FURTHERING USABILITY AND EFFICIENCY OF MASSIVELY PARALLEL GAUSSIAN PROCESS REGRESSION AS APPLIED TO MULTI-BEAM SONAR DATA

Phillip M. Parisi
University of Rhode Island, phillipmparisi@gmail.com

Follow this and additional works at: https://digitalcommons.uri.edu/theses

Recommended Citation
https://digitalcommons.uri.edu/theses/2322

This Thesis is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Master's Theses by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons-group@uri.edu.
FURTHERING USABILITY AND EFFICIENCY OF MASSIVELY PARALLEL GAUSSIAN PROCESS REGRESSION AS APPLIED TO MULTI-BEAM SONAR DATA

BY

PHILLIP M. PARISI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN
OCEAN ENGINEERING

UNIVERSITY OF RHODE ISLAND
2023
MASTER OF SCIENCE THESIS
OF
PHILLIP M. PARISI

APPROVED:
Thesis Committee:

Major Professor    Chris Roman
Mingxi Zhou
Peter Swaszek
Brenton DeBoef
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2023
ABSTRACT

Massively Parallel Gaussian Process Regression (MP-GPR) is a parallelized programmatic implementation of a machine learning algorithm used to process multi-beam sonar data and generate 2.5D terrain maps in close to real-time. This thesis furthered the capabilities of MP-GPR in two ways. The first introduced data reduction techniques to reduce the quantity of data inputted to the MP-GPR algorithm for computational efficiency. Various approaches were analyzed to determine an optimal data minimization strategy. The second informed the variances of regression estimates by considering beam angle and range uncertainty of the multi-beam sonar data being analyzed. In tandem, the presented methodologies decreased the GPR algorithm run-time and provided an improved uncertainty model. Trials were run using a Graphical Processing Unit (GPU) to process data from a Robot Operating System (ROS) data bag with C++. The data was obtained from a WASSP multi-beam sonar survey of the Wiggles Bank on St. Mary’s River in Georgia, USA. The results showed that there were no significant gains in accuracy by using more complex downsampling algorithms and instead using simple method (systematic, average, hybrid) was adequate.
ACKNOWLEDGMENTS

This thesis would not be possible without unique insights and guidance from my advisor, Chris Roman, and committee members Mingxi Zhou and Peter Swaszek. This thesis furthers Kristopher Krasnosky’s dissertation; Kris’ time and effort were crucial to success. Dr. Tzu-Ting Chen provided a valuable perspective on sonar uncertainty, and I am grateful for her advice. Last but not least, thank you to my fellow lab mates for keeping me focused and laughing along the way.
PREFACE

This thesis is presented in manuscript form in preparation for submission to IEEE Oceans.
# TABLE OF CONTENTS

**ABSTRACT** .................................................. ii

**ACKNOWLEDGMENTS** ....................................... iii

**PREFACE** .................................................. iv

**TABLE OF CONTENTS** .................................... v

**MANUSCRIPT**

1 Usability and Efficiency of Massively Parallel Gaussian Process Regression as Applied to Multi-Beam Sonar Data .... 1

1.1 Introduction .............................................. 2

1.1.1 Gaussian Process Regression ....................... 2

1.1.2 Massively Parallel Gaussian Process Regression .... 4

1.1.3 Approximate Methods .............................. 5

1.2 Implementation ......................................... 6

1.2.1 System Overview ................................... 9

1.2.2 Downsampling .................................... 10

1.2.3 Hyperparameter Training ......................... 14

1.2.4 Incorporating Uncertainty ...................... 14

1.3 Results ............................................... 16

1.4 Discussion ........................................... 20

1.4.1 Future Work ..................................... 22

1.5 Conclusion ........................................... 23

**LIST OF REFERENCES** ................................... 25
APPENDIX

A Notation and Relevant Equations ......................... 27
B Significant Parameters ................................. 30
C Downsamped Swaths .................................. 33
D Uncertainty Tiles ................................. 36
MANUSCRIPT 1

Usability and Efficiency of Massively Parallel Gaussian Process Regression as Applied to Multi-Beam Sonar Data

by

Phillip M. Parisi\textsuperscript{1}, Chris Roman\textsuperscript{1,2}, Kristopher Krasnosky\textsuperscript{3}

\textit{In preparation for submission to IEEE Oceans}

\textsuperscript{1} University of Rhode Island, Department of Ocean Engineering
\textsuperscript{2} University of Rhode Island, Graduate School of Oceanography
\textsuperscript{3} University of South Florida, Center for Ocean Mapping and Innovative Technologies
1.1 Introduction

Multi-beam echo sounder (MBES) surveys repeatedly ping the seafloor using acoustic transducers to determine the water depth. Each of these returns can contain upwards of one hundred data points, and surveys that ping the seafloor for hours will result in millions of data points. Forming bathymetric models of the seafloor from these data can be viewed as a regression problem. Regression, determining the relationship between a dependent variable and one or more independent variables, can be solved using a Gaussian Process Regression (GPR) [1]. GPRs can be used to generate water depth predictions over a dense grid of points along the seafloor and calculate the associated uncertainties to provide a level of confidence in the predictions. Modern robotic algorithms that utilize probabilistic approaches benefit from GPR’s uncertainty estimates in decision-making and estimation tasks. GPR has been applied to underwater terrain modeling [2], terrain aided navigation (TAN) [3], and simultaneous localization and mapping (SLAM) [4, 5]. It is, however, inefficient to compute a GPR on a large dataset; thus, utilizing a graphical processing unit (GPU) to reduce compute time has been explored [6, 7]. Krasnosky’s Massively Parallel Gaussian Process Regression (MP-GPR) algorithm was shown to improve the resolution of seafloor terrain models made from MBES data and generated an uncertainty model which other standard mapping methods (splines, gridded, polynomial, etc.) lack [5]. This work advances Krasnosky’s research by considering additional computational efficiencies and uncertainty from multi-beam sonar to improve bathymetric model accuracy and reduce compute time during live processing.

