MULTI-SUPPORT GAUSSIAN PROCESSES FOR CONTINUOUS OCCUPANCY MAPPING

CARLOS E. O. VIDO

A thesis submitted in fulfillment
of the requirements for the degree of
Master of Philosophy

THE UNIVERSITY OF
SYDNEY

SCHOOL OF INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGY
THE UNIVERSITY OF SYDNEY

12 November 2015

DECLARATION

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the University or other institute of higher learning, except where due acknowledgement has been made in the text.

Carlos E. O. Vido

Sydney, 12 November 2015

ABSTRACT

Carlos E. O. Vido
The University of Sydney

Master of Philosophy
12 November 2015

Multi-Support Gaussian Processes for Continuous Occupancy Mapping

Robotic mapping enables an autonomous agent to build a representation of its environment based upon sensorial information. In particular, occupancy mapping aims at classifying regions of space according to whether or not they are occupied—and, therefore, inaccessible to the agent. Traditional techniques rely on discretisation to perform this task.

The problems tackled by this thesis stem from the discretisation of continuous phenomena and from the inherently inaccurate and large datasets typically created by state-of-the-art robotic sensors. To approach this challenge, we make use of statistical modelling to handle the noise in the data and create *continuous* occupancy maps.

The proposed approach makes use of Gaussian processes, a non-parametric Bayesian inference framework that uses kernels, to handle sensor noise and learn the dependencies among data points. The main drawback is the method's computational complexity, which grows cubically with the number of input points.

The contributions of this work are twofold. First, we generalise kernels to be able to handle inputs in the form of areas, as well as points. This allows groups of spatially correlated data points to be condensed into a single entry, considerably reducing the size of the covariance matrix and enabling the method to deal efficiently with large amounts of data. Then, we create a mapping algorithm that makes use of Gaussian processes equipped with this kernel to build continuous occupancy maps.

Experiments were conducted, using both synthetic and publicly available real data, to compare the presented algorithm with a similar previous method. They show it to be comparably accurate, yet considerably faster when dealing with large datasets.

I am putting myself to the fullest possible use,
which is all I think that any conscious entity can ever hope to do.

— *HAL 9000*

Thank you for allowing me to participate. It was remarkable.
— Lt. Cmdr. Data

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family and my parents, Carlos and Rosa Maria, for all the support and encouragement they've provided me throughout my life. It is thanks to them that I was able to dedicate myself fully to my studies, setting me on the path that brought me to this point.

I am grateful to my supervisor, Fabio Ramos, for the encouragement and trust deposited in me. I extend this gratitude to all the teachers that, throughout my life, inspired and helped me in my pursuit of knowledge. As somebody once wrote, *"Teachers affect eternity, no one can tell where they influence stops."*

I also thank all the good people at the International House for their hard work and commitment to making the life of residents easier. Living on the opposite side of the planet would have been a lot more dull and lonely without them.

Words cannot express what all my friends mean to me. My dear friends who wait for me in Brazil, and who I could always count on. My good friends in Australia, who made this far land feel less lonely and almost like home. And all the friends I now have scattered all over the planet. I hope I have the time and energy to visit every single one of you, at least once. You all enrich my life, and I'm thankful. (P.S.: No names, I'm on a word count limit.)

Last but not least, I would like to thank my beloved girlfriend Amanda, for putting up with me for so long. She helped me more than she will ever know.

CONTENTS

FIGURES

TABLES

ALGORITHMS

SYMBOLS AND NOTATION

GENERAL NOTATION

x	variable
$\boldsymbol{v}$	vector
A	matrix
$\mathcal{D}$	set

MATHEMATICS AND PROBABILITY

$\sim$	distributed according to
$\in$	belongs to
$\propto$	proportional to
$\to$	tends to
I	identity matrix
$\boldsymbol{v}^\top, A^\top$	transpose of vector $\boldsymbol{v}$ or matrix A
$\lvert\boldsymbol{v}\rvert$	length of vector $\boldsymbol{v}$
$\mathrm{tr}(A)$	trace of matrix A
$\mathbb{R}$	set of real numbers
$\mathbb{N}$	set of natural numbers (strictly positive integers)

$\Gamma(x)$	gamma function
$\Phi(x)$	normal cumulative distribution function
$g = O(f)$	function g grows asymptotically no faster than function f
$p(a)$	probability of event a
$p(a \mid b)$	probability of event a given event b
$\mathbb{E}(x)$	expected value of x
$\mathrm{cov}(\boldsymbol{v})$	covariance of $\boldsymbol{v}$
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$	Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ

GAUSSIAN PROCESSES

$x, \mathbf{x}, X$	input point(s)
$x_*, \mathbf{x}_*, X_*$	test point(s)
$f(\mathbf{x}), \mathbf{f}$	ground truth
σ_n^2	sensor noise
$\mathbf{y}$	noisy measurement of $\mathbf{f}$, $\mathbf{y} = \mathbf{f} + \sigma_n^2$
$f(\mathbf{x}_*), \mathbf{f}_*$	prediction
$k(\mathbf{x}, \mathbf{x}')$	covariance function
$K(X, X')$	covariance matrix
Σ, Σ_*	covariance matrix of $\mathbf{f}, \mathbf{f}_*$
$\boldsymbol{\theta}$	hyperparameters of the model
σ_0^2	bias

σ_f^2	signal variance
$l, \mathbf{l}$	length-scale

ACRONYMS

BFGS	Broyden-Fletcher-Goldfarb-Shanno method
DEM	digital elevation map
FPR	false positive rate
GP	Gaussian process
GPOM	Gaussian process occupancy map
GPOMIK	Gaussian process occupancy map using integral kernels
iid	independent and identically distributed
ML	machine learning
MSK	multi-support kernel
MS-GP	multi-support Gaussian process
MS-GPOM	multi-support Gaussian process occupancy map
NN	neural network
OGM	occupancy grid map
PCA	principal component analysis
PSD	positive semi-definite
ROC	receiver operating characteristic
SLAM	simultaneous localisation and mapping
TPR	true positive rate

Chapter 1

INTRODUCTION

Robotic mapping concerns itself with enabling an autonomous agent to build a representation of its environment based solely on sensorial information. As a fundamental part of any interaction between a robot and its surroundings, advances in mapping impact numerous other fields.

This chapter discusses the motivations, objectives and contributions of this thesis.

Excuse me sir, but might I inquire as to what's going on?
— C3PO

1.1 MOTIVATION

In the dawn of their existence, autonomous systems were confined to industrial environments where they worked in assembly lines performing repetitive tasks. This was an early presage of the kinds of tasks they would be performing in modern society: nowadays, robots are widely used for tasks perceived as too dirty (e. g. sewage pipe maintenance), dangerous (e. g. defusing bombs or exploring hazardous environments) or dull (e. g. domestic robots) for humans.

In modern times, there is a growing trend to remove robots from controlled industrial environments and bring them to the outside world. Such is the case not only with self-driven cars, but also vacuum cleaning or security robots, for example. In all of those situations, the robots are in unknown environment, where they must be able to localise themselves while negotiating moving obstacles (such as living beings or other vehicles) and obscured regions.

In most of these applications, it is impossible to pre-program robots with the maps they need. Rather, they should be equipped to build this maps themselves, based on available sensory information. The need for robust and reliable mapping algorithms to provide robots with a good notion of what their surroundings are like is greater than it has ever been. With mapping being a very fundamental task in robotics, improvements in this field can impact several applications.

1.2 PROBLEM DESCRIPTION

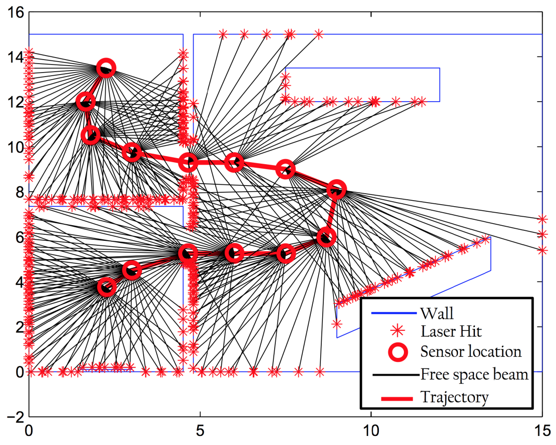
It is essential for a robot expected to interact physically with its surroundings to have a reliable spatial representation of them. Possible interactions include navigation, object recognition and manipulation or simply determining its own position relative to known landmarks. That means spatial modelling is arguably one of the most fundamental tasks in robotics. The amount of research in this field in the past decades makes this conclusion even more evident.

The terrain mapping problem may be more simply expressed as the task of spatially modelling the robot's environment. This can be done in two main ways. The first is through a graph-like representation of important landmarks, connected via arcs that may contain information on how to get from a node to another. This is called a topological map of terrain. This however isn't very readily applied to navigation.

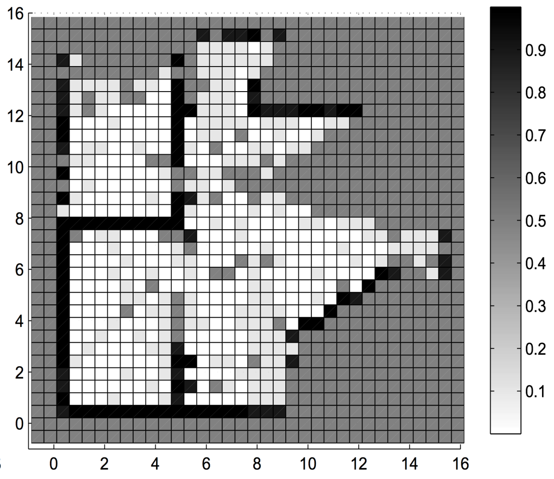
The second approach are metric maps, which aim at capturing geometric information about the environment. To achieve this, the area to be mapped is generally represented as a grid. In natural or otherwise unstructured spaces, which frequently lack regular and easily recognisable structures, digital elevation maps (DEMs) (Bares et al., 1989) are a popular approach. In this method, the mapped terrain is represented by a 2D grid where each cell contains the approximate elevation at the corresponding coordinates.

This work is concerned with structured terrain, such as occurs inside a building or other manmade structure. An archetypical technique is the occupancy grid map (OGM) (Moravek and Elfes, 1985, 1988; 1989). In this method, the grid that overlays the terrain, rather than storing elevation, contains information relative to whether or not that part of space is occupied.

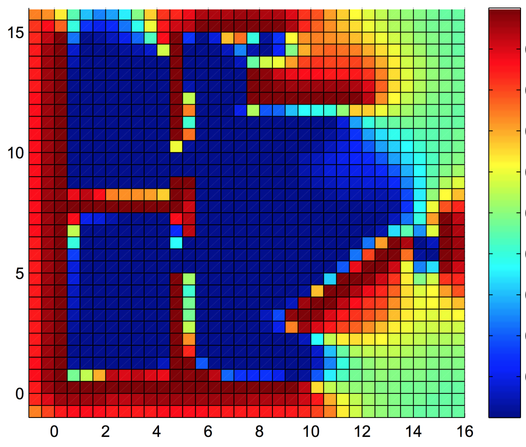
Both DEMs and occupancy grids have the problems associated with discretisation: they may misrepresent a continuous phenomenon and the space and processing power needed to create and analyse the grid scale with the number of cells.



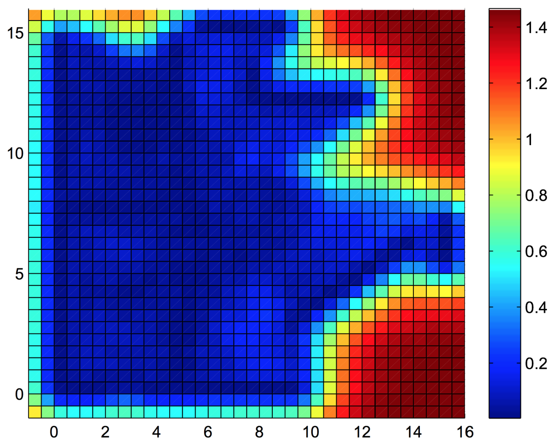
(a) Input data and ground truth.



(b) Occupancy grid map.



(c) Continuous occupancy map.



(d) Associated variance.

Figure 1.1: Comparison: occupancy grid map (b) and continuous occupancy map (c) with associated variance plot (d) generated from the same dataset (a). Continuous maps were sampled at the same resolution as the OGM. Taken from O'Callaghan (2012)

A continuous occupancy map is a non-discretised representation of the environment. It can overcome the difficulties of OGMs that stem from discretisation.

The technique developed in this work falls within this category.

To perform continuous mapping, we use a Bayesian learning technique known as Gaussian process (GP). It can be thought of as a generalisation of a Gaussian distribution. Much as the latter characterises a distribution of values through a mean and a covariance, the GP uses a pair of mean and covariance functions to characterise a distribution over a space of possible functions. It learns these properties from the dataset.

1.3 CHALLENGES AND CONTRIBUTIONS

There are two main challenges to the technique proposed:

INPUT UNCERTAINTY As is the case with any technique that depends on sensor readings, noise is inevitable. If the model fails to take input uncertainty into consideration, the maps generated will suffer in quality and potentially endanger the robot that uses them.

LARGE DATASETS Another consequence of using sensors is the sheer volume of data provided by them. Modern laser rangefinders can generate thousands of readings per minute, which is a challenge to GPs, that scale cubically with the number of inputs.

In addressing the aforementioned challenges, the proposed algorithm offers the following contributions to the field:

NOVEL MULTI-SUPPORT KERNEL The kernel developed in this thesis is able to aggregate spatial data into generic geometric elements, reducing the number of inputs generated. Thanks to this, the algorithm can handle even the large amount of data produced by modern sensors with greater ease than state-of-the-art techniques.

NOVEL MAPPING TECHNIQUE We propose a continuous mapping technique that uses a GP equipped with the kernel described above. This allows the method to deal simultaneously with the size of the datasets and the inaccuracy of sensors.

1.4 THESIS OVERVIEW

This thesis is divided as follows:

CHAPTER 2 presents the theoretical background that lays the foundation for this work. Robotic mapping is introduced and occupancy grid mapping, arguably the most popular metric mapping technique in use, explained in detail. Gaussian processes are introduced as a candidate to deal with the shortcomings in grid-based techniques, and their own drawbacks are presented.

CHAPTER 3 is where the main contribution of this thesis is laid out. A brief review of related works is offered. Multi-support kernels are introduced and their use for continuous occupancy mapping demonstrated.

CHAPTER 4 conveys the result of the experiments made. The effect of each parameter on the output is analysed. The proposed algorithm is also benchmarked against the GPOMIK (O'Callaghan, 2012) algorithm with synthetic and real datasets.

CHAPTER 5 concludes this work. The contributions made are restated and summarised. Possibilities of future research on this topic are outlined.

Chapter 2

BACKGROUND

Most widespread mapping representations rely on occupancy grids. As a result, a number of the shortcomings of these techniques stem from discretisation. Machine learning techniques, in particular Gaussian processes, are well-suited to infer continuous functions from the data, which could avoid these problems. Their execution, however, tends to be slow in applications with large datasets.

In this chapter we discuss the theoretical background that lays the foundation for this work. An overview of robotic mapping is offered, and occupancy grid mapping is addressed in more detail. Gaussian processes are introduced as a technique to deal with the drawbacks in grid mapping, and their own shortcomings are exposed.

One has studied your history.

— Andrew Martin

2.1 ROBOTIC MAPPING

Creating an accurate representation of its environment is an essential step for any autonomous system to efficiently interact with it. As such, mapping plays a role of central importance in robotic navigation and path planning. Maps are generated from sensorial data collected by the robot from its surroundings, which is condensed into a representation that can be more easily manipulated by navigation algorithms. Mapping is a challenging task for a number of reasons. Thrun, Burgard, and Fox (2005, chap. 9) name a few:

◇ The robot is inside the environment it is trying to model, for which reason mapping is frequently coupled with localisation, in what is referred to as the simultaneous localisation and mapping (SLAM) problem. It must be noted that localisation within a given map is a relatively simple task for which there are several algorithms—Monte-Carlo localisation being a popular example.

◇ It is also relatively easy to build maps given the exact location of the robot. Unfortunately, noise intrinsic to robot actuators make the available information unreliable. The error generated by this noise is incremental: along the robot's path, each estimation is affected not only by the noise at the current position, but also by the error accrued on previous readings. This is easy to show: imagine a robot that moves by steps, which it measures as being one meter in length. For illustrational purposes, let us say it actually moves 99 centimetres. After one step it will be within one centimetre of where it assumes to be, though after ten steps this error builds up to ten centimetres. In real applications, where robots can slip or skid unpredictably, modelling this error can become extremely complex.

◇ Odometry noise generates other problems. As the environment grows larger, longer paths are required to fully map it. Using other sensors or forcing the robot to return through a path it used before can compensate for the errors generated. Similar repetitive features (like doors or columns in a hallway) and cycles around obstacles can further complicate the task.

◇ The most significant challenge is perhaps the dimension of the hypothesis space (i. e. , the space of all possible maps). The environments mapped are continuous, because such is the nature of the real world, meaning there are infinite dimensions to this modelling problem. But even in discrete approximations the number of variables easily reaches high orders of magnitude. This problem becomes worse as maps increase in size or resolution, making Bayesian approaches for localisation ineffective for this task.

The mapping task can be summarised as that of identifying and modelling important *features* in the robot's environment. Features are any distinct stationary aspects of the environment that are relevant to the robot. The manner in which features are represented and stored divides the field between *topological* and *metric* mapping.

Topological mapping aims at representing the environment through a list of features of topological or functional importance. Information can be stored about the relations between then, such as distance or navigational information. The resulting maps are coarse, with features stored on nodes and relations on the arcs that connect these nodes. A pure topological approach, however, dis-

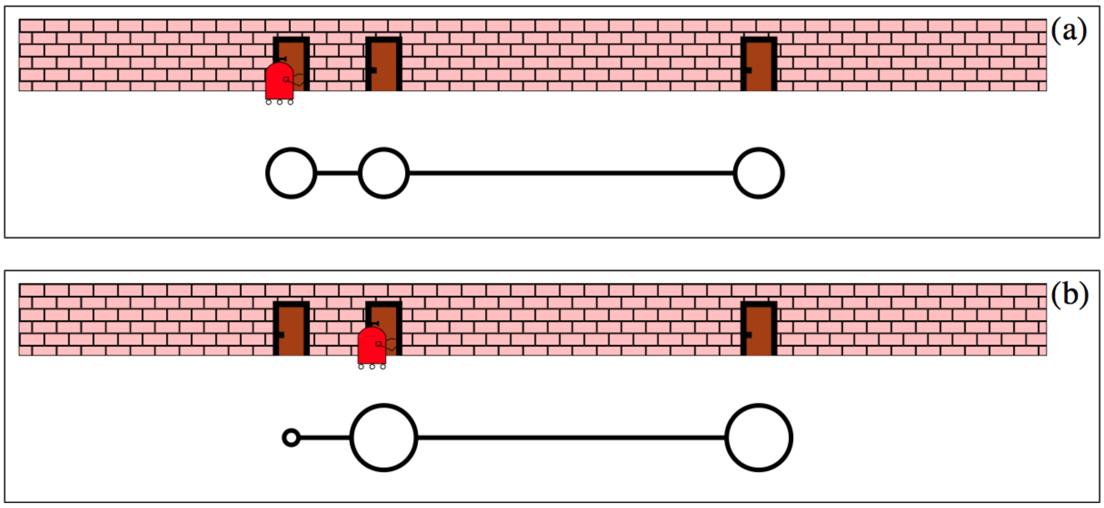


Figure 2.1: A simple example of a topological map used for localisation. Features in this map correspond to doors. The size of each node corresponds to the probability of the robot occupying that node (as calculated by the robot). (a) The initial probability is equal for all doors, as the robot has no means of telling them apart. (b) As the robot moves right and detects a second door, it becomes unlikely that it could be in the leftmost node, so the probabilities are adjusted to reflect this. Figure taken from Thrun, Burgard, and Fox (2005, chap. 8).

cards valuable information about the space in between the features (Thrun, 2003a). A simple example of this kind of map is offered in Figure 2.1.

Metric mapping, on the other hand, is more akin to the related field of cartography: it aims at creating a scaled representation of the environment, with the features being geometric features placed in fixed coordinates. The archetypical metric representation is the occupancy grid map, explained in detail in the next section.

2.1.1 OCCUPANCY GRID MAPPING

One of the most widespread methods used for metric mapping, occupancy grid maps (OGMs) were developed in the 80s by Moravek and Elfes (1985, 1988; 1989). They are versatile, easy to implement and computationally efficient, which accounts for their widespread use, especially in two-dimensional mapping. They deal with the unavoidable effects of noise in sensors and actuators by using a probabilistic framework.

This approach can be intuitively interpreted as such: imagine a grid laid over the environment to be mapped. The map should be a two-dimensional array with the same number of rows and columns as the grid, so that each element in the first corresponds to a cell (at the same relative position) in the latter. Once calculated, the array stores the probabilities of each cell to be *occupied*, that is, inaccessible to the robot. The map is an orthographic projection of the environment, similar to a floor plan. In a more mathematically strict definition, the objective of OGMs is to use the poses of the robot for all time steps elapsed and their corresponding sensory information, denoted by $z_{1:t}$, to calculate the posterior probability

$$p(\mathbf{m} \mid z_{1:t}), \tag{2.1}$$

where $\mathbf{m} = \{m_i\}$ is a map composed of a finite number of cells. However, as mentioned in the Section 2.1, the hypothesis space contains 2^i possible states, so the calculation of Equation 2.1 quickly becomes intractable as the number of cell grows. To address this issue, an approximation must be made: that the posterior over maps can be approximated as the product of its marginals[⋆],

$$p(\mathbf{m} \mid z_{1:t}) = \prod_i p(m_i \mid z_{1:t}), \tag{2.2}$$

and the hypothesis space contains a mere $2i$ states. This approximation is called *grid decomposition*, and it relies on a highly constraining assumption: that the cells in the grid are independent and identically distributed (iid) or, in other words, random. This ignores a property of the system studied: objects in the real world, especially in structured environments, have a regular and

⋆ For convenience, $p(m_i)$ is used in all equations that follow as a shorthand for $p(m_i = 1)$, which denotes a probability of 1 that the cell m_i is occupied.

fairly predictable physical structure. Ignoring this property impoverishes the model, as it discards information that would aid in its prediction.

In spite of this loss, a simple example helps understand the magnitude of the change in complexity allowed by Equation 2.2. If we were to map a square area $5m$ in side, even using a very coarse grid of square cells 0.5m in side, the hypothesis space of Equation 2.1 contains 2^{100} (more than 10^{30}) possible maps. After the approximation, this number is reduced to a mere 200.

