Gaussian Process Manifold Learning

By

Larin Cole Adams

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

January 31, 2021

Nashville, Tennessee

Approved:

Richard A. Peters, Ph.D

Eric J. Barth, Ph.D

Arthur F. Witulski, Ph.D

Xenofon D. Koutsoukos, Ph.D

Don M. Wilkes, Ph.D

Table of Contents

List of Figures

Table 1: Symbols for Gaussian Process Manifolds

Symbol	Meaning
M	Manifold consisting of a set of points on the manifold, coordinates corresponding to those points, and a function mapping between the two.
ϕ	Gaussian Process consisting of a covariance function k and associated hyper-parameters α
$\mathbf{Y}$	Matrix of data points. Each column represent one point of data.
$\mathbf{X}$	Matrix of latent variable/manifold coordinate points.
k	Covariance function between two latent variables.
$\vec{k}$	Covariance vector between latent variable and $\mathbf{X}$.
$\mathbf{K_X}$	Covariance matrix where each entry $\mathbf{K}_i j = k(\mathbf{X}_i, \mathbf{X}_j)$.
α	Hyper-parameters for Gaussian Process covariance function
$\gamma(t)$	Geodesic trajectory on a manifold
g_{ij}	Metric tensor
Γ^i_{jk}	Christoffel symbols
R^l_{ijk}, R_{ij}, R	Riemann, Ricci, and scalar curvatures

Chapter 1

Introduction

1.1 Structured Representation of Data

In the field of machine learning, most approaches can be divided into two categories: Black box solutions focused on matching inputs to outputs, and model based approaches seeking to give structure to a set of data. Both have their place for different types of problems, although this paper provides reasons to choose structured learning over black box solutions.

1.1.1 Black Box Learning

Black box learning can be defined as any approach that provides little to no explanation of how inputs are mapped to outputs or processed into answers. A popular contemporary example is the use of neural nets to solve problems in functional approximation, pattern recognition, or classification[42]. While a neural net can be understood in terms of the neuron layers and weights, the most that can be said about how a particular input produces a given output is that the weights have been trained to produce that result.

Patterns and structure are important for extending the usefulness of any machine learning

tool beyond the data it has been trained on [36]. In the case of black box methods, patterns in the data are obscured and learned in a hidden way. One benefit of this approach is that there doesn't need to be any prior assumptions made about models used to represent the data, often giving more flexibility in the types of problems that can be solved.

A natural downside to obscuring the patterns in the data that a black box method has learned is a lack of transparency when it comes to trying to understand how a system works. When a black box machine learning tool fails, it is difficult to quantify exactly what went wrong. Trying to remedy this downside by increasing transparency requires enforcing a known structure on the internals of the black box, which may not even be possible in some cases such as when working with neural nets.

1.1.2 Model Based Learning

Model based learning methods lie on the other end of the spectrum from black box methods. Instead of leaving the choice of structure up to the internals of the machine learning tool, the model is more explicitly specified and the model's parameters are varied systematically, usually through numerical optimization, to best fit the given data[4]. An example of a highly structured method is linear regression, where the model is specified and the learning component tries to find the parameters that best fit a line to the given set of data. While the loss of flexibility means that such a model can only be used when the data actually fits a linear model, it becomes much easier to understand what the model predicts and when or

why it may fail.

Another example of a more flexible model based learning tool is a Support Vector Machine (SVM). In an SVM, data is partitioned into classes based on a separating hyper-plane [43], [39]. This plane is chosen based a set of training data, with the goal of maximizing the orthogonal distance to the points closest to the plane. These closest points determine the hyper-plane, and are called the support vectors. The so called "kernel trick" casts the problem into an infinite dimensional dual space where a separating hyperplane can always be found.[35]. Due to the structure they impose, however, it is much easier to quantify what causes failure. When a new data point is incorrectly classified, it means that the support vectors chosen were inadequate, and the new point can be used to correct this flaw.

There is a balance between structure and flexibility when it comes to machine learning methods. Improving the flexibility by reducing the imposed structure allows for use in more cases and applications to more problems. By reducing the structure, however, the ability to understand the internal mechanisms of the machine learning tool is reduced. There is no one-size-fits-all when it comes to solving problems, and machine learning is no exception. The desired balance point between structure and flexibility must be decided in order to pick an appropriate technique.

1.2 Manifold Learning

Manifold learning is a type of model based learning where the data is assumed to lie on a manifold, a completely separable topological space that is locally homeomorphic to Euclidean space. The goal of the learning component is to determine the structure of this manifold. Often that structure can then be used to solve several types of problems, such as defining a notion of distance on the manifold or deciding if a new point belongs to the learned manifold or not. It is this type of model-based learning that will be discussed further throughout this paper.

1.3 Gaussian Processes and Manifolds

Gaussian processes (GPs) can be used for probabilistic functional regression when the model for a set of data is unknown [31],[45],[32]. A Gaussian process is a probability distribution over functions. Each point on the function is a random variable. Given an input x and prior input and output data X and Y respectively, a Gaussian process ϕ returns a distribution $N(y, \sigma^2)$ where y is the expected value of the Gaussian process at x. The result is computed by using the covariance of the new input x with prior inputs X through a covariance function. The details of Gaussian processes and covariance functions are discussed in greater detail in Section 2.

GPs can be used for dimensionality reduction through a Gaussian process latent variable model(GPLVM) [20]. A set of data Y is assumed to be a function of unknown latent variables

X, and a Gaussian process ϕ is used to model this mapping. The latent variables must be learned by finding X that maximizes the likelihood of the Gaussian process ϕ. This model of a set of data and lower dimensional representation implies a manifold structure. More details on GPLVMs are given in Section 2.

A manifold can be loosely defined as a set of points with a notion of a local neighborhood around each point that also belongs to the set. Points belonging to a manifold can be represented by coordinates, with a function mapping between the coordinates in $\Re^k$ to the manifold points, where k is the dimension of the manifold. It is common for the points of a manifold to exist in a higher dimensional space. When this is the case, the space where the points exist is called the ambient space, and the manifold is said to be embedded in this space. For instance, the surface of a sphere in $\Re^3$ is a 2-dimensional manifold that is embedded in a 3-dimensional ambient space. Points on the sphere can be represented by a pair of coordinates. A more detailed description of the basics of manifolds and differential geometry can be found in [26].

Properties can be defined on a manifold that, while enforcing more structure, provide extra utility in the application of a manifold to problems. One particularly useful property is the notion of distance, a metric which is required when any sort of analytic geometry or calculus is needed. A Riemannian manifold is one that incorporates local distance measurement through the use of a metric tensor at every point on the manifold. Section 5 will go into more detail on Riemannian manifolds and the metric tensor itself.

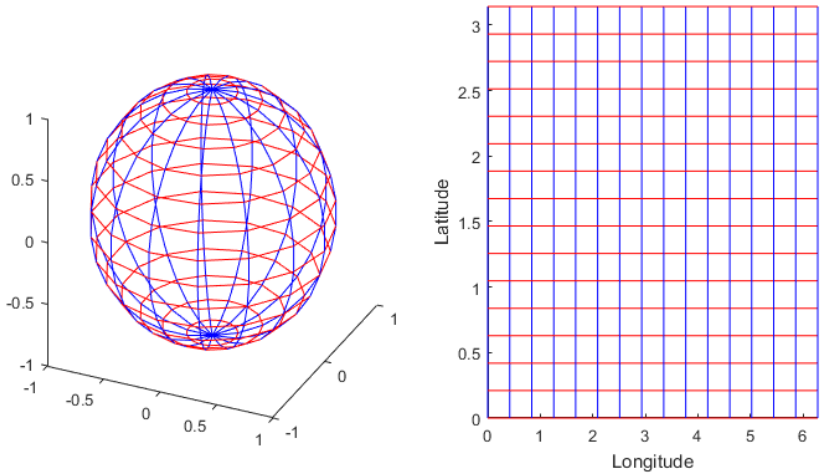

Figure 1.1: Example of a 2-D manifold embedded in a 3-D ambient space(left). The surface of sphere is considered to be 2-dimensional because it can be described by a 2-dimensional coordinate space(right). The mapping between sphere and rectangle has distances along longitudes varying between 0 and 2π and latitudes between 0 and π. Distances on the plane do not equal distances on the sphere.

This dissertation develops and presents a new method for working with learned manifold structures starting from an assumption of a stochastic manifold. A manifold can be said to be stochastic when the points on the manifold are random variables. The geometry and differential properties of a stochastic manifold differ slightly from a Riemannian one. Methods for working with a stochastic manifold are derived and presented in Section 5. Additionally, this approach is applied to practical problems to demonstrate the capabilities that a stochastic manifold can provide.

A method for learning a manifold from a set of data using Gaussian processes is developed and presented in Section 3, and some of the foundations for how to work with Gaussian processes are described. Section 4 discusses the use of Gaussian processes in a Bayes filter as a practical way of using latent variable modeling that simultaneously suggests a manifold interpretation. Section 5 covers in detail the adaptation of GPLVMs to produce a Gaussian

process manifold. Stochastic manifolds require modifications of the mathematical framework used for Riemannian manifolds and those modifications are derived and presented in Section 5. Finally, the applications of Gaussian process manifolds are presented in Section 6.

Chapter 2

Literature Review

2.1 Manifold Applications in Literature

The literature on manifold learning breaks down into two categories: (1)work that depends the geometric and differential properties of known manifolds and uses this structure to solve problems associated with manifold structures and (2)work that learns the manifold structure assumed to be present in a set of data with the goal of providing a simplified representation or regression to predict new data. Research for both of these categories is presented here.

The first category assumes a specific manifold structure or that the manifold is known analytically[1]. These applications adapt existing Euclidean space methods to work in a more general manifold context. When used, Gaussian processes model functions on the manifold and are not used to define the manifold itself.

Mazumdar[28] suggests path planning on a Riemannian manifold from the perspective of optimal control. The paper demonstrates that the trajectories obtained by solving the equations for optimal control are geodesics on the manifold itself. This work assumes that the Gaussian curvature[2] of the manifold evaluates to zero. The work does not include any

[1]Here analytically means that any derived manifold property such as the metric tensor or curvature can be evaluated at any location on the manifold and are infinitely differentiable

[2]Measure of how non-Euclidean the manifold is at a given point

application of these results to any motion planning problems. The manifolds are known analytically and are restricted to only 2 dimensions.

Zeestraten[47] demonstrates a technique for probabilistic imitation learning on Riemannian manifolds. Their approach to imitation learning uses multivariate Gaussians to describe the motion of a system, then uses the distributions to interpolate and extrapolate the behavior. To extend this approach to a Riemannian manifold, Euclidean methods are adapted for use in the tangent space[3] of the manifold. Their work shows how an existing imitation learning method can be applied in the context of a Riemannian manifold. While the methods used are probabilistic, the manifold itself is not. The manifold is deterministic and must be known analytically.

Englert[10] gives a method for planning motions across a sequence of manifolds in the state space of a system. Using sample-based motion planning, a trajectory that crosses several different manifolds can be found to generate a path. If the manifolds represent different behaviors of the system, this represents a way to chain together different learned behaviors to accomplish a goal. The motion planning techniques given in [10], however, are applicable only to manifolds that are known analytically. The techniques are not readily adaptable to learned manifolds where sampling away from the training data becomes impossible.

Mallasto[27] introduces a wrapped Gaussian process, a type of Gaussian process that learns local submanifold structure on a given Riemannian manifold. This allows for the Gaussian process to incorporate the structure of the manifold. The idea of a wrapped Gaussian process latent variable model(WGPLVM) follows from this. The WGPLVM effectively learns the structure of a submanifold, and is demonstrated to perform better than a GPLVM that has

[3]Local Euclidean vector space associated with each point

no knowledge of the larger manifold. The manifold that the WGPLVM is based on is assumed to be known analytically.

Coveney[7] uses Gaussian processes to interpolate values over a 2D triangle mesh manifold. The triangle mesh represents the surface of a 3D object, in this case a human heart. The quantities of interest have a covariance that is related to the distance between them on the surface of the manifold, rather than the Euclidean distance in 3 dimensions. That requires modification of the covariance function for the Gaussian process. This modification accounts for the manifold structure of the triangle mesh. Then the Gaussian process can be used to interpolate values across the surface of this manifold. This is another case where the manifold is known and not learned.

Gao[12] uses Gaussian processes to identify landmark points on a Riemannian manifold to quantify similarities between surfaces based on these landmark points. The work demonstrates superior performance when points are selected by maximum uncertainty under a Gaussian process as compared to manually selected landmark points. The manifold here is a triangle mesh, so again the manifold is not learned but instead provided.

Lin[25] provides an alternative framework for working with Gaussian processes on manifolds by embedding the manifold in such a way that the geometry is close to Euclidean. The work also applies to cases where the manifold is naturally embedded in a Euclidean space. While the work demonstrates that evaluating the covariance function in the ambient space works well, the manifolds they consider are known, not learned.

Samir[33] uses Gaussian processes on a Riemannian manifold to capture the structure of the space of strictly non-decreasing distribution functions. They define a Gaussian process on a known manifold, and use it to assess data known to lie on this manifold. The method

used is applicable only to the particular manifold, and not to learned manifolds.

All of these papers use a known manifold structure while the work presented in this dissertation considers manifolds where the structure is learned from a set of data via Gaussian processes. This allows for use in more general contexts when no prior information is available about the manifold structure of the data.

2.2 Manifold Learning in Literature

Papers in the second category learn a manifold from data. They assume that the data lies on a manifold of lower dimension than the data itself. The geometric and differential structure of the resulting manifold is not discussed and no method for using the manifold beyond simple regression is presented. Gaussian processes are often used to define the mapping to the manifold, but the manifold is always considered to be deterministic and not stochastic.

Barkan[3] discusses a technique for extending manifolds beyond the original training data. Gaussian processes are used to make predictions away from known samples through simple regression. This improves the usability of manifolds by extending the range of inputs that can be covered. The probabilistic nature of the Gaussian process is not used here, as the only concern is the accuracy of the points predicted by the Gaussian Process. GPLVMs are also not compared to other manifold learning methods, which is notable considering that the extension through Gaussian Process regression is intrinsic to GPLVM manifolds.

Thimmisetty[41] uses Gaussian processes to work with manifolds generated by diffusion maps over geological data. The manifold structure is found by using diffusion maps, and then Gaussian processes are used to interpolate and extrapolate this structure to predict

unknowns in the data space. Similar to Barkan[3], this work does not consider GPLVMs as an alternative for generating the manifold, but it does make more use of the statistical properties of Gaussian processes. The predictions made by Gaussian process regression include the variance produced by the Gaussian process, but the manifold is not considered to be stochastic.

Calandrea[5] describes a type of regression called Manifold Gaussian process. Although the name includes the word, no manifold is learned as a part of this approach. Instead, Manifold GPs use a GP-like covariance function for nonlinear regression. The procedure is to learn a feature space from the input data and regress on the features instead of the original data. While this does show some utility working with discontinuous data, it does not contribute much to manifold learning.

Guhaniyogi[13] address the problem of training a GP over large datasets by using compressed Gaussian processes. The approach is appropriate for cases where the data is assumed to lie on a manifold and so some of the information of the samples can be compressed. The primary goal of the work was to improve the speed of using a Gaussian process method. It does not consider the learned manifold in a probabilistic way.

Liang[24] presents a method for placing landmarks on a manifold using Gaussian processes. The goal of placing these landmarks is to find a minimal set of points that capture the structure of the manifold, or in some cases represent categories in datasets such as the NMIST or facial image data. Landmarks should be spread out as much as possible to maximize the information each one provides, and this can be done using Gaussian processes to maximize the variance of each point added with respect to landmarks already placed. This is extended further by allowing a landmark to be selected from the region of the manifold instead of just

choosing a specific sample, as it may be desirable for a landmark to represent a local average. This work does not start with a known manifold structure, but it also does not consider the manifold structure when selecting landmark points.

Yang[46] demonstrates how a manifold can be learned for regression by Bayesian methods using Gaussian processes. It is demonstrated how regression can occur while bypassing the need to learn the manifold completely; however, no real applications are shown and no methods are given to use this regression to work with manifold properties such as distance or trajectory planning.

Jorgenson[17] uses what he names an Iso Gaussian Process. It is a manifold learning approach that produces a distance and topology preserving manifold. This is achieved by first generating a neighborhood graph to ensure the local distances and topology of the data are preserved (similar to how Isomap[2] works), then maximizing the likelihood of this graph with respect to the latent variable representation. The approach is then applied to the Swiss roll and MNIST data sets to demonstrate the degree to which the Iso Gaussian Process preserves topology and local distances, producing manifolds that give an accurate representation of the original data. While the generation of Iso Gaussian Process manifolds is discussed, the use of these manifolds to solve real problems is not covered. No demonstration of how to use the topological structure and distance preservation is given.

