node name="gazebo_gui" pkg="gazebo_ros" type="gzclient" respawn="false" output="
    screen" />

<!-- Load XACRO -->
<param name="robot_description" command="$(find xacro)/xacro --inorder 'robot.xacro
    '" />

<!-- Run a script to send a service call to gazebo_ros to spawn a URDF robot -->
<node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
    output="screen" args="-urdf -model robot -x 0 -y 0 -z 1 -param
    robot_description"/>

</launch>
```

9.2 Evaluation Pipeline

In this section, we describe how the simulations are evaluated. New terminology is introduced that will be important for the evaluation of the results in chapter 10.

Using the simulation setup from the previous section, the SfM-CPP algorithm (algorithm 7.1) is executed with a given reference mesh. Subsequently, a SfM reconstruction is calculated from the resulting set of images using Agisoft Photoscan [Agisoft, 2017], which is to be compared with the reference mesh. Instead of the resulting textured SfM mesh, we directly use the dense SfM point cloud (see section 2) for this comparison. The reasoning behind this is that the dense cloud can be interpreted as the raw reconstruction data, since each point corresponds to multiple depth hypotheses of pixels in the recorded images. A subsequent SfM meshing step would distort our analysis, as a surface mesh is fit to the point cloud data, which heavily depends on the utilized surface reconstruction method. Due to interpolation, there could then be segments of the mesh that belong to areas where no data has been recorded. The fit would also further alter and smooth the reconstruction error (see [Berger et al., 2013]) and especially remove most of the outliers completely. Furthermore, working with discrete SfM points simplifies the analysis, since these can be associated with a reconstruction error, and an estimate of their expected reconstruction quality, as we will see later in this section.

9.2.1 SfM Reconstruction

We import the camera poses associated to all images obtained from the Gazebo simulation in Photoscan, which are used as initial guess for the camera pose estimation. This way the SfM reconstruction will have the same scale and orientation as the reference mesh and comparisons become easier. In order to eliminate reconstruction errors caused by parameter estimation, the camera intrinsics are provided and fixed[13]. This is achieved through `Tools` → `Camera Calibration...`, where we set the type to `Precalibrated` and check `Fix calibration` in the `Initial` tab. Then the camera alignment step is performed, followed by the point cloud densification. The parameters used in the SfM reconstruction workflow are given in table 9.1.

9.2.2 Analysis Procedure

We now want to evaluate how well our estimate of the expected reconstruction error reflects the actual reconstruction error. To achieve this, we first assume that a SfM point represents the reconstruction of a point on the reference mesh that has the smallest distance to the SfM point. These points are denoted as *reference points*. Using the mesh surface normals, we can define the signed distance to the mesh, which is called the *signed reconstruction error*. Note that the absolute value of the signed reconstruction error corresponds to the distance between reference point and SfM point. For the calculation of both signed reconstruction errors and reference points, a custom plugin for the point cloud and triangular mesh processing software CloudCompare [CloudCompare, 2019] has been developed. There the parallelized function `computeCloud2MeshDistance` is used to compute all reference points and

[13] These correspond to the sensor settings specified in file 9.4.

Table 9.1: Agisoft Photoscan paramters used in the SfM reconstruction workflow.

Name	Value
Align Photos	
Accuracy	Highest
Generic preselection	enabled
Reference preselection	enabled
Key point limit	40,000
Tie point limit	4,000
Adaptive camera model fitting	disabled
Build Dense Cloud	
Quality	Ultra high
Depth filtering	Aggressive

their associated signed reconstruction errors. It utilizes an octree segmentation to drastically reduce the number of mesh triangles considered for this calculation.

Since we are using a voxel discretization of the surface manifold, each reference point lies inside a voxel classified as occupied. The voxel centers are used as control points that provide local estimates of the expected reconstruction errors (see section 7.1.3). Therefore, all SfM points and their corresponding reference points can be associated to their closest control point. Note that some occupied voxels may not be observable due to occlusion (see figure 9.3). Hence, in order to analyze the quality of the local estimates, we do not consider voxels that are not observed in the SfM-NBV simulation. These are identified as voxels with control point precision matrix zero.

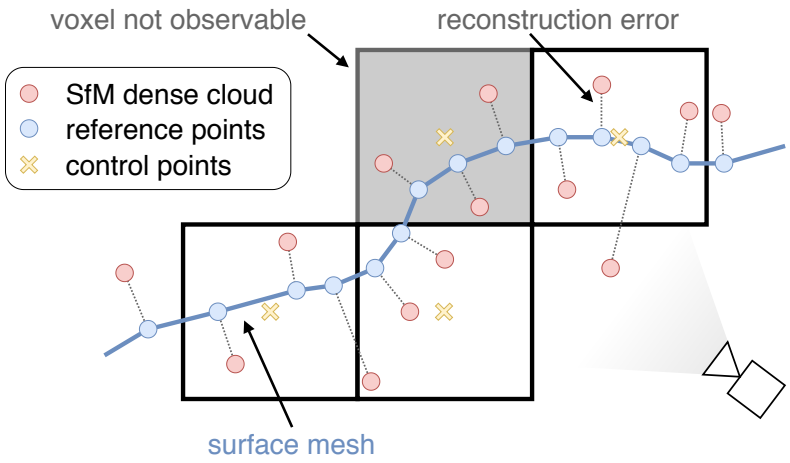


Figure 9.3: Different point types used in the SfM-NBV analysis procedure in 2D.

Finally, the estimates on the expected reconstruction quality of the control points need to be compared to all associated reconstruction errors of the SfM points. The former is given in the form of a precision matrix whose confidence ellipsoid can be used to compare both quantities. While the choice of the reference points can be a good approximation to the real ones that actually correspond to the SfM points, the direction of the reconstruction error can be arbitrary for small distortions of the reference points. Therefore, it does not make sense to directly test whether SfM

points are contained inside the confidence ellipsoid centered at their reference point. Especially in the case of narrow confidence ellipsoids this may lead to distorted results. This is also the case if the reference point is far away from the control point, i.e. close to voxel corners. For that reason, we test instead whether the signed reconstruction error is within a confidence interval defined by the largest confidence ellipsoid half-axis. This interval is given by

$$[-c_{qe}, +c_{qe}], \text{ with } c_{qe} = \sqrt{Q_1(0.99) \cdot 1/\lambda_{\min}[\Sigma^{-1}]}, \tag{9.1}$$

where Σ^{-1} denotes the beforehand mentioned precision matrix and Q_1 is as in section 3.1. We have chosen a 99 % confidence interval here, but the exact percentage is of less importance, which is explained in the next paragraph. The value c_{qe} is called *quality estimate* and can be interpreted as an approximation of the magnitude of the expected reconstruction error.

The size of the quality estimate still depends on the parameters n_{pix} and p_{conf} (see equation (6.2.2)), which represents our assumption on the pixel plane error. Note that these and the confidence percentage in (9.1) only act as a scaling factor to the quality estimate. Hence, we fix p_{conf} to 95 % and determine a suitable value for n_{pix} in simulation in section 10.1, such that the quality estimate adequately represents the magnitude of the real reconstruction error. More specifically, the parameter must be large enough, that most of the reconstruction errors of SfM points are within their associated intervals defined in (9.1). On the other hand, we want it to be sufficiently small such that the quality estimate still gives a good approximation to the real reconstruction error. However, these parameters have no influence on the NBV selection. In fact, if $\lambda_{\min}$ and $\lambda_{\max}$ are chosen as in (6.45), the clamped gain formulation (6.44) is independent of n_{pix}, p_{conf}, and even the camera intrinsic parameters, since they cancel out in the fraction. In real world applications, the quality estimate provides the user with an estimate of the expected reconstruction error.

Altogether, each SfM point is associated with a reference point, a signed reconstruction error, a control point, and consequently a quality estimate. Since the signed reconstruction error and the quality estimate are one-dimensional quantities, they can conveniently be visualized by colors in 3D, either on the surface of the reference mesh or as voxel in the voxel discretization. We will analyze and visualize the following quantities:

- **Signed reconstruction error**
  Consider the untextured, monochrome reference mesh with a simple shading model. The SfM point cloud is projected onto this mesh and each point is colored according to its associated signed reconstruction error. This visualization especially gives insight over regions with high reconstruction errors. Also, areas that could not be reconstructed can be identified as the mesh is visible there. A histogram containing the overall reconstruction error distribution is provided. Note that this only allows us to evaluate the quality of a SfM reconstruction and no conclusions can be drawn about the quality estimate.

- **Voxel quality estimate**
  The voxel discretization is visualized by rendering all occupied cells. Each of these cubes is colored according to the quality estimate of the associated control points. The result is compared to the 3D visualization of the signed

reconstruction error visually. We can check if areas with high or low quality estimates match the actual reconstruction error. Also, their overall coverage of the reference mesh can be compared. A histogram over all control points' quality estimates is provided. The magnitude of these expected reconstruction errors can be compared with the histogram of the signed reconstruction error.

- **Point quality estimate**
  Both previous visualizations only consider either the quality estimate or the signed reconstruction error. Each point of the SfM point cloud is assigned their associated quality estimate from their closest control point. We then divide a histogram over the quality estimates of all points into a set where the estimate holds (i.e. the absolute reconstruction error is smaller than the quality estimate) and another one where the estimate fails. The percentage of SfM points that satisfy the quality estimate is called *acceptance rate*. This histogram gives insight into the acceptance rate for different orders of magnitude of the quality estimate.

- **Distance to quality estimate**
  The visualization method from the previous section provides a statement over the correctness of the quality estimates. Here, we are highlighting the distance between reconstruction error and quality estimate, i.e.

$$|(\text{reconstruction error})| - (\text{quality estimate}). \tag{9.2}$$

  Note that in (9.2) a negative value means that the quality estimate holds, while a positive value indicates that it fails. Again, a corresponding histogram gives insight over the distribution of the distance to the quality estimate. Ideally, the percentage of points not meeting the quality estimate is reasonably small, while being close to zero for most other points. SfM points violating the quality estimate are projected onto the shaded, untextured, monochrome reference mesh and colored according to the size of the violation. This gives insight over the spatial position of these points.

Results

10

We will now present four different simulations using the simulation setup and evaluation pipeline of chapter 9. Each experiment consists of a unique reference mesh with high-resolution textures. Their respective reference meshes have varying geometric and topological properties. All the simulations highlight and analyze a different aspect of the SfM-CPP algorithm.

- **Lorsch Abbey King's Hall**
  Analysis is performed for two extreme camera resolutions (very high / very low). The main purpose here is to compare the behavior and performance of the SfM-CPP algorithm for these extreme resolution cases and to find a suitable value for the parameter n_{pix}.
- **Holbeach Cemetery Chapel**
  In the second simulation example the focus lies on the evolution of the SfM reconstruction and the corresponding quality estimates with increasing number of recorded images.
- **Tyche Sculpture**
  For the Tyche sculpture, the effect of the discretization error is analyzed. This is achieved by executing the SfM-CPP algorithm multiple times for varying discretization resolutions (i.e. voxel sizes of the voxel discretization).
- **Roman Temple of Evora**
  This simulation is designed such that all observability properties of the whole reference mesh are predeterminable. The focus lies on the comparison of the expected coverage with the SfM reconstruction obtained from the set of images of an exhaustive SfM-CPP run.

The different camera parameters used in the simulations are given in table 10.1.

Table 10.1: Parameters for all three cameras used in the simulations.

identifier	resolution	horizontal FOV	focal length (pixel)
`CAM_LR`	640×480	$80°$	381.362
`CAM_MR`	2240×1680	$80°$	1334.769
`CAM_HR`	5456×3632	$80°$	3251.115

10.1 Lorsch Abbey King's Hall

Here we consider the extreme camera configurations `CAM_LR` and `CAM_HR` and their effects on the comparison of quality estimate and reconstruction obtained by the SfM-CPP algorithm. Furthermore, a suitable value for the image plane pixel error assumption n_{pix} (with a confidence percentage of 95 %, see section 6.2.2) is determined, which will be used for all remaining simulations.

Model

A 3D model of the King's Hall of Lorsch Abbey is used as a reference mesh. The structure was build in the early 9th century A.D. and is the only remaining building of Lorsch Abbey. In 1991 the monastery was listed as UNESCO World Heritage Site. The 3D model was kindly provided by Christian Seitz, head of the ArchEye Automatics project [Seitz and Altenbach, 2011], [Seitz, 2012]. It was originally created from a set of several thousand ground- and drone-acquired photos using SfM, where access to the structure was kindly granted by the director of the world heritage site Lorsch monastery. Figure 1.1 shows the corresponding camera positions of the drone. The resulting mesh consists of 15 million faces and 7 million vertices. It was reduced to a size of 500 000 faces and 250 508 vertices using quadric edge collapse decimation (e.g. [Shaffer and Garland, 2001]) to increase the simulation performance. The high resolution texture was not modified, but re-mapped to the reduced mesh. Furthermore, the model was scaled to a size of 12.92 m × 9.12 m × 13.50 m (width × depth × height). Since the real dimensions where unknown to us, these values may not reflect the actual size of the King's Hall. However, having visited the site ourselves, we believe this is a reasonable scaling. The mesh is oriented such that it stands on the xy plane at $z = 0$. The reference mesh is visualized in figure 10.1.

SfM-CPP Algorithm

For each RRT Sample in the `getNextSegment` subroutine (see algorithm 7.2) we demand a minimum distance to the mesh of 1 m, a maximum distance of 5 m, a minimum altitude of 1 m, and a collision-free path from the last pose. In total, we create 8000 valid RRT samples that satisfy those conditions. More precisely, 1000 valid positions, each sampled 8 times with their yaw angle are considered. Pitch and roll angles of the camera are fixed to 0°. The required minimal image overlap $o_{\min}$ is set to 20 %. The utilized clamping parameters are given in table 10.2. We run the SfM-CPP algorithm for 300 iterations using a voxel discretization with voxel resolution of 5 cm. As previously mentioned, the choice of the parameter n_{pix} does not affect the NBV selection. Therefore, the simulations are only run once for each `CAM_LR` and `CAM_HR`. The quality estimates are adapted afterwards to reflect the respective choices of n_{pix}. The resulting camera poses are visualized in figure 10.2. We measured an average runtime of $L \cdot 3.75$ ms for each evaluation of the `getNextSegment` subroutine and 3.36 ms for the precision matrix updates (see algorithm 7.1) on an I7-4790k. Note that both of these measurements were dominated by visibility checks, i.e. computations of $\overline{\text{FOV}}[\xi_{n+1}]$. These took on average 3.30 ms to identify several thousand voxels, that are visible inside the viewing

Figure 10.1: King's hall of Lorsch Abbey (UNESCO World Heritage Site) reference mesh.
Source: [Lindner et al., 2019] ©2019 IEEE.

frustum.

Different cameras (without axis skew) scale the estimate on the expected reconstruction quality (precision matrix) by their focal lengths (see section 6.2.2). Since the clamped gain term (6.44) is a relative quantity, this scaling does not affect the gain computations. Furthermore, by choosing the clamping parameters as proposed in (6.45), this scaling does not affect the NBV selection and the objective function value (7.3a) remains the same. This effect is observed in figure 10.3, where gain terms for both `CAM_LR` and `CAM_HR` are almost identical.

Evaluation

The estimated reconstruction quality is given in figures 10.4 and 10.5. These estimates can be compared to the measured signed reconstruction errors in figures 10.6 and 10.7. There, the magnitude of the signed reconstruction error is similar to the magnitude of the quality estimate. While the quality estimate is higher than the reconstruction error in most areas, especially some edges and geometric details such as the stucco on the windows seem to violate the quality estimate. As previously stated in section 7.1, this behavior is caused by the observability error. This becomes most apparent on the edge of the roof (left), where the quality estimate is much lower than the reconstruction error. Images that capture the edge from both

Table 10.2: Clamping parameters (see section 6.3.2).

Parameter	Value (`CAM_LR`)	Value (`CAM_HR`)
$d_{\min}$	4 m	4 m
$\lambda_{\min}$	$n_{\text{pix}}^2 \cdot 2.864 \times 10^{-5}$	$n_{\text{pix}}^2 \cdot 3.941 \times 10^{-7}$
corresponding 99 % confidence interval	$\pm\, n_{\text{pix}} \cdot 1.38\,\text{cm}$	$\pm\, n_{\text{pix}} \cdot 0.16\,\text{cm}$
$d_{\max}$	100 m	100 m
$\lambda_{\max}$	$n_{\text{pix}}^2 \cdot 1.790 \times 10^{-2}$	$n_{\text{pix}}^2 \cdot 2.463 \times 10^{-4}$
corresponding 99 % confidence interval	$\pm\, n_{\text{pix}} \cdot 34.46\,\text{cm}$	$\pm\, n_{\text{pix}} \cdot 4.04\,\text{cm}$

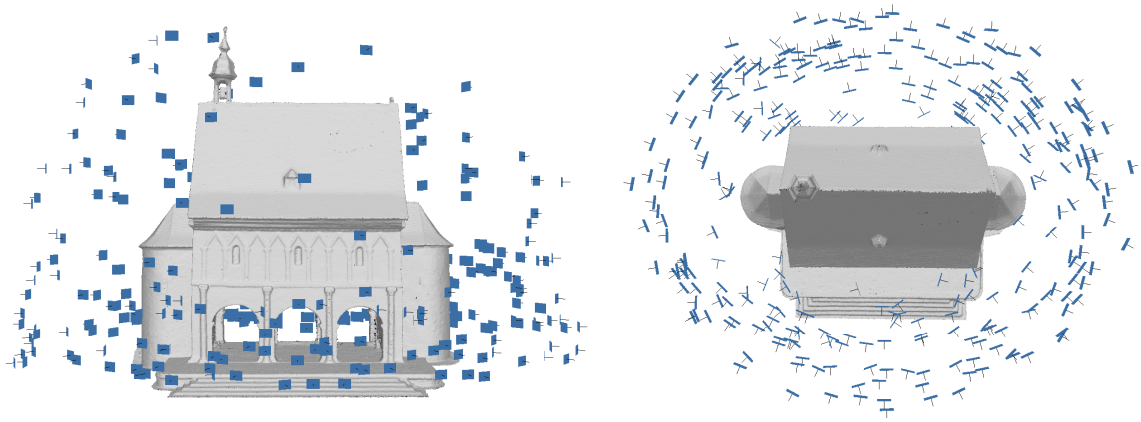


Figure 10.2: Camera positions after 300 SfM-CPP iterations using the high-resolution camera `CAM_HR`. Left: front view, right: top view.

sides are assumed to contribute to the quality estimate of the control point, while in reality only one side neighboring this edge is visible in these pictures.

More information on points that do not satisfy the quality estimate are given in figures 10.8 and 10.9, visualizing the distance to the quality estimate. Here our earlier assumption is confirmed. In most cases, only edges and areas with higher geometric fidelity do not satisfy the quality estimate. These areas may not be sufficiently approximated by the voxel discretization and are therefore subject to discretization and observability errors. This also applies to the `CAM_HR` visualization, however, due to the much larger point cloud size, noise distorts the visualization. There the point density is much higher in areas that are also highlighted in the `CAM_LR` case. Note that the shapes of both histograms look similar for all values of n_{pix}, especially for points violating the quality estimate. Overall, a good value for n_{pix} is given by 3. Then only a small fraction of points do not satisfy the quality estimate, while it is still a good guess for the reconstruction error, i.e. the distance to the quality estimate does not become too negative. In real world applications other values may be chosen. Smaller values for n_{pix} provide a better approximation of the reconstruction error (i.e. peaks in the histograms of figures 10.8 and 10.9 are closer to zero) at the expense of estimator confidence (i.e. more points do not

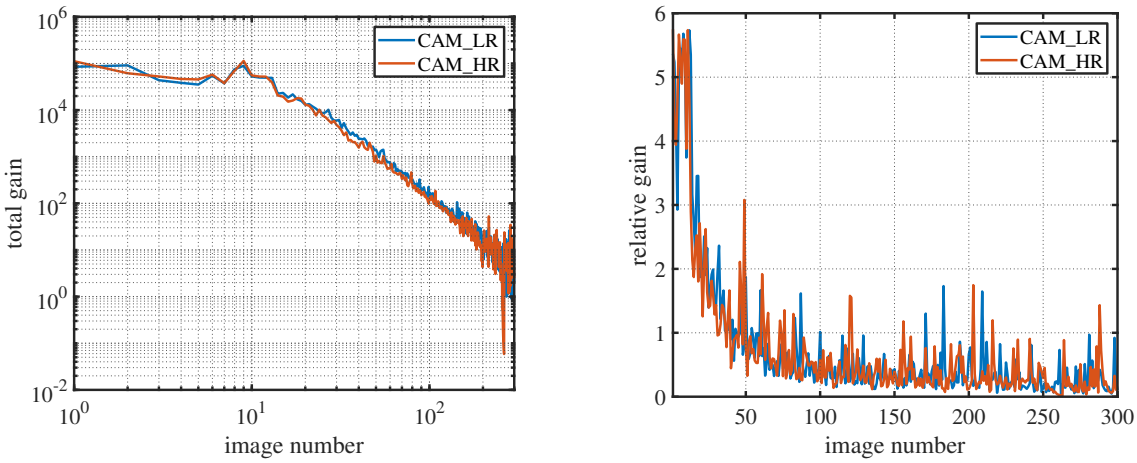


Figure 10.3: Gain objective function term (7.3a) for each selected NBV. Left: Logarithmic plot. Right: Relative gain, i.e. normalized by the number of all voxels that contribute to the gain with a term greater than zero.

satisfy the estimate). A value of $n_{\text{pix}} = 3$ proved to be a good balance between both properties and especially does not clutter the renderings of the distance to the quality estimate, highlighting the worst offenders.

Similar statements also result from the acceptance rate plots from figure 10.10 and 10.11, which focus on the relative amount of points that do not satisfy the quality estimate. Areas where the best quality is expected contain the largest amount of estimate violations. While the shape of the area where the estimate does not hold changes significantly for $n_{\text{pix}} \leq 3$, it remains almost identical for $n_{\text{pix}} \geq 3$. This statement is further substantiated by the total acceptance rate given in table 10.3.

Despite the huge difference in focal length and resolution between cameras `CAM_LR` and `CAM_HR`, all related plots look similar. This is particularly true for the provided histograms, which exhibit almost identical shapes. There, the axes corresponding to quality estimate and point count only differ by a constant common scaling factor. This shows that our algorithm performs similar for various camera settings. However, this comes to no surprise. As mentioned earlier in this section, the focal length is a linear scaling factor to the precision matrices of the expected reconstruction quality. In consequence, it also scales the quality estimate linearly. Hence, this factor can be calculated as the fraction between both focal lengths as 8.53. Since a similar scaling factor is observed for the signed reconstruction error (figures 10.6 and 10.7), the correctness of our estimator is substantiated.

The desired reconstruction quality is defined by λ_{min} (see table 10.2). The corresponding desired quality estimate (i.e. reconstruction error) for $n_{\text{pix}} = 2$ is given as 27.6 mm for `CAM_LR` and 3.2 mm for `CAM_HR`. We can verify that almost all SfM points satisfy this desired quality estimate from figures 10.6 and 10.7.

Table 10.3: Total acceptance rate of the quality estimate. The point cloud size only contains points that could be associated with a control point.

camera	point cloud size	$n_{\text{pix}} = 1$	$n_{\text{pix}} = 2$	$n_{\text{pix}} = 3$	$n_{\text{pix}} = 4$	$n_{\text{pix}} = 5$
`CAM_LR`	3 040 952	73 %	85 %	90 %	94 %	95 %
`CAM_HR`	242 286 040	76 %	89 %	93 %	96 %	97 %

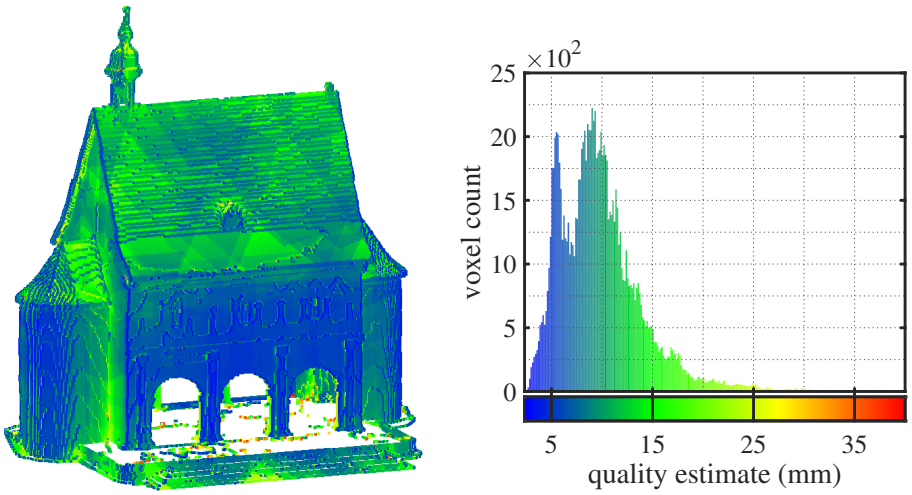


Figure 10.4: Voxels colored according to their quality estimate for the `CAM_LR` simulation and $n_{\text{pix}} = 2$.

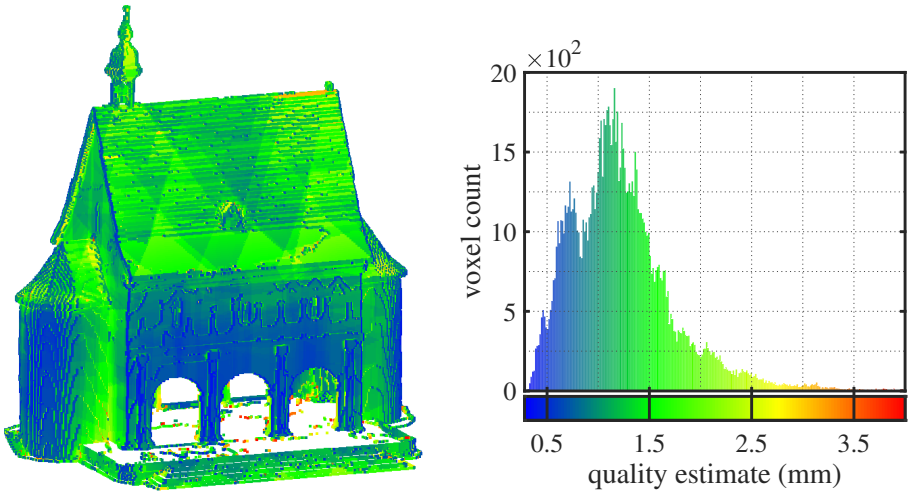


Figure 10.5: Voxels colored according to their quality estimate for the `CAM_HR` simulation and $n_{\text{pix}} = 2$.

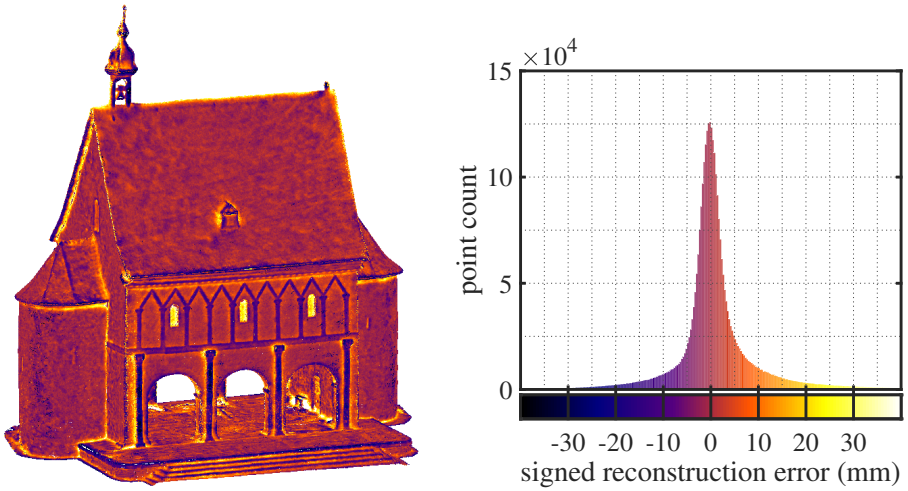


Figure 10.6: Projection of the SfM point cloud onto the reference mesh, colored according to the signed reconstruction error for the `CAM_LR` simulation.

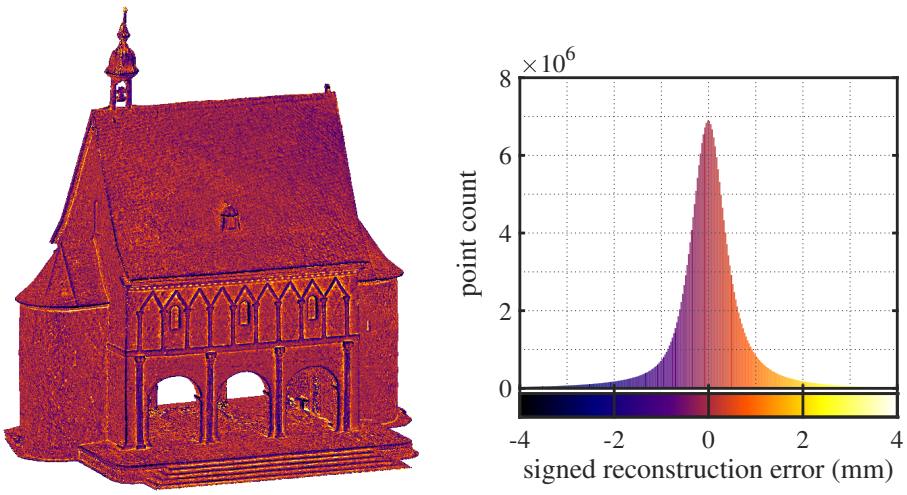


Figure 10.7: Projection of the SfM point cloud onto the reference mesh, colored according to the signed reconstruction error for the `CAM_HR` simulation.

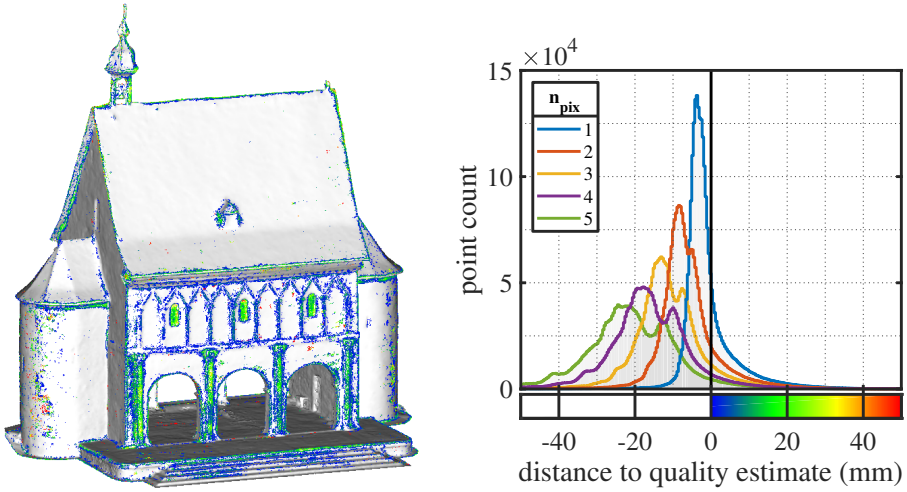


Figure 10.8: Distance to quality estimate for the `CAM_LR` simulation. All projected SfM points that do not satisfy the predicted quality estimate are visualized for $n_{\text{pix}} = 2$.

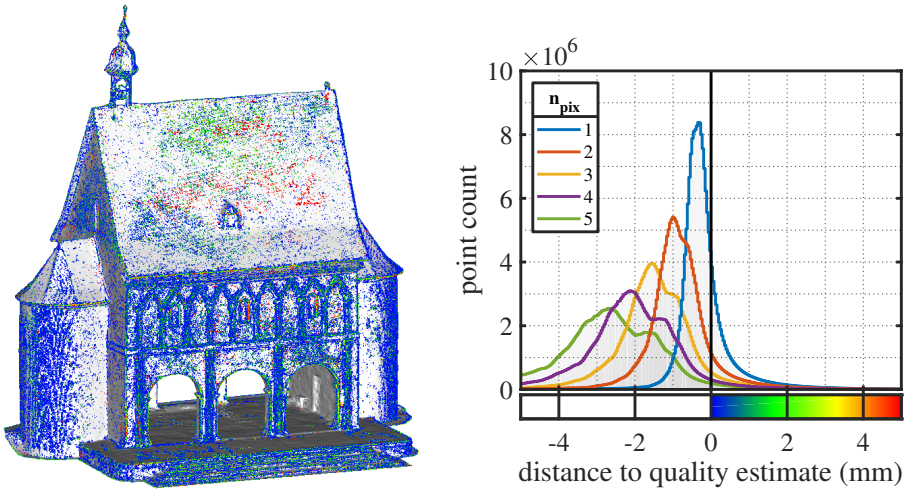
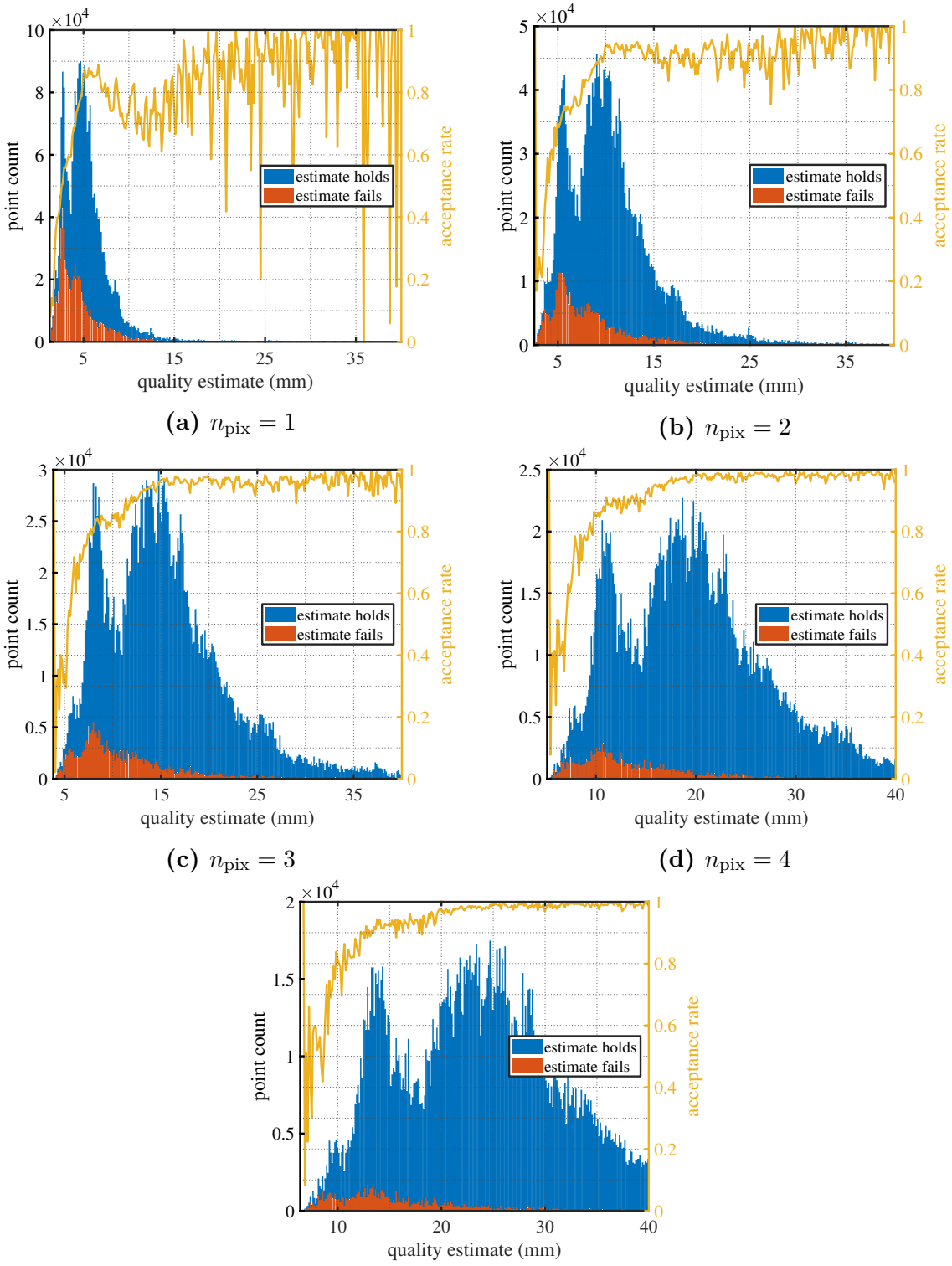


Figure 10.9: Distance to quality estimate for the `CAM_HR` simulation. All projected SfM points that do not satisfy the predicted quality estimate are visualized for $n_{\text{pix}} = 2$.