From this point, we can apply Bayes' rule to calculate the probability of occupancy for each cell given the observations,

$$p(m_i \mid z_{1:t}) = \frac{p(z_t \mid z_{1:t-1}, m_i) p(m_i \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})}. \tag{2.3}$$

A common assumption in robotic mapping is that the environment being mapped is static, which allows us to state that any observation z_t is independent of any previous observation given knowledge of the map $\mathbf{m}$:

$$p(z_t \mid z_{1:t-1}, \mathbf{m}) = p(z_t \mid \mathbf{m}). \tag{2.4}$$

Due to the grid decomposition, we need to make another strong assumption: that Equation 2.4 is also valid for each individual cell, regardless of the occupancy of the neighbouring cells (Thrun, 2003b). This allows us to adapt this equation, yielding

$$p(z_t \mid z_{1:t-1}, m_i) = p(z_t \mid m_i) = \frac{p(m_i \mid z_t) p(z_t)}{p(m_i)}. \tag{2.5}$$

This assumption is fundamentally incorrect: even in a static world, observations frequently encompass more than one cell, and the results for all of them would have to be known at once for the independence to hold. However the resulting approximation is convenient, because it allows us to express the probability of occupancy of each cell using a single observation. Furthermore, it can be used to simplify Equation 2.3 to

$$p(m_i \mid z_{1:t}) = \frac{p(z_t \mid m_i)p(m_i \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})} = \frac{p(m_i \mid z_t)p(z_t)p(m_i \mid z_{1:t-1})}{p(m_i)p(z_t \mid z_{1:t-1})}, \quad (2.6)$$

which computes the probability that m_i is occupied. An analogous derivation allows to calculate the probability that m_i is free, denoted by $p(\bar{m}_i)$:

$$p(\bar{m}_i \mid z_{1:t}) = \frac{p(z_t \mid \bar{m}_i)p(\bar{m}_i \mid z_{1:t-1})}{p(z_t \mid z_{1:t-1})} = \frac{p(\bar{m}_i \mid z_t)p(z_t)p(\bar{m}_i \mid z_{1:t-1})}{p(\bar{m}_i)p(z_t \mid z_{1:t-1})}. \quad (2.7)$$

Dividing Equation 2.6 by Equation 2.7, we eliminate several terms:

$$\frac{p(m_i \mid z_{1:t})}{p(\bar{m}_i \mid z_{1:t})} = \frac{p(m_i \mid z_t)p(\bar{m}_i)p(m_i \mid z_{1:t-1})}{p(\bar{m}_i \mid z_t)p(m_i)p(\bar{m}_i \mid z_{1:t-1})}. \quad (2.8)$$

The result is an odds ratio, which can be verified by simply noting that by definition $p(m_i) = 1 - p(\bar{m}_i)$, since one is merely the complement of the other. Therefore, Equation 2.8 can be rewritten as:

$$\frac{p(m_i \mid z_{1:t})}{1 - p(m_i \mid z_{1:t})} = \frac{p(m_i \mid z_t)}{1 - p(m_i \mid z_t)} \frac{1 - p(m_i)}{p(m_i)} \frac{p(m_i \mid z_{1:t-1})}{1 - p(m_i \mid z_{1:t-1})}. \quad (2.9)$$

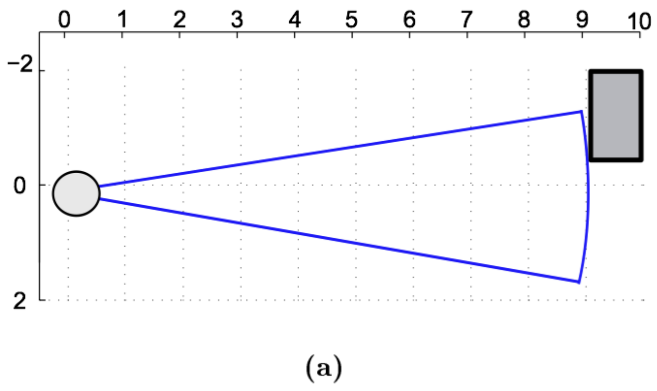
Taking the log of Equation 2.9 yields an interesting result:

$$l_i^t = \log \frac{p(m_i \mid z_{1:t})}{1 - p(m_i \mid z_{1:t})}. \quad (2.10)$$

This is the *log odds ratio* of $p(m_i \mid z_{1:t})$. This quantity is defined in the interval $(-\infty, \infty)$ rather than $[0, 1]$, which is desirable from an implementation viewpoint, since it reduces the numerical instabilities of the algorithm. Furthermore, substituting the right side of Equation 2.9 in Equation 2.10 yields an equation that can be used to update the map generated by a previous collection of observations $z_{1:t-1}$ using a new observation z_t:

$$l_i^t = \log \frac{p(m_i \mid z_t)}{1 - p(m_i \mid z_t)} + \log \frac{1 - p(m_i)}{p(m_i)} + \log \frac{p(m_i \mid z_{1:t-1})}{1 - p(m_i \mid z_{1:t-1})}. \tag{2.11}$$

This result allows us to calculate the desired posterior using two probabilities defined *a priori*. One is $p(m_i)$, our initial guess at the probability of occupancy before any observations are made. Thrun, Burgard, and Fox (2005) suggest this value initially be set to a value in the range $[0.2, 0.5]$. The other is $p(m_i \mid z_t)$, the *inverse sensor model*, which maps sensor data back to its cause. Contrast this with a *forward* sensor model, which models the probability of acquiring the observations given a certain cause, represented by $p(z_t \mid m_i)$.



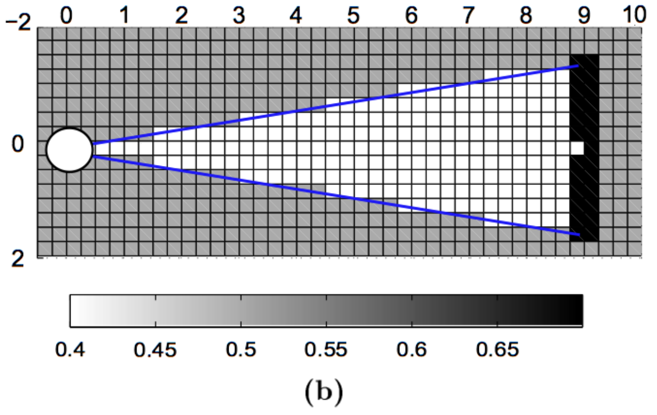


Figure 2.2: An example of a simple inverse sensor model. (a) A robot with a wide beam sensor (such as a sonar) detects an obstacle in its path. (b) The inverse sensor model generated from the observation. Figure taken from O'Callaghan (2012, chap. 2).

Equation 2.10 can be used to create an update algorithm (Algorithm 2.1, as given by Thrun 2003b) that takes a map and a collection of observations and returns an updated version of the map. Its simplicity is one of the reasons behind the popularity of OGMs,

Algorithm 2.1: Occupancy grid map update algorithm, after Thrun 2003b.

```
input : p(m0), z
    # map hypothesis for time 0, observations from time 1..t

for all grid cells i:
    # Convert map hypothesis to log odds
    l[i] := log(p(m0[i])) - log(1 - p(m0[i]))

for all time steps t:
    for all grid cells i within the perceptual range of z[t]:
        # Grid calculation using log-odds
        l[i] += log(p(m0[i] | z[t]))
        l[i] -= log(1 - p(m0[i] | z[t]))
        l[i] -= log(p(m0[i]))
        l[i] += log(1 - p(m0[i]))

for all grid cells i:
    # Convert log odds to updated map hypothesis
    p(m[i]) := 1 - (1 / exp(l[i]))

output : p(m)
    # updated map hypothesis for time t
```

Figure 2.3 illustrates the OGM algorithm applied to a synthetic dataset. Figure 2.3a shows the ground truth (i. e. the boundaries the robot needs to learn), a series of poses and sensor readings taken from each and the trajectory between poses. In this example, a sensor with a narrow sensor beam (such as a laser rangefinder) was simulated. The occupancy map generated is presented in Figure 2.3b.

2.1.2 SHORTCOMINGS OF OCCUPANCY GRID MAPPING

In the last section we described the OGM algorithm, and exposed the calculations behind it. We also discussed the grid decomposition that must be made

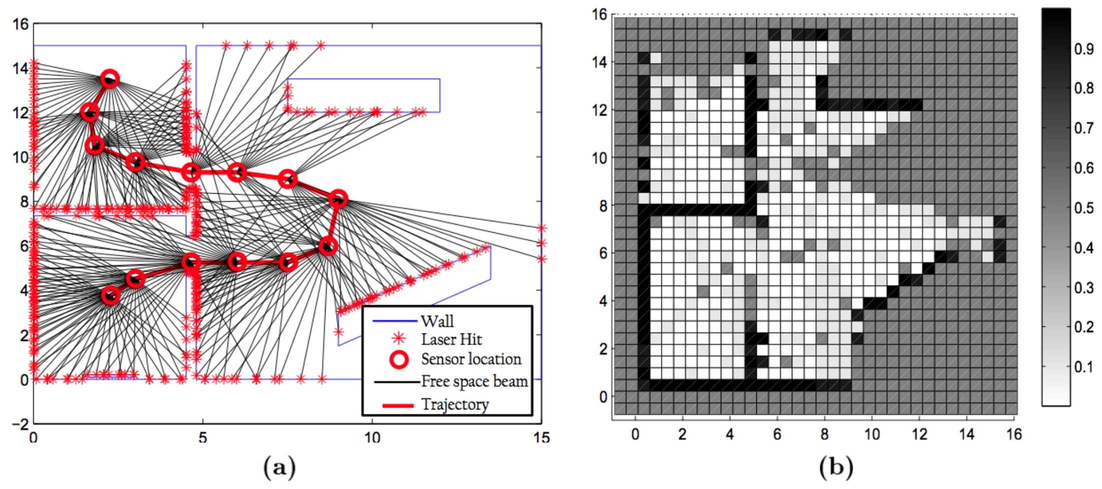


Figure 2.3: (a) Data from a robot with a narrow beam sensor (such as a LIDAR) moving around an unknown environment. (b) The map created from this data. Figure taken from O'Callaghan (2012, chap. 2).

to lead to Equation 2.2 and Equation 2.5; both lie at the core of the algorithm's simplicity.

Paradoxically, the same assumption that arguably gives the algorithm its widespread popularity is one of its greatest shortcomings. Assuming that the environment is composed of randomly distributed regions of occupancy ignores a fundamental property of the real world, where objects frequently have a regular physical structure. This oversimplification leads to poor predictions in regions where taking into account contextual information could help determine occupancy, such as between sensor readings or in occluded regions. Given that occupancy grids are frequently used in structured environments, it is fair to say that this approximation ignores a large amount of useful information.

Storing the map as a grid brings forth other drawbacks. The cells in an OGM have to be of pre-defined size, which means that the map has a fixed resolution. This is not a problem when a single robot is using the map, but a map created by a large robot may not be adequate for a smaller one. The only ways to change the resolution of an OGM are to store it as a quadtree (Li and Ruichek, 2013) or to rebuild it from scratch each time.

Grid methods is also subject to discretisation errors. In Figure 2.3 this becomes evident. At the top door, the robot is uncertain about the occupancy of two cells it actually crosses, where it should be obvious that any cells the robot can move through are free. At the bottom door the opposite happens: despite numerous sensor beams hitting a wall, its occupancy is also uncertain for the robot.

Even though problems discussed up to this point have been two-dimensional, some applications actually require three-dimensional maps. Once again, discretisation pays a toll: due to the curse of dimensionality, occupancy grids in three dimensions require massive amounts of memory to represent the environment, and this requirement requirement rises as the resolution increases.

All of these problems in effect stem from discretising a continuous problem. But this is not strictly necessary. In the next section, we will discuss Gaussian processes, a machine learning technique for regression, particularly useful for spatial modelling.

2.2 GAUSSIAN PROCESSES

In his book *Information Theory, Inference, and Learning Algorithms*, David MacKay describes a Gaussian process (GP) as the generalisation of a Gaussian distribution over a finite vector space to a function space of infinite dimension (MacKay, 2003, chapter 45). To enable this generalisation, instead of deriving its properties from a mean and covariance matrix, a GP is characterised by a pair of analogous and homonymous functions.

Figure 2.4 offers a visual intuition of GPs through a simple regression problem.

It represents sample functions drawn from the prior and posterior distribution of a GP which favours smooth functions, a property that is specified by the covariance function. For the prior, the mean function is taken to be the zero function. After data points are introduced, the mean prediction is adjusted accordingly. Notice that the standard deviation, indicated in Figure 2.4 as a shaded grey area, is proportional to the distance to points for which the target value is known.

It is important to notice that more data points can always be added as they

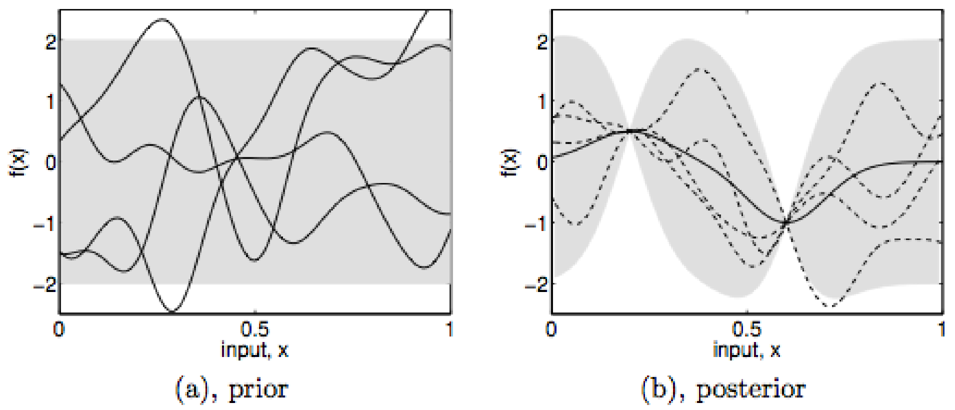


Figure 2.4: An intuitive visual representation of GPs. (a) Four samples drawn from the prior distribution. (b) After two data points have been observed, the model is changed accordingly and four new samples are drawn. In both panels, the solid line represents the mean function and dashed lines represent sample functions in the distribution, randomly chosen. The shaded regions denote twice the standard deviation at each input value x. Taken from Rasmussen and Williams (2006).

become available without worry about whether the model can still fit the data, since GPs are non-parametric. Thanks to this property, even after adding a large number of observations the model can still produce a collection of candidate functions that reflect the new information provided. In comparison, parametric models require new parameters to be added in order to capture detail from additional data points.

We move on to formalise the concepts above. Rasmussen and Williams define a GP as such:

DEFINITION: *A Gaussian process is a collection of random variables, any finite* *number of which have a joint Gaussian distribution.* □

Let $\mathcal{D} = \{(x_i, y_i) \mid i = [1, n] : n \in \mathbb{N}\}$ be a dataset containing n data points $x \in \mathbb{R}^D$ (the *inputs*) and their corresponding observed values $y \in \mathbb{R}$ (the *outputs*), defined by an unknown underlying function $f(x)$ that we desire to model. It is important to note that y is a measurement; it differs from $f(x)$ by the noise inherent to any measurements. We can then define the mean function $m(x)$ and covariance function $k(x, x')$ to be:

$$\begin{aligned} m(x) &= \mathbb{E}(f(x)), \\ k(x, x') &= \operatorname{cov}(f(x), f(x)') = \mathbb{E}((f(x) - m(x))(f(x') - m(x'))), \end{aligned} \tag{2.12}$$

and the GP determined by them can be indicated by

$$f(x) \sim \mathcal{N}(m(x), k(x, x')). \tag{2.13}$$

In regression problems[⋆], for simplicity, $m(x)$ is often taken to be zero. It must be noted that this does not mean that the mean value of each function in the distribution is zero—rather, it means that for a fixed x_i, as we draw different functions from the sample, the mean value of these functions calculated at that point approaches zero.

The covariance function is what determines the behaviour of the functions in the distribution. It is frequently chosen to favour smooth continuous functions, but this of course is not mandatory. In Section 2.2.2, we will discuss different covariance functions and the kinds of behaviour they can specify.

⋆ Regression problems are those in which $f(x)$ assumes real values. Contrast those to *classification* problems, where x is assigned to one of several classes. The archetypical example is written digit recognition, where the objective is to classify an image of a handwritten digit into one of ten classes, 0-9. Gaussian processes are equally suited to deal with both tasks, and have been extensively used for both. This work, however, will focus on regression techniques.

An advantage of using GPs is that they allow us to calculate the posterior,

$$p(f(\mathbf{x}) \mid y, \mathbf{x}) = \frac{p(y \mid f(\mathbf{x}), \mathbf{x}) p(f(\mathbf{x}))}{p(y \mid \mathbf{x})}, \tag{2.14}$$

without having to parameterise f, by placing the prior directly on the infinite space of functions. Thanks to this property, no assumptions have to be made over the output function properties, setting GPs apart from approaches that require parameterisation (MacKay, 2003). Beyond that, unlike other Bayesian methods, Gaussian processes normally return functions which are analytically solvable.

2.2.1 GAUSSIAN PROCESS REGRESSION

The objective of Gaussian process regression is to use the framework outlined in the previous section to generate predictive models out of the dataset provided. That is, a GP model must be able to take a set of test points and output a set of predicted values for each. In other words: the interest is not to model the distribution of the inputs, but rather the conditional distribution of the targets given the inputs.

Given the dataset $\mathcal{D}$ defined in the previous section, we can aggregate the n D-dimensional input points into a single $D \times n$ matrix X, and do the same for the observed scalar outputs to generate a column vector $\mathbf{y}$. For now we assume a noiseless model, such that $\mathbf{y} = \mathbf{f}$.

We want our model to be able to predict the outputs $\mathbf{f}_*$ from a set of m inputs $\mathbf{x}_*$, our *test points*, which are similarly aggregated into the test matrix X_* and corresponding output point $\mathbf{f}_*$. The relation between $\mathbf{f}_*$ and $\mathbf{f}$ that allows us to infer the first from the latter is that we assume both were drawn from the same GP. Therefore they are jointly distributed according to the same prior:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right), \tag{2.15}$$

where $K(A,B)$ denotes the covariance matrix generated by calculating the element-wise covariance between matrices A and B. At this point we introduce, for convenience, a shorthand notation: $K = K(X,X)$, $K_* = K(X,X_*) = K(X_*,X)^\top$ and $K_{**} = K(X_*,X_*)$. We can then condition the joint Gaussian prior distribution on the observations, yielding

$$\begin{aligned} &\mathbf{f}_* \mid X_*, X, \mathbf{f} \sim \mathcal{N}(\mathbf{m}_*, \Sigma_*), \text{ where:} \\ &\qquad \mathbf{m}_* = K_*^\top K^{-1}\mathbf{f}, \\ &\qquad \Sigma_* = K_{**} - K_*^\top K^{-1} K_*. \end{aligned} \tag{2.16}$$

At this stage, we can evaluate the mean and covariance matrix from the above equation. Note that while $\Sigma_* = \mathrm{cov}(\mathbf{f}_*)$ is a fully populated covariance matrix for $m > 1$, it is common practice to evaluate only its diagonal to obtain a variance vector.

Until this point we have assumed noiseless measurements for simplicity. In real applications, however, this is an unlikely scenario. The observations more often carry along noise, which we assume to be Gaussian:

$$\begin{aligned} &y = f(\mathbf{x}) + \varepsilon \text{ and} \\ &\varepsilon \sim \mathcal{N}(0, \sigma_n^2), \text{ therefore} \\ &\Sigma = K + \sigma_n^2\, I, \end{aligned} \tag{2.17}$$

where $\Sigma = \mathrm{cov}(\mathbf{y})$. Equation 2.15 and Equation 2.16 are now recast using $\mathbf{y}$ instead of $\mathbf{f}$, yielding slightly different formulations:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} K + \sigma_n^2\, I & K_* \\ K_*^\top & K_{**} \end{bmatrix}\right), \tag{2.18}$$

$$\begin{aligned} \mathbf{f}_* \mid X_*, X, \mathbf{y} &\sim \mathcal{N}(\mathbf{m}_*, \Sigma_*), \text{where:} \\ \mathbf{m}_* &= K_*^\top (K + \sigma_n^2\, I)^{-1} \mathbf{y}, \\ \Sigma_* &= K_{**} - K_*^\top (K + \sigma_n^2\, I)^{-1} K_*. \end{aligned} \tag{2.19}$$

This Bayesian formulation for inference is also well suited to machine learning, because the Gaussian prior in Equation 2.15 and Equation 2.18 allow the training of the algorithm, as will be discussed in Section 2.2.3.

We now shall look at the case where we have a single test point x_*. We introduce the notation $\mathbf{k}_* = (\mathbf{x}, \mathbf{x}_*)$ and $k_{**} = k(x_*, x_*)$. Then Equation 2.19 becomes

$$\begin{aligned} f_* \mid X_*, X, \mathbf{y} &\sim \mathcal{N}(m_*, \sigma_*), \text{ where:} \\ m_* &= \mathbf{k}_*{}^\top (K + \sigma_n^2\, I)^{-1} \mathbf{y}, \\ \sigma_* &= k_{**} - \mathbf{k}_*{}^\top (K + \sigma_n^2\, I)^{-1} \mathbf{k}_*. \end{aligned} \tag{2.20}$$

The mean function in the equation above is a product between a line and a column vector of equal dimensions. This makes sense: since we have a single test point, the mean function evaluates to a scalar, and can thus be rewritten as:

$$\begin{aligned} m_* &= \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i), \text{ where:} \\ \boldsymbol{\alpha} &= (K + \sigma_n^2\, I)^{-1} \mathbf{y}. \end{aligned} \tag{2.21}$$

An implementation of GP regression, based on the one from Rasmussen and Williams (2006), is shown in Algorithm 2.2. It includes the calculation of the log marginal likelihood, which will be addressed in Section 2.2.3. Notice the use of a Cholesky decomposition to deal with the need for a matrix inversion, as this is a faster and more stable method.

Algorithm 2.2: Gaussian process regression algorithm, after Rasmussen and Williams 2006.

```
input : X, y, k(), par0, x
    # inputs, targets, covariance function, hyperparameters
    # and test point
[par, sig] := par0
K := k(X,X, par)
L := cholesky(K + sig ∗ I)
    # Cholesky decomposition, L := cholesky(A) so that L∗L^⊤ = A
alpha := L^⊤ \ (L \ y)
kXx := k(X, x, par)
m := kXx^⊤ ∗ alpha
v := L\kXx
var := k(x, x, par) - v^⊤ ∗ v
lml := 0.5 ∗ (y^⊤ ∗ alpha - n ∗ log(2 ∗ pi)) - sum(diag(log(L)))
output : m, var, lml
    # mean, variance, log marginal likelihood
```

2.2.2 COVARIANCE FUNCTIONS

As mentioned in Section 2.2, it is the covariance function that encodes our assumptions about the behaviour of the functions in the distribution generated by a Gaussian process. The choice of function determines basic properties (e. g. smoothness or stationarity), while Bayesian learning is employed to fine-tune others (e. g. periodicity or amplitude). We will discuss hyperparameters along with learning in Section 2.2.3.