These papers demonstrate methods for learning manifold structure from sample data but do not use the learned manifold to solve practical problems beyond simple regression or classification. They make no use of the geometric or differential properties of the manifold. The work presented in this dissertation develops methods for working with the structure of learned manifolds and uses them to solve problems in constrained motion planning.

2.3 Closely Related Research

Some research does attempt to connect the learned manifold to applications while making use of the geometry and structure of the manifold. Havoutis[15] uses the manifold structure for robotic motion, learning constraints in the form of a “skill" manifold. Once the structure is known, the manifold is used to guide new motion–a new performance of this skill–without needing to optimize against the constraints. This creates benefits in terms of the speed at which new trajectories can be computed, making it possible to generate new trajectories in real time. Havoutis demonstrates this with the walk cycle of a bipedal robot. While the benefits of using manifold structure are shown here, only the tangent space of the manifold is actually learned. This means motion can be restricted to the tangent space of the manifold. No manifold coordinates are found, however, meaning there is no benefit from the reduced dimensionality of the manifold. Trajectories are found by following geodesics on the manifold, although the method used simply projects a line onto the manifold. Figure 2.1 shows how this naive implementation can fail to produce the intended results.

Havoutis extends the use of skill manifolds to handle real time disruptions, such as obstacles preventing travel in portions of the manifold, or disturbances that cause the system to leave the manifold entirely [16]. These additions help make the use of skill manifolds viable in practical situations, and given the speed with which they can be computed, makes them useful for path planning in robotic systems–both in physical space and in the robot's configuration space. Since only the tangent space is known, projecting points onto the

manifold in order to recover from disruptions requires an iterative process that finds a point where any motion in the tangent space would push it farther from the point being projected.

Li[23] improves on the work done by Havoutis[16]. This paper looks more specifically at the problem of constrained task manipulation and uses skill manifolds to plan trajectories for motion. Similar to Havoutis, the manifold is learned through the tangent space. This allows for motion to be restricted to the tangent space of the manifold, ensuring that the constraints learned by the skill manifold are satisfied. A Gaussian process is used to project points that have been perturbed away from the manifold to the closest on-manifold point. This results in a speed improvement over the iterative approach used by Havoutis. The skill manifold and the process through which trajectories are found still suffer from the same weaknesses of (1)no coordinate space to simplify computations and (2)a naive implementation of the shortest path.

Havoutis' work and Li's work are much closer than the other papers described above to the work presented in this dissertation, with some important distinctions. With no coordinate space learned for their manifold, all computations must take place in the ambient space of the manifold. All computations must be projected into the tangent space, incurring an additional cost on top of the need to work in a higher dimension than the manifold. The other major difference is the method by which trajectories are found. The approach given by the existing work is to project the shortest path in the ambient space onto the manifold and assumes that this is sufficient to generate a geodesic on the manifold. This assumption may hold on manifolds with low curvature or in cases where the endpoints of the path are close but it is not generally true (see Figure 2.1).

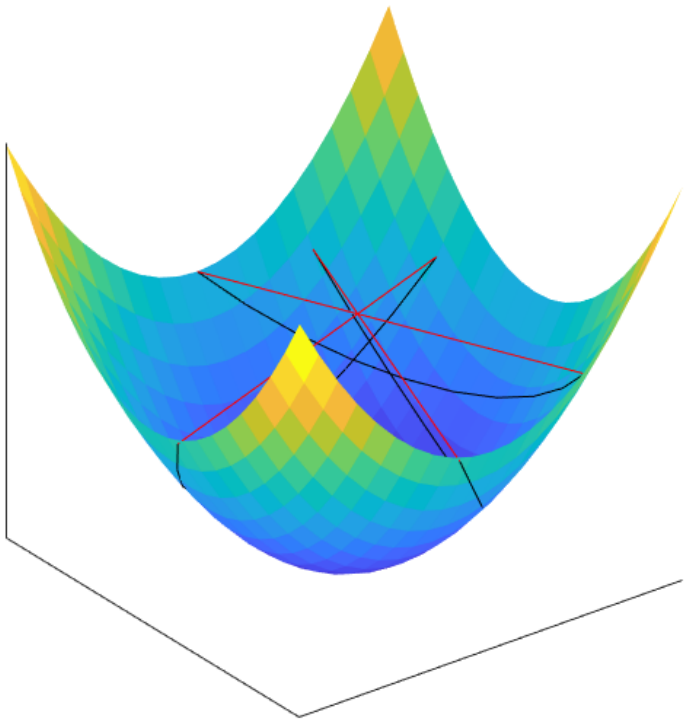

Figure 2.1: Three lines (red) connecting start and endpoints in the ambient space of the manifold and geodesics (black) on a parabolic surface connecting those same points. Note that although the three lines intersect at a single point, the geodesics have no equivalent intersection point. This means that projecting the lines to the manifold (which would produce a single intersection point on the manifold) would not result in geodesics.

2.4 Stochastic Manifolds

In the literature there exists very little work that considers the stochastic nature of a learned manifold while also trying to quantify the geometry and structure of the manifold. As discussed, most break down into one of the two categories mentioned with either an emphasis on the learning portion or focusing on the manifold structure. One recent paper discusses

some of the difficulties involved with using manifold geometry when considering learned manifolds.

Hauberg explores the benefits of manifold learning and the shortcomings that typical manifold learning approaches encounter when trying to use the manifold's differential structure[14]. A good representation of a manifold should allow for interpolation, a well-defined distance, and a measure that permits integration. Hauberg argues that current methods fail to provide these components and gives examples of cases where current methods will produce poor results because of the behavior they exhibit far from the training data. If the distance metric that defines the topology is small far from the data, trajectories will prefer paths away from the well-trained regions. Ideally, the boundary of the training data would serve as a boundary for the manifold. If instead the metric is large far from the data, the interpolation will not be smooth within the manifold. Trying to find a balance between the two extremes for the distance metrics results in problems when considering gaps in the manifold as there is no way to distinguish between structural holes and gaps in the training data.

Hauberg then discusses a probability-based manifold learning approach. With a probabilistic mapping from the coordinate representation to the manifold, it is possible to construct a stochastic metric which incorporates a measure of uncertainty. The points on the manifold are random variables, each point corresponding to a probability distribution as a function of the coordinate inputs. Quantifying this uncertainty in the points on the manifold allows for holes and boundaries within it to be well defined, solving some of the issues that arise in current methods. Hauberg further discusses the differences between a stochastic manifold (i.e. one

with quantified uncertainty) and a deterministic Riemann manifold. Using the expected value of the metric and applying traditional Riemannian methods for integration over the manifold provides an accurate approximation while still retaining the properties of a stochastic manifold.

Hauberg also discusses how to work with path length on stochastic manifolds[9]. In that paper, the need to treat distances on a stochastic manifold differently from regular manifolds is covered in detail. The framework for using the expected value of the curve length or the metric tensor is shown, although derivations and details on how to compute the expected value are not. Again, the use of this information to solve problems on manifolds is not demonstrated.

While Hauberg's work suggests that stochastic manifolds are needed to use the manifold geometry, no work was presented that does so. This dissertation addresses that gap with a novel method that learns manifold structure from a set of data and then uses the resultant geometric and differential properties to solve motion planning problems.

Chapter 3

Gaussian Processes

3.1 Fundamentals

Before the details of a Gaussian Process Manifold (GP-Manifold) can be discussed, the fundamentals of Gaussian Processes need to be explained.

3.1.1 Definitions

A Gaussian Process is defined as a collection of random variables, any finite number of which have a joint Gaussian distribution [32]. In practice, it forms a probability distribution over functions, treating each point on the function as a random variable. To ensure smoothness of the function, the covariance between points is defined as a function of the inputs to the Gaussian Process. (Note that this is a generalization of the standard statistical covariance.) Points which are close in the domain should be highly correlated and thus have a high covariance. With this interpretation, a Gaussian Process for a function $f(x) \in \mathbb{R}^d$ can be defined by a mean function, $m(x) \in \mathbb{R}^d$, and covariance function, $k(x, x') \in \mathbb{R}$, where:

$$m(x) = \mathbb{E}[f(x)] \tag{3.1}$$

$$k(x, x') = \mathbb{E}[(f(x) - m(x))^\top (f(x') - m(x'))] \tag{3.2}$$

These definitions are taken from Rasmussen [32]. In this case, the random variables of the Gaussian Process estimate the value of $f(x)$ at location x. This can be used to fit a curve to a set of data by choosing x as the input, $m(x)$ as the output, and selecting an appropriate functional form for the covariance function. That choice determines the characteristics of the curve being fit. An example is the radial basis function(RBF):

$$k(x, x') = \alpha_1 e^{\frac{-\alpha_2 \|x - x'\|^2}{2}} \tag{3.3}$$

where α_1 and α_2 are hyperparameters (constants that are used to control the behavior of the Gaussian process). α_1 represents the signal-to-noise ratio and α_2 represents the length scale of changes in the Gaussian Processes. Different covariance functions can be chosen to model different expectations of what the data represents. The radial basis function is a common choice because it models the expectation that points close to each other on the input will be close on the output. RBF covariance is what will be used as the standard covariance function for Gaussian processes in this work unless otherwise specified. Figure 3.1 shows a plot of the RBF covariance with respect to the distance between two points x and x'.

A covariance matrix $\mathbf{K}$ is used to store the covariances between a number of points from the same Gaussian process. It is an $N \times N$ symmetric positive definite matrix whose entries

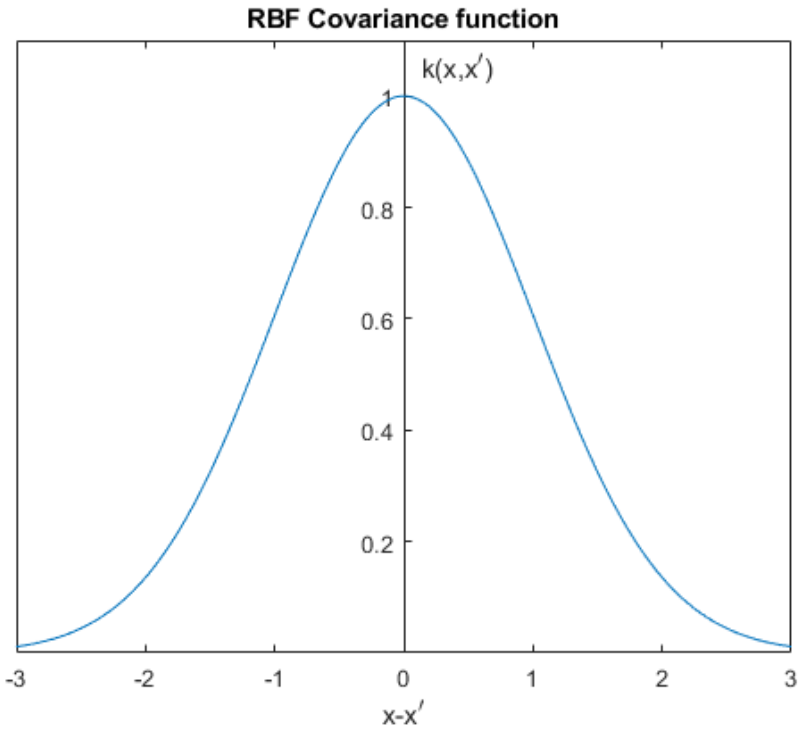

Figure 3.1: RBF Covariance $k(x, x')$ as a function of the distance between two points x and x'. $\alpha_1 = 1$, $\alpha_2 = 1$

are $K_{ij} = k(x_i, x_j)$ for $x_i, x_j \in X$, where X is the N-dimensional domain of inputs to the Gaussian Process.

3.1.2 Regression from Prior Information

Prior information must be incorporated by a Gaussian process in order to perform functional regression. Given a set of N prior inputs X and outputs Y along with the covariance function $k(x, x')$, a new output y' can be predicted for any new input x', using the following formulas:

$$y' = k(x', X)^\top \mathbf{K}^{-1} Y \tag{3.4}$$

$$V[y'] = k(x', x') - k(x', X)^\top \mathbf{K}^{-1} k(x', X) \tag{3.5}$$

Where $k(x', X)$ is the N-vector of covariances between x' and each of the points in X and K is the $N \times N$ covariance matrix with entries given by the covariance function over X. For a more detailed derivation of these equations, see [32]. In the second equation, the vector of covariances is multiplied through the inverse of the covariance matrix to produce a vector of coefficients that weight the components of $k(x', X)$ toward the points closest to x' in X. The result is the variance of the distribution for the point found at x'. In the first equation the function output at each point in X is similarly weighted to produce the mean of the distribution for y'. When considering prior points for a Gaussian process, the uncertainty of each measurement of the prior points $\alpha_3 \mathbf{I}$ is added to $\mathbf{K}$. This has a nice side effect of ensuring that $\mathbf{K}$ is positive definite when the covariance between points is small.

Taking the derivative of y' with respect to x' only requires taking the derivative of the covariance function $k(x', X)$, as neither $\mathbf{K}$ or Y have any dependence on x'. The RBF covariance is infinitely differentiable, so a Gaussian process using RBF covariance will also have this property. The derivatives can be found recursively according to:

$$\frac{dk(x', x)}{dx'} = (-\alpha_2 x')\alpha_1 e^{\frac{-\alpha_2 \|x - x'\|^2}{2}} \tag{3.6}$$

$$= (-\alpha_2 x')k(x', x) \tag{3.7}$$

$$\frac{d^n k(x', x)}{dx'^n} = -(n-1)\alpha_2 \frac{d^{n-2} k(x', x)}{dx'^{n-2}} - \alpha_2 x' \frac{d^{n-1} k(x', x)}{dx'^{n-1}} \tag{3.8}$$

$$\frac{d^{n+1}k(x',x)}{dx'^{n+1}} = \frac{d}{dx'}\left(-(n-1)\alpha_2\frac{d^{n-2}k(x',x)}{dx'^{n-2}} - \alpha_2 x'\frac{d^{n-1}k(x',x)}{dx'^{n-1}}\right) \tag{3.9}$$

$$= -(n-1)\alpha_2\frac{d^{n-1}k(x',x)}{dx'^{n-1}} - \alpha_2\frac{d^{n-1}k(x',x)}{dx'^{n-1}} - \alpha_2 x'\frac{d^{n}k(x',x)}{dx'^{n}} \tag{3.10}$$

$$= -n\alpha_2\frac{d^{n-1}k(x',x)}{dx'^{n-1}} - \alpha_2 x'\frac{d^{n}k(x',x)}{dx'^{n}} \tag{3.11}$$

Figure 3.2 shows an example using a Gaussian Process to fit a curve to the sample points from a hypothetical range-finder. Note that the variance increases with the distance from the sample points. The uncertainty increases and the accuracy falls off as the distance from a sample point increases. The red circles represent the y values of known sample points (x, y), while the blue curve are predicted points (x', y') based on (3.4). Both y and y' are Gaussian random variables, where the variance is shown by the gray region around the curve.

The hyperparameters of the covariance function can be used to tune the Gaussian Process to better fit the data. Figure 3.3 taken from [32] illustrates the importance of having the correct values for the hyper parameters. In these plots the length scale, or α_2 parameter from the radial basis function specified earlier, is varied while the remaining hyperparameters are optimized to maximize the marginal likelihood of the data (i.e., the probability of the observed data given the hyperparameters). In practice, all the hyperparameters are estimated through a method such as scaled conjugate gradient optimization to best fit the input data.

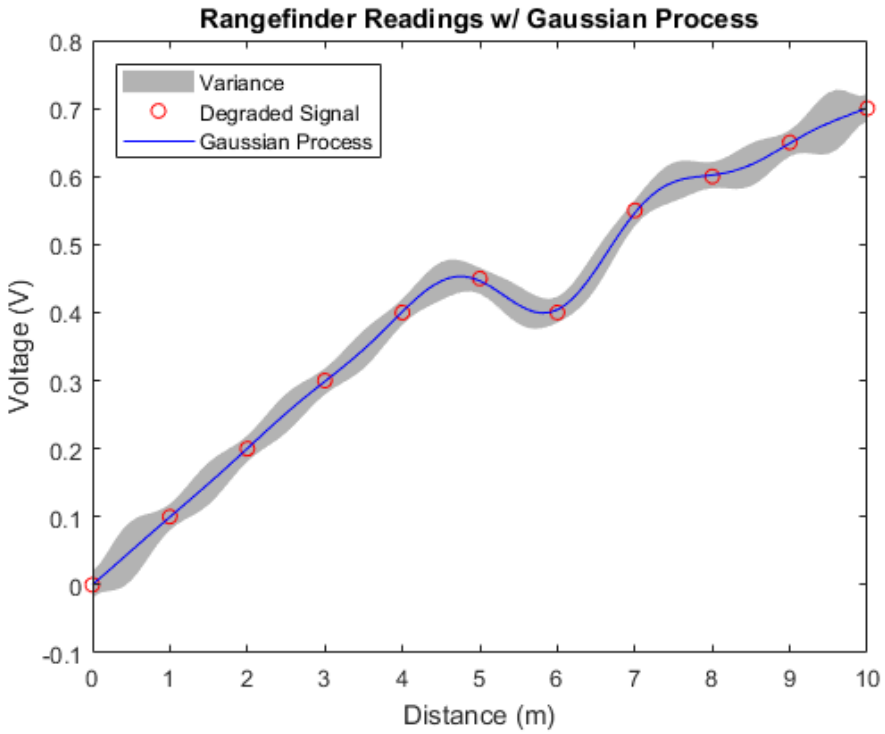

Figure 3.2: Gaussian Process curve fit to sample points from a hypothetical rangefinder. Sample points are given by the red circles, with the mean function of the Gaussian Process in blue and the variance envelope in gray.