(a) $n_{\text{pix}} = 1$ **(b)** $n_{\text{pix}} = 2$

(c) $n_{\text{pix}} = 3$ **(d)** $n_{\text{pix}} = 4$

(e) $n_{\text{pix}} = 5$

Figure 10.10: Quality estimate of SfM points divided into classes where the estimate holds and where the estimate fails for the `CAM_LR` simulation.

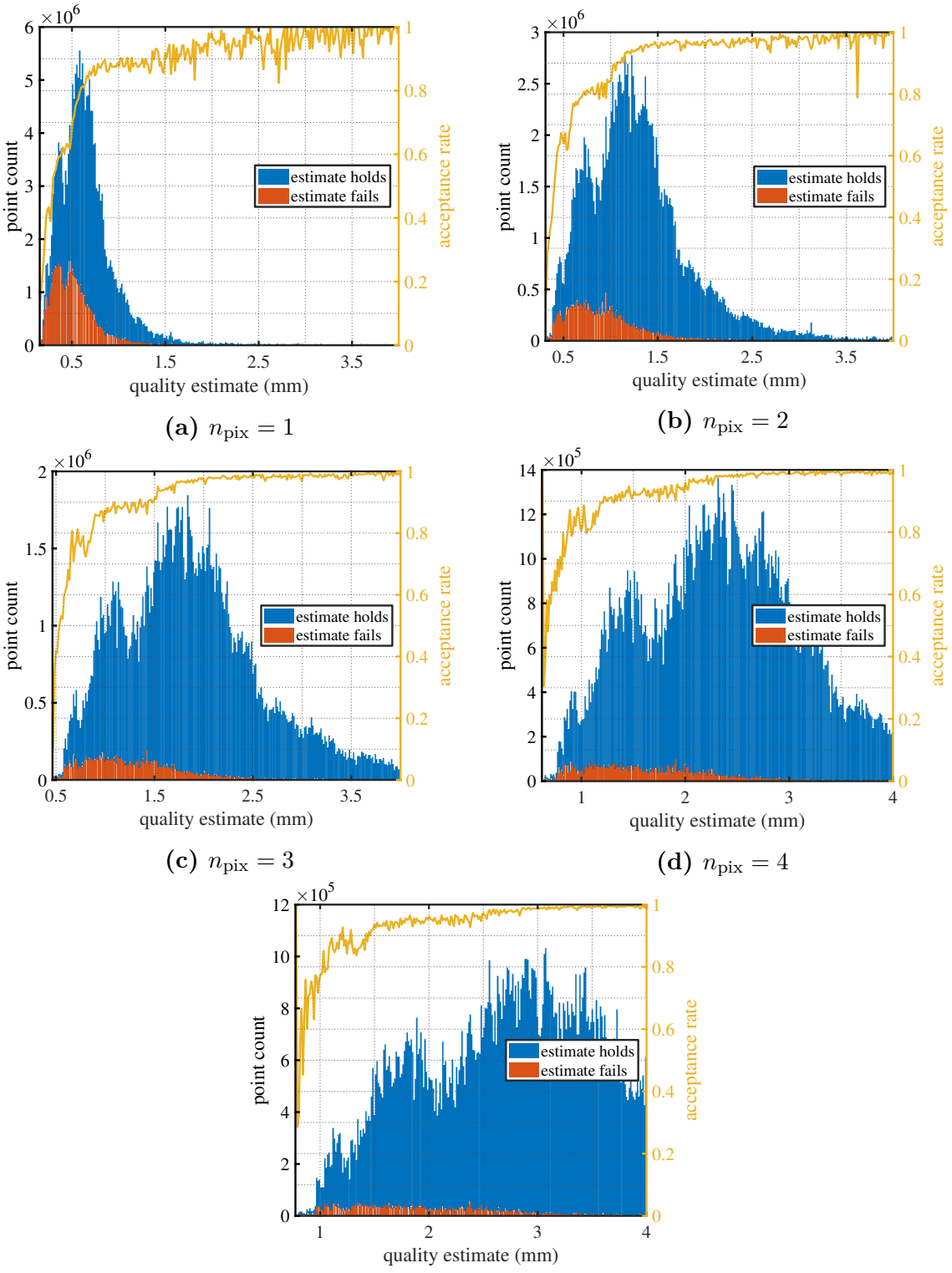


Figure 10.11: Quality estimate of SfM points divided into classes where the estimate holds and where the estimate fails for the `CAM_HR` simulation.

10.2 Holbeach Cemetery Chapel

The Holbeach Cemetery Chapel is used to analyze the evolution of the quality estimate and the corresponding SfM reconstruction with increasing number of images.

Model

The Cemetery chapel is located at Holbeach in the county of Lincolnshire in the United Kingdom. It is a prime example of Victorian architecture and was build in 1854. A high detailed 3D reconstruction of the chapel was created by [Glauser, 2017] in 2017 from approximately 1600 photos, which is available online and licensed under [CC BY 4.0, 2013]. This model is used with slight modifications as reference mesh in this simulation. We removed some faces at the base and additionally rotated and scaled it. The version of the mesh we use has 998 267 faces and 499 365 vertices. It is aligned with the coordinate axis and stands on the xy plane at $z = 0$. While this may not represent the actual size of the chapel, it is scaled to the reasonable size of 26.14 m $\times$ 14.43 m $\times$ 24.98 m (width $\times$ depth $\times$ height). The reference mesh is visualized in figure 10.12.

Figure 10.12: Holbeach Cemetery Chapel reference mesh.

SfM-CPP Algorithm

We geometrically constrain the cameras trajectory to have a minimum distance of 1 m and a maximum distance of 5 m to the mesh. The minimum altitude is 1 m. The camera is constraint to have a fixed roll angle of 0° and a pitch angle of 30°. In the `getNextSegment` subroutine (see algorithm 7.2), we use 8000 RRT samples, i.e. 1000 position samples that satisfy the geometric constraints, each sampled 8 times with their yaw angle. The required minimal image overlap is set to $o_{\min} = 20\,\%$. We are using the `CAM_MR` camera with clamping parameters given in table 10.4. Changes in the quality estimate and the associated SfM reconstruction and reconstruction error are analyzed for a total of 500 images, using a voxel discretization with a voxel resolution of 5 cm. The respective gain objective function values are given in figure 10.14. Note that the total gain gradually decreases as more and more voxels have been sufficiently observed, i.e. gain clamping for $\lambda_{\min}$ becomes active. Then these voxels do not contribute to the total gain anymore. Full SfM reconstructions where built from the $n_{\text{IMG}} \in \{10, 20, 30, 40, 50, 100, 200, 300, 400, 500\}$ first acquired images. Some of the resulting camera poses are visualized in figure 10.13. First, the structure is coarsely covered, then details are observed.

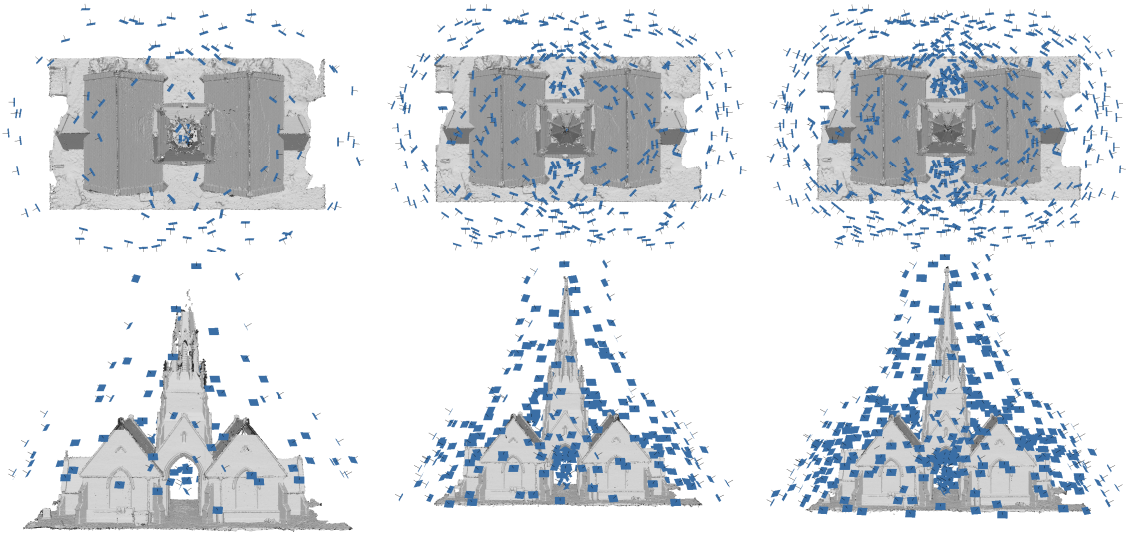
Table 10.4: Clamping parameters (see section 6.3.2).

Parameter	Value (`CAM_MR`)
n_{pix}	3
pixel error confidence	95 %
$d_{\min}$	2 m
$\lambda_{\min}$	5.260×10^{-6}
corresponding 99 % confidence interval	$\pm$ 0.59 cm
$d_{\max}$	50 m
$\lambda_{\max}$	3.288×10^{-3}
corresponding 99 % confidence interval	$\pm$ 14.77 cm

For this simulation, we measured a runtime of $L \cdot 9.38\,\text{ms}$ for each evaluation of the `getNextSegment` subroutine on an I7-4790k. This is roughly 2.5 times the runtime measured for the King's Hall of Lorsch Abbey. The reason behind this is the more complex object geometry, where ray casting is more expensive and on average much more voxels are visible in the FOV of cameras.

Evaluation

The quality estimates of all control points are visualized in figures 10.15 and 10.16. These can be compared to the signed reconstruction errors from figures 10.17 and



(a) $n_{\text{IMG}} = 100$ **(b)** $n_{\text{IMG}} = 300$ **(c)** $n_{\text{IMG}} = 500$

Figure 10.13: Camera positions after 100, 300 and 500 SfM-CPP iterations. Top: top view. Bottom: front view.

10.18. We expect all areas that correspond to a finite quality estimate to be reconstructable. From visual inspection, we conclude that this is true for most areas. Especially for $n_{\text{IMG}} = 500$ the complete geometry is covered and reconstructed with high detail. At first glance, it seems like few observations of the same area already suffice to get the final reconstruction error, since the magnitude of the reconstruction error does not change much between $n_{\text{IMG}} = 10$ and $n_{\text{IMG}} = 500$. On closer inspection, it becomes apparent that certain areas that correspond to low quality estimates are much more noisy than in reconstructions with higher image count. A prime example is the part of the roof on the right building that becomes reconstructable between $n_{\text{IMG}} = 40$ and $n_{\text{IMG}} = 50$. A high level of noise is visible for the signed reconstruction error, which is mitigated at $n_{\text{IMG}} = 100$ and vanishes for higher image numbers. This change is also correctly reflected in the quality estimate. Note that this effect can also be observed for higher values of n_{IMG}, as can be seen in the detailed view of the tower in figure 10.18. Even though it is already reconstructable with high accuracy for $n_{\text{IMG}} = 300$, point density and accuracy further increase with additional photos.

Details on the accuracy of the quality estimate can be extracted from figures 10.19 and 10.20. As for the previous simulation, we realize that high-density areas of SfM points that do not satisfy the quality estimate correspond to edges. Figures associated with higher number of images exhibit more additional outliers. Their relative amount, measured on the size of the SfM point clouds, however, only gradually increases as can be seen in table 10.5. Additionally, we realize a shift of the peaks in the histograms towards zero. This effect can be explained as follows.

A point that is assumed to be observable in a photo may not contribute to the reconstruction of said point. This could be due to observability errors, difficulties when creating depth map hypothesis for each image, when associating multiple depth map hypothesis, or other error sources in the SfM pipeline. Hence, the total number of images that contribute to the reconstruction quality of a point can be less then the amount of photos contributing to its quality estimate. With higher image

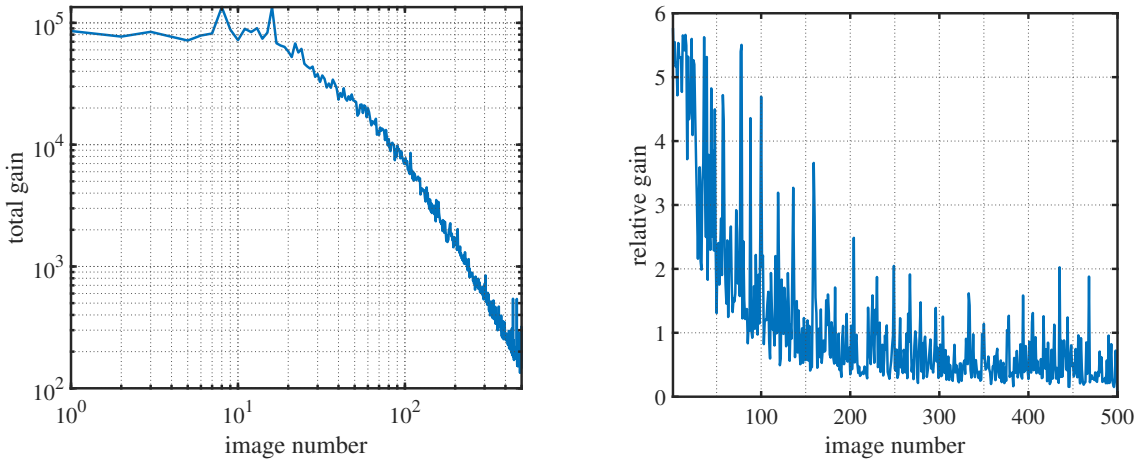
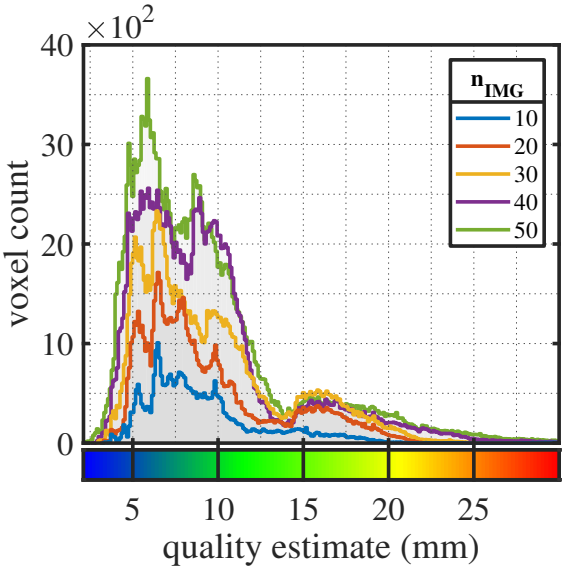


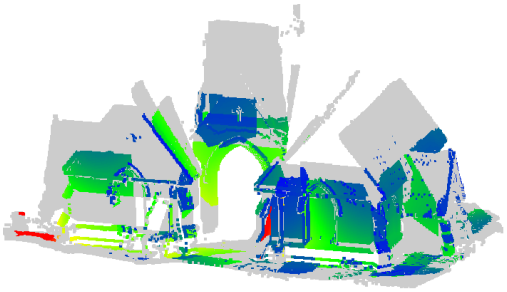
Figure 10.14: Gain objective function term (7.3a) for each selected NBV. Left: Logarithmic plot. Right: Relative gain, i.e. normalized by the number of all voxels that contribute to the gain with a term greater than zero.

count this effect is amplified and accumulates. This claim is further supported by figures 10.21 and 10.22, as well as table 10.5. Points with small quality estimates are assumed to have been observed the most on average. Hence, these points violate the quality estimate most, especially for large number of images. However, since these are the most frequently recorded points, their reconstruction error is usually still already smaller than the desired target accuracy defined by λ_{min} (here: $\pm$ 5.9 mm), even if they violate the quality estimate.

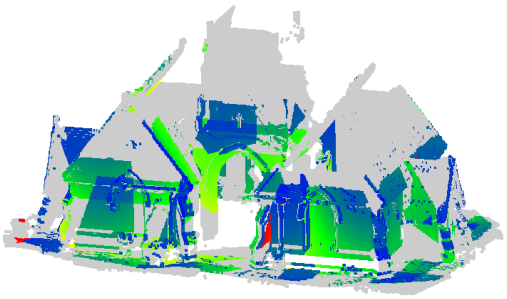
Table 10.5: Total acceptance rate of the quality estimate. The point cloud size only contains points that could be associated with a control point.

n_{IMG}	point cloud size	acceptance rate
10	5 255 850	98.51 %
20	9 796 686	98.32 %
30	16 744 189	98.06 %
40	22 892 641	97.95 %
50	29 653 231	97.84 %
100	49 441 712	96.81 %
200	70 579 815	94.79 %
300	79 726 281	92.48 %
400	88 290 206	90.66 %
500	94 770 892	89.04 %

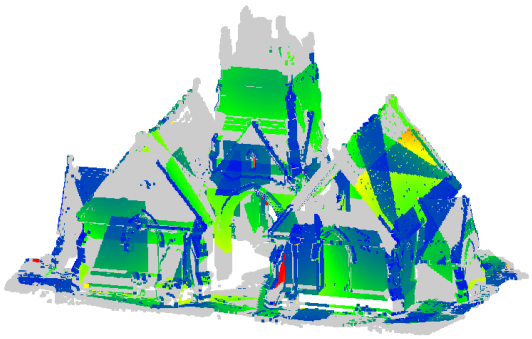




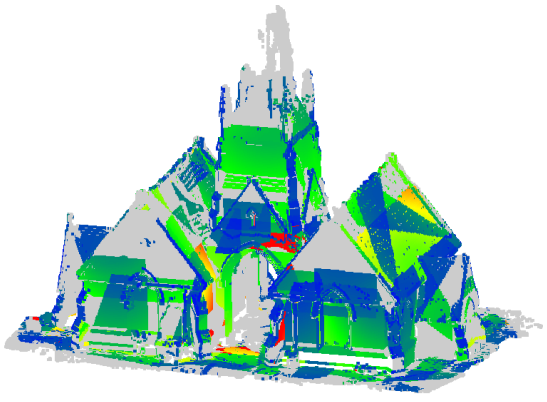
(a) $n_{\text{IMG}} = 10$



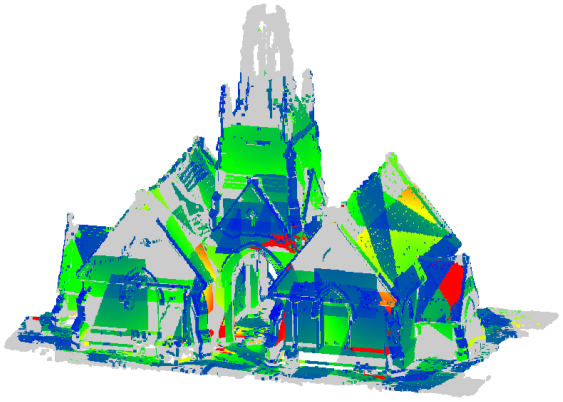
(b) $n_{\text{IMG}} = 20$



(c) $n_{\text{IMG}} = 30$

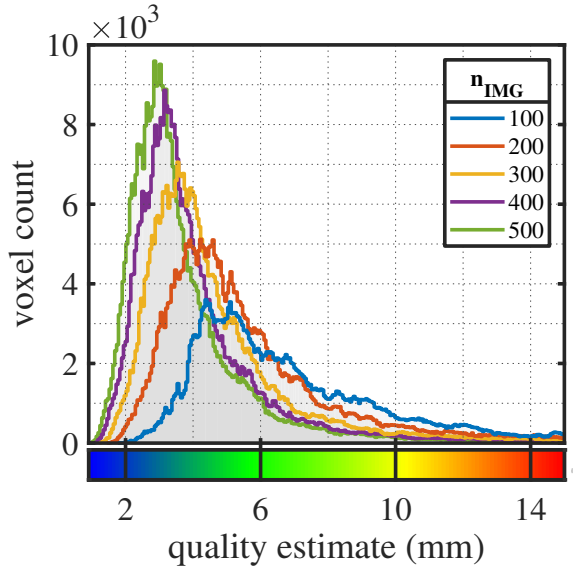


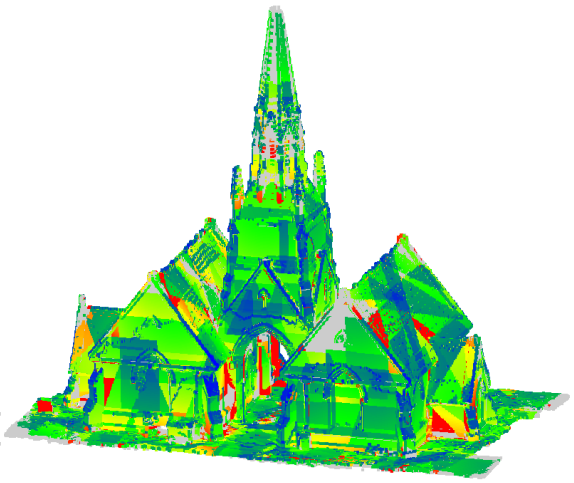
(d) $n_{\text{IMG}} = 40$



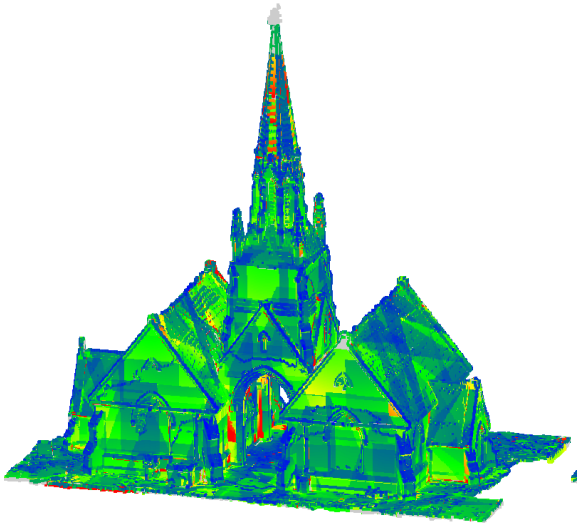
(e) $n_{\text{IMG}} = 50$

Figure 10.15: Voxels colored according to their quality estimate for n_{IMG} between 10 and 50. Gray areas correspond to control points that have only been observe once, i.e. they are estimated to not be reconstructable yet.

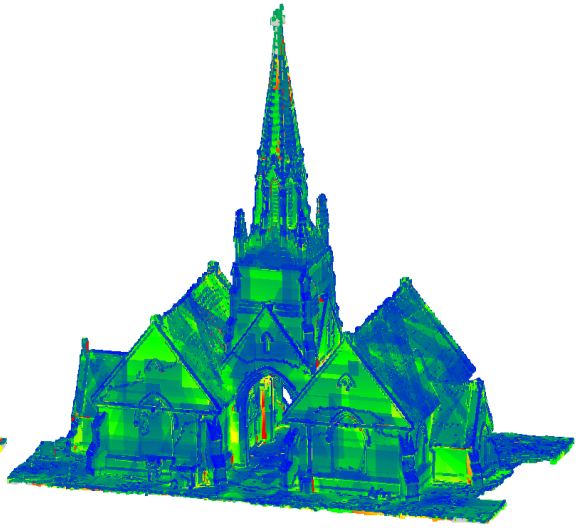




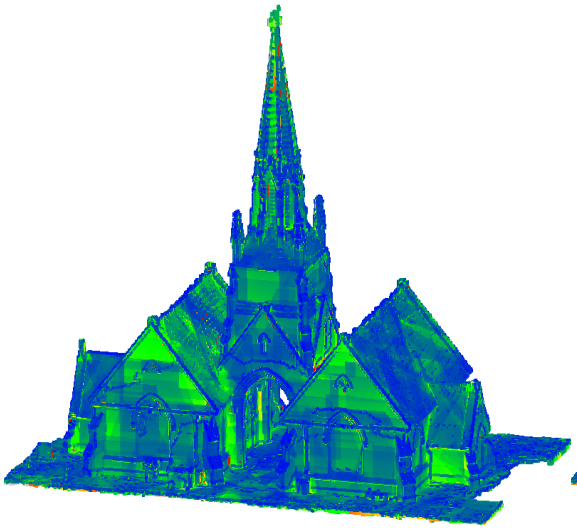
(a) $n_{\text{IMG}} = 100$



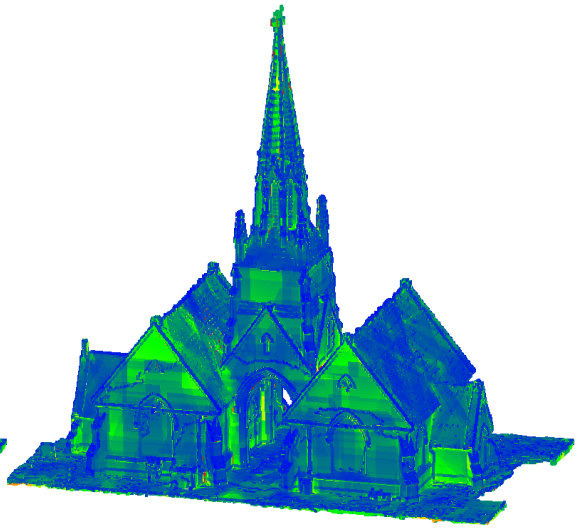
(b) $n_{\text{IMG}} = 200$



(c) $n_{\text{IMG}} = 300$

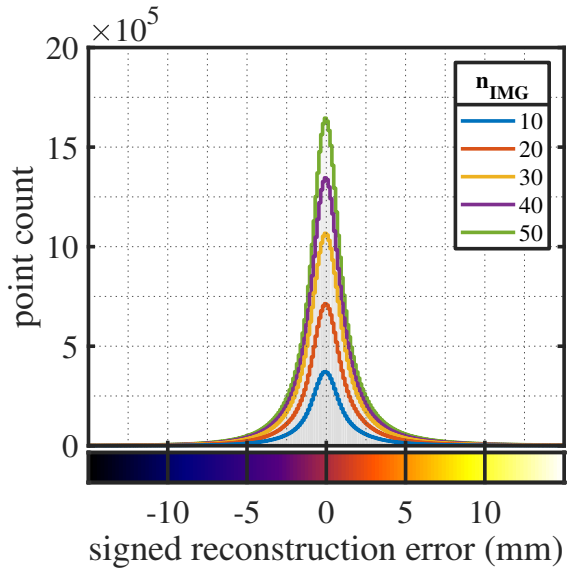


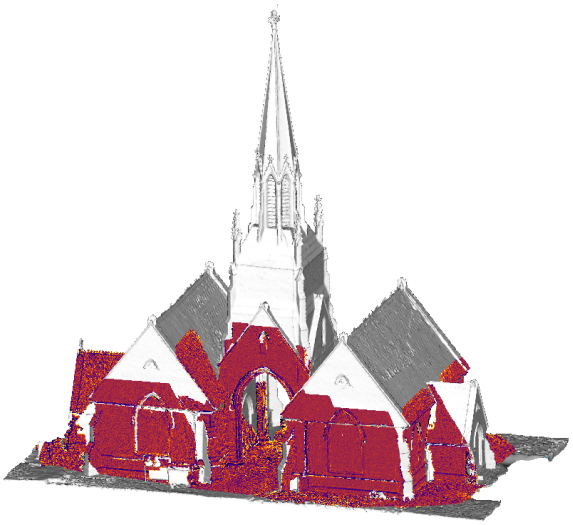
(d) $n_{\text{IMG}} = 400$



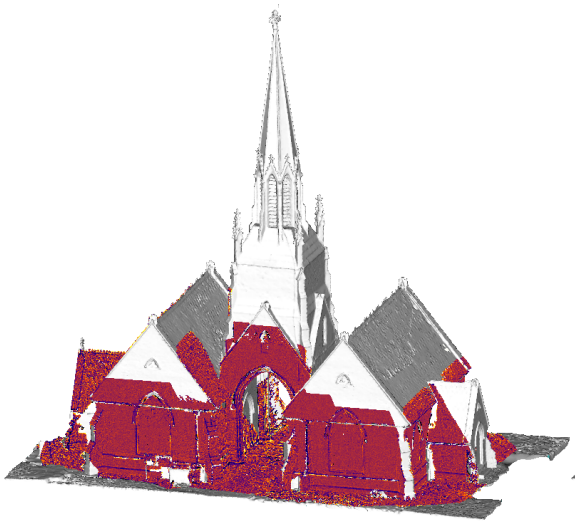
(e) $n_{\text{IMG}} = 500$

Figure 10.16: Voxels colored according to their quality estimate for n_{IMG} between 100 and 500. Gray areas correspond to control points that have only been observe once, i.e. they are estimated to not be reconstructable yet.

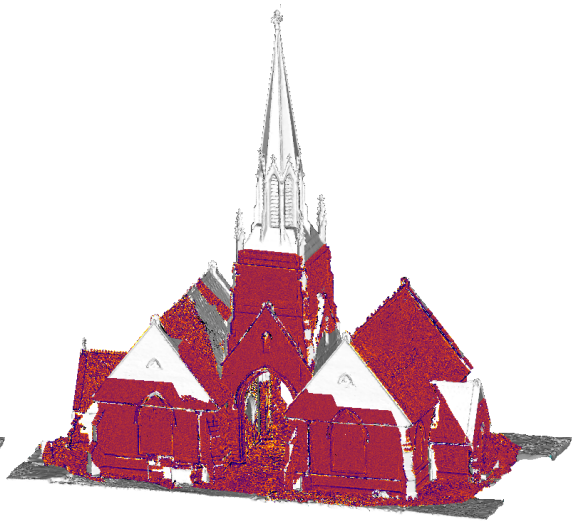




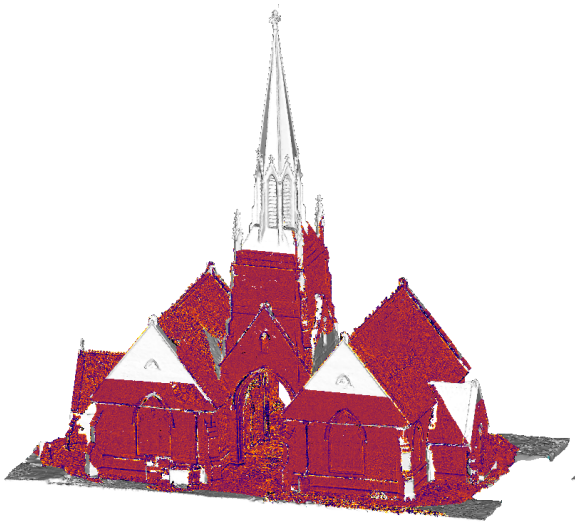
(a) $n_{\mathrm{IMG}} = 10$



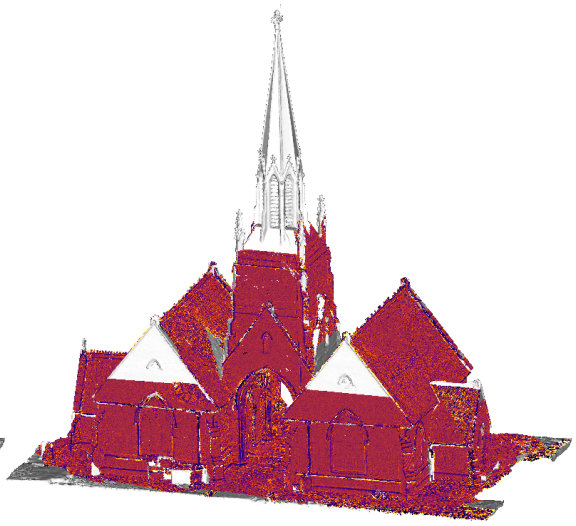
(b) $n_{\mathrm{IMG}} = 20$



(c) $n_{\mathrm{IMG}} = 30$

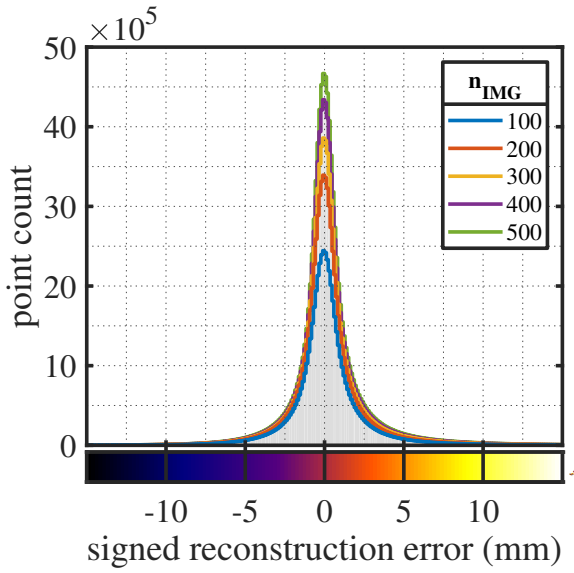


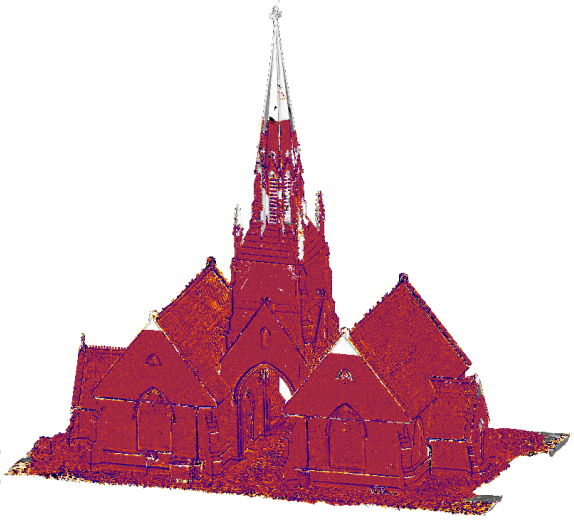
(d) $n_{\mathrm{IMG}} = 40$



(e) $n_{\mathrm{IMG}} = 50$

Figure 10.17: Projection of the SfM point cloud onto the reference mesh, colored according to the signed reconstruction error for n_{IMG} between 10 and 50.