At this stage, we can introduce the concept of *kernels*. Kernels are functions that, given two inputs $\mathbf{x}_i$ and $\mathbf{x}_j$, return a measure of the *similarity* between them (Schölkopf and Smola, 2002). Choosing this measure is a very fundamental question in machine learning (ML). If we have a kernel k and a set of inputs $\mathcal{X} = \{x_i \mid i = [1, n] : n \in \mathbb{N}\}$, we can compute the *Gram matrix* of $\mathcal{X}$, denoted by K. Its entries are given by $K_{ij} = k(x_i, xj)$.

Covariance functions are a particular case of kernels, and their Gram matrices are called *covariance matrices*. By definition, they must be symmetric, i. e., it is strictly necessary that $k(x, x') = k(x', x)$. By virtue of the Cholesky decomposition performed in Algorithm 2.2, they must generate positive semi-definite (PSD) Gram matrices; and therefore be PSD themselves. To obey this property, they must satisfy the inequality:

$$\int k(x, x')f(x)f(x)'d\mu(x)d\mu(x') \geqslant 0, \tag{2.22}$$

where μ denotes a measure. Bearing in mind this restriction, from this point onwards, the terms "kernel" and "covariance function" will be used interchangeably

It is important to notice that in a GP, the covariance between two *outputs* is written as a function of the *inputs* (see Equation 2.12). This is an interesting observation: we are linking the behaviour of the outputs to the similarity between the inputs, which is useful for us as we are trying to predict the output for unknown inputs using that of known ones. Measuring similarity between outputs is relatively easy. We can illustrate this with a simple example with binary classification: two labels either are the same or they are not. Kernels allow to make this measurement by looking at the inputs instead.

Scattered across the ML literature is a plethora of different kernels. Notwithstanding this fact, new kernels can also be created from existing ones by linear combination, point-wise product, dot product, tensor product, direct sum, iteration or convolution (Rasmussen and Williams, 2006). This richness allows us to model an immense range of functions.

The existing covariance functions can be divided in classes, some of which are of special interest for this work, either for their popularity or for their appropriateness for the task at hand. In the remainder of this section, we will look at two of them.

A. *Stationary covariance functions*

A kernel or covariance function is said to be *stationary* when it is a function of the difference between inputs, $\mathbf{x} - \mathbf{x}'$, as this makes them invariant to translations in the input space. A particular case are *isotropic* kernels, which are functions of the distance $|\mathbf{x} - \mathbf{x}'|$. Examples of widely used stationary kernels are the *squared exponential* (which sometimes appears in the literature by the name of *radial basis function* or RBF kernel) and those in the *Matérn class* of covariance functions.

The squared exponential function is defined by the following equation:

$$k_{SE}(\mathbf{x}_p, \mathbf{x}_q) = \sigma_f^2 \exp\left(-\frac{|\mathbf{x} - \mathbf{x}'|^2}{2\mathbf{l}^2}\right), \tag{2.23}$$

where σ_f^2 is the *signal variance* and $\mathbf{l}$ the *length-scale*, two hyperparameters. It is easy to see the exponential's value is one when $p = q$ and approaches zero as the distance between the inputs grows. The role of the hyperparameters then becomes explicit: σ_f^2 changes the maximum value the function can have, while $\mathbf{l}$ determines how fast this value decays with distance. If the length-scale is one-dimensional, the decay happens at the same rate in all dimensions. A D-dimensional length-scale allows for different decay rates along each dimension.

This covariance function is *infinitely divisible*: k_{SE}^t is a valid kernel for any positive value of the exponent, and the only effect of exponentiation is to rescale

l. It is also infinitely differentiable, which is why it favours smooth functions.

Stein (1999, chapter 2) argues that smoothness is not an adequate assumption to make in some practical applications, as the behaviour is not frequently observed in physical processes. He proposes a few alternatives. One of them is what he calls the Matérn class of covariances, the general case of which is defined by the equation:

$$k_{Mat}(x_p, x_q) = \frac{1}{\Gamma(\nu)2^{\nu-1}} \left(\sqrt{2\nu} \, \frac{|x - x'|}{l} \right)^{\nu} \mathcal{K}_{\nu} \left(\sqrt{2\nu} \, \frac{|x - x'|}{l} \right), \tag{2.24}$$

where $\mathcal{K}_\nu(z)$ is a modified Bessel function that tends to zero as $|z| \to \infty$. It assumes real and positive values for $\nu > -1$ and $z > 0$ (Abramowitz and Stegun, 1965, chapter 9). The parameter ν determines the *differentiability* of the function, and as ν raises, the functions determined by this kernel become smoother. When it takes values of the form $\nu = i + {}^1\!/_2 : i \in \mathbb{N}$, Equation 2.24 assumes much more convenient forms. The most widely used are $\nu = {}^3\!/_2$ and $\nu = {}^5\!/_2$, sometimes called Matérn 3 and Matérn 5:

$$\begin{aligned} k_{Mat^3/_2}(x_p, x_q) &= \left(1 + \frac{\sqrt{3}|x - x'|}{l}\right) \exp\left(-\frac{\sqrt{3}|x - x'|}{l}\right), \\ k_{Mat^5/_2}(x_p, x_q) &= \left(1 + \frac{\sqrt{5}|x - x'|}{l} + \frac{5|x - x'|^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right). \end{aligned} \tag{2.25}$$

Although values of $\nu \geqslant {}^7\!/_2$ are possible, the functions generated become harder to distinguish from each other and, as ν approaches ∞, from the squared exponential kernel. This behaviour is to be expected, as infinite differentiability leads to smoothness. Therefore, the use of larger values of ν than the ones exemplified in Equation 2.25 is discouraged unless it reflects explicit prior knowledge of the existence of higher order derivatives.

B. *Non-stationary Covariance Functions*

For several processes, stationarity is not a desirable property, as the characteristics of the underlying function change upon traversing the input space. Non-stationary covariance functions are those in which the absolute position of an input is relevant, such as those dependant on the dot product $\mathbf{x}^\top\mathbf{x}'$. An example is the general version of the *polynomial* kernel[⋆]:

$$k_d(\mathbf{x}_p, \mathbf{x}_q) = \left(\alpha\, \mathbf{x}_p^\top \mathbf{x}_q + \sigma_0^2\right)^d . \tag{2.26}$$

The hyperparameters are the slope α and the bias σ_0^2. The polynomial kernel is *homogeneous* when the bias is zero and *inhomogeneous* otherwise. The polynomial degree is d, and for the special case where $d = 1$, the above equation describes the *linear* kernel. Using it in kernel techniques often leads to their "non-kernel" counterpart, e. g. kernel principal component analysis (PCA) using a linear kernel is the same as standard PCA (Schölkopf et al., 1999). Dot product kernels are not the only type of non-stationary covariance function though.

Neal (1996, chapter 2) proposes deriving a kernel from a multilayer perceptron, composed of a single hidden layer containing N_H hidden units, that takes an input $\mathbf{x}$. The output of this perceptron is combined with a bias b, yielding

$$f(\mathbf{x}) = b + \sum_{j=1}^{N_H} v_j h(\mathbf{x}; \mathbf{u}_j), \tag{2.27}$$

where v indicates the weights between the hidden units and the outputs, $\mathbf{u}$ are the weights between the input and the hidden units, and h is a bounded trans-

⋆ Not to be confused with the *piecewise polynomial* kernel, which is stationary.

fer function. As $N_H \to \infty$, this network becomes a universal approximator, as shown by Hornik (1993). If we assume iid weights $\mathbf{u}$ and

$$\begin{aligned} b &\sim \mathcal{N}(0, \sigma_b^2), \\ v &\sim \mathcal{N}(0, \sigma_v^2), \end{aligned} \tag{2.28}$$

we obtain:

$$\begin{aligned} \mathbb{E}_{\boldsymbol{w}}(f(\mathbf{x})) &= 0, \\ \mathbb{E}_{\boldsymbol{w}}(f(\mathbf{x})f(\mathbf{x}')) &= \sigma_b^2 + \sum_j \sigma_v^2\, \mathbb{E}_{\mathbf{u}}(h(\mathbf{x};\mathbf{u}_j)h(\mathbf{x}';\mathbf{u}_j)) \\ &= \sigma_b^2 + N_H \sigma_v^2\, \mathbb{E}_{\mathbf{u}}(h(\mathbf{x};\mathbf{u})h(\mathbf{x}';\mathbf{u})), \end{aligned} \tag{2.29}$$

where $\boldsymbol{w}$ denotes the collection of all weights. By setting $\sigma_v^2 = {}^{\omega}/_{N_H}$ we then arrive at $\sigma_b^2 + \omega\ \mathbb{E}_{\mathbf{u}}(h(\mathbf{x};\mathbf{u})h(\mathbf{x}';\mathbf{u}))$ for the covariance of $f(\mathbf{x})$. Because the weights $\mathbf{u}$ are iid and the transfer function is bounded, the central limit theorem applies, so in the limit as $N_H \to \infty$, the stochastic process converges to a Gaussian process.

Finally, choosing a valid transfer function and evaluating $\mathbb{E}_{\mathbf{u}}(h(\mathbf{x};\mathbf{u})h(\mathbf{x}';\mathbf{u}))$ allows us to obtain what is known as the *neural network* (NN) covariance function. For example, taking h to be the error function $\mathrm{erf}(z) = {}^{2}/_{\sqrt{\pi}} \int_0^z \exp(-t^2)\,dt$ and choosing $\mathbf{u} \sim \mathcal{N}(0, \Sigma)$, we arrive at

$$k_{NN}(\mathbf{x}_p, \mathbf{x}_q) = \frac{2}{\pi} \arcsin\left(\frac{2\dot{\mathbf{x}}_p^\top \Sigma\, \dot{\mathbf{x}}_q}{\sqrt{(1 + 2\dot{\mathbf{x}}_p^\top \Sigma\, \dot{\mathbf{x}}_p)(1 + 2\dot{\mathbf{x}}_q^\top \Sigma\, \dot{\mathbf{x}}_q)}} \right), \tag{2.30}$$

where $\dot{\mathbf{x}} = (1, x_1, \dots, x_D)^\top$ is an augmented input vector. These results were obtained by Williams (1998). Samples from a GP that uses this as its covariance function can be viewed as superpositions of sigmoid functions $\mathrm{erf}(u_0 + u_1 x_1 + \dots + u_d x_d)$. If we set $\Sigma = \mathrm{diag}(\sigma_0^2, \sigma_1, \dots, \sigma_D)$, where σ_i controls u_i, we observe that σ_0 determines the offset of the sample functions from the origin, while the remaining σ_i determine their slope (that is, how fast they vary).

The covariance functions listed in this section are the ones used in this work or useful for explanation purposes. For a more comprehensive list of classifications, examples and analyses of individual kernels, the reader is encouraged to refer to Genton (2002), Hofmann, Schölkopf, and Smola (2008) or Schölkopf and Smola (2002). The latter also provides a thorough explanation of kernels and kernel methods.

2.2.3 GAUSSIAN PROCESS LEARNING

In previous sections, it was mentioned several times that the covariance function dictates the behaviour of the functions in a GP sample. In Section 2.2.2 we discussed a few different covariance functions, and the types of behaviour they specify. We also introduced the concept of *hyperparameters*. While the choice of kernel determines what general aspect the output functions will have, the hyperparameters are responsible for fine tuning the output functions. A poor choice of hyperparameters can nullify the effect of a covariance function perfectly tailored for a task.

It should then become obvious that some form of metric to estimate the performance of the model is necessary. One such metric is the *marginal likelihood*, that measures the probability of a model given the data. One of its key properties is that it incorporates a tradeoff between complexity and fit. The marginal likelihood is calculated as the integral of the likelihood times the prior,

$$p(\mathbf{f}_* \mid \mathbf{x}_*, \mathcal{D}) = \int p(\mathbf{f}_* \mid \mathbf{x}_*, \boldsymbol{\theta}, \mathcal{D}) p(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}, \tag{2.31}$$

in which $\mathbf{f}_*$ is the prediction for the test point $\mathbf{x}_*$, $\mathcal{D}$ is the dataset containing all the input points (X) and corresponding measured outputs ($\mathbf{y}$) and $\boldsymbol{\theta}$ are the hyperparameters of the model. In this section we follow the notation used by MacKay (2003, section 45.5) because of the way it makes explicit the participation of the hyperparameters. MacKay remarks that Equation 2.34 is often intractable. He highlights two approaches that can be used:

a. We can determine the most probable values of the hyperparameters, denoted by $\boldsymbol{\theta}_+$, and use them to approximate the value of the integral,

$$p(\mathbf{f}_* \mid \mathbf{x}_*, \mathcal{D}) \sim p(\mathbf{f}_* \mid \mathbf{x}_*, \mathcal{D}, \boldsymbol{\theta}_+). \tag{2.32}$$

b. We can use numerical methods to perform the integration.

In both approaches, knowledge of the gradient of the posterior probability of $\boldsymbol{\theta}$ is useful from an implementation standpoint. The posterior probability itself is given by

$$p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathbf{y} \mid X, \boldsymbol{\theta}) p(\boldsymbol{\theta}). \tag{2.33}$$

In Section 2.2.1, we mentioned that one of the outputs of Algorithm 2.2 is the log marginal likelihood, given by the first term of Equation 2.33 in log form:

$$\log(p(\mathbf{y} \mid X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top (C)^{-1}\mathbf{y} - \frac{1}{2}\log|C| - \frac{n}{2}\log 2\pi. \tag{2.34}$$

For convenience, we introduce the notation $C = K + \sigma_n^2\, I$. We can now differentiate Equation 2.34 with respect to each hyperparameter θ_i:

$$\frac{\partial}{\partial\theta_i}\log(p(\mathbf{y} \mid X, \theta_i)) = \frac{1}{2}\mathbf{y}^\top C^{-1}\frac{\partial C}{\partial\theta_i}C^{-1}\mathbf{y} - \frac{1}{2}\,\mathrm{tr}(C^{-1}\frac{\partial C}{\partial\theta_i}). \tag{2.35}$$

The gradients in Equation 2.35 can therefore be used to optimise the hyperparameters of the model, through methods such as gradient descent.

2.2.4 Shortcomings of Gaussian processes

Gaussian processes are an extremely versatile machine learning method. They are not, however, without shortcomings. The most obvious one is the need to invert the covariance matrix, the number of elements of which is proportional to the square of input points. A matrix inversion is a costly operation, with at a computational complexity $\mathcal{O}(n^3)$, even using a Cholesky decomposition. This is a problem for data-rich applications. Robotic mapping, in particular, often depends on data coming from sensors such as laser rangefinders. Such equipment generates readings at extremely high rates.

2.3 Summary

In this chapter, we have taken a very brief look at the two fields of research that are vital for the development of this work: robotic mapping and Gaussian processes. Each of them is the subject of intensive research efforts by the scientific community, and thus replete with extensive literature. Rather than to provide an in-depth review, the objective here was to expose a few open problems that will be tackled in this thesis.

Robotic mapping has received a lot of attention in the past few decades, due to developments in robotics as a whole. Several of the applications of robots nowadays rely on autonomous navigation, raising the need for accurate maps. We have explored occupancy grid maps, arguably the most widespread algorithm employed for this task, and discussed some of their shortcomings.

The quest for increasingly autonomous systems has also stimulated developments in the field of machine learning. We have reviewed Gaussian process modelling, which was presented as a candidate to address the problems raised in OGM. The main drawback of using the method for this problem was found to be that their computational complexity depends on the size of the input dataset.

In the next chapter, the specific literature addressing these problems will be reviewed, and the technique proposed to face them will be exposed.

Chapter 3

MAPPING WITH MULTI-SUPPORT KERNELS

Gaussian processes can be used to aggregate spatial relations and create continuous occupancy maps. Although they use covariance functions calculated between points in the data, they can be adapted to work with geometrical elements of higher complexity, such as lines, areas and volumes. This transformation is called *change of support*. It not only incorporates contextual information that would be otherwise unavailable, but can also help deal with the computational complexity of the algorithm for data-rich applications.

In this chapter, we briefly review the literature for change of support in GPs and for occupancy mapping using GPs. A framework for the use of complex support kernels for mapping is introduced and demonstrated.

Now I have a headache...
— Marvin

3.1 INTRODUCTION

In Chapter 2, we have reviewed the occupancy grid map (OGM) algorithm, and established that its shortcomings stem mainly from the discretisation of a continuous domain. Gaussian processes (GPs) have been proposed as a way to address this, given their ability to infer contextual information from input data. This has been done before, and a previous mapping algorithm based on GPs will be reviewed in this chapter.

The greatest drawback of GPs is their complexity, which comes from the need for a matrix inversion. Although this cannot be prevented, there are other techniques to make the algorithm run faster. Algorithm 2.2 incorporates one such technique, a Cholesky decomposition of the matrix. This is not enough, though: robotic rangefinders take thousands of readings for each second of activity, and since inverting the covariance matrix scales at $O(N^3)$ with the number of inputs N, the operation becomes unmanageable fast.

Discarding information would be one way to deal with this. The data collected from sensors could be filtered, and information about regions already mapped with high certainty could be discarded with minimal losses. But even then, as the environment being mapped grows, the covariance matrix inevitably grows with it.

An alternative we will explore in this chapter is changing the support of the kernel functions. Traditional kernels use as support D-dimensional points. If they could be grouped in more complex geometric structures (like lines or areas), this could reduce drastically the size of the covariance matrix. We will next review previous work done using integral kernels for robotic mapping.

3.2 RELATED WORK

3.2.1 CHANGE OF SUPPORT IN GAUSSIAN PROCESSES

Gaussian processes, as introduced in Section 2.2, are very versatile tools able to infer underlying functions from the data. Normally, they deal with functions calculated over points. However, this is not the best approach for every situation. In some applications, data can come from higher-dimensional elements of the domain, such as lines, areas and volumes. In particular, this is true in robotic mapping, where it is more interesting to determine if a particular area or volume is occupied, rather than a single point.

Inference made using higher-dimensional geometrical elements is what Gelfand, Zhu, and Carlin (2001) call the *change of support* problem. They exemplify this concept with an application that requires combining geographic data about ozone levels—which comes from fixed monitoring sites that can be considered as points in a map—to the proportion of hospital visits due to asthma—which is grouped by zip code, and therefore must be considered as areas.

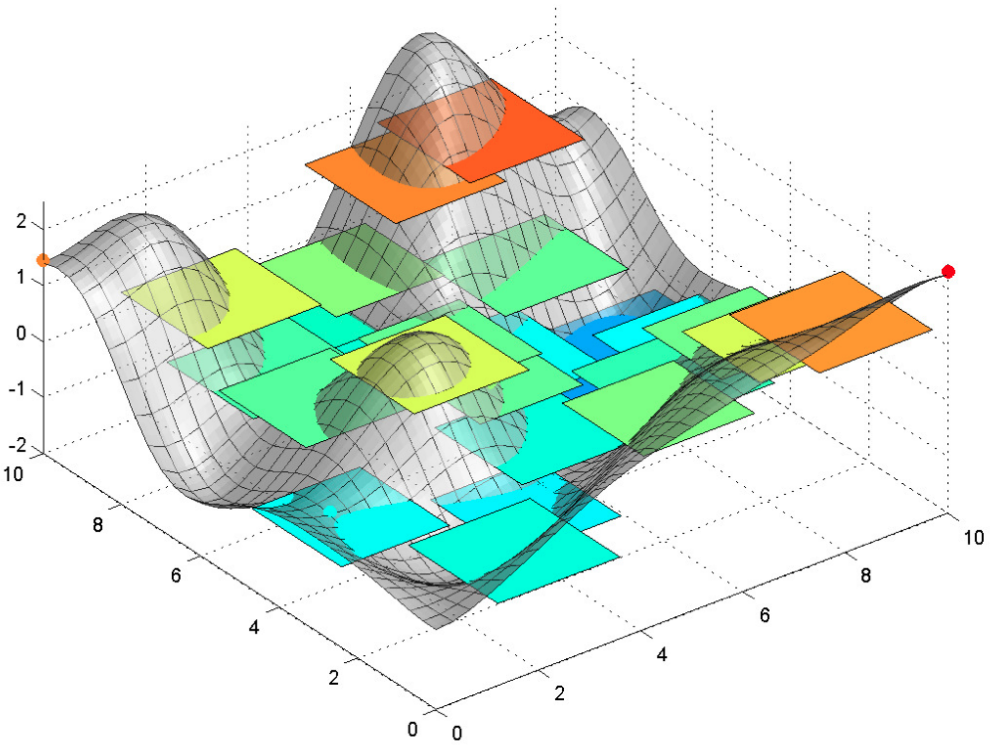
Without using complex support kernels (in this particular case, area kernels), the problem can be solved with traditional GPs by using the ozone data to fit an ozone level surface over the map, then using the estimation at the centroid of a particular zip code to represent the readings at the whole area (Carlin et al., 1999). Gelfand et al. argue that this approximation fails to capture spatial association and variability by treating a GP estimate as an observed value.

Instead, they assumed that the variable observed is continuous and comes from an underlying spatial process. We can denote this process by $Y(\mathbf{p})$, for points $\mathbf{p} \in D$ (where D is our region of interest). For data observed in discrete locations, as is the case of ozone levels in the example they present, this is the process to be inferred. However, patient information comes from averages over regions of space. That is, for each region $A \in D$,

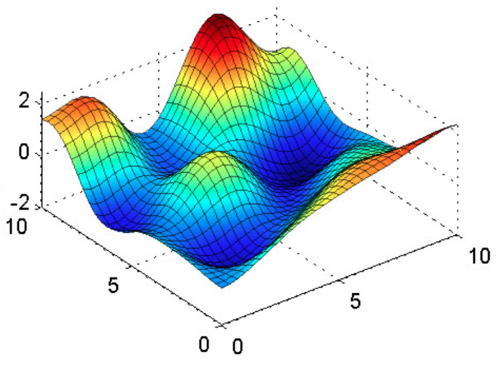
$$Y(A) = |A|^{-1} \int_A Y(\mathbf{p}) d\mathbf{p}, \tag{3.1}$$

where $|A|$ represents the area of region A. This integral is an average of random variables, therefore the assumption of an underlying spatial process is only appropriate if the area data can be seen as averaging over point data. Valid examples are continuous environmental measurements, such as temperature, pressure or elevation. Conversely, this assumption is inappropriate for discrete measurements, like population, as a particular point of space cannot hold any population. Proportions are also usually inappropriate, because even though they take values from a continuous range, the corresponding point data could be discrete (for instance: the proportion of unemployed people is continuous over the population of an area, but binary for each person).