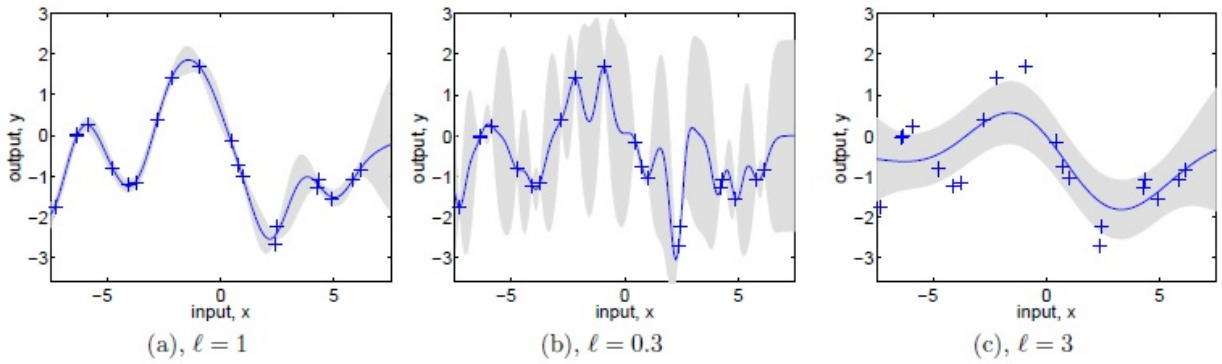

Figure 3.3: Three Gaussian processes for the same data set. Crosses represent the prior data and the smooth line shows the corresponding Gaussian process, with the gray region showing the 95% confidence interval. (a) has the optimal set of hyperparameters. (b) has the length scale set too small while (c) has the length scale set too high [32].

3.2 GP-LVM and Dimensionality Reduction

Gaussian Processes can also be used for dimensionality reduction. In many cases when working with high dimensional data, it can be desirable to simplify the problem by reducing

the dimension to a more manageable lower dimension, often called the latent space. This is often the case when the intrinsic dimensionality of the problem is likely to be less than the dimension of the data. In such a case there may be redundancies in the data across its dimensions. An example of this might be a warped plane in three dimensions. If the exact nature of the warping is known, any analysis or problems can be solved more easily in two dimensions and then warped back into three dimensions. A number of techniques for dimensionality reduction exist for different types of problems.

3.2.1 Existing Dimensionality Reduction Techniques

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that finds the dimensions of highest variance within a multidimensional data set. PCA does this by decomposing the data into an orthogonal basis ordered by the magnitude of variance along each axis [37]. This works well for problems where the data is linear, but is less effective when data is non-linear.

Isomap [40] is another dimensionality reduction technique that can accurately model some non-linear data. It works by computing distances between points and forming a graph of connected points based on k-nearest neighbors or within a certain fixed radius. The resulting distances are used in a multi-dimensional scaling algorithm [29] to provide dimensionality reduction. In effect Isomap finds the geodesics on the high dimensional set. When using time series data from a dynamical system, subsequent points can be assumed to be connected,

providing some additional information. However, Isomaps are sensitive to noise, especially in initial conditions [2]. Given slightly different initial conditions highly similar processes may lead to significantly different manifolds (in shape and extent) and to latent mappings that exhibit little similarity. This limits the applicability of Isomap to nondeterministic physical systems.

Kernel PCA is a non-linear approach similar to PCA where the data is first mapped to an arbitrary higher dimension where it can be easier to separate points [34]. PCA is performed in this higher dimensional space to give the final non-linear low dimensional representation. This is accomplished using a kernel method similar to that of a support vector machine that works without actually mapping to a higher dimensional space first, instead defining a kernel function that represents an inner product in the higher space. While Kernel PCA is capable of handling non-linear structure within a data set and performs well for data clustering, it is not particularly applicable to the regressions necessary to approximate dynamical systems.

3.2.2 GP-LVM Optimization

Gaussian Processes can be used for dimensionality reduction as well, by treating the change in dimension as a mapping from the low dimensional latent space X to the high dimensional space Y. The mapping is chosen from low to high so that it is possible to have an injective (one-to-one) mapping between the two spaces. If the mapping was performed from high to low then it would not necessarily be possible to ensure this constraint. The low dimensional

space X is then taken as an input to a Gaussian Process with output Y, and the marginal likelihood of a given X can be found from the following equation taken from Lawrence [20]:

$$p(Y|X,\alpha) = \frac{1}{(2\pi)^{\frac{DN}{2}}|K|^{\frac{D}{2}}} e^{\frac{-\operatorname{trace}(K^{-1}YY^{\top})}{2}} \tag{3.12}$$

Here D is the dimension of the data set Y, and N is the number of data points. The covariance matrix K is constructed over the latent variables using the covariance function, $k(x, x')$. Maximizing this likelihood with respect to X gives a latent space that best reproduces the data set Y. The hyperparameters are also included in the maximization to ensure that the mapping between the two spaces is as accurate as possible. The resulting Gaussian Process Latent Variable Model (GP-LVM) is a non-linear probabilistic dimensionality reduction method that can be used in a wide range of applications [20].

The optimization itself is typically performed using a scaled conjugate gradient method that is initialized using some other approach such as Principle Component Analysis or graph diffusion. The negative log likelihood is taken and minimized to simplify computations:

$$\begin{aligned} L(X) &= -\ln\left(\frac{1}{(2\pi)^{\frac{DN}{2}}|\mathbf{K}|^{\frac{D}{2}}} e^{\frac{-\operatorname{trace}(\mathbf{K}^{-1}YY^{\top})}{2}}\right) && (3.13)\\ &= -\ln\left(\frac{1}{(2\pi)^{\frac{DN}{2}}|\mathbf{K}|^{\frac{D}{2}}}\right) - \frac{-\operatorname{trace}(\mathbf{K}^{-1}YY^{\top})}{2}\Big) && (3.14)\\ &= \ln((2\pi)^{\frac{DN}{2}}) + \frac{D}{2}\ln(|\mathbf{K}|) + \frac{1}{2}\operatorname{trace}(\mathbf{K}^{-1}YY^{\top})) && (3.15) \end{aligned}$$

For optimization, the gradient of this likelihood must be taken with respect to the latent variables X and the hyperparameters α. This gradient is easier to break into two parts, the first being the gradient of L with respect to $\mathbf{K}$:

$$\frac{dL}{d\mathbf{K}} = \frac{D}{2}\mathbf{K}^{-1} - \frac{1}{2}\mathbf{K}^{-1}YY^{\top}\mathbf{K}^{-1} \tag{3.16}$$

The final gradient with respect to latent variables and hyperparameters can then be found through the chain rule:

$$\left[\frac{\partial \mathbf{K}}{\partial X_{ij}}\right]_{mn} = \begin{cases} \alpha_2 \left(x_{nj} - x_{ij}\right) k\left(\mathbf{x}_i, \mathbf{x}_n\right), & \text{for} \quad m = i, \\ \alpha_2 \left(x_{mj} - x_{ij}\right) k\left(\mathbf{x}_m, \mathbf{x}_i\right), & \text{for} \quad n = i, \\ 0, \text{ otherwise} \end{cases} \tag{3.17}$$

$$\frac{\partial \mathbf{K}}{\partial \alpha_1} = \frac{\mathbf{K}}{\alpha_1} \tag{3.18}$$

$$\frac{\partial \mathbf{K}}{\partial \alpha_2} = -\tfrac{1}{2}\operatorname{dist}^2(X) \odot \mathbf{K} \tag{3.19}$$

$$\tag{3.20}$$

Details on these derivation can be found in the Appendix.

3.3 External Information

In any machine learning method, it is useful to be able to incorporate extra information about the data used for training. This extra information can take many forms; in some cases the data may be a time series, leading to information about adjacent states. The

relative closeness of points may also be known, and the method used for learning can use this to better preserve the structure of the data. In some cases, extra information about the data is available (in the form of time-progression or known distance graphs). This can be added to the GP-LVM optimization process to improve accuracy through extra terms on the likelihood function. The following are examples of how external information can be added.

3.3.1 Wang and Low Dimensional Character Representation

Wang used Gaussian Processes to model motion capture data with the intent of producing a lower dimensional representation of the high dimensional animation [44]. He adds an additional Gaussian Process to model the sequential time-series nature of the data, which takes a latent space point as input and as output produces the next point in the time-series. The resulting latent space trajectories corresponding to the input animation sequences are much smoother when using this additional information.

3.3.2 Levine and Latent Space Control

Levine adds a “connectivity" term to the likelihood to allow for control over which states should be close to each other in order to preserve in the latent space the closeness of points that were close in the data space [22]. The connectivity term is a graph structure linking points together with a notion of distance. The latent space that results from using this term allows for additional control over the distance between latent points. Levine uses this to ensure the endpoints of motion capture data are close to each other to allow for transitions

from one animation sequence to another. This is combined with a control system that operates in the latent space to fulfill goals in the state space.

3.3.3 Ko and Latent Space Labels

Ko uses information about the desired latent space to guide the optimization process to match some points to desired coordinates in latent space [19]. This allows for more direct control over the final latent coordinates produced for a given data set, which can be useful for giving some meaning to the latent space. The method used allows for weighted “labels" that can be modified to change the degree to which the optimization tries to adhere to the labels. While there is some utility with this addition, adding labels can result in less optimal mappings due to the extra constraints on the optimization process.

Chapter 4

GP-Bayes Filter

Latent variable modeling is a useful dimensionality reduction tool that helps simplify a complex system, but often times the latent variables that form the basis for the model are not directly measurable. This is especially common when the latent variables themselves are generated from machine learning, but can be true even when they have physical meaning. An example of this might be a robot containing a variety of rangefinders or other telemetry sensors where the latent variables in question are the components of the position of the robot. Although the sensors produce information related to the position, such as distance to a landmark, the position is not directly measurable. To solve this type of problem, Bayes filters in the form of Kalman or particle filters are used to track the quantities that cannot be measured [6].

4.1 GP-Bayes Filter

A typical Bayes filter consists of two main components. The first is a model of the system and how it evolves through time. This is used to predict the next state of the system based on the current state as well as how the uncertainty propagates. The second component uses measurements from whatever sensors are available to update the quantities of the system being tracked by the filter. Since the desired quantities cannot be measured directly, the

measurement component instead predicts what the sensors are most likely to read based on the current state of the system and compares the prediction to what was actually observed. The difference between these two is then used to update the state accordingly. In a typical Bayes filter, these two components are called update and measurement respectively. At each time-step, the system is updated and then measured to accurately track the desired quantities.

4.1.1 Basic GP-Bayes Filter

The difficulty in producing a well performing Bayes filter generally lies in the difficulty of finding a good model for the update and measurement steps. Producing a good model often requires a deep understanding of how the system being tracked works. A GP-Bayes filter as presented originally by Ko seeks to apply Gaussian Process regression to the update and measurement models [18]. Two separate Gaussian Processes are used, one for the update step and one for the measurement. Both are generated from a set of training data consisting of known state and sensor measurements, using hyperparameter optimization to ensure a good fit.

Consider a system with unobservable state x and observable outputs z. Then a GP-Bayes filter tracking this system can be described by the following equations:

$$\vec{x}_i = \vec{k}_{i-1}^T \mathbf{K}_X \mathbf{X}_{2:N} \tag{4.1}$$

$$\mathbf{\Sigma}_i^x = (k(\vec{x}_{i-1}, \vec{x}_{i-1}) - \vec{k}_{i-1}^T \mathbf{K}_X^{-1} \vec{k}_{i-1})\mathbf{I} \tag{4.2}$$

$$\vec{z}_i = \vec{k}_i^T \mathbf{K}_Y \mathbf{Y} \tag{4.3}$$

$$\mathbf{\Sigma}_i^z = (k(\vec{x}_i, \vec{x}_i) - \vec{k}_i^T \mathbf{K}_Y^{-1} \vec{k}_i)\mathbf{I} \tag{4.4}$$

where $\mathbf{K}$ is the covariance matrix with entries from the covariance function k. The vector $\vec{k}$ is the same as $k(x, \mathbf{X})$ from equation3.4, produced by evaluating the covariance function for a given x against every column from $\mathbf{X}$. $\mathbf{X}$ represents the latent space values and $\mathbf{Y}$ represents the observable outputs. Since the filter predicts state i from state $i-1$, $\mathbf{X}$ is offset so that an input of x_{i-1} has an output of x_i, which is why $\mathbf{X}$ is indexed from 2 to N. This information can be used in a Kalman filter according to the following series of equations:

$$\hat{x}_i = \vec{k}_{i-1}^T \mathbf{K}_X \mathbf{X}_{2:N} \tag{4.5}$$

$$\mathbf{Q}_i = (k(\vec{x}_{i-1}, \vec{x}_{i-1}) - \vec{k}_{i-1}^T \mathbf{K}_X^{-1} \vec{k}_{i-1})\mathbf{I} \tag{4.6}$$

$$\mathbf{G}_i = \frac{\partial(\hat{x}_i)}{\partial \vec{x}_{i-1}} \tag{4.7}$$

$$\hat{\mathbf{\Sigma}}_i = \mathbf{G}_i \hat{\mathbf{\Sigma}}_{i-1} \mathbf{G}_i^T + \mathbf{Q}_i \tag{4.8}$$

$$\tag{4.9}$$

This set of equations is the "update" step. Here $\hat{x}$ is the prediction for the next latent state, $\mathbf{Q}$ is the Gaussian process' variance matrix, $\mathbf{G}$ is the Jacobian of the Gaussian process for this prediction, and $\hat{\mathbf{\Sigma}}$ is the covariance matrix for the prediction $\hat{x}$.

$$\vec{z}_i = \vec{k}_i^T \mathbf{K}_Y \mathbf{Y} \tag{4.10}$$

$$\mathbf{\Sigma}_i^z = (k(\vec{x}_i, \vec{x}_i) - \vec{k}_i^T \mathbf{K}_Y^{-1} \vec{k}_i)\mathbf{I} \tag{4.11}$$

$$\mathbf{H}_i = \frac{\partial(\vec{z}_i)}{\partial \vec{x}_i} \tag{4.12}$$

$$\mathbf{K}_{gain,i} = \hat{\mathbf{\Sigma}}_i \mathbf{H}_i^T (\mathbf{H}_i \hat{\mathbf{\Sigma}}_i \mathbf{H}_i^T + \mathbf{\Sigma}_i^z)^{-1} \tag{4.13}$$

$$\vec{x}_i = \hat{x}_i + \mathbf{K}_i(\hat{y}_i - \vec{z}_i) \tag{4.14}$$

$$\mathbf{\Sigma}_i = (\mathbf{I} - \mathbf{K}_i \mathbf{H}_i)\hat{\mathbf{\Sigma}}_i \tag{4.15}$$

This set of equations is the "measurement" step. Here $\vec{z}$ is the predicted measurement and $\mathbf{\Sigma}_i^z$ is the associated covariance matrix. $\mathbf{H}$ is the Jacobian of the Gaussian Process for this prediction and $\hat{y}$ is the current measured data value. $\mathbf{K}_{gain,i}$ is the Kalman gain, an optimal term that minimizes the error between the prediction $\vec{z}$ and measurement $\hat{y}$. The Jacobians for $\frac{\partial(\hat{x}_i)}{\partial \vec{x}_{i-1}}$ and $\frac{\partial(\vec{z}_i)}{\partial \vec{x}_i}$ are relatively easy to derive and are shown in Ko[19].

4.1.2 GP-LVM Addition

The GP-Bayes filter requires some amount of training data to use, and more importantly requires knowledge of the system state that is being tracked. In some cases, the system state may not be known directly or there may not be a ground truth system state available at all. This could happen when the state of a system is constrained in such a way that a smaller hidden variable set determines the system's behavior. The GP-Bayes filter can be extended to use a GP-LVM latent space as the system state to be tracked instead. In the GP-LVM case, X is the derived latent variables while in the normal GP-Bayes filter X is a set of known system state variables. The latent space then becomes a virtual system state space. While

this does not change the GP-Bayes filter itself significantly, it does allow for easier use of the latent space as it provides a way to estimate the current latent coordinates of a system.