1.1.1 Gaussian Process Regression

GPR is a supervised machine learning approach for determining input-output mappings from training data. The basic formulation of the problem uses empirical
data points $D$ from a set of $N$ observations, known as *training data*, consisting of inputs $x$ (independent variables) and the corresponding targets $y$ (dependent variable). Each observation $x_i$ of $d$ dimensions maps to a scalar target $y_i$, which is a noisy realization of the underlying function of interest $f$. That is,

$$D = \{(x_i, y_i)|i = 1, \ldots, N\}$$

$$x \in \mathbb{R}^{N \times d}, y \in \mathbb{R}^N, y_i = f(x_i) + \epsilon,$$

where $\epsilon$ denotes error and bold symbols indicate matrix quantities. The error comes from sensor observation and is often assumed be normally distributed with zero mean, such that $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. When applied to seafloor mapping, $x_i = (\text{northing, easting})$ are the position variables, $y_i = \text{depth}$ is the corresponding depth at that location, and $f$ is the ‘function’ that describes the 2.5-dimensional seafloor surface. During the training portion of the problem, $D$ is used to learn the underlying function $f$. During the inference step, $f$ is used to predict depths $y_* \in \mathbb{R}^{A \times 1}$ over a new set of geographically dense inputs $x_* \in \mathbb{R}^{A \times d}$ to form a point cloud map. A given prediction value $y_*$ is fully described by a Gaussian distribution $y_* \sim \mathcal{N}(\mu_*, \sigma_*^2)$ with a mean depth value $\mu_*$ and variance $\sigma_*$ (uncertainty).

GPR is a non-parametric regression model that utilizes a kernel to describe the relevant length scale and process noise of the function being modeled. A commonly used kernel is the squared exponential (SE) kernel,

$$k(x, x') = \sigma^2_f \exp\left(-\frac{|x - x'|^2}{2l^2}\right),$$

(1)

where $|x - x'|$ denotes the magnitude of the difference between vectors $x$ and $x'$ (Euclidean distance), and $\theta = \{l, \sigma_f^2\}$ are hyperparameters determined during model training. The set of GPR equations are provided in the Appendix.

GPR’s major advantage is providing uncertainty estimates at all prediction points. Most modern regression methods lack these estimates and hence fail to
inform users of the confidence bounds at prediction points, whereas GPR does.

### 1.1.2 Massively Parallel Gaussian Process Regression

Krasnosky’s Massively Parallel GPR (MP-GPR) is an extension of GPR that creates an updateable 2.5D model of the seafloor using multi-beam sonar data collected in a push-broom manner. The outputted model is composed of spatially-organized dense point clouds with uncertainties that can be used as terrain models for SLAM and other applications.

The underlying computation of a GPR suffers from mathematical inefficiencies during training and inference on large datasets, primarily due to the required inverse of an $N \times N$ covariance matrix which faces cubic time complexity, $O(N^3)$. This renders the basic implementation of GPR intractable for large data volumes [1]. Multi-beam sonar surveys often generate millions of data points and hence cannot be practically computed using a standard GPR.

MP-GPR relies on a parallel computational framework to speed up the computation. Its updateable GPR model leverages parallel processor cores on a GPU, iterative matrix updates that incorporate new bathymetric data when collected, sparse matrix representation, and an exactly sparse approximation to the SE kernel,

$$
\begin{align*}
    k(\mathbf{x}, \mathbf{x}') &= \begin{cases} 
        \sigma_f^2 \left[ \frac{2 + \cos(2\pi \frac{d'}{l})}{3} (1 - \frac{d'}{l}) + \frac{1}{2\pi} \sin(2\pi \frac{d'}{l}) \right] & \text{if } d' < l \\
        0 & \text{if } d' \geq l
    \end{cases}
\end{align*}
$$

(2)

where $d' = |\mathbf{x} - \mathbf{x}'|$. This kernel was devised by Melkumyan and Ramos [8] and closely approximates the actual squared exponential kernel with the added benefit of setting covariances to zero beyond the trained length scale while preventing discontinuities. More zero elements in covariance matrices improve the performance of the sparse matrix representations that MP-GPR employs.

These modifications allow for a tractable, online, and ‘exact’ GPR solution
that generates an in-situ terrain model with uncertainties. MP-GPR can operate in real-time but requires significant computing power (e.g. a NVIDIA 2080ti GPU with 4352 CUDA cores). Modern mobile platforms, such as autonomous underwater/surface vehicles (AUVs/ASVs), are unlikely to possess high-end GPUs due to their significant power consumption. With ever increasing sensor resolution and acquisition rates, there is a need to further reduce the GPR compute time to enable real-time performance on a wide-range of hardware, such as the smaller NVIDIA Jetson NX Xavier, to make MP-GPR broadly applicable and relatively platform agnostic.

1.1.3 Approximate Methods

GPR and MP-GPR calculate an exact solution, that is, a solution that incorporates every data point in $D$ for model training and inference. Alternatively, approximate methods have been developed to reduce compute time and calculate an approximate solution to GPR. There are many families of approximate methods: subset of regressors [9], subset of data [10], projected processes [11], and modern methods such as Sparse Variational Gaussian Processes [12].

Most approximate methods present alternative formulations to the underlying GPR equations and diverge from MP-GPR’s implementation. However, data-centric methods incorporate techniques to replace training data $D$ with a fabricated dataset $D_R$ containing $M$ points, where $M \ll N$ reduces the computational burden. $D_R$ can be composed of a subset from the training data $D$ or of newly generated pseudo-data points that may not be present in $D$, such that $D_R \not\subset D$. $D_R$ then serves as the training data for the model in lieu of $D$, reducing compute time and model accuracy. Determining $D_R$ from $D$ will be broadly referred to as downsampling. Downsampling data as an input to MP-GPR results in a faster online GPR solution.
In the standard GPR formulation, the algorithm time complexity when using $M \ll N$ points would be $O(M^3)$. MP-GPR avoids the $N \times N$ matrix inversion by utilizing Cholesky decompositions that are updated throughout a survey. Additionally, usage of a GPU distributes computes over numerous cores for parallel processing. Occasional memory transfers between the CPU and GPU incurs additional costs. MP-GPR time complexity is an improvement over cubic complexity, but is not well-defined due to the complex nature of the algorithm.

1.2 Implementation

MP-GPR was implemented using the Robotic Operating System (ROS) architecture [13], C++ code, and CUDA [14] code for GPU operations. The work described in this thesis added to Krasnosky’s original code base and was tested on a pre-recorded ROSbag to simulate real-time operation. The machine used had the following specifications: System76 Oryx Pro, Ubuntu 20.04 (64-bit), 8GB RTX 3070 GPU w/ 5120 CUDA cores, 4.6GHz i7 processor.