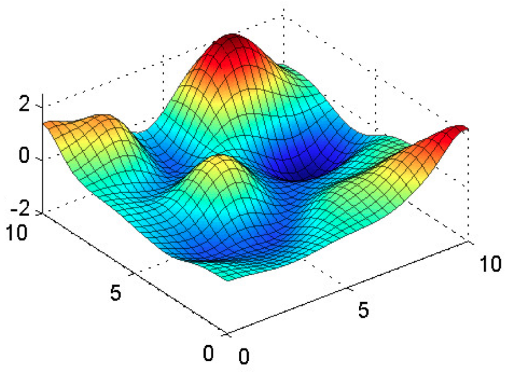
Reid, in his doctoral thesis (2011), formulates a way of using geometrical elements as support for a GP kernel, illustrated in Figure 3.1.



(a) Colored rectangles and points are the input data, grey surface is the prediction.



(b) Ground truth



(c) Estimate

Figure 3.1: Example of Gaussian process using kernels supported on areas and points. Taken from Reid (2011, section 3.3).

Besides points x, the higher-dimensional elements that Reid uses are lines L (when the domain is one-dimensional) or areas A (when two-dimensional), such that Equation 3.1 becomes:

$$\begin{aligned} Y(L) &= |L|^{-1} \int_{x \in L} Y(x)\,dx \text{ and} \\ Y(A) &= |A|^{-1} \iint_{x \in A} Y(x)\,dx. \end{aligned} \tag{3.2}$$

Having established the relationships between points and lines or areas, Reid

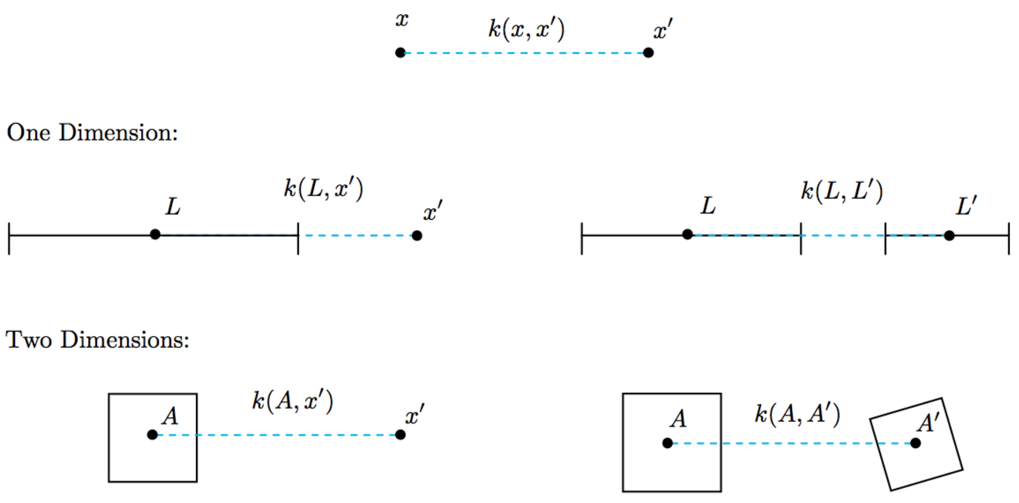


Figure 3.2: Schematic representation of kernels using points, lines and areas. Reid (2011, section 3.3).

proceeds to define covariance functions that accept these elements as input. The equations for these kernels, illustrated in Figure 3.2, are:

$$\begin{aligned} K_{x,x'} &= k(x, x'), \\ K_{L,x'} &= k(L, x') = \frac{1}{|L|} \int_{x \in L} k(x, x')\,dx, \\ K_{L,L'} &= k(L, L') = \frac{1}{|L||L'|} \int_{x \in L} \int_{x' \in L'} k(x, x')\,dx\,dx', \\ K_{A,x'} &= k(A, x') = \frac{1}{|A|} \iint_{x \in A} k(x, x')\,dx, \\ K_{A,A'} &= k(A, A') = \frac{1}{|A||A'|} \iint_{x \in A} \iint_{x' \in A'} k(x, x')\,dx\,dx'. \end{aligned} \tag{3.3}$$

Analysing the above integrals makes it easy to see that, due to their being inversely proportional to the size of the higher-dimension element, the complex support kernels tend to zero as $|L|$ or $|A|$ are increased—that is, as the resolution becomes coarser. A quick look through the covariance functions shown in Section 2.2.2 shows that it can be challenging to integrate analytically several of them. Reid chooses the squared exponential function to demonstrate the technique, for two reasons. The first is that it is fairly easy to integrate to obtain the line-to-point and line-to-line integral kernels. If L is taken to be a line segment between points x_l and x_r, we obtain:

$$\begin{aligned}
K_{x,x'} &= \sigma_f^2 \exp\left(-\frac{|x-x'|}{2l^2}\right), \\
K_{L,x'} &= \frac{\sigma_f^2}{x_r - x_l} \int_{x_r}^{x_l} K_{x,x'} dx \\
&= \sqrt{\frac{\pi}{2}} \left(\frac{\sigma_f^2 l}{x_r - x_l}\right) \left[\operatorname{erf}\left(\frac{x - x_r}{\sqrt{2}l}\right) - \operatorname{erf}\left(\frac{x - x_l}{\sqrt{2}l}\right)\right], \\
K_{L,L'} &= \frac{\sigma_f^2}{(x_r - x_l)(x'_r - x'_l)} \int_{x'_r}^{x'_l} K_{L,x'} dx' \\
&= \sqrt{\frac{\pi}{2}} \left(\frac{\sigma_f^2 l}{(x_r - x_l)(x'_r - x'_l)}\right) \left[(x_r - x'_l)\operatorname{erf}\left(\frac{x_r - x'_l}{\sqrt{2}l}\right) - (x_l - x'_l)\operatorname{erf}\left(\frac{x_l - x'_l}{\sqrt{2}l}\right)\right. \\
&\quad \left. + (x_l - x'_r)\operatorname{erf}\left(\frac{x_l - x'_r}{\sqrt{2}l}\right) - (x_r - x'_r)\operatorname{erf}\left(\frac{x_r - x'_r}{\sqrt{2}l}\right)\right] \\
&\quad + \frac{l^2}{2(x_r - x_l)(x'_r - x'_l)} \left[\exp\left(-\frac{|x_r - x'_l|}{2l^2}\right) - \exp\left(-\frac{|x_l - x'_l|}{2l^2}\right)\right. \\
&\quad \left. + \exp\left(-\frac{|x_l - x'_r|}{2l^2}\right) - \exp\left(-\frac{|x_r - x'_r|}{2l^2}\right)\right].
\end{aligned} \tag{3.4}$$

The second reason is that the function is separable in a two-dimensional Euclidean space; that is,

$$\begin{aligned}
K_{x,x'} &= \exp(|x - x'|^2) \\
&= \exp((x_1 - x'_1)^2) \exp((x_2 - x'_2)^2).
\end{aligned} \tag{3.5}$$

This property means that the area-to-point and area-to-area functions for a rectangular region can be represented as the cross product of its diagonals. If we take an input vector $\mathbf{x}$ containing both rectangles A (with diagonals D_1 and D_2) and points $\mathbf{p}$, we obtain:

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix} A \\ \mathbf{p} \end{bmatrix}, \\ k(\mathbf{x}, \mathbf{x}') &= \begin{bmatrix} k(D_1, D_1') & k(D_1, \mathbf{p}') \\ k(\mathbf{p}, D_1') & k(\mathbf{p}, \mathbf{p}') \end{bmatrix} \times \begin{bmatrix} k(D_2, D_2') & k(D_2, \mathbf{p}') \\ k(\mathbf{p}, D_2') & k(\mathbf{p}, \mathbf{p}') \end{bmatrix} \end{aligned} \tag{3.6}$$

We have noted, however, that the computation of covariance functions is the most computationally demanding part of the GP algorithm. Reid lessens this burden by using a piecewise approximation to the isotropic squared exponential kernel. However, an isotropic kernel is not a good option for robotic mapping, since it presumes equal variation of the observed phenomenon in the direction of each axis which does not always correspond to the way obstacles are distributed in real world.

The method outlined above requires extra calculations, since we need to calculate two full covariance matrices, and this represents an increase in computational cost. Besides that, the mathematical equations involved in one- and two-dimensional kernels are long and complicated (see Equations 3.4 and 3.6), representing a challenge to implementation.. Moreover, were the work in this thesis to be extended to three-dimensional data (as is intended), the equations would become even larger, making this approach impractical.

3.2.2 Gaussian process occupancy mapping

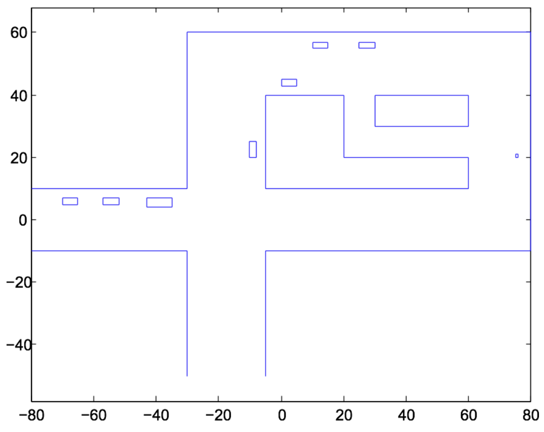
As discussed in Section 2.1, discrete representations of space bring a number of shortcomings with them. Their efficiency relies on a highly restrictive assumption about the environment: that cells are independent and identically distributed (iid). This assumption disregards spatial correlations that might be useful in inferring occupation in unobserved regions. In order to avoid this problem, O'Callaghan (2012) proposed a mapping approach using GPs, transforming the mapping task to a continuous spatial classification problem. The method developed is called Gaussian process occupancy map (GPOM).

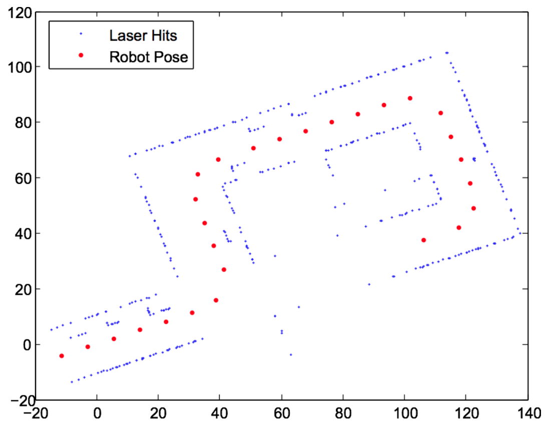
GPOMs are GP classifiers that take rangefinder sensor data and use it to classify regions of space as occupied or free. Figure 3.3 illustrates the method with a simple synthetic dataset. It is important to notice that the GP outputs a continuous surface of probability (Figure 3.3e), which is then filtered to obtain a map (Figure 3.3f) that can be compared to a traditional OGM (Figure 3.3c).

The input points, though, cannot be limited to the laser hits—as those only indicate obstacles and give no information about the free areas. In order to indicate free areas, a sensor beam can be, for instance, represented as a collection of free-space points. However, that would cause the training set to be inflated very fast, and make the algorithm run even slower, since its complexity grows cubically with the number of input points.

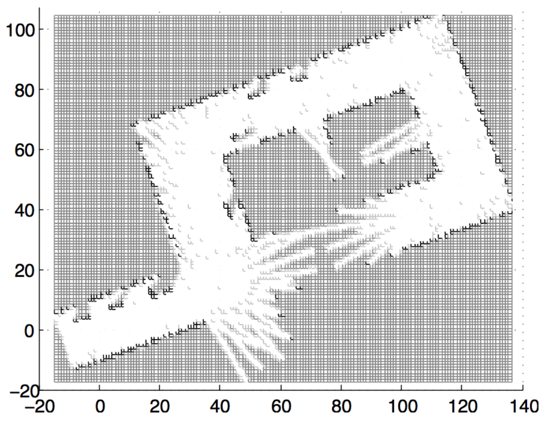
In order to represent the free areas without inflating the input set, O'Callaghan uses Gaussian processes with covariance functions supported on lines instead of points, as discussed in the previous chapter. This allows modelling the rangefinder beam as a continuous line segment between the robot's pose and the point where each beam taken from this location hit an obstacle.

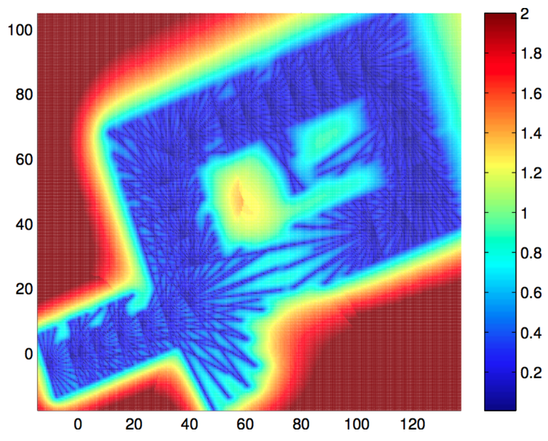
In order to allow for online learning, the covariance matrix can be stored

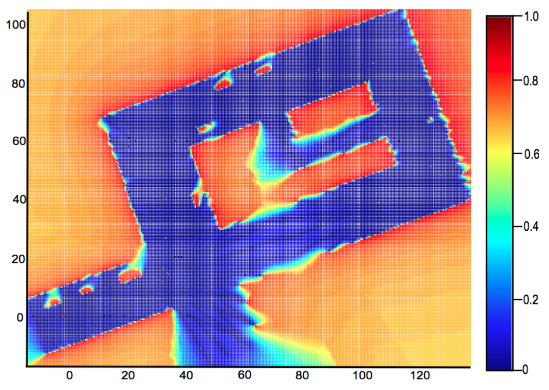

(a) Ground truth



(b) Laser returns and robot pose


(c) Traditional OGM


(d) Predictive variance of Gaussian process


(e) Probability of occupancy versus location

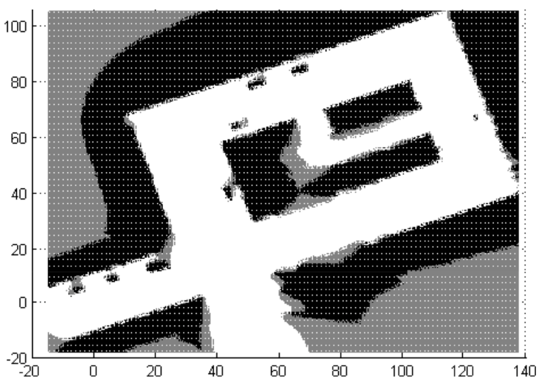

(f) Classified GPOM. Black = occupied ($p(\text{Occupied}) \geqslant 0.65$), white = free ($p(\text{Occupied}) \leqslant 0.35$), grey = unsure.

Figure 3.3: GPOM demonstration with simulated 2D dataset. O'Callaghan (2012).

and applied to multiple query points. Therefore, matrix inversion need not be performed more than once, which removes the $O(n^3)$ cost from further iterations. Instead, the covariance matrix is updated through a $O(n^2)$ operation. A comparison of performance with, and without, this optimisation is shown in

Figure 3.4a. To allow online learning, this optimisation is aided by an active sampling technique and by splitting the covariance matrix once it grows too large.

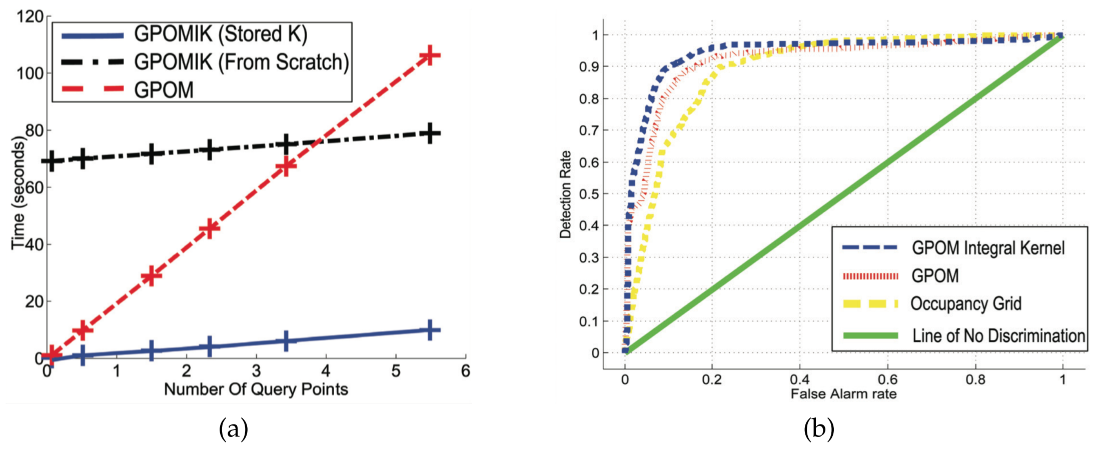


Figure 3.4: Time (a) and accuracy (b) comparison between the GPOMIK and the GPOM on the 3D simulated dataset. GPOMIK "from scratch" indicates the method where the kernel is recalculated in each iteration, while "stored K" represents optimised update. O'Callaghan and Ramos (2011).

Table 3.1: Quantitative comparison of experimental results. False positive rates are calculated for a 90% true positive rate. O'Callaghan and Ramos (2011).

	Area under the curve	False positive rate
GPOMIK	0.9441	10.1%
Previous GP Method	0.9162	79.57%
Occupancy Grid	0.8938	21.9%
No Discrimination	0.5	90%

Experiments with real and simulated data in two and three dimensions compare the GPOMs with and without integral kernels, as well as regular OGMs. The results are displayed in Figure 3.4 and Table 3.1. Gaussian process occupancy maps using integral kernels (GPOMIKs) were shown to perform better: they run faster then GPOMs and generate less false positives than both competing methods.

3.3 MULTI-SUPPORT KERNELS

One objective of this work is to propose a kernel that can use higher-dimensional support elements—lines, rectangles and hyperrectangles—for robotic mapping. Instead of integrating over the laser beams, however, the objective is to integrate over elements of the same dimension of the map. In other words, we'll use areas (when building 2D maps) and volumes (in 3D maps) as support to the covariance functions. It is also desirable that the method is able to handle arrays containing these geometrical features as well as points, or any combination between them—hence the chosen name, *multi-support kernel* (MSK).

The kernels obtained by the method described in Section 3.2.1 would, theoretically, fulfil these requirements. However, analytically integrating a more complicated kernel, such as the neural network (NN) or those of the Matérn class, can be exceedingly difficult. Numerical methods, on the other hand, can be computationally demanding and unstable. An alternative must be presented that is both easy to implement and allows to draw upon the richness and abundance of existing kernel functions.

In the next section, we propose a novel method to allow kernels to be calculated using data about complex geometrical shapes. From here onwards, these will be referred to as *elements of support* or *elements*, to differentiate them from data points.

3.3.1 Proposed kernel

To fulfil all the criteria priorly discussed, instead of integrating the traditional kernels as the methods discussed in Section 3.2, the element kernels are approximated by sampling the data points inside the element and calculating the mean of the kernels between all of them. Following that, for generic geometrical elements, Equation 3.3 can be restated as:

$$\begin{aligned}
K_{x,x'} &= k(x,x'),\\
K_{E,x'} &= k(E,x') = \frac{1}{|E|}\int_{x\in E} k(x,x')dx,\\
K_{E,E'} &= k(E,E') = \frac{1}{|E||E'|}\int_{x\in E}\int_{x'\in E'} k(x,x')dxdx',
\end{aligned} \tag{3.7}$$

which, given a finite number of measurements $x \in E$, can be approximated as:

$$\begin{aligned}
K_{x,x'} &= k(x,x'),\\
K_{E,x'} &= k(E,x') = \frac{1}{n_E}\sum_{x\in E} k(x,x'),\\
K_{E,E'} &= k(E,E') = \frac{1}{n_E n_{E'}}\sum_{x\in E}\sum_{x'\in E'} k(x,x'),
\end{aligned} \tag{3.8}$$

where E is a geometrical element of any dimension and n_E indicates the number of data points sampled from E. Being constructed by direct sum, it is a valid positive semi-definite (PSD) kernel (Hastie and Tibshirani, 1990; Rasmussen and Williams, 2006, section 4.2.4). A similar and more rigorous derivation can also be found in Gelfand, Zhu, and Carlin (2001, section 2), and an analogous approach was also used by Muandet and Schölkopf (2013, section 3.2) to calculate kernels over probability distributions.

We call the kernel described in Equation 3.8 a multi-support kernel (MSK), because it allows us to calculate element-to-element and element-to-point covariances. Their algorithmic forms are rather simple[⋆]:

Algorithm 3.1: Element-to-element kernel algorithm.

```
input : E1, E2, k(), par
    # element, point, covariance function and parameters
Kpp := []
Kep := [] # empty arrays
for all p1 in E1:
   for all p2 in E2:
      Kpp.append(k(p1, p2))
   Kep.append(mean(Kpp))
K := mean(Kep)
output : K
    # covariance between E1 and E2
```

Algorithm 3.2: Element-to-point kernel algorithm.

```
input : E1, p2, k(), par
    # element, point, covariance function and parameters
Kpp := [] # empty array
for all p1 in E1:
   Kpp.append(k(p1, p2))
K := mean(Kpp)
output : K
    # covariance between E1 and p2
```

The above algorithms clearly use elements for support, however each alone is constrained to a single support. In order to enable them to calculate a full covariance matrix between two sets of observations containing any combi-

⋆ Worthy of mention is that, although the way these algorithms are written here suggests a calculation between a single element and a single point, both of them can be vectorised, allowing the input of arrays of elements and points.

nation of supports (and thus making it truly multi-support), we return to Equation 3.6, which now takes a much simpler form:

$$
\begin{aligned}
\mathbf{x} &= \begin{bmatrix} E \\ \mathbf{p} \end{bmatrix}, \quad \mathbf{x}' = \begin{bmatrix} E' \\ \mathbf{p}' \end{bmatrix}, \\
k(\mathbf{x}, \mathbf{x}') &= \begin{bmatrix} k(E, E') & k(E, \mathbf{p}') \\ k(\mathbf{p}, E') & k(\mathbf{p}, \mathbf{p}') \end{bmatrix}.
\end{aligned}
\tag{3.9}
$$

We can use this result to build, using Algorithms 3.2 and 3.1, a novel multi-support covariance function able to handle inputs containing both elements and points:

Algorithm 3.3: Multi-support kernel algorithm.

```
input : x1, x2, k(), par
    # array, array, covariance function and parameters.
    # arrays are assumed to be of the format x := [E, p]$^\top$

Kee := kee(x1.E, x2.E, k(), par)   # element-to-element covariance
Kep := kep(x1.E, x2.p, k(), par)   # element-to-point covariance
Kpe := kep(x1.p, x2.E, k(), par)$^\top$  # point-to-element covariance
Kpp := k(x1.p, x2.p, par)          # point-to-point covariance

K := [[Kee, Kep],
      [Kpe, Kpp]]

output : K
    # covariance between x1 and x2
```

The greatest advantage of the proposed kernel is that we can now aggregate several input points into a single element. When we use it as a covariance function for a GP, it can significantly reduce the size of the covariance matrix, which directly impacts the cost of its inversion.