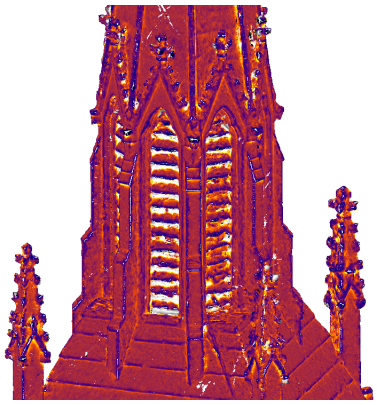


(a) $n_{\text{IMG}} = 100$

(b) $n_{\text{IMG}} = 300$

(c) $n_{\text{IMG}} = 500$

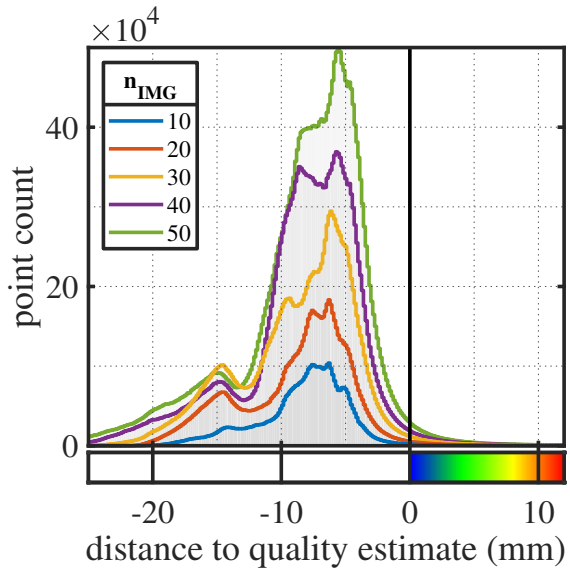
(d) $n_{\text{IMG}} = 100$

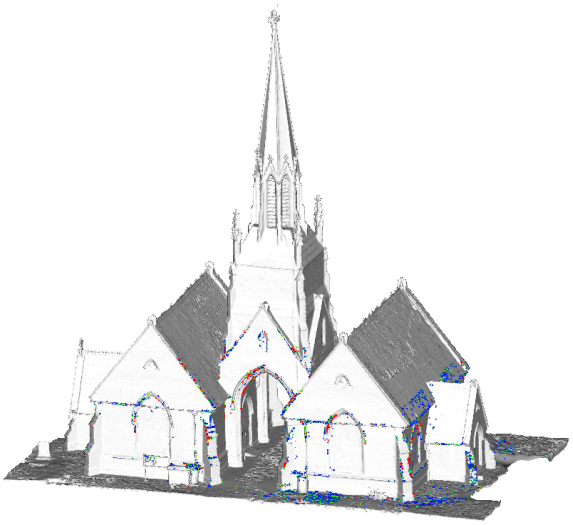


(e) $n_{\text{IMG}} = 300$

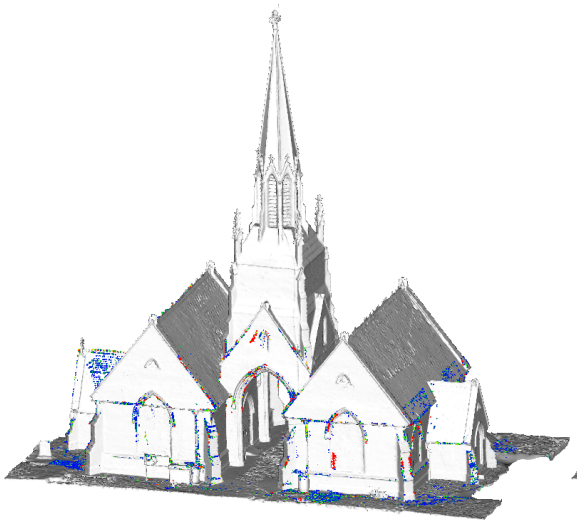
(f) $n_{\text{IMG}} = 500$

Figure 10.18: Projection of the SfM point cloud onto the reference mesh, colored according to the signed reconstruction error for n_{IMG} between 100 and 500.

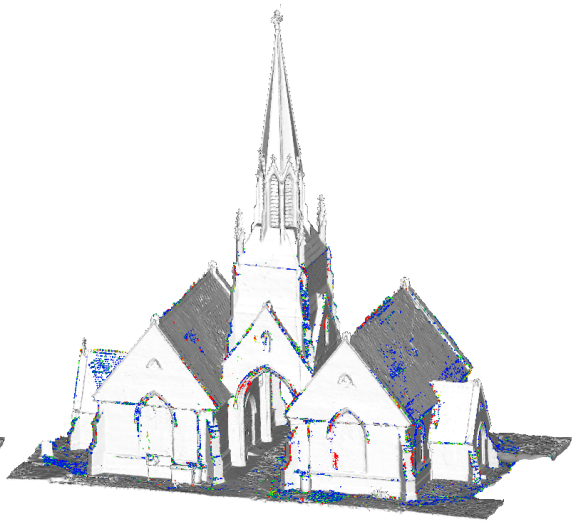




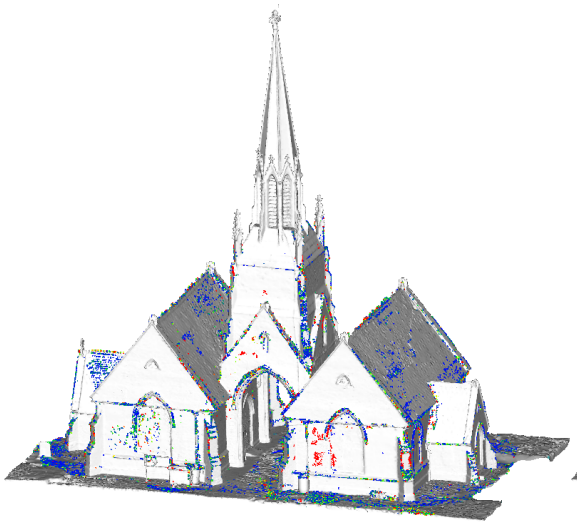
(a) $n_{\text{IMG}} = 10$



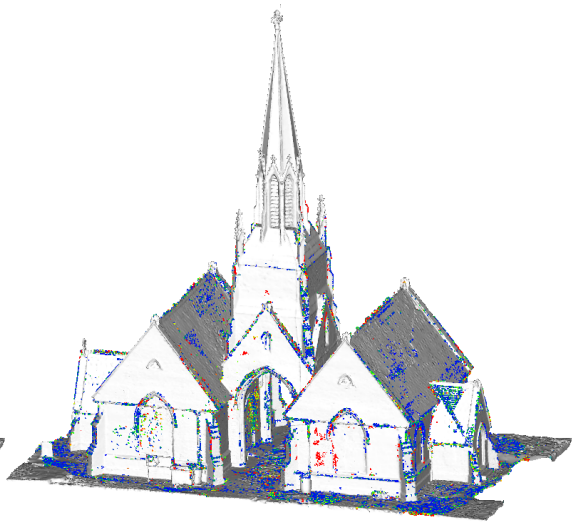
(b) $n_{\text{IMG}} = 20$



(c) $n_{\text{IMG}} = 30$

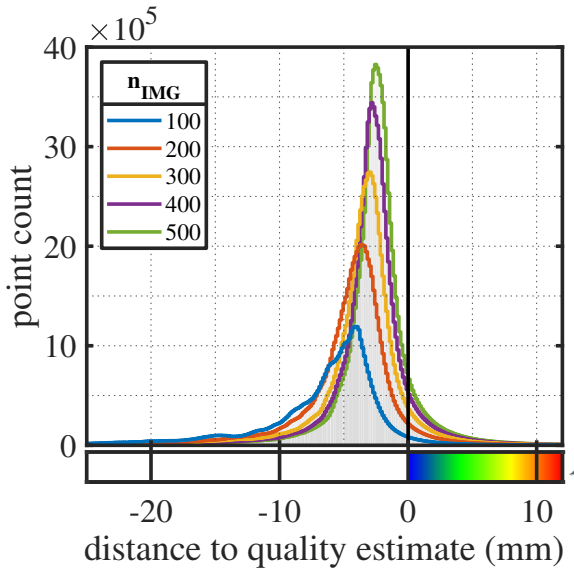


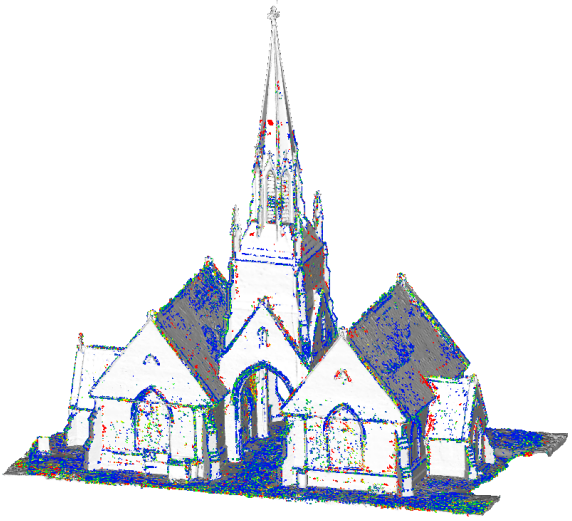
(d) $n_{\text{IMG}} = 40$



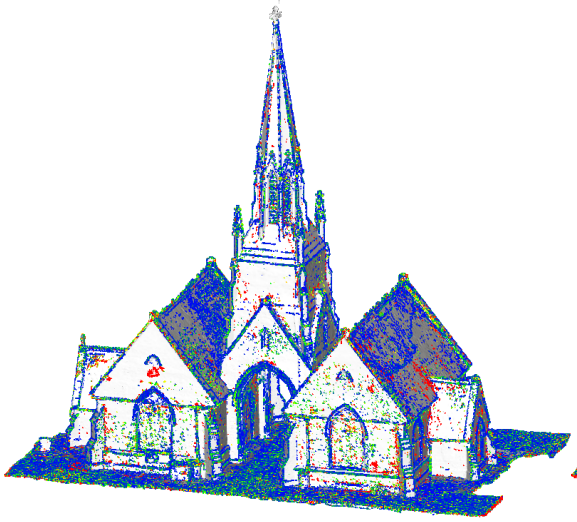
(e) $n_{\text{IMG}} = 50$

Figure 10.19: Distance to the quality estimate for n_{IMG} between 10 and 50. All projected SfM points that do not satisfy the predicted quality estimate are visualized.



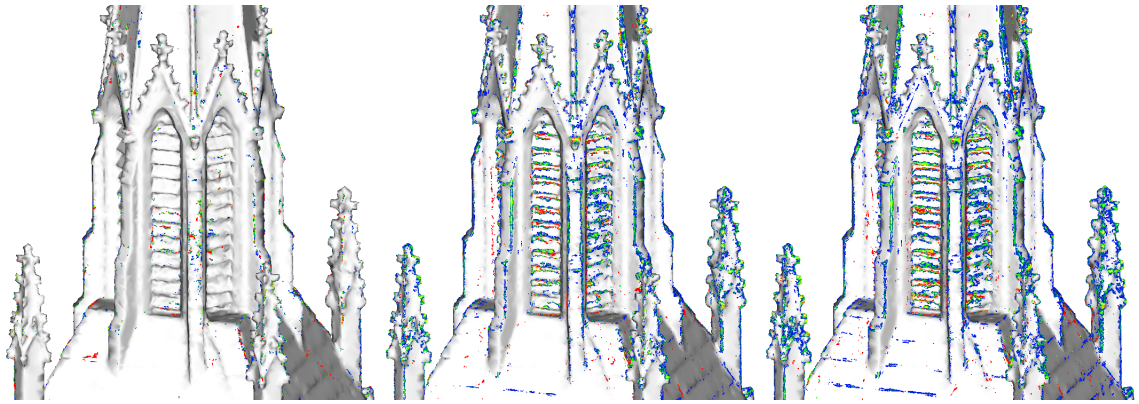


(a) $n_{\text{IMG}} = 100$



(b) $n_{\text{IMG}} = 300$

(c) $n_{\text{IMG}} = 500$



(d) $n_{\text{IMG}} = 100$ **(e)** $n_{\text{IMG}} = 300$ **(f)** $n_{\text{IMG}} = 500$

Figure 10.20: Distance to the quality estimate for n_{IMG} between 100 and 500. All projected SfM points that do not satisfy the predicted quality estimate are visualized.

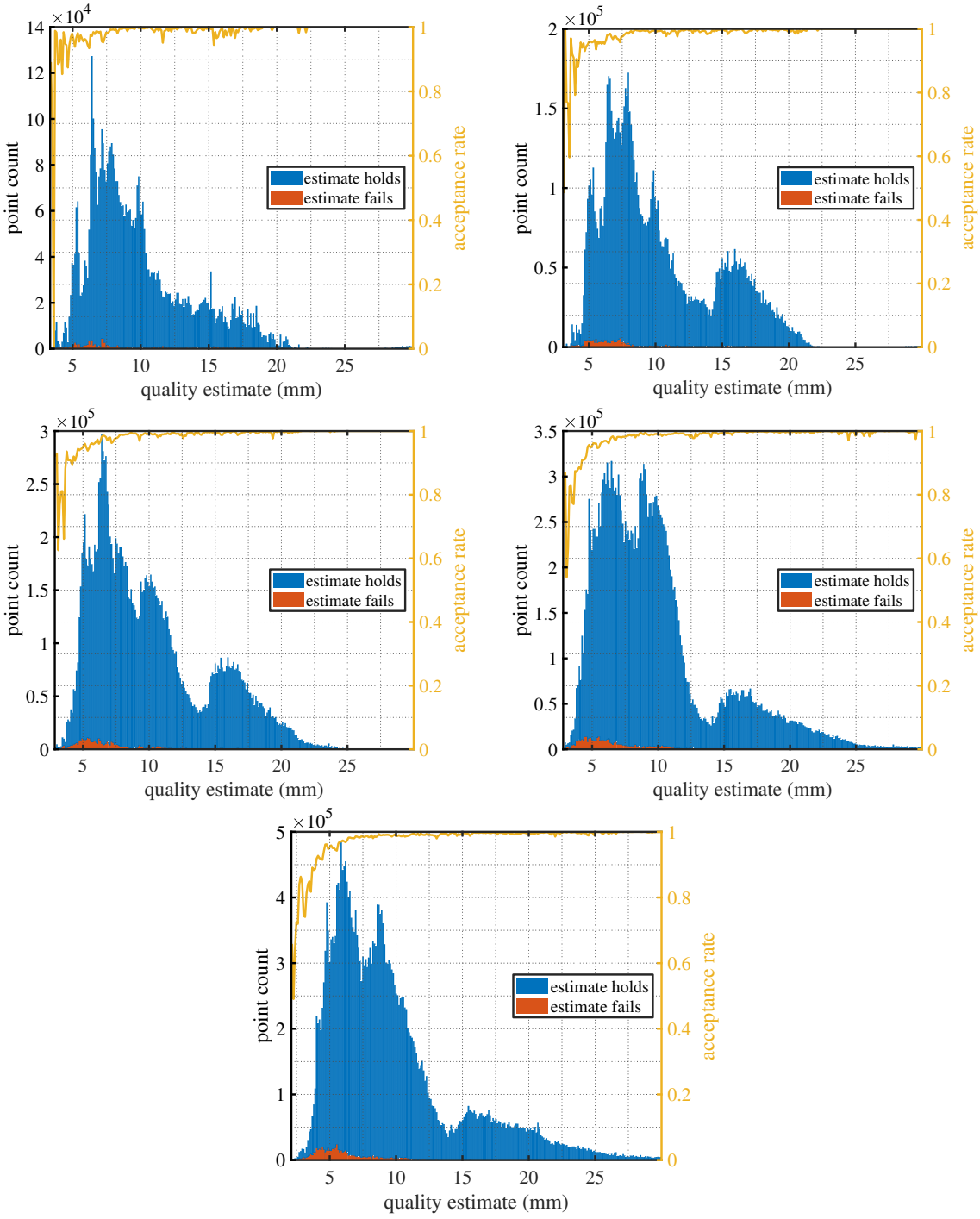


Figure 10.21: Quality estimate of SfM points divided into classes where the estimate holds and where the estimate fails for n_{IMG} between 10 and 50.

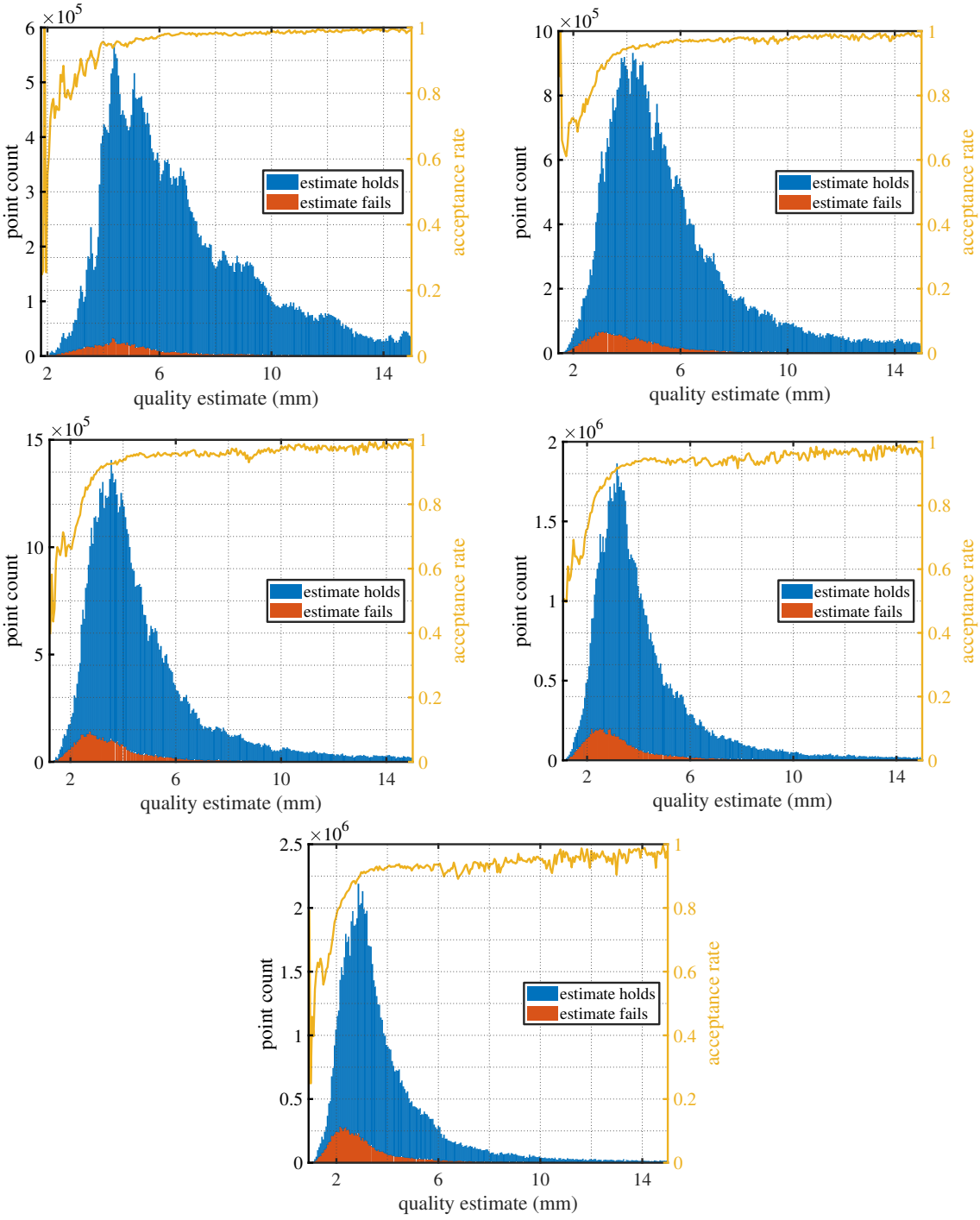


Figure 10.22: Quality estimate of SfM points divided into classes where the estimate holds and where the estimate fails for n_{IMG} between 100 and 500.

10.3 Tyche Sculpture

The computational complexity of the SfM-CPP algorithm is proportional to the number of voxels observed from cameras. Therefore, especially a lower voxel discretization resolution could significantly improve the overall runtime. However, increasing the size of the individual voxels could deteriorate the accuracy of the quality estimation due to higher observability and discretization errors. In this simulation example, the effects of different voxel discretization resolutions on quality estimate and the resulting SfM reconstruction are evaluated.

Model

We consider a marble sculpture from the second century B.C. of Tyche, the Greek goddess of destiny and fortune. It is currently located in the Musee du Cinquantenaire in Brussels, Belgium. A high detail 3D reconstruction of this sculpture (see figure 10.23), which is licensed under [CC BY-NC 4.0, 2013], was obtained from [Marchal, 2018]. It consists of 563 890 faces, 282 696 vertices and high resolution textures. The true dimensions of the model were unknown to us, however we estimated its scale from photos. Hence, it was scaled to a bounding box of $0.77\,\mathrm{m} \times 0.38\,\mathrm{m} \times 1.70\,\mathrm{m}$ (width $\times$ depth $\times$ height). Furthermore, it has been transformed to be axis-aligned and stands on the xy plane at $z = 0$. This model is the smallest in scale we chose as reference mesh for our simulations. Therefore, we can experiment with much lower voxel discretization resolutions without running out of memory or experiencing exorbitant runtime.

Figure 10.23: Tycho sculpture reference mesh. Left: full view. Right: detailed view.

SfM-CPP Algorithm

The camera is constrained to a minimum distance of 0.2 m and a maximum distance of 2 m to the mesh, and a minimum altitude of 0.1 m. Its roll and pitch angles are fixed to 0°. We consider 8000 RRT samples (1000 position samples, each with 8 yaw samples) that satisfy these geometric constrains for each NBV iteration. The required image overlap with previously recorded photos $o_{\min}$ is set to 20 %. The simulation utilizes the `CAM_MR` camera with gain clamping parameters given in table 10.6. In total, the SfM-CPP algorithm is run for 50 iterations for three different voxel discretization resolutions (10 mm, 5 mm, 2.5 mm). In order to allow for a better comparison of observability properties, an additional estimate of the reconstruction quality is created with a voxel size of 2.5 mm and camera poses identical to the 10 mm simulation. This simulation is identified by an additional asterisk, i.e. 2.5*mm.

The gain objective function term is given in figure 10.25. The "smoothness" and almost-monotonicity of the curves indicates that the approximate solution of the OED problem (7.3) using RRTs is close to a global solution. Note that the curves, corresponding to different voxel discretization sizes, only differ approximately by a constant linear scaling factor. This comes to no surprise, because by doubling the voxels' cube length we increase the number of voxels on the surface by roughly four. A similar scaling can be observed when looking at the individual runtime of each simulation given in table 10.7.

Table 10.6: Clamping parameters (see section 6.3.2).

Parameter	Value (`CAM_MR`)
n_{pix}	3
pixel error confidence	95 %
$d_{\min}$	1 m
$\lambda_{\min}$	1.315×10^{-6}
corresponding 99 % confidence interval	$\pm$ 0.30 cm
$d_{\max}$	25 m
$\lambda_{\max}$	8.219×10^{-4}
corresponding 99 % confidence interval	$\pm$ 7.39 cm

Evaluation

The quality estimate of individual voxels is visualized in figure 10.26. There, the histograms have almost identical shapes. The associated renderings are also almost identically colored, despite the strongly varying voxel sizes. Especially the shoulder section is estimated to have a bad reconstruction quality. This observation is consistent with the signed reconstruction errors in figure 10.27, where the same section

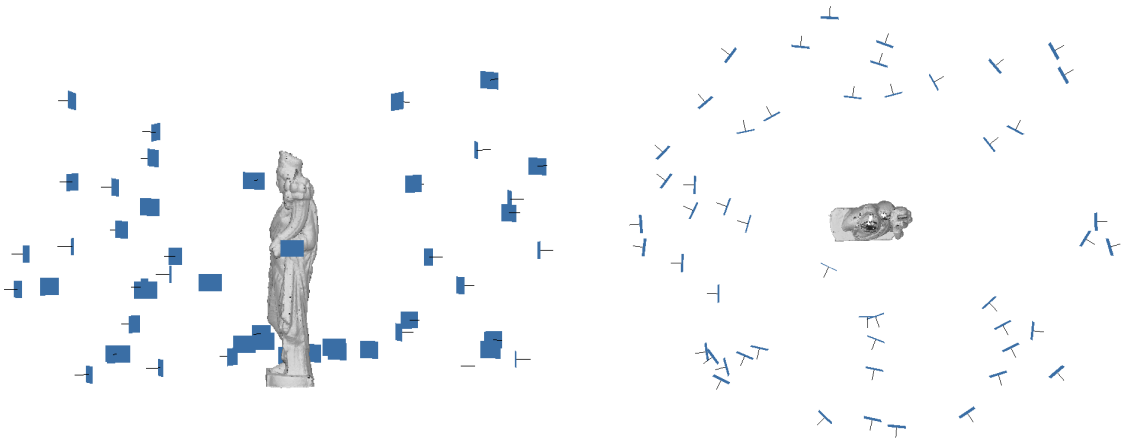


Figure 10.24: Camera positions after 50 SfM-CPP iterations for the 10 mm voxel resolution simulation. Left: side view, right: top view.

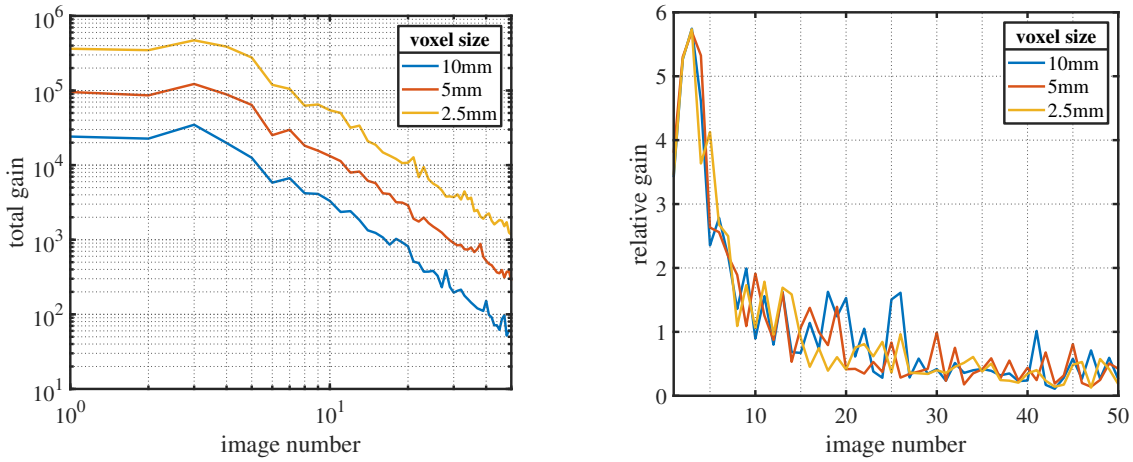


Figure 10.25: Gain objective function term (7.3a) for each selected NBV. Left: Logarithmic plot. Right: Relative gain, i.e. normalized by the number of all voxels that contribute to the gain with a term greater than zero.

shows deficiencies in the reconstruction. Ultimately, the quality estimate fits the reconstruction error very well. For example, consider folds in the dress and fruits in the basket. Also, the histograms of the signed reconstruction error and the quality estimates have the same order of magnitude.

The comparability of all different voxel sizes becomes more apparent when looking at the distance to the quality estimate plots in figure 10.28. With the exception of a few outliers where the visualization differs, areas with high point densities are located very similarly. Even the magnitude of the quality estimate violation is comparable in these regions. The histograms are also almost identical, especially for the 10 mm and 2.5*mm voxel sizes. The acceptance rates in figure 10.29 and table 10.8 further show the huge similarity between the conducted simulations.

These simulations have shown that the control point density on the surface mesh does not need to be too large, as the obtained results are almost identical beyond a certain point. The voxel size should therefore be as large as possible, in order to reduce the runtime of the algorithm, but still be small enough to sufficiently capture significant details.

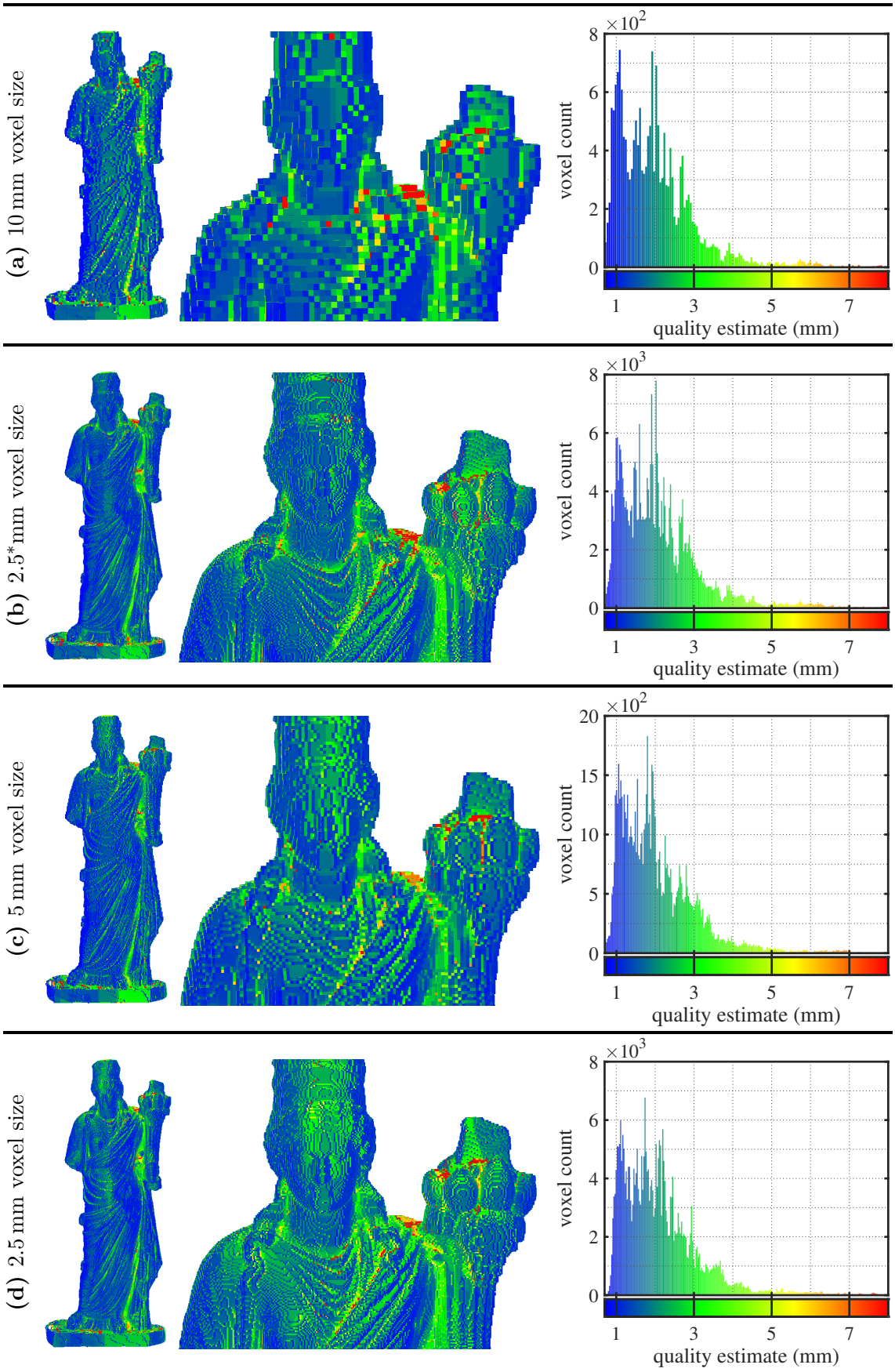
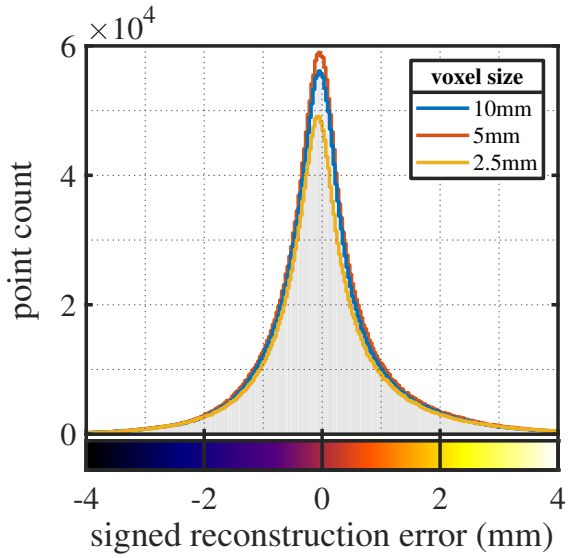


Figure 10.26: Voxels colored according to their quality estimate for different voxel discretization resolutions.