4.2 Ko Implementation

Ko & Fox presented both the GP-Bayes filter and the extension to GP-LVMs in their papers [18],[19]. While the GP-Bayes filter itself worked well, the GP-LVM extension allowed for robust training of a robotic manipulator to mimic a training set. This was due to the use of the GP uncertainty estimates to pick system inputs that lowered the uncertainty in subsequent steps. The control system was then trained exclusively on the control inputs that reduced the Gaussian Process variance, i.e. only control inputs that resulted in a state with higher certainty were used. Essentially, this used the certainty of the latent state as a proxy for stability and ensured that the system stayed in stable regions rather than diverging to unlearned portions of the latent space. This highlights one of the biggest advantages to using Gaussian Processes–a measurement of the accuracy of a prediction is a very desirable trait for a machine learning approach.

4.2.1 Modifications to Ko

While the GP-Bayes filter presented in Ko works well, for the research presented in this dissertation a change was made in the implementation of the filter's measurement. In Ko's original equation, (4.11) above, the uncertainty in the measurement is only a function of the Gaussian Process used to find the predicted measurement based on the latent space

coordinates. In practice, sensors have their own uncertainty in the results they report. A sensor measurement can be considered as a random variable, and appropriate considerations need to be made for the distribution of this random variable. A modification to equation (4.11) above accounts for this:

$$\mathbf{\Sigma}_i^z = (k(\vec{x}_i, \vec{x}_i) - \vec{k}_i^T \mathbf{K}_Y^{-1} \vec{k}_i)\mathbf{I} + \mathbf{R} \tag{4.16}$$

Here $\mathbf{R}$ represents the covariance matrix for the sensor inputs that produce $\hat{y}_i$ in equation (4.14). This covariance matrix can be found by appropriate testing of the sensors in a controlled environment.

4.3 Manifold Interpretation of GP-LVM

The idea of using latent variable modeling as a low dimensional representation of a data set is fundamentally trying to describe the data set as a low dimensional object despite the high dimensional space it is embedded in. In this way, the data set is considered as a manifold existing in this high dimensional space but with constraints that allow it to be described with the low dimensional representation due to the latent variable model. These latent variables are the coordinates of the manifold and more accurately represent the true dimensionality. It is possible that the manifold is in fact of a higher dimension, in which case the latent space coordinates are an approximation to the manifold. However, if this approximation and error estimates can be quantified then it can remain useful for most cases.

Chapter 5

GP Manifolds

The latent space variables of a GP-LVM function as coordinates for the data space manifold. Interpreting this way is consistent with the GP-Bayes filter, and provides some interesting usages for GP-LVMs. The resulting Gaussian process manifolds (GP-Manifold) produced by this interpretation are stochastic, meaning the manifold points are random variables rather than simple points.

5.1 GP Manifold Definition

There exist many formalized definitions for manifolds, so the definition used here must be specified clearly. A manifold M is a set of points $p \in M$ with the property that for each point p there exists an open neighborhood (or collection of points in M) that is homeomorphic[1] to an open set in $\Re^k$, where k is the dimension of M [26]. The collection of homeomorphic functions that map between $\Re^k$ to M are called charts, and have the constraint that for overlapping regions with different charts it is possible to define a change of coordinates between the two that is a continuous function. Typically the definition of these charts requires that they cover the manifold M completely, but in the cases discussed here it is not necessary to completely define all the charts needed to cover M. Instead, the manifold M is

[1] A homeomorphic function is one that is bijective and continuous

considered in patches that are covered by the training data.

For a Gaussian process manifold (GP-Manifold), the definition is modified slightly. Given a set of training points Y and a corresponding GP-LVM with a Gaussian process ϕ and latent variables X, a GP-Manifold is $M_{GP} = \{Y, \phi, X\}$. The Gaussian process ϕ contains the hyperparameters and covariance functions needed to provide the mapping from X to Y. Under this definition, Y are the manifold points, X are the manifold coordinates, and ϕ is the chart connecting coordinates to manifold points.

5.1.1 Notation

When working with manifolds, it is common to use mathematical objects known as tensors. A tensor of type (r, s) can be written as $T^{k_1 \cdots k_r}_{l_1 \cdots l_s}$, where r and s are the number of contravariant and covariant components respectively. Contravariant tensor components are ones which change with the inverse of a coordinate change, and are represented by superscript. Covariant tensor components change directly with the coordinates, and are represented by subscript. The contravariant and covariant components determine how the tensor changes under a change of coordinates, and the rank of the tensor is $(r + s)$.

In a coordinate system with points $p \in \Re^n$, the indices of the tensor T can range from $1 \ldots n$. The Einstein summation convention will be used when writing out tensors [26]:

$$A_{ij} B^j = \sum_{j=1}^{n} A_{ij} B^j \tag{5.1}$$

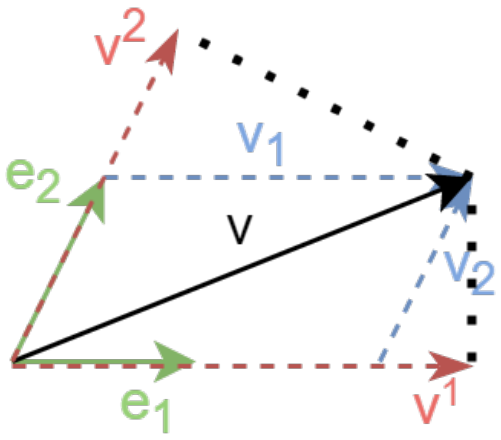

Figure 5.1: Diagram showing the contravariant and covariant components of a vector v with respect to basis vectors e_1 and e_2. Contravariant components are shown in blue, and represent a linear combination of basis vectors. Covariant component are shown in red, and represent the dot product of v with the basis vectors.

There are a few familiar mathematical objects than can be described as tensors. The first is that a vector is a tensor of rank 1, where the difference between a $(1, 0)$ and a $(0, 1)$ tensor is the difference in how the quantity changes under a coordinate transformation. Matrices can be considered as rank 2 tensors, where a matrix is typically written as a $(0, 2)$ tensor M_{ij} and the inverse is written as a $(2, 0)$ tensor M^{ij}. A scalar quantity would be a tensor of rank 0. An example of a rank 3 tensor could be the gradient of a matrix, where the gradient is taken with respect to each of the components of the matrix.

5.1.2 Importance

The interpretation of a data set as a manifold comes with benefits in terms of dimensionality reduction and the additional tools available. Reducing the dimensionality of the data improves the speed of operations and leads to more numerical stability. A manifold also allows the use of differential geometry, which is a field of mathematics that works with manifold structures and the application of calculus to non-Euclidean spaces [26]. Once a data set

can be understood as a differentiable manifold, the many theorems and tools studied in differential geometry become available to use to solve different types of problems.

One of the more important benefits of a manifold interpretation is a better definition of distance with respect to the manifold. Differential geometry allows for the computation of a metric tensor (see Section 5.2) on the manifold, which captures the idea of distance within the tangent space of the manifold. The tangent space is a Euclidean space associated with each point on the manifold with a dimension equal to the dimension of the manifold itself, while the metric tensor is a bilinear form[2] that defines an inner product within this tangent space. By considering the data set as a manifold that is embedded within the full state space, it possible to define distance *from* the manifold and to measure how far away a new sample point is from the manifold. Notions of area, angle, and volume are re-defined in relation to the manifold through the use of a metric tensor.

An example of how the manifold structure can be used is present in Havoutis [15], where path planning is constrained by learning a manifold structure and restricted to geodesics (see Section 5.3) along the manifold. The author's original training data is prepared by using constraints that are complicated and require a significant amount of time to compute trajectories. Once the manifold that is generated from these constraints is found, however, these constraints become implicit in the structure of the manifold. By definition, the manifold consists only of points that meet these constraints, so any point that is part of the manifold must satisfy the constraints. New trajectories generated from geodesics along the manifold are

[2]Mapping from a pair of vectors to a scalar

significantly faster to solve for while still meeting the original constraints with a high degree of accuracy. In Havoutis only the tangent space of the manifold is actually learned. The downside of learning only the tangent space means that you only have a linear approximation of the manifold at each point. This approximation is fine for manifolds that are close to Euclidean, but for more highly curved manifolds the errors in approximation will be more severe. Despite these downsides, the manifold structure provides significant benefits when extrapolating the training data.

5.1.3 Bayesian Manifolds

While there are many benefits to the manifold interpretation, they come with a cost of ensuring that the resulting manifold behaves consistently when using the metric tensor. Hauberg presents an argument for why typical manifold learning approaches fall short of their goal in [14]. The two main issues are extrapolating the manifold beyond the training set and interpolating through holes in the manifold that are real–actually part of the structure–rather than an absence of data. In the case of extrapolation, if the metric tensor tends to a constant as trajectories move away from the training data then the length of these paths as measured by the metric does not increase, allowing for paths to diverge from the manifold and "teleport" to another point. Trying to resolve this by linear extrapolation makes the assumption that far from the training data the manifold is effectively flat, and so in regions where there is a gap (such as in a torus) the real manifold structure is lost. Figure 5.2 from [14] illustrates both of these cases with a circular manifold.

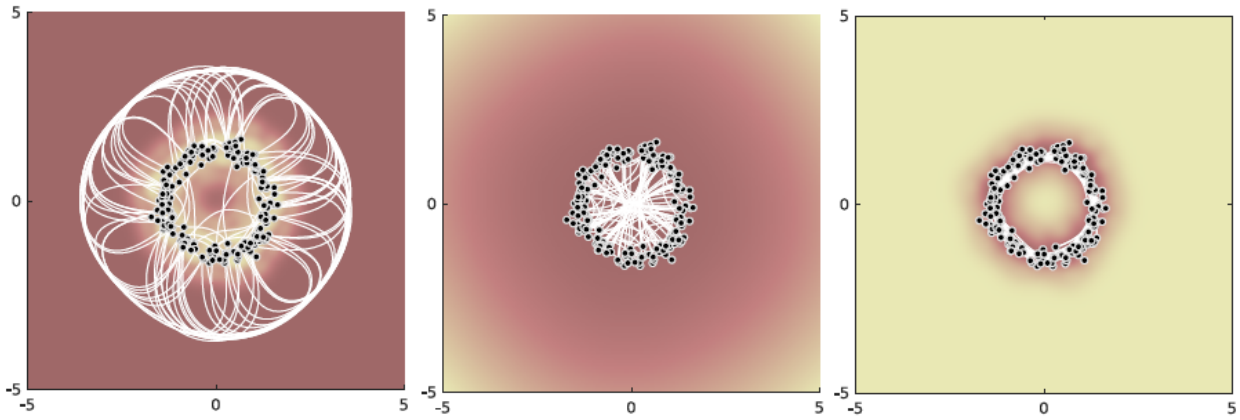

Figure 5.2: Geodesics on a circular manifold, shown in white, for three cases. Left is the case where the metric tensor disappears far from the manifold, and geodesics "teleport" around the edges since travel away from the training data carries no distance. Center is the case where the metric tensor is extrapolated linearly, and geodesics cross through structural holes. Right is the case using a probabilistic metric, where the structure of the circle is captured more accurately. The background color shows the magnitude of the distance metric, with light colors indicating larger values. [14]

According to Hauberg[14], trying to resolve these issues from a deterministic standpoint requires either giving up the smoothness of the manifold or giving up on trying to learn the structure accurately. Hauberg offers a solution in the form of changing the approach: using a Bayesian method to learn the manifold structure. The metric is now stochastic, but it can be well approximated by a deterministic one–the expected value of the metric tensor. The variance in the metric becomes the necessary component to solve the issues of extrapolation and interpolation. In regions that are not near training data, the metric tensor becomes larger due to the higher variance associated with being far from the training data. In effect, measurements of distance take into account the variance associated with it.

A Gaussian process makes a good candidate for Bayesian manifold learning since as presented earlier the GP-LVM can be interpreted as learning the manifold structure of a

data set. Since Gaussian processes naturally include the variance when making predictions, it fits nicely into Hauberg's requirements for an accurate manifold learning approach. In fact, Hauberg uses Gaussian processes as an example demonstrating the ability of a Bayesian approach to learn the structure of a manifold where deterministic methods fail [14]. The details of how to use a Gaussian process for tasks involving manifold structures will be presented below.

5.2 Metric Tensor

A manifold is called Riemannian if it has an associated metric tensor at each point on the manifold. Embedding a manifold in $\Re^n$ is sufficient to define such a metric tensor, typically written as g, or g_{ij} when considering the components of the tensor. The metric tensor defines an inner product in the tangent space of a manifold, giving a basis for the measurement of angles and lengths of tangent space vectors. The metric tensor is used in many common formulas in differential geometry, so it is a good place to start when working with a manifold.

5.2.1 Derivation from GP-Manifold

For a manifold of dimension k embedded in a Euclidean space with a chart $\phi(x)$ mapping from a coordinate space $\Re^k$ to the embedded coordinates in $\Re^n$ (where $k \leq n$), the metric tensor g_{ij} is the inner product of the Jacobian of ϕ with itself. Mathematically we have the following:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \phi_1}{\partial x_1} & \cdots & \frac{\partial \phi_1}{\partial x_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial \phi_n}{\partial x_1} & \cdots & \frac{\partial \phi_n}{\partial x_k} \end{bmatrix} \tag{5.2}$$

$$g = \mathbf{J}^\top \mathbf{J} \tag{5.3}$$

$$g_{ij} = \frac{\partial \phi}{\partial x_i} \cdot \frac{\partial \phi}{\partial x_j} \tag{5.4}$$

When using a GP-LVM for the manifold coordinates, $\phi(x)$ becomes a Gaussian process. Recall from section 2 the formula for evaluating a Gaussian process at a specific point (modified here to fit the context of a manifold):

$$\phi(x) = k(x, \mathbf{X})^\top \mathbf{K}^{-1} \mathbf{M} \tag{5.5}$$

Here $\mathbf{X}$ and $\mathbf{M}$ are the latent variable coordinates and the manifold points respectively from the initial training data. Computing the derivatives with respect to x needed for the Jacobian are straightforward since $\mathbf{K}$ and $\mathbf{M}$ are constant with respect to x, since they come from the training data while x represents a new sample point.

Taking the derivative of the Gaussian process comes down to taking the derivative of the covariance function $k(x, \mathbf{X})$. An RBF kernel is typically used due to some of the nice properties it has, including the relative ease of taking derivatives. As shown earlier, an RBF covariance function has the following form:

$$k(x, x') = \alpha_1 e^{\frac{-\alpha_2 \|x - x'\|^2}{2}} \tag{5.6}$$

Taking the derivative with respect to the ith component of x yields:

$$\begin{aligned} \frac{\partial k(x, x')}{\partial x_i} &= (-\alpha_2 x_i)\alpha_1 e^{\frac{-\alpha_2 \|x - x'\|^2}{2}} && (5.7) \\ &= (-\alpha_2 x_i) k(x, x') && (5.8) \end{aligned}$$

These derivatives form the Jacobian, and the variance is given by equation (3.5). The distance metric is given by the inner product of the Jacobian with itself, but care must be taken as the desired result is the expected value for g_{ij}, not just the deterministic value. The following equation shows how this is done:

$$\begin{aligned} \mathbb{E}[g_{ij}] &= \mathbb{E}[\mathbf{J}^\top \mathbf{J}]_{ij} && (5.9) \\ &= [\mathbb{E}[\mathbf{J}^\top]\mathbb{E}[\mathbf{J}]]_{ij} + V[\mathbf{J}]_{ij} && (5.10) \end{aligned}$$

The Jacobian is itself a Gaussian process, where the covariance is taken between derivatives. From Rasmussen [32], the following relationship is used:

$$\text{cov}(f_i, \frac{\partial f_j}{\partial x_j}) = \frac{\partial k(x_i, x_j)}{\partial x_j}, \quad \text{cov}(\frac{\partial f_i}{\partial x_i}, \frac{\partial f_j}{\partial x_j}) = \frac{\partial^2 k(x_i, x_j)}{\partial x_{di} \partial x_j} \tag{5.11}$$

Here the derivatives of a Gaussian process can be estimated through the covariance of

the derivatives with respect to the prior sample points. This covariance is found by taking the derivative of the covariance function. This results in the following expressions for the expected value and variance of the Jacobian:

$$\mathbb{E}[\mathbf{J}]_{ij} = \frac{\partial k(x, \mathbf{X})}{\partial x_j}^{\top} \mathbf{K}^{-1} \mathbf{M}_i \tag{5.12}$$

$$V[\mathbf{J}]_{ij} = \frac{\partial^2 k(x_i, x_j)}{\partial x_i \partial x_j} - \frac{\partial k(x, \mathbf{X})}{\partial x_i}^{\top} K^{-1} \frac{\partial k(x, \mathbf{X})}{\partial x_j} \tag{5.13}$$

Where M_i is the i^{th} component of all the manifold points. The variance term can be understood as computing the variance of the derivative, then subtracting the covariance of the derivatives with respect to the prior sample points. Once this expected value of the metric tensor has been computed, the manifold can be treated as though it were Riemannian using this metric tensor.