The optimal downsampling approach that minimized runtime and error was to be determined. To do this, each downsampling method was tested by running a simulated live bathymetry survey. The process was:

- a sonar ping was obtained
- outliers were removed from the ping
- the ping was downsampling per a given method and downsample percent
- the ping was spatially sorted and uncertainty values were calculated at each point
- pings were added to spatial ‘blocks’ for efficient processing
- once a block was full, GPR inference was performed to create a prediction tile (dense set of prediction points)
- many prediction tiles were aligned to make a map
Figure 1. Hundreds of individual sonar soundings (red dots) form a ping.

Figure 2. Many consecutive sonar pings form a swath.
Figure 3. A group of GPR inference tiles with local depth heatmaps are calculated and aligned.

After a run was performed, a point cloud model of the seafloor remained. The time to run each method was recorded, and the accuracy of each model was computed through comparison with the exact MP-GPR model (all data points were used).

Figure 4. Snapshot of ROS Rviz simulation running MP-GPR on a raw sonar swath (left) and with a 10% random downsamplend swath (right). Individual soundings (data points) are shown in red. GPR solution tiles are shown in color.

The dataset used in this study was collected with a WASSP sonar system in
the St. Mary’s River (Georgia, USA). The WASSP Multibeam Sonar retrieved a maximum of 256 points per ping at 5-20Hz depending on depth. The river was well mixed and the measured sound speed profile was nearly constant with depth, which alleviated the need for ray tracing corrections. The mapping system and survey details are fully explained by Krasnosky, et. al [15].

1.2.1 System Overview

The system ingested a live sonar data-stream one ping at a time. A navigation solution paired to each sonar ping accounted for dynamic effects and positions the data in the correct mapping frame. Sonar pings were passed to a ‘blockulator.’ The blockulator applied an outlier filter to remove erroneous soundings, downsampled the data, and queued it into spatial ‘blocks.’ Once a local neighborhood of blocks was full, GPR inference was performed to generate a large, dense prediction tile (2.5D pointcloud). These tiles were added to the seafloor model as they became available during the survey [5].

Each downsampling method was implemented within a blockulator. The downsampling methods analyzed were:

- a *Uniform Random* method that selected points using a uniform random distribution,
- a *Systematic Random* method that selected every $j$th point (decimation),
- a *Random Hybrid* method that evenly combined systematic and uniform random selection,
- a *Point Averaging* method that averaged dense local points into a set of sparse points,
- a *Dissimilar Neighbor* method that ranked points based on a ratio of nearest-neighbor distances to select ‘interesting’ points,
- and a *K-means* method that clustered points into groups based on similarity.
1.2.2 Downsampling

The downsampling methods were cheap to compute and implement on raw bathymetry data as it streamed in during a multi-beam survey. Each ping of sonar data was downsampled when received ($D$ with $N$ points into $D_R$ with $M$ points), agnostic to other processes and data collected. Downsampled pings were queued into blocks which were processed periodically and added to the GPR solution.

When downsampling, it was of utmost importance that no outliers were included in $D_R$. The WASSP sonar had a built-in filter to remove significantly irregular points, yet a few poor points persisted. Thus, an additional outlier filter was added before downsampling. A given point $p$ was considered an outlier if its depth value exceeded 2.0x the standard deviation of the neighboring points’ mean (excluding $p$). This was especially relevant for the dissimilar neighbor method, which was biased towards outlier inclusion. Thus, outliers were removed from the set, and the clean $D_R$ was passed to a downsampling algorithm.

Uniform Random

Uniform random sampling is a common and fast method for downsampling. For each sonar ping, a new random seed was used. Uniform random sampling is often a benchmark for evaluating the performance of other methods.

Systematic Random

Systematic random sampling (‘decimation’) sorted the data by the spatial $y$-coordinate (left to right along a ping) and selected every $j$th point along that dimension. In MATLAB, the indices were expressed $x_o : j : x_{end}$. To avoid along track selection bias, the first index of each ping $x_o$ was uniform randomly selected from 0 to $j - 1$. 
Figure 5. 20% systematic downsampling with biased (left) and unbiased (right) approaches

**Hybrid Random**

The hybrid random method recognized the combined value of systematic samples to ensure a spatial spread and random sampling to select unique points. Thus, half of $D_R$ was selected spatially with systematic sampling. Then, the second half was selected from the remaining points using a uniform random distribution.

**Point Averaging**

Point averaging calculated the average of local data points to create a new representative data point. This was done using a moving window across each ping.

**Dissimilar Neighbor**

The k-nearest-neighbor (KNN) algorithm is commonly used in machine learning for classification and regression. KNN uses a difference metric to determine if a new point is more similar to one group than another. The base assumption is that points with small nearest neighbor distances are more likely to be similar to those neighbors [16].

In the spirit of seeking ‘unique’ points to represent the distribution of a sonar ping, dissimilar neighbor ("neighbor") checked the two neighboring points of $p_i$ along a ping, $p_{i-1}$ and $p_{i+1}$, and calculated Euclidean distances $d_p$. Points that
had the smallest distance ratio \( d_r = \min(d_p)/\max(d_p) \) became inclusion points. A point with a ratio close to 1 was evenly-spaced with its neighbors and assumed to be uninteresting, and vice versa. A ratio was used to scale distances as the spacing of points varied along a ping.

**K-means**

K-means is a popular clustering algorithm that splits a group of samples into k-clusters (subgroups) based on their similarity. The inclusion points chosen by uniform random sampling were used as the initial K-means centroids to encourage unique point distributions. To combat performance issues [17] and keep processing time minimal, the number of algorithm iterations was set to 20. Then, the centroids of each cluster served as the new data points in \( D_R \).

**Downsampling Implementation**

Each method had one adjustable parameter, *downsample percent* \( s \), which was the percent of points that remained after downsampling a given ping (e.g., if \( s = 0.2 \), a 200-point ping was downsampled to 40 points). Using a large value of \( s \) included fewer data, decreased computation time, and decreased solution accuracy. A small value of \( s \) drastically reduced the amount of data, rapidly decreased computation time, and further decreased solution accuracy (in theory). \( s \) was systematically varied for each method to determine the optimal value that minimized compute time and maximized solution accuracy.

