Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datavetenskap
2019 | LIU-IDA/LITH-EX-A--19/016--SE

Bayesian optimization for selecting training and validation data for supervised machine learning

– using Gaussian processes both to learn the relationship between sets of training data and model performance, and to estimate model performance over the entire problem domain

---

Bayesiansk optimering för val av träning- och valideringsdata för övervakad maskininlärning

David Bergström

Supervisor : Mattias Tiger
Examiner : Fredrik Heintz

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Validation and verification in machine learning is an open problem which becomes increasingly important as its applications becomes more critical. Amongst the applications are autonomous vehicles and medical diagnostics. These systems all needs to be validated before being put into use or else the consequences might be fatal.

This master's thesis focuses on improving both learning and validating machine learning models in cases where data can either be generated or collected based on a chosen position. This can for example be taking and labeling photos at the position or running some simulation which generates data from the chosen positions.

The approach is twofold. The first part concerns modeling the relationship between any fixed-size set of positions and some real valued performance measure. The second part involves calculating such a performance measure by estimating the performance over a region of positions.

The result is two different algorithms, both variations of Bayesian optimization. The first algorithm models the relationship between a set of points and some performance measure while also optimizing the function and thus finding the set of points which yields the highest performance. The second algorithm uses Bayesian optimization to approximate the integral of performance over the region of interest. The resulting algorithms are validated in two different simulated environments.

The resulting algorithms are applicable not only to machine learning but can also be used to optimize any function which takes a set of positions and returns a value, but are more suitable when the function is expensive to evaluate.

Acknowledgments

Throughout the entirety of this master's thesis project I have received a a great deal of support and advice. First of all I would like to thank my supervisor Mattias Tiger for his nonstop encouragement and expertise. Secondly, I would like to thank my examiner Fredrik Heintz for his valuable suggestions for how to improve both the project and the report.

I would also like to thank my opponent Andreas Norrstig for our discussions and his insightful input. Furthermore, I would like to extend my gratitude to my friends and colleagues at the university who shared their experiences in academia and made me feel welcome.

At last, but not least, I would like to thank my family and my partner for their boundless support throughout my years of study and during this master's thesis project.

Linköping, May 2019
David Bergström

Contents

List of Figures

Notation

Summary of notation used in report:

Notation	Description
$\mathcal{M}$	Supervised machine learning model
X	Input space for supervised machine learning model
Y	Output space for supervised machine learning model
$\mathcal{D}$	The obtainable subset of all labeled data points, more precisely a subset of all input-output pairs, where each data point is $(x, y) \in \mathcal{D} \subset X \times Y$, where $x \in X$ and $y \in Y$.
$\mathcal{D}_{\text{train}}$	Subset of $\mathcal{D}$ used for training
$\mathcal{D}_{\text{test}}$	Subset of $\mathcal{D}$ used for testing
f_{gen}	Data generating function which maps from Euclidean free space $\mathbf{X}_{\text{free}}$ to $\mathcal{D}$
$\mathbf{X}_{\text{free}}$	Set of all possible inputs to f_{gen}
$\mathcal{X}_{\text{free}}$	N-ary Cartesian power $\mathcal{X}_{\text{free}} = \mathbf{X}_{\text{free}}^N$, meaning that each element in $\mathcal{X}_{\text{free}}$ is a N-tuple where each element lies in $\mathbf{X}_{\text{free}}$. Used to represent configurations where multiple objects are placed, where each element in the N-tuple represents the position of an object.

1 Introduction

Machine learning is on the rise and there is a wide range of possible applications. One of the main areas in machine learning is supervised learning. The core idea is to make predictions using previously collected data. The data consists of pairs (x, y) and the idea is to analyze these examples in order to predict y given a new x. A model is chosen based on some assumptions about the data. Most models have several parameters to allow it to fit to several types of data distributions. The model parameters are updated to fit the data, with a process called training. The resulting model can then be used to make predictions.

Having trained the model, we want to know how well it works. Previously the model has only seen one set of data and it is possible that the model has only memorized what values of x corresponds to what values for y. This means it cannot make a prediction once it encounters a new value which it has never seen before. This is called overfitting, meaning that the model has overfitted to the training data. Conversely, if the model works well for new data points it has generalized outside the training data set.

Traditionally data sets are divided up into several parts in order to use separate sets of data for training and for evaluating how well the approach generalizes. These sets are called training sets and test sets, respectively. However, what happens if the test set is very similar to the training data set? This would result in the measure of generalization being poor and not giving any information about the actual performance of the model. Consequently, having a good training set and test set is essential for supervised machine learning.

1.1 Motivation

For some problems it is possible to collect or to generate data from a specific spatial position. This can be taking a set of photos while standing at the position and labeling them, measuring the signal strength at the position or running a simulation which simulates some property at the position. The process of collecting or generating data is viewed as a function which takes a position p_i and generates one or several data points (x_i, y_i).

For a single data point (x, y), loss is defined as the distance between y and the models prediction y^*. Having low loss means that the model is able to make a good prediction and high loss means that the prediction is poor. There are many different distance functions available and consequently many different loss functions, but which one is being used is not important in this context.

It is possible to apply this data generating function to the problem of supervised learning by using it to generate training and test data sets. In the case of generating test data, it is possible to associate the positions from which the data is generated and the loss for that data point. This means that the concept of generality can be rephrased as having low loss in a certain area of interest. For example, the area of interest might be a certain area where the model is going to be used, such as a certain building or city block.

A training data set can be generated by selecting a set of points $c = \{p_1, p_2, ..., p_n\}$ and passing them to the data generating function. After the model has been trained on the data it can be tested, either using the data generating function described above or using some other method. The total loss in the area of interest depends on the choice of c, meaning that it is desirable to find the set of points c such that the lowest possible total loss over the area of interest is achieved.

These two applications are special cases of two different types of more general problems. The first application, iteratively selecting points to get as good as possible estimate of the total model loss, is a special case of iteratively selecting points for estimating any function or integral of such function. The second application, selecting a set of points from which to generate training data, can be generalized to finding a set of points which maximizes function which returns a real number.

The solutions to these general problems can be applied to a wide range of problems. In this master's thesis two problems in this category are defined, modelled and solved using a common notation and method. The first of the two problems is the problem of placing a set of LIDAR sensors in a building and the second is placing a set of wireless access points when setting up a wireless network. Both are cases of placing a set of objects in a physical space and measuring how good the placement is according to some measure. In the case of LIDAR sensors, we want to maximize the total coverage of the LIDAR sensors. Similarly, in the case of placing wireless access points, we want to maximize the total signal strength in the building.

1.2 Aim

This thesis has two main purposes. The first is to model functions which maps from sets of points in Euclidean space to real numbers. The second is to estimate integrals with as few function evaluations as possible. The overarching aim is to model the relationship between model performance and the choice of training data in the context of supervised machine learning, where training and test data can be generated from a specific spatial location.

1.3 Research questions

1. How can Bayesian optimization be extended to allow optimization of a function over sets?

2. How can Bayesian optimization be adapted to sample-efficiently estimate the integral of a function over a closed domain?

3. How can a fixed-size training data set be chosen for a supervised machine learning model?

   The fixed-size training data set is a finite subset of all possible data and is generated by some method as a function of a spatial location. It is chosen such that the supervised model's performance is maximized, according to some given performance measure. The set of all possible data might be, and most often is, infinite in size.

4. How can the total loss over a closed Euclidean space be estimated to measure the generality of a supervised machine learning model?

1.4 Delimitations

While the two applications of Gaussian processes and Bayesian optimization has many potential use-cases, this report focuses on cases where the input space is some kind of closed Euclidean space. More specifically, the elements of the sets are elements from some Euclidean space and the function which is approximated. The output space is also assumed to be reasonably smooth, meaning that points that are close in the input space should also have similar values in the output space.

1.5 Contributions

The contributions of this master's thesis can be divided into two separate categories: describing and defining classes of problems and proposing extensions to the Bayesian optimization algorithm to allow it to be applied to these classes of problems.

This thesis describes a class of stochastic set optimization problems, where a set needs to be chosen such that a function is maximized, while also estimating the function and minimizing the total number of function evaluations. To the authors knowledge, this class of problems has not been studied in this setting before. The class is also extended to include the cases where the function cannot be evaluated directly, but rather is an integral which first has to be estimated.

This thesis adapts a pre-existing method for creating permutation invariant kernels to create a kernel which is suitable for describing the distance between two sets. By using this kernel it is possible to apply Bayesian optimization to solve the first class of problems, i.e. stochastic set optimization. This thesis introduces a new acquisition function which allows Bayesian optimization to be used for estimating of functions rather than optimizing them. Finally, the Bayesian optimization for function estimation is combined with the Bayesian optimization for stochastic set optimization, resulting in a method for solving the second class of problems.

This work resulted in a publication in the proceedings of the Swedish AI Society [2].

1.6 Outline

The next chapter, problem description, outlines and further describes the problem which this thesis aims to solve. After that, the theory chapter describes the relevant theory. This is followed by the method chapter, which describes and adapts the theory and proposes a method for solving the problem. This chapter also describes a few different experiments done to validate parts of the proposed method. The result chapter then presents the result of these experiments. The last two chapters, discussion and conclusion, aims to answer the research questions by analyzing the results.

2 Problem description

There are a few problems which can be solved using the methods proposed in this master's thesis. The purpose of this chapter is to explain what these are and how they relate.

2.1 Camera/LIDAR placement

Consider the problem of placing cameras when installing a camera surveillance system. In order to surveil the building it is important to observe as much of the building as possible. Assume that the budget is predetermined, meaning that you can only use a fixed number of cameras. How should these cameras be placed to observe as much of the building as possible?

Another similar problem is if you want to create a 3D model of a building when you have a floor plan. In that case you have some sort of a scanner which can be placed on the floor. How can you use the floor plan to minimize the total number of scans and thus save time?

The purpose of this section is to describe and define the problem of placing either a set of cameras or a set of LIDAR sensors in a building. The aim is to place the cameras or LIDAR sensors such that they observe as much of the building as possible. The general layout of the building is known, typically obtained from a floor plan or previous scan. This means that the sensors should be placed such that their line of sight does not overlap, since there is no point in scanning the same wall twice or having two sensors surveil the same part of a room.

It is assumed that all cameras or LIDAR sensors have 360 degrees field-of-view, meaning that the orientation of the camera/sensor is irrelevant. It is also assumed that there is no length limit of how far either a camera or a LIDAR sensor can see, or at least that the building is small enough. There are differences between the two sensor types, but in this thesis the problem of placing them is reduces to the same problem.

For the sake of simplicity, the building can be assumed to be divisible into two parts. The first part of the building, $\mathbf{X}_{\text{free}}$, is either free floor or roof, meaning wherever it is possible to place cameras or LIDAR sensors. The second part of the building, $\mathbf{X}_{\text{wanted}}$, are things we can observe, i.e. not floor but either walls or other obstacles. This second part is what we want to observe as much of as possible. As an example figure 2.1 shows a map of a building, where black cells are walls and white are floor. The buildings discussed in this master thesis are discrete, meaning that they consist of several small squares which are referred to as cells throughout the report.

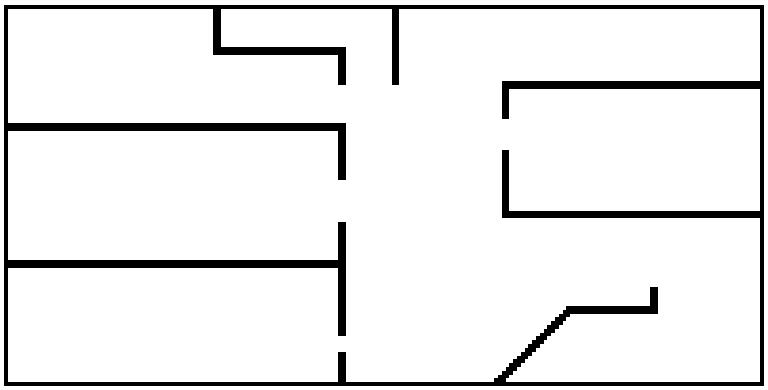


Figure 2.1: Example of building. The map is divided into square cells, black cells are the ones we want to observe and the white cells are where it is possible to place cameras or LIDAR sensors.

Whilst it is possible to place one camera at the time and keep placing cameras outside the view of the previously placed cameras until all cameras are placed, this greedy approach is unlikely to produce an optimal solution. Therefore, the cameras should all be placed at once. The rest of the section aims to explain this problem with the common mathematical notation which will be used throughout the report.

The two previously explained parts of the building can be seen as the free space $\mathbf{X}_{\text{free}}$, which is where cameras or LIDAR sensors can be placed, and the observable space $\mathbf{X}_{\text{wanted}}$, which we want to observe as much of as possible. Both these two can be viewed as sets of positions in Euclidean space.

For each placement $p \in \mathbf{X}_{\text{free}}$ of either a camera or a LIDAR sensor, the subset of all observable cells which are observable from a position p can be written as a function:

$$f_{\text{observe}}(p) = X_{\text{observed}} \subseteq \mathbf{X}_{\text{wanted}}. \tag{2.1}$$

Given that either N cameras or LIDAR sensors are to be placed, the set of all possible configurations can be defined as the N-ary Cartesian power of the $\mathbf{X}_{\text{free}}$ set:

$$\mathcal{X}_{\text{free}} = \mathbf{X}_{\text{free}}^{N} = \underbrace{\mathbf{X}_{\text{free}} \times \mathbf{X}_{\text{free}} \times \cdots \times \mathbf{X}_{\text{free}}}_{N}, \tag{2.2}$$

meaning that each element in $\mathcal{X}_{\text{free}}$ is a set of N points where each point lies in $\mathbf{X}_{\text{free}}$.