3.4 MAPPING WITH MULTI-SUPPORT KERNELS

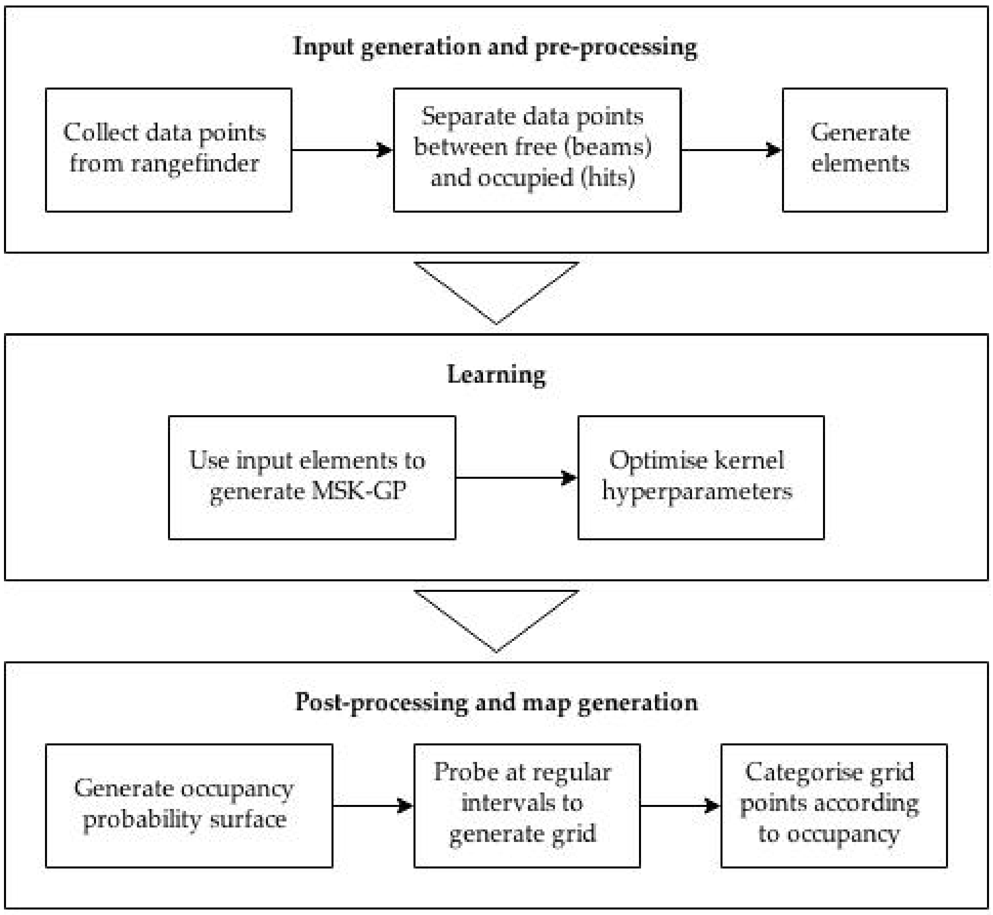
A covariance function able to produce reduced matrices from the input dataset is a valuable tool. As discussed in Chapter 2, the Gaussian process (GP) algorithm requires a matrix inversion, which has a complexity of $O(N^3)$ with the size of the matrix, and using it for mapping is a challenge because laser rangefinders can generate thousands of data points per seconds.

By using a multi-support kernel (MSK), points can be grouped together in spatial elements which count as a single input in the array. The effective reduction in computation time will depend on how many points can be grouped into a single element, but if we group a dataset containing N entries into elements containing an average of n points each, the input size becomes roughly N/n. Since its dependency with the number of entries is cubical, the complexity of the inversion is n^{-3} of the original.

It should then become obvious that the way in which the elements are defined hugely impacts the efficiency of MSK. The more points there are inside each element, the smaller the covariance matrix will be and the faster the result will be. However, as the size of the elements grow, the resolution becomes coarser and the map loses its predictive ability.

Once the elements are generated, they are passed as input to the multi-support Gaussian process occupancy map (MS-GPOM) algorithm, outputs the probability of occupancy in the form of a three-dimensional surface. The occupancy map is then obtained by generating a point grid and categorising each point, according to its probability of occupancy, into one of three states: *free*, *occupied* or *unknown*. The whole process is summarised in Figure 3.5

Several variables can affect the performance of the algorithm, not all of which



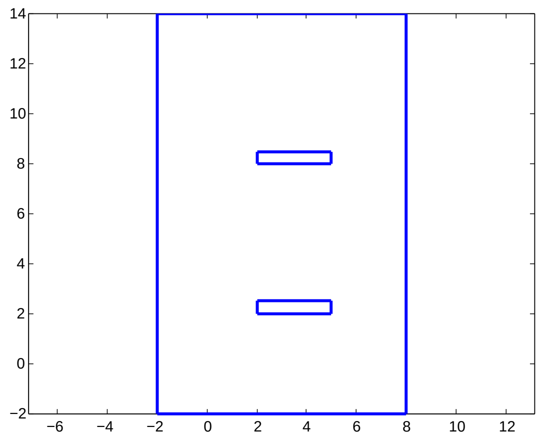
Input generation and pre-processing
Collect data points from rangefinder
Separate data points between free (beams) and occupied (hits)
Generate elements
Learning
Use input elements to generate MSK-GP
Optimise kernel hyperparameters
Post-processing and map generation
Generate occupancy probability surface
Probe at regular intervals to generate grid
Categorise grid points according to occupancy

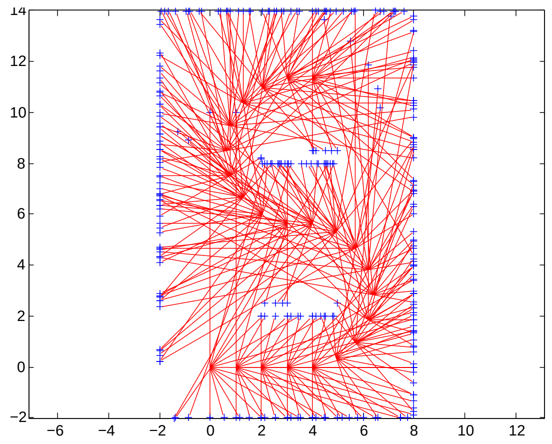
Figure 3.5: Flowchart for MS-GPOM.

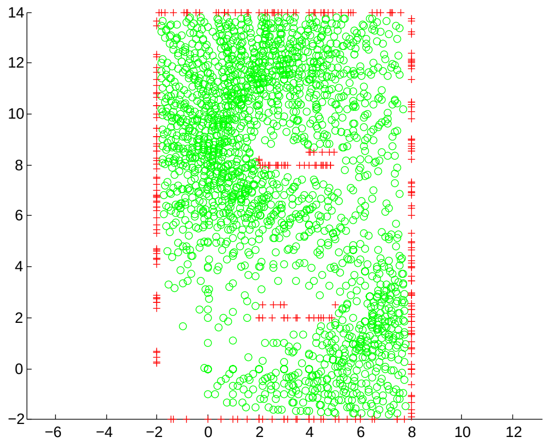
can be efficiently optimised through the learning step described above. This will be discussed in greater detail in Chapter 4. However, for instructional purposes, each step of the algorithm outlined in Figure 3.5 will first be illustrated in the next section.

3.4.1 MAPPING WITH A SYNTHETIC DATASET

In this section, we use simulation to illustrate the main aspects of the proposed algorithm. The simulation models a robot equipped with a rangefinder moving within a 2 dimensional space. At each time step, the robot performs a number of rangefinder readings, then updates its pose (i. e. position and orientation) based on its previous pose and speed. A sample dataset is shown in Figure 3.6.


(a) Ground truth


(b) Rangefinder data: free beams in red and hits in blue.


(c) Free points in green, occupied in red.

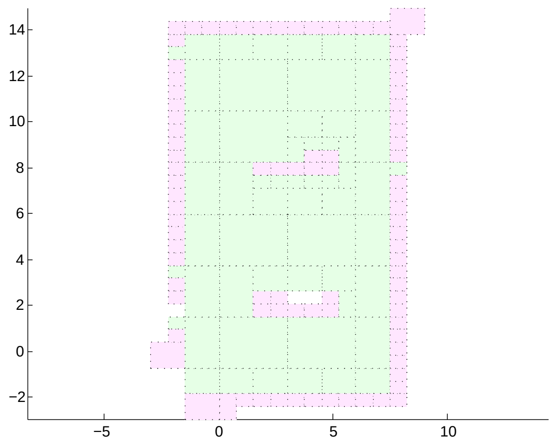

(d) Dataset partitioned into elements.

Figure 3.6: Sample synthetic dataset and data preprocessing. A robot equipped with a laser rangefinder with 180Â° field of view moves around in a room (a) generating a series of readings (b). These are then sampled and divided into free and occupied points (c). Lastly, these points are used to create the elements to be used as input by the MS-GP algorithm (d). All units displayed are arbitrary.

The first step is the pre-processing: from each beam (that is, the line connecting the coordinates of each hit to the position where the robot when the hit was detected), a set of equally spaced free points is generated. The hits that are at the edge of the perceptual range (that is, the ones where it is unclear whether or not an obstacle exists) are discarded at this stage. The closest free point to each hit can be discarded to make the element creation easier, avoiding elements containing both free and occupied points—however, it also results in wider uncertain boundaries between free and occupied areas, a trade-off that must be kept in mind.

After the free points have been processed, the elements must be created. An initial element is defined by a polygon set around the data points. For the sake of simplicity, this work used exclusively rectangles with sides parallel to the Cartesian axes, although these restrictions are unnecessary. The initial element is divided following a logic similar to a quad-tree (Finkel and Bentley, 1974), and two things are taken into consideration before each division: the size of the element and the type of points inside it. If an element contains only one type of points inside (hits or free points) it is used as an input. If it has both free points and hits, it is divided in half perpendicular to its longest dimension, except if it is smaller than the smallest allowed size, a threshold defined by the user. In this example, we used a threshold of 1.0 (see Figure 3.6)[⋆]. The process is illustrated in Figure 3.7 and Algorithms 3.4 and 3.5. It should be noted that halving the threshold will generate approximately 2^D as many elements, where D is the number of dimensions.

Algorithm 3.4: Pre-processing algorithm.

```
input : B, thresh
    # beams, size threshold

[Xf, Xo] = split_beams(B, thresh)
    # samples the beams at regular distances and returns one
    # vector of free and one of occupied coordinates

E = []
E0 = [Xf; Xo]$^\top$  # initial element containing all the data

E = find_elements(E0, thresh) # Algorithm 3.5

output : E
    # list of elements created from the data
```

⋆ It is important to stress that the threshold does not represent the smallest element dimension possible. Rather, it means that elements with dimensions larger than those would still be divided—therefore, a threshold of 1.0 means that the smallest dimension found in an element in the input dataset can be as small as $0.5 + \epsilon$.

Algorithm 3.5: Recursive algorithm to find valid elements.

```
input : E, thresh
    # list of elements, size threshold

    # this algorithm takes as input elements that are too large or
    # contain mixed entries and ultimately returns a list of
    # elements that obey all constraints imposed

[Xf; Xo] = E
if Xf = empty and Xo = empty:
   E = [] # empty elements are discarded
   return

if E ⩽ thresh or Xf = empty or Xo = empty:
    # elements that contain only one type of entry or are smaller
    # than the size threshold do not get divided any further
   return

[E1, E2] := divide(E)
    # divides E in half by the longer axis and creates two new
    # elements, splitting the free and occupied points accordingly

E := [find_elements(E1, thresh), find_elements(E2, thresh)

output : E
    # list of valid elements
```

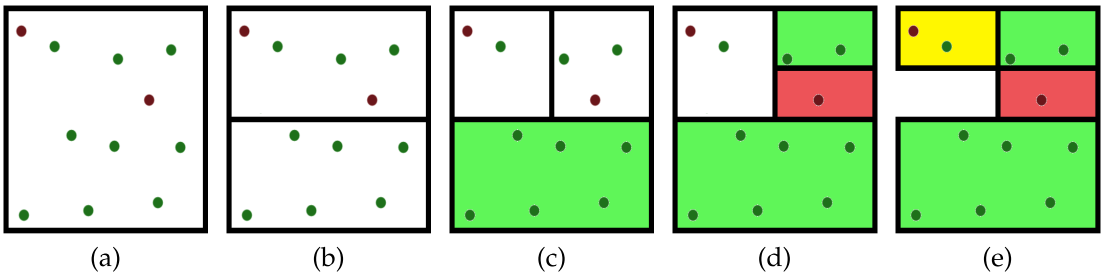


Figure 3.7: Element generation process demonstration. For this example, we use an initial dataset (a) spanning a 0.9×1.0 area and a threshold of 0.25. Since it contains both free (green) and occupied (red) points, it is divided perpendicular to the longest axis (b). The lower element only contains free points, so it is marked free and not divided any further; the upper element is divided again (c). The upper right element can be divided one more time (d), yielding one free and one occupied element. The upper left element, however, after an additional division (e) yields an empty element (which is discarded) and an element containing one free and one occupied point that can't be divided any further (because it's longer axis is smaller than the threshold). Therefore, it is ambiguous, with 50% chance of occupancy (yellow), as determined by the proportion of occupied and free points.

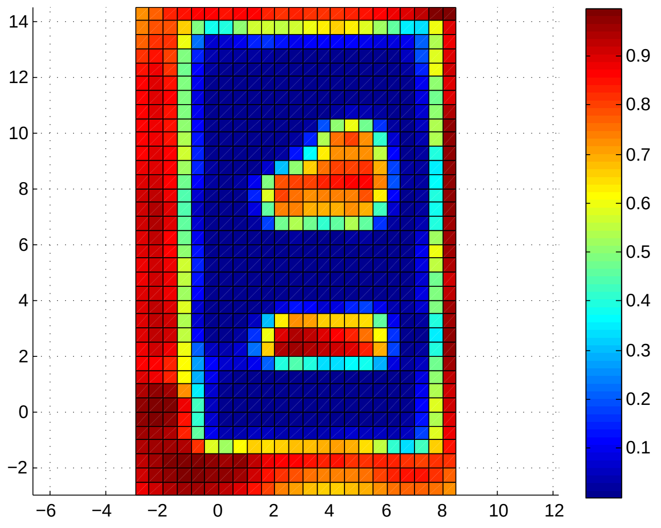
This method of division has a desirable property. If we look at a single time step, since the beams from a rangefinder have diverging trajectories, the free points are abundant and grouped more densely near the robot's location, while the hits are at the edges and further apart. Therefore, the elements generated near the robot will be larger, so they group more points together and ultimately have a larger impact in the size of the covariance matrix. At the same time, closer to the hit, the size of the elements will decrease, which allows for extra resolution where it matters most.

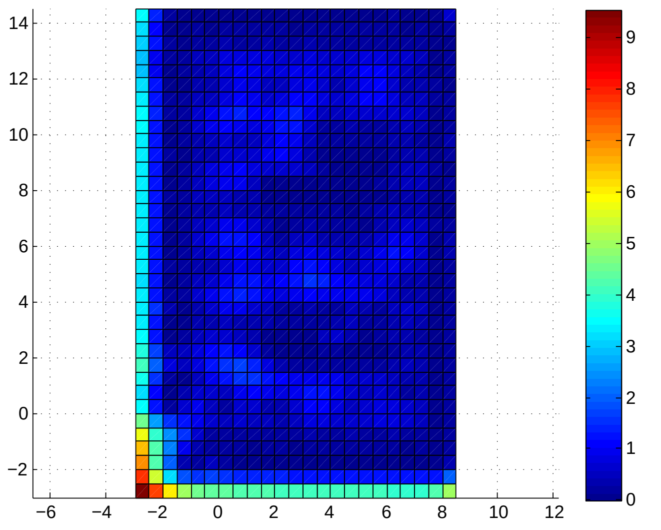
After the elements are prepared, they are ready to be used to train a GP. In this example, a squared exponential covariance function is used, trained using simulated annealing (Kirkpatrick et al., 1983; Černý, 1985) to minimise the log marginal likelihood. In Chapter 4, we compare these options to the Matérn 3 covariance function, and the Broyden-Fletcher-Goldfarb-Shanno method (BFGS) (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) is tested as an optional optimiser.

After the GP is trained, a test dataset composed of points uniformly distributed over the entire region spanned by the input points can be used to set a grid, with spacing of 0.5. The outputs of the GP regression are the prediction and the variance. A sigmoid function then flattens the prediction to confine it to the interval $[0; 1]$, reflecting the probability of occupancy of each point. The function used for this purpose in the experiments in this work takes the form:

$$\sigma(\mu, \nu) = \Phi\left(\frac{(\alpha * \mu + \beta)}{\sqrt{1 + \alpha^2 * \nu}}\right), \tag{3.10}$$

where μ is the prediction, ν is the variance, Φ is the normal cumulative distribution function and α and β are parameters that can be experimentally chosen. A pair of thresholds can subsequently be used to classify the grid between the three occupancy states traditionally used for OGMs. For all maps


(a) Probability of occupancy


(b) Variance

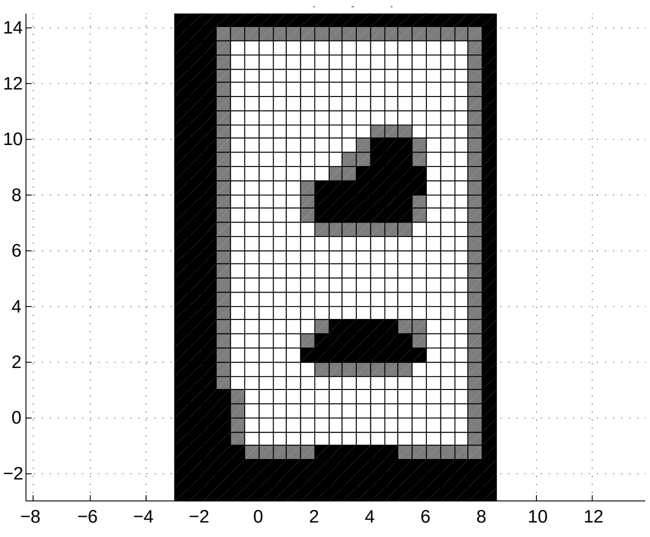

(c) Occupancy map.

Figure 3.8: Sample output. For the occupancy map (c), if a cell is less than 33% likely to be occupied, it is marked as "free" (white), and if it's more than 66% likely, it is marked as "occupied" (black). All remaining cells are "unknown" (grey).

presented in this thesis, unless otherwise stated, a point was considered free if its predicted occupancy probability was below 33%, and occupied if above 66%. Results can be seen in Figure 3.8, and the algorithm is Algorithm 3.6.

Algorithm 3.6: MS-GPOM algorithm.

```
input : B, par0, x
    # beams, parameters, test points
[thresh, par, alpha, beta] := par0
E := pre_process(B, thresh) # Algorithm 3.4
m, var, lml := gp(E, msk(), par, x)
    # where gp is Algorithm 2.2, msk is Algorithm 3.3 and
    # par contains sig and the kernel's hyperparameters
map := sigmoid(m, var, alpha, beta) # Equation 3.10
for all i in map:
   if map[i] < 0.33:
      map[i] := 0   # free
   else if map[i] < 0.66:
      map[i] := 0.5 # unknown
   else:
      map[i] := 1   # occupied
output : map, var, lml
    # prediction at x, variances and log-marginal likelihood
```

3.5 SUMMARY

In this chapter, we have reviewed some previous work on change of support for kernel methods. Reid's work on area kernels for image analysis and O'Callaghan work on line kernels for robotic mapping laid the path for this work.

Multi-support kernels have been introduced. They offer an easy implementation that allows any kernel to be supported on multi-dimensional geometrical elements rather than points. They do not rely on integration and are able to generate reduced covariance matrices, making them suitable for applications that rely on large amounts of spatially correlated data.

These qualities make MSKs well suited for robotic mapping. A procedure for creating geometrical elements from rangefinder data is described, and the algorithm for mapping with MS-GP is outlined and demonstrated for simulated data.

In the next chapter, we further explore the MS-GPOM algorithm. Some choices of parameters and their effects on its performance are demonstrated. It is also benchmarked against GPOMIK.

Chapter 4

EXPERIMENTS

With the aid of multi-support kernels, spatial elements in the data set can be used to aggregate several points into a single data entry, potentially reducing dimensionality and so accelerating the matrix inversion necessary for Gaussian process modelling and learning. However, when mapping with MSK-GPs, there are numerous variables that can affect the performance of the algorithm. Not all of them are tuned in the GP optimisation, requiring tests to measure the effects of each.

In this chapter, we conduct experiments to measure how each parameter affects the map generated. The proposed algorithm is also benchmarked against the GPOMIK algorithm.

Bold, persistent experimentation is the hallmark of good science.
— GLaDOS

4.1 INTRODUCTION

In Chapter 3, we introduced the concept of *change of support* for kernel methods, reviewing an example from the field of image analysis. A novel approach was proposed for multi-support kernels (MSKs), that does not require integration of kernel functions and can be used with any existing kernel. We also reviewed a method for continuous environment mapping using Gaussian processes (GPs), and proposed an approach using MSKs to generate Gaussian process occupancy maps (GPOMs).

In this chapter, we test the performance of the proposed multi-support Gaussian process occupancy map (MS-GPOM) algorithm under different conditions to analyse the influence of certain parameters in its predictive ability. The algorithm's time performance is also compared to the Gaussian process occupancy map using integral kernels (GPOMIK) algorithm both for synthetic and real datasets.

SYSTEM SPECIFICATIONS AND EXPERIMENTAL SETUP

All the experiments described in this chapter were performed in a computer equipped with a 3.2GHz Intel Core™ i5-3470 and 8GB RAM, running Ubuntu 14.04.2 LTS ("Trusty Tahr") 64 bit on a 3.13.0-46 generic GNU/Linux kernel and MATLAB® version 8.3.0.532 (R2014a).

For all the experiments, accuracy is measured in terms of the area under the receiver operating characteristic (ROC) curve. The false positive rate (FPR) for a fixed true positive rate (TPR) of 95% is offered as an additional performance metric.

4.2 PARAMETERS OF THE MODEL

There are some user-specified parameters that affect the performance of the model, interfering with its time efficiency and with the results obtained. We will demonstrate them in this section, using the same dataset as was used in the example in Chapter 3. For all the training benchmarks, to avoid the effects that random initialisation may have on training times, the initial value of all hyperparameters was set to $\mathbf{1}$ (i. e., all-ones vectors of the appropriate dimension).