5.2.2 Intrinsic Nature

The metric tensor of a manifold has the property that it is *intrinsic*, meaning it does not depend on a specific coordinate system or how the manifold is embedded in a higher dimensional space [26]. In practice this means that the exact choice of the latent coordinates in the GP-LVM should not significantly impact the metric tensor, as long as the mapping the Gaussian process produces is one the minimizes the likelihood function in Section 3.2.2 to best fit the training data. Figure 5.3 shows how a manifold can have multiple coordinate mappings. Regardless of the coordinate system used, the metric tensor remains the same under an appropriate change of coordinates.

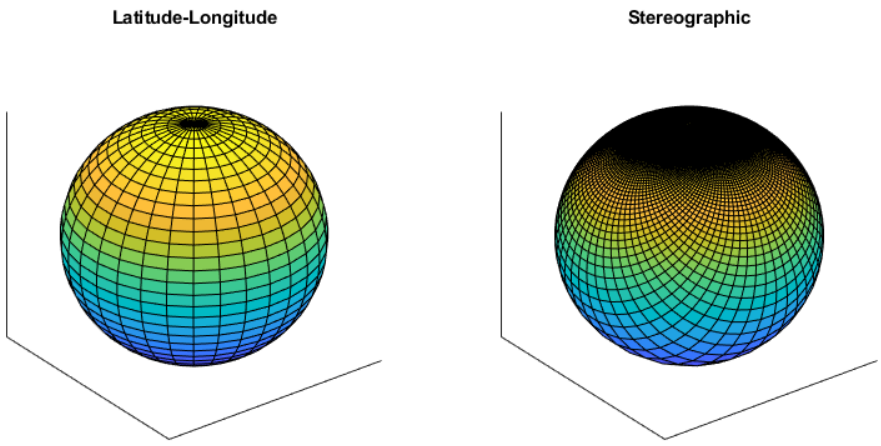

Figure 5.3: Two different coordinate systems for a spherical manifold. Left is the standard latitude-longitude, right is a stereographic projection. While the coordinate lines differ, geodesics and distance on the surface remains invariant.

5.3 Geodesics

Geodesics are the manifold analog of straight lines in a Euclidean space. They represent trajectories on the manifold which have no perpendicular acceleration within the tangent space of the manifold. Because of this geodesics form minimal energy solutions to path integrals on the manifold between points, or in other words they represent local minima in terms of distance between points. Solving for geodesics is tantamount to finding the shortest distance between two points on a manifold, a useful application.

For a given point p on a manifold and a tangent space vector $\vec{v}$, there exists a unique

geodesic $\gamma(t)$ such that:

$$\gamma(0) = p \tag{5.14}$$

$$\gamma'(0) = \vec{v} \tag{5.15}$$

When considering a geodesic connecting two points on a manifold, however, the uniqueness is not guaranteed. The existence of a geodesic given two points holds only if the manifold is compact, or contains all boundary points[26]. Such a manifold is said to be geodesically complete, and it can be shown that GP-Manifolds meet this criteria

5.3.1 Geodesics on GP-Manifolds

Solving for geodesics on a manifold can be done in several ways. The most straight forward approach is to solve the differential equations that define a geodesic. Borrowing from [26], the differential equations take the following form:

$$\frac{d^2\gamma^i}{dt^2} + \Gamma^i_{jk}(\gamma(t))\frac{d\gamma^j}{dt}\frac{d\gamma^k}{dt} = 0 \text{ for all } i \tag{5.16}$$

Here we use the Christoffel symbols Γ^i_{jk}, a tensor-like object that represents how the tangent space of the manifold changes from point to point where i, j, and k are indices from $1...dim(M)$. The Christoffel symbols are given by the following equation (simplified through Einstein summation):

$$\Gamma^i_{jk} = \frac{1}{2} g^{il} \left(\frac{\partial g_{lj}}{\partial x_k} + \frac{\partial g_{lk}}{\partial x_j} - \frac{\partial g_{jk}}{\partial x_l} \right) \tag{5.17}$$

Combining this equation with the definition of g_{ij} in equation (5.4) produces a much simpler expression in terms of the Gaussian process charts for the manifold:

$$\begin{aligned}
\Gamma^i_{jk} &= \frac{1}{2} g^{il} \left(\frac{\partial}{\partial x_k}(\phi_l \cdot \phi_j) + \frac{\partial}{\partial x_j}(\phi_l \cdot \phi_k) - \frac{\partial}{\partial x_l}(\phi_j \cdot \phi_k) \right) && (5.18) \\
&= \frac{1}{2} g^{il} (\phi_{lk} \cdot \phi_j + \phi_l \cdot \phi_{jk} + \phi_{lj} \cdot \phi_k + \phi_l \cdot \phi_{jk} - \phi_{lj} \cdot \phi_k - \phi_{lk} \cdot \phi_j) && (5.19) \\
&= g^{il} (\phi_l \cdot \phi_{jk}) && (5.20)
\end{aligned}$$

Note that the Christoffel symbols require a second derivative of the Gaussian process to compute. Taking (5.8) as a starting point, the second derivative is then:

$$\begin{aligned}
\frac{\partial^2 k(x, x')}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_j}(-\alpha_2 x_i) k(x, x') && (5.21) \\
&= -\alpha_2 \delta_{ij} k(x, x') + (\alpha_2)^2 x_i x_j k(x, x') && (5.22) \\
&= (\alpha_2 x_i x_j - \delta_{ij}) \alpha_2 k(x, x') && (5.23)
\end{aligned}$$

where δ_{ij} is the Kronecker delta function[3]. When using the metric tensor of a GP-Manifold, recall that it is necessary to use the expected value of the metric tensor from equation 5.10. The derivative of the variance term must be taken and used in equation 5.17. The derivatives for the variance term entries $V[\mathbf{J}]_{ij}$ are given by:

[3] $\delta_{ij} = 1$ for $i = j$, 0 otherwise

$$\frac{\partial V[\mathbf{J}]_{ij}}{\partial x_k} = \frac{\partial}{\partial x_k}\left(\frac{\partial^2 k(x_i,x_j)}{\partial x_i \partial x_j} - \frac{\partial k_*}{\partial x_i}^\top K^{-1}\frac{\partial k_*}{\partial x_j}\right) \tag{5.24}$$

$$= \frac{\partial^3 k(x_i,x_j)}{\partial x_i \partial x_j \partial x_k} - \frac{\partial^2 k_*}{\partial x_i \partial x_k}^\top K^{-1}\frac{\partial k_*}{\partial x_j} - \frac{\partial k_*}{\partial x_i}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_j \partial x_k} \tag{5.25}$$

where $k_* = k(x, \mathbf{X})$. The first term $\frac{\partial^3 k(x_i,x_j)}{\partial x_i \partial x_j \partial x_k}$ is different from the previous derivative terms, since it only involves individual components of the input vector x. Using the RBF covariance function defined earlier, we have:

$$k(x_i,x_j) = \alpha_1 e^{\frac{-\alpha_2(x_i^2+x_j^2)}{2}} \tag{5.26}$$

$$\frac{\partial^2 k(x_i,x_j)}{\partial x_i \partial x_j} = \alpha_2^2 x_i x_j k(x_i,x_j) \; for \; i \neq j \tag{5.27}$$

$$\frac{\partial^3 k(x_i,x_j)}{\partial x_i \partial x_j \partial x_k} = \begin{cases} 0 & \text{for } i = j, k \neq i, j \\ (\alpha_2^2 x_j - \alpha_2^3 x_i^2 x_j)k(x_i,x_j) & \text{for } k = i \\ (\alpha_2^2 x_i - \alpha_2^3 x_i x_j^2)k(x_i,x_j) & \text{for } k = j \end{cases} \tag{5.28}$$

Combining these in the equation for the Christoffel symbols results in the following expression for the variance term, which can then be added to the expression in equation 5.17. After all the term canceling, the final expression for the Christoffel symbols is:

$$\Gamma^i_{jk} = g^{il}\left(\phi_l \cdot \phi_{jk} - \frac{\partial {k_*}^\top}{\partial x_l} K^{-1} \frac{\partial^2 k_*}{\partial x_j \partial x_k} + \delta_{jk}(1-\delta_{jl})(1-\delta_{kl})(\alpha_2^2 x_l - \alpha_2^3 x_i x_j^2) k(x_j, x_k)\right) \tag{5.29}$$

Once the necessary components of the differential equations have been found, all that remains to find a geodesic between two points is to solve a boundary value differential equation problem. Solvers for these types of problems exist already and are generally sufficient to find the necessary solution. In some cases, however, the solution to a particular problem can be quite sensitive and difficult for a solver to converge. This is typically the case when the metric tensor is large, which occurs for a GP-manifold when the training data is sparse in the region where the geodesic is computed. In these cases another approach to finding geodesics can be used, called curve shortening flows.

5.3.2 Curve Shortening Flows

The principle behind curve shortening flows is to iteratively and smoothly adjust a path so that it is as short as possible [11]. To do this, points on the path are moved to minimize the amount of curving or turning that the path does. In a Euclidean space, this would result in a straight line. More generally, however, it results in a geodesic when applied to a curve on a manifold. To formalize the notion of the amount of turning that a path does requires the concept of the curvature κ of a curve.

Consider a curve $\gamma(t)$ in $\Re^n$. At each point on γ, let $\hat{T}(t)$ be the unit vector tangent to

the curve. $\hat{T}$ can be found by dividing $\vec{\gamma}'(t)$ by its length, and represents motion in a straight line. Turning motion can be found by taking the derivative of the unit tangent vector, noting that $\hat{T}'(t) \cdot \hat{T}(t) = 0$ since the length of $\hat{T}$ is constant. This new vector is normal to $\gamma(t)$, and the unit vector $\hat{N}(t)$ is used to describe its direction. Curvature is defined by the norm $\kappa(t) = \|\hat{T}'(t)\|$, giving the relationship $\hat{T}'(t) = \kappa(t)\hat{N}(t)$.

The curvature represents how much $\gamma(t)$ is deviating from a straight line at a given point. To straighten out the curve, each point is deformed along $\hat{N}(t)$ by an amount proportional to the curvature. Iterating this approach results in a path that is as straight as possible, and by extension the shortest path between the start and end points. Figure 5.4 shows one step of this iteration.

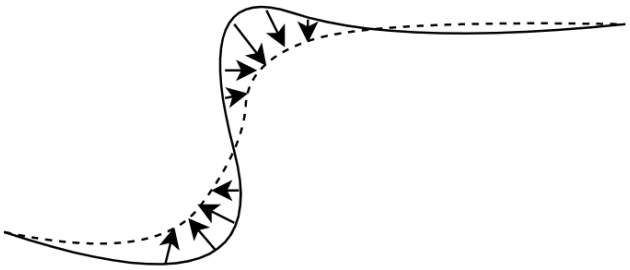

Figure 5.4: Diagram showing one iteration of a curve shortening flow. The solid black curve is evaluated to find the flow (arrows) that will reduce the curvature at each point. The resulting dashed curve is shorter than the original. Repeating this process results in a straight line.

5.3.3 Curves on Manifolds

Curve shortening flows become more complicated when considering curves that lie on a manifold. The curvature of a trajectory on a manifold must take into account the curvature

due to the way the manifold itself is curved. Solving the differential equations in the first approach requires no additional changes due to the way it works entirely in the coordinate system. Curve shortening flows, however, require finding a tangent space derivative along a curve.

To take the appropriate derivative, the notion of a covariant derivative is needed. In literature the covariant derivative of a tangent space vector field V along γ, given the notation D_tV is defined as follows [26]:

$$(D_tV)^k = \dot{v}^k + \Gamma^k_{ij}\dot{\gamma}^i v^j \tag{5.30}$$

where v^k are the components of the vector field V and Γ^k_{ij} are the Christoffel symbols. With this definition, the previous approach of curve shortening flows is possible using this more general definition of a derivative.

One advantage to the curve shortening flow approach is that due to its iterative nature, it is possible to obtain approximate solutions by terminating the process early. This means in some applications where the time available for solving is constrained, it is possible to guarantee at least an approximate solution within the available time. Additionally, the solution can be improved later if time allows, making this option very useful in real time scenarios.

5.4 Curvature Tensor

An important part of a manifold interpretation for a data set is understanding the geometry of the manifold. The metric tensor and geodesics offer some insight into how the geometry impacts distance and angle measurement, but there are more effects caused by the geometry of the manifold. The Riemann curvature tensor offers a quantitative way to talk about these effects, such as how vector fields transport along the manifold. In Euclidean geometry, Lie brackets are used to measure the commutativity when traveling along different fields. When vectors are transported along vector fields, the order in which they are transported can result in differences in the final vector, and it is this difference that Lie brackets measure. Manifolds present an additional way in which the commutativity changes, and this is measured by the curvature tensor.

Along with measuring the transport of vectors, the Riemann curvature tensor can be contracted to find the Ricci curvature tensor and the scalar curvature. These two tensors give a measure of how the volume of a shape deviates from a Euclidean space. The Ricci tensor measures this deviation when moving along a geodesic, while the scalar curvature is an average at a point when considering the Ricci tensor in all directions. Both are helpful in understanding the geometry of a manifold.

5.4.1 Derivation from Metric Tensor

The Riemann curvature tensor is derived from the metric tensor, making it also an intrinsic property of a manifold. The mathematical form is given by the following [26]:

$$R^l_{ijk} = \frac{\partial \Gamma^l_{jk}}{\partial x^i} - \frac{\partial \Gamma^l_{ik}}{\partial x^j} + \Gamma^h_{jk}\Gamma^l_{ih} - \Gamma^h_{ik}\Gamma^l_{jh} \tag{5.31}$$

Recall that the Christoffel symbols Γ^i_{jk} are derived from the metric tensor g_{ij}, meaning R^l_{ijk} can be written in terms of the metric tensor itself. From the definition in (5.17), it first appears that the curvature tensor requires 3rd order derivatives of the original Gaussian process charts. Examination of the first two terms shows:

$$\begin{aligned}
\frac{\partial \Gamma^l_{jk}}{\partial x_i} - \frac{\partial \Gamma^l_{ik}}{\partial x_j} &= \frac{\partial}{\partial x_i}(g^{lm}(\phi_m \cdot \phi_{jk})) - \frac{\partial}{\partial x_j}(g^{lm}(\phi_m \cdot \phi_{ik})) && (5.32)\\
&= \frac{\partial g^{lm}}{\partial x_i}(\phi_p \cdot \phi_{jk}) - \frac{\partial g^{lm}}{\partial x_j}(\phi_m \cdot \phi_{ik}) && (5.33)\\
&\quad + g^{lm}(\phi_{mi} \cdot \phi_{jk}) + g^{lm}(\phi_m \cdot \phi_{ijk}) && (5.34)\\
&\quad - g^{lm}(\phi_{mj} \cdot \phi_{ik}) - g^{lm}(\phi_m \cdot \phi_{ijk}) && (5.35)
\end{aligned}$$

and it can be seen that the terms with a 3rd order derivative cancel. When using the expected value of the metric tensor, the variance terms need to be included as well. Recall from equation (5.29) and consider only the terms coming from the variance, we have:

$$[\Gamma^i_{jk}]_{var} = g^{il}(-\frac{\partial k_*}{\partial x_l}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_j \partial x_k} + \delta_{jk}(1-\delta_{jl})(1-\delta_{kl})(\alpha_2^2 x_l - \alpha_2^3 x_l x_j^2)k(x_j,x_k)) \quad (5.36)$$

$$[\Gamma^i_{jk}]_A = g^{il}(-\frac{\partial k_*}{\partial x_l}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_j \partial x_k}) \quad (5.37)$$

$$[\Gamma^i_{jk}]_B = g^{il}(\delta_{jk}(1-\delta_{jl})(1-\delta_{kl})(\alpha_2^2 x_l - \alpha_2^3 x_l x_j^2)k(x_j,x_k)) \quad (5.38)$$

$[\Gamma^i_{jk}]_A$ is fairly straightforward to evaluate for the curvature tensor:

$$\begin{aligned}\frac{\partial [\Gamma^l_{jk}]_A}{\partial x_i} - \frac{\partial [\Gamma^l_{ik}]_A}{\partial x_j} &= \frac{\partial}{\partial x_i}(g^{lm}(-\frac{\partial k_*}{\partial x_m}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_j \partial x_k})) \\ &\quad -\frac{\partial}{\partial x_j}(g^{lm}(-\frac{\partial k_*}{\partial x_m}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_i \partial x_k})) \quad (5.39)\\ &= \frac{\partial g^{lm}}{\partial x_j}(\frac{\partial k_*}{\partial x_m}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_i \partial x_k}) - \frac{\partial g^{lm}}{\partial x_i}(\frac{\partial k_*}{\partial x_m}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_j \partial x_k}) \\ &\quad + g^{lm}(\frac{\partial^2 k_*}{\partial x_m \partial x_j}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_i \partial x_k} - \frac{\partial^2 k_*}{\partial x_m \partial x_i}^\top K^{-1}\frac{\partial^2 k_*}{\partial x_j \partial x_k} \\ &\quad + \frac{\partial k_*}{\partial x_m}^\top K^{-1}\frac{\partial^3 k_*}{\partial x_i \partial x_j \partial x_k} - \frac{\partial k_*}{\partial x_m}^\top K^{-1}\frac{\partial^3 k_*}{\partial x_i \partial x_j \partial x_k}) \quad (5.40)\end{aligned}$$

Terms involving a 3rd order derivative again cancel here. For the final term $[\Gamma^i_{jk}]_B$, the single component covariance function from (5.26) is being differentiated here:

$$\begin{aligned}\frac{\partial [\Gamma^l_{jk}]_A}{\partial x_i} - \frac{\partial [\Gamma^l_{ik}]_A}{\partial x_j} &= \frac{\partial}{\partial x_i} g^{lm}(\delta_{jk}(1-\delta_{jl})(1-\delta_{kl})(\alpha_2^2 x_l - \alpha_2^3 x_l x_j^2)k(x_j,x_k)) \\ &\quad -\frac{\partial}{\partial x_j} g^{lm}(\delta_{ik}(1-\delta_{il})(1-\delta_{kl})(\alpha_2^2 x_l - \alpha_2^3 x_l x_i^2)k(x_i,x_k)) \quad (5.41)\end{aligned}$$

If $i \neq j$, then both terms differentiate to zero. If $i = j$, then both terms cancel. This leaves only the terms from $[\Gamma^i_{jk}]_A$ in the final equation for the curvature tensor.