As positions can only be observed once, the total set of all observed cells F_{observe} for configuration $c \in \mathcal{X}_{\text{free}}$ can be written as a union of all the individual sensor observations

$$F_{\text{observe}}(c) = \bigcup_{p \in c} f_{\text{observe}}(p). \tag{2.3}$$

If all positions are equally important to observe, the placement problem can be described as finding the element $c^* \in \mathcal{X}_{\text{free}}$ such that the total number of observed cells is maximized

$$c^* = \arg\max_{c \in \mathcal{X}_{\text{free}}} = |F(c)|. \tag{2.4}$$

2.2 Wireless access point placement for maximum coverage

Consider the problem of placing a finite set of wireless access points to provide good wireless connectivity in an entire building When connecting a device to a wireless network the most important factor is the distance to the access point. The signal strength also decreases whenever it passes through solid objects such as walls. A device can generally only connect to one

access point at the time, meaning that there is no point of having more than one access point per room as the device will typically connect to the access point which has the highest signal strength.

A few assumptions are made. Firstly, the general layout of the building is known. More precisely the set of all possible positions to place access points $\mathbf{X}_{\text{free}}$ and the set of all possible positions where wireless connectivity is wanted, $\mathbf{X}_{\text{wanted}}$, is known. Secondly, the total number of access points N is fixed and known from the beginning. Finally, all wireless access points are assumed to provide the same signal strength and at the same distance. The last assumption is not a necessary for the proposed method to work, but simplifies the notation.

Given that N access points are to be placed, the set of all configurations $\mathcal{X}_{\text{free}}$ can be defined as in equation 2.2. After having placed the access points according to a configuration $c \in \mathcal{X}_{\text{free}}$ it is possible to measure the signal strength in a position $x \in \mathbf{X}_{\text{wanted}}$. This can be formalized as writing the signal strength as a function of both the configuration c and the measurement position x: $s(p, c) = \max_{x \in c} s(p, x)$, where $s(p, x)$ is signal strength at the point p given by access point at the point x. The reason for the max function is that the device is assumed to only connect to the access point to which it has the highest signal strength.

The total signal strength in the entire building can then be written as:

$$S(c) = \int_{\mathbf{X}_{\text{wanted}}} s(x, c)dx, \tag{2.5}$$

where $c \in \mathcal{X}_{\text{free}}$.

The problem of finding the best placement for the set of access points can then be formalized as finding the c^* which maximizes the total signal strength, which can be written as

$$\begin{aligned} c^* &= \arg\max_{c \in \mathcal{X}_{\text{free}}} S(c) \\ &= \arg\max_{c \in \mathcal{X}_{\text{free}}} \int_{\mathbf{X}_{\text{wanted}}} s(x, c)dx. \end{aligned} \tag{2.6}$$

2.3 Training data selection problem

Consider the problem of selecting data for a model where a data generating function $f_{\text{gen}} : \mathbf{X}_{\text{free}} \to \mathcal{D}$ is provided together with a finite test data set $\mathcal{D}_{\text{test}}$. The data generating function f_{gen} is a general construct, which takes a point in some Euclidean space $\mathbf{X}_{\text{free}}$ and returns a data point $(x, y) \in \mathcal{D}$. It might for example consist of having a robot collect data somewhere and then having an operator label the data. It could also be projecting a prelabeled point-cloud onto a 2D image, creating a labeled image as well as a RGB image as described by Järemo Lawin et al. [15]. Another alternative is running some sort of realistic simulator to generate similar training data consisting of labeled images as well as RGB images, e.g. CARLA [8].

The goal is to use the model $\mathcal{M}$ to model the relationship between the input space X and the output space Y. The model is trained using training data consisting of pairs of (x, y), where $x \in X$ and $y \in Y$. How the model is trained is not important here, but the data is analyzed in some way and the model is updated. Once the model has been trained, the prediction function $f : X \times \mathcal{M} \mapsto Y$ can be used to make predictions using the model parameters.

The data generating function f_{gen} is used to create the data sets used for training the model. It takes a position $p \in \mathbf{X}_{\text{free}}$ and returns one or more data points (x, y), where $x \in X$ and $y \in Y$. A set of training data $\mathcal{D}_{\text{train}}$ can be generated by choosing a set of points, in other words a configuration. Finally, each point p in the configuration c is passed to the function f_{gen}:

$$\mathcal{D}_{\text{train}} = \{(x, y) \in \mathcal{D} | (x, y) = f_{\text{gen}}(p), p \in c\}. \tag{2.7}$$

In order to evaluate the performance of the model the test data set $\mathcal{D}_{\text{test}}$ is used, it consists of set of corresponding data points (x, y), where $x \in X$ and $y \in Y$. Using the test data set it

is possible to define the loss function for the model and the data set

$$\mathcal{L}(\mathcal{M}, \mathcal{D}_{\text{test}}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_{\text{test}}} \ell(y, f(x, \mathcal{M})), \tag{2.8}$$

where $\ell(.,.)$ is any point-wise loss function, e.g. the mean square error $\ell_{\text{mse}}(y, \hat{y}) = \|y - \hat{y}\|_2^2$ where $\|\dots\|_2$ is the ℓ^2 norm. The loss function is low whenever the model performs well and high when the model performs poorly.

Now, the problem can be formalized as finding a configuration $c = \{p_i\}_{i\in(0,N)}$, where $p_i \in \mathbf{X}_{\text{free}}$ such that the resulting model $\mathcal{M}$ has a low loss $\mathcal{L}$. More precisely

$$\begin{aligned} c^* &= \arg\min_{c\in\mathcal{X}_{\text{free}}} \mathcal{L}(\mathcal{M}, \mathcal{D}_{\text{test}}) \\ &= \arg\min_{c\in\mathcal{X}_{\text{free}}} \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_{\text{test}}} \ell(y, f(x, \mathcal{M})), \end{aligned} \tag{2.9}$$

where $\mathcal{M}$ has been trained on the training data set $\mathcal{D}_{\text{train}}$ generated from c as described in equation 2.7. Another important thing to note is that the training data set has to be different from the test data set $\mathcal{D}_{\text{test}}$ to avoid overestimating how well the model $\mathcal{M}$ has generalized.

2.4 Loss estimation

The aim of evaluating a model is to determine how well the model generalizes outside the training data set. This performance is often measured using a set of data points $\mathcal{D}_{\text{test}}$, a test data set. The evaluation set is a subset of all possible data $\mathcal{D}$. This leads us to the problem with this approach, the loss for a subset of all possibly data does not necessarily represent the loss for all possible data. This section defines the problem of estimating the loss over the entire set of possible data, but limits it to the case where data can be generated by a data generating function which takes points from a closed Euclidean space as input.

Given that the model is trained on some data set $\mathcal{D}_{\text{train}}$ the total loss over the set of all possible data pairs $\mathcal{D}$ is defined as:

$$\mathcal{L}(\mathcal{D}_{\text{train}}) = \sum_{(x,y)\in\mathcal{D}} \ell(y, f(x, \mathcal{M})), \tag{2.10}$$

where $\ell(.,.)$ is some point-wise loss function. Unfortunately, in most cases the set of all possible data is not available, making evaluating the sum impossible.

Usually loss is estimated using a finite test subset $\mathcal{D}_{\text{test}} \subset \mathcal{D}$, which results in a mean loss estimate:

$$\mathcal{L}_{\text{estimate}}(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_{\text{test}}} \ell(y, f(x, \mathcal{M})). \tag{2.11}$$

The accuracy of this estimate depends on how well the test subset $\mathcal{D}_{\text{test}}$ is able to represent all possible data pair $\mathcal{D}$. As the loss is a mean of several point-wise losses, it will not converge to the actual total loss but rather be proportional to the actual loss. As mentioned in the previous section, the accuracy is also affected by the similarity between the test data set and the training data set. If the two sets are similar, but disimilar to the rest of $\mathcal{D}$ the estimated loss will not be representative of the total loss $\mathcal{L}$. This is because we want to measure generality, the model will often perform well on the training set but we want the model to perform well on all possible data.

Another approach can be used if the previously discussed data generating function f_{gen} is available. Remember, the data generating function f_{gen} takes a point in some closed Euclidean space $\mathbf{X}_{\text{free}}$ and returns a data point (x, y), where $x \in X$ and $y \in Y$. Using the data generating function f_{gen} it is possible to rewrite equation 2.10:

$$\mathcal{L}(\mathcal{D}_{\text{train}}) = \int_{\mathbf{X}_{\text{free}}} \ell(y, f(x, \mathcal{M}))dp, \text{ where } (x, y) = f_{\text{gen}}(p). \tag{2.12}$$

However, this integral might still be impractical to use as the function f_{gen} might not be integratable and the set $\mathbf{X}_{\text{free}}$ might have holes in it as shown in figure 2.1. It is possible to extract the function which maps from a point p to the point-wise loss in that point:

$$\ell(p, f(., \mathcal{M})) = \ell(y, f(x, \mathcal{M})), \text{ where } (x, y) = f_{\text{gen}}(p). \tag{2.13}$$

The total loss can then be written:

$$\mathcal{L}(\mathcal{D}_{\text{train}}) = \int_{\mathbf{X}_{\text{free}}} \ell(p, f(., \mathcal{M}))dp. \tag{2.14}$$

It is possible to approximate the function ℓ and thus approximate the overall loss function $\mathcal{L}$. Assuming that the data generating function is reasonably smooth, the function should produce two similar data points when given two points which lies close in Euclidean space. This smoothness can be used by the model which approximates ℓ.

The problem of loss estimation can then be formalized as finding a set of points which minimizes both the uncertainty about the approximation of ℓ and its integral over $\mathbf{X}_{\text{free}}$.

2.5 Putting it all together

The camera/LIDAR placement, the wireless access point placement and the training data selection problem all have a few common traits. All three deals with spatial data, where the problem is to select a set of points all lying in some free space $\mathbf{X}_{\text{free}}$. The performance is measured differently. The aim in camera/LIDAR placement case is to maximize the total number of observed cells, in the access point placement it is to maximize the total signal strength and in the training data selection problem it is to minimize loss.

In some cases the performance can only be evaluated point-wise and then the total loss has to be estimated before it can be taken into account.

The aim of this master's thesis is to find a common solution to these problems by modeling the relationship between a set of Euclidean points and some performance measure, without taking the details of the underlying problem into account, and to either maximize or minimize the performance measure. It is assumed in all three cases that the free space $\mathbf{X}_{\text{free}}$ is a closed Euclidean space and that the performance measure is smooth over this space.

2.6 Related work

There are several areas which relate to this master's thesis, the aim of this section is to give a brief overview of what these are and how they relate to this work.

Next best view

The next best view problem is defined as finding the best way to place a sensor using previous measurements with the goal of observing as much as possible of a scene or object. The problem occurs in the context of 3D reconstruction where a 3D model is being created of an object using as few measurements of the object as possible [5]. It also occurs in instances of autonomous exploration where an agent is tasked with exploring an area and has to choose where to explore next [3]. In both cases the problem occurs as part of an iterative process where one point is placed and information is gathered at each iteration. The problem is to use the information gathered so far to choose where to place the sensor, the aim being to maximize the total amount of information gathered.

In this master's thesis one of the problems under consideration is placing a set of sensors such that they observe as much of possible of the environment. More specifically, the problem is to find a set of points such that they minimize some loss function, e.g. total amount of unknown, or to maximize some performance measure, e.g. total wireless coverage.

In the case where only one sensor is to be placed the two problems, i.e. the next best view and the sensor placing problem, becomes quite similar. The aims are different however, given some previous observations the aim in the case of the next best view problem is to find a new point which maximizes the total amount of new information. The aim of this master's thesis is to use the previous observations to pick a point which maximizes the total amount of information observable from that single point, it does not matter whether the information is new or not.

Another connection is the case of the next best view problem but where there is previous knowledge about the scene available. In that case it is possible to use the proposed method to select the best views using simulation and the previous knowledge. The previous knowledge can for example be a model of area/volume of the scene coming from previous observations or a floor plan. This has three major advantages. Firstly, the planning can be done before the exploration starts. This means that the planning can be done on a separate location where more computational resources are available and thus saving time. The robot itself can also be made smaller and lighter as it does not require the computational power to calculate the next best view on board, which also can save time. Secondly, it is possible to observe more with fewer observations as several different configurations can be tested in simulation and the total amount of overlap of the views can be minimized. Lastly, the plan can also be executed in parallel if several robots are available.

In summary the two problems are similar, but the problem in this thesis deals with several points at once and the aim is to find a set of points which optimizes some function rather than dealing with one point at a time and iteratively building a model of some physical object.

AutoML

Automatic machine learning (AutoML) is a field of research which focuses on finding ways to automate several aspects of machine learning, with the end goal of making the process entirely automatic. The long term aim is to enable users to use machine learning without any knowledge of machine learning, by allowing the user to simply provide data and letting an automated system do the rest [14]. There are several problems which the field deals with, such as preprocessing the data, hyperparameter optimization, model selection and even architecture selection for neural networks.

One of the first AutoML-systems is the Auto-WEKA [14], it uses Bayesian optimization to solve the problem of combined algorithm selection and hyperparameter optimization problem (CASH) [25]. The latest version allows the user to tune both hyperparameters whilst also doing model selection with the press of a single button [18]. This is done while keeping both the training and evaluation data fixed.

The Auto-WEKA system itself is based on Sequential Model-Based Optimization for General Algorithm Configuration (SMAC) [29]. SMAC is a form of Bayesian optimization which uses random forest regression to model the relationship between an algorithm coupled with its parameters and its performance. The usage of random forests allows the optimization over algorithm configurations, consisting of both categorical and real values.

Google's AI research team are also working on AutoML but are primarily interested jointly selecting neural network architectures and their weights [6].

One focus of this master's thesis report is generating data for an already chosen model, while keeping the model and its hyperparameters fixed. This data generation can either be generating data automatically, e.g. with simulation or generating subsets from already existing data sets, or to guide manual data gathering. AutoML focuses more on where data has already been gathered and how to preprocess data, choose a model and tune the hyperparameters. The method proposed in this work might be seen as a step for generating the data which can later be used for AutoML. It might even be possible to jointly generate data and select the model, but that is outside the scope of this master's thesis.

Another focus of this master's thesis is to use the data generating function to better evaluate models. This evaluation of models might be useful in AutoML to better guide the model selection process as it makes it possible to reason about the model's performance in an entire area of interest.

Sensor networks

Sensor networks is a field of research dedicated to placing a large number of small sensors in real-world environments [1]. The sensors themselves are ideally inexpensive to make and dispensable, making it possible to have many sensors distributed in an environment. The applications of such sensor networks are vast, ranging from measuring seismic on active volcanoes [28], potato farming [19] and even to aerospace engineering [9].