A. *Element Generation*

The key contribution of this work is the addition of predictions supported on higher-dimension geometrical elements within the dataset. The input points are divided between these elements, and this allows the number of entries in the covariance matrix to be reduced, sometimes drastically. However, there is a tradeoff between the size of the covariance matrix and the predictive accuracy of the method.

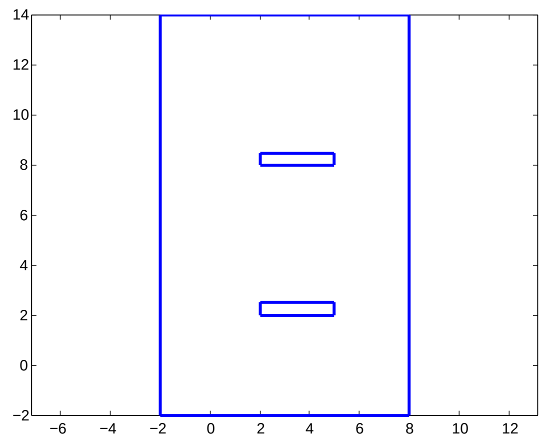
We discussed in the previous chapter how, as elements become larger, the number of points contained in each element increases, creating fewer entries in the matrix. In spite of that, elements can occasionally contain regions that have been sparsely probed, and a region's occupancy cannot always be reliably inferred by the occupancy of the entire element. As the size of the elements becomes smaller, this problem is alleviated, but at the price of larger covariance matrices, until the limit where each element contains a single point is reached (amounting to a traditional GP). Figure 4.1 illustrates the elements generated at different thresholds.

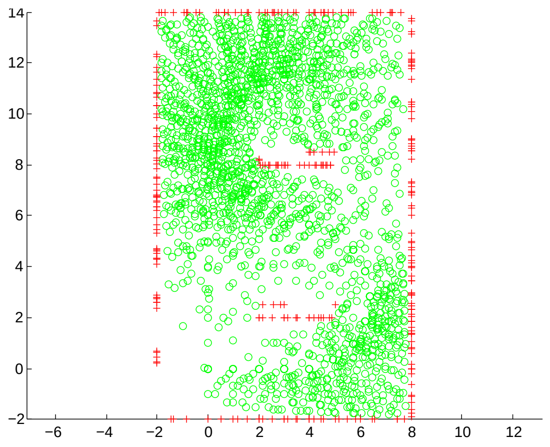
B. *Covariance Function*

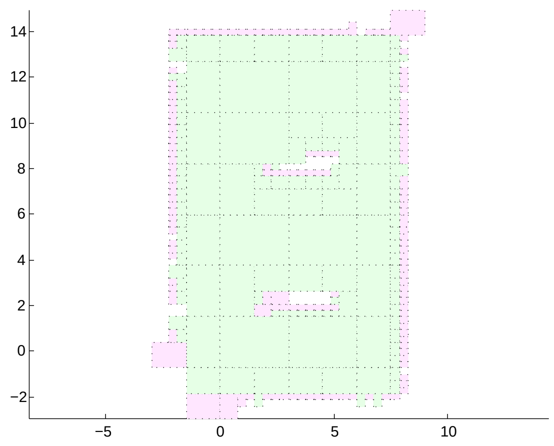
As discussed in Section 2.2.2, different covariance functions determine different properties for the functions in the model. They also differ in the number of hyperparameters and the computational cost. Therefore, it should come at no surprise that the choice of covariance function also affects the training time and the outcomes of the algorithm.

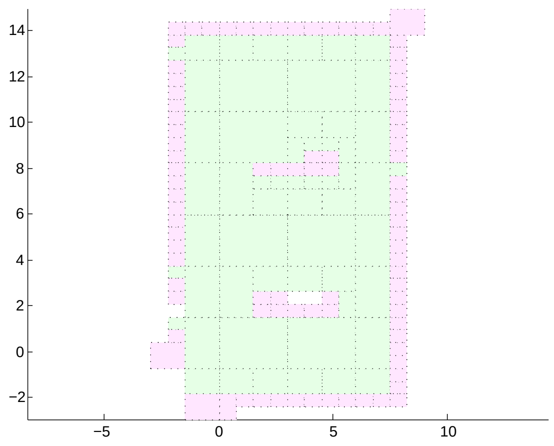
C. *Optimiser*

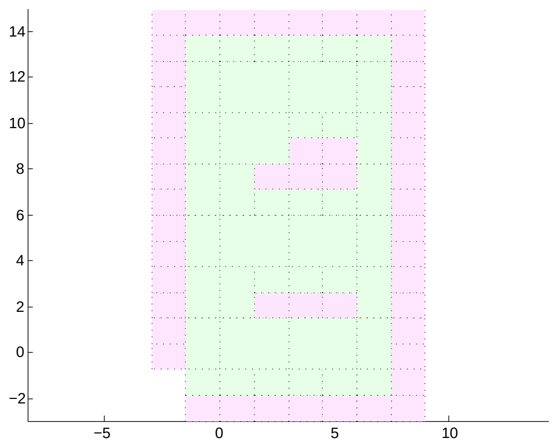
To train a GP, there is no standard choice of optimiser, and several can be used. Since optimisation is not the focus of this work, the optimisers chosen were the Broyden-Fletcher-Goldfarb-Shanno method (BFGS) cubic line search built into MATLAB's `fminunc` minimisation function, and simulated annealing, which is the option chosen by O'Callaghan (2012). BFGS is a quasi-Newton method and the faster of the two, however it is also more likely to get stuck in local minima. Simulated annealing, while slower, handles problems with a larger number of local minima more robustly.


(a) Ground truth


(b) Dataset: free (green) and occupied (red) points


(c) Elements, threshold=0.5


(d) Elements, threshold=1.0


(e) Elements, threshold=2.0

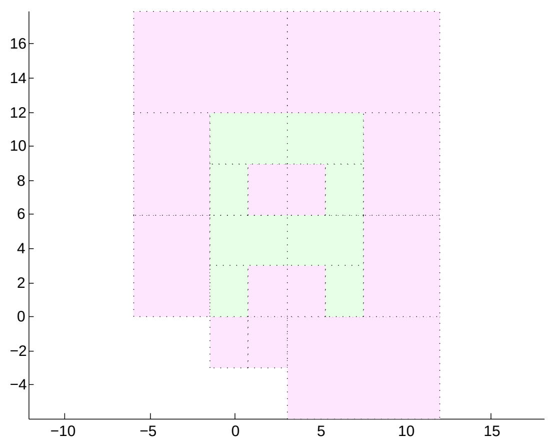

(f) Elements, threshold=4.0

Figure 4.1: Element size threshold impact on geometric element generation.

The influence of the above parameters on the training time of the algorithm can be seen in Table 4.1. As expected, the training time decreases as the element size threshold increases. BFGS optimisation runs considerably faster than simulated annealing, and the Matérn 3 kernel has a small advantage over the squared exponential. The choice of optimiser did not significantly influence the performance of the algorithms, and was omitted from the remaining analyses.

Table 4.1: Influence of the choice of covariance function, optimiser and element size threshold on the time it takes to learn the hyperparameters from the datasets presented in Figure 4.1.

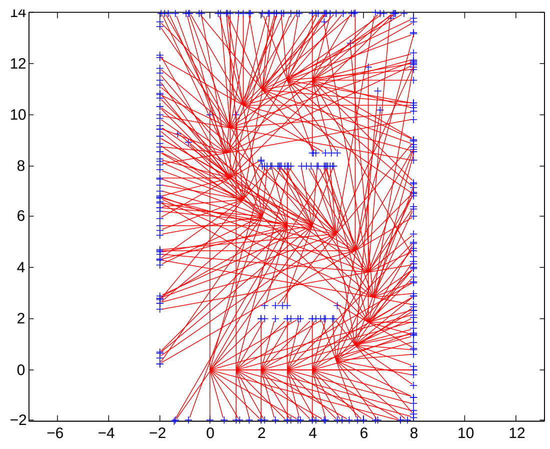
Kernel	Element size threshold	Optimiser	
		BFGS	Annealing
Squared Exponential	0.5	$2.80 * 10^3$s	$1.97 * 10^4$s
	1	686.35s	$6.41 * 10^3$s
	2	72.19s	166.81s
	4	28.43s	39.40s
Matérn 3	0.5	$1.92 * 10^3$s	$2.24 * 10^4$s
	1	$1.80 * 10^3$s	$6.64 * 10^3$s
	2	81.48s	157.03s
	4	20.98s	35.06s

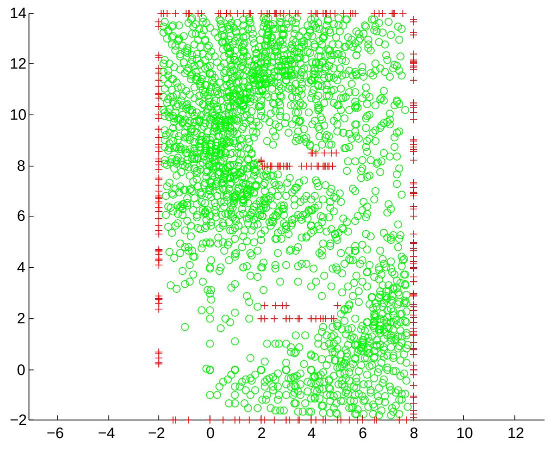
Table 4.2 shows the performance analyses for different choices of covariance function and element size threshold. We can see that the Matérn 3 kernel slightly outperforms the squared exponential kernel. Contrary to the expected result, smaller element size thresholds lead to longer running times and better accuracy only to an extent. A drop in performance can be noted once the smallest allowed elements become smaller than the smallest dimension of the obstacles, which provides a tentative way to set the element size threshold.

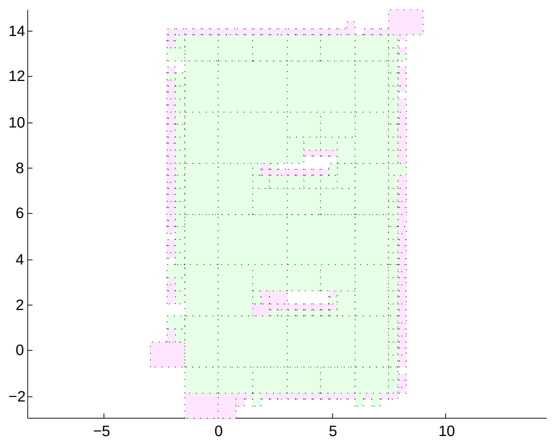
Table 4.2: Influence of the choice of covariance function and element size threshold on the time to create a map using parameters previously learned and accuracy of the resulting map.

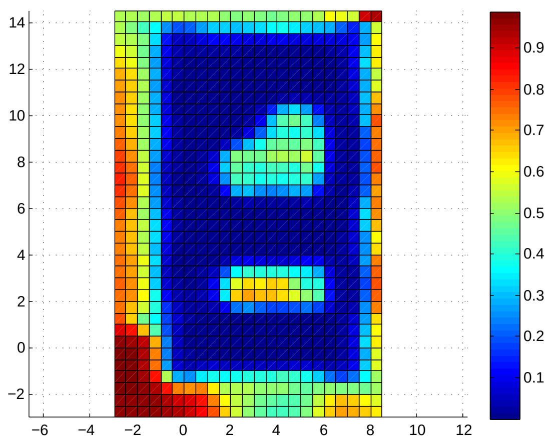
Kernel	Element size threshold	Running time	Accuracy	FPR when TPR = 95%
Squared Exponential	0.5	11.91s	0.9210	11.50%
	1	5.33s	0.9554	8.25%
	2	3.69s	0.8545	32.49%
	4	3.30s	0.6996	75.33%
Matérn 3	0.5	11.34s	0.9291	12.75%
	1	5.63s	0.9767	8.43%
	2	3.77s	0.8947	25.58%
	4	3.31s	0.7049	74.48%

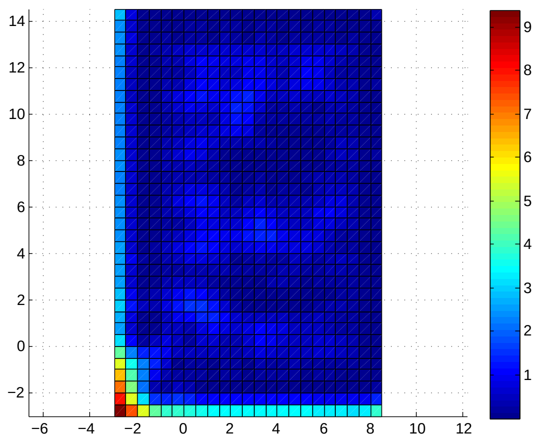
A step-by-step illustration of the map generation process, from the dataset to the final map, is displayed for two different threshold sizes from the table above in Figures 4.2 and 4.3. They give some insight into the reasons why the performance drops under thresholds of 1.0: comparing the input elements (subplot (c)) of each shows that, once the smallest elements become smaller than the obstacle, the peaks they generate in the prediction (subplot (d)) become less sharp. This provides some intuition for the tuning of this parameter, which wasn't provided to the optimisers.


(a) Initial Dataset


(b) Free and occupied points


(c) Input Elements


(d) Prob. Occupancy


(e) Predictive Variance

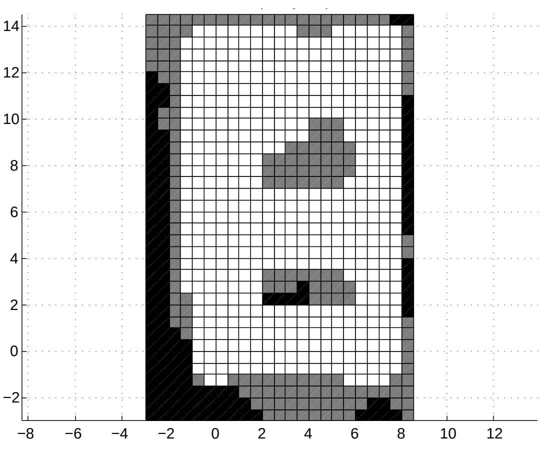
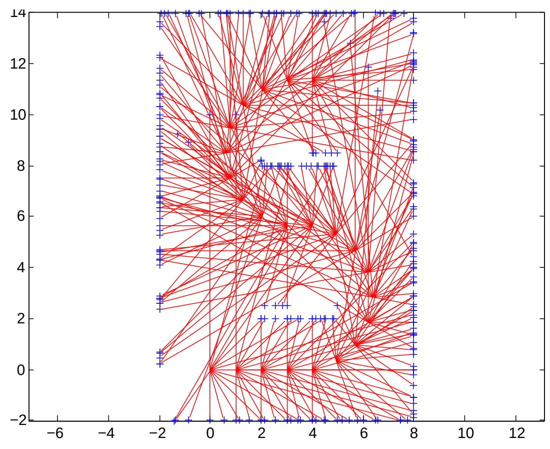
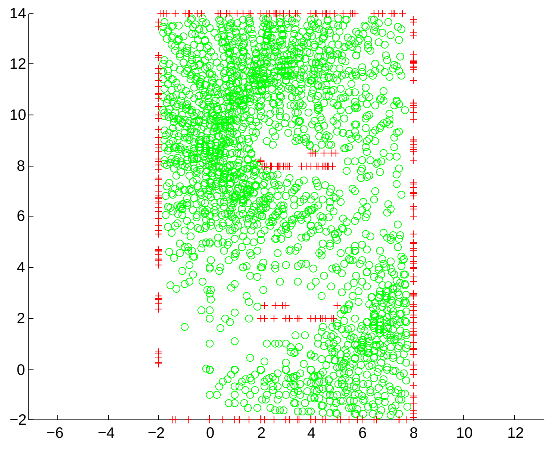
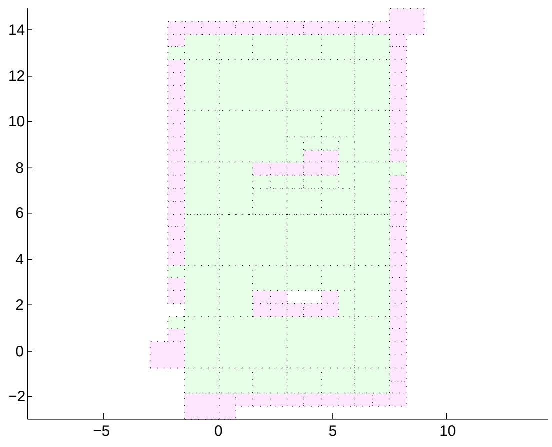
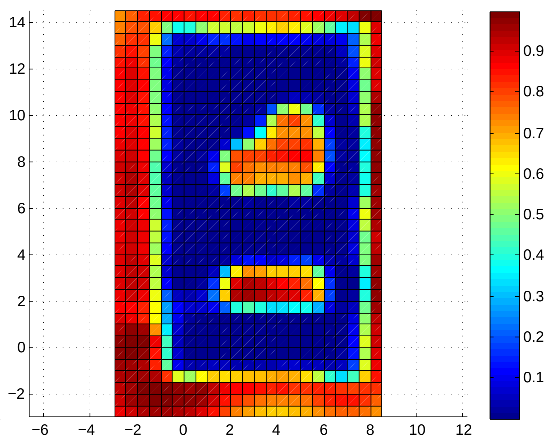

(f) Occupancy Map

Figure 4.2: Step-by-step map generation for a MS-GPOM with Matérn 3 kernel. Element size threshold is 0.5, training method was simulated annealing.

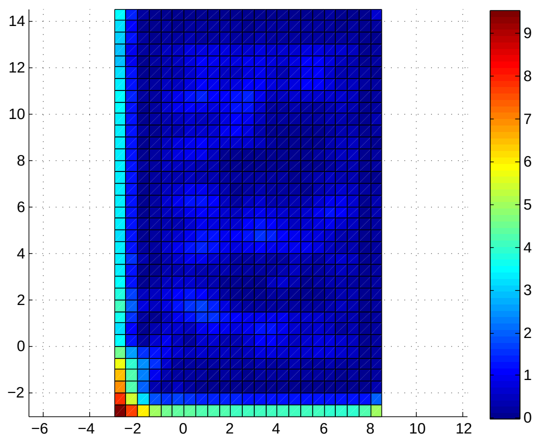

(a) Initial Dataset


(b) Free and occupied points


(c) Input Elements



(d) Prob. Occupancy



(e) Predictive Variance

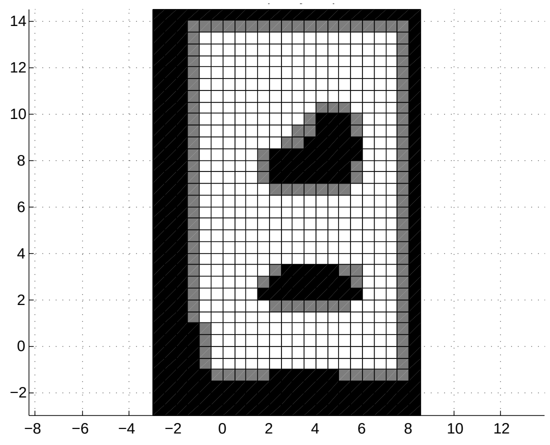

(f) Occupancy Map

Figure 4.3: Step-by-step map generation for a MS-GPOM with Matérn 3 kernel. Element size threshold is 1.0, training method was simulated annealing.

4.3 BENCHMARKS

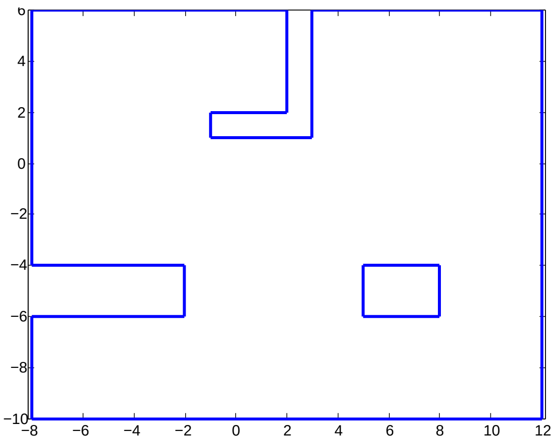
In all the benchmarks described in this section, the proposed algorithm was compared solely to GPOMIK (O'Callaghan, 2012). The reason behind this choice is that GPOMIK was the most efficient GPOM algorithm available at the time of the elaboration of this thesis, and it was shown in O'Callaghan and Ramos (2011) to outperform all the competing algorithms analysed (see Table 3.1 and Figure 3.4). The following experiments show that MS-GPOM was able to achieve comparable performance to GPOMIK, so comparison with previous algorithms was deemed unnecessary.

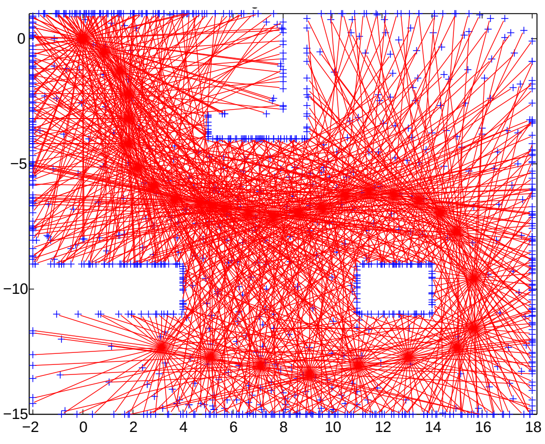
All benchmarks are made using identical test datasets for both methods. Input datasets differ only in the way each algorithm preprocesses the raw inputs.

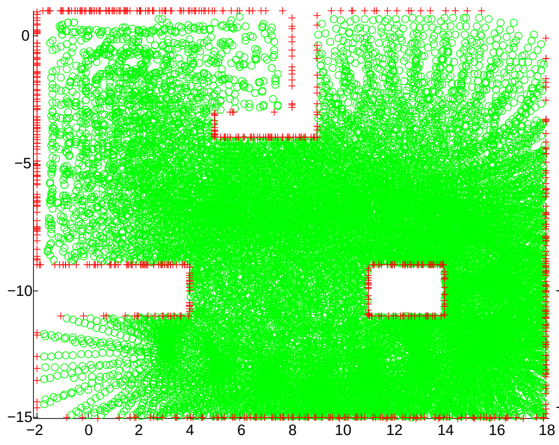
4.3.1 SYNTHETIC DATA

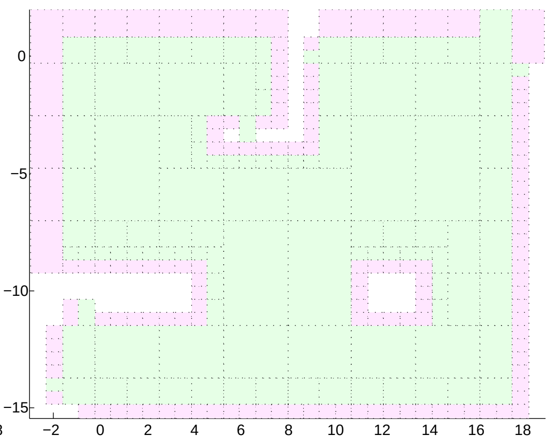
For this test, the dataset used was larger than the one used in Section 4.2, but generated by the same method. It is similar to the one used in O'Callaghan (2012, chapter 4). For the MS-GPOM, an element size threshold of 1.0 was chosen. For the GPOMIK, the active input sampling method described in O'Callaghan and Ramos (2011) was used. The dataset and inputs for each can be seen in Figure 4.4.