Once the Riemann curvature tensor has been found, it can be used to find the Ricci curvature tensor as well as the scalar curvature. Contracting the tensor requires summing across indexes on the same tensor. Note that the variable R is reused, with the tensor rank indicating which curvature it is. The Riemann curvature tensor has a (1,4) tensor rank, the Ricci curvature tensor has a (0,2) tensor rank, and the scalar curvature is simply a scalar quantity.

$$R_{ij} = R^k_{kij} \tag{5.42}$$

$$R = \text{trace}\, R_{ij} = g^{ij} R_{ij} \tag{5.43}$$

5.4.2 Interpretation of Curvature

Curvature represents how the manifold behaves in a non-Euclidean way. To estimate of how important it is to compensate for non-linear behavior, the magnitude of the scalar curvature is useful. Values near zero represent a relatively flat region, where linear approximations will work well. When the magnitude is higher, the manifold is more highly curved and methods to account for the non-linearity will need to be used. Particular applications could consider setting a threshold for the scalar curvature above which the type of model used could be changed.

The Riemann curvature tensor has some extra utility when considering control methods applied to a manifold. In control theory, Lie brackets are a tool used to describe different control strategies and how their application in different orders changes the outcome [38]. Importantly, these typically apply to Euclidean control spaces. When applying the same methods to a manifold, the curvature adds another layer to how the ordering of different controls produces a different output. Both the Lie bracket and the Riemann curvature tensor are needed to completely describe the system's behavior.

To understand why the curvature needs to be taken into account, consider a typical scenario involving a Lie bracket where a point is moved along two vector fields in different orders. The Lie bracket describes how separate the endpoints are depending on which order the two vector fields are traversed. In a case where motion is along straight lines, the Lie bracket is zero, representing a commutative operation. The case of moving along straight lines is analogous to tracing the edges of a parallelogram when considering Euclidean spaces. The behavior on a manifold with non-zero Riemann curvature is more complicated.

Consider a sphere and the case of moving along perpendicular geodesics. On a flat manifold, switching between two perpendicular geodesics would trace out a rectangle. On a sphere, however, the geodesics are the great circles of the sphere. If the two perpendicular geodesics are traced for a quarter of the total circumference, the two endpoints can be connected by another geodesic perpendicular to both with a length equal to a quarter of the circumference. The final shape is an equilateral triangle on the sphere with right angles at all 3 points. This sort of geometric oddity is expressed by the Riemann curvature tensor.

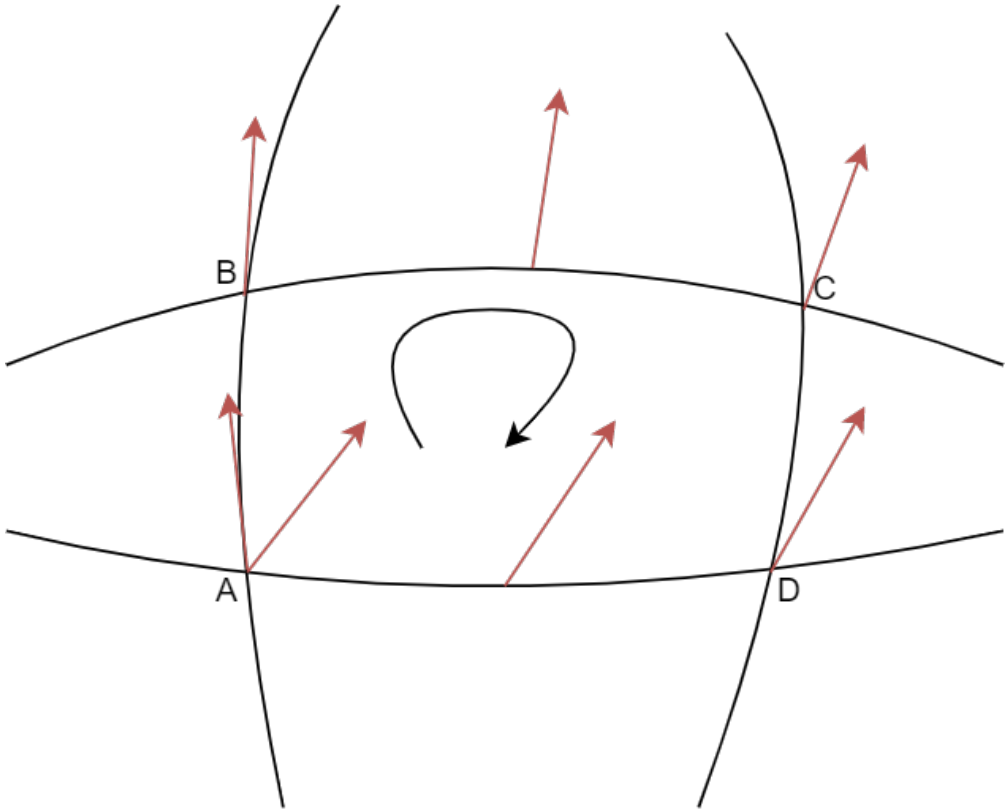

Figure 5.5: Parallel vector transport along geodesics for a non-Euclidean manifold. The red vector is moved along the geodesics such that the angle relative to the tangent of the line it follows remains constant. Transport around this closed loop results in changes to the vector based on the curvature of the manifold.

Another example of how the curvature operates is in the case of parallel vector transport along geodesics. Figure 5.5 shows how the curvature of a manifold impacts the way that vectors change when they are transported. The curvature tensor mathematically describes these changes.

5.4.3 Optimizing for Curvature Preservation

Using curvature as a measurement of the geometry of a manifold presents an opportunity for incorporating another source of extra information. The GP-LVM optimization process can be modified to include a likelihood term corresponding to the curvature, which can be used if there is any available curvature information. Even if it was only available for some points in the data set, it could still help produce manifolds that better fit the training data. To find the needed likelihood term, the curvature itself would need to be completely specified by the Gaussian processes used to find it.

Chapter 6

Example Usages/Experiments

It remains to show the viability of Gaussian process manifolds applied to practical problems. One limitation of a Gaussian process approach as compared to a more popular neural net is that the GP-LVM needs to keep the entire training data set while a neural net only needs to keep the neuron weights. A method to reduce the necessary data is presented by resampling the training data. Next the usage of the GP-Bayes filter in the case of radiation induced sensor data is demonstrated, showing that a GP-Bayes filter can help recover sensor information. Finally, the benefits of using a GP-manifold over traditional manifold learning approach is shown by using geodesics for path planning.

6.1 GP-LVM Resampling and Compression

6.1.1 GP-LVM vs Neural Net Size Comparison

One major downside to the use of GP-LVMs is they typically require storing and using the entire training data set. Because of the way the covariance function works, all of the data points in the training set are needed to compute the covariance vector that is used to evaluate the Gaussian process at a new data point. This is in contrast to other methods such as neural

nets, which store neuron weights but do not require the original training data once the net has been trained.

For small training sets, Gaussian processes work remarkably well, and don't overfit to the same extent that a neural net trained on a small data set would. As the size of the training data grows, the Gaussian process model improves but the time it takes to compute the latent variables for optimization increases as does the time needed to compute the output for a new input. One approach to improve the speed of Gaussian processes during optimization is to use stochastic methods that pick a sample of points and optimize them locally, similar to what is done with large neural nets [21]. While this is an improvement, it does not remove the need for retaining a potentially large amount of training data after optimization is complete.

Since Gaussian processes do perform well with a low number of training points, it is possible to re-sample the training data to find a smaller set that provides sufficient accuracy while still retaining the benefits of optimizing using all the data available. The number of points to keep can be selected to provide accuracy within specified margins, ensuring that the quality of the GP-LVM remains good. There are several approaches to selecting a representative set of points, each with their own drawbacks and benefits.

6.1.2 Re-sampling for Compression

To reduce the size of a trained GP-LVM, the training data points used can be re-sampled to select a smaller representative set. It is important that the training data selected does not decrease the accuracy of the Gaussian processes making up the GP-LVM. To find the best approach, different methods will be compared to each other. There are several metrics by which the accuracy of a given method can be evaluated.

The first metric is the error when reproducing the training data set from the latent variables using the Gaussian processes from the re-sampled set. Given a training data set Y with latent variables X and re-sampled data and latent variables $\bar{Y}$ and $\bar{X}$, compute an estimate $\tilde{Y}$ of Y using the latent variables X according to:

$$\tilde{Y} = k(X, \bar{X})\mathbf{K}_{\bar{X}}\bar{Y} \tag{6.1}$$

where $\mathbf{K}_{\bar{X}}$ is the covariance matrix formed using $\bar{X}$. The RMSE of the estimate is then computed from the Euclidean difference of $\tilde{Y}$ and Y. This error is straightforward to measure and serves as a metric for the quality of a given re-sampling of the training data.

Another metric to consider is how well the geometry of the manifold is preserved. Using the scalar curvature as a measure of local geometry, the re-sampled manifold can be compared to the original. While the full curvature tensor might be a more complete picture of the

manifold geometry, it is much easier to measure error with the scalar curvature. The equations for computing the scalar curvature given in section 5.4 are repeated here:

$$R = g^{ij} R_{ij} \tag{6.2}$$

$$R_{ij} = R^k_{kij} \tag{6.3}$$

$$R^l_{ijk} = \frac{\partial \Gamma^l_{jk}}{\partial x^i} - \frac{\partial \Gamma^l_{ik}}{\partial x^j} + \Gamma^h_{jk}\Gamma^l_{ih} - \Gamma^h_{ik}\Gamma^l_{jh} \tag{6.4}$$

$$\Gamma^i_{jk} = g^{il}(\phi_l \cdot \phi_{jk} - \frac{\partial k_*}{\partial x_l}^\top K^{-1} \frac{\partial^2 k_*}{\partial x_j \partial x_k} + \delta_{jk}(1-\delta_{jl})(1-\delta_{kl})(\alpha_2^2 x_l - \alpha_2^3 x_i x_j^2) k(x_j, x_k)) \tag{6.5}$$

A third metric that is not always available is a measure of how well the re-sampled manifold satisfies the original constraints that produced the training data. In some cases the constraint function is known and easy to evaluate, offering another way to measure the quality of a particular re-sampling. In the case of a robotic manipulator, the constraint could be minimization of total joint angles. Each point produced by the re-sampled manifold could then be evaluated in terms of the robotic manipulator state that the point corresponds to.

To test different methods, a known manifold is used to train a GP-LVM. The manifold consists of a set of points that satisfy a known constraint along with a coordinate map that is used to initialize the latent variables of the GP-LVM before they are optimized. Different methods are then applied to re-sample for a smaller number of points, and each of the three listed metrics are compared. The test manifold is a surface known as a Monkey Saddle, shown

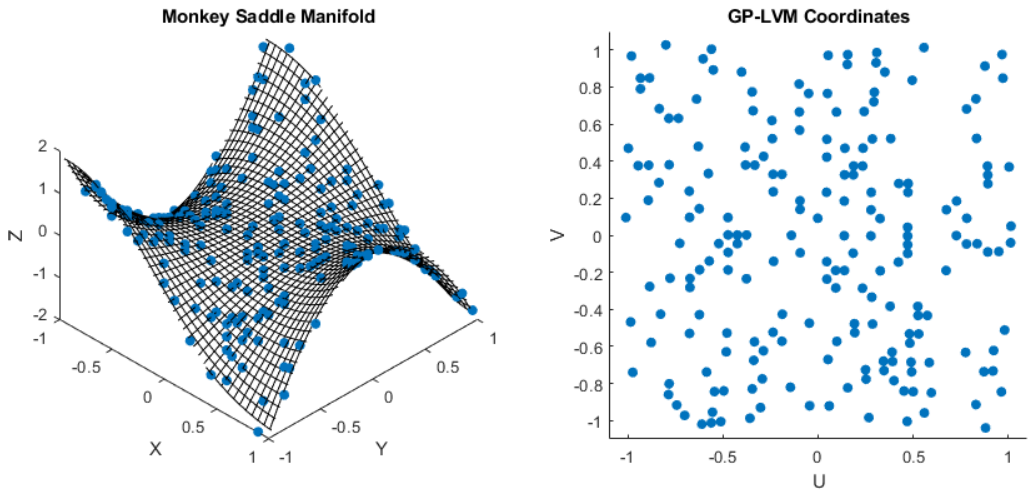

Figure 6.1: Monkey Saddle surface used to test GP-LVM re-sampling. Left shows the manifold, with points used for the baseline GP-LVM shown. Right shows the coordinates of the manifold produced by the GP-LVM

in Figure 6.1. Points on this surface satisfy the constraint of $z = x^3 - 3xy^2$, so it is possible to check the third metric of how well a given GP-LVM satisfies the manifold's constraints.

6.1.3 Re-sampling Methods

Random Re-sample The most straightforward method to re-sampling is to simply randomly select a number of points from the original set and use those. While this is simple to implement, it does have some significant downsides. The first is that a truly random sampling is prone to clustering, meaning some regions of the manifold will have lots of information while others may be more sparse. In regions where there are few or no sample points, the manifold accuracy will suffer due to the lack of information, while the extra samples in the clustered locations will provide no additional benefit in terms of manifold performance. The second issue is that in cases where the original data is also clustered, it only makes the first problem worse. When the original training data has regions of dense

clusters, a true random sample will tend to select points from these dense clusters resulting in an amplification of the clustering issue that random sampling already tends to experience

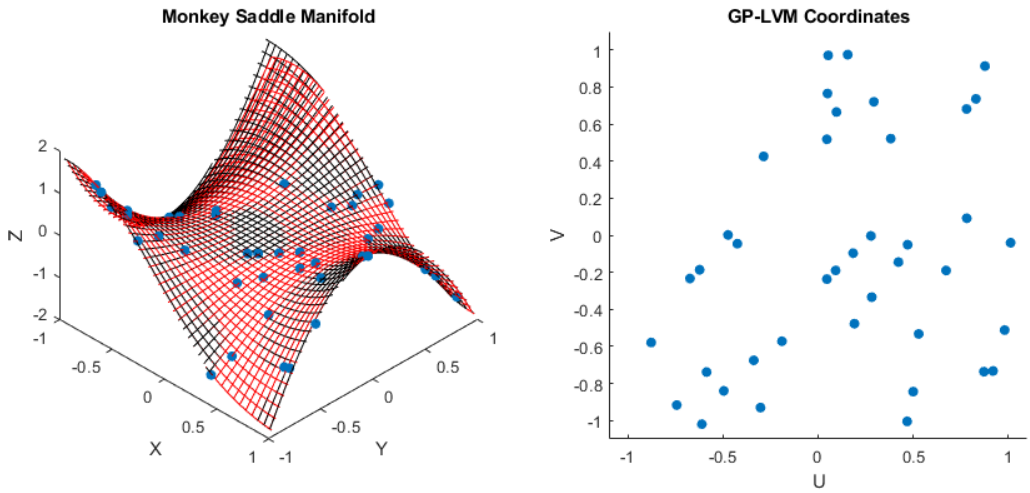

Figure 6.2: Randomized Re-sampling of manifold data. Left shows the original manifold in black and the re-sampled manifold in red. Right shows the sample of coordinates obtained by this method

As can be seen in Figure 6.2, the problem of clustering shows up as the higher concentration of points near the middle, with a large gap in the top left of the coordinate plot. Looking at the predicted manifold in red compared to the original in black, it is visually apparent that in this region there is a higher degree of inaccuracy. The red lattice does not fit well to the black ground truth, meaning trying to use this portion of the manifold will be prone to errors.