Once the sensors have been deployed, a decentralized wireless ad hoc network will be setup in order to send the information towards the receiver. This means that each sensor will only communicate directly to other nodes which are physically close. The individual sensors can then use less power to transmit their measurements compared to if all sensors were transmitting directly to the receiver. The network is also dynamic such that if a sensor node fails or becomes unavailable for some reason, the network is able to reconfigure with little if any data loss.

In some cases it is possible to know the environment in which the sensors should be placed, e.g. in the case of placing sensors on an active volcano [28]. In that case the placement of sensors is similar to both the problem of placing wireless access points and the problem of placing camera/LIDAR sensors. It relates to both problems since there is a trade-off between how much of the environment the sensors are able to observe and how well it is able to transfer this information to the other nodes in the network.

In other cases, it is more energy efficient to use an autonomous vehicle to collect and relay the data. For example in underwater sensor networks, an autonomous underwater vehicle (AUV) might be used to collect the data from the sensor network and relay it back to a buoy at the water surface [12]. Another example is where an unmanned aerial vehicle is used to collect data from a set of ground nodes, as discussed in [29]. In both these cases, the problem of selecting what positions the unmanned vehicles should go to in order to reach as many of the nodes as possible can be treated similarly to the problem of placing wireless access points described earlier in this chapter.

Stochastic optimization over sets

There has been other works treating the problem of stochastic optimization over sets. The most recent, as far as the author could find, also uses Gaussian processes and a variant of Bayesian optimization to place weather sensors [11]. The authors of this paper uses a custom variant of the Earth mover's distance to compare different sets. The Earth mover's distance is a distance function between discrete probability distributions [23]. It can be intuitively understood by thinking about the two distributions as two dirt piles. The Earth mover's is then the least amount of dirt one has to move around in the first pile to make it look like the second.

This work was found late in the project and thus the distance they used is not included or compared to the other distance functions proposed in this report. However, the distance itself is the solution for a linear program, which means that it might be costly to evaluate. The number of operations required for calculating the Earth mover's distance is $\mathcal{O}(n^3)$ [11], while the two proposed distance functions in this master's thesis have the complexity $\mathcal{O}(n^2)$ and $\mathcal{O}((n!)^2)$ respectively. This places the Earth mover's distance right between the two proposed distance functions, at least in terms of complexity.

3 Theory

This chapter aims to both solidify what supervised machine learning is in the context of this master's thesis and define the Gaussian process and the Bayesian optimization algorithm.

3.1 Supervised machine learning

The aim of supervised machine learning is to find some function f between a domain X and a domain Y by analyzing preexisting examples, also called training data, consisting of pairs of (x, y), where $x \in X$ and $y \in Y$. The domain X is the input space of f and the domain Y the output space.

A classic example is the case of linear regression, where both the input and output space consists of all real numbers $\mathbb{R}$. We might have a few examples $[(0, 5), (1, 8), (5, 20)]$, where the first element of every pair is the value in the input domain and the second value is the value in the output domain. A reasonable model for this training data set is $y = f(x) = kx + m$, where $y \in Y$, $x \in X$ and both k and m are free variables. In this case the aim of supervised machine learning is to find k and m such that the model is able to predict the y given a value of x.

While this example is solvable by applying some basic algebra, not all problems are this easily solvable. For example, there might not be a single model that perfectly models the data or the data might contain noise from measurement errors. In these cases the concept of loss is used to explain how well the model performs in a point $x \in X$. The point-wise loss can be described as a function which takes two arguments, the first being the truth $y \in Y$ and the second being the model's the predicted value $\hat{y} = f(x, \mathcal{M})$. An example of a loss function is the mean square error:

$$\ell_{\text{mse}}(y, \hat{y}) = \|y - \hat{y}\|_2^2, \tag{3.1}$$

where $\|\dots\|_2$ is the ℓ^2 norm.

The training data set can be viewed as a subset of the set of all possible data $\mathcal{D}$. For many methods it is not advisable to use the same set of data for training and for evaluating the loss function. This is because these methods might overfit to the training data, meaning that they only perform well on the training data and poorly on the rest of the data points in $\mathcal{D}$. In these cases a separate finite subset of $\mathcal{D}$ called a test set $\mathcal{D}_{\text{test}}$ is used to more accurately measure the general loss over $\mathcal{D}$. Using a test set the total loss for a training data set $\mathcal{D}_{\text{train}}$ is defined

as a sum:

$$\mathcal{L}(\mathcal{D}_{\text{train}}) = \sum_{(x,y)\epsilon\mathcal{D}_{\text{test}}} \ell(y, f(x, \mathcal{M})), \tag{3.2}$$

where $\mathcal{M}$ is trained on the training data set.

In other cases a data generating function is available, making it possible to acquire new subsets of $\mathcal{D}$. While in some cases it is possible to use the entire set of $\mathcal{D}$ as both training and test data, it might be practically impossible due to the set being large or even infinite in size. However, in these cases it is still possible to use the data generating function to create representative subsets of $\mathcal{D}$ to use for training and testing.

3.2 Gaussian processes

Consider the problem of regression, where the aim is to predict y for a given x. The observations of y are noisy, more precisely $y = f(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ is some measurement noise. By placing a Gaussian process prior on the function f, it is possible to model the distribution over the function itself, as described by Rasmussen and Williams [22].

The Gaussian process is defined as:

$$f \sim \mathcal{GP}(m(.), k(.,.)), \tag{3.3}$$

where $m(.)$ is the mean function of the process and $k(.,.)$ is the kernel function.

For a set of points X_* of interest the distribution of the function values $f(X_*) = f_*$ is:

$$f_* \sim \mathcal{N}(\mu(X_*), K(X_*, X_*)), \tag{3.4}$$

which can be sampled and each sample will correspond to a possible function.

If some points X have known function values y, it is possible to incorporate these in the model and get the conditional distribution:

$$\begin{aligned} f_*|X, y, X_* &\sim \mathcal{N}(\bar{f}_*, \text{cov}(f_*)), \text{ where} \\ \bar{f}_* &= K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y \\ \text{cov}(f_*) &= K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*) \end{aligned} \tag{3.5}$$

where X_* can be any point or set of points of interest. In the case where X_* is a single point, its corresponding y-value will be distributed according to Gaussian distribution. Similarly, for a set of points the output will be distributed according to a multivariate Gaussian distribution.

The kernel describes how much two points influence each other. A common kernel is the square exponential kernel (SE) and it is defined as [10]

$$k_{\text{SE}}(x, x') = \sigma_f^2 \exp\left(-\frac{||x - x'||_2^2}{2\ell^2}\right), \tag{3.6}$$

where ℓ is the lengthscale and σ_f is the output variance, both being free parameters.

There are a few free variables to a Gaussian process, the measurement noise variance σ_n^2 as well as the kernel parameters. It is possible to set priors on all parameters and to do predictions by sampling the priors and then evaluating the posterior.

Another alternative is to minimize the negative marginal log likelihood, as described by Rasmussen and Williams [22]:

$$-\log p(y|X) = -\frac{1}{2} y^T (K + \sigma_n^2 I)^{-1} y - \frac{1}{2} \log |K + \sigma_n^2 I| - \frac{n}{2} \log 2\pi. \tag{3.7}$$

This is done by minimizing the equation above for a set of known data (X, y) by varying the values for the free variables.

Sparse Gaussian processes

Applying Gaussian processes to larger datasets is infeasible since it scales poorly. The kernel matrix K has dimensions $n \times n$, where n is the number of data points. This results in the storage of matrix growing with complexity $\mathcal{O}(n^2)$ and the inverse of the matrix requiring $\mathcal{O}(n^3)$ operations.

This can be approximated using a sparse Gaussian process which uses a set of inducing points X_m. The points X_m can be viewed as the set of m points in the input space which best represents the data overall, where m typically is much smaller than n. Note that this is not necessarily a subset of the total training data, but can be any points in input space. This approximation reduces the time complexity for predictions to $\mathcal{O}(nm^2)$, enabling the usage of larger data sets at the cost of some precision.

Titsias introduces an approach to jointly learn the position of a set of inducing points and the hyperparameters of the Gaussian process [26]. He selects the points by minimizing the Kullback-Leibler distance between the sparse Gaussian process and the Gaussian process by maximizing the variational lower bound of the true log marginal likelihood.

After having chosen the points, the predictive distribution for the resulting sparse Gaussian process has the form:

$$p(y_*|y) = N(y_*|m_y^q(x_*), k_y^q(x_*, x_*) + \sigma^2), \tag{3.8}$$

where:

$$\begin{aligned} m_y^q(x) &= K_{xm}K_{mm}^{-1}\sigma^{-2}K_{mm}\Sigma K_{mn}y \\ k_y^q(x,x') &= k(x,x') - K_{xm}K_{mm}^{-1}K_{mx'} + K_{xm}\Sigma K_{mx'} \\ \Sigma &= (K_{mm} + \sigma^{-2}K_{mn}K_{nm})^{-1}. \end{aligned} \tag{3.9}$$

Forcing vector element order invariance

If the order of the elements of the input vector does not matter, a special kernel can be constructed to model this explicitly. This allows points to be close if one of their permutations are close, even though they might be far away according to a regular kernel, for example the squared exponential kernel. A method of constructing such a kernel is described by Duvenaud in his PhD thesis [10]:

$$k_{\text{exact}}(x,x') = \sum_{g\in G}\sum_{g'\in G} k(g(x), g'(x')), \tag{3.10}$$

where G is a set of functions, where each function changes the order of the elements of x in some way. All permutations which are equivalent should be represented by individual functions in G. For example, if the order of the first two elements does not matter the set would be $G = \{g_1, g_2\}$, where:

$$\begin{aligned} g_1(\{x_1, x_2, ...\}) &= \{x_2, x_1, ...\} \\ g_2(\{x_1, x_2, ...\}) &= \{x_1, x_2, ...\}. \end{aligned} \tag{3.11}$$

In general if we have a set of point $[x_1, x_2, ..., x_n]$ whose order should be made independent we need to include every permutation of the set of points in G. Therefore, the size of the set G grows factorially with the size of the set of points, more precisely $|G| = n!$.

3.3 Bayesian optimization

Bayesian optimization is a method for optimizing a function with as few function evaluations as possible [24]. It is useful when the function to be optimized is expensive to evaluate, either time-consuming or costly. Internally it uses a Gaussian process as a surrogate function to model the function which is being optimized. At each iteration one point of the actual function is evaluated and added to the Gaussian process.

What point to evaluate at each iteration is determined by an acquisition function. The acquisition function only depends on predictions made by the Gaussian process, meaning it is relatively inexpensive to evaluate since it does not need to evaluate the expensive function which the Gaussian process surrogate model. It also determines the trade-off between exploration and exploitation. Exploration being selecting points in mostly unknown areas, meaning areas with high variance, in order to learn more about the general trends and, ultimately, to find good areas for exploitation. Conversely, exploitation means evaluating points close to areas known previously as good, where the variance is lower and the mean is higher.

The Bayesian optimization algorithm is described step by step in algorithm 1.

Algorithm 1 Bayesian Optimization

Input: Function to be maximized f; max iteration N; function input space X; acquisition function α

Result: Best estimate x^* of the highest function value

1: **for** $i \leftarrow 1$, N **do**
2: $\quad \mathcal{GP} \leftarrow$ Gaussian Process regression with data $\langle x_i, y_i \rangle_{j=1}^{i-1}$
3: $\quad$ Select $x_i \in \arg\max_{x \in X} \alpha(x, \mathcal{GP})$
4: $\quad y_i \leftarrow f(x_i)$
5: **end for**
6: **return** $x^* \leftarrow \arg\max_{x_i \in \{x_1, \ldots, x_N\}} y_i$

One commonly used acquisition function is the expected improvement (EI) and it is defined as [16]:

$$E[I(x)] = E[\max(f_{\min} - Y, 0)], \tag{3.12}$$

where $f_{\min}$ is the lowest encountered value so far. The improvement $I(x) = \max(f_{\min} - Y, 0)$ is zero for values which are higher than $f_{\min}$ and positive for values which are lower, indicating an improvement. Note that this definition holds whenever a function is being minimized and that improvement can be defined similarly when the function is being maximized. Remember, if using a Gaussian process the predictive distribution for a single point x^* is a Gaussian distribution: $f^*|X, y, x^* \sim N(\mu(x), \sigma^2(x))$. When expressing the expected improvement when using a Gaussian process as a surrogate function, the lowest mean prediction $f^*_{\min}$ is used instead of $f_{\min}$. The expected improvement then has the following closed form:

$$E[I(x)] = (f^*_{\min} - \mu(x))\Phi\left(\frac{f^*_{\min} - \mu(x)}{\sigma(x)}\right) + \sigma(x)\varphi\left(\frac{f^*_{\min} - \mu(x)}{\sigma(x)}\right), \tag{3.13}$$

where $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$ is the standard normal density function and Φ the standard normal cumulative distribution function.

Another acquisition function is based on confidence bounds and was first introduced as lower confidence bound (LCB) by Cox and John [7]. However, it is most commonly referred to upper confidence bound (UCB) and is written[4]:

$$\text{UCB}(x) = \mu(x) + \kappa\sigma(x), \tag{3.14}$$

where $\mu(x)$ and $\sigma(x)$ is the predicted mean and predicted standard deviation at point x, and κ is a tuning parameters which allows for controlling the trade-off between exploration and exploitation. High values for κ means more exploration, as points with high predicted standard deviation are prioritized and lower values means more exploitation, as points with high predicted mean are prioritized.

4 Method

This chapter has two main purposes. The first is to explain the proposed method for finding a solution for sensor placement, wireless access point placement and data selection. The second is to explain how the method is tested and validated.

4.1 Relationship between set of Euclidean points and performance

This section formalizes a method for maximizing some performance measure by selecting a set of points from a closed Euclidean space. The idea is to first extend the Gaussian process model to sets of points and then to use the extended Gaussian process to do Bayesian optimization.