(a) 10 mm voxel size

(b) 5 mm voxel size

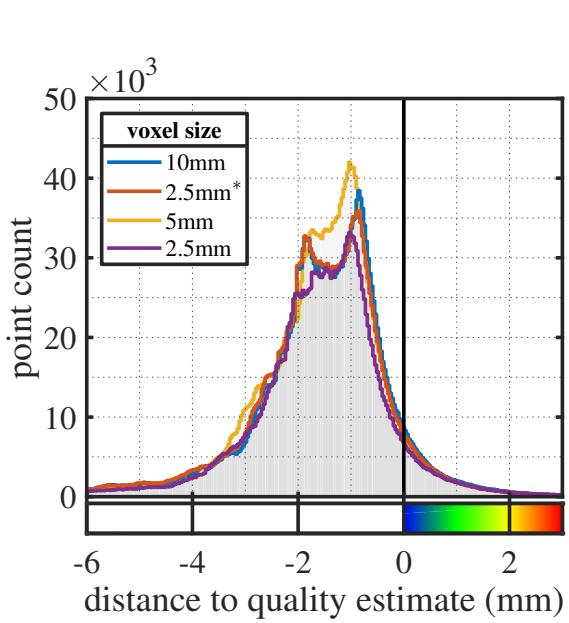
(c) 2.5 mm voxel size

(d) 10 mm voxel size

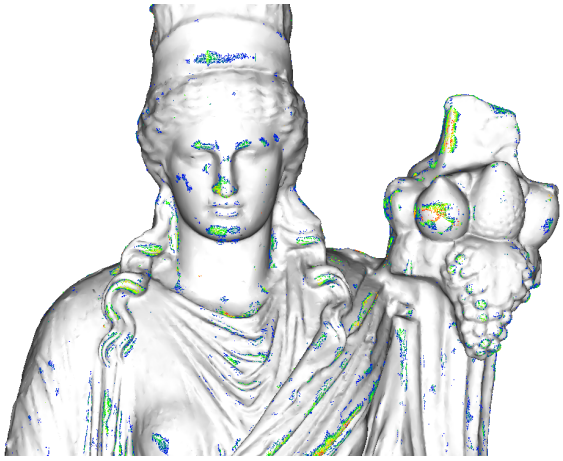
(e) 5 mm voxel size

(f) 2.5 mm voxel size

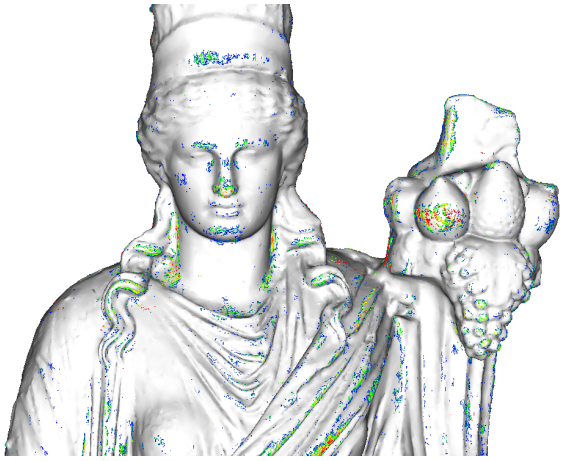
Figure 10.27: Projection of the SfM point cloud onto the reference mesh, colored according to the signed reconstruction error for different voxel discretization resolutions.



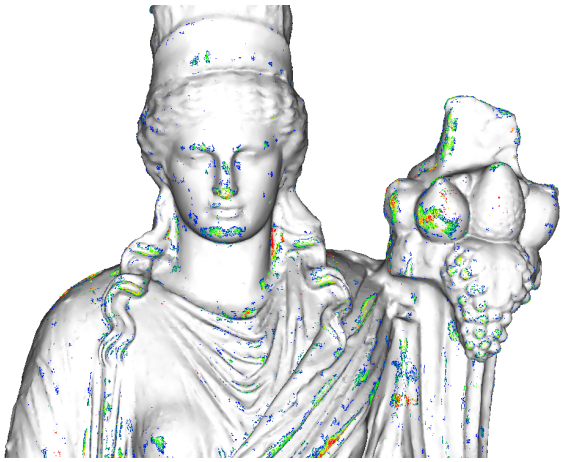
(a) 10 mm voxel size **(b)** 2.5*mm voxel size



(c) 10 mm voxel size



(d) 2.5*mm voxel size



(e) 5 mm voxel size

(f) 2.5 mm voxel size

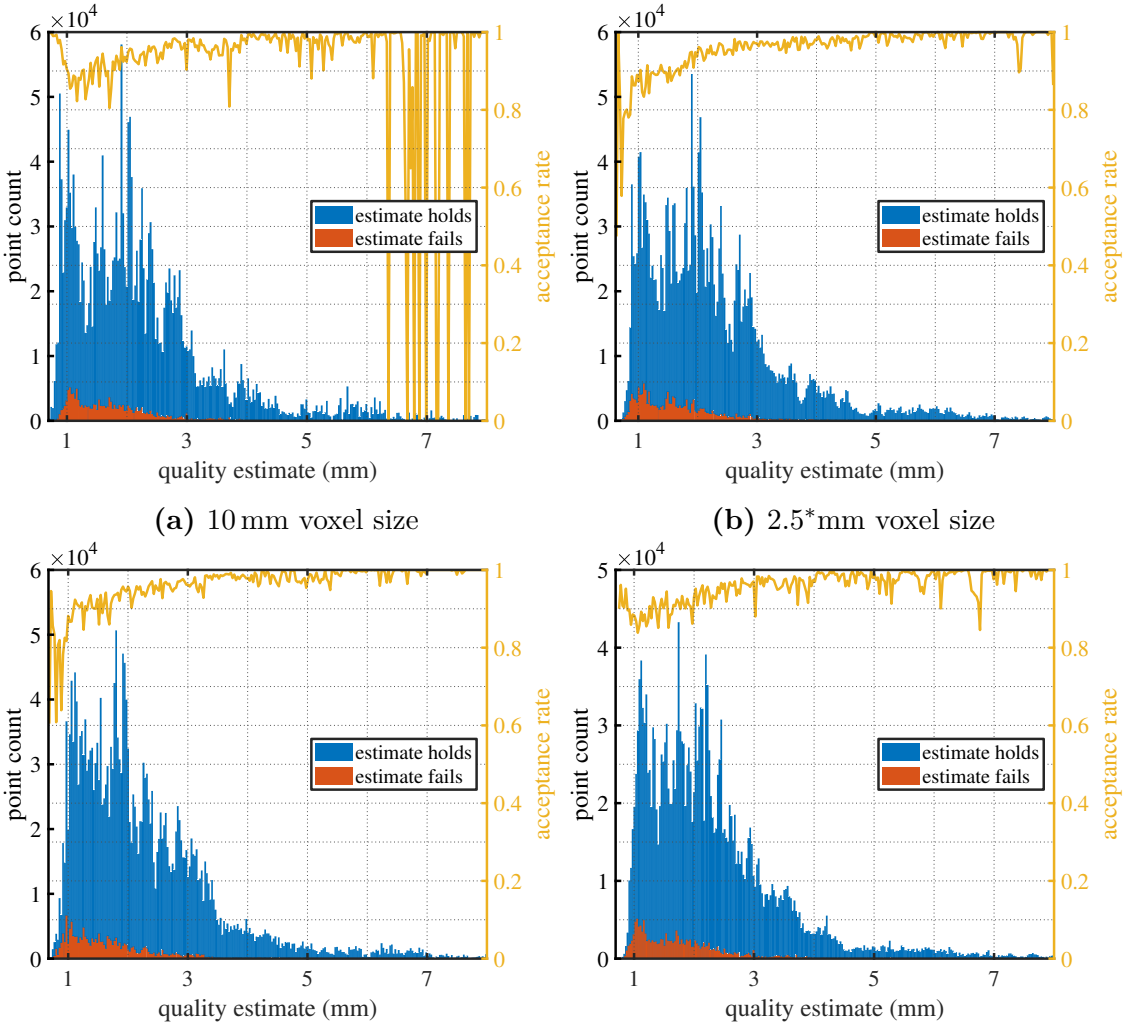
Figure 10.28: Distance to the quality estimate for different voxel discretization resolutions. All projected SfM points that do not satisfy the predicted quality estimate are visualized.

Table 10.7: Runtime information. L denotes the number of RRT samples used in the `getNextSegment` subroutine.

voxel size	average getNextSegment runtime
10 mm	$L \cdot$ 1.58 ms
5 mm	$L \cdot$ 5.83 ms
2.5 mm	$L \cdot$ 25.82 ms

Table 10.8: Total acceptance rate of the quality estimate. The point cloud size only contains points that could be associated with a control point.

voxel size	point cloud size	acceptance rate
10 mm	2 390 758	92.72 %
2.5*mm	2 387 441	93.20 %
5 mm	2 534 972	93.45 %
2.5 mm	2 136 472	92.95 %



(a) 10 mm voxel size

(b) 2.5*mm voxel size

(c) 5 mm voxel size

(d) 2.5 mm voxel size

Figure 10.29: Quality estimate of SfM points divided into classes where the estimate holds and where the estimate fails for different voxel discretization resolutions.

10.4 Roman Temple of Évora

In this simulation, we consider a large model with difficult topological and observability properties. The camera pose is intentionally constraint such that certain regions are not observable. The main focus here lies on analyzing the total coverage of the reconstruction, both for the whole model and details.

Model

We consider the 3D model of the Roman Temple of the Portuguese city of Évora in the Alentejo region. Presumably constructed by the Romans around the first century A.D. the building experienced multiple modifications and adaptions for different use cases throughout the centuries. Its current form was achieved in the 19th century, when all non-roman elements were removed. The temple is part of historical town center, a UNESCO World Heritage Site. A 3D model of the structure is obtained from [Global Digital Heritage, 2019], where it is available for free and licensed under [CC BY-NC 4.0, 2013]. It is already scaled correctly according to the real structures dimensions with a size of 25.94 m × 15.64 m × 13.27 m (width × depth × height). Hence, only slight modifications to the model were made. We removed any stray vertices that were not part of a face, such that we obtain a total vertex count of 1 063 843 and an unmodified number of 2 125 853 faces. Additionally, the mesh is translated along the z-axis to stand on the xy plane at $z = 0$. The resulting reference mesh is visualized in figure 10.30.

Figure 10.30: Roman Temple of Évora reference mesh. Left: Overview of the whole model. Right: Detailed view of a single capital.

SfM-CPP Algorithm

The utilized `CAM_MR` camera is constraint to have a minimum distance of 1 m and a maximum distance of 5 m to the mesh. Its minimum altitude is set to 1 m. The pitch angle is fixed to 30°, the roll angle to 0°. Note that by the choice of this slanted camera view, we intentionally create areas on the mesh that are not observable, e.g. the bottom side of the stone beams. This should later be correctly

reflected in the quality estimate, signed reconstruction errors and SfM point cloud. As for the other simulations, 8000 poses are obtained from the RRT algorithm in the `getNextSegment` algorithm 7.2. These are composed of 1000 poses that meet the geometric constraints, each with 8 sampled yaw angles. The required minimal image overlap o_{min} is set to 20 %. The utilized clamping parameters are given in table 10.9. We run the SfM-CPP algorithm for a total of 500 images using a voxel discretization with a voxel resolution of 5 cm. The resulting NBV camera poses are visualized in figure 10.31. The corresponding gain objective function values are given in figure 10.32.

The runtime of each evaluation of the `getNextSegment` subroutine was measured to be $L \cdot 10.16\,\text{ms}$, with the number of RRT samples L. This performance is similar to the one of Holbeach Cemetery Chapel, since due to the tilted view a large number of voxels are visible for the camera poses on average. The pillars also contribute to this runtime, as they increase the complexity of ray casting computations.

Table 10.9: Clamping parameters (see section 6.3.2).

Parameter	Value (`CAM_MR`)
n_{pix}	3
pixel error confidence	95 %
d_{min}	2 m
λ_{min}	5.260×10^{-6}
corresponding 99 % confidence interval	± 0.59 cm
d_{max}	50 m
λ_{max}	3.288×10^{-3}
corresponding 99 % confidence interval	± 14.77 cm

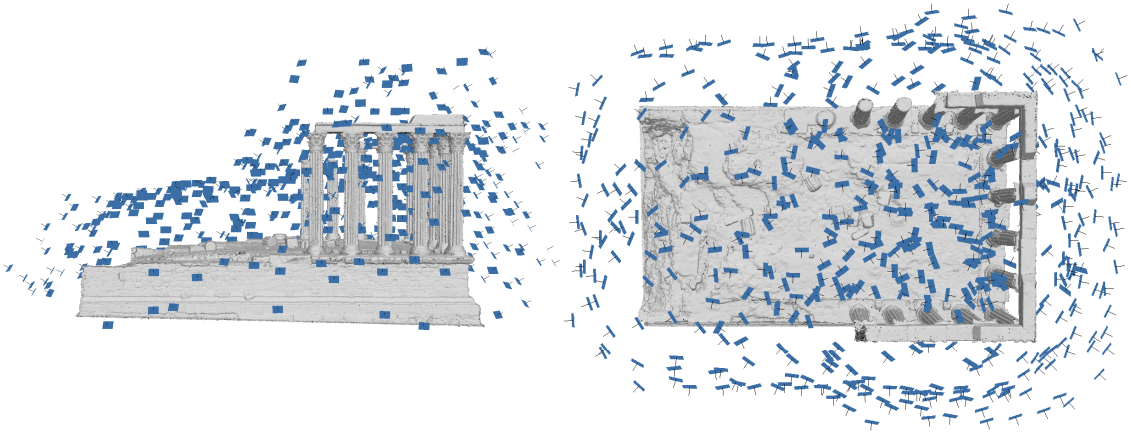


Figure 10.31: Camera positions after 500 SfM-CPP iterations. Left: side view, right: top view.

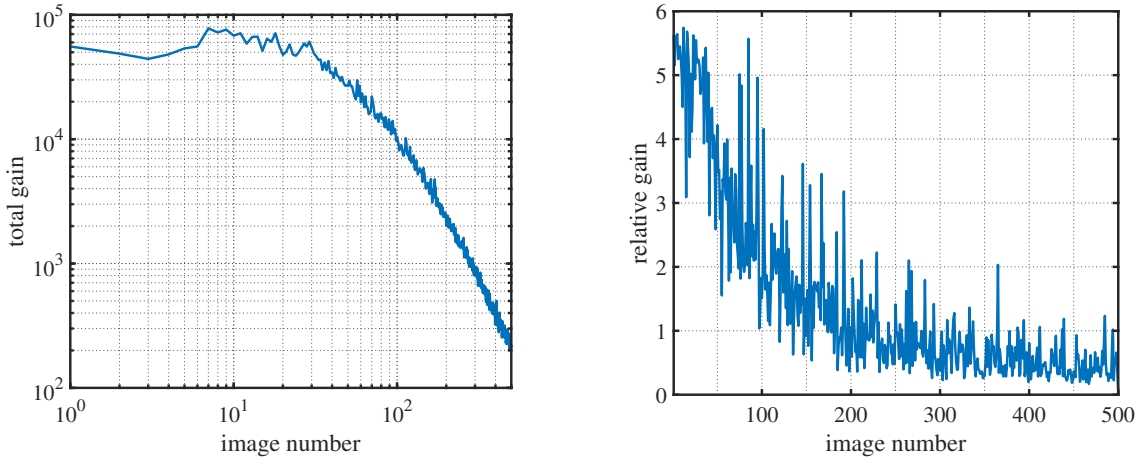


Figure 10.32: Gain objective function term (7.3a) for each selected NBV. Left: Logarithmic plot. Right: Relative gain, i.e. normalized by the number of all voxels that contribute to the gain with a term greater than zero.

Evaluation

The quality estimate is given in figure 10.33 and can be compared to the signed reconstruction error in figure 10.34. Recessed areas in the columns are expected to have a higher reconstruction error than the rest of the columns, what is correctly reflected in the computed signed reconstruction error. However, the magnitude of the quality estimate is smaller than the magnitude of the signed reconstruction error in many areas. This is further highlighted by the distance to the quality estimate in figure 10.35. There, a lot of points violate the quality estimate (see table 10.10 for acceptance rate). Consider the side wall and the floor of the temple. The signed reconstruction error for both sections is similar in figure 10.34. However, there is a large discrepancy between their respective quality estimates in figure 10.33. Note that the quality estimate of the side wall roughly fits the reconstruction error, which becomes apparent in figure 10.34. There, as usual, edges are the biggest violators. It is therefore necessary to explain why the estimate for the floor does not hold for this high number of points. In fact, we observe the same effect as in the Holbeach Cemetery Chapel simulation. This area is visible in a large number of photos. However, only a fraction of these photos contributes to the reconstruction, due to discrepancies between expected and real observability of individual points and effects from the SfM reconstruction algorithm. This effect can further be observed in the acceptance rate plot in figure 10.37 and is explained in more detail in the evaluation section of the Holbeach Cemetery Chapel simulation. Nevertheless, it most prominently appears for areas that are already far beyond the desired reconstruction quality defined by λ_{min} (see table 10.9), what can be verified in figure 10.37.

Through a visual inspection, we verify that the entire geometry has been completely captured with a single exception. This exception is highlighted in figure 10.36, which shows the bottom of the stone beam together with a detailed view of a single capital. As expected, the bottom segments could not be reconstructed due to the fixed pitch angle of 30°. This applies equally to small details of the capital, which would require other camera pitch angles. This suggests that the SfM-CPP algorithm achieves full mutli-view coverage up to geometric constraints, i.e. robot constraints and geometric limitations of the structure.

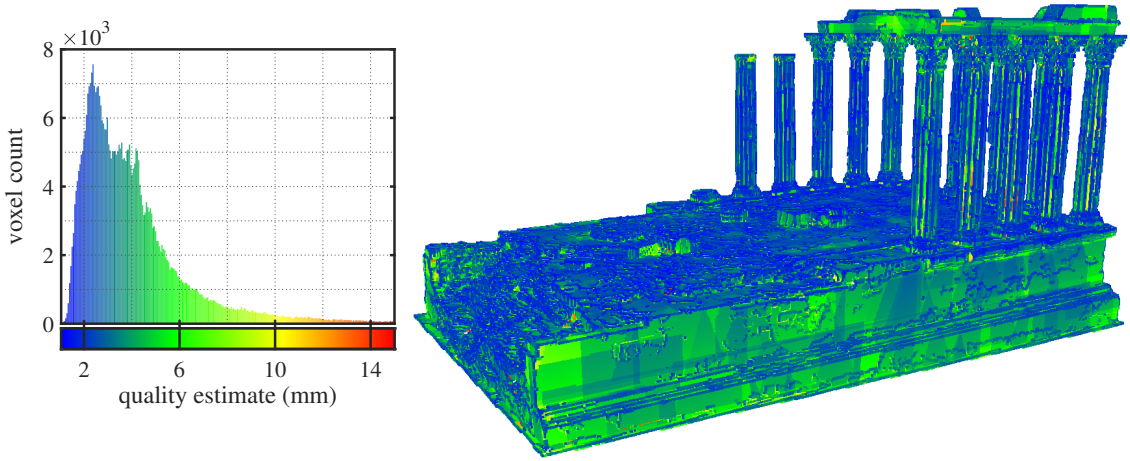


Figure 10.33: Voxels colored according to their quality estimate.

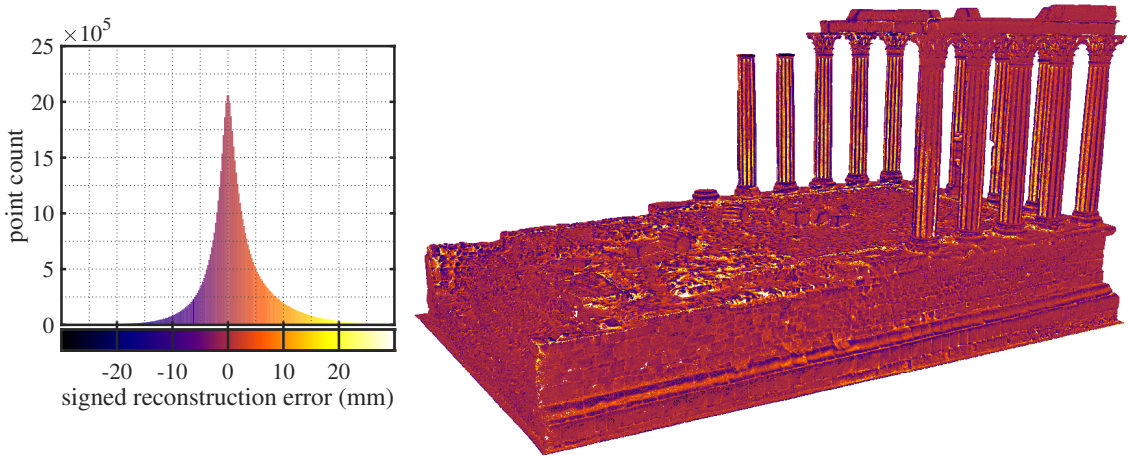


Figure 10.34: Projection of the SfM point cloud onto the reference mesh, colored according to the signed reconstruction error.

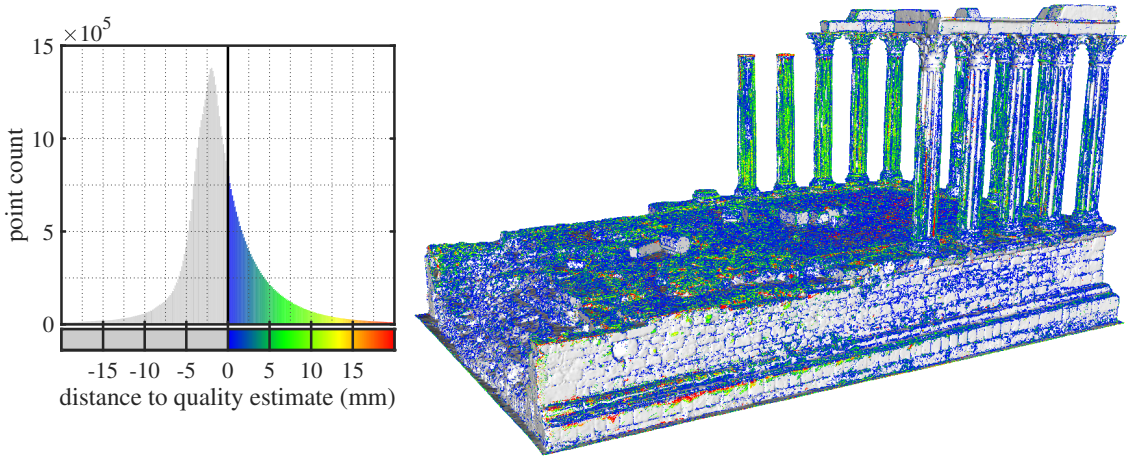


Figure 10.35: Distance to the quality estimate. All projected SfM points that do not satisfy the predicted quality estimate are visualized.

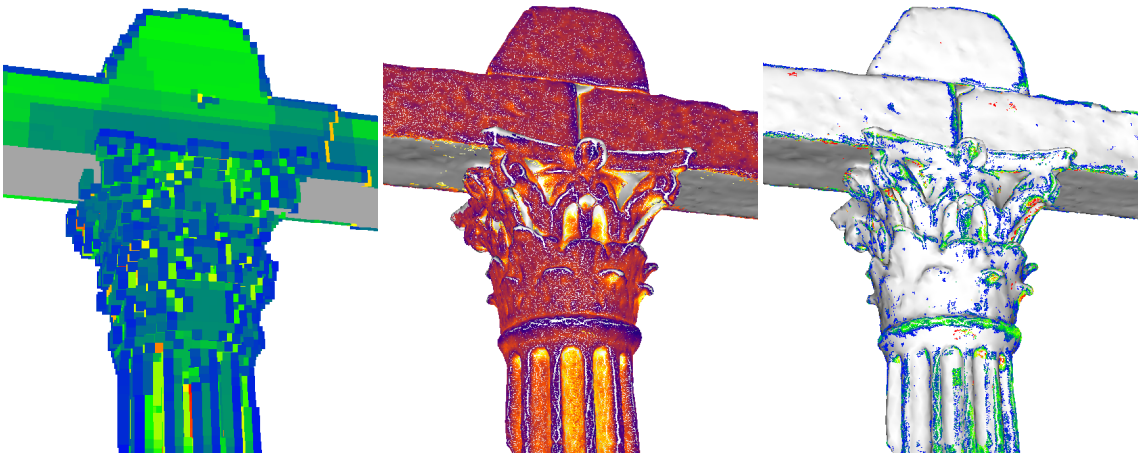


Figure 10.36: Detailed view of a single capital. Left: Voxels colored according to their quality estimate (same colors as the histogram in figure 10.33). The gray voxels have not been observed at all. Middle: Projected signed reconstruction error (see figure 10.34). Right: Distance to quality estimate (see figure 10.35).

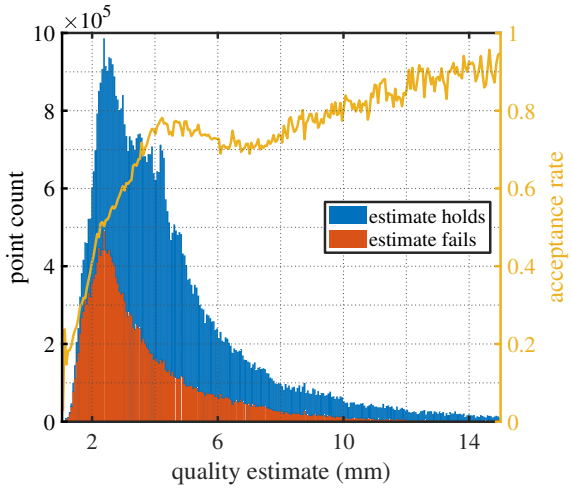


Figure 10.37: Quality estimate of SfM points divided into classes where the estimate holds and where the estimate fails.

Table 10.10: Total acceptance rate of the quality estimate. The point cloud size only contains points that could be associated with a control point.

point cloud size	acceptance rate
61 343 840	65.81 %

Part III

Towards Autonomous Gaussian Process Implicit Surface Next-Best-View Planning

GPIS Surface Estimation

11

A surface can implicitly be described by a function. Given a set of training points, such a function can be given by the posterior mean of a Gaussian process. This method of implicit surface modeling is called *Gaussian process implicit surface* (GPIS) [Williams and Fitzgibbon, 2007]. It combines the great inter- and extrapolation properties of Gaussian process regression with a (theoretical) infinite level of detail for the surface estimation, as no grid discretization is required. Measurement errors are also modeled and will be used to control the smoothness of the resulting surfaces. In addition, variance information of surface points can easily be computed. They can later be used as a quality measure for the surface quality. Then an (entropy based) gain formulation similar to section 6.3.1 can be defined and used in the context of NBV planning.

In this chapter we will cover the theory of GPIS using polyharmonic kernels. Since the original paper [Williams and Fitzgibbon, 2007] is rather short and contains some errors, we will give a clean, more detailed mathematical derivation. They proposed to utilize polyharmonic kernels that originate from polyharmonic spline interpolation for the task of implicit surface modeling. These give the GP the property to keep the curvature or orientation of the implied surfaces almost constant during surface extrapolation. First, the connection between Gaussian processes and regularized fitting problems is derived. Then the regularization term of the polyharmonic spline interpolation is used to construct polyharmonic kernels. Finally, we allow for derivative observations, i.e. observations of the surface curvature to eliminate the need of interior and exterior control points during the training of the Gaussian process. We use the same notation as in chapter 5.

For the remainder of this thesis, implicit surfaces are defined as follows. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a function. Then it defines an implicit surface as the set of points $\{x \in \mathbb{R}^d \mid f(x) = 0\}$. Moreover, we define interior and exterior of this surface as

$$f(x) \begin{cases} > 0 & x \text{ is an exterior point} \\ = 0 & x \text{ is a surface point} \\ < 0 & x \text{ is an interior point.} \end{cases} \tag{11.1}$$

11.1 Duality to Regularization Formulation

In this section we will discover that a Gaussian process can be interpreted as a regularized spline interpolation problem, following [Rasmussen and Williams, 2006, chapter 6] and [Wahba, 1990, chapter 1]. This will allows us to model custom Gaussian processes where a desired behavior can be controlled by the regularization term,

or to find out more about properties of existing processes. In order to gain these insights, we musts first relate kernel functions to reproducing kernels of reproducing kernel Hilbert spaces:

Definition 11.1 (Reproducing kernel Hilbert space (RKHS))**.** Let $\mathcal{H}$ be a Hilbert space of real-valued functions defined on an arbitrary set $\mathcal{X}$. Then $\mathcal{H}$ is called *reproducing kernel Hilbert space* if there exists a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, called the *reproducing kernel* of $\mathcal{H}$, with

1. $k(x, \cdot) \in \mathcal{H} \quad \forall x \in \mathcal{X}$ and
2. k has the reproducing property $\langle f(\cdot), k(x, \cdot)\rangle_{\mathcal{H}} = f(x)$. Note that this especially implies that $\langle k(x, \cdot), k(\cdot, x')\rangle_{\mathcal{H}} = k(x, x')$.

From definition 11.1 it follows directly that all reproducing kernels of RKHS must be symmetric and positive definite functions. As it turns out, this observation is bidirectional according to the Moore-Aronszajn theorem:

Theorem 11.1 (Moore-Aronszajn theorem [Aronszajn, 1950])**.**
Let $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric, positive definite function on a set $\mathcal{X}$. Then there exists a unique Hilbert space for which k is its reproducing kernel.

The relation of RKHS to regularized spline interpolation is given by the representer theorem:

Theorem 11.2 (Representer theorem [Wahba, 1990, chapter 1])**.**
Let $\mathcal{H} = \mathcal{H}_0 \oplus \mathcal{H}_1$ be a RKHS and P_1 be the orthogonal projection from $\mathcal{H}$ onto $\mathcal{H}_1$. Let $\mathcal{L}_i$, $i = 1, \ldots, n$ be bounded linear functionals on $\mathcal{H}$ with given measurement observations

$$f_i = \mathcal{L}_i f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, n. \tag{11.2}$$

Then the function $f \in \mathcal{H}$ minimizing

$$\sum_{i=1}^{n} \left(f_i - \mathcal{L}_i f(x_i)\right)^2 + \lambda \|P_1 f\|_{\mathcal{H}}^2 \tag{11.3}$$

with some constant $\lambda \in \mathbb{R}$ is given by

$$f = \sum_{i=1}^{M} d_i \phi_i + \sum_{i=1}^{n} c_i \xi_i, \tag{11.4}$$

where $\phi_1, \ldots, \phi_M$ span the null space ($\mathcal{H}_0$) of P_1 and $\xi_i = P_1 \eta_i$, with η_i being the representer for $\mathcal{L}_i$ in $\mathcal{H}$, i.e.

$$\langle \eta_i, g\rangle_{\mathcal{H}} = \mathcal{L}_i g(x_i), \quad \forall g \in \mathcal{H}. \tag{11.5}$$

The weights $c = (c_1, \ldots, c_n)$ and $d = (d_1, \ldots, d_M)$ are given as solution to the linear system

$$\begin{aligned} (K + \lambda I)c + Td &= (f_1, \ldots, f_n)^T \\ T^T c &= 0, \end{aligned} \tag{11.6}$$

with

$$K = \{\langle \xi_i, \xi_j \rangle_{\mathcal{H}}\}_{i=1\,j=1}^{n\quad n} \quad \in \mathbb{R}^{n \times n} \tag{11.7}$$

$$T = \{L_i \phi_j(x_i)\}_{i=1\,j=1}^{n\quad M} \quad \in \mathbb{R}^{n \times M}. \tag{11.8}$$

What is impressive about the representer theorem is that it explicitly states a solution to an infinite-dimensional minimization problem in a finite-dimensional subspace. Under certain conditions, this solution can be formulated in terms of reproducing kernels. Consider the special case with $\mathcal{H}_0 \perp \mathcal{H}_1$, where k_0 is the reproducing kernel of $\mathcal{H}_0$ and k_1 the reproducing kernel of $\mathcal{H}_1$. Then the reproducing kernel of $\mathcal{H}$ is given by $k = k_0 + k_1$. Using the reproducing property of k and the fact that η_i is the representer of $\mathcal{L}_i$ we obtain the explicit form

$$\xi_i(x) = \langle P_1 \eta_i, k(x, \cdot) \rangle_{\mathcal{H}} = \langle \eta_i, P_1 k(x, \cdot) \rangle_{\mathcal{H}} = \mathcal{L}_i^2 k_1(x, x_i). \tag{11.9}$$

Furthermore, as $\xi_i \in \mathcal{H}_1$ and $\xi_i - \eta_i \in \mathcal{H}_0$,

$$\langle \xi_i, \xi_j \rangle_{\mathcal{H}} = \langle \xi_i + \eta_i - \eta_i, \xi_j \rangle_{\mathcal{H}} = \langle \eta_i, \xi_j \rangle_{\mathcal{H}} = \mathcal{L}_i \xi_j(x_i) = \mathcal{L}_i^1 \mathcal{L}_i^2 k_1(x_i, x_j). \tag{11.10}$$

This allows us to directly evaluate calculate (11.7) and subsequently simplifies solving the system of equations (11.6).

Since all kernels of Gaussian processes are positive definite, each of them can be associated with their respective RKHS. Hence, each Gaussian process can be reformulated as a smoothing spline interpolation problem and vice versa. Although they used a different terminology, this fact was first stated in [Kimeldorf and Wahba, 1971]. As a direct consequence of the representer theorem (theorem 11.2), we can formulate the following statement:

Theorem 11.3. Let $f \sim \mathcal{GP}(0, k)$ be a Gaussian process and measurement observations be given by

$$f_i = \mathcal{L}_i f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, n, \tag{11.11}$$

with the bounded linear operators $\mathcal{L}_i : \mathcal{H} \to \mathcal{H}$ and the RKHS $\mathcal{H}$ spanned by its reproducing kernel k. Then the posterior mean minimizes

$$\frac{1}{\sigma^2} \sum_{i=1}^{n} \left(f_i - \mathcal{L}_i f(x_i)\right)^2 + \|f\|_{\mathcal{H}}^2 \tag{11.12}$$

in $\mathcal{H}$.

Proof. Consider the representer theorem 11.2 with $P_1 = id$ and $\lambda = \sigma^2$. Hence, the dimension of $\mathcal{H}_0$ is 0. From (11.10) we obtain the matrix K as

$$K = \{\mathcal{L}_i^1 \mathcal{L}_j^2 k(x_i, x_j)\}_{i=1\,j=1}^{n\quad n}. \tag{11.13}$$

The minimizer of

$$\frac{1}{\sigma^2}\sum_{i=1}^{n}\left(f_i - \mathcal{L}_i f(x_i)\right)^2 + \|f\|_{\mathcal{H}}^2 \tag{11.14}$$

is then given by

$$f = \sum_{i=1}^{n} c_i \mathcal{L}_i k(x, x_i), \tag{11.15}$$

where

$$c_{1:n} = \left[K + \sigma^2 I\right]^{-1} f_{1:n}. \tag{11.16}$$

This is equivalent to the posterior mean (5.10) using theorem 5.1. ■

There is also a similar analogy for $P_1 \neq id$ and non zero mean (see [Wahba, 1990, chapter 1.5] and [Rasmussen and Williams, 2006, chapter 6.3]). However, for our use case in the following sections, the formulation from theorem 11.3 suffices. Note that the weight of the regularization term can be controlled by weighting k or, equivalently, by inverse-weighting the measurement error variance. Another important observation can be deduced from the previous proof. Equation (11.15) gives an explicit formulation of the posterior mean as weighted sum of functions $\mathcal{L}_i k(x, x_i)$. Hence, in the special case of $\mathcal{L}_i = id$, these functions are given as the kernel of the Gaussian process.