For consistency, each method was applied to the same sonar swath in Figure 7. The swath exhibited desirable characteristics, including regions with and without overlap, a turn with high point density, and considerable relative depth variation (between 6m and 12m). Images of downsampled swaths for each method can be found in the Appendix.
Figure 6. Inclusions ($D_R$) at 20% downsampling for each method. Data had 100 soundings and each method chose 20 soundings.

Figure 7. Top view of raw bathymetry swath used for testing, from the Wiggles Bank on the St. Mary’s River in Georgia. Depth shown in color.
1.2.3 Hyperparameter Training

Optimal kernel hyperparameters \( \theta = \{l, \sigma_f^2\} \) maximize the log marginal likelihood (LML) \([1]\). LML rewards model fit and punishes model complexity. It is far too computationally cumbersome to calculate LML using large datasets. Thus, the training process was performed offline on the same 10m \( \times \) 10m patch before running a survey. A LML heat map was calculated over ranges of \( l \) and \( \sigma_f^2 \) (alternatively, gradient descent can be used). The largest LML value determined the selection of \( \theta \). These parameters were inputted to MP-GPR and applied across the entire survey. Every variation of downsampling percent \( s \) for each downsampling method required retraining \( \theta \). Trained hyperparameter values can be found in the Appendix.

![Log marginal likelihood plot determined optimal hyperparameters for \( s = 30 \) with systematic downsampling (maximum value shown in red).](image)

1.2.4 Incorporating Uncertainty

The original MP-GPR formulation assumed each training data point contained the same uncertainty value \( \sigma_n^2 \). Uncertainty values were stored in \( W \), the sensor
noise matrix, such that \( W = I \sigma_n^2 \) where \( I \) was an \( N \times N \) identity matrix. Each data point had one associated uncertainty value that represented a \( d \)-dimensional Gaussian distributed error.

Data collected by MBES are subject to multiple uncertainty sources, such as roll error, pitch error, heave error, refraction error (due to the sound speed profile), sonar system error, etc. [18]. Importantly, each data point obtained using a sonar system has unique uncertainty values in every input dimension (northing, easting, and depth).

The \( d \)-dimensional uncertainty of a sonar sounding can be visualized as a ‘3D uncertainty sphere’ around the data point. For bathymetry data in GPR, \( \sigma_n^2 \) was properly expressed as an uncertainty vector \( \sigma_n^2 = [\sigma_1^2, \sigma_2^2, ..., \sigma_N^2]^T \) with each uncertainty value corresponding to a point in \( x \). The sensor noise matrix became \( W = \text{diag}(\sigma_n^2) \). Due to this formulation, capturing the dominant sources of uncertainty included secondary error sources in other dimensions within the uncertainty sphere. Calculating \( \sigma_n^2 \) was cheap as to not subvert the computational gains from downsampling.

Capturing the dominant error in training data target values \( y \), water depth, was prioritized. The largest relative source of depth error came from the MBES’s range and beam angle uncertainty. Outer beams in each sonar ping were subject to greater uncertainty than the inner beams. For a given beam angle \( \phi_i \) and range \( r_i \), uncertainty \( \sigma_i^2 \) was calculated for each data point as,

\[
\sigma_i^2 = \left( \frac{\partial y_i}{\partial r_i} \sigma_r \right)^2 + \left( \frac{\partial y_i}{\partial \phi_i} \sigma_\phi \right)^2 = (\cos(\phi_i) \sigma_r)^2 + (r_i \sin(\phi_i) \sigma_\phi)^2
\]

where \( \sigma_r \) and \( \sigma_\phi \) were range and beam angle uncertainty.

The range variance is sonar system dependent, and the WASSP used in these surveys reported range resolution of 0.1m (\( \pm 0.1m = 0.2m \)). Assuming a 6\( \sigma \) reported measurement, \( \sigma_r = 0.2m/6 = 0.0333m \). Beam angle resolution was roughly
estimated as total beam width divided by the number of beams. For the WASSP with a 120-degree beam width and 256 beams, \( \sigma_\phi = \frac{2\pi/3}{256} = 0.0081 \) radians/beam. A comparison was shown in Figure 9 between the (updated) changing uncertainty vector \( \sigma_n^2 \) and constant uncertainty \( \sigma_n^2 \) (used previously in MP-GPR).

![95% Confidence [+/-2\sigma] Intervals on a Sonar Ping](image)

**Figure 9.** Uncertainty values for a given sonar ping (across track) for changing uncertainties (dashed blue) and constant uncertainties (solid red).

The predicted uncertainties for an entire survey swath can be viewed in Figure 10. As expected, uncertainty were small near the center of the swath and grew towards the outer edges.

### 1.3 Results

By varying downsampling percent \( s \) for each downsampling method, compute time and solution accuracy could be compared to the exact MP-GPR solution with all data points. Two different metrics were timed. Mean processing time was the
time only for performing inference (calculating GPR solutions over dense input points). System time was the overall time the machine used to run through the entire rosbag, downsample, perform inference, and perform any other general processes. ‘Normalized’ and ‘mean’ metrics were divided by the number of prediction tiles generated. Prediction tiles were only calculated where there were sufficient datapoints present. A given downsampling method did or did not include enough points spatially to merit a prediction tile, and thus the total number of tiles computed was different for each method and $s$.

Root mean square error (RMSE) and mean absolute error (MAE) were calculated for each downsampled MP-GPR inference with respect to exact MP-GPR
inference over all $x_*$ dense inference points. All methods exhibited rapid error growth for smaller $s$. On average, systematic and average performed the best, followed by hybrid, random, K-means, and neighbor.

![Error (RMSE) vs. Downsample](image)

Figure 11. Root mean square error versus downsample percent $s$ for different methods.

When running MP-GPR over the swath, hundreds of inference tiles were produced and the time to process each tile was recorded. Taking the mean of the processing tile times resulted in mean processing time, a normalized metric for method comparison.

The inference processing time curves for every method exhibited similar shapes, as inference time decreased linearly with decreased $s$ in Figure 12. Neighbor at $s = 0.2, 0.1$ saw high compute time due to poor spatial distribution leading to denser (slower) matrices to compute. Inference for all methods was faster than inference using all datapoints in the exact solution.