Extending the Gaussian process to a set of points

A Gaussian Process is most commonly used to model the relationship between a single point x and some value y. It is possible to extend the Gaussian Process to model the relationship between a set of points of fixed size by concatenating all the points into one large point. If the set has n elements and every point is a point in the domain $\mathbf{D}$ with cardinality c the set of points is written $\mathbf{D}^{n\times c}$. When concatenated, a set of points can be written as a single point in $\mathbf{D}^{nc}$.

Also, the order of the points in a set does not matter and thus a kernel invariant to the order of the points can be constructed. This is achieved by defining the set G as follows:

$$
\begin{aligned}
g(x_1, x_2, x_3, \ldots, x_n) &= (x_1, x_2, x_3, \ldots, x_n) \\
g(x_1, x_2, x_3, \ldots, x_n) &= (x_1, x_3, x_2, \ldots, x_n) \\
g(x_1, x_2, x_3, \ldots, x_n) &= (x_2, x_1, x_3, \ldots, x_n) \\
g(x_1, x_2, x_3, \ldots, x_n) &= (x_2, x_3, x_1, \ldots, x_n) \\
g(x_1, x_2, x_3, \ldots, x_n) &= (x_3, x_1, x_2, \ldots, x_n) \\
g(x_1, x_2, x_3, \ldots, x_n) &= (x_3, x_2, x_1, \ldots, x_n) \\
&\vdots
\end{aligned}
$$

and then expanding each point x_i to a set of elements $\{x_{i,1}, x_{i,2}, \ldots, x_{i,c}\}$. That way, the kernel becomes invariant to the order of the points but not to the order of the elements in the

individual points. The number of elements in G is $n!$, meaning that its size only depends on the total number of points n and is not affected by the cardinality c of the individual points. Unless otherwise specified the underlying kernel, the one being summed, is the squared exponential kernel.

Approximating the permutation invariant kernel

The size of G grows fast as the number of points increases. Recall the formula for the exact permutation invariant kernel, described in equation 3.10. As the resulting exact permutation invariant kernel is calculated by summing over the set G twice, the total number of sums is $(n!)^2$.

It is possible to make an approximation by only including pair-wise permutations:

$$k_{\text{approx}}(x, x') = \sum_i \sum_j K(x_i, x_j). \tag{4.1}$$

This results in a kernel consisting of a sum of n^2 kernels, where n is the number of points in the set. Another approximation is the sum of the previous kernel and the standard squared exponential kernel:

$$k_{\text{sum}}(x, x') = k_{\text{SE}}(x, x') + k_{\text{approx}}(x, x'). \tag{4.2}$$

The creation and evaluation of both these kernels have the complexity $\mathcal{O}(n^2)$, while the original permutation invariant kernel has the complexity $\mathcal{O}((n!)^2)$.

Applying the new kernel

Recall the previous definition of $\mathcal{X}_{\text{free}}$ as the entire configuration space where each element is a set consisting of position for each LIDAR sensor, camera, access point etc. Using the invariant kernel or its approximation it is possible to do Bayesian optimization with $\mathcal{X}_{\text{free}}$ as the input space and the wanted performance measure as the output space. This is described further in algorithm 2.

Algorithm 2 Bayesian set optimization

Input: Performance measure function f; max iterations N; set of all configurations $\mathcal{X}_{\text{free}}$; acquisition function α

Result: Best configuration c^* for sufficiently large N

1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: $\quad c_i \leftarrow \arg\max_{c \in \mathcal{X}_{\text{free}}} \alpha(c, \mathcal{GP})$
3: $\quad \mathcal{GP} \leftarrow$ Gaussian process regression with data $\langle c_i, f(c_i)\rangle_{j=1}^{i-1}$ using the permutation invariant kernels or one of its approximations
4: **end for**
5: **return** $c^* \leftarrow \arg\max_{c_i \in \{c_1, \ldots, c_N\}} f(c_i)$

This algorithm can also be further extended to do data selection where data is generated from a data generating function f_{gen}. Instead of evaluating a single function, such as total signal strength or measuring the total observed area, the chosen set is used as input to the data generating function f_{gen} which output in turn is used as training data for a supervised machine learning model. The loss function is assumed to be known, e.g. when evaluating the model on a known representative test set. The resulting algorithm is shown in algorithm 3.

4.2 Approximating the loss surface and its integral

The aim of this section is to estimate the total loss over the entire free space $\mathbf{X}_{\text{free}}$. There are two problems which needs to be resolved. Firstly, the integral is difficult to calculate analytically. Secondly, the loss function itself is expensive to calculate.

Algorithm 3 Bayesian set optimization for data selection

Input: Model to be optimized $\mathcal{M}$; known loss function $\mathcal{L}$; data generating function f_{gen}; max iteration N; set of all configurations $\mathcal{X}_{\text{free}}$; acquisition function α
Result: Best training set c^* for training the model

for $i \leftarrow 1, \ldots, N$ **do**
  Select $c_i \in \arg\max_{c \in \mathcal{X}_{\text{free}}} \alpha(c, \mathcal{GP})$
  Generate data set $X_i \leftarrow \{(x, y)$, where $(x, y) \leftarrow f_{\text{gen}}(p)\}_{p \in c_i}$
  $f(., \mathcal{M}) \leftarrow$ retrain the model on data set X_i
  Evaluate the model $L_i \leftarrow \mathcal{L}(\mathcal{M})$
  $\mathcal{GP} \leftarrow$ Gaussian process regression with data $\langle c_j, L_j \rangle_{j=1}^{i-1}$ using the permutation invariant kernels or one of its approximations
end for
return $c^* \leftarrow \arg\max_{c_i \in \{c_1, \ldots, c_N\}} f(c_i)$

Remember the total loss is the integral over $\ell(., f(., \mathcal{M}))$:

$$\mathcal{L}(f(., \mathcal{M})) = \int_{\mathbf{X}_{\text{free}}} \ell(p, f(x, \mathcal{M}))dp. \tag{4.3}$$

This integral is difficult to calculate, as free space might have holes in it or consist of several rooms. It can be rewritten by splitting the free space into several smaller areas which can be more easily integrated:

$$\mathcal{L}(f(., \mathcal{M})) = \sum_{a \in \mathbf{X}_{\text{free}}} \int_a \ell(p, f(x, \mathcal{M}))dp. \tag{4.4}$$

However, the analytical integral of a Gaussian process is non-trivial and thus saved for future work. Instead the loss function is approximated by a Riemann sum [21] by splitting $\mathbf{X}_{\text{free}}$ into several small regions Δ_k:

$$\mathcal{L}(f(., \mathcal{M})) \approx \sum_k \ell(p_k) a(\Delta_k), \tag{4.5}$$

where $a(\Delta_k)$ denotes the area or volume of the region Δ_k and p_k is a point in the region Δ_k.

In the case where $\mathbf{X}_{\text{free}}$ is discrete and has non-infinite size, it does not need to be approximated. Rather, the loss can be calculated per discrete element and summed together:

$$\mathcal{L}(f(., \mathcal{M})) = \sum_{p \in \mathbf{X}_{\text{free}}} \ell(p, f(., \mathcal{M})). \tag{4.6}$$

In both cases, the loss function is still expensive to evaluate since it is defined in terms of the data generator function f_{gen}:

$$\ell(p, f(., \mathcal{M})) = \ell(y, f(x, \mathcal{M})), \text{ where } (x, y) = f_{\text{gen}}(p). \tag{4.7}$$

The function can be estimated by placing a GP prior on it.

In order to minimize the total number of function evaluations, a new acquisition function for Bayesian optimization is proposed. Bayesian optimization is designed for finding a maximum of a given function while also reducing the number of function evaluations. The maximum is not of interest here, but rather minimizing the uncertainty about the loss function and the resulting integral. Recall the UCB acquisition function from equation 3.14, which has an explicit trade-off between exploration and exploitation κ. By setting the value very high, the function will prioritize points with high uncertainty and thus minimize the uncertainty of the function estimation. For sufficiently large value of κ the acquisition function will only depend on the predicted standard deviation in the point. Thus, the pure exploration acquisition function is defined as:

$$\alpha_{\text{PE}}(x) = \sigma(x), \tag{4.8}$$

where $\sigma(x)$ is the predicted standard deviation in the point x.

Using such an acquisition function the regular Bayesian optimization algorithm can be used to estimate the loss function, resulting in the approximation denoted $\hat{\ell}$. Note that a regular squared exponential kernel is sufficient here, as only one point needs to be chosen at the time.

Once the loss function has been estimated, the integral can be calculated as described in equation 4.5 but with the predicted mean $\mu_{\hat{\ell}}$ instead of ℓ:

$$\mathcal{L}(f(.,\mathcal{M})) \approx \sum_k \mu_{\hat{\ell}}(p_k) a(\Delta_k), \tag{4.9}$$

The Gaussian process predictive distribution for a single point p is a regular normal distribution, $\hat{\ell}(p) \sim \mathcal{N}(\mu_{\hat{\ell}}(p), \sigma^2_{\hat{\ell}}(p))$. This means that the sum from the case where $\mathbf{X}_{\text{free}}$ is assumed to be discrete, equation 4.6, can be rewritten:

$$\begin{aligned}\mathcal{L}(f(.,\mathcal{M})) &\approx \sum_k \hat{\ell}, \text{where } \hat{\ell} \sim \mathcal{N}(\mu_{\hat{\ell}}(p_k), \sigma^2_{\hat{\ell}}(p_k)) \\ &\sim \mathcal{N}(\mu, \sigma^2), \text{where } \mu = \sum_{p \in \mathbf{X}_{\text{free}}} \mu_{\hat{\ell}}(p) \text{ and } \sigma^2 = \sum_{p \in \mathbf{X}_{\text{free}}} \sigma^2_{\hat{\ell}}(p).\end{aligned} \tag{4.10}$$

As a summary, the proposed method for approximating performance surfaces and their integrals is shown in algorithm 4 for the continuous case and in algorithm 5 for the discrete case.

Algorithm 4 Loss approximation algorithm for continuous $\mathbf{X}_{\text{free}}$

Input: Model to be evaluated $\mathcal{M}$; point-wise loss function ℓ; max iteration N; free space $\mathbf{X}_{\text{free}}$; acquisition function α
Result: Approximation of total loss $\hat{\mathcal{L}}$

1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: Select $x \in \arg\max_{x \in \mathbf{X}_{\text{free}}} \alpha_{PE}(x, \mathcal{GP})$
3: $\mathcal{GP} \leftarrow$ Gaussian process regression with data $\langle x_j, \ell(x_j) \rangle_{j=1}^{i-1}$
4: **end for**
5: **return** $\hat{\mathcal{L}} \leftarrow \sum_k \mu_{\hat{\ell}}(p_k) a(\Delta_k)$, where $\mu_{\hat{\ell}}$ is the predicted mean from the Gaussian process $\mathcal{GP}$.

Algorithm 5 Loss approximation algorithm for discrete $\mathbf{X}_{\text{free}}$

Input: Model to be evaluated $\mathcal{M}$; point-wise loss function ℓ; max iteration N; free space $\mathbf{X}_{\text{free}}$; acquisition function α
Result: Normal distribution describing estimate of total loss $\hat{\mathcal{L}}$

1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: Select $x \in \arg\max_{x \in \mathbf{X}_{\text{free}}} \alpha_{PE}(x, \mathcal{GP})$
3: $\mathcal{GP} \leftarrow$ Gaussian process regression with data $\langle x_j, \ell(x_j) \rangle_{j=1}^{i-1}$
4: **end for**
5: **return** $\mathcal{N}(\mu, \sigma^2)$, where $\mu = \sum_{p \in \mathbf{X}_{\text{free}}} \mu_{\hat{\ell}}(p)$ and $\sigma^2 = \sum_{p \in \mathbf{X}_{\text{free}}} \sigma^2_{\hat{\ell}}(p)$, where $\mu_{\hat{\ell}}(.)$ and $\sigma^2_{\hat{\ell}}(.)$ is the predicted mean and variance for the Gaussian process $\mathcal{GP}$.

4.3 Data selection with approximated loss surfaces

By bringing the two previous sections together, it is possible to do data selection when only the point-wise loss function ℓ is available. The result is shown in algorithm 6. The idea is to use the Bayesian optimization for data selection algorithm as a basis and to replace the loss evaluation with the loss evaluation approximation described in the previous section.

Algorithm 6 Bayesian set optimization for data selection with estimated loss surface

Input: Model to be optimized $\mathcal{M}$; point-wise loss function ℓ; data generating function f_{gen}; max iteration N, set of all configurations $\mathcal{X}_{\text{free}}$; free space $\mathbf{X}_{\text{free}}$; acquisition function α
Result: Best training set c^* for training the model
1: **for** $i \leftarrow 1, \ldots, N$ **do**
2: Select $c_i \in \arg\max_{c \in \mathcal{X}_{\text{free}}} \alpha(c, \mathcal{GP})$
3: Generate data set $X_i \leftarrow \{(x, y)$, where $(x, y) \leftarrow f_{\text{gen}}(p)\}_{p \in c_i}$
4: $f(., \mathcal{M}) \leftarrow$ retrain the model of the data set X_i
5: $\hat{\mathcal{L}}_i \leftarrow$ estimate total loss with $f(., \mathcal{M})$ over $\mathbf{X}_{\text{free}}$ according to algorithm 4 or 5 depending on whether $\mathbf{X}_{\text{free}}$ is continuous or discrete
6: $\mathcal{GP} \leftarrow$ Gaussian process regression with data $\langle c_j, \hat{\mathcal{L}}_j \rangle_{j=1}^{i-1}$
7: **end for**
8: **return** $c^* \leftarrow \arg\min_{c_i \in \{c_1, \ldots, c_N\}} \hat{\mathcal{L}}_i$

4.4 Implementation

The aim of this part of the method is to explain the properties of the two implemented simulators as well as describing what frameworks were used during the master's thesis project.

Ray casting

A small 2D ray casting simulator was implemented for testing the case of LIDAR sensor placement. It takes a set of 2D points and performs ray casting in a simulated environment and for every point in the set it will try to draw a line from the point and to every yet unobserved cell in the scene. For every line, the first occupied cell it intersects will be marked as observed if has not yet been observed. This means that once an occupied cell has been intersected, everything about the cell is known and it there is no need to scan it again. The result from the ray casting is a 2D grid with the same size as the map, containing a true value if the cell was observed and false otherwise. Figure 4.1 shows what cells the rays passes through for two different configurations.