Mitchell's Best Candidate Ideally, the set of points that produce the best match to the original data are the ones that would be chosen. This would require examining all possible subsets of the original data; however, even for modest sizes of data sets there are far too many possibilities. One observation based on the flaws of the previous method is

that having the sample points spread out evenly results in better performance. A method known as Mitchell's best candidate [30] is used to produce a sample where no points are closet together than some minimum distance. The resulting distribution of points is evenly spaced across the sample space. The way this method works is that points are chosen incrementally. To add a new point first a number of candidates are generated and the candidate that is furthest from any other point already included is selected as the best candidate, then added to the existing set of points. Mathematically,

$$D(p) = \min_{x_i}(dist(p, x_i)) \tag{6.6}$$

where p is a candidate point and x_i are the points currently in the re-sampled set. The point with largest value of $D(p)$ among the candidates is then added to the re-sampled set.

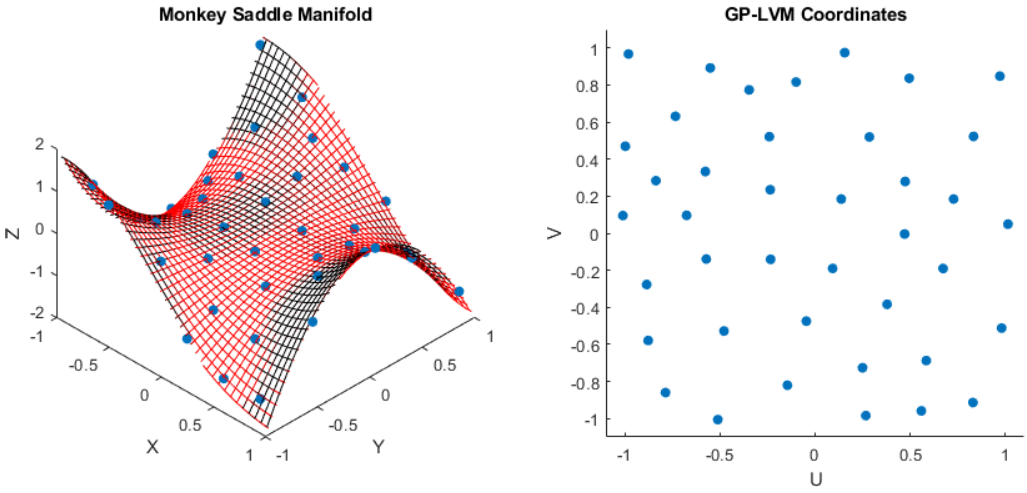

Figure 6.3: Mitchell's best candidate re-sampling of manifold data. Left shows the original manifold in black and the re-sampled manifold in red. Right shows the sample of coordinates obtained by this method

As can be seen in Figure 6.3, the coordinates from the re-sample are much more spread

out and cover the space much better. The problem of clustering is gone entirely, and the improvements in manifold quality are evident when observing how close the re-sampled red manifold approximates the original black one. Measuring the quality by the metrics described above also show significant improvements.

Maximal Variance While Mitchell's best candidate approach works well, it is natural to consider in what ways it may be improved. The core of the best candidate method is that a number of points are considered and the one with the highest score according to some scoring function is chosen. The most obvious place to look for improvements is with this scoring function.

Several different modifications were tested to find a scoring method that resulted in better re-sampling according to the metrics chosen. Without containing a component relating to the distance between points, however, the quality was always inferior compared to the original Mitchell's best candidate method. This makes sense, as without a distance component to the scoring function the resulting re-sample will tend to contain a degree of clustering. By considering functions that incorporate the distance between sample points, a superior method was discovered by taking the product of distance with the Gaussian process variance of any candidate point. The scoring function for the modified Mitchell's best candidate method then looks like:

$$S(p) = D(p)V[f(p)] \tag{6.7}$$

where $D(p)$ is the distance function from equation (6.6), and $V[f(p)]$ is the variance of the Gaussian process evaluated at p. To compute the variance for candidate points, the partial Gaussian process based on the current set of included points is evaluated and used to find the variance according to equation (3.5). The product of the distance and this variance is then used as the score to choose the best candidate. Intuitively, the resulting approach can be understood as trying to find candidates that are as far away from other points as possible and are not well predicted by the current set of points. For the first several points added, the variance will always be high and so the distance term is the important one. As the number of points in the set grows, the distance shrinks and the variance term becomes more important in the selection of candidates. As the variance term begins to dominate, the selection method increasingly prioritizes adding points which are not well predicted and thus improves the overall accuracy of the re-sample.

6.1.4 Quality of Re-sampled Data

In addition to the Monkey Saddle surface, other data sets were used to test these different methods. Data from the robotic arm discussed in section 6.2 was used in addition to data from the Robonaut experiments [1]. While the manifolds for these are not easily visualized, the re-sampling methods described above can still be applied and the quality measured according to the metrics discussed.

	Randomized	Mitchell's Best	Maximal Variance
Monkey Saddle	3.1×10^{-3}	9.37×10^{-6}	4.55×10^{-6}
HEBI Manipulator	1.88×10^{-1}	2.35×10^{-2}	8.53×10^{-3}
Robonaut Data	1.23×10^{-2}	3.35×10^{-3}	5.59×10^{-4}

Table 6.1: Training set squared error comparison of re-sampling methods on different manifolds. Average point error averaged over 100 trials.

	Randomized	Mitchell's Best	Maximal Variance
Monkey Saddle	1.01×10^{-2}	2.32×10^{-3}	2.09×10^{-3}
HEBI Manipulator	1.04×10^{5}	1.39×10^{5}	1.14×10^{5}
Robonaut Data	1.16×10^{6}	1.13×10^{6}	3.29×10^{6}

Table 6.2: Curvature squared error comparison of re-sampling methods on different manifolds. Average curvature error averaged over 100 trials.

For the Monkey Saddle data, 200 training points on the manifold are used to form the initial GP-LVM, and it is down-sampled to 40 points. The 3DoF HEBI manipulator contains 309 training points in the original GP-LVM, and is down-sampled to 60 points. Finally, the Robonaut data used a set of sample trajectories consisting of 933 points down-sampled to 200.

Table 6.1 shows the results of the three re-sampling methods applied to each test case when looking at the error on reproducing the training set. In the case of the Monkey Saddle, the error is the Euclidean distance error when looking at points on the surface. The HEBI and Robonaut error is in the joint angles and has the units of radians. As can be seen from the results, the randomized re-sampling method performs the worst, with improvement by the Mitchell's best method and the least error when incorporating the variance into the candidate scoring function.

When looking at the scalar curvature(6.2) error to get a sense of how well the re-sample preserves the geometry, some complications appear. Table 6.2 shows the error in curvature for the three different manifolds considered. Of these three, the Monkey Saddle is the only one that is a simulated data set, and is the only one where the curvature error is low enough to be considered good. The curvature error on the two manifolds from actual data shows a value far too high to be considered accurate, prompting consideration on the value of the curvature as a measure of manifold geometry.

The highly inaccurate curvature values have a few interpretations. One possible interpretation is that the dimension of the manifold is insufficient to accurately capture the geometry. This could be further investigated by varying the dimension of the manifold, but seems unlikely to generate such large error. The more probable explanation is the higher noise present in the physical data resulting in curvature values that vary much more sharply than expected, meaning that the curvature does not adequately describe the geometry of the manifold and is a poor metric for determining the quality of a given re-sampling.

6.2 GP-Bayes: Sensor Reconstruction

GP-LVMs have a practical usage in the form of GP-Bayes filters. As shown in Ko where a GP-Bayes filter was used to help train a robotic arm to mimic training data in a closed loop setting, it is usable in cases not just with motion capture data as in Wang or Levine, but also in physical systems. Taking this in a different direction, it can be used for sensor correction

in complex systems.

6.2.1 Problem Statement

In environments with high levels of radiation robotic systems present a way to perform tasks without putting human operators at risk. Robots do not provide a complete solution, however, due to the way that radiation causes electronic components to degrade. Solutions to this problem typically focus on hardening the components of a system to increase their lifetime. A software solution involving machine learning offers a parallel approach that can supplement other techniques by improving overall system performance as individual components degrade or begin to fail.

In cases where the degradation behavior of the affected component is known, it's often possible to re-calibrate and correct the degraded behavior. Often times however the way in which a component fails is either not known, or the component fails completely rather than degrading smoothly. In these situations, modeling of the component to the degree needed to re-calibrate is impossible. By considering the entire system instead, some of the component's behavior can be estimated sufficient to continue operation of the system.

To recreate the necessary information, it needs to be assumed that there exists some sort of composite system state that contains the information sufficient to estimate the degraded component's behavior. Latent variable modeling can be used to find this unobservable system

state through the use of a GP-LVM. Once there is a defined system state, a GP-Bayes filter is used to estimate the state with whatever components are still functioning properly, and then the system state is used to reproduce the missing or degraded component information.

6.2.2 Experimental Setup

An experiment was designed to test the viability of the GP-Bayes filter in the event of sensor failure for a 3DoF planar manipulator. Since the control systems for a robotic arm require accurate joint angle measurements to provide closed-loop control, when a sensor fails it causes undefined and often undesirable behavior. A GP-Bayes filter can be added to the control loop to provide estimates for the missing sensor data, allowing the arm to continue to function.

Figure 6.4 shows the 3DoF manipulator that was used, with 3 joints consisting of HEBI X-Series actuators. These actuators are self contained motor and sensor packages, each capable of working independently. All three actuators are connected via ethernet cables to a computer running the control software, receiving the sensor data, and relaying the control commands. Each actuator has three channels of sensor information: joint angle, angular velocity, and applied torque. Including data from all three actuators results in 9 channels of data representing a 9 dimensional data space. As can be seen from the figure, the robotic manipulator is restricted to a single plane of motion meaning there are more degrees of freedom than needed for simple point-to-point motion.

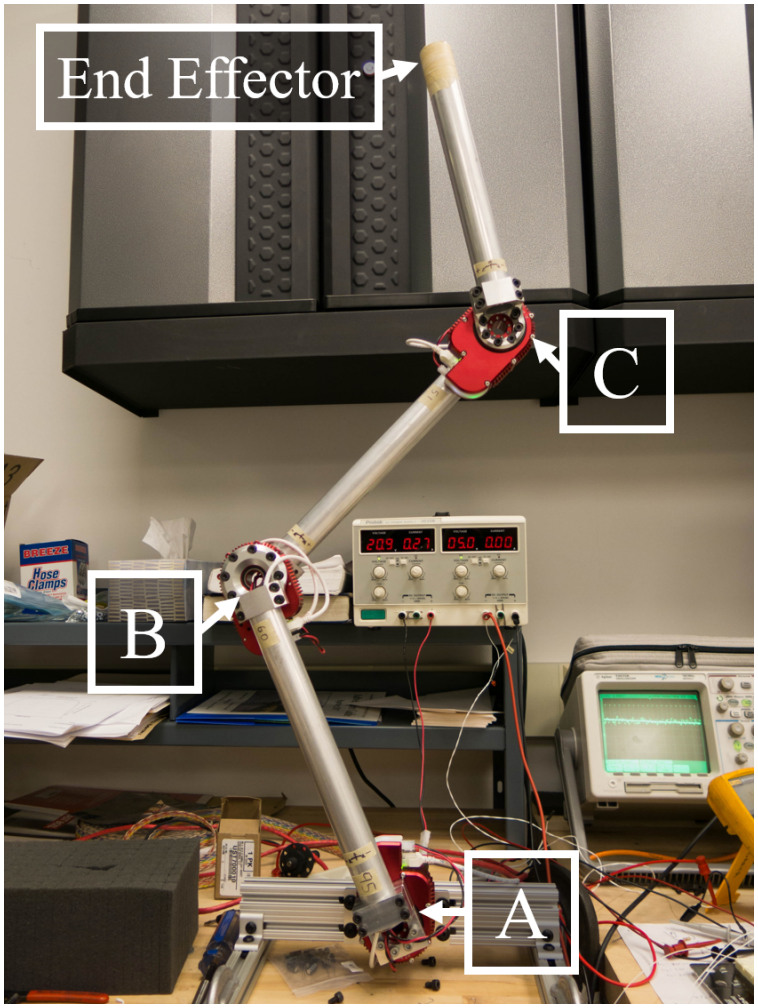

Figure 6.4: Photograph of the 3DoF robotic manipulator used to test the GP-Bayes filter

In order to apply the GP-Bayes filter to the robotic manipulator, there needed to be a well defined task that the GP-LVM necessary for the filter could be trained on. The task was to move the end effector of the arm along a straight line, then change directions and move to a 3rd final point. As the robot was under-constrained due the 3 degrees of freedom in the plane, an extra constraint of minimizing total joint angle was added.

Once training data was collected, a baseline for what sensor failure produces in the system was also tested. Figure 6.5 shows the end effector trajectory along the designated path, as

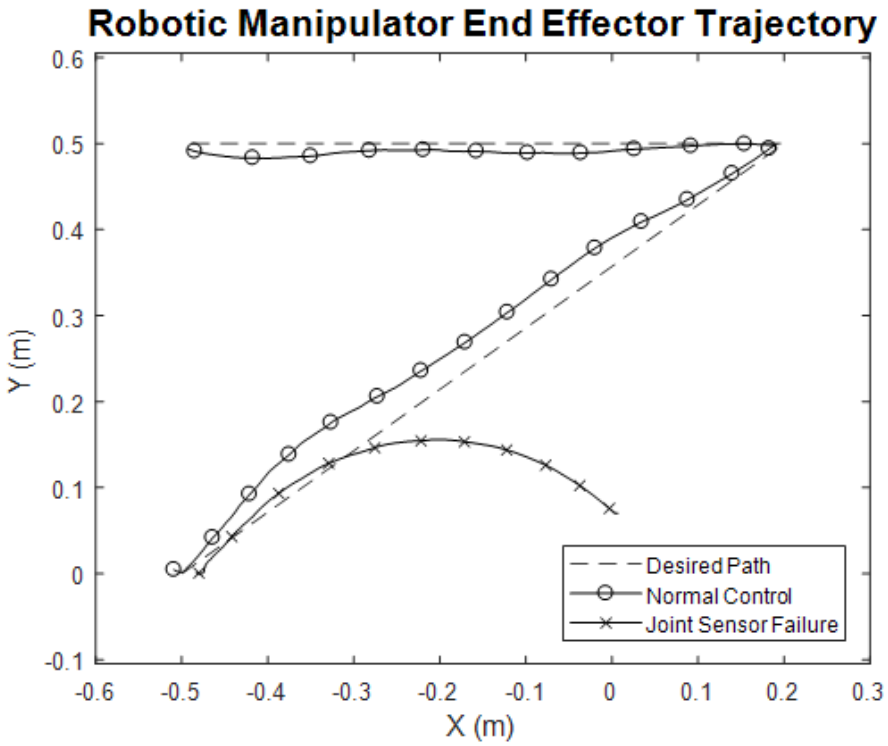

Figure 6.5: Spatial plots showing the trajectory of the end effector position for both training data and failure test case

well as the case when joint A's angle sensor experienced a simulated failure. With no feedback from the joint angle sensor, the motor continues to drive the joint well past the desired angle and the end effector diverges from the intended path. Exactly what happens to the trajectory of the end effector depends on the details of how the angle sensor fails, leaving the behavior undefined when the failure mode of the sensor is unknown.

6.2.3 Usage of GP-Bayes Filter

To use a GP-Bayes filter for the described system, a GP-LVM is used to create a hidden state that will be tracked and used to recreate the missing sensor information. While the resulting planar position of the end effector could potentially be used as a hidden state,

the extra degrees of freedom mean that there are multiple possible joint angles for a given position. Constraining this with the minimal total joint angle resolves this to some degree, but as this is a soft constraint it does not completely alleviate the problem. Using a GP-LVM allows for a hidden state that is better suited to the particulars of the robot's motion.

The GP-LVM optimizes the latent space for three dimensions, since the robot has three degrees of freedom. To ensure that the latent space dynamics Gaussian process is learned properly, the dynamics kernel as mentioned in Wang and Levine is included in the optimization process (shown below). The resulting latent space variables and Gaussian processes are then able to be used for the GP-Bayes filter.

$$\vec{x}_i = \vec{k}_{i-1}^T \mathbf{K}_X \mathbf{X}_{2:N} \tag{6.8}$$

Figure 6.6 shows how the GP-Bayes filter is used to track the state of a system, using available sensor data to improve estimates of the system state. First the system state for the current time-step is predicted based on the most recent estimate. The system state prediction is used to predict what measurements should be observed, which are then compared to the actual observations. Since not all sensor data is valid, only available measurements are compared. Based on the relative errors in the prediction compared to measurements, the system state is then adjusted to better fit observation. Once the final system state estimate is calculated, the measurements for the missing sensors can be predicted from this final latent variable state estimate.