Figure 13 shows RMSE against inference time. In general, $s$ increased from
Figure 12. Mean processing time (inference) versus downsample percent \( s \) for different methods.

Figure 13. RMSE versus mean processing time (inference).
0.10 on the left to 0.9 on the right. Points close to the origin represented an ideal solution with both high accuracy and minimum processing time.

![Figure 14. Total system runtime versus downsample percent s](image)

To understand the time required to run the entire MP-GPR algorithm, total system runtime was recorded for each method and shown in Figure 14. K-means with $s = 0.9$ was the only approach that required more overall time to compute a solution than the exact solution. When the system runtime was normalized using the number of tiles computed (shown in Figure 15) K-mean, systematic, hybrid, and neighbor at $s = 0.9$ were more expensive to compute than exact. After $s < 0.8$ all methods were more efficient than exact and trended linearly.

1.4 Discussion

As $s$ decreased, every method trended towards $\frac{1}{\sqrt{s}}$, which matched the expected behavior of RMSE metrics. This indicated that the downsampling methods performed no better or worse than any other error metric experiencing data
Figure 15. Normalized system runtime versus downsample percent $s$

reduction. The inference time curves’ linearity may imply MP-GPR inference with
downsampling had a linear time complexity and was indifferent to the method se-
lected. This differed from the GPR time complexity of $O(N^3)$ due to the numerous
computational speedups available in MP-GPR.

Neighbor’s poor performance was likely due to absence of data in key spatial
regions of the swath (e.g. center of the sonar swath, see image in Appendix). K-means was expected to be an improvement over random sampling and point
averaging, but there was no significant reduction of error. Additionally hybrid
sampling failed to offer notable improvement over random and systematic sampling
for $s \leq 0.5$, yet performed best when $s > 0.5$.

RMSE was largely affected by the number of inclusion points in $D_R$, but chosen
hyperparameters as well as random seeds may have played a role. Optimizing
hyperparameters was an inexact process for this data as the true maximum LML
value for each trial risked divergence and was replaced by a conservative subjective
estimate.

The results indicated that for $s \geq \frac{1}{3}$, it was better to use simple and cheap methods (systematic, average, hybrid, random) over advanced methods (K-means and neighbor). For very small $s$ there were benefits to using unique downsampling approach, though the $s < 10$ solution space was not fully explored. In a generic MBES survey, using hybrid, systematic, or point averaging methods would be preferred based on their minimal RMSE and fast compute time.

1.4.1 Future Work

There may be easily obtainable computational speed-ups in the downsampling algorithms as the programmer was an engineer, not a computer scientist. Some methods required index sorting to ensure spatially-organized blocks, which increased runtime. Additionally, more trials should be conducted to ensure system runtime variability was not a major factor in time metrics.

MP-GPR should be compiled on an embedded system with a GPU, such as the Nvidia Jetson Xavier, and tested to determine if real-time computation is attainable. If so, the algorithm should be deployed on an AUV/ASV for seafloor mapping and use the maximum uncertainty regions from GPR inference as waypoints in an autonomous path-planning algorithm to re-map low confidence areas. Additionally, as any variable (not only seafloor depth) can be inputted to GPR, the algorithm is primed to map other variables such as salinity, temperature, ocean current, or other spatially-distributed environmental variables.

Currently, operations such as downsampling were performed on each sonar ping. While this approach aligns with processing data as it obtained, expanding the operations to run over multiple pings gathered together over time (i.e. ‘sub-swath’) may show improvements. This could cause along track and across track data density to be similar, improving GPR training and inference. This approach
would also enable calculation and comparison of 3D surface normals to contribute to the uncertainty analysis.

Alternative kernels should be explored. Particularly, the Matern kernel that trains unique length scales for each dimension of the data could improve GPR inference as downsampled data in this study suffered from irregular spacing at larger $s$ values (dense along track, sparse across track). Matern kernels also have a smoothness parameter $\nu$ which may offer a smoother seafloor curvature fit.

An efficient, intelligent hyperparameter trainer would provide consistent means to train parameters. This would also contribute to live, adaptive hyperparameters that can be retrained during a survey. Adaptive parameters would allow MP-GPR to accurately map environments as they change (i.e. from sand ripples to shipwreck). However, the high compute cost of the log marginal likelihood is prohibitive.

The uncertainty vector calculations could be paired with a higher fidelity uncertainty model, such as the Combined Uncertainty and Bathymetric Estimator (CUBE) [19]. CUBE a statistical processing algorithm for multi-beam sonar that would be a useful tool for applications that require precisely-bounded uncertainty estimates at each sounding.

1.5 Conclusion

This study built upon the in-situ MP-GPR mapping algorithm by introducing multiple downsampling methods to reduce required compute time. A preprocessing filter was applied to remove outliers as to not bias certain methods from selecting outliers as ‘interesting points.’ For each downsampled data point, an uncertainty value was assigned using an improved model that combined the beam angle and range uncertainty along the seafloor. Optimal hyperparameters were trained offline on a sub-swath of data. During live simulations, raw pings
were streamed in, filtered, downsamped, assigned uncertainty values, and queued together into blocks. Inference was performed on blocks by periodically generating a dense grid of point cloud tiles to form a 2.5D model of the seafloor.

The downsampling methods explored were random, systematic, hybrid, point averaging, neighbor, and K-means. Decreasing the downsample percent led to a reduction in computation time for each method, but also resulted in an increased RMSE. The best methods were hybrid, systematic (decimation), and point averaging, with random downsampling as a reasonable alternative. K-means and neighbor under-performed due to high RMSE on average.
LIST OF REFERENCES


[14] “Cuda runtime api documentation.”


APPENDIX A

Notation and Relevant Equations

The following variables and parameters are present in GPR equations. Bold symbols indicate matrix quantities and standard roman characters indicate scalar values.