The ray casting can be seen as a function $f : \mathbb{R}^{2 \times n} \to \mathbb{B}^{r \times c}$, where n is the number of 2D points, $\mathbb{B}$ is the set of $\{\text{true}, \text{false}\}$ and r, c are the number of discretized rows and columns of the simulated environment.

The loss for a set of points can be defined as:

$$\mathcal{L}(X) = (\#\text{observable cells}) - (\#\text{observed cells}), \tag{4.11}$$

where $X \in \mathbb{R}^{2 \times n}$ is a set of 2D points.

Two different maps were used for experiments with the ray casting simulator. Both are represented using black and white images, where black pixels represents observable cells (can be considered walls) and white are unobservable cells (empty space). The first is a simpler and smaller apartment and the second is a larger and more complex house, both can be seen in figure 4.2.

If we use a single point and place it on every possible location and evaluate the function, we get a map of how many cells can be observed from every position. This can be seen in figure 4.3 for the apartment and in figure 4.4 for the larger, more complex building.

Wireless access point placement

Another simulator was implemented for evaluating the application of Bayesian optimization to wireless access point placement. The simulator is similar to the ray casting simulator, both use the same maps and is built on discrete underlying grids. In this simulator, a set of access points are to be placed in the environment.

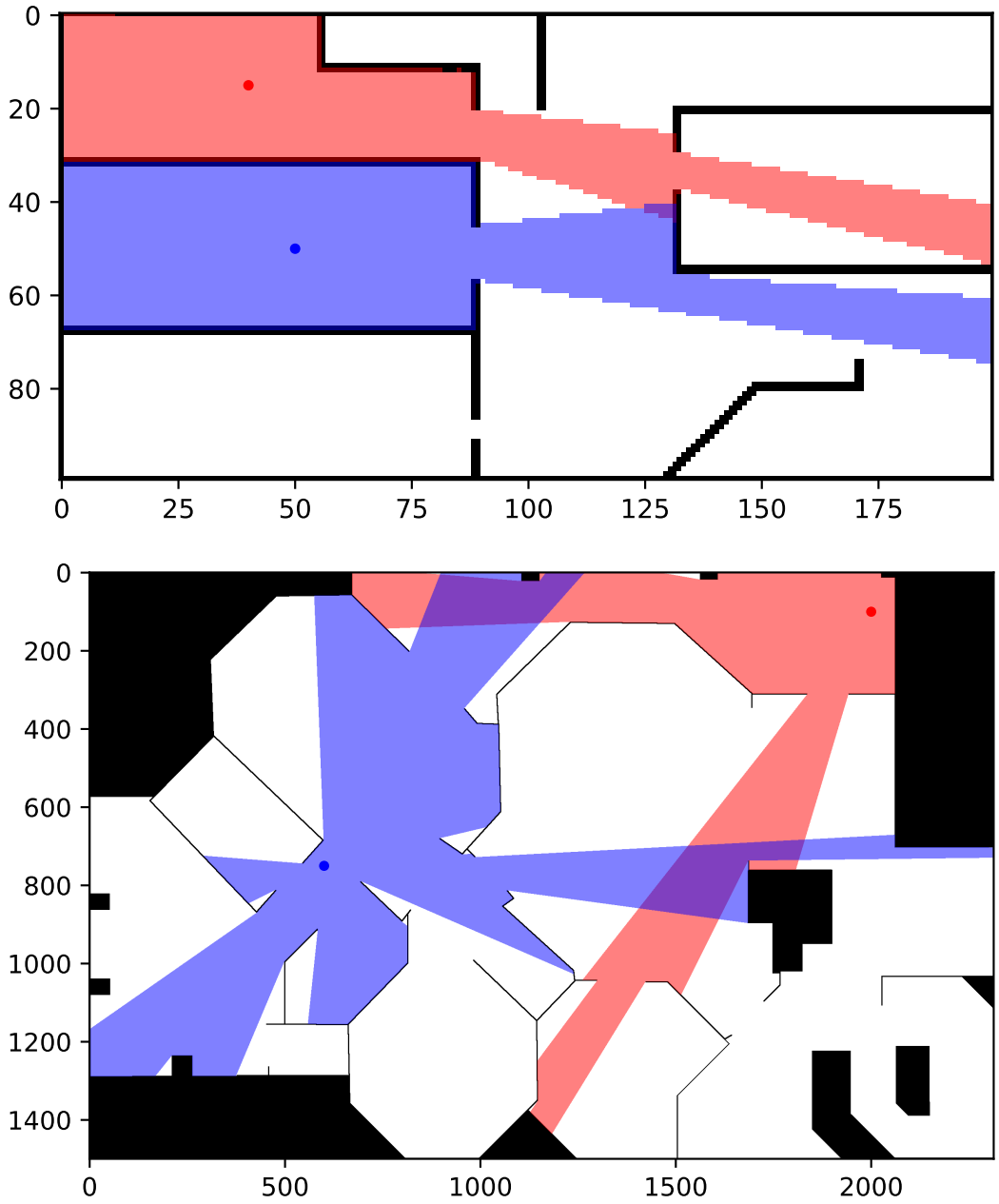


Figure 4.1: Visualization of the ray casting simulator for the two different buildings. The positions of the respective LIDAR sensors are shown with a dot. The free cells are colored by the rays which passes through them, the colors are the same as the dot of the sensor from which the rays originate.

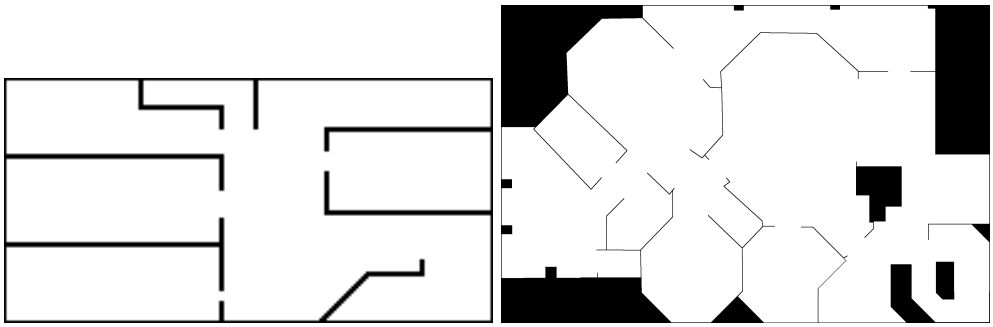


Figure 4.2: The two environment used in ray casting, the smaller simpler apartment on the left and the larger and more complex house on the right. Black cells are occupied and white cells are free. Rays will only collide with the black occupied cells.

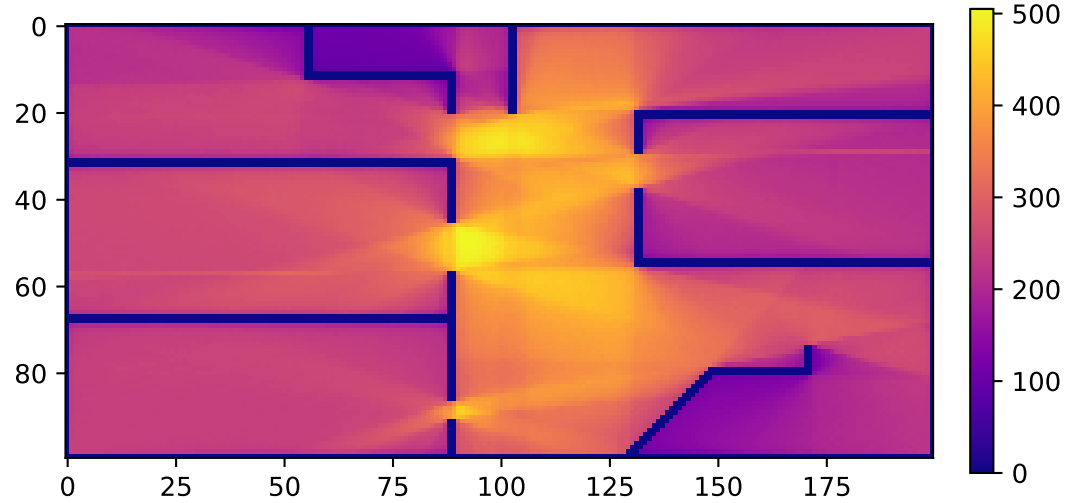


Figure 4.3: Total number of observable cells for each possible position, brighter color represents a higher value. A higher value corresponds to more observable cells being visible and thus lower loss.

The signal strength between a measurement point p and an access point x, both lying in $\mathbf{X}_{\text{free}}$, is defined as:

$$s_{\text{pairwise}}(p, x) = \frac{1}{\|x - p\|_2^2 + 1} * 0.75^{(\#\text{walls})}, \tag{4.12}$$

where #walls is the number of occupied cells the signal passes through if going in a straight line between p and x. An example of this function is shown in figure 4.5.

As described in the problem description, a device can typically only connect to one access point at the time. This means that once a set of access points have been placed, the signal strength can be evaluated in a point p is defined as:

$$s(p, c) = \max_{x \in c} s_{\text{pairwise}}(p, x). \tag{4.13}$$

Figure 4.6 shows an example where the signal strength have been measured after two access points have been placed, one at $(250, 1000)$ and one at $(2000, 900)$, in the larger building.

If the total signal strength were to be maximized directly, it might be optimal to place several access points in the larger room since it might give high signal strength in that region and poor signal strength in the rest of the building. Keeping this in mind, the loss function is

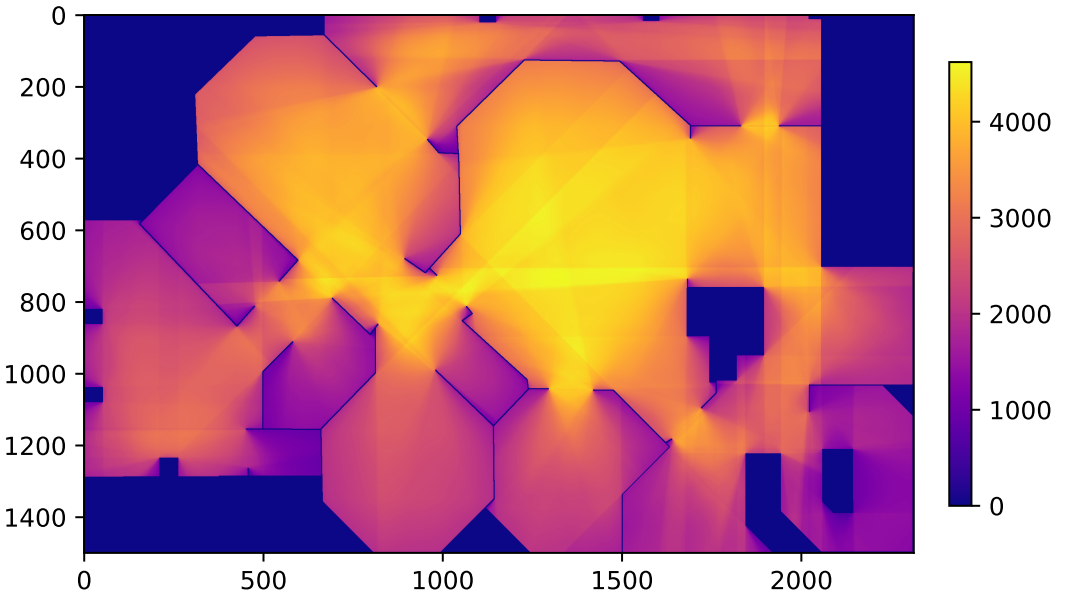


Figure 4.4: Total number of observable cells for each possible position, brighter color represents a higher value. A higher value corresponds to more observable cells being visible and thus lower loss.

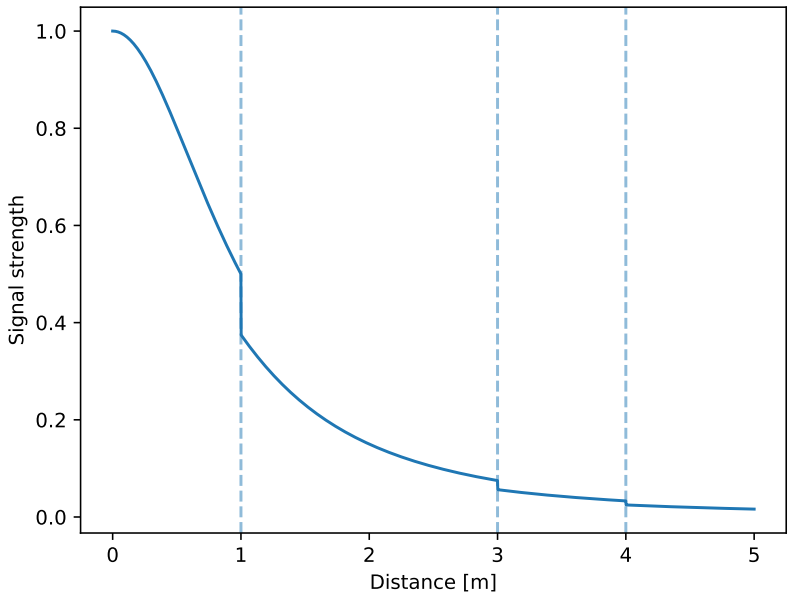


Figure 4.5: Example of signal strength function. Three walls have been placed and their positions are denoted by dashed vertical lines.