11.2 Polyharmonic Kernels

Minimizing equation (11.12) is similar to a regularized least square formulation. The first (fitting) term ensures that the resulting function follows the measurement data, the second (regularization) term characterizes the function's behavior. We now formulate such a regularized fit for the task of surface estimation with specific regularization properties and derive the associated Gaussian process kernel. Let measurements of a function $f : \mathcal{B} \to \mathbb{R}$ be given as

$$f_i^p = f(x_i^p) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, \sigma_p^2), \quad i = 1, \ldots, N_p, \tag{11.17}$$

where $\mathcal{B}$ is any compact subset of interest in $\mathbb{R}^d$, i.e. containing all training points x_i^p and potential test points. When interpreting the function as a surface, a regularization term related to certain measures on its derivatives seems natural. For this purpose consider the class of polyharmonic (or smoothing) splines. They are given as minimizers of

$$\min_{f\in\mathcal{H}} \sum_{i=1}^{N}\left(f_i^p - f(x_i^p)\right)^2 + \lambda \int_{\mathcal{B}\subset\mathbb{R}^d} \|\nabla^m f(x)\|_2^2 \, dx, \tag{11.18}$$

with some weighting factor $\lambda \in \mathbb{R}_{\geq 0}$ that controls the smoothing. By ∇^m we denote the column vector of all m-th order partial derivative operators, e.g. for $m = 2$ and input dimension $d = 3$

$$\nabla^2 = \begin{bmatrix} \frac{\partial^2}{\partial^2 x_1} & \frac{\partial^2}{\partial x_1 \partial x_2} & \frac{\partial^2}{\partial x_1 \partial x_3} & \frac{\partial^2}{\partial x_2 \partial x_1} & \frac{\partial^2}{\partial^2 x_2} & \frac{\partial^2}{\partial x_2 \partial x_3} & \frac{\partial^2}{\partial x_3 \partial x_1} & \frac{\partial^2}{\partial x_3 \partial x_2} & \frac{\partial^2}{\partial^2 x_3} \end{bmatrix}^T. \tag{11.19}$$

The value $m \in \mathbb{N}_{\geq 0}$ is the degree of derivatives considered in the regularization term. For example, a value of $m = 2$ minimizes second derivatives, such that the

function derivatives tend to stay constant, especially outside of the measurements points. This specific polyharmonic spline is referred to as *thin plate spline* and often associated with the physical motivation of minimizing the bending energy of a thin plate. Similarly, for $m = 3$, terms related to third order derivatives are minimized. This can be imagined as a fitted surface that tries to maintain its curvature. A solution to (11.18) for arbitrary Hilbert spaces $\mathcal{H}$ can be computed with theorem 11.2. Using the notation from this theorem, the function space of the classic polyharmonic spline problem can be described. There, $\mathcal{H}_1$ is spanned by the polyharmonic *radial basis functions* (RBF)

$$\xi_i(x) = \begin{cases} \|x - x_i\|_2^{2m-d} \log(\|x - x_i\|_2) & \text{if } 2m-d \text{ is an even integer} \\ \|x - x_i\|_2^{2m-d} & \text{otherwise} \end{cases} \tag{11.20}$$

and $\mathcal{H}_0$ usually consists of all constant and linear polynomial functions.

The regularization term from (11.18) is now used to derive polyharmonic kernels, i.e. kernels that give the posterior mean of a Gaussian process similar smoothing properties to the polyharmonic splines. According to theorem 11.3, it suffices to find a kernel function k, such that $\int_{\mathcal{B}} \|\nabla^m f(x)\|_2^2 \, dx$ defines a norm in the RKHS associated to the kernel. We first introduce the new operator

$$D^\alpha := \frac{\partial^{|\alpha|}}{\partial^{\alpha_1} x_1 \dots \partial^{\alpha_d} x_d}, \quad \alpha = (\alpha_1, \dots, \alpha_d), \quad |\alpha| = \sum_{i=1}^{d} \alpha_i. \tag{11.21}$$

The desired kernel function can then be obtained implicitly:

Lemma 11.1. Let $\mathcal{B} \subset \mathbb{R}^d$ be a compact subset and $k : \mathcal{B} \times \mathcal{B} \to \mathbb{R}$ be a function that satisfies the polyharmonic equation

$$(-1)^m \Delta_x^m k(x, x') = \delta(x - x'), \quad \forall x, x' \in \mathcal{B}, \tag{11.22}$$

with the m-th order Laplace operator Δ^m and the Dirac delta δ. Furthermore, the following requirements on k must hold:

- All m-th order partial derivatives of k are in $L_2(\mathcal{B})$. (11.23)
- $D^\alpha k(\cdot, x')$ vanishes towards $\partial\mathcal{B}$, $\quad \forall |\alpha| < m, \forall x' \in \tilde{\mathcal{B}} \subset \mathcal{B}$. (11.24)

Then k is symmetric, positive definite and defines a RKHS $\mathcal{H}(\tilde{\mathcal{B}})$ with induced norm

$$\|f\|_{\mathcal{H}(\tilde{\mathcal{B}})}^2 = \int_{\mathcal{B}} \|\nabla^m f(x)\|_2^2 \, dx. \tag{11.25}$$

Proof. Let $\mathcal{H}(\tilde{\mathcal{B}})$ be the Hilbert space that contains all functions f, g with all m-th order partial derivatives in $L_2(\mathcal{B})$, such that all partial derivatives of order up to $m-1$ vanish towards $\partial\mathcal{B}$. Then $[D^\alpha f] \cdot [D^\beta g]$ vanishes towards $\partial\mathcal{B}$ for all $|\alpha| + |\beta| = 2m-1$. We can equip $\mathcal{H}(\tilde{\mathcal{B}})$ with the inner product

$$\langle f, g \rangle_{\mathcal{H}(\tilde{\mathcal{B}})} := \int_{\mathcal{B}} (\nabla^m f(x))^T \cdot (\nabla^m g(x)) \, dx, \tag{11.26}$$

which induces the norm given in (11.25). Hence, it suffices to show that k is the reproducing kernel of $\mathcal{H}(\tilde{\mathcal{B}})$. Clearly, the requirements on $k(x,\cdot)$ imply that it must be in $\mathcal{H}(\tilde{\mathcal{B}})$ for all $x \in \tilde{\mathcal{B}}$. We will now show the reproducing property of k to conclude the proof. Equation (11.26) is identical to

$$\int_{\mathcal{B}} \sum_{|\alpha|=m} \left(D^\alpha f(x)\right) \left(D^\alpha g(x)\right) \, dx. \tag{11.27}$$

We can now shift derivatives from f to g using integration by parts d times. Then (11.27) expands to

$$\begin{aligned} &\sum_{|\alpha|=m-1} \int_{\mathcal{B}} \sum_{i=1}^{d} \frac{\partial}{\partial x_i} \Bigg[\overbrace{\left(D^\alpha f(x)\right) \cdot \left(\frac{\partial}{\partial x_i} D^\alpha g(x)\right)}^{h(x)} \Bigg] \, dx \\ - &\sum_{|\alpha|=m-1} \int_{\mathcal{B}} \underbrace{\sum_{i=1}^{d} \left(D^\alpha f(x)\right) \cdot \left(\frac{\partial^2}{\partial^2 x_i} D^\alpha g(x)\right)}_{\left(D^\alpha f(x)\right) \cdot \left(\Delta D^\alpha g(x)\right) \, dx}, \end{aligned} \tag{11.28}$$

with the Laplace operator Δ. Due to the requirements on the Hilbert space, $h(x)$ vanishes towards $\partial\mathcal{B}$. Hence, using the divergence theorem we obtain

$$\int_{\mathcal{B}} \sum_{i=1}^{d} \frac{\partial}{\partial x_i} h(x) \, dx = 0. \tag{11.29}$$

In consequence, the first term in (11.28) is also zero. This variable shifting process can be repeated analogously $m-1$ times, until we obtain the identity

$$\begin{aligned} \langle f, g \rangle_{\mathcal{H}(\tilde{\mathcal{B}})} &= (-1)^m \int_{\mathcal{B}} f(x) \left(\prod_{i=1}^{m} \Delta \right) g(x) \, dx \\ &= (-1)^m \int_{\mathcal{B}} f(x) \Delta^m g(x) \, dx \end{aligned} \tag{11.30}$$

This equality is used to reformulate the reproducing property, i.e.

$$\left.\begin{aligned} \langle f(\cdot), k(x,\cdot) \rangle_{\mathcal{H}(\tilde{\mathcal{B}})} &= \int_{\mathcal{B}} (\nabla_y^m f(y))^T \cdot (\nabla_y^m k(x,y)) \, dy \\ &\overset{(11.30)}{=} (-1)^m \int_{\mathcal{B}} f(y) \Delta_y^m k(x,y) \, dy \\ &\overset{!}{=} f(x) \end{aligned}\right\}, \forall f \in \mathcal{H}(\tilde{\mathcal{B}}), \forall x \in \tilde{\mathcal{B}}. \tag{11.31}$$

The last identity is true if and only if

$$(-1)^m \Delta_y^m k(x,y) = \delta(x-y), \quad \forall x \in \tilde{\mathcal{B}}, \forall y \in \mathcal{B}. \tag{11.32}$$

Due to the reproducing property, k is especially also positive definite. Note that this proof works identically for $\mathcal{B} = \tilde{\mathcal{B}} = \mathbb{R}^d \cup \{\infty\}$. However, as we will see later, then no solution to (11.22) exists that satisfies (11.24). ■

Solutions to the differential equation (11.22) can be identified as Greens functions, which have been extensively studied in literature. Explicit solutions for this differential equation are given in [Wahba, 1990, chapter 2.4] as

$$\begin{aligned} k(x,x') &= G(\|x-x'\|_2) \\ \text{with } G(\tau) &= \begin{cases} \theta_{m,d}\tau^{2m-d}\log\tau & \text{if } 2m-d \text{ is an even integer} \\ \theta_{m,d}\tau^{2m-d} & \text{otherwise,} \end{cases} \end{aligned} \tag{11.33}$$

with weights $\theta_{m,d}$ defined by

$$\theta_{m,d} = \begin{cases} \dfrac{(-1)^{d/2+1+m}}{2^{2m-1}\pi^{d/2}(m-1)!(m-d/2)!} & \text{if } 2m-d \text{ is an even integer} \\ \dfrac{\Gamma(d/2-m)}{2^{2m}\pi^{d/2}(m-1)!} & \text{otherwise.} \end{cases} \tag{11.34}$$

Some explicit values of these weights are given in table 11.1.

Table 11.1: Values of $\theta_{m,d}$ for various m and d.

d \ m	**0**	**1**	**2**	**3**	**4**
1	0	$-\frac{1}{2}$	$\frac{1}{12}$	$-\frac{1}{240}$	$\frac{1}{10080}$
2	0	$-\frac{1}{2\pi}$	$\frac{1}{8\pi}$	$-\frac{1}{128\pi}$	$\frac{1}{4608\pi}$
3	0	$\frac{1}{4\pi}$	$-\frac{1}{8\pi}$	$\frac{1}{96\pi}$	$-\frac{1}{2880\pi}$
4	0	0	$-\frac{1}{8\pi^2}$	$\frac{1}{64\pi^2}$	$-\frac{1}{1536\pi^2}$

The functions $G(\tau)$ are again the polyharmonic RBFs (see (11.20)). For (11.33) to be a valid kernel, all additional requirements from lemma 11.1 must be met. However, this is currently not possible, as especially $G(\tau)$ does not vanish for points other then $\tau = 0$, i.e. (11.24) does not hold and k is not positive definite. Hence, the function

$$\begin{aligned} \tilde{k}(x,x') &= \begin{cases} \tilde{G}(\|x-x'\|_2) & x,x' \in \tilde{\mathcal{B}} \\ 0 & x,x' \in \mathcal{B}\setminus\tilde{\mathcal{B}} \end{cases} \\ \text{with } \tilde{G}(\tau) &= G(\tau) + \sum_{i=0}^{m-1} a_i\tau^{2i}, \quad a_0,\dots a_{m-1} \in \mathbb{R} \end{aligned} \tag{11.35}$$

is considered instead. This new formulation is also an isotropic solution to the polyharmonic equation (11.22), since $\Delta^m\|x-x'\|_2^i \equiv 0$ if and only if i is even and $i < 2m$. Now we see why $\tilde{\mathcal{B}}$ can not be $\mathbb{R}^d$. In that case

$$\lim_{\tau\to\infty} \tilde{G}(\tau) \to \infty, \tag{11.36}$$

such that $\tilde{k}$ could not vanish at infinity (requirement (11.24)) and would therefore not be a reproducing kernel nor positive definite.

For lemma 11.1 to apply, it is additionally required that partial derivatives of order m must be in $L_2(\mathcal{B})$ (11.23). This implies that all partial derivatives of order up to $m-1$ of $\tilde{k}$ must be continuous, i.e. especially the transition in the case distinction in (11.35) must be smooth. Mixed partial derivatives of order i of $\tilde{k}$ contain terms

$$\frac{\partial^j}{\partial^j \tau}\tilde{G}(\tau), \quad j = 0, \dots, i, \tag{11.37}$$

which must all vanish towards ∂B. Note that $\tilde{G}$ is a radial function. Hence, we can pick a fixed radius $R \in \mathbb{R}_{>0}$ and impose the constraints

$$\frac{\partial^i}{\partial^i \tau}\tilde{G}(R) \stackrel{!}{=} 0, \quad \forall i \in \{0, \dots, m-1\} \tag{11.38}$$

to achieve the desired smoothness at $\|x - x'\|_2 = R$. These m constraints then uniquely determine $a_0, \dots, a_{m-1}$. The value R is chosen as the maximum distance of two points in the set $\tilde{\mathcal{B}}$, which contains all relevant test and training points[14]. The minimal set of $\mathcal{B}$ then includes all points with a maximum distance of R to $\tilde{\mathcal{B}}$. Finally, one can easily verify that all partial derivatives of order m of $k(\cdot, x')$ are in $L_2(\mathbb{R}^d)$ if and only if $2m - d > 0$ to avoid a singularity at $x = x'$, i.e. they satisfy (11.23). Now the polyharmonic kernel functions can uniquely be computed. These are referred to as

$$k_{d,m}(x, x') = G_{d,m}(\|x - x'\|_2), \quad \forall x, x' \in \tilde{\mathcal{B}}. \tag{11.39}$$

Some explicit examples are given in table 11.2. Note that all of them are smooth functions for all $\|x - x'\|_2 \in [0, R]$. This especially implies mean square continuity of the corresponding Gaussian processes (theorem 5.2), and the smoothness of posterior mean (5.10) and covariance function (5.11). Furthermore, as stated in the previous section, the posterior mean is a weighted sum of its kernel functions in this case. For odd $2m - d$, for example, it is piecewise polynomial.

Table 11.2: Polyharmonic kernel functions (see equation (11.39)).

d	**m**	$\mathbf{G_{d,m}}(\tau)$
1	2	$\frac{1}{24}\left(2\tau^3 - 3R\tau^2 + R^3\right)$
1	3	$\frac{1}{1920}\left(-8\tau^5 + 15R\tau^4 - 10R^3\tau^2 + 3R^5\right)$
2	2	$\frac{1}{16\pi}\left(2\tau^2 \log \tau - (1 + 2\log R)\tau^2 + R^2\right)$
2	3	$\frac{1}{512\pi}\left(-4\tau^4 \log \tau + (3 + 4\log R)\tau^4 - 4R^2\tau^2 + R^4\right)$
3	2	$\frac{1}{16\pi}\left(\frac{1}{R}\tau^2 - 2\tau + R\right)$
3	3	$\frac{1}{768\pi}\left(-\frac{3}{R}\tau^4 + 8\tau^3 - 6R\tau^2 + R^3\right)$

At this point we want to highlight some differences to the polyharmonic kernels introduced in [Williams and Fitzgibbon, 2007]. Note that for $(d, m) \in \{(1, 2), (2, 2)\}$

[14] The shape of the largest possible set in two dimensions that meets this requirement is a Reuleaux triangle.

the kernel functions are scaled differently. Strictly speaking, their version only satisfy a scaled version of the polyharmonic equation (11.22). Ultimately this only affects the weight λ of the regularization term in (11.18). Furthermore, their kernel function for $(d, m) = (3, 2)$ actually satisfies a scaled version of a polyharmonic equation with $m = 3$. Additionally, this kernel does not satisfy (11.38). It has been pointed out in literature (e.g. [Martens et al., 2017]) that they used a wrong sign in their formulation[15]. Even with the correct sign, equation (11.38) is only satisfied up to the first derivative. Hence, (11.29) does not hold, such that the kernel relates to a different regularization term that additionally penalizes other mixed partial derivative terms.

The polyharmonic kernel (11.39) shall now be used for Gaussian process regression. Theorem 11.3 states that the measurement error σ_p^2 (see equation (11.17)) is a weighting factor between fitting and regularization term, i.e. it has the same task as λ in (11.18). Hence, besides its statistical interpretation as measurement error, it is used to control smoothing. An example of a one-dimensional Gaussian process posterior using a polyharmonic kernel is given in figure 11.1. Note that the slope of the function remains almost constant for test points lower in the intervals $[-2, -1]$ and $[1, 2]$. Furthermore, the curve looks stiff (i.e. bending is kept to a minimum), what is to be expected when minimizing second derivatives. This behavior can be compared to a Gaussian process fit with exponential kernel in figure 11.2. There, the posterior rapidly goes to the mean value of zero on the sides.

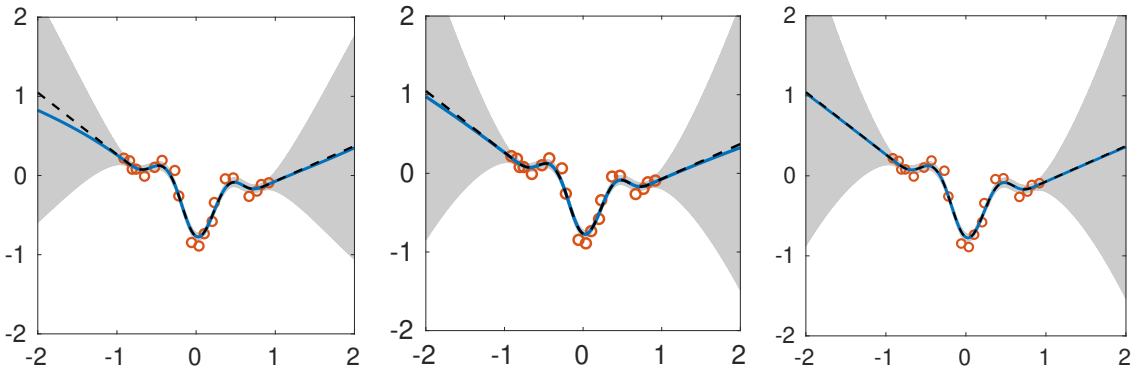


Figure 11.1: Gaussian process posterior distribution for 20 given training points using the $k_{1,2}$ kernel with $\sigma_p^2 = 0.001$. The gray area corresponds to their 99 % confidence intervals. From left to right: $R = 4, R = 20, R = 100$. The dotted black line is the solution to the polyharmonic spline regression fitting problem (11.18).

We will now highlight the key differences to polyharmonic spline regression using the example given in figure 11.1. While both regression methods are almost identical close to the training points, their values differ towards the sides. This effect is most prominent for small R and is explained in the following. As mentioned in section 11.1, a Gaussian processes posterior mean can be written as a weighted sum of kernel functions. By restricting ourselves to the domain $\mathcal{B}$, equation (11.38) implies that the posterior mean and its derivatives of order up to $m - 1$ are zero for all points with a distance of R to their closest training point. Moreover, since $G_{1,2}$ is a polynomial of degree 3, this effect becomes more observable with increasing distance to measurement points. This behavior can be explained in a different way by looking

[15] Equation (11b) in [Williams and Fitzgibbon, 2007] should therefore be $c(r) = 2|r|^3 - 3Rr^2 + R^3$.

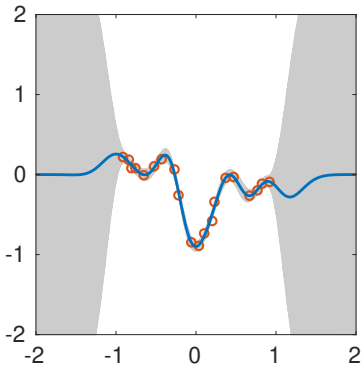
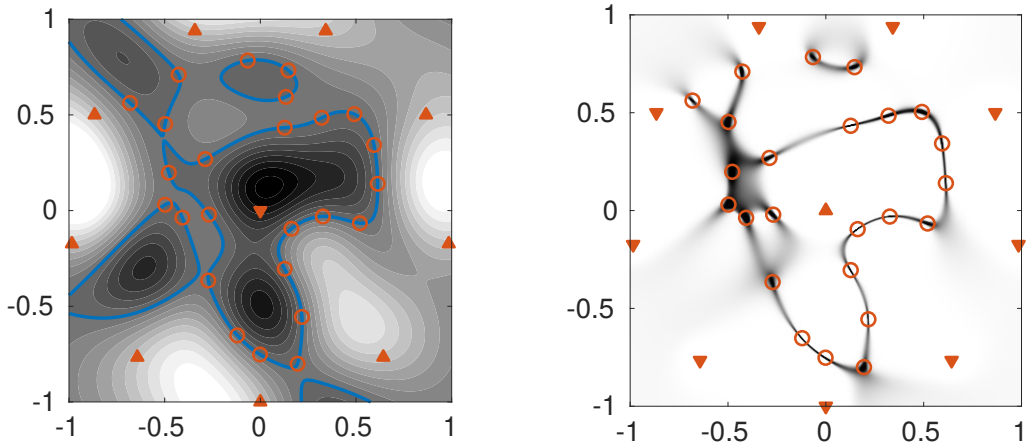


Figure 11.2: Exponential kernel $k(x, x') = e^{-10\|x-x'\|_2^2}$ one-dimensional Gaussian process regression example with $\sigma_p^2 = 0.001$. The 20 training points are identical to those given in figure 11.1. The gray area corresponds to 99 % confidence intervals.

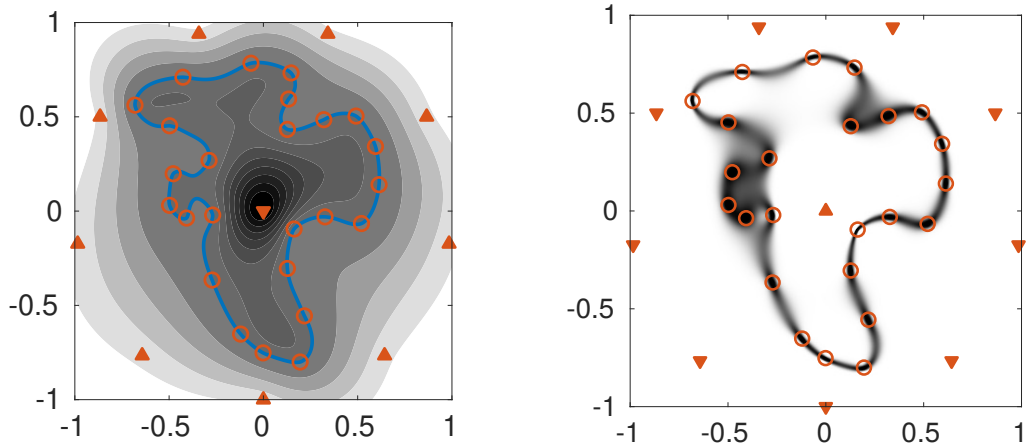
at the RKHS $\mathcal{H}$ of each kernel function. The corresponding induced norm (11.25) does not penalize polynomials in d variables of total degree less than m. Therefore, they are not included in $\mathcal{H}$ and not considered for the Gaussian process regression fit. However, using the representer theorem 11.2, a minimizer to the functional (11.18) can be calculated, which includes functions from the null space of the regularization term. Thus, polyharmonic spline regression can minimizes over a larger function space, that especially considers these polynomial functions. In the example of figure 11.1, the difference between the function spaces consist of all constant and linear polynomials. This results in straighter extrapolation behavior for points with a large distance to training points.

For our use case, the absence of polynomial function of order less then m does not pose a problem. From the representer theorem 11.2 we realize that these additional basis functions of the null space do not depend on the measurement positions. Therefore, they only describe the global behavior of the regression fit. Since we are only interested in the implicit surface described by the zero level set of the Gaussian process posterior mean, it is desirable that the weights for those terms are zero. Without any training points, this posterior mean is constant zero. Inserting measurements deforms this plane, while locally minimizing m-th order partial derivatives. Geometrically speaking, the training points are used to "draw" contour lines onto the plane.

For the application in object reconstruction, measurements usually correspond to surface points, i.e. observations of the zero level set. However, these training points do not suffice, since this would again result in a constant zero posterior mean. Additional interior and exterior measurement points are used to obtain the desired implicit surface. In this context we want to highlight the importance of the constructed polyharmonic kernel and its superiority compared to other kernels when constructing implicit functions in figure 11.3. Another two-dimensional GPIS example is given in figure 11.4, while a three-dimensional example is given in figure 11.5. In the latter we see the effect of different choices of m. In three dimensions, the regularization term (11.25) with $m = 2$ only considers two partial derivatives in each term, leading to bumps in the surface. For $m = 3$ the curvature tends to stay constant, resulting in much smoother surfaces. From figure 11.5b we also realize the importance of carefully placed interior and exterior points, since these may lead to unintended deformations. Overcoming this issue is the topic of the following section.



(a) Kernel function $k(x, x') = e^{-10\|x-x'\|_2^2}$.



(b) Kernel function $k(x, x') = k_{22}(x, x')$, with $R = 3$.

Figure 11.3: Comparison of implicit surfaces obtained from the zero level set of the posterior mean of different Gaussian processes in two dimensions. The training points are given in red, indicating function values of ▲ $= 1$, ▼ $= -1$ and ○ $= 0$. The measurement error is set to $\sigma_p^2 = 1 \times 10^{-6}$. Left: Contour line of Gaussian process posterior mean. The blue line indicates the zero level set. Right: Probability of posterior to be close to 0, i.e. inside $[-\epsilon, \epsilon]$, visualized with shades of gray.

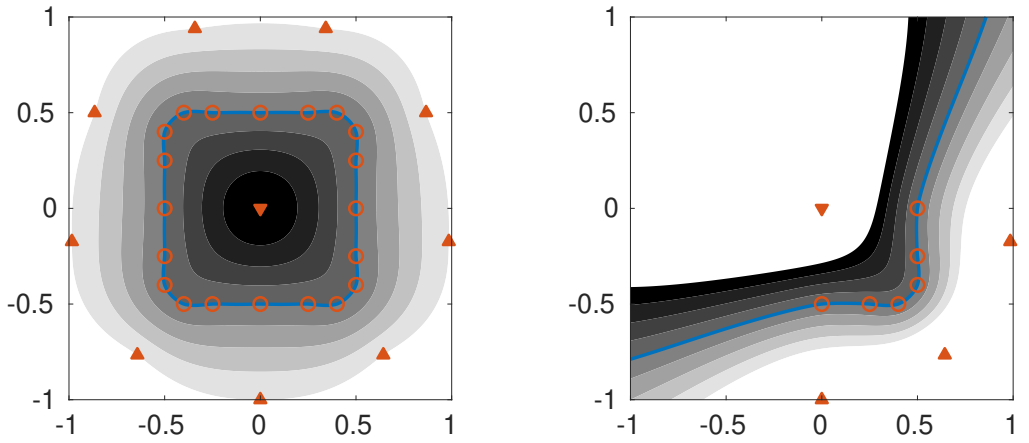
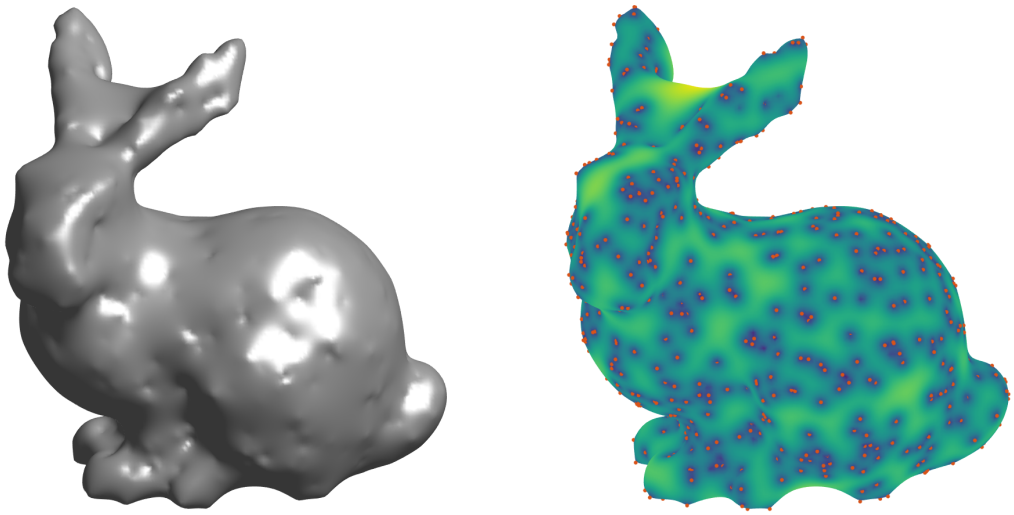
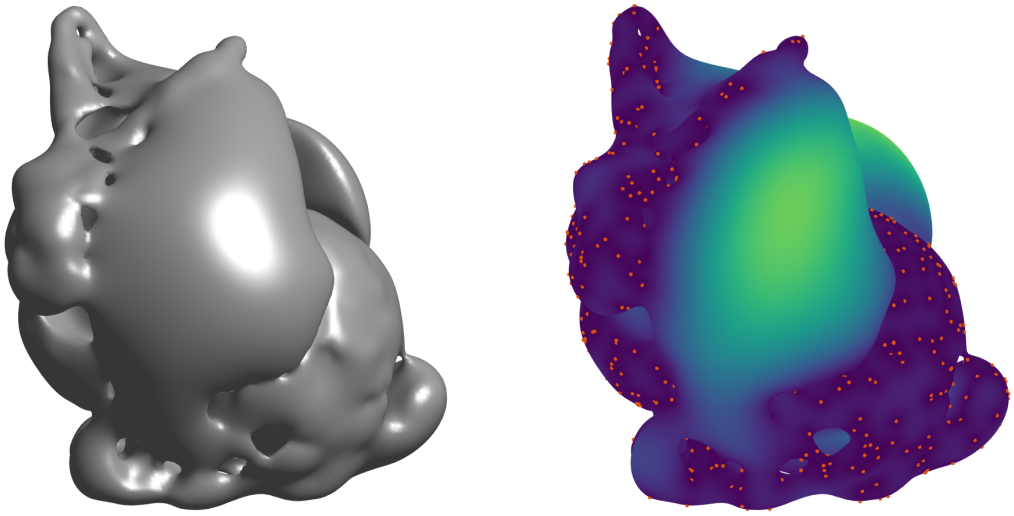


Figure 11.4: Further two-dimensional GPIS examples. The notation and utilized parameters are identical to the one in figure 11.3b (left).



(a) Kernel function $k(x, x') = k_{32}(x, x')$.



(b) Kernel function $k(x, x') = k_{33}(x, x')$.

Figure 11.5: Visualization of a 3D GPIS of the Stanford bunny trained on 800 surface (= 0, red points), 1 interior (= −1, at center) and 42 exterior (= 1, grid points of geodesic dome) points. Left: Shaded zero level set of the posterior mean. Right: Zero level set colored according to the posterior variance.

11.3 Adding derivative observations

As we have seen in the previous section, interior and exterior points have to be placed carefully when modeling implicit surfaces from point observations. If their position is chosen insufficiently, the implicit surface may not represent the desired geometry (e.g. figure 11.5b). The problem is that the model shape is rather complex and due to the relatively small number of interior and exterior points the GPIS has a high uncertainty in classifying space in between. A similar effect is observable in figure 11.3b. There, the probability of points being on the surface does not define a sharp curve in some areas, i.e. the Gaussian process is uncertain about the implicit surface location. A slightly different choice of interior or exterior points could therefore change the shape. Even with apparently simple shapes undesirable effects occur. While the contour lines in figure 11.4 follow the measurements, the placement of interior and exterior points affects how strongly the corners of the implicit surfaces are rounded. Furthermore, in the right plot we observe how their choice alters the extrapolation properties, i.e. by bending the curve outwards.

In the context of surface modeling, a common approach is to automatically place a surface point with additional slightly offset interior and exterior points along each measurement ray. However, even there corner-cases exist where problems can occur during surface reconstruction and we have to maintain three times the measurement points. In order to overcome these issues we will eliminate the need for off-surface training points in this section. Instead, surface normal measurements are included into the Gaussian process. The idea is that a single point together with its surface normal sufficiently defines the desired surface locally. Ultimately, a much more accurate model representation is achieved with fewer measurement locations. Integrating gradient measurements into a Gaussian process is not new. It is already successfully employed for a variety of tasks, such as nonlinear systems modelling [Solak et al., 2003], Bayesian optimization and quadrature [Wu et al., 2017] or GPIS with exponential kernels [Dragiev et al., 2011]. In practical application, these surface normal measurements are usually easily obtained. Our main use case is a sensor that measures a number of surface points simultaneously. Then the surface normal measurements can be computed from a set of nearest neighbors.

We will now show how these measurements can be integrated into the GPIS. For that purpose, first consider a general zero mean Gaussian process

$$f \sim \mathcal{GP}(0, k), \tag{11.40}$$

with a positive definite kernel function

$$k : U \times U \to \mathbb{R}, \quad (x, x') \mapsto k(x, x'), \quad \text{with } U \subset \mathbb{R}^d. \tag{11.41}$$

We are now interested in the joint distribution of f and ∇f. Using theorem 5.1 with the linear operator $\mathcal{L} = \begin{bmatrix} 1 & \nabla^T \end{bmatrix}^T$, this is obtained as

$$\mathcal{L}f = \begin{bmatrix} f \\ \nabla f \end{bmatrix} \sim \mathcal{GP}\left(0, \begin{bmatrix} k(x,x') & \nabla_{x'} k(x,x') \\ \nabla_x k(x,x') & \nabla_x \nabla_{x'} k(x,x') \end{bmatrix}\right). \tag{11.42}$$

Note that it is required that $\nabla_x \nabla_{x'} k(x, x')$ exists and is well defined for all $x, x' \in U$.