- $D$ is the set of training data, consisting of input vectors mapping to target scalars, which is the input to a GPR model, $D = \{(x_i, y_i) | i = 1, \ldots, N\}$
- $N$ is the number of data points in the training data
- $d$ is the number of dimensions of the input data $x$
- $x$ is a $N \times d$ matrix of input values and the independent variables in the training data, $x_i = [x_1, x_2, \ldots, x_d]$
- $y$ is a $N \times 1$ matrix of target values and part of the training data
- $f$ is the underlying function of interest that maps $x_i$ to $y_i$, such that $y_i = f(x_i) + \epsilon$
- $\epsilon$ is the observation error assumed to be independent identically distributed Gaussian noise with $\mu = 0$, $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$
- $\sigma_n^2$ is the scalar sensor noise variance
- $\sigma_n^2$ is the sensor noise vector, $\sigma_n^2 = [\sigma_1^2, \sigma_2^2, \ldots, \sigma_N^2]^T$
- $\sigma_\phi$ is sonar beam angle uncertainty, used to calculate sensor noise $\sigma_n^2$
- $\sigma_r$ is sonar range uncertainty, used to calculate sensor noise $\sigma_n^2$
- $A$ is the number of prediction points for inference
- $x_*$ is a $A \times d$ matrix of prediction input points for inference
- $y_*$ is a $A \times 1$ matrix of the solution means of GPR (output prediction points determined by inference with the GPR model using $x_*$)
- $\Sigma_*$ is a $A \times A$ output covariance matrix with diagonal values as the solution
variances

- \( k(x, x') \) is the kernel function used to calculate covariances
- \( d' \) is the distance between two points used in a kernel, such that \( d' = |x - x'| \)
- \( l \) is the lengthscale present in the kernel function as a hyperparameter
- \( \sigma_f^2 \) is the process noise (variance) of the underlying function \( f \) and is present in the kernel function as a hyperparameter
- \( \theta \) is the set of hyperparameters in the kernel function, \( \theta = \{ l, \sigma_f^2 \} \)
- \( s \) is the downsample percent parameter, \( 0 \leq s \leq 1 \)
- \( K \) is a \( N \times N \) auto-covariance matrix for input points \( x, K = k(x, x) \)
- \( K_\ast \) is a \( A \times N \) covariance matrix for prediction input points \( x_\ast \) and input points \( x, K_\ast = k(x_\ast, x) \)
- \( K_{\ast\ast} \) is a \( A \times A \) covariance matrix for prediction input points \( x_\ast, K_{\ast\ast} = k(x_\ast, x_\ast) \)
- \( W \) is a \( N \times N \) sensor noise matrix with variances pertaining to each input point. For constant variances, \( W = \sigma_n^2 I \). For datapoints with changing variances, \( W = diag(\sigma_n^2) \).
- \( I \) is the identity matrix
- \( V \) is a \( N \times N \) complete input covariance matrix, \( V = K + W \)

The standard equations that govern GPR with a squared exponential kernel are:

\[
k(x, x') = \sigma_f^2 \exp\left(-\frac{|x - x'|^2}{2l^2}\right) \quad (A.1)
\]

\[
y_\ast = K_\ast \cdot V^{-1} \cdot y \quad (A.2)
\]

\[
\Sigma_\ast = K_{\ast\ast} - K_\ast \cdot V^{-1} \cdot K_\ast^T \quad (A.3)
\]

MP-GPR uses Cholesky decompositions for speed and to ensure solution convergence and utilizes a modified version of these standard equations. The motivation and equations are described in great detail by Krasnosky [5].
The approximation to the squared exponential kernel used in MP-GPR is:

\[
k(x, x') = \begin{cases} 
\sigma_f^2 \left[ \frac{2 + \cos(2\pi \frac{d}{l})}{4} \right] (1 - \frac{d}{l}) + \frac{1}{2\pi} \sin(2\pi \frac{d}{l}) & \text{if } d < l \\
0 & \text{if } d \geq l
\end{cases}
\]  

(A.4)

The log marginal likelihood (LML) is defined as:

\[
LML = -\frac{1}{2} y^T V^{-1} y - \frac{1}{2} \log |V| - \frac{N}{2} \log(2\pi)
\]  

(A.5)

Equations relating to sonar uncertainty calculations are:

\[
y_i = r_i \cos(\phi_i)
\]

\[
\sigma_i^2 = \sigma_{\text{di}}^2 = \left( \frac{\partial y_i}{\partial r_i} \sigma_r \right)^2 + \left( \frac{\partial y_i}{\partial \phi_i} \sigma_\phi \right)^2
\]

\[
\sigma_i^2 = (\cos(\phi_i) \sigma_r)^2 + (r_i \sin(\phi_i) \sigma_\phi)^2
\]

The likelihood that an approximate solution is equal to an exact solution as a given inference point \( x_i \) is given:

\[
\text{pointLikelihood} = p(z_{\text{approx}} - z_{\text{exact}} = 0) = \exp \left( -\frac{1}{2} \frac{(\mu_{\text{approx}} - \mu_{\text{exact}})^2}{\sigma_{\text{approx}}^2 + \sigma_{\text{exact}}^2} \right) \frac{1}{\sqrt{2\pi(\sigma_{\text{approx}}^2 + \sigma_{\text{exact}}^2)}}
\]  

(A.6)

The formula for root mean square error is:

\[
RMSE = \sqrt{\frac{1}{A} \sum_{i=1}^{A} (y_{i,\text{exact}}^i - y_{i,\text{approx}}^i)^2}
\]  

(A.7)

The formula for mean absolute error is:

\[
MAE = \frac{1}{A} \sum_{i=1}^{A} (y_{i,\text{exact}}^i - y_{i,\text{approx}}^i)
\]  

(A.8)

MP-GPR uses Cholesky decompositions for speed and convergence. Those equations are described in great detail by Krasnosky [5].
APPENDIX B

Significant Parameters

This section lists the compute time and trained hyperparameters and for each downsampling method. Compute time is the total compute time (downsample time plus inference time) divided by the number of inference tiles computed. An inference tile is the output of MP-GPR and a dense point cloud \( (x_*, \{\text{northing, easting}\}, y_*, \{\text{depth}\}) \).