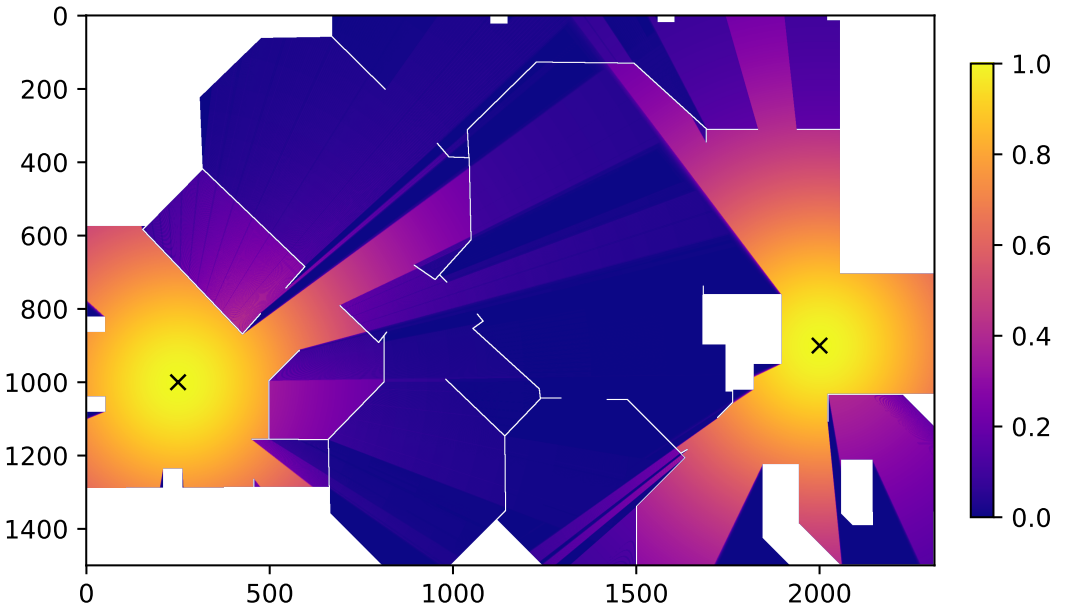


Figure 4.6: Example of signal strength in the case of wireless access point placement. Brighter color corresponds to higher signal strength and darker to lower.

defined as the square of the inverted signal:

$$\mathcal{L}(p, c) = \frac{1}{(s(p, c) + 1)^2}. \tag{4.14}$$

This loss function is low in points where the signal is high and high wherever the signal strength is low. The square penalizes low signal strengths by making the small values even smaller and thus promoting having decent performance everywhere rather than good signal strength in some regions while having no coverage in others. The addition of one is to avoid division by zero if the signal is zero. Figure 4.7 shows this loss for the same access point configuration as shown previously.

If wireless coverage is assumed to be equally important in all of the free space $\mathbf{X}_{\text{free}}$, the total loss for a configuration c is:

$$\mathcal{L}(c) = \sum_{x \in \mathbf{X}_{\text{free}}} \mathcal{L}(x, c). \tag{4.15}$$

Note that since the simulator is grid-based a sum is sufficient, if the simulator used a continuous coordinate system $\mathbf{X}_{\text{free}}$ would be continuous and an integral would be used instead of a sum.

Gaussian processes and Bayesian optimization

All Gaussian processes are implemented using GPflow, a Python package which allows for the construction, optimization and evaluation of Gaussian processes [20]. As a basis for the Bayesian optimization the package GPflowOpt [17] is used.

Both the exact and approximated permutation invariant kernel was implemented as kernels for the GPflow package. The pure-exploration acquisition function was implemented as an acquisition function for the GPflowOpt package. All figures were generated using Matplotlib [13].

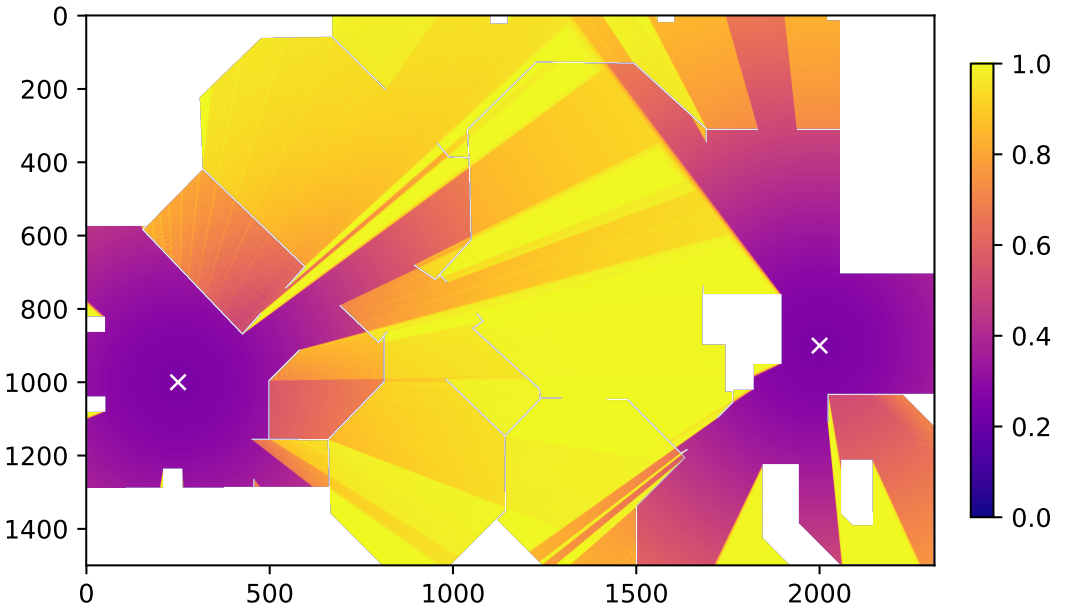


Figure 4.7: Example of loss in the case of wireless access point placement. Brighter color corresponds to higher loss and darker to lower.

4.5 Experiments

This section describes the two experiments used to validate the proposed methods. The first experiment compares the different kernel options by running both simulators and varying the number of points placed. The second approximates the loss surface of the wireless access point placement simulator and its integral.

Evaluating and comparing kernels

The aim of this experiment is to evaluate the different choices for kernel functions when modeling the function from a set of points to a value using a Gaussian process. The kernels of interest are:

- Squared exponential kernel, equation 3.6
- Exact permutation kernel, equation 3.10
- Approximated permutation kernel, equation 4.1
- Approximated permutation kernel plus regular squared exponential kernel, equation 4.2

The size of the set of points used as input to the ray casting simulator is varied in order to see how different kernels scale. Algorithm 2, the Bayesian set optimization algorithm, is used to find the optimal placement for the points for every combination of kernel and set size, running for 20 iterations for each combination. The expected improvement acquisition function was used for picking the next set at each iteration.

The experiments ran for two different loss functions; one for each simulator. See equation 4.11 and equation 4.15 for the ray casting and wireless access point placement loss functions. For the ray casting loss function the smaller map was used and for the wireless access point placement loss function the larger map was used.

Evaluating pure exploration for loss surface estimation

The aim of this experiment is to approximate loss surface of the final configuration from the wireless access point placement problem. Algorithm 5, loss approximation for discrete $\mathbf{X}_{\text{free}}$, is used for estimating the loss surface and its integral, resulting in the estimate $\hat{\ell}(.)$ and its sum $\hat{\mathcal{L}} \sim \mathcal{N}(\mu, \sigma^2)$. The configuration used is the best configuration with 5 access points placed from the previous experiment. The algorithm ran for 500 iterations, meaning it evaluated the true loss function 500 times. The loss surface is compared to the ground truth by calculating the cell-wise mean absolute error:

$$e(p) = |\ell(p) - \mu_{\hat{\ell}}(p)|^2. \tag{4.16}$$

The sum is also compared to the sum of the ground truth loss surface.

5 Result

The aim of this chapter is to show the result of the experiments proposed in the later part of the method chapter. Each section corresponds to one experiment.

5.1 Evaluating and comparing kernels

Figure 5.1 shows the loss on the Y-axis and the number of placed LIDAR sensors on the X-axis, with one line for each kernel choice. Similarly, figure 5.3 also shows the loss on the Y-axis and the number of placed wireless access points on the X-axis.

The time taken for the entire Bayesian optimization, including running the simulator once for each of the 20 iterations, is shown in figures 5.2 and 5.4 for the ray casting and wireless access point placement experiment respectively. Due to time constraints, runs with the exact permutation invariant kernel was limited to 6 points.

5.2 Evaluating pure exploration for loss estimation

Figure 5.5 shows the estimated loss surface, where the blue dots mark where the ground truth has been observed. The true loss surface is shown in figure 5.6. Figure 5.7 shows the cell-wise absolute error. The predicted total loss $\hat{\mathcal{L}} = \mathcal{N}(\mu = 1362460, \sigma = 2664.6)$, while the true total loss $\mathcal{L} = 1352958$. The probability density function of predicted distribution is compared to the true total truth in figure 5.8.

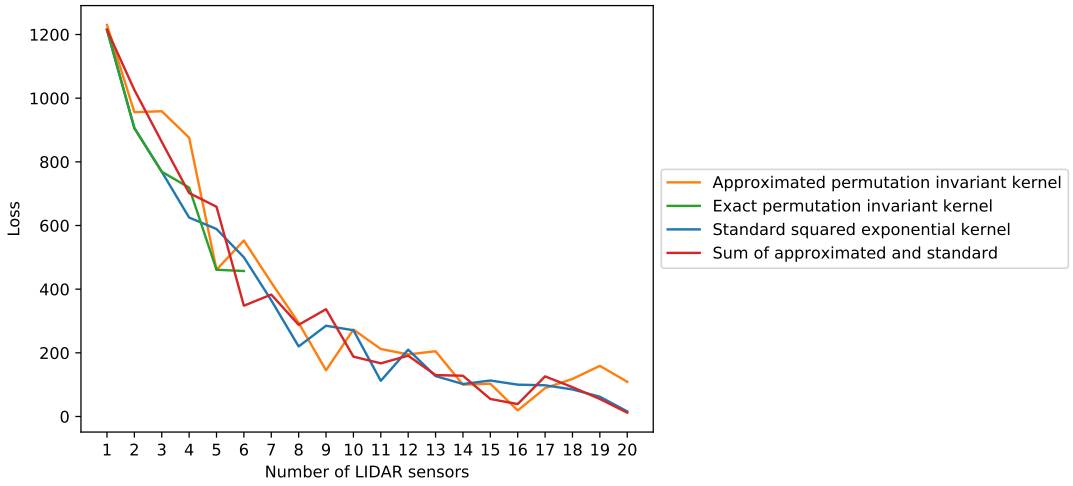


Figure 5.1: Comparison of the different kernel alternatives for using Bayesian optimization to place LIDAR sensors in the simulated apartment-sized environment. Each line corresponds to a different kernel. The Y-axis shows the loss, that is the total number of non-observed cells, for the resulting configuration and the X-axis is the total number of LIDAR sensors.

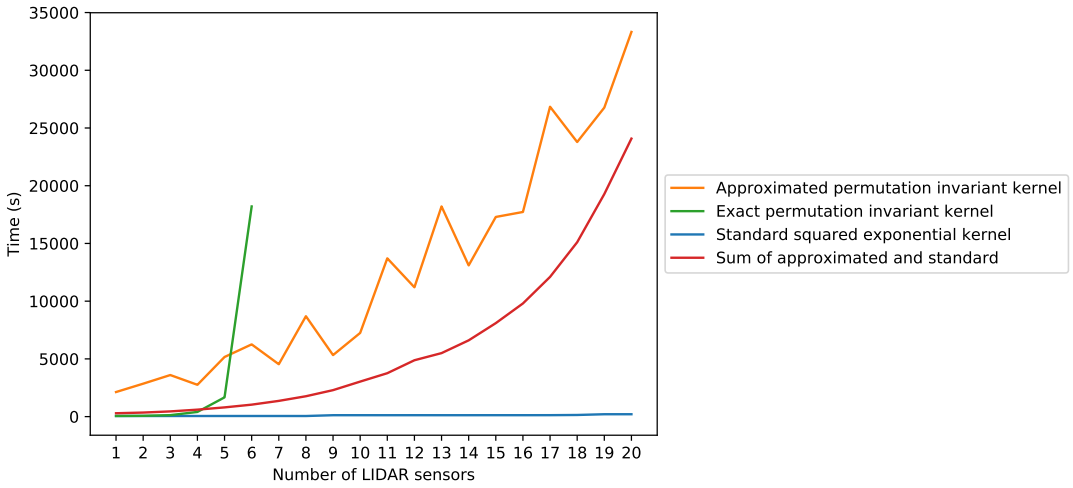


Figure 5.2: Comparison of the different kernel alternatives for using Bayesian optimization to place LIDAR sensors in the simulated apartment-sized environment. Each line corresponds to a different kernel. The Y-axis is total number of seconds required for running the algorithm and the X-axis is the total number of LIDAR sensors.

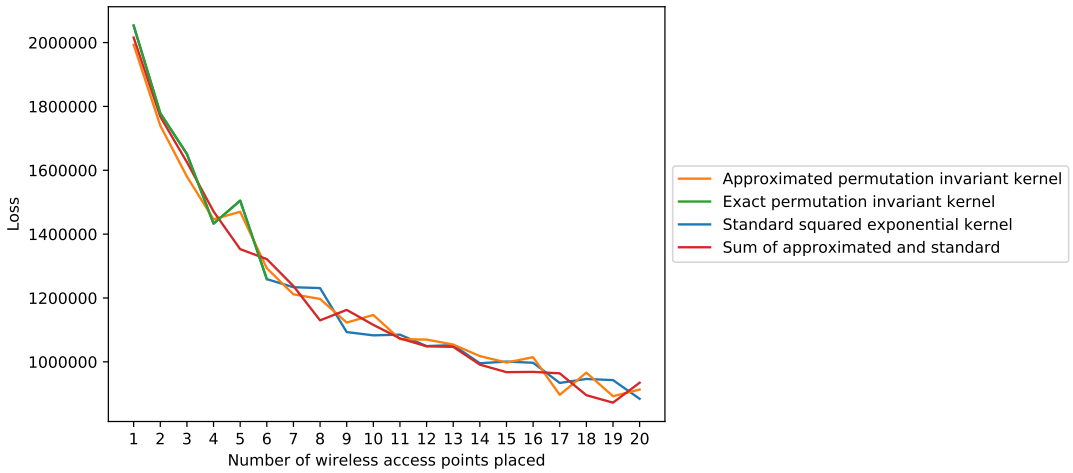


Figure 5.3: Comparison of the different kernel alternatives for using Bayesian optimization to place wireless access points in the simulated house-sized environment. Each line corresponds to a different kernel. The Y-axis shows the loss for the resulting configuration and the X-axis is the total number of LIDAR sensors.