Similar to section 5.1, sets of point and gradient measurements

$$\begin{aligned} X^p &:= \left\{x_i^p \in \mathbb{R}^d\right\}_{i=1}^{N_p}, \quad \text{training points (point measurements)} \\ X^g &:= \left\{x_i^g \in \mathbb{R}^d\right\}_{i=1}^{N_g}, \quad \text{training points (gradient measurements)} \\ f^p &:= f(X^p) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_p^2 \cdot I_{N_p}) \\ f^g &:= \nabla f(X^g) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_g^2 \cdot I_{N_g \cdot d}), \end{aligned} \tag{11.43}$$

can be defined. Given these measurements, we are now interested in evaluating a trained Gaussian process at

$$X^* := \left\{x_i^* \in \mathbb{R}^d\right\}_{i=1}^{M}, \quad \text{test points (point evaluations)} \tag{11.44}$$

to obtain an estimate for

$$f^* := f(X^*) = \left[f(x_1^*), \ldots, f(x_M^*)\right]^T. \tag{11.45}$$

The joint distribution of test and training data is given as

$$\begin{bmatrix} f^p \\ f^g \\ f^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K & k(X^p, X^*) \\ k(X^*, X^p) & \nabla_{x'} k(X^*, X^g) \quad k(X^*, X^*) \end{bmatrix}\right), \tag{11.46}$$

with

$$K = \begin{bmatrix} k(X^p, X^p) & \nabla_{x'} k(X^p, X^g) \\ \nabla_x k(X^g, X^p) & \nabla_x \nabla_{x'} k(X^g, X^g) \end{bmatrix} + \begin{bmatrix} \sigma_p^2 \cdot I_{N_p} & 0 \\ 0 & \sigma_g^2 \cdot I_{N_g \cdot d} I \end{bmatrix}. \tag{11.47}$$

Then the Gaussian process posterior is obtained analogous to section 5.1 by calculating the conditional distribution

$$\begin{aligned} &f^* | X^p, f^p, X^g, f^g, X^* \sim \mathcal{N}\left(m_{\text{post}}(X^*), k_{\text{post}}(X^*)\right), \quad \text{with} \\ &m_{\text{post}}(X^*) = \begin{bmatrix} k(X^p, X^*) \\ \nabla_x k(X^g, X^*) \end{bmatrix}^T K^{-1} \begin{bmatrix} f^p \\ f^g \end{bmatrix} \\ &k_{\text{post}}(X^*, X^*) = k(X^*, X^*) - \begin{bmatrix} k(X^p, X^*) \\ \nabla_x k(X^g, X^*) \end{bmatrix}^T K^{-1} \begin{bmatrix} k(X^p, X^*) \\ \nabla_x k(X^g, X^*) \end{bmatrix} \end{aligned} \tag{11.48}$$

that defines a new Gaussian process

$$f^* \sim \mathcal{GP}(m_{\text{post}}, k_{\text{post}}). \tag{11.49}$$

Note that this derivation also works analogously for other measurement quantities that can be described using a linear operator. First, the joint Gaussian process of all training and test quantities must be determined (equation (11.42)). Subsequently, the joint distribution of all measurement and evaluation data can be calculated (equation (11.46)). The posterior process is then obtained from a conditional distribution (equation (11.48)).

Using polyharmonic kernels, the Gaussian process prior is given by

$$f_{d,m}^g \sim \mathcal{GP}\left(0, k_{d,m}^g\right), \tag{11.50}$$

with the kernel function

$$k^g_{d,m}(x,x') = \begin{bmatrix} k_{d,m}(x,x') & \nabla_{x'} k_{d,m}(x,x') \\ \nabla_x k_{d,m}(x,x') & \nabla_x \nabla_{x'} k_{d,m}(x,x') \end{bmatrix}. \tag{11.51}$$

For our use case, we simultaneously obtain surface normals for each surface point measurement, i.e. $X^p = X^g$. With this simplification, the posterior (11.48) is obtained equivalently by first constructing the joint Gaussian process posterior of both point and gradient quantities (see equations (5.10), (5.11)), and then applying the linear operator $\begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}$ to it. In order for $f^g_{d,m}$ to be a valid Gaussian process, $k^g_{d,m}(x,x')$ must be well defined for all x, x' with $\|x - x'\|_2 \leq R$. This is now further investigated for some values of d and m. The required first and second order partial derivatives of $k_{d,m}$ are given in table 11.3 and 11.4. Note that $\frac{\partial}{\partial x_i} k_{d,m}(x,x') = -\frac{\partial}{\partial x'_i} k_{d,m}(x,x')$ for all $i \in \{1,\dots,d\}$[16].

Table 11.3: First order partial derivatives of polyharmonic kernel functions.

d	**m**	$\frac{\partial}{\partial \mathbf{x_i}} \mathbf{k_{d,m}}(\mathbf{x},\mathbf{x}'), \quad \tau = \|x - x'\|_2$	$\frac{\partial}{\partial \mathbf{x_i}} \mathbf{k_{d,m}}(\mathbf{x},\mathbf{x}')$ **at** $\mathbf{x} \to \mathbf{x}'$
1	2	$\frac{1}{4}(x_i - x'_i)\,(\tau - R)$	0
1	3	$\frac{1}{96}(x_i - x'_i)\,(-2\tau^3 + 3R\tau^2 - R^3)$	0
2	2	$\frac{1}{4\pi}(x_i - x'_i)\,(\log\tau - \log R)$	0
2	3	$\frac{1}{64\pi}(x_i - x'_i)\,(-2\tau^2 \log\tau + (1 + 2\log R)\tau^2 - R^2)$	0
3	2	$\frac{1}{8\pi}(x_i - x'_i)\left(\frac{1}{R} - \frac{1}{\tau}\right)$	$undefined$
3	3	$\frac{1}{64\pi}(x_i - x'_i)\left(-\frac{1}{R}\tau^2 + 2\tau - R\right)$	0

We observe that $\frac{\partial^2}{\partial x_i \partial x'_j} k_{d,m}(x,x')$ is undefined at $x = x'$ for $(d,m) \in \{(2,2),(3,2)\}$. The first and second order partial derivatives of all other (d,m) pairs are continuous on $\|x - x'\|_2 \in [0,R]$. Hence, theorem 5.2 ensures the mean-square continuity of the associated Gaussian processes. Equation (11.48) additionally implies that the posterior mean and kernel functions are continuous as well and in thus also the curves implicitly defined by the posterior mean zero level set.

For the purpose of surface modeling, the inter- and extrapolation behavior of the Gaussian process posterior mean is important. A natural choice of kernel functions is $m = 3$. This kernel minimizes third order partial derivatives, i.e. the bending of the surface between observations, and therefore strives to keep curvature constant. Consider the case of observing solely a straight wall. Then this GPIS describe a plane. Areas with varying surface normals are interpolated with a uniform curvature, resulting in smooth shapes. However, as seen in figure 11.5, this choice of m does not yield the desired results if solely point measurements are considered. Additional gradient observations allow for a regression fit that has knowledge about an estimate of a function's local orientation. In the context of GPIS the local orientation of the implicit surface is described. A three-dimensional example is given in figure 11.6.

[16] This actually holds for all stationary kernel functions.

Table 11.4: Second order partial derivatives of polyharmonic kernel functions. Here, δ_{ij} denotes the Kronecker delta.

$\mathbf{d}$	$\mathbf{m}$	$\frac{\partial^2}{\partial \mathbf{x_i} \partial \mathbf{x'_j}} \mathbf{k_{d,m}}(\mathbf{x}, \mathbf{x}'), \quad \tau = \|x - x'\|_2$	$\frac{\partial^2}{\partial \mathbf{x_i} \partial \mathbf{x'_j}} \mathbf{k_{d,m}}(\mathbf{x}, \mathbf{x}')$ **at** $\mathbf{x} \to \mathbf{x}'$
1	2	$-\frac{1}{4}(2\tau - R)$	$\frac{R}{4}$
1	3	$-\frac{1}{96}(-8\tau^3 + 9R\tau^2 - R^3)$	$\frac{R^3}{96}$
2	2	$-\frac{1}{4\pi}\left[\frac{(x_i - x'_i)(x_j - x'_j)}{\tau^2} + \delta_{ij}(\log\tau - \log R)\right]$	$undefined$
2	3	$-\frac{1}{16\pi}\left[(x_i - x'_i)(x_j - x'_j)(\log R - \log\tau) + \delta_{ij}\frac{1}{4}(-2\tau^2 \log r + (1 + 2\log R)\tau^2 - R^2)\right]$	$\delta_{ij}\frac{R^2}{64\pi}$
3	2	$-\frac{1}{8\pi}\left[\frac{(x_i - x'_i)(x_j - x'_j)}{\tau^3} + \delta_{ij}\left(\frac{1}{R} - \frac{1}{\tau}\right)\right]$	$undefined$
3	3	$-\frac{1}{32\pi}\left[(x_i - x'_i)(x_j - x'_j)\left(\frac{1}{\tau} - \frac{1}{R}\right) + \delta_{ij}\frac{1}{2}\left(-\frac{1}{R}\tau^2 + 2\tau - R\right)\right]$	$\delta_{ij}\frac{R}{64\pi}$

Although there are far fewer measurement positions than in the figure 11.5, the GPIS approximates the geometry of the Stanford bunny much better without the need for interior or exterior points. Similarly, a two-dimensional GPIS including normal measurements is given in figure 11.7. Compared to figure 11.3b, the contour lines now follow the zero level set more tightly and the certainty in this curve is much higher. This behavior is similar in two further examples in figure 11.8a. Due to normals describing the local orientation of the surface, especially sharper edges can be achieved compared to figure 11.4.

As previously stated, the gradient measurements are usually obtained from a set of nearest neighbor surface points. Depending on the pairwise distance between points and the geometric fidelity of the true model, these measurement normals may be subject to a high error. However, this does not pose a problem, since in this case a larger value σ_g can be chosen. In fact, we experienced that this value can be picked much larger then the point measurement variance σ_p. The obtained surfaces are then very robust to surface normal measurement errors, as illustrated on a two-dimensional example in figure 11.8b. The measurement normals only need to indicate an approximate direction from interior to exterior.

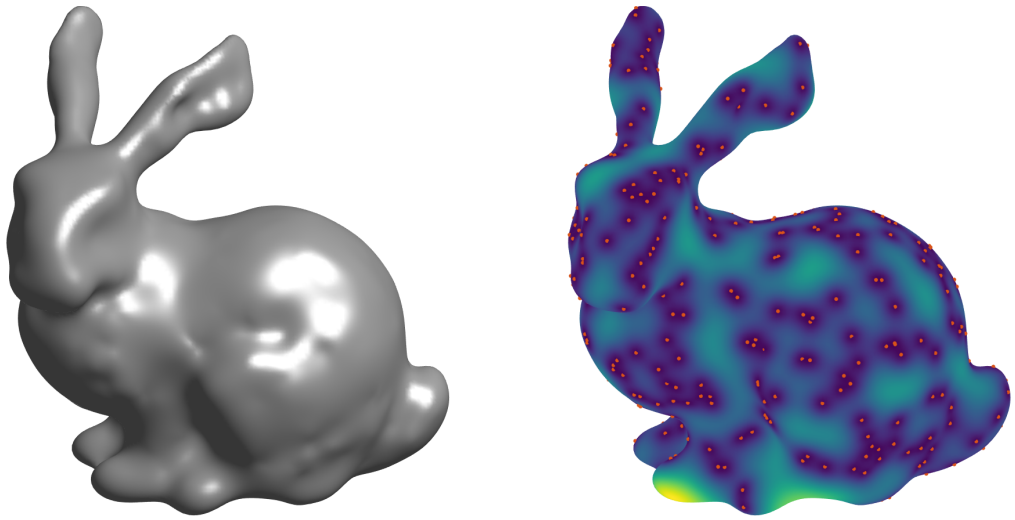


Figure 11.6: GPIS representation of the Stanford bunny using the $k^g_{3,3}$ kernel. Here a set of 400 training points is used, which form a subset of the surface points in figure 11.5. No interior or exterior points are considered, however, each training point corresponds to both surface and normal measurement.

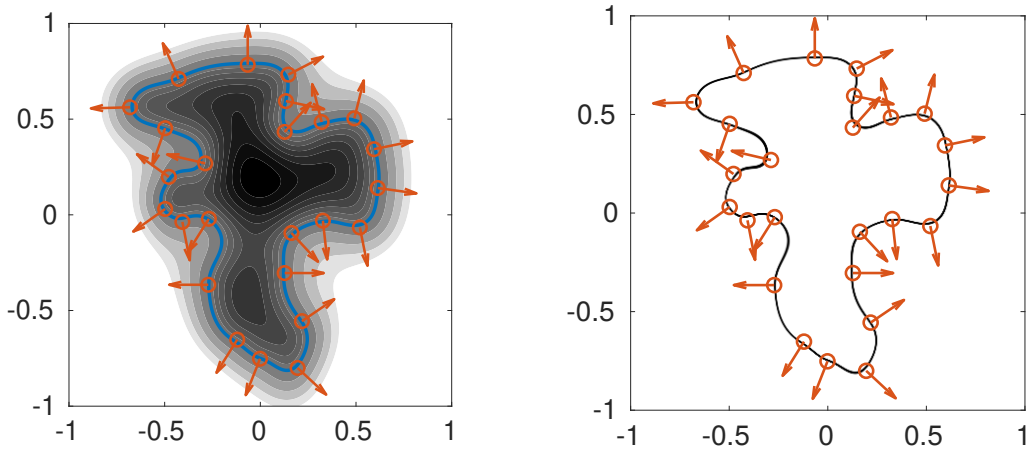
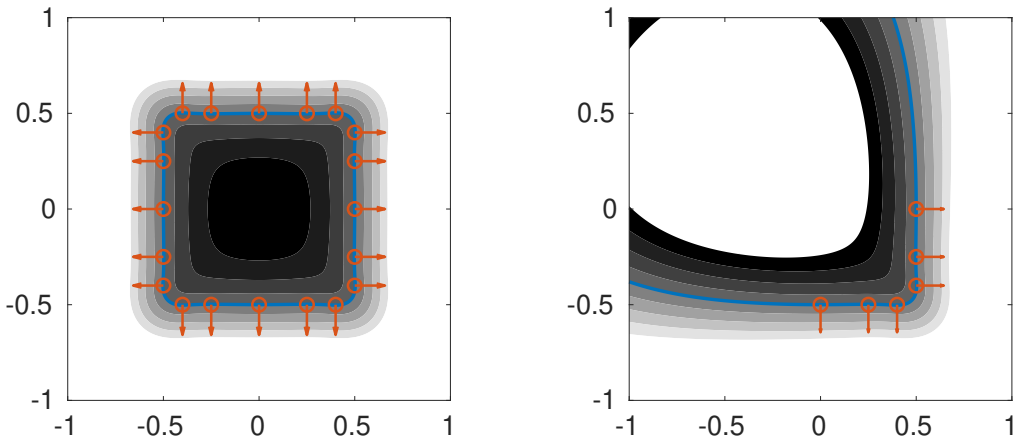
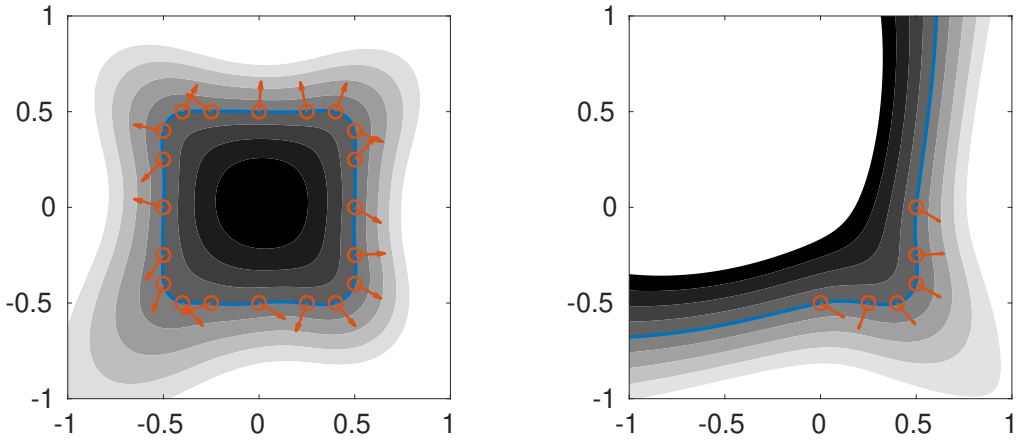


Figure 11.7: Two-dimensional GPIS example with point and surface normal measurements. The surface training point positions are identical to figure 11.3. The kernel function is $k^g_{2,3}$ with $R = 3$. Red dots and arrows indicate point and surface normal training data. The measurement error is set to $\sigma_p^2 = \sigma_g^2 = 1 \times 10^{-6}$. Left: Contour line of Gaussian process posterior. The blue line indicates the zero level set. Right: Probability of posterior to be close to 0, i.e. inside $[-\epsilon, \epsilon]$ visualized with shades of gray.



(a) Exact normal measurements with $\sigma_g^2 = 1 \times 10^{-6}$.



(b) Disturbed normal measurements with $\sigma_g^2 = 1 \times 10^{-2}$.

Figure 11.8: Comparison of disturbed and exact normal measurements in two 2D GPIS scenarios for a cube and an edge. The surface training point positions are identical to figure 11.4. The Gaussian process uses the $k_{2,3}^g$ kernel with $R = 3$. The point measurement error is set to $\sigma_p^2 = 1 \times 10^{-6}$.

GPIS Next-Best-View Planning

12

The GPIS surface estimation explained in the previous chapter shall now be used in the context of NBV planning. This representation provides information about the quality of the surface reconstruction using the posterior covariance function. A NBV can then be picked as a sensor pose that maximizes the information gained about the surface geometry. Since we are dealing with depth sensors, the task of assessing the quality of an arbitrary sensor location can be divided into two components. First, the set of points that describe intersections between measurement rays and the GPIS need to be computed, which we denote as $\mathcal{I}$. These are subsequently used to evaluate the local surface quality in a second step. Using a Gaussian process, the latter task can be related to the covariance function of the posterior. In [Hollinger et al., 2012], two metrics are defined that relate to the certainty of the entire implicit surface. They pick a sensor viewpoint that minimizes either of

$$\begin{aligned} J_{\text{avg}} &= \int_{\mathcal{S}} \tilde{k}_{\text{post}}(x,x)\ dx \\ J_{\text{max}} &= \max_{x \in \mathcal{S}} \tilde{k}_{\text{post}}(x,x) \end{aligned} \qquad \text{with } \mathcal{S} = \{x \in \tilde{\mathcal{B}} \mid \tilde{m}_{\text{post}}(x) = 0\}, \tag{12.1}$$

i.e. the average or maximum surface variance, where k_{post} and m_{post} define the current GPIS (see (11.48)), while $\tilde{m}_{\text{post}}$ and $\tilde{k}_{\text{post}}$ describe the GPIS after inserting additional expected surface measurements at training positions $\mathcal{I}$. The formulations in (12.1) consider the entire implicit surface. While it is desirable to also include uninstrumented surface points in this metric, its computational complexity grows with the size of the surface. This makes an application for large scale reconstructions unfeasible, especially in the non-model-based online case. Hence, we propose a gain formulation similar to that in section 6.3, i.e. the estimated change in entropy in the surface intersection points $\mathcal{I}$ (see equation (6.35)). Such a metric could be given analogous to the clamped gain formulation of section 6.3.2 as

$$J_{\text{H}} = \sum_{x \in \mathcal{I}} \log \left(\frac{\max\left\{\lambda_{\min}, \tilde{k}(x,x)\right\}}{\max\left\{\lambda_{\min}, k(x,x)\right\}} \right), \tag{12.2}$$

where $\lambda_{\min} \in \mathbb{R}_{>0}$ is again a clamping parameter that describes a desired quality threshold that is to be achieved. A NBV can then by calculated as a sensor pose maximizing J_{H}. Formulation (12.2) is now independent of the size of the implicit surface. Given a set of points $\mathcal{I}$, its computational complexity solely depends on the computational complexity of the posterior kernel functions, since the maximum cardinality of $\mathcal{I}$ is given by the specifications of the utilized sensor.

The main task now is to efficiently calculate the set $\mathcal{I}$ for arbitrary sensor poses, which is the topic of the remainder of this chapter. The surface is represented by an

implicit function, which generally does not allow for an explicit formulation. Therefore, the posterior mean must be evaluated multiple times to obtain its zero level set. The complexity of each call to this function grows non-linearly with the number of training points (see equation (11.48)). Since this number of training points easily exceeds several million in practical application, it becomes completely infeasible to evaluate the posterior mean. Hence, in section 12.1, we first introduce a local version of GPIS with an almost constant complexity. Afterward, two approaches for the computation of GPIS-ray intersections are presented, whose runtimes are also almost independent of the number of training points. Note, however, that these are only outlined and not fully analyzed due to time limitations and are still subject to further research. In the first approach in section 12.2, a complete surface mesh is generated such that these points can be extracted from the depth-buffer of a GPU. This allows for a very fast computation of intersection points, however, requires the generation of the full surface mesh first, which entails a computational overhead in each NBV iteration. There, a compromise must be made between mesh approximation accuracy of the implicit surface and runtime. Using a tetrahedral structure allows for iterative surface mesh improvements and changes when inserting new measurements. A second, completely mesh-less approach, is given in section 12.3. GPIS-ray intersections are iteratively computed using a ray-marching method. This eliminates the overhead necessary for maintaining a surface mesh representation, but the computation of each intersection point is more expensive.

12.1 Local GPIS

Consider the posterior GPIS distribution (11.49). Note that given a set of training points, some quantities of (11.48) can be precomputed to allow for a faster evaluation of posterior mean and kernel. This precomputation step will be called *training* and includes the construction and inversion of K, as well as the multiplication

$$K^{-1} \begin{bmatrix} f^p \\ f^g \end{bmatrix}. \tag{12.3}$$

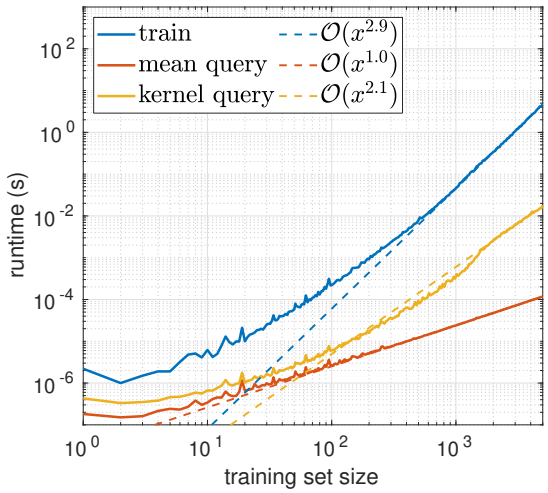
Since it is symmetric and positive definite, the matrix K can efficiently be inverted using the Cholesky decomposition $LL^T = K$, and by solving a triangular system to obtain L^{-1}. Note that all of these operations can be executed in-place[17], such that only a single $\mathbb{R}^{(d+1)N_p \times (d+1)N_p}$ matrix must be stored, which dominates memory consumption for large training sets. More precisely, L and L^{-1} are stored in a single matrix, while

$$L^{-1} \begin{bmatrix} f^p \\ f^g \end{bmatrix} \quad \text{and} \quad \underbrace{L^{-1^T} L^{-1}}_{K^{-1}} \begin{bmatrix} f^p \\ f^g \end{bmatrix} \tag{12.4}$$

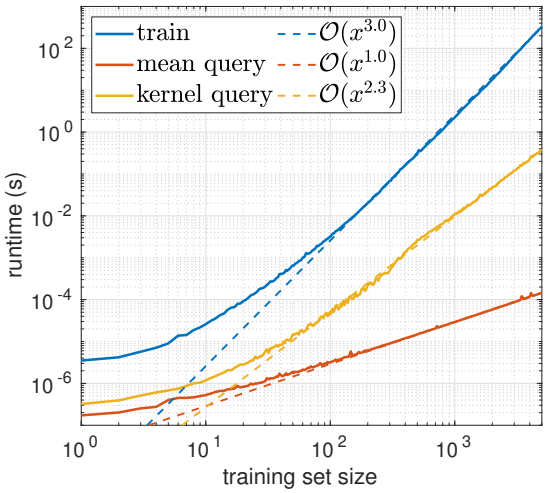
are contained in vectors. When adding additional measurements, matrix and vectors can be updated without the need for a new full Cholesky decomposition of the joint training data. Ultimately the training step has a computational complexity of $\mathcal{O}(N_p^3)$. Evaluating m_{post} at a single test point then involves a single vector-vector multiplication with a runtime of $\mathcal{O}(N_p)$. Similarly, matrix-vector products

[17] These in-place operations are implemented, for example, in the Eigen3 library [Guennebaud et al., 2010].

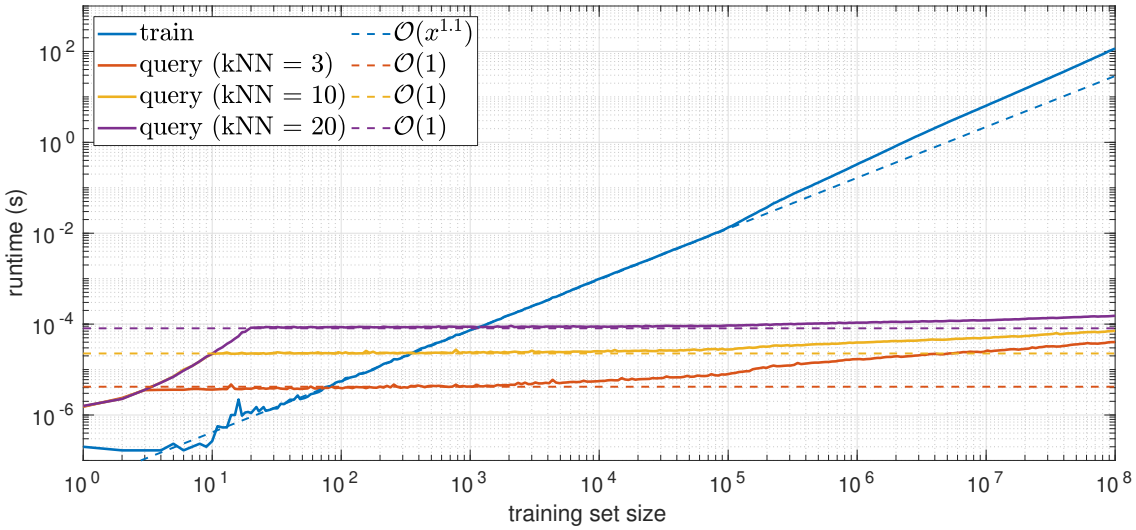
are calculated for a call to k_{post}, leading to a complexity of $\mathcal{O}(N_p^2)$. This is verified with a benchmark in figures 12.1a and 12.1b.



(a) GPIS: Point observations only.



(b) GPIS: Point and gradient observations.



(c) Local GPIS: Point and gradient observations. The query runtime is dominated by the inversion of the matrix K, such that posterior mean and variance function evaluations behave almost identical.

Figure 12.1: Mean Runtime comparison between GPIS and local GPIS. Each Gaussian process is trained 10 times for each training set size that is sampled from the surface of the Stanford bunny. Each of these instances is queried 100 times for posterior mean and variance.

The standard GPIS approach is practical for relatively small training set sizes due to the fast evaluation of mean and kernel functions after training. For example, all images in chapter 11 were created this way by sampling the posteriors with test values on a fine grid[18]. As previously mentioned, a standard use case for surface reconstruction contains up to several million training points. This introduces horrendous runtimes for training and querying. Moreover, keeping the matrix containing L and L^{-1} in memory becomes infeasible for most computers. Therefore, a

[18]The surface is obtained using the *contourf* (two dimensional) and *isosurface* (three dimensions) functions of Matlab.

local GPIS strategy is used similar to the one proposed in [Vasudevan et al., 2009]. Instead of training the Gaussian process on all training points, only a small subset is considered, which locally approximates the GPIS. Consider a single test point where the process is to be evaluated. A subset of $n_{kNN} \in \mathbb{N}$ training points is obtained from a k-nearest neighbor (k-NN) algorithm. These are used to train a GPIS, which is subsequently queried at the test point. Note that now no components of the posterior functions can be precomputed, since the k-NN set may change with every test point. Hence, the full formulation (11.48) is computed for each query test point, i.e. especially the matrix K is recomputed and inverted each time. In order to improve the runtime for the k-NN search, a K-d tree of all training points is maintained[19]. Then the training step only consists of creating or updating the K-d tree. The computational complexity of this new training step is $\mathcal{O}(N_p)$. Furthermore, since n_{kNN} is fixed and the nearest neighbors are obtained in $\mathcal{O}(1)$, the computational complexity of each query is also constant. This is yet again verified with a benchmark in figure 12.1c.

Local GPIS versions of previous two dimensional examples are given in figures 12.2 and 12.3. The discontinuities correspond to changes of the k-NN set of training points and describe Voronoi cells. Note that the smoothness of the contours increases closer to the zero level set. The implicit surfaces are almost identical to their counterparts from section 11.3. Similarly, the local GPIS still proofs to be robust to large gradient measurement errors, as illustrated in figure 12.4. Finally, a three dimensional local GPIS example is given in figure 12.5. Both the surface geometry from the posterior mean zero level set and the posterior variances look almost identical to their non-local counterpart from figure 11.6.

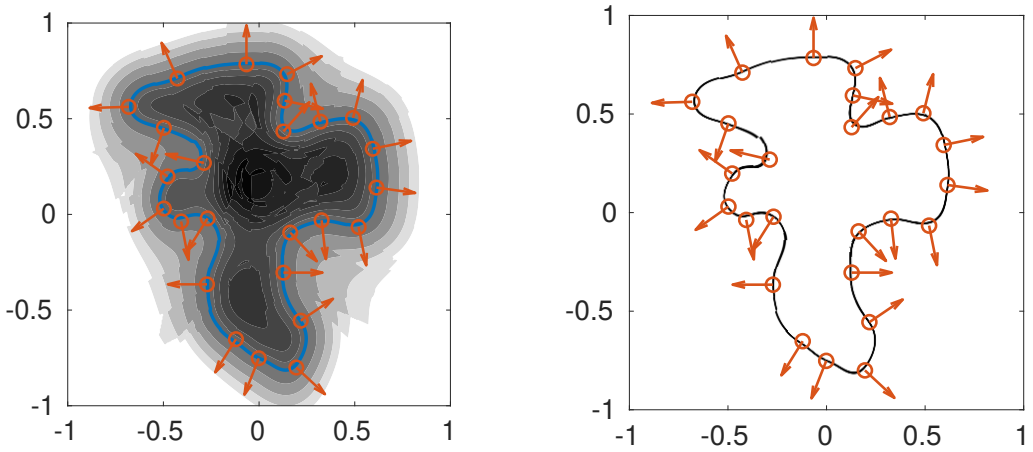
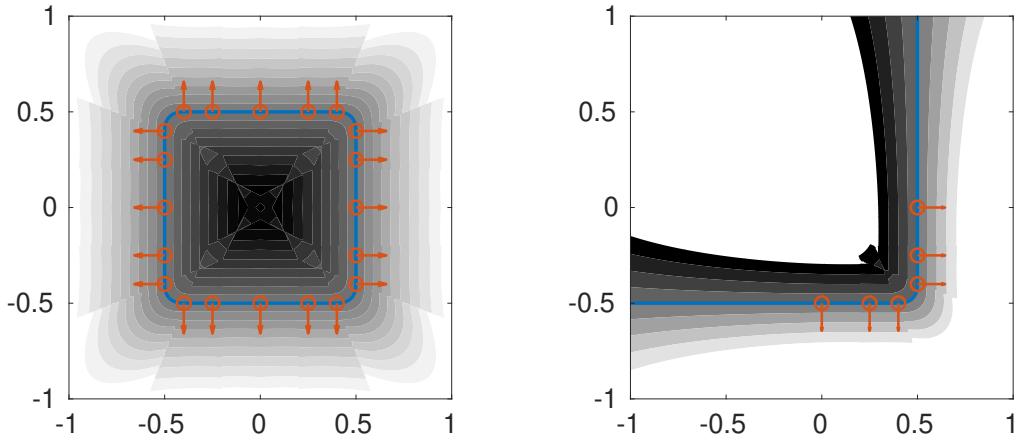
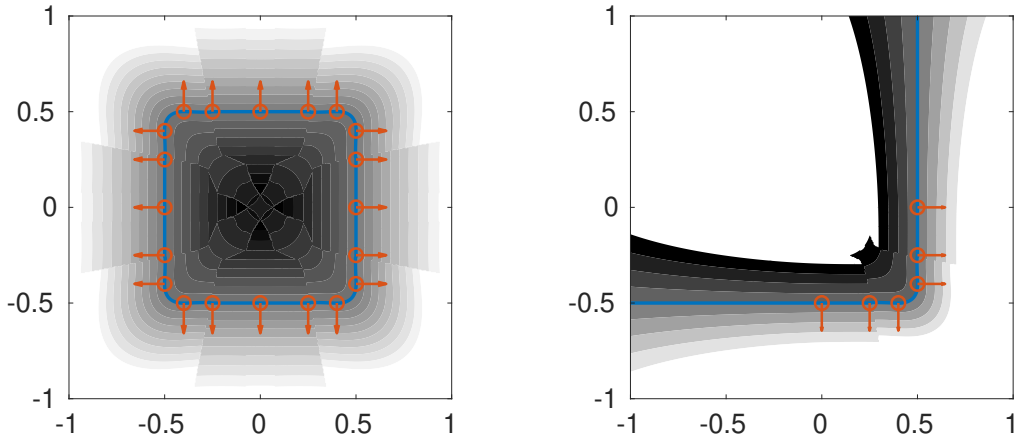


Figure 12.2: Two dimensional local GPIS example with point and surface normal measurements. The setup is identical to figure 11.7. The amount of points considered for the local approximation is $n_{kNN} = 5$. Left: Contour line of Gaussian process posterior. The blue line indicates the zero level set. Right: Probability of posterior to be close to 0, i.e. inside $[-\epsilon, \epsilon]$ visualized with shades of gray.

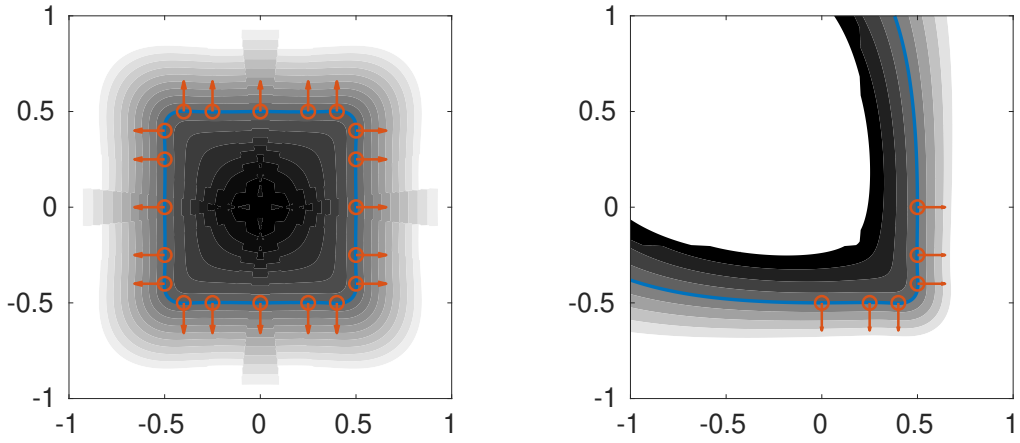
[19] We are using the nanoflann library [Blanco and Rai, 2014] for that purpose in our implementation.