<table>
<thead>
<tr>
<th>( s )</th>
<th>Random</th>
<th>Hybrid</th>
<th>DN</th>
<th>K-means</th>
<th>Systematic</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>0.238</td>
<td>0.371</td>
<td>0.836</td>
<td>0.852</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>80%</td>
<td>0.211</td>
<td>0.308</td>
<td>0.645</td>
<td>0.708</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>70%</td>
<td>0.185</td>
<td>0.244</td>
<td>0.534</td>
<td>0.602</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>60%</td>
<td>0.157</td>
<td>0.226</td>
<td>0.394</td>
<td>0.470</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>50%</td>
<td>0.134</td>
<td>0.201</td>
<td>0.320</td>
<td>0.389</td>
<td>0.128</td>
<td>0.134</td>
</tr>
<tr>
<td>40%</td>
<td>0.111</td>
<td>0.160</td>
<td>0.219</td>
<td>0.303</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>33%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.093</td>
<td>0.088</td>
</tr>
<tr>
<td>30%</td>
<td>0.089</td>
<td>0.110</td>
<td>0.154</td>
<td>0.214</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>25%</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.075</td>
<td>0.072</td>
</tr>
<tr>
<td>20%</td>
<td>0.063</td>
<td>0.083</td>
<td>0.101</td>
<td>0.132</td>
<td>0.062</td>
<td>0.061</td>
</tr>
<tr>
<td>10%</td>
<td>0.047</td>
<td>0.055</td>
<td>0.061</td>
<td>0.074</td>
<td>0.044</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Table B.1. Compute Times (seconds) for Downsampling Methods

<table>
<thead>
<tr>
<th>( s )</th>
<th>RMSE</th>
<th>MAE</th>
<th>( l )</th>
<th>( \sigma_f )</th>
<th>Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>0.0517</td>
<td>0.0319</td>
<td>1.06</td>
<td>0.09</td>
<td>0.238</td>
</tr>
<tr>
<td>80%</td>
<td>0.0543</td>
<td>0.0346</td>
<td>1.06</td>
<td>0.11</td>
<td>0.211</td>
</tr>
<tr>
<td>70%</td>
<td>0.0744</td>
<td>0.0408</td>
<td>1.06</td>
<td>0.15</td>
<td>0.185</td>
</tr>
<tr>
<td>60%</td>
<td>0.0665</td>
<td>0.0370</td>
<td>1.06</td>
<td>0.15</td>
<td>0.157</td>
</tr>
<tr>
<td>50%</td>
<td>0.0638</td>
<td>0.0385</td>
<td>1.06</td>
<td>0.19</td>
<td>0.134</td>
</tr>
<tr>
<td>40%</td>
<td>0.0625</td>
<td>0.0384</td>
<td>1.06</td>
<td>0.24</td>
<td>0.111</td>
</tr>
<tr>
<td>30%</td>
<td>0.0830</td>
<td>0.0446</td>
<td>1.27</td>
<td>0.25</td>
<td>0.089</td>
</tr>
<tr>
<td>20%</td>
<td>0.0830</td>
<td>0.0477</td>
<td>1.27</td>
<td>0.39</td>
<td>0.063</td>
</tr>
<tr>
<td>10%</td>
<td>0.1819</td>
<td>0.0849</td>
<td>1.45</td>
<td>0.60</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Table B.2. Results and Parameters for Random Downsampling
<table>
<thead>
<tr>
<th>s</th>
<th>RMSE</th>
<th>MAE</th>
<th>( l )</th>
<th>( \sigma_f )</th>
<th>Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>0.0851</td>
<td>0.0522</td>
<td>1.06</td>
<td>0.11</td>
<td>0.371</td>
</tr>
<tr>
<td>80%</td>
<td>0.1047</td>
<td>0.0595</td>
<td>1.06</td>
<td>0.11</td>
<td>0.308</td>
</tr>
<tr>
<td>70%</td>
<td>0.0727</td>
<td>0.0436</td>
<td>1.06</td>
<td>0.11</td>
<td>0.244</td>
</tr>
<tr>
<td>60%</td>
<td>0.0851</td>
<td>0.0509</td>
<td>1.06</td>
<td>0.15</td>
<td>0.226</td>
</tr>
<tr>
<td>50%</td>
<td>0.0979</td>
<td>0.0576</td>
<td>1.08</td>
<td>0.19</td>
<td>0.201</td>
</tr>
<tr>
<td>40%</td>
<td>0.0964</td>
<td>0.0586</td>
<td>1.08</td>
<td>0.23</td>
<td>0.160</td>
</tr>
<tr>
<td>30%</td>
<td>0.0864</td>
<td>0.0526</td>
<td>1.08</td>
<td>0.30</td>
<td>0.110</td>
</tr>
<tr>
<td>20%</td>
<td>0.1162</td>
<td>0.0676</td>
<td>1.08</td>
<td>0.50</td>
<td>0.083</td>
</tr>
<tr>
<td>10%</td>
<td>0.1659</td>
<td>0.1009</td>
<td>1.18</td>
<td>0.62</td>
<td>0.055</td>
</tr>
</tbody>
</table>

Table B.3. Results and Parameters for Hybrid Downsampling

<table>
<thead>
<tr>
<th>s</th>
<th>RMSE</th>
<th>MAE</th>
<th>( l )</th>
<th>( \sigma_f )</th>
<th>Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>0.1213</td>
<td>0.0723</td>
<td>1.25</td>
<td>0.09</td>
<td>0.836</td>
</tr>
<tr>
<td>80%</td>
<td>0.1308</td>
<td>0.0814</td>
<td>1.06</td>
<td>0.09</td>
<td>0.645</td>
</tr>
<tr>
<td>70%</td>
<td>0.1219</td>
<td>0.0749</td>
<td>1.06</td>
<td>0.11</td>
<td>0.534</td>
</tr>
<tr>
<td>60%</td>
<td>0.1636</td>
<td>0.0919</td>
<td>1.06</td>
<td>0.13</td>
<td>0.394</td>
</tr>
<tr>
<td>50%</td>
<td>0.1250</td>
<td>0.0728</td>
<td>1.06</td>
<td>0.21</td>
<td>0.320</td>
</tr>
<tr>
<td>40%</td>
<td>0.1348</td>
<td>0.0828</td>
<td>0.96</td>
<td>0.31</td>
<td>0.219</td>
</tr>
<tr>
<td>30%</td>
<td>0.1402</td>
<td>0.0917</td>
<td>0.96</td>
<td>0.39</td>
<td>0.154</td>
</tr>
<tr>
<td>20%</td>
<td>0.1631</td>
<td>0.1094</td>
<td>0.96</td>
<td>0.49</td>
<td>0.101</td>
</tr>
<tr>
<td>10%</td>
<td>0.1979</td>
<td>0.1376</td>
<td>0.96</td>
<td>0.71</td>
<td>0.061</td>
</tr>
</tbody>
</table>