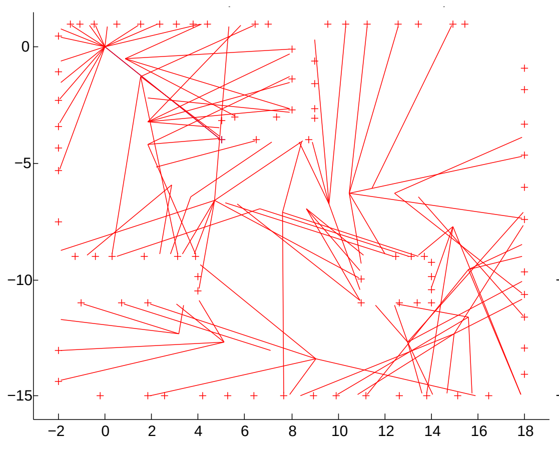
Both algorithms were tested with the Matérn 3 and the squared exponential kernel, each trained through simulated annealing. The runtimes and performance analyses for the methods are shown on Table 4.3 and Figure 4.5. The predictions made by each are shown in figures Figure 4.6 and Figure 4.7.


(a) Ground truth


(b) Rangefinder data: free beams and hits.


(c) Free points in green, occupied in red.


(d) Dataset partitioned into elements.


(e) Beams and hits chosen by active sampling for the squared exponential kernel.

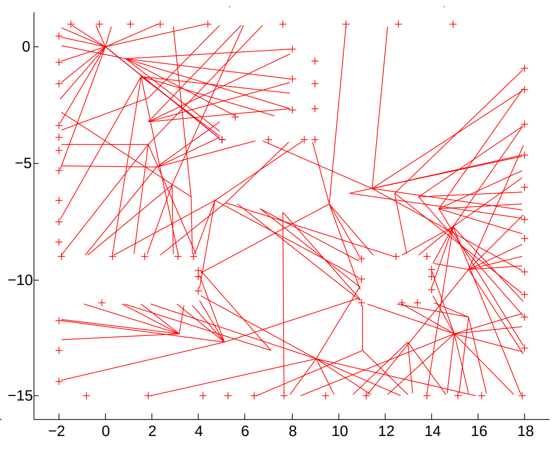
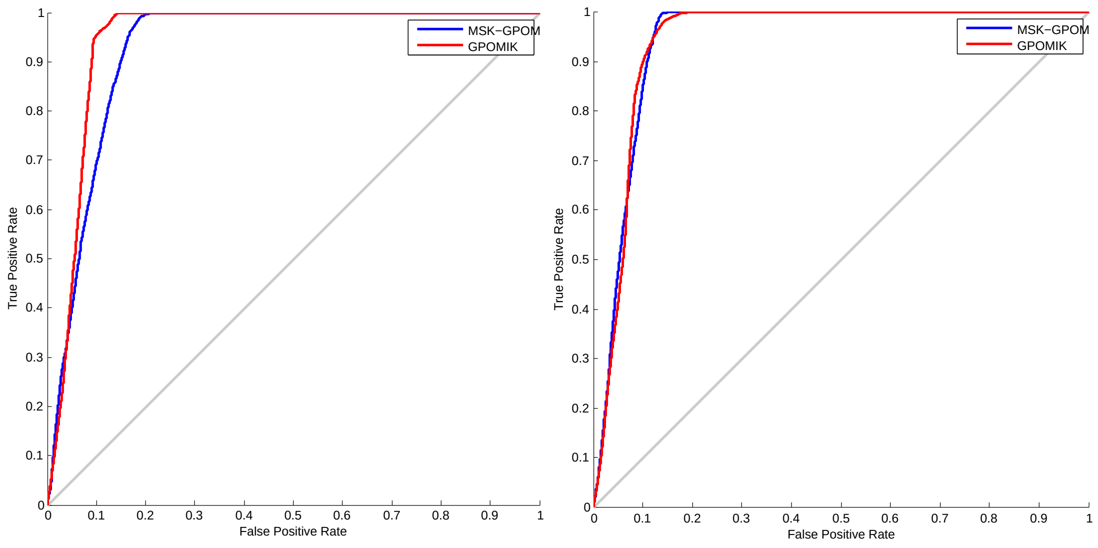

(f) Beams and hits chosen by active sampling for the Matérn 3 kernel.

Figure 4.4: Benchmark dataset and inputs.

Table 4.3: Benchmark times for GPOMIK and MS-GPOM, both trained with simulated annealing.

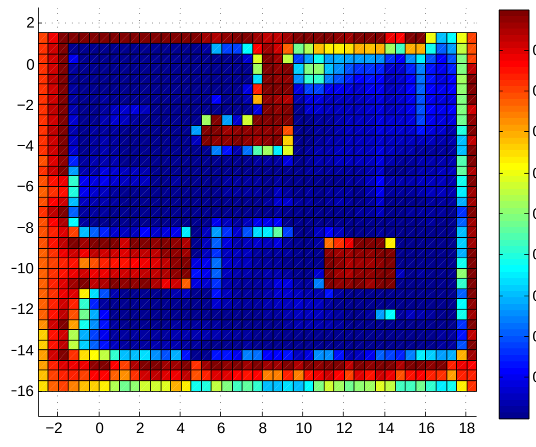
Method	Kernel	Running time	Accuracy	FPR when TPR = 95%
GPOMIK	Matérn 3	79.46s	0.9447	12.25%
	Squared Exponential	86.58s	0.9415	9.76%
MS-GPOM	Matérn 3	23.76s	0.9416	12.34%
	Squared Exponential	19.33s	0.9266	16.30%

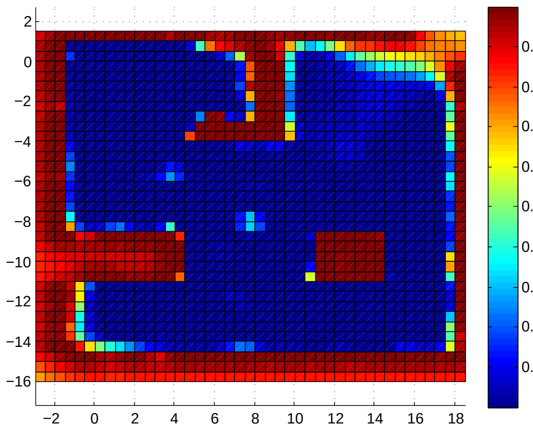


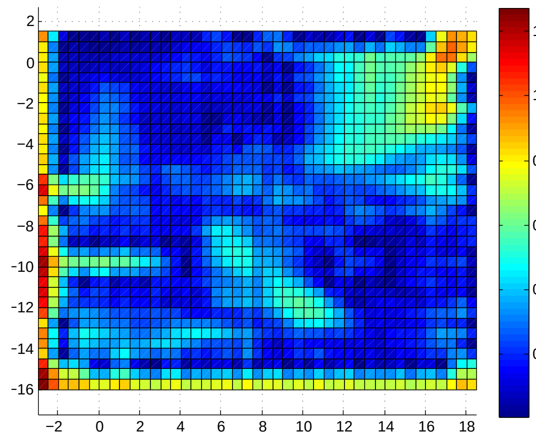
(a) Squared exponential kernel

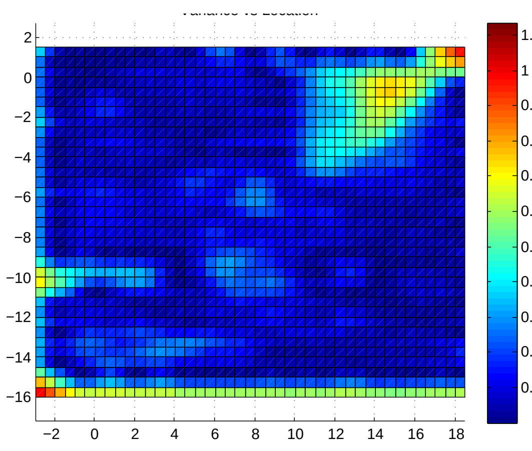
(b) Matérn 3 kernel

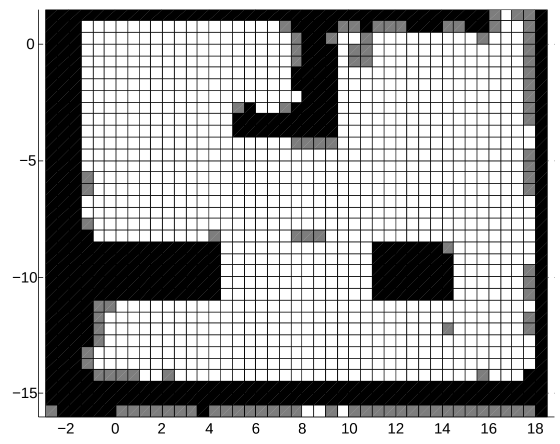
Figure 4.5: ROC curves for the synthetic dataset. The curve for MS-GPOM is represented in blue, GPOMIK in red.


(a) Probability of Occupancy


(b) Probability of Occupancy


(c) Predictive Variance


(d) Predictive Variance


(e) Occupancy Map

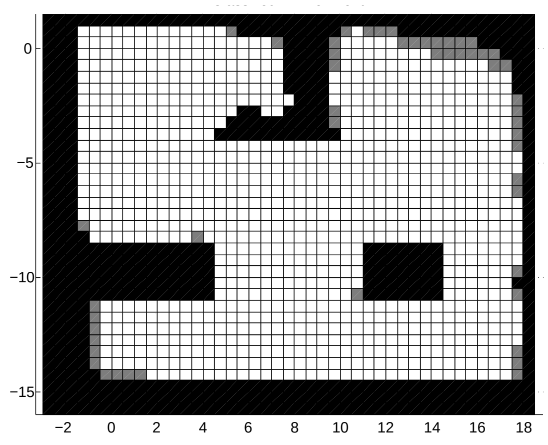
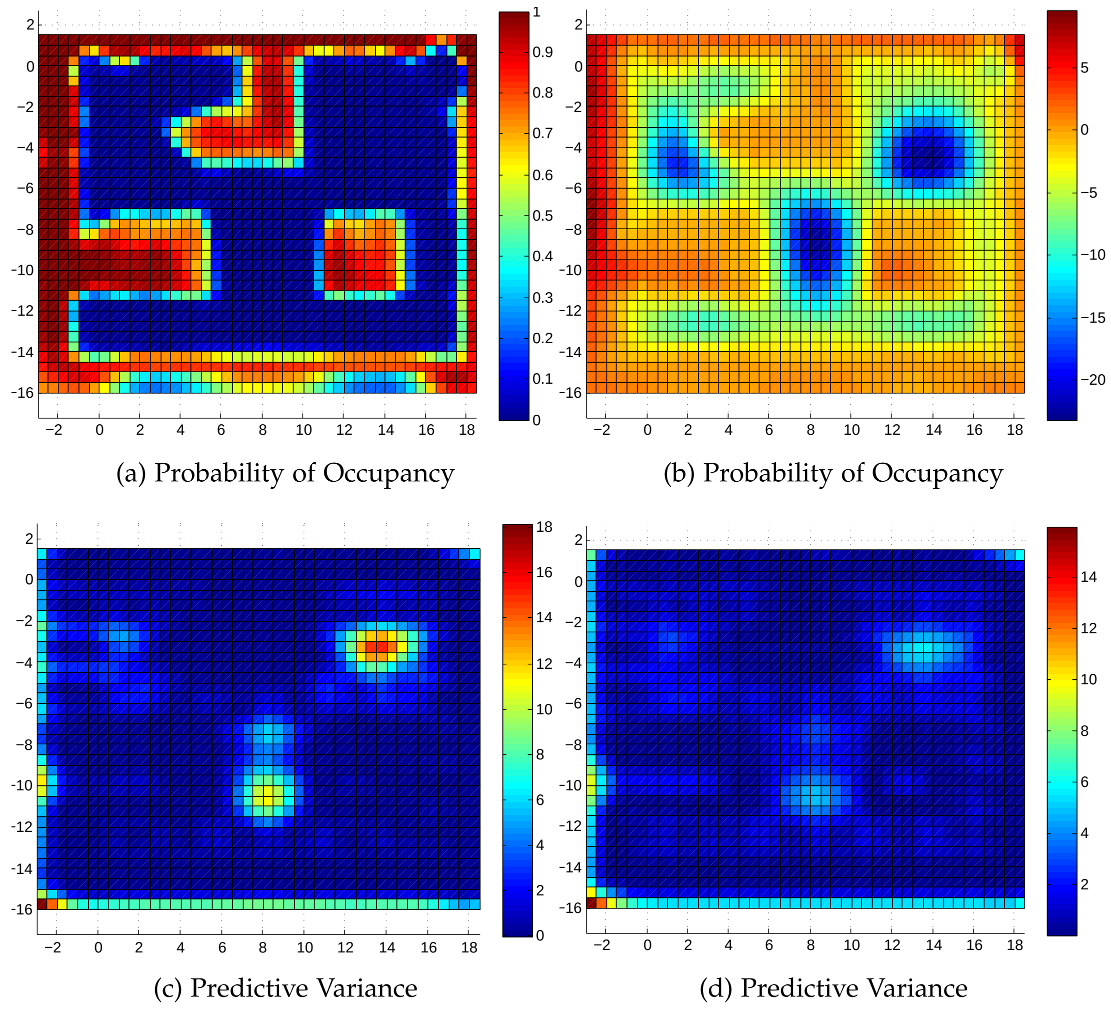

(f) Occupancy Map

Figure 4.6: GPOMIK outputs for synthetic dataset. Left column: squared exponential kernel. Right column: Matérn 3 kernel.


(a) Probability of Occupancy (b) Probability of Occupancy

(c) Predictive Variance (d) Predictive Variance

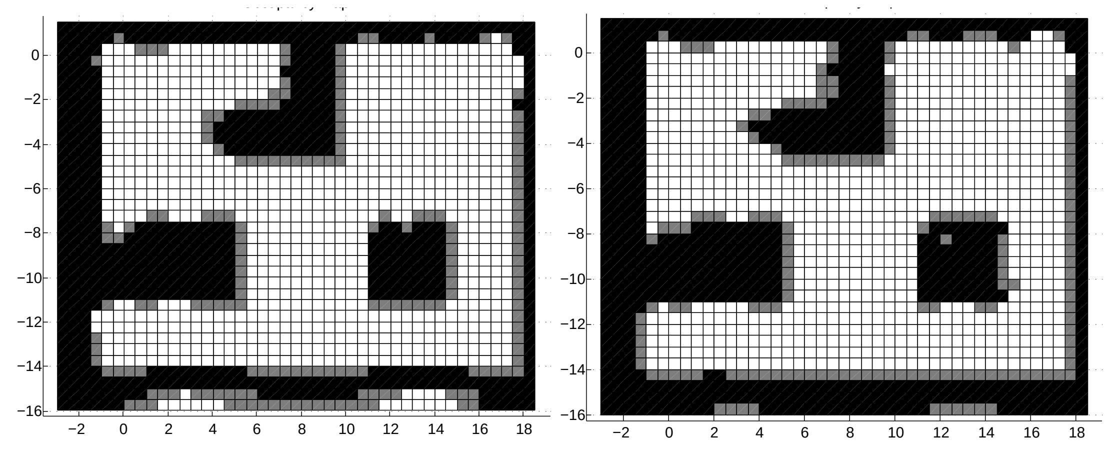

(e) Occupancy Map (f) Occupancy Map

Figure 4.7: MS-GPOM outputs for synthetic dataset. Left column: squared exponential kernel. Right column: Matérn 3 kernel.

4.3.2 REAL DATA

Although the benchmarks in the last section suggest the algorithm does indeed present an advantage over GPOMIK, synthetic data offers a lot of conveniences that in a real setting wouldn't exist, such as lack of noise. To be able to truly assess the usability of the MS-GPOM algorithm, we must test it using datasets taken by real robots.

The raw data were provided by Dirk Haehnel, and contains 395 poses with 361 equally spaced laser readings spanning 180 degrees taken from each. The data were collected at the Belgioioso Castle, located in Milan. It is made available on the Robotics Datasets webpage, maintained by Cyrill Stachniss[⋆]. The data are presented both in raw form and after loop closure; the latter was used in this benchmark.

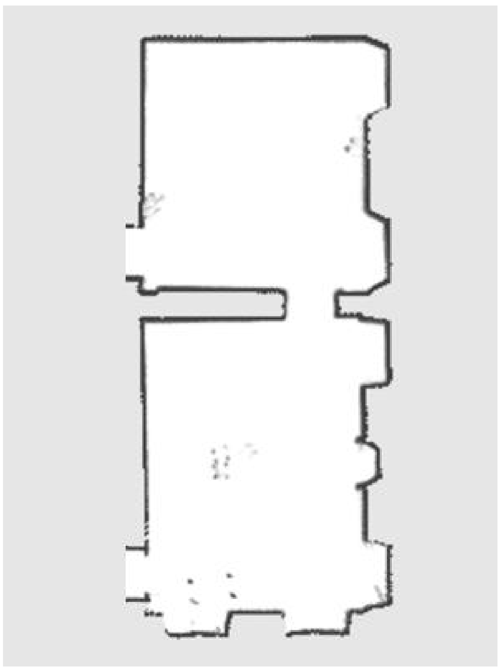
To test the algorithm, a subset of 52 poses was used, from which 37 equally spaced laser beams were taken into consideration. These data refer to two adjacent rooms in the castle, and can be seen in Figure 4.8.

The proposed algorithm was trained using BFGS, with an element size threshold of 1.0 on both the squared exponential and the Matérn 3 kernels. For the contending algorithm, the parameters used were those presented in O'Callaghan and Ramos (2011), presumably reflecting the algorithm's best configuration.

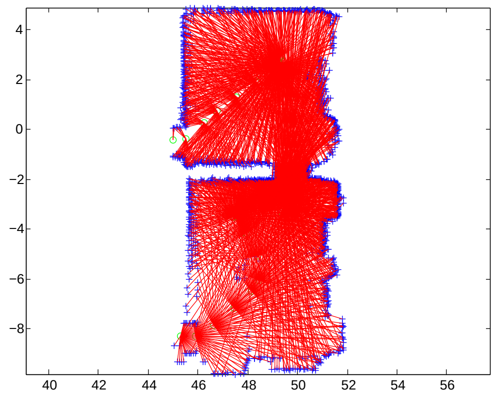
Accuracy was measured by using the trained GPs to predict the occupancy of two thousand points divided evenly between free and occupied, randomly sampled from the beams that weren't used to generate the predictions above. The sample used was the same for all algorithms. Running times and performance results are displayed in Table 4.4 and Figure 4.9. The predictions can be seen in Figure 4.10 and Figure 4.11.

[⋆] http://www2.informatik.uni-freiburg.de/~stachnis/datasets/

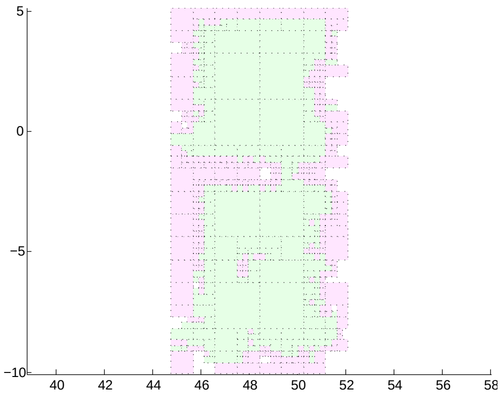
(a) Photograph of a room in Belgioioso Castle, not necessarily where the readings were taken



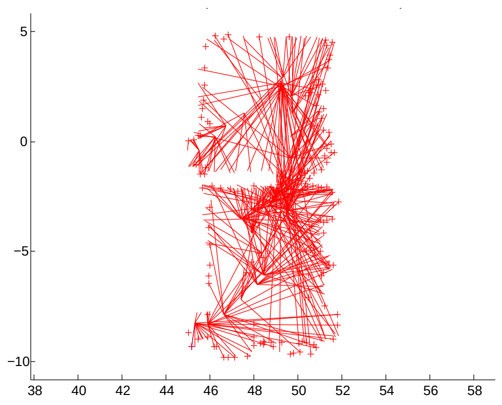
(b) OGM of the rooms where the readings were taken (Stachniss)



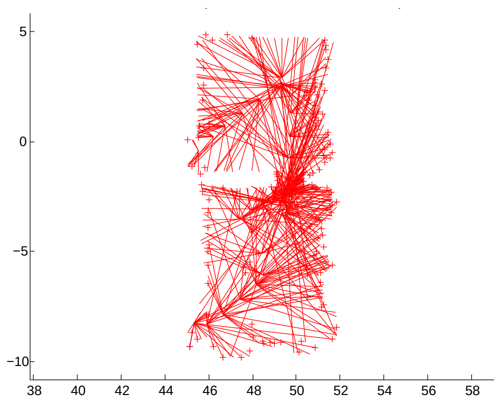
(c) Rangefinder data



(d) Elements, threshold=0.5



(e) Active set (squared exponential kernel)

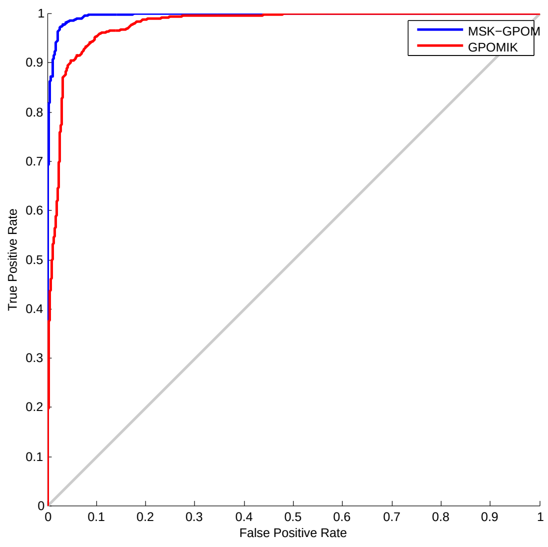


(f) Active set (Matérn 3 kernel)

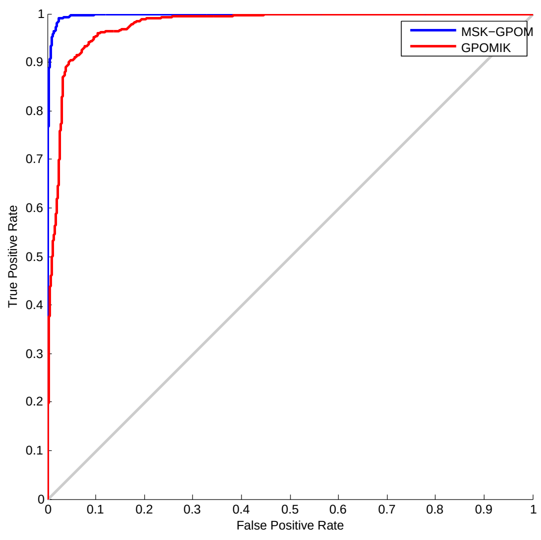
Figure 4.8: Tentative ground truth, rangefinder data and inputs for each method.