(a) $n_{kNN} = 2$



(b) $n_{kNN} = 3$



(c) $n_{kNN} = 5$

Figure 12.3: Cube and corner two dimensional local GPIS example with different numbers of nearest neighbors. The setup is identical to figure 11.8a. Here, however, we use the local GPIS.

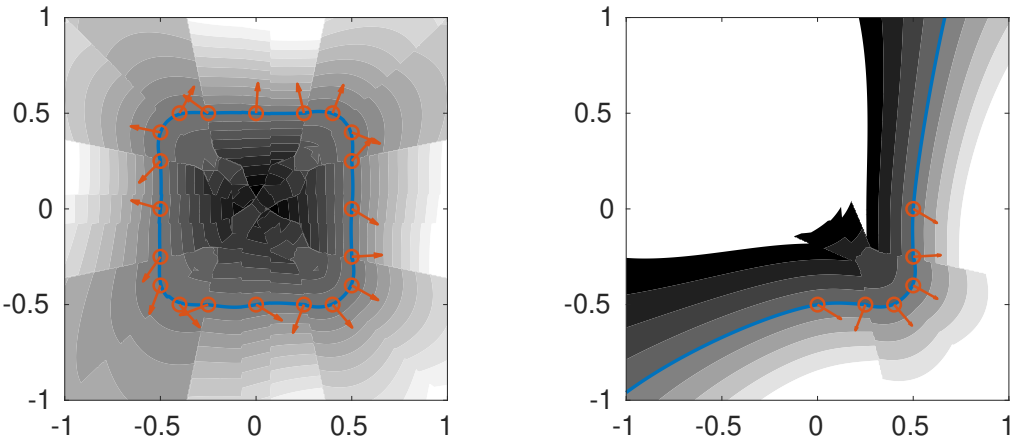


Figure 12.4: Local GPIS approximations for disturbed normal measurements in two dimensions. The setup is identical to figure 11.8b. Here, however, we use the local GPIS with $n_{kNN} = 3$.

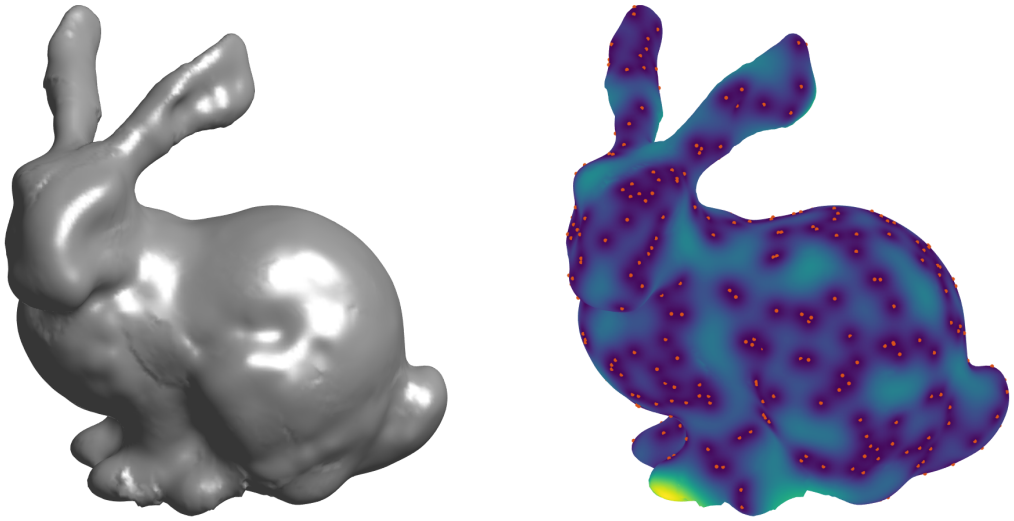


Figure 12.5: Three dimension local GPIS example of the Stanford bunny. The setup is identical to figure 11.6. Here, however, we use the local GPIS with $n_{kNN} = 10$.

12.2 Intersections - Meshing

When computing a NBV using a sampling based approach, many different view candidates need to be evaluated. GPIS-ray intersections need to be computed for every individual pose and measurement ray. Depending on the utilized sensor and the number of RRT samples used in each NBV iteration, the amount of required intersection computations can be very large and heavily dominates the total runtime. Therefore we propose a new approach for a fast computation of these intersections.

The task we are trying to solve in this section is very similar to topics from computer graphics that include ray-casting. One of its most widespread application includes a standard rendering pipeline of three dimensional objects. This area has undergone a constant development in the last decades, so that today specialized hardware (GPUs) is available that is predestined to efficiently solve these problems in a highly parallelized way. A two dimensional rasterized (pixel grid) orthographic projection of an object can be obtained from its representation as a polygon mesh. Each pixel of the rendered image can be thought of as a ray from the point of view through the pixel center to the object. For opaque objects, a technique called *depth buffering* is commonly employed to only visualize non-occluded parts. The buffer is a two dimensional array with the same resolution as the rendered image and contains the distance from the camera center to the mesh along the associated rays. It is computed on the GPU as follows. For a triangular mesh, each triangle is projected to the image plane and rasterized. The distance to the camera center is then stored in the associated depth buffer pixels. If another triangle must be rendered at a previously set pixel, the depth value is overridden if the new pixel is closer to the camera. Hence, given a mesh representation of the implicit surface and the measurement ray pattern, GPIS-ray intersection points are easily obtainable using this rendering pipeline. The three dimensional point positions are then obtained from the direction of the individual measurement rays and the depth buffer.

Now the main difficulty lies in obtaining and maintaining this mesh representation of the surface. There exist a large number of approaches to build meshes from point cloud data that may also include normal observations. A comprehensive overview over the state of the art in that field is given in [Berger et al., 2014], together with a benchmark comparison in [Berger et al., 2013]. Especially methods using RBFs are of interest in our context, because they solve (11.18), i.e. they are polyharmonic splines that implicitly define a surface. For example, in [Carr et al., 2001] a RBF approach is presented that implicitly describes a surface and also considers normal measurements. They achieve impressive reconstruction results and their algorithm even permits a considerable number of measurements. However, the number of observations directly translates to larger linear systems that must be solved (iteratively) and ultimately total runtime. While this is fine for the use case of object reconstruction, it is not sufficient for NBV planning. There, new views must be computed sufficiently fast, especially in the non-model-based case. In addition, the runtime should not increase excessively for a large number of observations. For this purpose, a meshing method is desirable that iteratively updates the surface mesh for each new measurement data obtained from the NBV iteration.

The main idea of our approach is to maintain a spatial tetrahedral mesh that describes space classified as interior. A triangular surface mesh is then given by its facets. The choice of a tetrahedral structure has two main reasons:

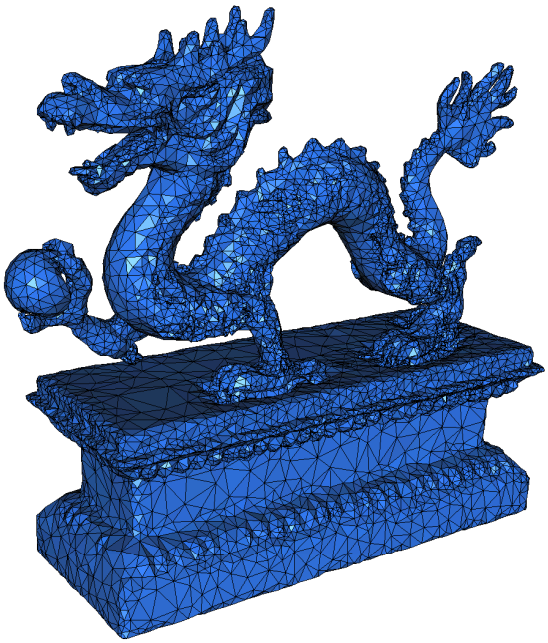
- It is easy to refine. While the (local) GPIS is an implicit function, the associated Gaussian process explicitly assigns each test point its level set. This value can be interpreted as a distance to the implicit surface and can be used iteratively to obtain an approximate surface point. Hence, a Delaunay refinement strategy can be used to iteratively approximate the surface mesh from the boundary of a tetrahedral mesh (see [Alliez et al., 2019]). Starting from a single tetrahedron, the volumetric mesh is subdivided until a desired accuracy, that is defined by the beforehand mentioned zero level set distance, is achieved. In order to keep memory usage low but still achieve a high approximation accuracy, the tetrahedral mesh is usually designed to have a much finer structure towards its facets. This way it adequately approximates the implicit function, while improving memory consumption.

- It is easy to locally remesh. As previously mentioned, it is inefficient to recompute the full mesh in order to integrate new measurement observations. A tetrahedral mesh allows for local remeshing by applying the steps described in the previous point to only a subregion of the tetrahedral mesh. The extend of this local area is also limited by the point selection of the local GPIS approach. This means that only areas that contain test points that are affected by the new measurements must be remeshed. Those are all points with kNN sets that contain new observations. Unfortunately, this has not yet been further analyzed due to time limitations.

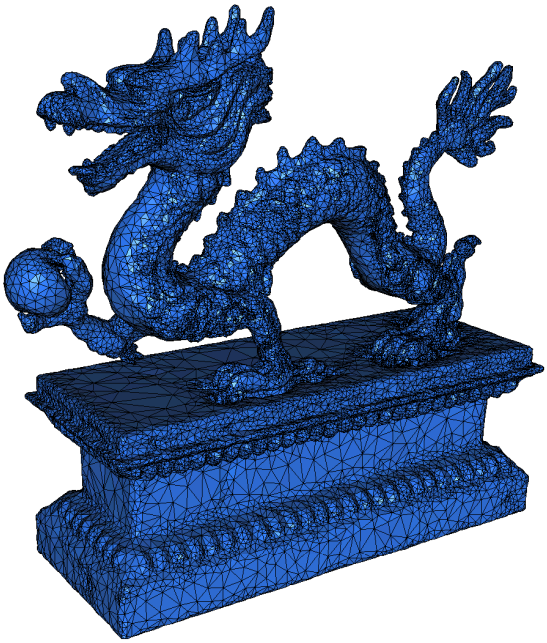
  Note that such a remeshing step would be much harder to implement for a simple non-volumetric surface mesh representation, because each new measurement could significantly affect the topology.

We will now give some examples on the described mesh representation, the meshing runtime and the performance of the GPIS-ray intersection computations. For that purpose, a point cloud including point normals is generated from a mesh. These points are then treated as a single, large measurement observation. A full surface representation is then constructed using local GPIS and the 3D (tetrahedral) mesh generation component [Alliez et al., 2019] of the C++ *Computational Geometry Algorithms Library* (CGAL) [The CGAL Project, 2019]. The tetrahedral meshes are visualized using the tool MEDIT [Frey, 2001]. Figure (12.6) shows the example of a dragon sculpture for different target accuracies. Here, the Delaunay refinement can be observed, i.e. the refinement of the mesh around geometric details. In these areas the number of polygons is higher while their sizes are smaller. Furthermore, despite a substantial amount of measurement points, a reasonable runtime can be achieved. Note, however, that there is still always a trade-off between runtime and surface representation accuracy. Depending on the complexity of the implicit surface described, the computational overhead increases and is proportional to the the relation between objects surface area and interior volume. This effect is noticeable in figure (12.7). Despite a similar magnitude of measurement points the meshing step requires a much higher runtime than in the previous example. Note that for practical application in a NBV-RRT context, the model representation is updated with new measurement data in each iteration, such that the actual meshing cost is much lower. Ultimately this methods strength lies in the fast computation of GPIS-ray intersection points for viewpoint information gain assessment. This is visualized

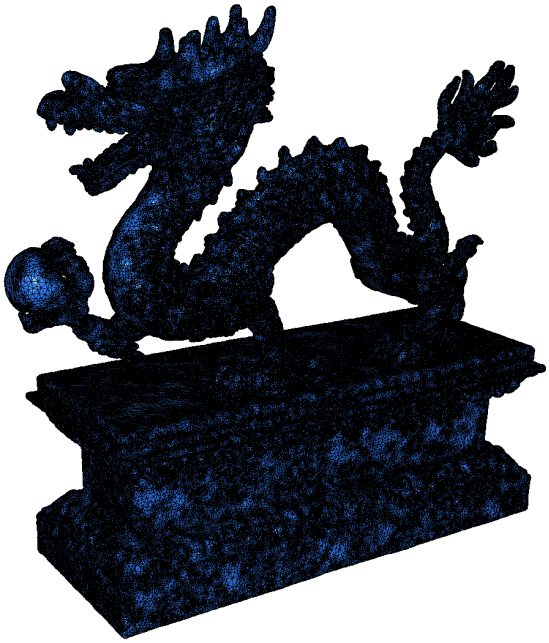
in figure 12.8. There, we measured a runtime of 7.6×10^{-4} s for the computation of a pattern of 100×100 intersection points using a NVIDIA GTX 1050 TI GPU.



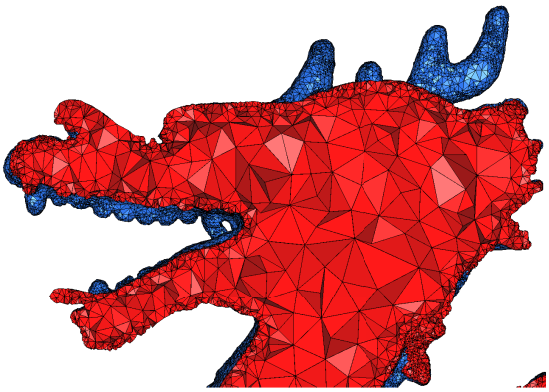
(a) Low meshing accuracy (73 850 faces). Required runtime of 18.30 s.



(b) Medium meshing accuracy (244 916 faces). Required runtime of 59.32 s.



(c) High meshing accuracy (3 383 556 faces). Required runtime of 744.23 s.



(d) Section through the tetrahedral mesh from figure 12.6c. Tetrahedrons are visualized in red, while facets are colored blue. Notice the finer mesh resolutions towards the surface.

Figure 12.6: Tetrahedral mesh of the local GPIS of a dragon sculpture. A total amount of 105 259 points together with their normals were used as measurement data. All computations were run on an Intel i7 4790k CPU in parallel using 8 threads.

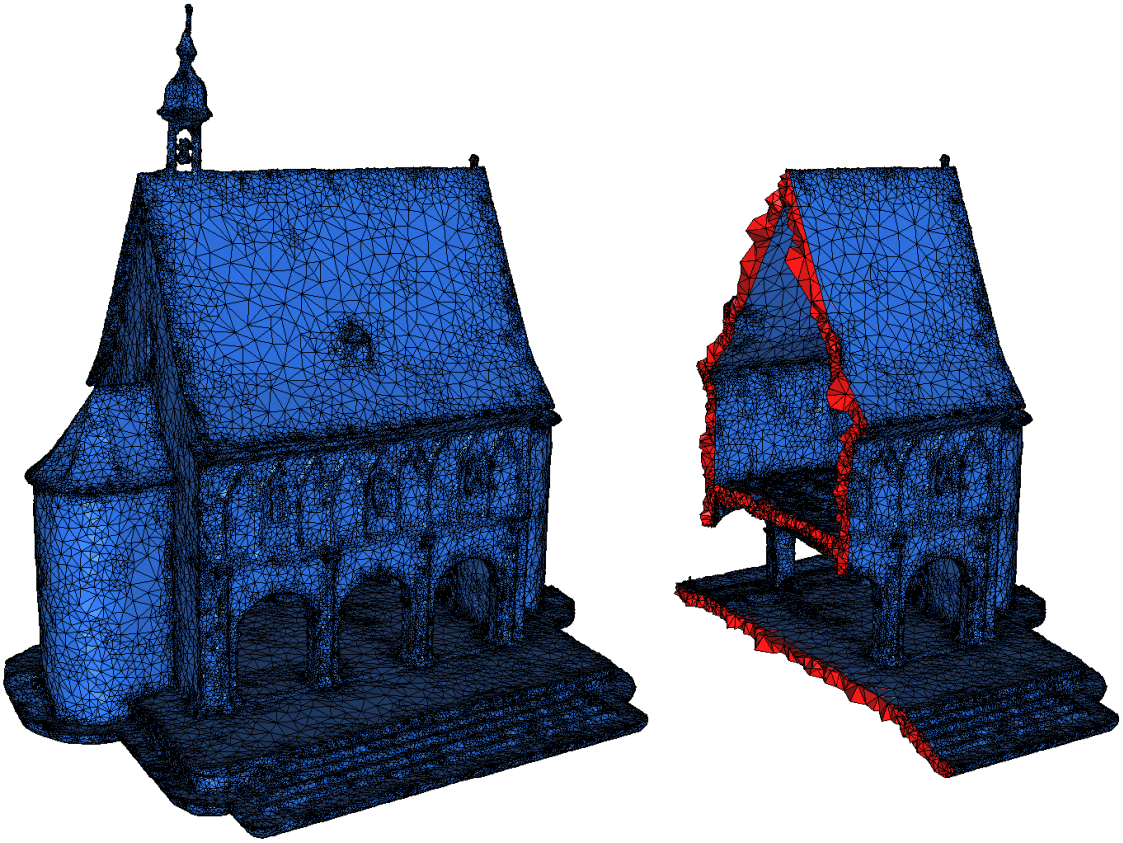


Figure 12.7: Tetrahedral mesh of the local GPIS of the King's Hall of Lorsch Abbey (see section 10.1 for details on the model). A total amount of 250 508 points together with their normals were used as measurement data. The surface meshing required 107 s. The resulting surface mesh consists of 567 170 triangles and is visualized in blue. The right image illustrates a cut through the tetrahedral mesh with tetrahedrons colored in red. All computations were run on an Intel i7 4790k CPU in parallel using 8 threads.

12.3 Intersections - Ray-Marching

As we have seen in the last section, computing GPIS-ray intersection from a triangular mesh representation is very efficient. Creating the mesh, however, introduces a computational overhead that grows proportional with a desired meshing accuracy. Therefore, in this section we present another method to calculate these intersections directly, without the need for any model representation. It is again motivated from rendering approaches of implicit surfaces from the field of computer graphics and relies on ray marching. This means that for a given sensor position, each measurement ray is iterated until a stopping criteria is met (e.g. it intersects the implicit surface). In our case, this criteria is satisfied when the local GPIS evaluated at the current ray position is zero. A pseudo code for the computation of GPIS-ray intersection points is given in algorithm 12.1. The main difficulty of ray-marching is the choice of the step lengths. A common technique to determine the step length is called sphere marching [Hart, 1996]. There, the stepsize is computed from a distance function, that returns the distance to the implicit surface. Unfortunately, this approach is not applicable to our problem, since the (Euclidean) distance to the GPIS can not be calculated. Other popular methods employ a gradient based root finding method,

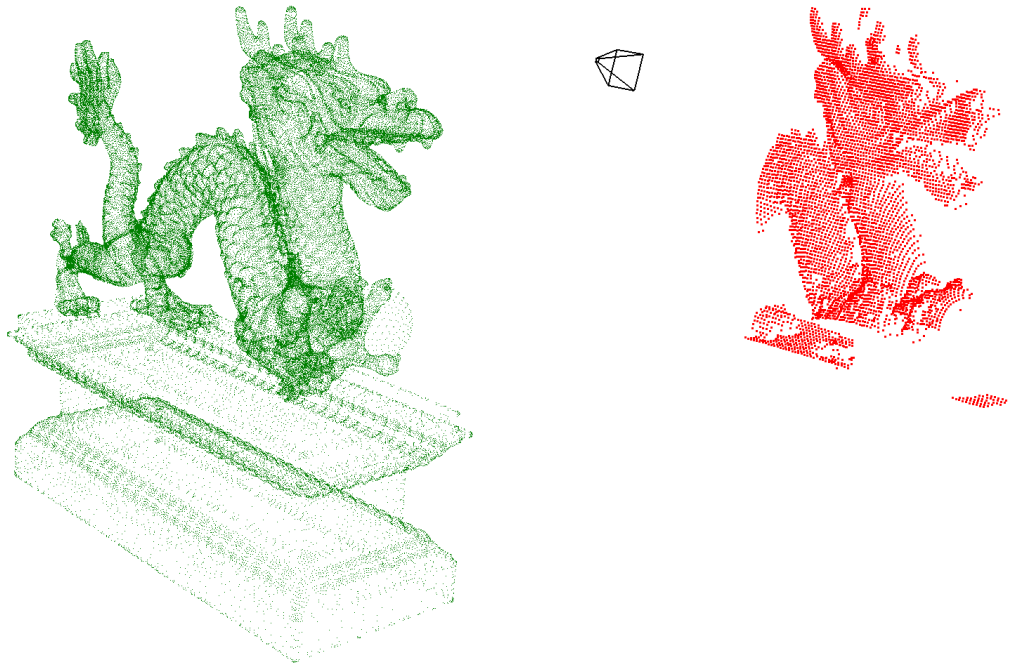


Figure 12.8: Visualization of GPIS-ray intersection points for the dragon sculpture data. Left: Measurement data consisting of 105 259 surface points and their normals. Right: Computed GPIS-ray intersection points for a depth sensor with 100×100 measurement rays.

which can efficiently be implemented on a GPU (e.g. [Singh and Narayanan, 2010]). While the GPIS allows evaluation of gradient information, discontinuities due to the local kNN approximation may lead to problems here as well. This needs to be analyzed in future work. As a proof of concept we implemented a simple stepsize computation. It is computed as follows:

- If no interior point (i.e. negative mean posterior) has been observed during ray-marching, the stepsize is chosen as 0.5 times the distance of the current ray point to the closest point in the current kNN set.

- If an interior point has been observed, the stepsize is computed from a linearization of the interior and exterior ray points which are closest to the zero level set. More precisely, let $t^E, t^I \in \mathbb{R}_{>0}$ be the distances on the ray where exterior and interior points have been observed with a posterior mean value of $m^E_{\text{post}} \in \mathbb{R}_{>0}$ and $m^E_{\text{post}} \in \mathbb{R}_{<0}$. Then

$$\texttt{step\_size} = \frac{m^E_{\text{post}}(t^I - t^E)}{m^E_{\text{post}} - m^I_{\text{post}}}. \tag{12.5}$$

Using this naive implementation, a runtime around 0.6 s - 1.2 s is achieved when computing 100×100 intersection points (e.g. figure 12.8) on 8 threads of an Intel i7-4790k CPU. We believe that a significant speedup could be achieved if the same code is ported to a GPU, since all rays can be handled independently. Then all measurement points and normals are stored in GPU memory and a graphics card accelerated kNN and GPIS implementation would be required.

Algorithm 12.1: Pseudo-code of the ray-marching approach for computing GPIS-ray intersections.

```
// max_dist - maximum distance threshold
// max_it   - maximum number of iterations
// eps      - small positive constant

// returns [bool, point], where the boolean indicates if the ray intersects the
// surface. If this boolean is true, the returned point is the intersection point.
function computeIntersectionRaymarching(origin, direction)

   ray_dir   = normalize(ray_dir); // direction of measurement ray
   ray_point = origin;             // current point on the ray
   ray_dist  = 0;                  // distance traveled along the ray
   step_size = 0;                  // step size from current to new ray point
   it_count  = 0;                  // iteration counter

   while (ray_dist < max_dist) && (it_count < max_it)
      // evaluate GPIS
      kNN_set = {kNN(ray_point)};
      mean = GPIS(kNN_set, ray_point)

      // check if implicit surface has been hit
      if (abs(mean) < eps)
         return [true, ray_point];
      end if

      // compute new step size and increment iteration count
      step_size = raymarchingStepsize(...);  // see text for further details
      it_count = it_count + 1;

      // iterate to new ray point
      ray_dist  = ray_dist + step_size;
      ray_point = origin + ray_dist * direction;
   end

   return [false, ray_point];
end function
```

Since the meshing approach only evaluates the implicit mesh at only a few points for each surface polygon, the intersection points computed there suffer from the meshing accuracy trade-off. This is not the case for ray-marching, because the GPIS is evaluate directly at the intersection points for each ray that hits the surface when the iteration terminates. Although we do not need a mesh representation (which would introduce a computational overhead) using the ray-marching approach, individual intersection points cannot be calculated with the same efficiency due to the iterative nature of the algorithm.

12.4 Final Remark

We want to point out a problem that may arise when executing the NBV algorithm outlined above. Due to extrapolation errors during surface estimation, the expected GPIS-ray intersection points for a NBV are not identical to real measurements. In the worst case, each measurement ray can even completely miss the object. Therefore we believe that an additional representation of the observed space as a voxel map is unavoidable. Then a new gain formulation is required that combines (12.2) with the visibility of the associated voxels.

Part IV

Conclusion & Future Work

Conclusion

13

In this thesis, we propose new methods for optimization-based data collection for the purpose of obtaining 3D reconstructions with high geometric accuracy that scale well for large scale applications. They are formulated as general CPP problems that are solved iteratively. In each iteration, an RRT is spanned that gives a set of sensor viewpoint candidates together with collision-free trajectories that connect them with the current pose. The NBV is chosen from this set as the pose that yields the highest information gain in reconstruction quality, formulated by a probabilistic model. For this purpose, we consider two classes of sensors, which are used fundamentally differently for obtaining the reconstructions.

- **Image sensors**
  A reconstruction is created from a set of images using SfM. This photogrammetric method is able to achieve high detail photorealistic reconstructions at the cost of a high computation complexity, which makes it unfeasible for real-time applications. The main difficulty, therefore, lies in capturing an object sufficiently prior to the reconstruction while keeping the total number of images reasonable. This motivates the necessity for a CPP procedure, which estimates the expected reconstruction quality at the time of recording for an appropriate NBV selection. We quantify this expected reconstruction quality of surface points in terms of covariance matrix information, constructed from an assumption on the image plane error and a camera model that can be updated efficiently. This allows us to formulate an entropy-based information gain function that is motivated by OED. While some knowledge of the geometry is required, we present various options for model representations, each of which targeted at individual use cases. We also describe how our approach can be used in combination with an autonomous exploration algorithm, eliminating the need for preliminary information.

  Depth can not be extracted from a single image, such that each individual surface points must be observed in multiple images to allow for a reconstruction. This is the key difference to classic CPP that only considers single-view coverage. Hence, to the best of our knowledge, our approach constitutes the first multi-view CPP algorithm for recording arbitrary geometric shapes. Moreover, we do not simply distinguish between non-reconstructable and reconstructable surface points. By quantifying the expected reconstruction quality in each step of the iterative NBV approach, a desired reconstruction quality can be achieved that functions as the stopping criterion for the recording procedure. Using an elaborate simulation environment and evaluation pipeline, we verify the correctness of our approach. There, the obtained SfM reconstruc-

tion is compared with a ground-truth mesh and the expected reconstruction quality. The simulation results show that a high level of detail is achievable from a reasonably sized set of photographs obtained from our CPP method. Furthermore, the quality estimate provides information about the actual reconstruction error during the recording procedure. While this provides a good estimate most of the time, we have observed some discrepancies that are most noticeable in the case of many images observing the same areas. These arise because photographs that are assumed to contribute to the reconstruction of these areas may eventually be neglected in the SfM pipeline, or there are differences between real and estimated observability.

- **Depth sensors**
  The model representation is given as a point cloud, such that a different assessment of model reconstruction quality is required. For this purpose, we employ GPIS, where the object's surface is described implicitly from a set of point observations. The posterior kernel function then provides a measure of the surface quality and can be used in an information gain formulation for NBV computations. In order to evaluate such a function, intersections between the implicit surface and the measurement rays have to be determined. Two approaches are presented that compute these quantities:

  - A surface mesh is created from the implicit function using a tetrahedral mesh approximation. The intersection points are then extracted from the depth buffer of a GPU, which is extremely efficient.
  - Using ray-marching, intersection points are computed iteratively. This approach does not require a model representation besides the raw measurement data.

  Polyharmonic kernels are used in the Gaussian process that implicitly describe the surface. We give an elaborate motivation and theoretical derivation of polyharmonic kernels for probabilistic surface modeling to address some shortcomings that currently exist in the literature. Especially the connection to polyharmonic splines as regularized regression problem is highlighted and explicit kernel functions are given. Furthermore, we show how the quality of the surface representation can be significantly improved by including normal observations. A local GPIS approach is presented, where Gaussian processes trained from small subsets of measurements are considered that locally describe the surface implicitly. This makes the application to large scale models and a large point clouds feasible. Especially the GPIS-ray intersection computation profits from local GPIS. While the tetrahedral meshing approach still introduces an overhead for mesh generation that depends on the point cloud size, we believe that runtimes can be improved with incremental update formulations in the future. On the other hand, the ray-marching approach allows for constant complexity independent of point cloud size. Ultimately, we lay the foundation for a depth sensor CPP approach that maintains a model representation with infinite detail. First tests showed very promising results regarding runtime and surface representation quality that indicate real-time capabilities.

Through iterative computation of NBVs, ultimately full coverage is achieved up to geometric and dynamic constraints. Due to the general CPP formulation many RRT variations are possible, such that especially dynamics and control of a sensor mounted robotic system can be considered. This allows for a wide range of robot-aided applications, e.g. using UAVs, UGVs or UUVs or arbitrary multi-sensor setups. All proposed CPP approaches were designed with real-time application in mind. We have achieved promising results regarding the algorithm's computational complexity, as they scale well with the size of the object that is to be recorded. This indicated the feasibility of real-time applications, and an implementation for non-model-based (online) planning is therefore conceivable. Ultimately, we believe that our new methods have enormous potential for future application, especially for medium to large scale scenarios in areas such as civil engineering, terrestrial surveying, and archaeology.

Future Work

14

The CPP algorithms presented in this thesis are given in a very general formulation. While this offers a high degree of flexibility and allows for a variety of application scenarios, those are not presented here and are therefore subject to further research. This especially includes UAV mounted sensors for autonomous recordings.

On these systems, new questions regarding runtime complexity arise. As previously stated, we believe that current performance culprits of our algorithms could receive significant speedup if implemented with GPU acceleration. Those especially include voxel visibility checks for the SfM-CPP algorithm (see section 8.3.2) and the computation of local GPIS-ray intersections in the ray-marching subroutine (section 12.3). In this context, a performance analysis is desirable that considers low-power embedded GPUs such as the NVIDIA Jetson Xavier, which are suitable for robotic platforms. Similarly, the meshing approach for computing GPIS-ray intersections (section 12.2) requires additional performance improvements. While the computation of the actual intersection points is exceptionally fast, a surface mesh representation is required, which introduces a computational overhead that increases with the number of observations. We outlined how this overhead could be reduced with incremental mesh updates, but we still lack an implementation, along with a runtime analysis. Additionally, the question arises how well such an algorithm scales for large numbers of measurement observations.

Other interesting topics arise when distinguishing between model-based and non-model-based planning. For example, practical realizations of completely autonomous CPP methods could be considered, that combine our algorithms with autonomous exploration frameworks. Then the SfM-CPP algorithm would be able to record structures without any prior information on the geometry. Future research could also focus on obtaining optimal[20] coverage trajectories in the case of offline planning. One approach commonly employed in the NBV-RRT framework for CPP considers RRT resampling strategies (e.g. [Englot and Hover, 2013]). Such a method must be adapted accordingly in order to be applicable to our CPP algorithms.

Finally, the surface meshes obtained from the meshing approach for the computation of GPIS-ray (section 12.2) showed interesting properties regarding approximation quality and especially runtime. Therefore, the question arises how this compares to other surface reconstruction methods, such as Poisson or RBF.

[20] E.g. shortest / fastest path, minimum number of recording poses.

Appendix

A Weyl's inequality

Theorem .1 (Weyl's inequality [Weyl, 1912])**.** Let $A, B \in \mathbb{R}^{N\times N}$ and $\lambda_1(A) \leq \lambda_2(A) \leq \ldots \leq \lambda_N(A)$ denote the eigenvalues of A. Then

$$\begin{aligned} &\lambda_i(A) + \lambda_j(B) \leq \lambda_{i+j-1}(A+B) \\ &\lambda_i(A) + \lambda_j(B) \geq \lambda_{i+j-N}(A+B) \end{aligned}, \quad \forall i,j \text{ with } 1 \leq i+j-1 \leq N. \tag{A.1}$$

A special case of Weyl's inequality which is interesting for us is given for positive semi-definite matrices. Here, all eigenvalues are greater or equal to zero. Letting $i = 1$ (or $j = 1$ respectively), we get a new estimate of the eigenvalues of the sum, which is given by

$$0 \leq \left\{ \begin{array}{c} \lambda_i(A) \\ \lambda_i(B) \end{array} \right\} \leq \lambda_i(A+B), \quad \forall i = 1, \ldots, N. \tag{A.2}$$

Note that strict positive definiteness equivalently results in strict inequalities.

B Gain Function Derivative

Solving an OCP using gradient-based approaches requires derivatives of the objective function with respect to parameters describing a discretized control function. In the context of the OED given in (7.3), the derivatives of the clamped gain function (6.44) are required. In this section such derivatives will be derived analytically. Suppose the camera pose at time t_{n+1} depends on a parameter $u \in \mathbb{R}$, i.e.

$$\xi_{n+1} := \{R(u), t(u)\} : u \in \mathbb{R} \mapsto SO(3) \times \mathbb{R}^3. \tag{B.1}$$

As Ξ_n does not depend on u, the derivative of the gain is then given as

$$\frac{d}{du}\overline{\text{gain}}(p, \Xi_n, \xi_{n+1}) = \frac{\left(\frac{d}{du}\overline{\det}\Sigma^{-1}(p, \Xi_{n+1})\right)}{2\ \overline{\det}\Sigma^{-1}(p, \Xi_{n+1})}. \tag{B.2}$$

The difficulty now lies in calculating the derivative of the determinant term in the numerator. For this purpose, we will first analyze the case without clamping, i.e.

$$\frac{d}{du}\det\Sigma^{-1}(p, \Xi_{n+1}). \tag{B.3}$$

Consider the update formula (6.27). For $p \notin \mathrm{FOV}(\xi_{n+1})$ the updated precision matrix $\Sigma^{-1}(p, \Xi_{n+1})$ does not depend on ξ_{n+1} and the derivative (B.3) is zero. Hence, we are now interested in the case $p \in \mathrm{FOV}(\xi_{n+1})$. From update formula (6.27) and the definition of Θ^{-1} (see equation (6.14)) we obtain the identity

$$\begin{aligned}\Sigma^{-1}(p, \Xi_{n+1}) &= \Sigma^{-1}(p, \Xi_n) + \Theta^{-1}(p, \xi_{n+1}) \\ &= \Sigma^{-1}(p, \Xi_n) + \frac{1}{\gamma^2 \langle p - t(u), R^z(u) \rangle^2} \left(I_3 - \frac{(p - t(u))}{\|p - t(u)\|_2} \frac{(p - t(u))^T}{\|p - t(u)\|_2} \right),\end{aligned} \tag{B.4}$$

where $R^z(u)$ is the z component of the matrix $R(u)$, i.e. $R^z(u) = R(u)e_3$. We can now express the derivative of (B.3) in terms of the derivative of (B.4) using Jacobi's formula:

Theorem .2 (Jacobi's formula)**.** Let $A(u) : \mathbb{R} \to \mathbb{R}^{N \times N}$ be a map. Then

$$\frac{d}{dp} \det(A(u)) = \mathrm{tr} \left(\mathrm{adj}(A(u)) \frac{dA(u)}{du} \right), \tag{B.5}$$

where $\mathrm{adj}(\cdot)$ is the adjugate operator.