Table B.4. Results and Parameters for Dissimilar Neighbor Downsampling

<table>
<thead>
<tr>
<th>s</th>
<th>RMSE</th>
<th>MAE</th>
<th>( l )</th>
<th>( \sigma_f )</th>
<th>Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>0.1281</td>
<td>0.0729</td>
<td>0.88</td>
<td>0.13</td>
<td>0.852</td>
</tr>
<tr>
<td>80%</td>
<td>0.1085</td>
<td>0.0619</td>
<td>1.06</td>
<td>0.17</td>
<td>0.708</td>
</tr>
<tr>
<td>70%</td>
<td>0.1127</td>
<td>0.0632</td>
<td>1.06</td>
<td>0.17</td>
<td>0.602</td>
</tr>
<tr>
<td>60%</td>
<td>0.1243</td>
<td>0.0709</td>
<td>1.06</td>
<td>0.17</td>
<td>0.470</td>
</tr>
<tr>
<td>50%</td>
<td>0.1111</td>
<td>0.0635</td>
<td>1.14</td>
<td>0.19</td>
<td>0.389</td>
</tr>
<tr>
<td>40%</td>
<td>0.0949</td>
<td>0.0565</td>
<td>1.23</td>
<td>0.22</td>
<td>0.303</td>
</tr>
<tr>
<td>30%</td>
<td>0.1060</td>
<td>0.0624</td>
<td>1.05</td>
<td>0.39</td>
<td>0.214</td>
</tr>
<tr>
<td>20%</td>
<td>0.1010</td>
<td>0.0626</td>
<td>1.01</td>
<td>0.70</td>
<td>0.132</td>
</tr>
<tr>
<td>10%</td>
<td>0.1513</td>
<td>0.0939</td>
<td>1.01</td>
<td>1.64</td>
<td>0.074</td>
</tr>
</tbody>
</table>

Table B.5. Results and Parameters for K-means Downsampling
<table>
<thead>
<tr>
<th>s</th>
<th>RMSE</th>
<th>MAE</th>
<th>l</th>
<th>$\sigma_f$</th>
<th>Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>0.0564</td>
<td>0.0362</td>
<td>1.06</td>
<td>0.18</td>
<td>0.128</td>
</tr>
<tr>
<td>33%</td>
<td>0.0674</td>
<td>0.0393</td>
<td>1.06</td>
<td>0.27</td>
<td>0.093</td>
</tr>
<tr>
<td>25%</td>
<td>0.0718</td>
<td>0.0422</td>
<td>1.06</td>
<td>0.36</td>
<td>0.075</td>
</tr>
<tr>
<td>20%</td>
<td>0.0874</td>
<td>0.0512</td>
<td>1.07</td>
<td>0.62</td>
<td>0.062</td>
</tr>
<tr>
<td>10%</td>
<td>0.2134</td>
<td>0.1034</td>
<td>1.01</td>
<td>1.12</td>
<td>0.044</td>
</tr>
</tbody>
</table>

Table B.6. Results and Parameters for Systematic Downsampling

<table>
<thead>
<tr>
<th>s</th>
<th>RMSE</th>
<th>MAE</th>
<th>l</th>
<th>$\sigma_f$</th>
<th>Compute Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>0.0580</td>
<td>0.0355</td>
<td>1.08</td>
<td>0.16</td>
<td>0.134</td>
</tr>
<tr>
<td>33%</td>
<td>0.0683</td>
<td>0.0412</td>
<td>1.08</td>
<td>0.28</td>
<td>0.088</td>
</tr>
<tr>
<td>25%</td>
<td>0.0847</td>
<td>0.0460</td>
<td>1.08</td>
<td>0.39</td>
<td>0.072</td>
</tr>
<tr>
<td>20%</td>
<td>0.1047</td>
<td>0.0531</td>
<td>1.08</td>
<td>0.46</td>
<td>0.061</td>
</tr>
<tr>
<td>10%</td>
<td>0.1663</td>
<td>0.0953</td>
<td>1.11</td>
<td>1.60</td>
<td>0.047</td>
</tr>
</tbody>
</table>

Table B.7. Results and Parameters for Average Downsampling
APPENDIX C

Downsampled Swaths

The following swaths show the bathymetry used. The raw bathymetry contains all data streamed in from the MBES. The downsampled swaths show all methods for $s = 20\%$. The raw swath was 150m wide and 60m tall with 330,000 soundings.

Figure C.1. Top view of the raw bathymetry.

Figure C.2. Top view of the 20\% random downsampled swath.
Figure C.3. Top view of the 20% systematic downsampled swath.

Figure C.4. Top view of the 20% hybrid downsampled swath.

Figure C.5. Top view of the 20% average downsampled swath.
Figure C.6. Top view of the 20% dissimilar neighbor downsampled swath.

Figure C.7. Top view of the 20% K-means downsampled swath.
APPENDIX D

Uncertainty Tiles

The following uncertainty maps show the predictive uncertainties at each dense prediction point (grouped into tiles). All methods are shown with 20% downsample. MP-GPR solution tiles were not computed when downsampled points were too sparse in a given region, as seen with 20% neighbor, and hence 30% neighbor is shown as well. In general, uncertainties were small near the center of the swath (due to high density and higher confidence of data points), grew larger towards the edges of the swath, and were maximum beyond the training data.

Figure D.1. Top view of the uncertainty prediction tiles for exact (100% of raw data used.)
Figure D.2. Top view of the uncertainty prediction tiles for random $s = 0.2$.

Figure D.3. Top view of the uncertainty prediction tiles for systematic $s = 0.2$. 
Figure D.4. Top view of the uncertainty prediction tiles for hybrid $s = 0.2$.

Figure D.5. Top view of the uncertainty prediction tiles for point averaging $s = 0.2$. 
Figure D.6. Top view of the uncertainty prediction tiles for neighbor $s = 0.2$.

Figure D.7. Top view of the uncertainty prediction tiles for neighbor $s = 0.3$.
Figure D.8. Top view of the uncertainty prediction tiles for K-means $s = 0.2$. 