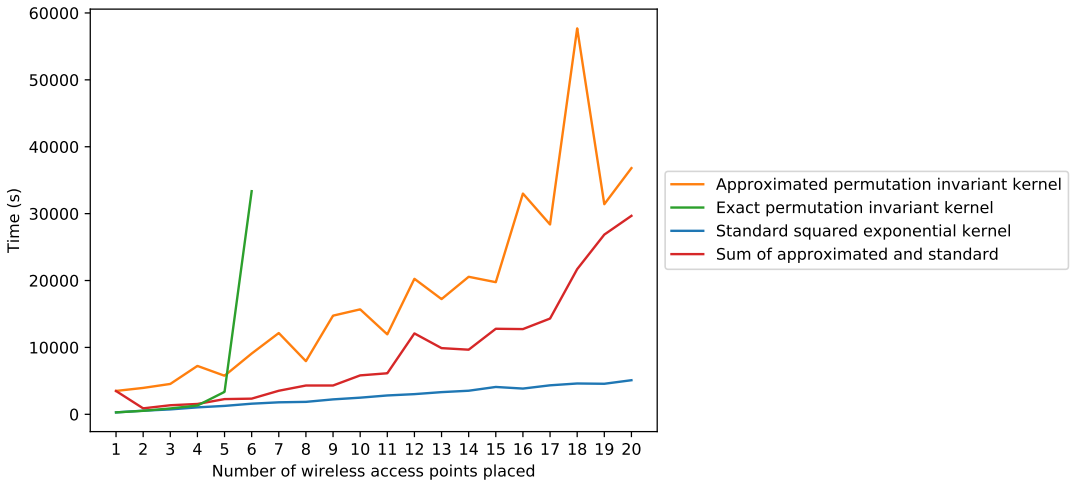


Figure 5.4: Comparison of the different kernel alternatives for using Bayesian optimization to place wireless access points in the simulated house-sized environment. Each line corresponds to a different kernel. The Y-axis is total number of seconds required for running the algorithm and the X-axis is the total number of LIDAR sensors.

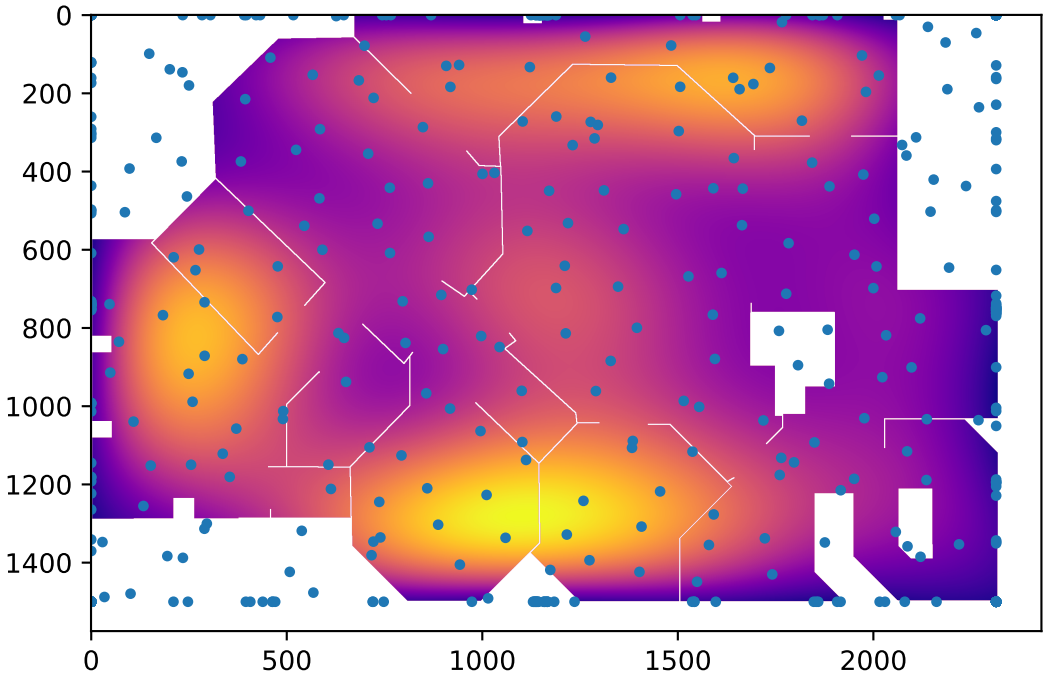


Figure 5.5: Estimated loss surface. The blue dots are where the algorithm has chosen to observe the underlying loss surface.

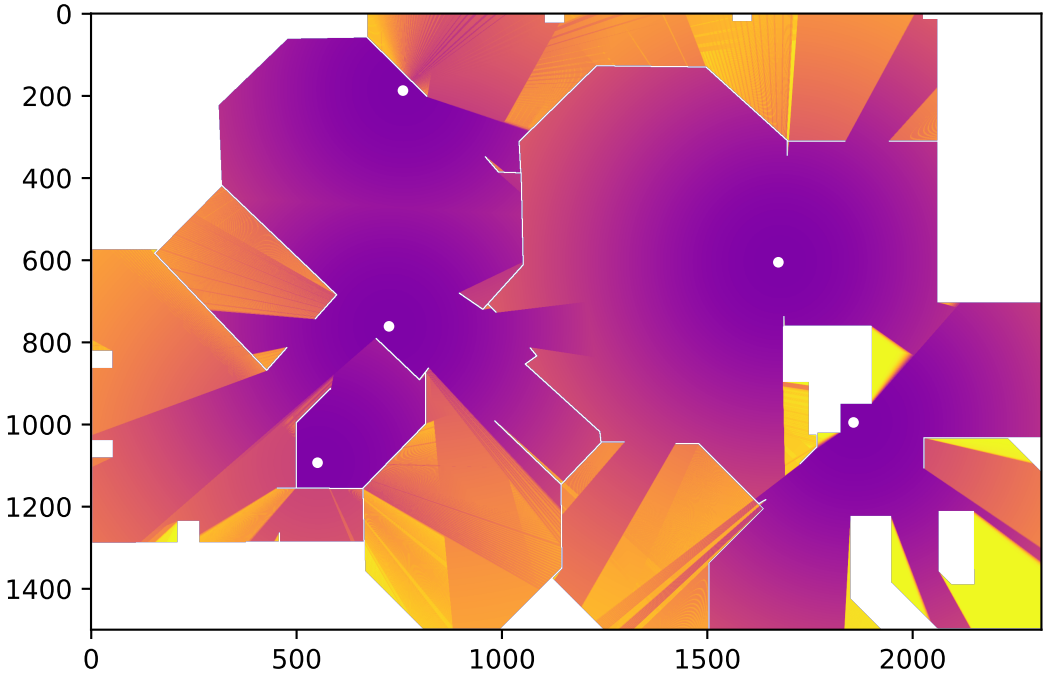


Figure 5.6: Ground truth loss surface. The white dots show where wireless access points have been placed.

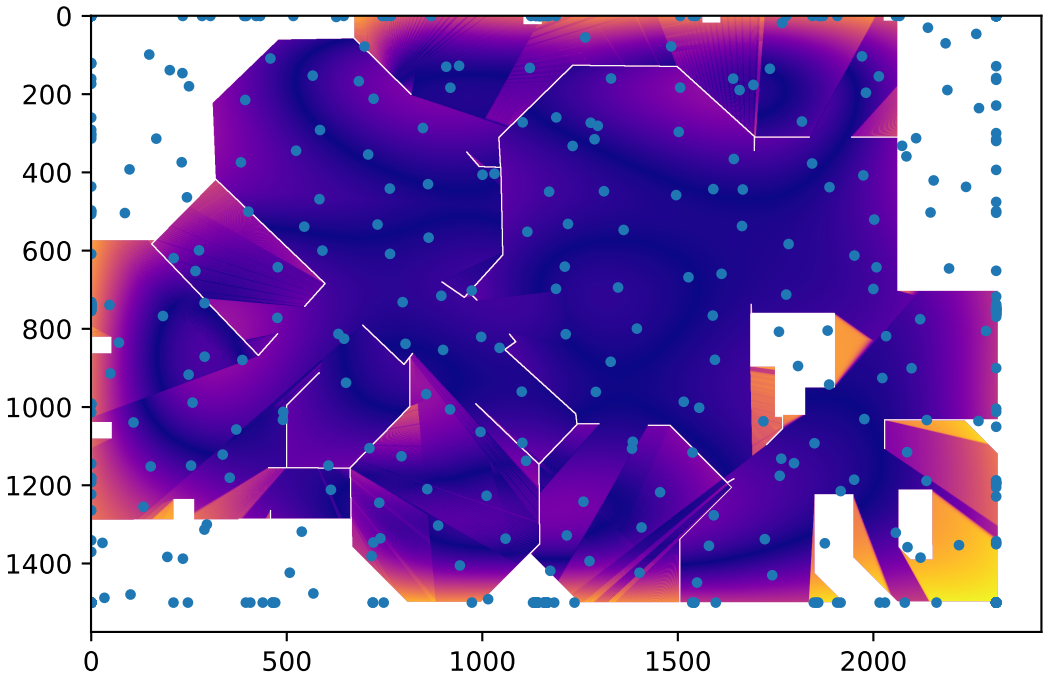


Figure 5.7: Cell-wise absolute error comparing the estimate and the ground truth loss surface. The blue dots are where the algorithm has chosen to observe the underlying loss surface.

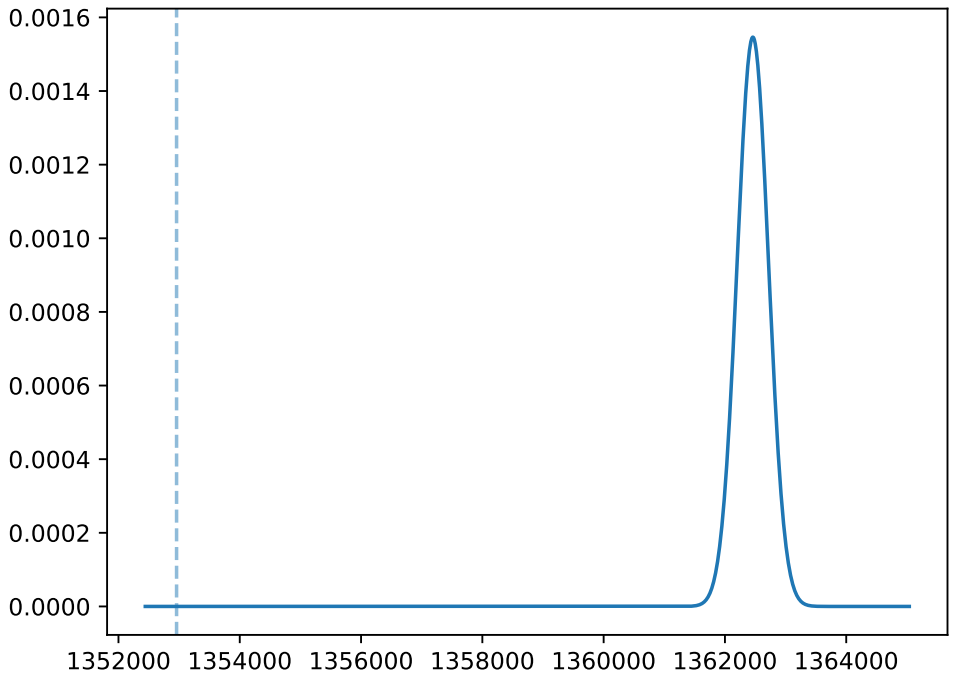


Figure 5.8: Predicted total loss shown as normal distribution. True total loss is shown as dashed line.

6 Discussion

The aim of this chapter is to discuss the method and result, as well as reflect on the impact of the work in a larger context.

6.1 Result

The discussion of the result is split into two parts, one for each experiment.

Evaluating and comparing kernels

There is no notable difference between the kernels in terms of performance, the main difference lies in the total time needed for running the Bayesian optimization algorithm. The time required for running the algorithm using the exact permutation invariant kernel grows fast as the number of points in the configuration set increases. This is expected, as the time complexity is $\mathcal{O}((n!)^2)$. The rapid growth makes the kernel infeasible for real-world applications when more than six points needs to be chosen or placed.

It does not seem to be much of a difference between using a permutation invariant kernel and a regular squared exponential kernel. The cause might be some implementation errors. However, it is unlikely that the kernels would perform roughly as well as the squared exponential kernel but not requiring the same amount of time. Another reason might be the complexity of the loss functions optimized, or rather the lack of complexity in the two loss functions tested. There might be a larger difference in performance if the kernels were applied to a real-world experiment with far more points.

The sum of the approximated permutation invariant kernel and the squared exponential kernel performed faster than just using the approximated permutation invariant kernel. This is an interesting result since the time-complexities are the same for the two and that the sum kernel contains more calculations to create the kernel. The reason for this could be that the sum might have smoother gradients, which makes it possible to find the maximum of the acquisition function in less time.

Evaluating pure exploration for loss estimation

The resulting total loss mean prediction is quite close to the true total loss, but the certainty measure is off as the actual total loss lies far outside the predicted distribution.

The main problem with the result is the number of points placed outside of the free space $\mathbf{X}_{\text{free}}$, as seen in figure 5.5. This comes from a limitation in GPflow framework where the input space can only be chosen to be simple ranges where the environment used in the simulator is more complex. Loss is defined as 0 for points not in $\mathbf{X}_{\text{free}}$, which results in low loss "bleeding" in from the walls. More specifically, this results in lower loss along walls when compared to the ground truth loss surface, shown in figure 5.6, and ultimately results in a high cell-wise absolute error along walls and in corners, shown in figure 5.7. This also results in many points being placed inside the walls, because there is a large change in loss moving from a position inside a wall and a position inside $\mathbf{X}_{\text{free}}$.

The actual loss lies far outside of normal distribution of the predicted total loss, as shown in figure 5.8. This indicates that the estimate is overly confident. The overconfidence in the estimate might have been caused by the points being placed outside of the free space $\mathbf{X}_{\text{free}}$ as well, since the loss for points inside walls does not vary at all, the predicted variance is zero in these areas. This in itself is not a problem, but points with zero predicted variance might influence points in $\mathbf{X}_{\text{free}}$ which are close to walls and might cause overconfidence in the predicted mean there.