For simplicity of notation the argument u is now omitted. The derivative of (B.4) is computed as

$$\begin{aligned}\frac{d}{du}\Sigma^{-1}(p, \Xi_{n+1}) &= \\ &= \frac{2\left(\langle \frac{d}{du}t, R^z \rangle - \langle p - t, \frac{d}{du}R^z \rangle\right)}{\gamma^2 \langle p - t, R^z \rangle^3} \left(I_3 - \frac{(p-t)}{\|p-t\|_2} \frac{(p-t)^T}{\|p-t\|_2} \right) \\ &- \frac{1}{\gamma^2 \langle p - t, R^z \rangle^2} \left(\frac{2\langle p - t, \frac{d}{du}t \rangle}{\|p-t\|_2^4} (p-t)(p-t)^T \right) \\ &+ \frac{1}{\gamma^2 \langle p - t, R^z \rangle^2} \left(\frac{1}{\|p-t\|_2^2} \left[\left(\frac{d}{du}t \right) (p-t)^T + (p-t) \left(\frac{d}{du}t \right)^T \right] \right).\end{aligned} \tag{B.6}$$

After regrouping terms and substituting $v := p - t$, this simplifies to

$$\frac{d}{du}\Sigma^{-1}(p, \Xi_{n+1}) = \alpha_0 \left(\alpha_1 I_3 + \alpha_2 v v^T + \alpha_3 \left[\left(\frac{d}{du}t \right) v^T + v \left(\frac{d}{du}t \right)^T \right] \right), \tag{B.7}$$

where the scalars $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ are given by

$$\begin{aligned}\alpha_0 &= \frac{1}{\gamma^2 \langle p - t, R^z \rangle^2} \\ \alpha_1 &= \frac{2\left(\langle \frac{d}{du}t, R^z \rangle - \langle p - t, \frac{d}{du}R^z \rangle\right)}{\langle p - t, R^z \rangle} \\ \alpha_2 &= -\left(\frac{2\left(\langle \frac{d}{du}t, R^z \rangle - \langle p - t, \frac{d}{du}R^z \rangle\right)}{\langle p - t, R^z \rangle} + \frac{2\langle p - t, \frac{d}{du}t \rangle}{\|p-t\|_2^4} \right) \\ \alpha_3 &= \frac{1}{\|p-t\|_2^2}.\end{aligned} \tag{B.8}$$

Now Jacobi's formula could be applied. However, it would require the computation of an adjugate matrix and a matrix-matrix product in each step, which is computationally expensive. Therefore, we derive an alternative analytical expression in terms of eigenvalues and eigenvectors.

Let the singular value decomposition of $\Sigma^{-1}(p, \Xi_{n+1})$ be given by

$$\Sigma^{-1}(p, \Xi_{n+1}) = \begin{bmatrix} | & | & | \\ u_1 & u_2 & u_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & \\ & \lambda_2 & \\ & & \lambda_3 \end{bmatrix} \begin{bmatrix} — & u_1^T & — \\ — & u_2^T & — \\ — & u_3^T & — \end{bmatrix}, \tag{B.9}$$

where $0 \leq \lambda_1 \leq \lambda_2 \leq \lambda_3 \in \mathbb{R}$ are the eigenvalues of $\Sigma^{-1}(p, \Xi_{n+1})$ with associated eigenvectors $u_1, u_2, u_3 \in \mathbb{R}^3$. Then the adjugate of $\Sigma^{-1}(p, \Xi_{n+1})$ can be expressed as

$$\begin{aligned} \mathcal{A} :&= \operatorname{adj}\left(\Sigma^{-1}(p, \Xi_{n+1})\right) \\ &= \begin{bmatrix} | & | & | \\ u_1 & u_2 & u_3 \\ | & | & | \end{bmatrix} \begin{bmatrix} \lambda_2\lambda_3 & & \\ & \lambda_1\lambda_3 & \\ & & \lambda_1\lambda_2 \end{bmatrix} \begin{bmatrix} — & u_1^T & — \\ — & u_2^T & — \\ — & u_3^T & — \end{bmatrix}. \end{aligned} \tag{B.10}$$

Using (B.7) and (B.10) we can now apply Jacobi's formula, resulting in

$$\begin{aligned} \frac{d}{du} \det\left(\Sigma^{-1}(p, \Xi_{n+1})\right) &= \operatorname{tr}\left(\mathcal{A} \frac{d}{du} \Sigma^{-1}(p, \Xi_{n+1})\right) \\ &= \alpha_0 \left(\alpha_1 \operatorname{tr}(\mathcal{A}) + \alpha_2 v^T \mathcal{A} v + 2\alpha_3 v^T \mathcal{A} \left(\frac{d}{du} t\right)\right). \end{aligned} \tag{B.11}$$

Note that most of the $\operatorname{tr}(\cdot)$ operators could be eliminated due to the invariance of the trace under cyclic permutations.

As the determinant of a matrix is the product of its eigenvalues, another formulation for the derivative of $\det \Sigma^{-1}(p, \Xi_{n+1})$ in terms of its eigenvalues is given by

$$\frac{d}{du} \det\left(\Sigma^{-1}(p, \Xi_{n+1})\right) = \lambda_2\lambda_3 \left(\frac{d}{du}\lambda_1\right) + \lambda_1\lambda_3 \left(\frac{d}{du}\lambda_2\right) + \lambda_1\lambda_2 \left(\frac{d}{du}\lambda_3\right). \tag{B.12}$$

Since all terms in equation (B.11) that contain the adjugate can also be expressed in terms of eigenvalues and eigenvectors, i.e.

$$\begin{aligned} \operatorname{tr}(\mathcal{A}) \quad &= \lambda_2\lambda_3 && + \lambda_1\lambda_3 && + \lambda_1\lambda_2 \\ v^T \mathcal{A} v \quad &= \lambda_2\lambda_3 \langle u_1, v\rangle^2 && + \lambda_1\lambda_3 \langle u_2, v\rangle^2 && + \lambda_1\lambda_2 \langle u_3, v\rangle^2 \\ v^T \mathcal{A} \left(\frac{d}{du} t\right) \quad &= \lambda_2\lambda_3 \langle u_1, v\rangle \langle u_1, \frac{d}{du} t\rangle && + \lambda_1\lambda_3 \langle u_2, v\rangle \langle u_2, \frac{d}{du} t\rangle && + \lambda_1\lambda_2 \langle u_3, v\rangle \langle u_3, \frac{d}{du} t\rangle, \end{aligned} \tag{B.13}$$

we can compare coefficients between (B.11, B.13) and (B.12) and identify

$$\frac{d}{du}\lambda_i = \alpha_0 \left(\alpha_1 + \alpha_2 \langle u_i, v\rangle^2 + 2\alpha_3 \langle u_i, v\rangle \left\langle u_i, \frac{d}{du} t\right\rangle\right), \quad i \in \{1, 2, 3\}. \tag{B.14}$$

Instead of matrix-matrix products, this formulation only requires vector-vector products. Having derived the derivative of the determinant of the precision matrix we

can now analyze the corresponding clamped formulation. Similar to (B.12), we can write

$$\frac{d}{du}\overline{\det}\left(\Sigma^{-1}(p,\Xi_{n+1})\right) = \overline{\lambda}_2\overline{\lambda}_3\left(\frac{d}{du}\overline{\lambda}_1\right) + \overline{\lambda}_1\overline{\lambda}_3\left(\frac{d}{du}\overline{\lambda}_2\right) + \overline{\lambda}_1\overline{\lambda}_2\left(\frac{d}{du}\overline{\lambda}_3\right), \quad \text{(B.15)}$$

where

$$\overline{\lambda_i} = \text{clamp}\left(\frac{1}{\lambda_{\max}} \leq \lambda_i \leq \frac{1}{\lambda_{\min}}\right), \quad i \in \{1,2,3\} \quad \text{(B.16)}$$

are the clamped eigenvalues of $\Sigma^{-1}(p,\Xi_{n+1})$ similar to (6.44). The constants $\lambda_{\min}$ and $\lambda_{\max}$ are as in section 6.3.2. Formulation (B.15) deviates from (B.12) if clamping is active for eigenvalues. Hence, those cases need to be distinguished in order to determine $\frac{d}{du}\overline{\lambda}_i$. We could now set the clamped eigenvalue derivative to zero if clamping is active. This makes sense for the $1/\lambda_{\min}$ case as it indicates that a point has been sufficiently observed. Additional observations of the point p would even result in over-observation, resulting in a larger image set and, ultimately, higher SfM reconstruction times. If clamping to $1/\lambda_{\max}$ is active, the information gained for that point is too small. While the gain should also be zero in that case, we can and should still use the derivatives of the un-clamped eigenvalues, as they still point in the direction of highest gain improvements. This can also be realized from figure 6.3. Hence, we only clamp the derivative of the clamped eigenvalue to $1/\lambda_{\min}$ and define

$$\frac{d}{du}\overline{\lambda}_i := \begin{cases} \frac{d}{du}\lambda_i & \text{if } \lambda_i \leq 1/\lambda_{\min} \\ 0 & \text{otherwise} \end{cases}, \quad i \in \{1,2,3\}. \quad \text{(B.17)}$$

Summarizing equations (B.2) and (B.15), the clamped gain derivative is then given by

$$\begin{aligned} \frac{d}{du}\overline{\text{gain}}(p,\Xi_n,\xi_{n+1}) &= \frac{\frac{d}{du}\overline{\det}\left(\Sigma^{-1}(p,\Xi_{n+1})\right)}{2\overline{\lambda}_1\overline{\lambda}_2\overline{\lambda}_3} \\ &= \frac{1}{2\overline{\lambda}_1}\left(\frac{d}{du}\overline{\lambda}_1\right) + \frac{1}{2\overline{\lambda}_2}\left(\frac{d}{du}\overline{\lambda}_2\right) + \frac{1}{2\overline{\lambda}_3}\left(\frac{d}{du}\overline{\lambda}_3\right). \end{aligned} \quad \text{(B.18)}$$

Despite requiring eigenvalues and eigenvectors, the clamped gain derivative does not introduce a significant computational overhead. The eigenvalues are already available since they are also required for gain clamping in gain formulation (6.44). Additional eigenvectors are required, however, due to the structure of $\Sigma^{-1}(p,\Xi_{n+1})$ (symmetric, 3×3) those can be computed cost-efficiently using a single cross-product operation (see appendix C).

C Computing Eigenvalues and Eigenvectors of Symmetric 3×3 Matrices

Let $A \in \mathbb{R}^{3\times3}$ be a symmetric matrix. Applying Cardanos method on the characteristic polynomial (of order 3) of A, an analytic representation can be derived for

the eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3$ of A. They are given by (see [Kopp, 2008]):

$$\begin{aligned} \lambda_3 &= q + 2p\cos(\phi) \\ \lambda_2 &= q + 2p\cos\left(\phi + \frac{2\pi}{3}\right) \\ \lambda_1 &= q + 2p\cos\left(\phi - \frac{2\pi}{3}\right) \\ &= 3q - \lambda_2 - \lambda_3, \end{aligned} \tag{C.1}$$

where

$$\begin{aligned} q &= \frac{\operatorname{tr}(A)}{3} = a_{11} + a_{22} + a_{33} \\ p &= \sqrt{\frac{\operatorname{tr}\left((A - qI)^2\right)}{6}} \\ \operatorname{tr}\left((A - qI)^2\right) &= (a_{11} - q)^2 + (a_{22} - q)^2 + (a_{33} - q)^2 \\ &\quad + 2a_{12}^2 + 2a_{13}^2 + 2a_{23}^2 \\ \phi &= \frac{1}{3}\arccos\frac{\det\left((1/p)(A - qI)\right)}{2}. \end{aligned} \tag{C.2}$$

In addition, the associated eigenvectors are also obtainable analytically. Due to the symmetry of A, the null space of $A - \lambda_i I$ is perpendicular to its column space. Hence, the cross product of two linear independent columns of $A - \lambda_i I$ yields the eigenvector of the eigenvalue λ_i.

A special case arises when $A - \lambda_i I$ does not contain two linear independent columns, but is not zero. Then λ_i is a degenerate eigenvalue with multiplicity 2, i.e. the null space of $A - \lambda_i I$ has dimension 2. The corresponding eigenvectors are still obtainable from a non-zero column u of $A - \lambda_i I$ and any vector $v \in \mathbb{R}^3$ that is not parallel to u with $v \times u$ and $(v \times u) \times v$.

In [Kopp, 2008] the numeric stability of these analytic solutions is analyzed. They observe that due to cancellation, large errors are introduced in the case of high condition numbers. Hence, a hybrid approach is proposed which falls back to computing eigenvalues and eigenvectors using a QL decomposition if the lengths of analytically computed, unnormalized eigenvectors becomes too small. The implementation of this hybrid approach proved to be even faster than the pure analytic method, as fewer conditional branches are necessary.

List of Figures

List of Tables

List of Algorithms

List of Files

Bibliography

[Adler, 2010] Adler, R. J. (2010). *The Geometry of Random Fields*. Society for Industrial and Applied Mathematics.

[Agarwal et al., 2011] Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., and Szeliski, R. (2011). Building rome in a day. *Commun. ACM*, 54(10):105–112.

[Agisoft, 2017] Agisoft (2017*). AgiSoft PhotoScan Professional (Version 1.3.3). `http://www.agisoft.com/downloads/installer/` (Software).

[Agrell, 2019] Agrell, C. (2019). Gaussian processes with linear operator inequality constraints. *Journal of Machine Learning Research*, 20(135):1–36.

[Aicardi et al., 2016] Aicardi, I., Chiabrando, F., Grasso, N., LINGUA, A., Noardo, F., and Spano, A. (2016). Uav photogrammetry with oblique images: First analysis on data acquisition and processing. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B1:835–842.

[Alliez et al., 2019] Alliez, P., Jamin, C., Rineau, L., Tayeb, S., Tournois, J., and Yvinec, M. (2019). 3D mesh generation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0 edition.

[Amanatides and Woo, 1987] Amanatides, J. and Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10.

[Aronszajn, 1950] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404.

[Attard et al., 2018] Attard, L., Debono, C. J., Valentino, G., and Di Castro, M. (2018). Tunnel inspection using photogrammetric techniques and image processing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 144:180 – 188.

[Bakker and Lane, 2017] Bakker, M. and Lane, S. N. (2017). Archival photogrammetric analysis of river–floodplain systems using structure from motion (sfm) methods. *Earth Surface Processes and Landforms*, 42(8):1274–1286.

[Banta et al., 2000] Banta, J. E., Wong, L. M., Dumont, C., and Abidi, M. A. (2000). A next-best-view system for autonomous 3-d object reconstruction. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 30(5):589–598.

[Bay et al., 2006] Bay, H., Tuytelaars, T., and Van Gool, L. (2006). Surf: Speeded up robust features. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Berger et al., 2013] Berger, M., Levine, J. A., Nonato, L. G., Taubin, G., and Silva, C. T. (2013). A benchmark for surface reconstruction. *ACM Trans. Graph.*, 32(2):20:1–20:17.

[Berger et al., 2014] Berger, M., Tagliasacchi, A., Seversky, L., Alliez, P., Levine, J., Sharf, A., and Silva, C. (2014). State of the art in surface reconstruction from point clouds. In *Eurographics 2014 - State of the Art Reports*, volume 1 of *EUROGRAPHICS star report*, pages 161–185, Strasbourg, France.

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-D shapes. In Schenker, P. S., editor, *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586 – 606. International Society for Optics and Photonics, SPIE.

[Betts, 2010] Betts, J. T. (2010). *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming: Second Edition*. Advances in Design and Control. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104).

[Bhadrakom and Chaiyasarn, 2016] Bhadrakom, B. and Chaiyasarn, K. (2016). As-built 3d modeling based on structure from motion for deformation assessment of historical buildings. *International Journal of Geomate*, 11:2378–2384.

[Bircher et al., 2017] Bircher, A., Alexis, K., Schwesinger, U., Omari, S., Burri, M., and Siegwart, R. (2017). An incremental sampling-based approach to inspection planning: the rapidly exploring random tree of trees. *Robotica*, 35(6):1327–1340.

[Bircher et al., 2016] Bircher, A., Kamel, M., Alexis, K., Burri, M., Oettershagen, P., Omari, S., Mantel, T., and Siegwart, R. (2016). Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. *Autonomous Robots*, 40(6):1059–1078.

[Bircher et al., 2018] Bircher, A., Kamel, M., Alexis, K., Oleynikova, H., and Siegwart, R. (2018). Receding horizon path planning for 3d exploration and surface inspection. *Autonomous Robots*, 42(2):291–306.

[Blanco and Rai, 2014] Blanco, J. L. and Rai, P. K. (2014). nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. `https://github.com/jlblancoc/nanoflann`.

[Blinn, 1977] Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198.

[Bolognesi et al., 2014] Bolognesi, M., Furini, A., Russo, V., Pellegrinelli, A., and Russo, P. (2014). Accuracy of cultural heritage 3d models by rpas and terrestrial photogrammetry. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5:113–119.

[Caccamo et al., 2016] Caccamo, S., Bekiroglu, Y., Ek, C. H., and Kragic, D. (2016). Active exploration using gaussian random fields and gaussian process implicit surfaces. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 582–589.

[Carr et al., 2001] Carr, J. C., Beatson, R. K., Cherrie, J. B., Mitchell, T. J., Fright, W. R., McCallum, B. C., and Evans, T. R. (2001). Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 67–76, New York, NY, USA. ACM.

[Carraro et al., 2019] Carraro, F., Monego, M., Callegaro, C., Mazzariol, A., Perticarini, M., Menin, A., Achilli, V., Bonetto, J., and Giordano, A. (2019). The 3d survey of the roman bridge of san lorenzo in padova (italy): A comparison between sfm and tls methodologies applied to the arch structure. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W15:255–262.

[CC BY 4.0, 2013] CC BY 4.0 (2013). Creative commons attribution. `https://creativecommons.org/licenses/by/4.0/`. Last accessed on Dec 17, 2019.

[CC BY-NC 4.0, 2013] CC BY-NC 4.0 (2013). Creative commons attribution-noncommercial. `https://creativecommons.org/licenses/by-nc/4.0/`. Last accessed on Dec 17, 2019.

[Chen et al., 2019] Chen, S., Laefer, D., Mangina, E., Zolanvari, I., and Byrne, J. (2019). Uav bridge inspection through evaluated 3d reconstructions. *Journal of Bridge Engineering*, 24.

[Chen et al., 2011] Chen, S., Li, Y., and Kwok, N. M. (2011). Active vision in robotic systems: A survey of recent developments. *The International Journal of Robotics Research*, 30(11):1343–1377.

[Chiabrando et al., 2015] Chiabrando, F., Donadio, E., and Rinaudo, F. (2015). Sfm for orthophoto to generation: A winning approach for cultural heritage knowledge. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W7:91–98.

[CloudCompare, 2019] CloudCompare (2019). CloudCompare (Version 2.10.1 Zephyrus). `http://www.cloudcompare.org/` (Software).

[Cover et al., 2013] Cover, H., Choudhury, S., Scherer, S., and Singh, S. (2013). Sparse tangential network (spartan): Motion planning for micro aerial vehicles. In *2013 IEEE International Conference on Robotics and Automation*, pages 2820–2825.

[Delmerico et al., 2018] Delmerico, J., Isler, S., Sabzevari, R., and Scaramuzza, D. (2018). A comparison of volumetric information gain metrics for active 3d object reconstruction. *Autonomous Robots*, 42(2):197–208.

[Dragiev et al., 2011] Dragiev, S., Toussaint, M., and Gienger, M. (2011). Gaussian process implicit surfaces for shape estimation and grasping. In *2011 IEEE International Conference on Robotics and Automation*, pages 2845–2850.

[Englot and Hover, 2013] Englot, B. and Hover, F. S. (2013). Sampling-based coverage path planning for inspection of complex structures. In *Proceedings of the Twenty-Second International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'12, pages 29–37. AAAI Press.

[Frey, 2001] Frey, P. (2001). MEDIT : An interactive Mesh visualization Software. Technical Report RT-0253, INRIA.

[Galceran et al., 2014] Galceran, E., Campos, R., Palomeras, N., Carreras, M., and Ridao, P. (2014). Coverage Path Planning with Realtime Replanning for Inspection of 3D Underwater Structures. In *ICRA*, pages 6586–6591. IEEE.

[Galceran and Carreras, 2013] Galceran, E. and Carreras, M. (2013). A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276.

[Glaser et al., 2009] Glaser, S., Woodall, W., and Haschke, R. (2009). Xacro. `http://wiki.ros.org/xacro`. Last accessed on Jan 9, 2019.

[Glauser, 2017] Glauser, J. M. (2017). 3D Model of Holbeach Cemetery Chapel, Lincolnshire, UK. `https://sketchfab.com/3d-models/11d45085a31d422cb2b28eb7328d989a`. Last accessed on Dec 17, 2019.

[Global Digital Heritage, 2019] Global Digital Heritage (2019). 3D Model of the Roman Temple of Evora, Alentejo, Portugal. `https://sketchfab.com/3d-models/935f17a3824d49f7b2505a0686450d51`. Last accessed on Dec 17, 2019.

[Green et al., 2014] Green, S., Bevan, A., and Shapland, M. (2014). A comparative assessment of structure from motion methods for archaeological research. *Journal of Archaeological Science*, 46:173 – 181.

[Guennebaud et al., 2010] Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. `http://eigen.tuxfamily.org`.

[Hadsell et al., 2010] Hadsell, R., Bagnell, J. A., Huber, D., and Hebert, M. (2010). Space-carving kernels for accurate rough terrain estimation. *The International Journal of Robotics Research*, 29(8):981–996.

[Hallermann et al., 2014] Hallermann, N., Morgenthal, G., and Rodehorst, V. (2014). Vision-based deformation monitoring of large scale structures using unmanned aerial systems. In *IABSE Symposium Report*, volume 102, pages 2852–2859.

[Hart, 1996] Hart, J. (1996). Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527 – 545.

[Hermann et al., 2014] Hermann, A., Drews, F., Bauer, J., Klemm, S., Roennau, A., and Dillmann, R. (2014). Unified gpu voxel collision detection for mobile manipulation planning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4154–4160.

[Hess et al., 2016] Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278.

[Hollinger et al., 2012] Hollinger, G. A., Englot, B., Hover, F., Mitra, U., and Sukhatme, G. S. (2012). Uncertainty-driven view planning for underwater inspection. In *2012 IEEE International Conference on Robotics and Automation*, pages 4884–4891.

[Hornung et al., 2013] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206.

[Huang and Hermans, 2019] Huang, K. and Hermans, T. (2019). Building 3d object models during manipulation by reconstruction-aware trajectory optimization.

[Jadidi et al., 2014] Jadidi, M. G., Miro, J. V., Valencia, R., and Andrade-Cetto, J. (2014). Exploration on continuous gaussian process frontier maps. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6077–6082.

[Jahanshahi et al., 2013] Jahanshahi, M., Masri, S., Padgett, C., and Sukhatme, G. (2013). An innovative methodology for detection and quantification of cracks through incorporation of depth perception. *Machine Vision and Applications*, 24.

[Jidling et al., 2017] Jidling, C., Wahlström, N., Wills, A., and Schön, T. B. (2017). Linearly constrained gaussian processes. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 1215–1224. Curran Associates, Inc.

[Karaman and Frazzoli, 2011] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894.

[Kimeldorf and Wahba, 1971] Kimeldorf, G. and Wahba, G. (1971). *Some Results on Tchebycheffian Spline Functions*. MRC technical summary report. University of Wisconsin, United States Army, Mathematics Research Center.

[Klingensmith et al., 2015] Klingensmith, M., Dryanovski, I., Srinivasa, S., and Xiao, J. (2015). Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device using Spatially Hashed Signed Distance Fields. In *Robotics: Science and Systems XI*. Robotics: Science and Systems Foundation.

[Koenig and Howard, 2004] Koenig, N. P. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3.

[Kopp, 2008] Kopp, J. (2008). Efficient numerical diagonalization of hermitian 3 x 3 matrices. *International Journal of Modern Physics C*, 19(03):523–548.

[Körkel, 2002] Körkel, S. (2002). *Numerische Methoden für optimale Versuchsplanungsprobleme bei nichtlinearen DAE-Modellen*. PhD thesis, Heidelberg University.

[Krause et al., 2008] Krause, A., Singh, A., and Guestrin, C. (2008). Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284.

[Kriegel et al., 2015] Kriegel, S., Rink, C., Bodenmüller, T., and Suppa, M. (2015). Efficient next-best-scan planning for autonomous 3d surface reconstruction of unknown objects. *Journal of Real-Time Image Processing*, 10(4):611–631.

[Kuwata et al., 2009] Kuwata, Y., Theo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J. P. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118.

[Lattanzi and Miller, 2014] Lattanzi, D. and Miller, G. (2014). 3d scene reconstruction for robotic bridge inspection. *Journal of Infrastructure Systems*, 21:04014041.

[Lindner et al., 2019] Lindner, S., Garbe, C., and Mombaur, K. (2019). Optimization based multi-view coverage path planning for autonomous structure from motion recordings. *IEEE Robotics and Automation Letters*, 4(4):3278–3285.

[Lo Brutto et al., 2014] Lo Brutto, M., Garraffa, A., and Meli, P. (2014). Uav platforms for cultural heritage survey: First results. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-5.

[Lo Brutto and Meli, 2012] Lo Brutto, M. and Meli, P. (2012). Computer vision tools for 3d modeling in archaeology. *International Journal of Heritage in the Digital Era*, 1:1–6.

[López et al., 2016] López, J. B., Jiménez, G. A., Romero, M. S., García, E. A., Martín, S. F., Medina, A. L., and Guerrero, J. E. (2016). 3d modelling in archaeology: The application of structure from motion methods to the study of the megalithic necropolis of panoria (granada, spain). *Journal of Archaeological Science: Reports*, 10:495–506.

[Lowe, 1999] Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2.

[Lucieer et al., 2014a] Lucieer, A., de Jong, S. M., and Turner, D. (2014a). Mapping landslide displacements using structure from motion (sfm) and image correlation of multi-temporal uav photography. *Progress in Physical Geography: Earth and Environment*, 38(1):97–116.

[Lucieer et al., 2014b] Lucieer, A., Turner, D., King, D. H., and Robinson, S. A. (2014b). Using an unmanned aerial vehicle (uav) to capture micro-topography of antarctic moss beds. *International Journal of Applied Earth Observation and Geoinformation*, 27:53 – 62. Special Issue on Polar Remote Sensing 2013.

[Mancini et al., 2013] Mancini, F., Dubbini, M., Gattelli, M., Stecchi, F., Fabbri, S., and Gabbianelli, G. (2013). Using unmanned aerial vehicles (uav) for high-resolution reconstruction of topography: The structure from motion approach on coastal environments. *Remote Sensing*, 5(12):6880–6898.

[Marchal, 2018] Marchal, G. (2018). 3D Model of a Tyche Sculpture, Musee du Cinquantenaire (Brussels, Belgium). `https://sketchfab.com/3d-models/f27fa510181946f4924080e3ccda3946`. Last accessed on Dec 17, 2019.

[Martens et al., 2017] Martens, W., Poffet, Y., Soria, P. R., Fitch, R., and Sukkarieh, S. (2017). Geometric priors for gaussian process implicit surfaces. *IEEE Robotics and Automation Letters*, 2(2):373–380.

[Min, 2019] Min, P. (2004 - 2019). binvox. `http://www.patrickmin.com/binvox`. Last accessed on Jan 9, 2019.

[Moravec and Elfes, 1985] Moravec, H. P. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121.

[Newcombe et al., 2011] Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011). DTAM: Dense tracking and mapping in real-time. In *ICCV*, pages 2320–2327. IEEE.

[Nießner et al., 2013] Nießner, M., Zollhöfer, M., Izadi, S., and Stamminger, M. (2013). Real-time 3D Reconstruction at Scale using Voxel Hashing. *ACM Transactions on Graphics*, 32(6):169:1–169:11.

[Nooruddin and Turk, 2003] Nooruddin, F. S. and Turk, G. (2003). Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):191–205.

[O'callaghan and Ramos, 2012] O'callaghan, S. T. and Ramos, F. T. (2012). Gaussian process occupancy maps. *The International Journal of Robotics Research*, 31(1):42–62.

[Ottenhaus et al., 2016] Ottenhaus, S., Miller, M., Schiebener, D., Vahrenkamp, N., and Asfour, T. (2016). Local implicit surface estimation for haptic exploration. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 850–856.

[Papadopoulos et al., 2013] Papadopoulos, G., Kurniawati, H., and Patrikalakis, N. M. (2013). Asymptotically Optimal Inspection Planning using Systems with Differential Constraints. In *ICRA*, pages 4126–4133. IEEE.

[Papoulis et al., 2002] Papoulis, A., Pillai, S., and Pillai, S. (2002). *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill electrical and electronic engineering series. McGraw-Hill.

[Peng Cheng et al., 2008] Peng Cheng, Keller, J., and Kumar, V. (2008). Time-Optimal UAV Trajectory Planning for 3D Urban Structure Coverage. In *IROS*, pages 2750–2757. IEEE.

[Pietroni et al., 2010] Pietroni, N., Tarini, M., and Cignoni, P. (2010). Almost isometric mesh parameterization through abstract domains. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):621–635.

[Pito, 1999] Pito, R. (1999). A solution to the next best view problem for automated surface acquisition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1016–1030.

[Pizarro et al., 2004] Pizarro, O., Eustice, R., and Singh, H. (2004). Large area 3d reconstructions from underwater surveys. In *Oceans '04 MTS/IEEE Techno-Ocean '04 (IEEE Cat. No.04CH37600)*, volume 2, pages 678–687 Vol.2.

[Quigley et al., 2009] Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Y. Ng, A. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

[Ramos and Ott, 2016] Ramos, F. and Ott, L. (2016). Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *The International Journal of Robotics Research*, 35(14):1717–1730.

[Rasmussen and Williams, 2006] Rasmussen, C. and Williams, C. (2006). *Gaussian Processes for Machine Learning*. Adaptative computation and machine learning series. University Press Group Limited.

[Scott et al., 2003] Scott, W. R., Roth, G., and Rivest, J.-F. (2003). View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv.*, 35(1):64–96.

[Seitz, 2012] Seitz, C. (2012). Vom foto zum 3d-modell: Open-source-nahbereichsphotogrammetrie im einsatz für die archäologie. Master's thesis, Heidelberg University.

[Seitz and Altenbach, 2011] Seitz, C. and Altenbach, H. (2011). Project archeye–the quadrocopter as the archaeologist's eye. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci*, 38(1).

[Shaffer and Garland, 2001] Shaffer, E. and Garland, M. (2001). Efficient adaptive simplification of massive meshes. In *Proceedings of the Conference on Visualization '01*, VIS '01, page 127–134, USA. IEEE Computer Society.

[Singh and Narayanan, 2010] Singh, J. M. and Narayanan, P. J. (2010). Real-time ray tracing of implicit surfaces on the gpu. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):261–272.

[Skarlatos and Kiparissi, 2012] Skarlatos, D. and Kiparissi, S. (2012). Comparison of laser scanning, photogrammetry and sfm-mvs pipeline applied in structures and artificial surfaces. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, I-3:299–304.

[Solak et al., 2003] Solak, E., Murray-smith, R., Leithead, W. E., Leith, D. J., and Rasmussen, C. E. (2003). Derivative observations in gaussian process models of dynamic systems. In Becker, S., Thrun, S., and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 1057–1064. MIT Press.

[Spackman and Willis, 1991] Spackman, J. and Willis, P. (1991). The smart navigation of a ray through an oct-tree. *Computers & Graphics*, 15(2):185 – 194.

[Storlazzi et al., 2016] Storlazzi, C., Dartnell, P., Hatcher, G., and Gibbs, A. (2016). End of the chain? rugosity and fine-scale bathymetry from existing underwater digital imagery using structure-from-motion (sfm) technology. *Coral Reefs*, 35:889–894.

[The CGAL Project, 2019] The CGAL Project (2019). *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0 edition.

[Torres et al., 2016] Torres, M., Pelta, D. A., Verdegay, J. L., and Torres, J. C. (2016). Coverage path planning with unmanned aerial vehicles for 3d terrain reconstruction. *Expert Systems with Applications*, 55:441 – 451.

[Valenti and Paternò, 2019] Valenti, R. and Paternò, E. (2019). A comparison between tls and uav technologies for historical investigation. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W9:739–745.

[Vasquez-Gomez et al., 2014] Vasquez-Gomez, J. I., Sucar, L. E., Murrieta-Cid, R., and Lopez-Damian, E. (2014). Volumetric next-best-view planning for 3d object reconstruction with positioning error. *International Journal of Advanced Robotic Systems*, 11(10):159.

[Vasudevan et al., 2009] Vasudevan, S., Ramos, F., Nettleton, E., and Durrant-Whyte, H. (2009). Gaussian process modeling of large-scale terrain. *Journal of Field Robotics*, 26(10):812–840.

[Wahba, 1990] Wahba, G. (1990). *Spline Models for Observational Data*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics.

[Webb and van den Berg, 2013] Webb, D. J. and van den Berg, J. (2013). Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061.

[Weyl, 1912] Weyl, H. (1912). Das asymptotische verteilungsgesetz der eigenwerte linearer partieller differentialgleichungen (mit einer anwendung auf die theorie der hohlraumstrahlung). *Mathematische Annalen*, 71(4):441–479.

[Williams and Fitzgibbon, 2007] Williams, O. and Fitzgibbon, A. (2007). Gaussian process implicit surfaces. In *Gaussian Processes in Practice*.

[Wu et al., 2017] Wu, A., Aoi, M. C., and Pillow, J. W. (2017). Exploiting gradients and hessians in bayesian optimization and bayesian quadrature.

[Xu et al., 2014] Xu, Z., Wu, L., Shen, Y., Li, F., Wang, Q., and Wang, R. (2014). Tridimensional reconstruction applied to cultural heritage with the use of camera-equipped uav and terrestrial laser scanner. *Remote Sensing*, 6(11):10413–10434.

[Yakoubi and Laskri, 2016] Yakoubi, M. A. and Laskri, M. T. (2016). The path planning of cleaner robot for coverage region using genetic algorithms. *Journal of Innovation in Digital Ecosystems*, 3(1):37 – 43. Special issue on Pattern Analysis and Intelligent Systems – With revised selected papers of the PAIS conference.

[Zhang et al., 2016] Zhang, R., Schneider, D., and Strauß, B. (2016). Generation and comparison of tls and sfm based 3d models of solid shapes in hydromechanic research. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B5:925–929.