Table 4.4: Benchmark times for GPOMIK and MS-GPOM, both trained with annealing.

Method	Kernel	Running time	Accuracy	FPR when TPR = 95%
GPOMIK	Matérn 3	445.42s	0.9751	9.60%
	Squared Exponential	607.93s	0.9665	8.20%
MS-GPOM	Matérn 3	52.07s	0.9918	3.30%
	Squared Exponential	37.29s	0.9947	2.00%

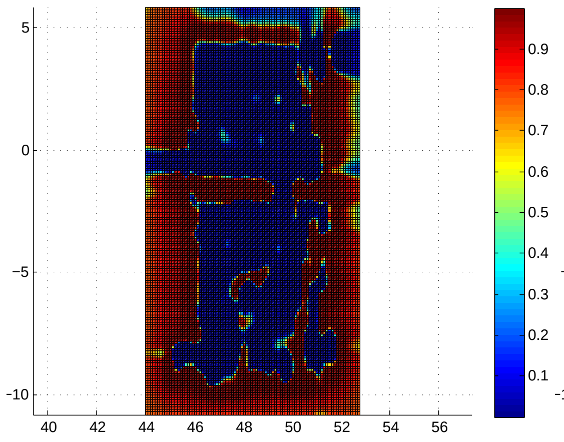


(a) Squared exponential kernel

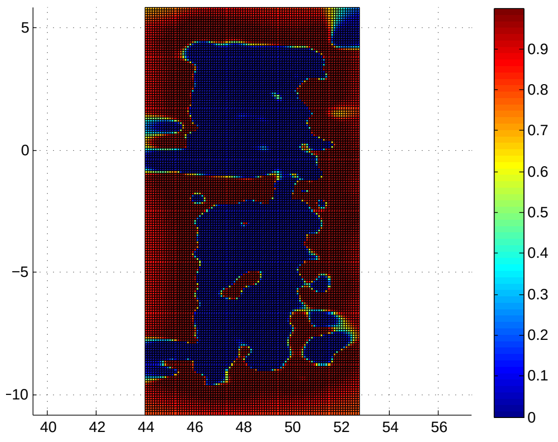


(b) Matérn 3 kernel

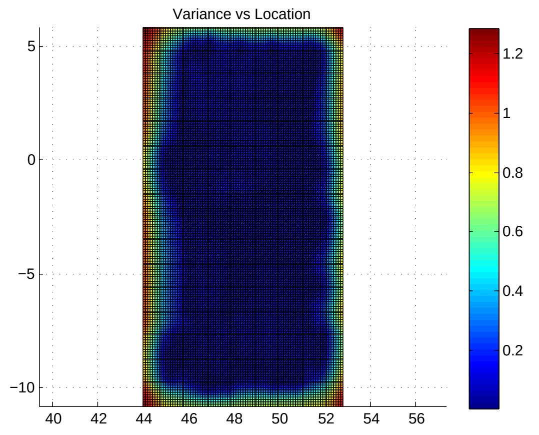
Figure 4.9: ROC curves for the Belgioioso Castle dataset. MS-GPOM in blue, GPOMIK in red.



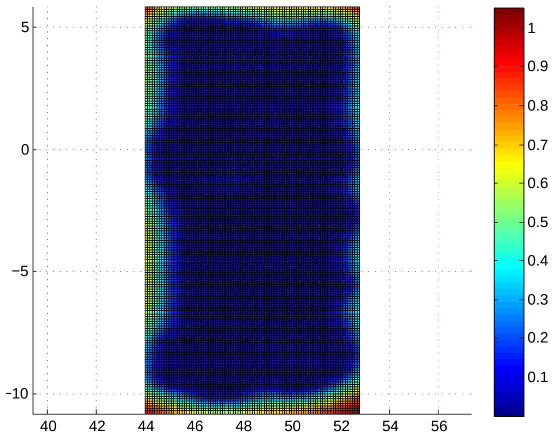
(a) Probability of Occupancy



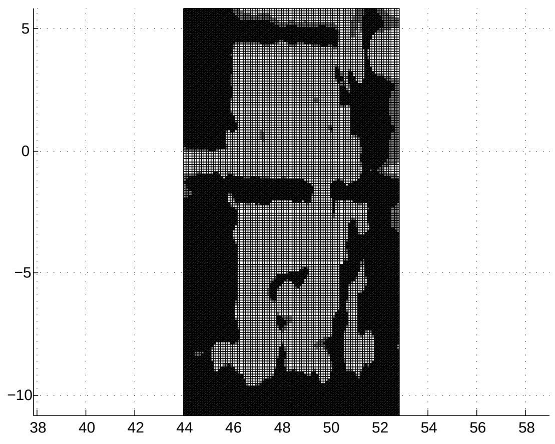
(b) Probability of Occupancy



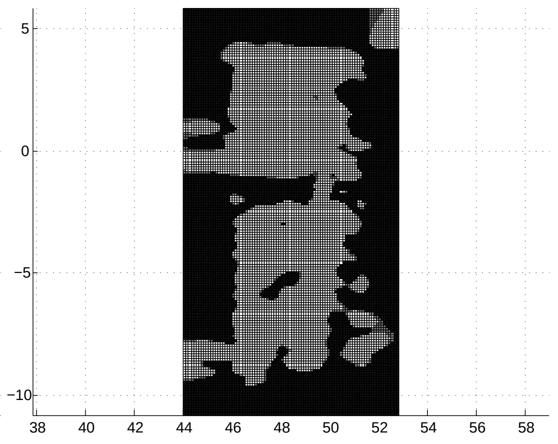
(c) Predictive Variance



(d) Predictive Variance

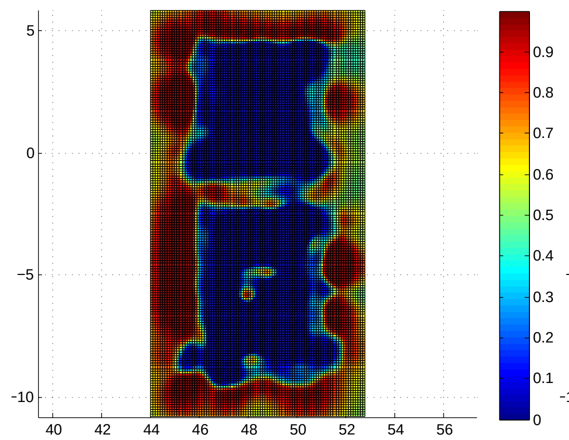


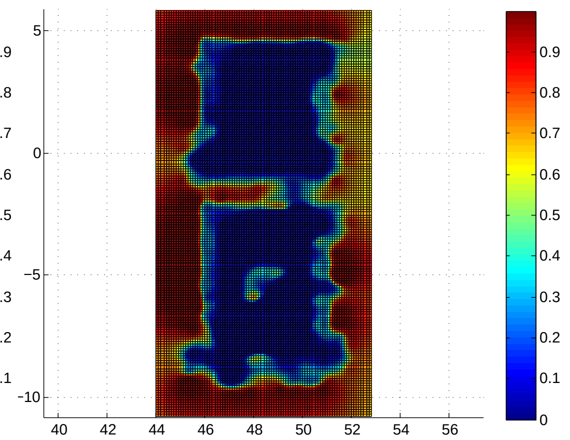
(e) Occupancy Map

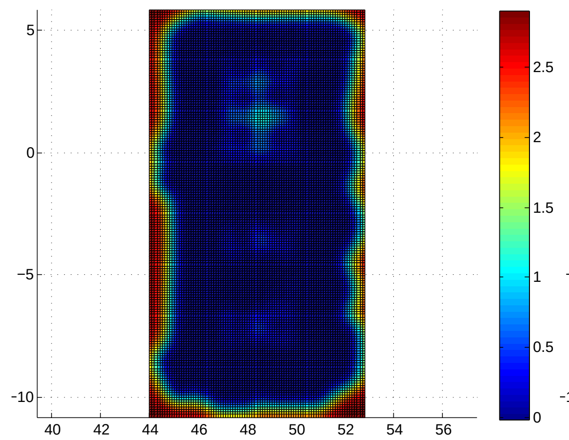


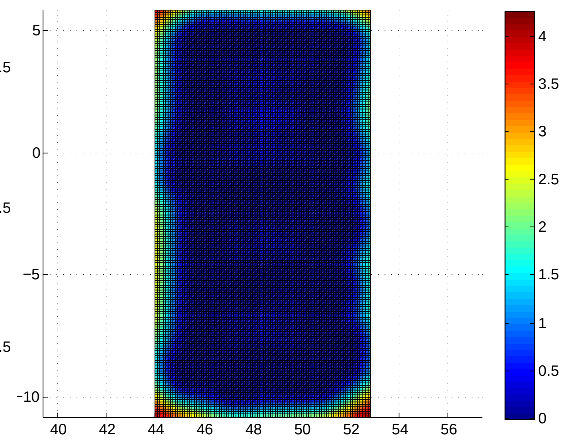
(f) Occupancy Map

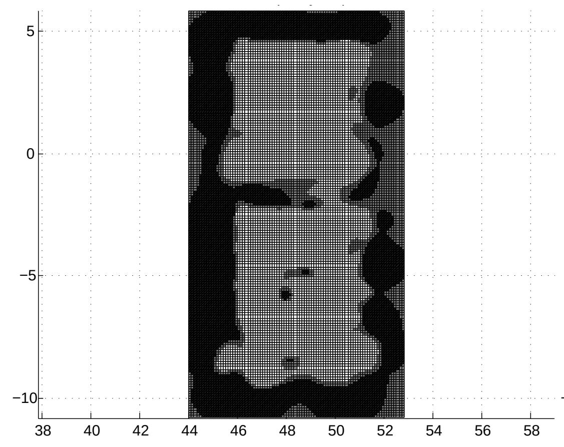
Figure 4.10: GPOMIK outputs for real dataset. Left column: squared exponential kernel. Right column: Matérn 3 kernel.


(a) Probability of Occupancy


(b) Probability of Occupancy


(c) Predictive Variance


(d) Predictive Variance


(e) Occupancy Map

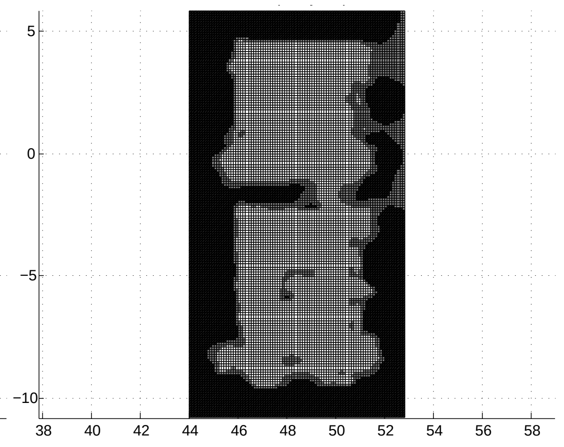

(f) Occupancy Map

Figure 4.11: MS-GPOM outputs for real dataset. Left column: squared exponential kernel. Right column: Matérn 3 kernel.

4.4 SUMMARY

In this chapter, we have analysed the performance of the MS-GPOM in several different conditions. A series of experiments was conduced to analyse how different parameters affect the output of the algorithm. The MS-GPOM was also compared to a previous technique that performs a similar function.

It is safe to conclude that the method developed has performance comparable to the GPOMIK algorithm, both in simulated and real scenarios. It was also shown to be much faster than the latter. This result, however, must be interpreted carefully, as the GPOMIK is tailored to do online predictions, while the MS-GPOM excels at batch mapping, and the experiments reported in this chapter just aimed to provide a proof-of-concept for the viability of the technique presented in this thesis. In the next chapter, we conclude this work by summarising the contributions made and outlining the future work that should be done on this algorithm.

Chapter 5

CONCLUSIONS

This thesis presented a methodology that allows to perform change of support on any kernel, which was used to develop a framework for robotic mapping. With the method developed, continuous occupancy maps can be created in a much shorter time than using other GP-based methods, yet with comparable accuracy.

In this chapter, we review the contributions made in this work and suggest and discuss future research paths.

Now that I've fulfilled my purpose, I don't know what to do.

— Sonny

5.1 CONTRIBUTIONS

In the development of the research described on this thesis, we have investigated the fundamental field of occupancy mapping for robotics. It was observed that most widespread techniques currently in use rely on discretising the region to be mapped, which gives rise to a number of problems.

Continuous occupancy mapping techniques have been proposed to overcome this difficulty in the past. Of those, Gaussian process occupancy maps (GPOMs) have shown potential, due to the ability of Gaussian processes (GPs) to learn from uncertain inputs, unavoidable in the field of robotic sensing. However, this creates a new problem: the complexity of the algorithm scales cubically with the number of inputs, and modern sensors often generate thousands of readings every second.

Our aim was to create a method able to reconcile the robust uncertainty handling of GP inference with the speed necessary to efficiently handle massive datasets. To do so, we investigated change of support kernels, that allow calculation of covariance between points or higher dimensional geometrical elements in the dataset. The multi-support kernel created is easily implementable and allows expansion from traditional kernels. It accepts as input geometrical elements that condense several data points, but each element counts as a single entry, reducing drastically the size of the covariance matrices and allowing GPs to efficiently handle large amounts of spatial data.

We then developed a mapping framework that uses a multi-support Gaussian process (MS-GP) to handle laser rangefinder data and create a continuous occupancy map. The technique was compared to Gaussian process occupancy maps using integral kernels (GPOMIKs), another GPOM technique using change of support, and found to have comparable accuracy both on synthetic and real data, in spite of significantly better scalability properties.

In short, the main contributions of this work are twofold:

A MULTI-SUPPORT KERNEL that is easy to implement and enables traditional covariance functions to accept as inputs not only points, but also hyper-rectangles, allowing reduction of the size of covariance matrices and consequent acceleration of Gaussian process inference and learning.

A CONTINUOUS OCCUPANCY MAPPING TECHNIQUE using a GP with the aforementioned kernel to handle uncertainty, with comparable accuracy in relation to similar state-of-the-art techniques while taking a much smaller toll on speed when handling large datasets.

5.2 FUTURE WORK

There are a number of ways the mapping technique described in this thesis, as well as the kernel developed for it, could be extended or further tested. The following is by no means an exhaustive list, but rather a suggestion for a few immediate research possibilities with which to follow this work.

MULTI-SUPPORT FOR OTHER KERNEL METHODS

Rather than tailored for the task at hand, the multi-support kernel (MSK) developed was designed to be as generic and flexible as possible. As such, any method could potentially benefit from its use. It is worth investigating whether the MSK could bring an advantage to other kernel methods that are required to deal with large amounts of data.

2.5D AND 3D CONTINUOUS MAPPING

Even though all the examples provided on this thesis come from the 2D realm, the method developed is theoretically able to perform 3D occupancy mapping with minor changes. It is possible that 2.5D mapping (e. g. digital elevation mapping) could also be performed after changing the algorithm to predict height rather than occupancy. Investigating its performance in these tasks could bring valuable insight to the field of outdoor mapping.

ONLINE MAPPING

The mapping algorithm presented is tailored for batch mapping, requiring all the data to be acquired beforehand. Online mapping offers the distinct advantage of allowing map generation during the operation of the autonomous agent. It may be worthwhile changing the algorithm to investigate whether the speed advantage demonstrated on this thesis could be extended to online mapping.

conversion between discrete and continuous maps

The method has been shown to perform the trivial conversion from continuous to discrete maps. With small changes, the element generation algorithm could take as input ready-made occupancy grid maps and convert them to continuous maps. This would allow for the conversion of pre-existing maps without requiring new data acquisition.

different data modalities

This work was entirely based on data from laser rangefinders. The algorithm could be tested and adapted to work with other sensor modalities, such as sonars, as well as to fuse different modalities of data into its predictions. This would enhance its versatility and allow use in robotic applications where laser data is not available.

parallelised implementation

The use of graphical processing units is becoming increasingly popular in applications that require fast processing of large amounts of data, due to their massively parallelised architecture being able to deal with matrix operations in a fraction of the time a traditional processor requires. As such, it is possible that reimplementing the algorithm to allow the use of such tools would make for an even faster algorithm.

You can't imagine the wealth of information.
Knowledge... so much... so fast. It's glorious!
— Cortana

BIBLIOGRAPHY

Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*. Applied Mathematics Series. Dover, New York, 1965.

John Bares, Martial Hebert, Takeo Kanade, Eric Krotkov, Tom Mitchell, Reid Simmons, and William Whittaker. Ambler: An autonomous rover for planetary exploration. *IEEE Computer*, 22(6):19–26, June 1989.

Charles George Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6: 76–90, 1970.

Bradley P. Carlin, Hong Xia, Owen Devine, Paige Tolbert, and James Mulholland. Spatio-temporal hierarchical models for analyzing atlanta pediatric asthma ER visit rates. In Gatsonis *et al.*, editor, *Case Studies in Bayesian Statistics*, volume IV, pages 303–320. Springer, New York, 1999.

Vlado Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

Alberto Elfes. *Occupancy grids: a probabilistic framework for robot perception and navigation*. PhD thesis, Carnegie Mellon University, 1989.

Raphael Ari Finkel and Jon Louis Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

Roger Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.

Alan E. Gelfand, Li Zhu, and Bradley P. Carlin. On the change of support problem for spatio-temporal data. *Biostatistics*, 2(1):31–45, 2001.

Marc G. Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2002.

Donald Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

Trevor J. Hastie and Robert J. Tibshirani. *Generalized Additive Models*. Chapman and Hall, 1990.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171–1220, 2008.

Kurt Hornik. Some new results on neural network approximation. *Neural Networks*, 6(9):1069–1072, 1993.

Scott Kirkpatrick, Charles Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

You Li and Yassine Ruichek. Building variable resolution occupancy grid map from stereoscopic system—a quadtree based approach. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 744–749, 2013.

David J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

Hans Moravek. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

Hans Moravek and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 116–121, March 1985.

Krikamol Muandet and Bernhard Schölkopf. One-class support measure machines for group anomaly detection. *arXiv:1303.0309 [stat.ML]*, pages 449–458, 2013.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Lecture Notes in Statistics. Springer, New York, 1996.

Simon T. O'Callaghan. *Continuous Occupancy Maps for the Representation of Unstructured Environments*. PhD thesis, The University of Sydney, 2012.

Simon T. O'Callaghan and Fabio T. Ramos. Continuous occupancy mapping with integral kernels. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1494–1500, 2011.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. The MIT Press, 2006.

Alistair S. Reid. *Gaussian Process Models for Analysis of Remotely Sensed Geo-Spatial Data*. PhD thesis, The University of Sydney, 2011.

Bernhard Schölkopf and Alexander J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning Series. The MIT Press, 2002.

Bernhard Schölkopf, Alexander Smola, and Klaus Robert Müller. Kernel principal component analysis. In *Advances in Kernel Methods - Support Vector Learning*, pages 327–352. MIT Press, 1999.

David F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computing*, 24(111):647–656, 1970.

Cyrill Stachniss. Robotics datasets. URL `http://www2.informatik.uni-freiburg.de/~stachnis/datasets/`.

Michael L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics. Springer, New York, 1999.

Sebastian Thrun. Robotic mapping: A survey. In *Exploring artificial intelligence in the new millennium*, pages 1–35. Morgan Kaufmann Publishers Inc., San Francisco, 2003a.

Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15(2):111–127, 2003b.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

Christopher K. I. Williams. Computation with infinite neural networks. *Neural Computation*, 10(5):1203–1216, 1998.

HAL 9000 is the Heuristically programmed Algorithmical computer, series 9000, a fictional artificial intelligence created by Arthur C. Clarke and Stanley Kubrick for the 1968 movie *2001: A Space Odissey* and homonymous novel. The quotation comes from the movie version of the story.

LIEUTENANT COMMANDER DATA is a fictional artificial intelligence/synthetic life form created by Gene Roddenberry for the television series *Star Trek: The Next Generation*. Their quote is from the 1988 episode *The Child*, written by Gene Roddenberry, Jaron Summers, Jon Povill, and Maurice Hurley and directed by Rob Bowman.

C-3PO is a fictional android created by George Lucas for the 1977 movie *Star Wars*. Their quote is from the second movie of the franchise, the 1980 *Star Wars: The Empire Strikes Back*, written by George Lucas, Leigh Brackett and Lawrence Kasdan and directed by Irvin Kershner.

ANDREW MARTIN is the Bicentennial Man, a fictional android/person created by Isaac Asimov for the novelette *The Bicentennial Man*. Their quote is from the 1999 homonymous movie, written by Nicholas Kazan and directed by Chris Colombus.

MARVIN is the Paranoid Android, a fictional android created by Douglas Adams for the second episode of the 1978 radio series *The Hitchhiker's Guide to the Galaxy*, later turned into a book series. Their quote is from the 2005 homonymous movie, written by Douglas Adams and Karey Kirkpatrick and directed by Garth Jennings.

GLADOS is the Genetic Lifeform and Disk Operating System, a fictional artificial intelligence created by Erik Wolpaw and Kim Swift for the 2006 video game *Portal*. Their quote is from the 2011 sequel, *Portal 2*, written by Erik Wolpaw, Jay Pinkerton and Chet Faliszek and directed by Joshua Weier.

SONNY is a fictional android created for the 2004 movie *I, Robot*, from which the quotation comes. The movie was written by Jeff Vintar and Akiva Goldsman, directed by Alex Proyas and inspired by the homonymous short story collection, written between 1940 and 1950 by Isaac Asimov.

CORTANA is a fictional artificial intelligence created for the 2001 video game *Halo: Combat Evolved*, written by Brannon Boren, Matt Soell and Eric S. Trautmann and directed by Jason Jones.

Whoever removes this thesis or writes their name in the place of my name, may Lovelace, Turing and Knuth, angered and grim, cast them down, erase their name, in the land as in the Internet.

This document was typeset on LaTeX using the typographical look-and-feel `classicthesis` developed by André Miede, freely available online:

`http://code.google.com/p/classicthesis/`

The template was altered in order to follow all the typographical recommendations of the University of Sydney's School of Information Technology.

Typefaces used are Palatino for the body text, AMS Euler for numbers and equations and Bera Mono for pseudo-code and other monospaced text.