A possible solution to this would be to use a sampling based approach to maximizing the acquisition function, which would allow for picking only points which lie in $\mathbf{X}_{\text{free}}$. First sample a large number of positions from $\mathbf{X}_{\text{free}}$, evaluate the acquisition function in these points and select the point with the largest acquisition value as the next point for the Bayesian optimization algorithm. The downside here would be that the number of positions sampled from $\mathbf{X}_{\text{free}}$ will affect the quality of the choice of evaluation point and the number would need to be tuned on a case-by-case basis.

Once the issue has been fixed, the method will work better with fewer iterations needed, because less function evaluations will be wasted on in-wall evaluations. It is not clear whether this will solve the overconfidence of the distribution, but it is hard to tell without first fixing the issue of evaluating outside of $\mathbf{X}_{\text{free}}$.

6.2 Method

In this section several aspects of the method will be discussed.

Frameworks

The GPflow framework is a useful tool for testing different kernels and coupled with GPflowOpt it is possible to do Bayesian optimization. However, there are a few drawbacks using this solution.

As of writing the report, the Bayesian optimization library is several versions behind GPflow and using it requires manually installing a prerelease version in order to get compatibility. This version might contain unknown bugs, but no major bugs were observed during the master's thesis project.

Moreover, it is difficult to get insight into some parts of the Bayesian optimization algorithm. By default the algorithm can be run for fixed number of iterations using a single method call. If the intermediate result is of interest the method can be called with the iterations argument set to 1 and calling the method once for each wanted iteration. Whenever a new point is added to the underlying Gaussian process, the parameters of the kernel is re-optimized in order to fit the data as well as possible. There is no way to access the result of this optimization to know how many iterations it took. These issues makes debugging difficult and time consuming.

The main bottleneck when using Gaussian processes with the permutation invariant kernel is memory usage. It is possible that this is due to the number of kernels required in the sum to create the exact permutation invariant kernel. The reason for the high memory usage is that this sum is implemented in the TensorFlow computation graph directly, resulting in an enormous graph. While the graph might be useful for automatically calculating the gradients when optimizing the parameters of kernel, it might be possible to save time by simply calculating the sum directly without using a computation graph and then use a gradient-free optimization algorithm.

Bayesian optimization

In this master's thesis, a regular kernel is used when the exact and approximate permutation invariant kernels are constructed and when using Bayesian optimization for estimating loss-surfaces. While there are many different kernel choices available for Gaussian processes, the squared exponential kernel was the only kernel tested. This to reduce the number of parameters in the experiments and to save time. Note that both the exact and approximate permutation invariant kernel can use any underlying kernel and that its performance might vary depending on that choice. Evaluating which kernels are more or less useful for different applications is left for future work.

Similarly, there are many choices for acquisition functions when using Bayesian optimization. Due to time constraints the only acquisition function tested was the expected improvement kernel, with exception of the pure exploration kernel tested for loss surface estimation.

A possible alternative to the pure exploration acquisition function would be one which uses the expected reduction in uncertainty of the loss surface. Remember, the idea of the pure exploration acquisition function is to use the Bayesian optimization algorithm to only pick points where the predicted variance is high and thus reduce the overall uncertainty. If using an acquisitions function which uses the expected reduction in uncertainty, this behavior would be more explicitly defined and perhaps perform better. This new acquisition function can be further extended by taking into account that the estimated loss function is later used for calculating a Riemann sum. The acquisition function can then be the estimated total reduction in the local area where the Riemann sum is calculated.

The use of a permutation invariant kernel can be used in scenarios with non-homogeneous sensors, where different sensors have different capabilities, by encoding the sets into G separately. For example, if we want to place 5 sensors with one type of coverage and 5 sensors with a different type of coverage, in that case a possible configuration will consist of 5 positions for the first type of sensor and 3 positions of the second type of sensor. The set G will then contain all permutations of the first 5 elements as well as all permutations of the last 3 elements, but keep the order between the two different sensor types. This might be useful if one has access to a couple of cameras with wide angle lenses and a few with a more narrow field of view.

Simulation

There are both advantages and disadvantages to using a simplified simulation compared to real-world experiments. While the simulators are simplified in comparison to doing real-world experiments of the problem of camera/LIDAR placement and the problem of placing wireless access points, there are a few important properties that hold for both the simulation and the real-world experiments.

Firstly, the order of the points does not matter. Secondly, placing two points close to each other will generally not result in better performance since the improvement given by the two points will overlap.

Lastly, the space is believed to be similarly smooth in both cases. In the data selection case it should be reasonable to assume that the total amount of information obtainable from a certain position is somewhat correlated with the total area of uniquely observable walls.

There are also some advantages to using a simulated environment. It takes less time and resources to place points in a simulated environment than either placing sensors or training a machine learning model. Not only does this allow for faster debugging but it also allows for testing different problem sizes and seeing how the proposed method scales.

Ideally the method would be tested on both a simulated environment, preferably early in the project to iron out bugs and to validate the method, and in real-world experiments, further validating both the method and result from the simulations. However, due to unforeseen problems with setting up the real-world experiments causing delays, this was not possible within the scope of this master's thesis.

Note that the formulas for the wireless access point placement simulator are not based on a real physical model, but rather a well educated guess about signals from wireless access points. While the formulas depend on the intuition, the proposed loss function should still be applicable to any function which operates on a set and returns either a loss or some other performance measure.

Moreover, the aim of the wireless access point placement simulator is for it to have a more complex loss function which can only be evaluated point wise. The performance measure for the wireless access point placement problem depends both on the position x_m where the measurement is taken and the configuration c. Since it is a simulation it is possible to evaluate the loss function exactly as well as sampling for a position x_m. This allows testing and validating the performance/loss surface estimation separately from the Bayesian optimization applied to sets of positions.

6.3 Supervised learning

The original intent of this project was to apply this proposed method to supervised machine learning. The idea was to use the framework UnrealCV, which contains simulated environments suitable for training and evaluating computer vision systems. The framework would be used as the data generating function, taking a coordinate in 3D space and generating a training data pair of a 2D image and a semantically segmented image. The goal was to train a convolutional neural network to predict the semantically segmented image from the RGB image. The training set for the convolutional neural network would be chosen using the Bayesian set optimization algorithm, using the loss estimation algorithm to estimate the loss over the entire space of possible coordinates.

Unfortunately the framework has many issues. Without going into too much detail, as this is slightly out of scope, the problems took several weeks of time to resolve, each problem revealing new issues. This ultimately resulted in the idea being abandoned due to the last issue being so time consuming it was outside of the time limit of this master's thesis project. Therefore, the proposed algorithm is tested part by part and not applied to supervised machine learning. In the future a more suitable framework might be available, making it possible to test and evaluate the proposed algorithms fully.

However, it is still possible to discuss the proposed algorithm since its parts were evaluated. It is possible to approximate the total loss over a closed space containing all possible training data by using a Gaussian process with a custom pure-exploration kernel and simple numerical integration. Using either the permutation invariant kernel or the proposed approximation, it is possible to select a set of points which minimizes or maximizes a function. This function can be loss or some other performance measure, as shown in the method and result chapter.

While the report does not test this directly, it shows that it is possible to use a loss function to select a subset of possibly infinite training data. Therefore, since the proposed algorithms consists of combining both these parts it should be feasible to use the full algorithm to generate a representative training data set $\mathcal{D}_{\text{train}}$ which yields low overall loss in the whole free space $\mathbf{X}_{\text{free}}$.

6.4 The work in a wider context

While this master's thesis mostly concerns simulated data there are a couple of ways it relates to a wider context.

The method proposed and evaluated in this master's thesis is computationally heavy and thus affects the environment by using electricity and, by extension, natural resources. Some experiments in this master's thesis require either the full use of a GPU or a CPU, both drawing around 100 to 200 watts, for several hours or even days. However, as this is a general method with many possible applications it is possible that the method might be used in the future to make some improvement which ultimately results in a net gain for the environment. For example, if this method in some way, either directly or indirectly, enables machine learning to make autonomous taxi services a reality it might reduce the total number of cars in the world.

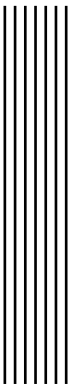
The methods proposed in this report can be used for placing surveillance cameras in an Orwellian society. However, while the proposed methods might be used to improve the placement of the cameras and maximize the total coverage with fewer cameras, the methods themselves are not bad. Rather, as with many types of technology and research, it depends on the wielder whether it is used for good or bad. Currently an estimated 53.6% of the worlds population have access to internet [27]. A positive example of where the methods proposed in this report can be used for good is by helping bring internet to the last 46.4% of the population. This could be achieved be applying the proposed methods to placing internet infrastructure, e.g. wireless base stations, more efficiently.

7 Conclusion

The two main research questions have been partially answered and discussed in the previous chapter. In summary, the proposed pure exploration based algorithm has been shown to work for exploring, estimating and ultimately integrating a complex loss function. Furthermore, the proposed Bayesian set optimization algorithm has been shown to work for selecting a set of points in order to minimize a complex loss function. The loss function in both these cases is not the one originally intended which was based on a supervised machine learning use case, but rather a simulated environment with similar characteristics. With a few adjustments, the proposed methods should be suitable for applications on the originally intended use case of supervised machine learning.

The approach of applying Bayesian optimization to sets rather than individual points is general and should be applicable to problems outside the scope of this master's thesis. Pure exploration for Bayesian optimization is also a general method, which can be used to estimate functions when the maximum of the function is not of interest but rather the function itself.

Future work could focus on improving the acquisition function for the continuous loss surface estimation, by taking into account that the estimated surface will be used in a Riemann sum. It could also look into ways to better approximate the permutation invariant kernel. Finally, applying the proposed method for selecting data in a supervised machine learning setting is perhaps the most interesting potential future work related to this master's thesis.



Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. "Wireless sensor networks: a survey". In: *Computer Networks* 38.4 (2002), pp. 393–422. ISSN: 1389-1286.

[2] D. Bergström, M. Tiger, and F. Heintz. "Bayesian optimization for selecting training and validation data for supervised machine learning". In: *31st Swedish AI Society Workshop (SAIS)*. 2019.

[3] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. "Receding Horizon "Next-Best-View" Planner for 3D Exploration". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. May 2016, pp. 1462–1468.

[4] Eric Brochu, Vlad M. Cora, and Nando de Freitas. "A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning". In: (Dec. 2010).

[5] C. Connolly. "The determination of next best views". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. Mar. 1985, pp. 432–435.

[6] Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. "AdaNet: Adaptive Structural Learning of Artificial Neural Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 874–883.

[7] D. D. Cox and S. John. "A statistical method for global optimization". In: *[Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics*. Oct. 1992, 1241–1246 vol.2.

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[9] M. Drobczyk and A. Lübken. "Novel wireless protocol architecture for intra-spacecraft wireless sensor networks (inspaWSN)". In: *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*. Dec. 2018, pp. 89–94.

[10] David Duvenaud. "Automatic model construction with Gaussian processes". PhD thesis. University of Cambridge, 2014.

[11] R. Garnett, M. A. Osborne, and S. J. Roberts. "Bayesian Optimization for Sensor Set Selection". In: *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. IPSN '10. Stockholm, Sweden: ACM, 2010, pp. 209–219. ISBN: 978-1-60558-988-6.

[12] John Heidemann, Milica Stojanovic, and Michele Zorzi. "Underwater sensor networks: applications, advances and challenges". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 370.1958 (2012), pp. 158–175.

[13] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[14] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automatic Machine Learning: Methods, Systems, Challenges*. In press, available at http://automl.org/book. Springer, 2018.

[15] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. "Deep Projective 3D Semantic Segmentation". In: *Computer Analysis of Images and Patterns*. Ed. by Michael Felsberg, Anders Heyden, and Norbert Krüger. Cham: Springer International Publishing, 2017, pp. 95–107. ISBN: 978-3-319-64689-3.

[16] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions". In: *Journal of Global Optimization* 13.4 (Dec. 1998), pp. 455–492. ISSN: 1573-2916.

[17] Nicolas Knudde, Joachim van der Herten, Tom Dhaene, and Ivo Couckuyt. "GPflowOpt: A Bayesian Optimization Library using TensorFlow". In: *arXiv preprint – arXiv:1711.03845* (2017).

[18] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 826–830.

[19] K. Langendoen, A. Baggio, and O. Visser. "Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture". In: *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. Apr. 2006, 8 pp.-.

[20] Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo Le'on-Villagr'a, Zoubin Ghahramani, and James Hensman. "GPflow: A Gaussian process library using TensorFlow". In: *Journal of Machine Learning Research* 18.40 (Apr. 2017), pp. 1–6.

[21] Arne Persson and Lars-Christer Böiers. *Analys i flera variabler*. Studentlitteratur, 2005.

[22] Carl Edward Rasmussen and Christopher KI Williams. *Gaussian processes for machine learning*. The MIT Press, 2006. ISBN: 0-262-18253-X.

[23] Y. Rubner, C. Tomasi, and L. J. Guibas. "A metric for distributions with applications to image databases". In: *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*. Jan. 1998, pp. 59–66.

[24] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 2951–2959. URL: `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf`.

[25] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms". In: *Proc. of KDD-2013*. 2013, pp. 847–855.

[26] Michalis Titsias. "Variational Learning of Inducing Variables in Sparse Gaussian Processes". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Ed. by David van Dyk and Max Welling. Vol. 5. Proceedings of Machine Learning Research. Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, 16–18 Apr 2009, pp. 567–574.

[27] International Telecommunication Union. *ICT facts and figures 2017*. 2017.

[28] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh. "Deploying a wireless sensor network on an active volcano". In: *IEEE Internet Computing* 10.2 (Mar. 2006), pp. 18–25. ISSN: 1089-7801.

[29] Cheng Zhan, Yong Zeng, and Rui Zhang. "Energy-Efficient Data Collection in UAV Enabled Wireless Sensor Network". In: *CoRR* abs/1708.00221 (2017).