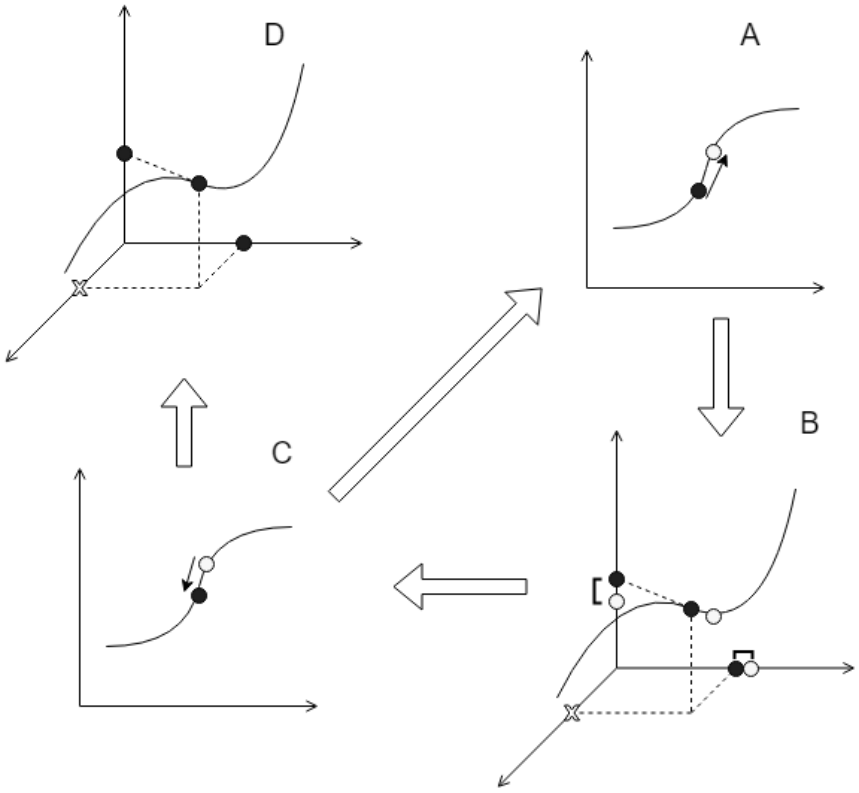

Figure 6.6: Illustration of GP-Bayes filter and sensor reconstruction. A) New latent coordinates (white) predicted based on current (black). B) Observed measurements (black) compared to predicted (white). C) Latent coordinates (white) adjusted to fit measurements resulting in final estimate (black). D) Missing sensor data (X) is estimated based on predictions (black)

6.2.4 Results

To more accurately assess the performance of the GP-Bayes filter in extending the use of a system beyond individual component failure, it needs to be compared to a baseline approach for handling sensor failure. The most straightforward solution is to simply playback the sensor information, ignoring any input from available sensors. The recorded motion should be relatively close to the original intended task, as long as the system and the external environment hasn't change significantly. While it may seem that performance would be better if only the missing

sensor data is replayed, in practice it is difficult to synchronize the playback with the other sensors and results in worse performance compared to playing back all the sensor data at once.

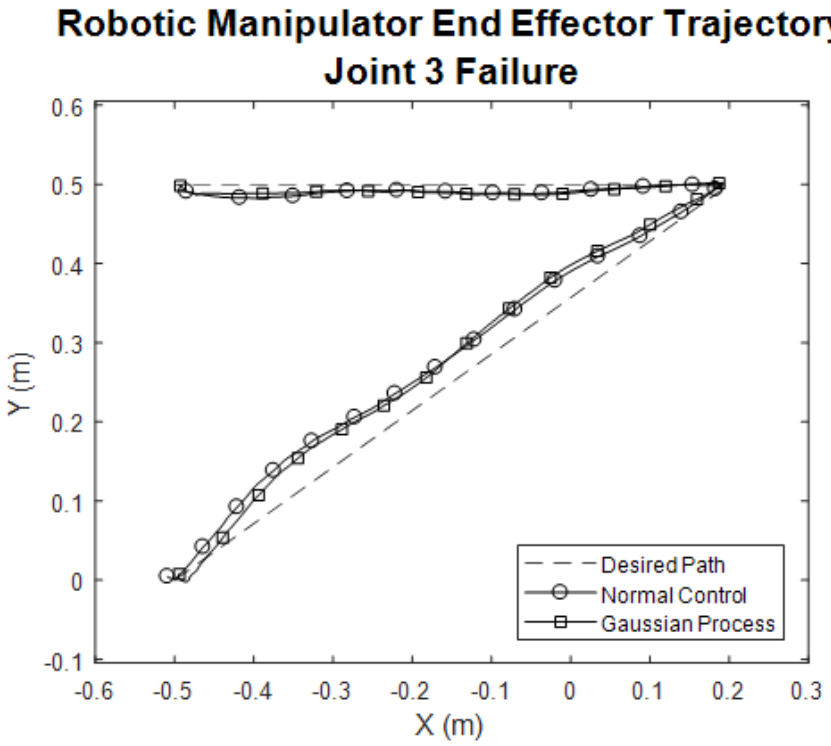

Figure 6.7: End Effector trajectory normal operation vs GP-Bayes filter control

Under conditions identical to the training data, the replay approach works almost perfectly. Figure 6.7 shows the GP-Bayes filter sensor replacement compared to the training data, demonstrating the viability of the method under ideal conditions. What is more interesting is to look at cases where conditions have changed somewhat and the control loop needs accurate sensor information to compensate. To test the robustness of the GP-Bayes filter, two separate cases involving changes to the motor speeds (simulating a load or change in power supply) and an obstacle in the path (external differences) are examined as well.

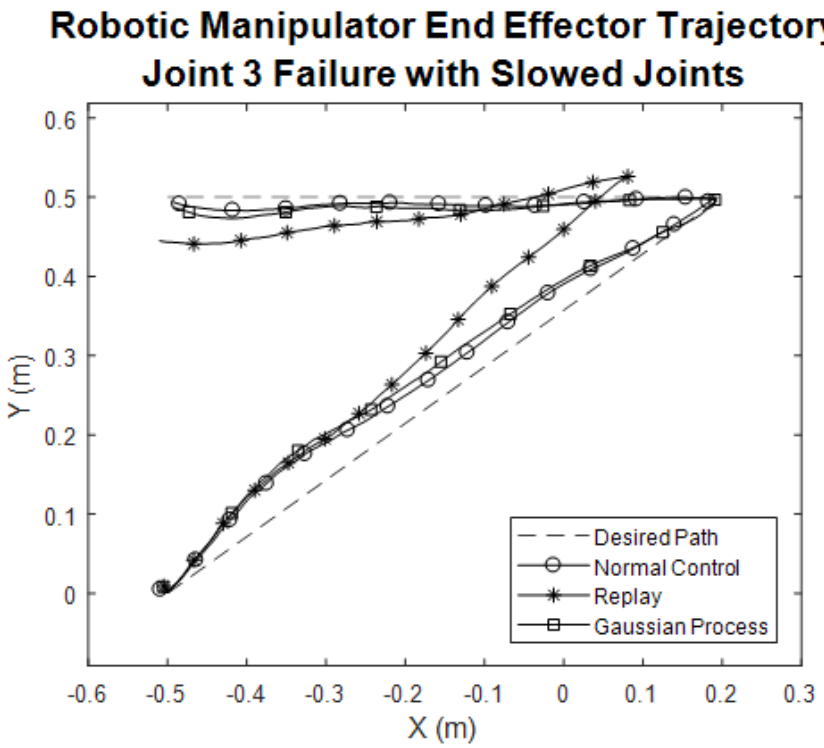

Figure 6.8: End Effector motion for the frictional case. Normal control represent the training data with all sensors working properly.

To test the usefulness of the GP-Bayes filter in cases where the system behavior has changed, the joint motors were slowed down to represent the manipulator carrying a load or running on low power. This was simulated by taking any motor velocity commands and reducing the commanded velocity before sending to the actuators themselves. Figure 6.8 shows the end effector trajectories that result from this change. Note that the replay trajectory diverges quite quickly, and only recovers somewhat due to the change in direction at the corner. The GP-Bayes filter, however, results in motion still very similar to the original.

Figure 6.9 helps quantify the performance of the two approaches. The error is computed by measuring the distance to the normal path at each point. The error for the replay method drops to zero only as it crosses the intended path, but remains well above

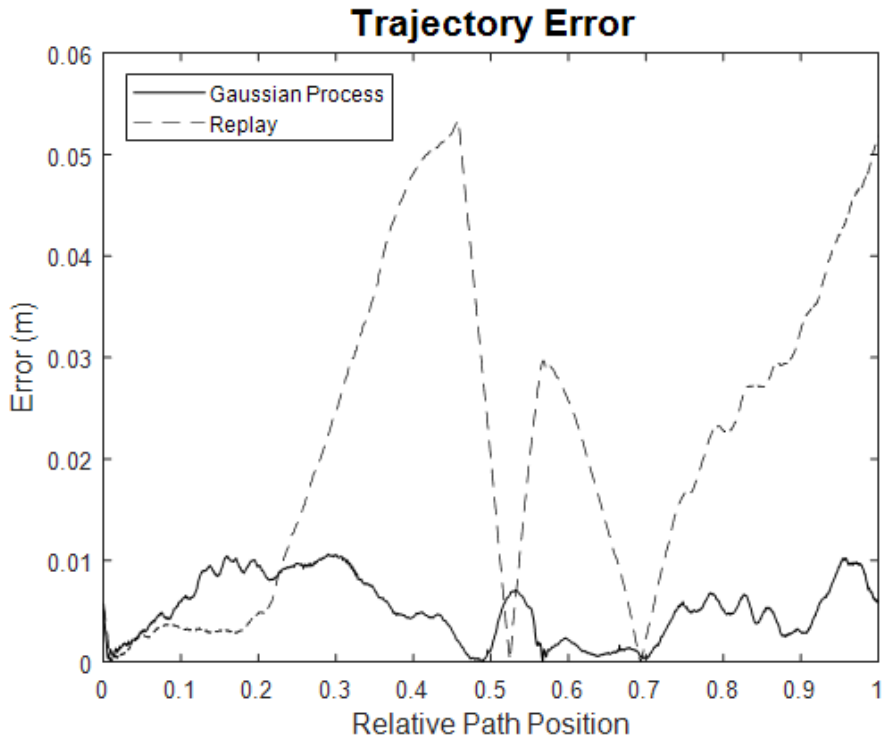

Figure 6.9: End Effector positional error for the slowed joint velocity case. Distance is measured to the closest point on the normal path.

the GP-Bayes filter error for the majority of the trajectory. This demonstrates the ability of the GP-Bayes filter to use the remaining sensor information to compensate for missing data.

Introducing an obstacle along the path demonstrates the ability of the GP-Bayes filter to correct for external disruptions while missing information necessary to recover. Figure 6.10 shows the case where an obstacle is introduced that deflects the end effector to the side. The trajectory replay makes no attempt to compensate and instead continues on a now offset path. The GP-Bayes filter manages to recover, and although it overshoots on the corner it is able to compensate and winds up much closer.

Plotting the error shown in Figure 6.11 tells much the same story as the trajectory

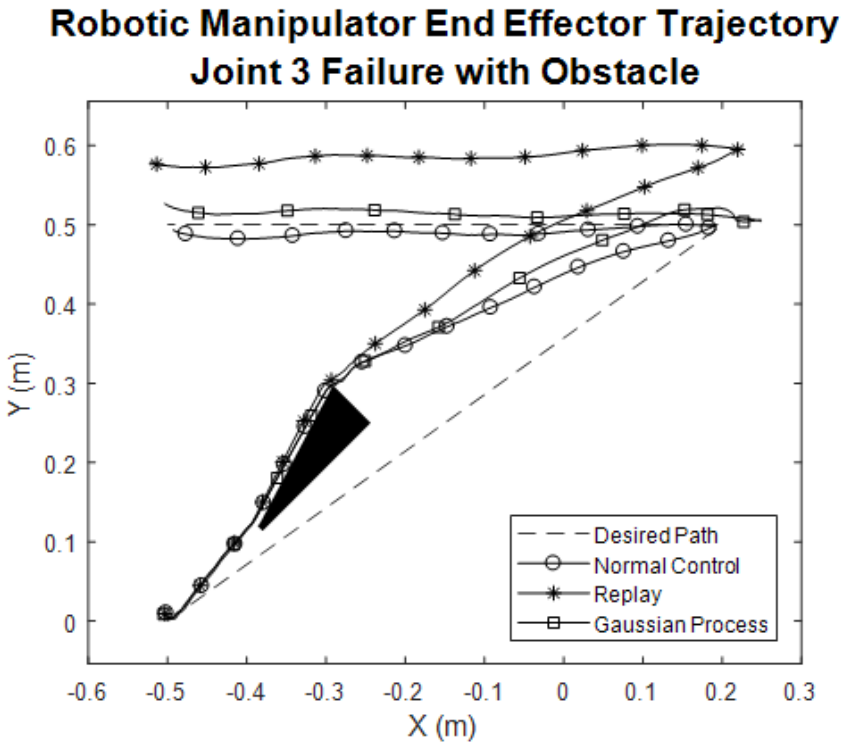

Figure 6.10: End Effector motion for the case with an obstacle present. Normal control represent the training data with all sensors working properly.

plot. The error for the GP-Bayes filter overshoots briefly before recovering, while the replay approach never recovers. An important note is that the joint with the simulated failure is Joint C as shown in figure 6.4 which is deflected more than the other joints, making this the case with the worst error.

6.2.5 Observations

The results of this experiment show that a GP-Bayes filter works to solve some of the original problems with regards to component failure in complex systems. While the tests performed focused on single component failure for a simple task, it could easily be extended to multiple component failures across several tasks, perhaps even composing simple tasks together through a library of trained actions. While the uncertainty information from the

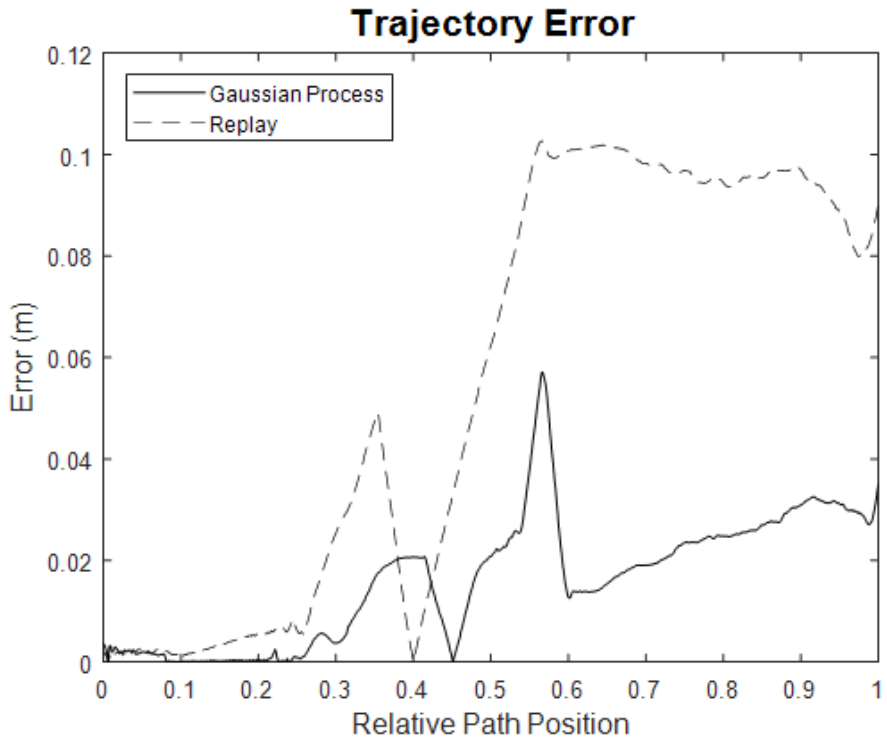

Figure 6.11: End Effector positional error for the case with an obstacle present. Distance is measured to the closest point on the normal path.

Gaussian processes was used for the GP-Bayes filter, it could also be used within the control system to give a measure of how close the system is to a known state, giving more assurances with regards to the accuracy of recovered information.

It would also be possible to simply use the GP-Bayes filter within a system to cause an alert if it sees inconsistent behavior such as that caused by component failure. When comparing observed measurements to the predictions made by the GP-Bayes filter, large deviations from the predictions would indicate inconsistent behavior. In this way the GP-Bayes filter acts as a safeguard that detects component degradation or failure rather than a part of the control loop. This might be more desirable in some cases where precision is important or other concerns prevent using machine learning tools within the control loop.

6.3 GP Manifold: Metric Tensor and Geodesics

One of the more practical use cases of manifold learning applied to a system is the ability to perform path planning on the manifold using geodesics. The constraints of the desired system behavior are contained within the manifold representation, so any trajectory that lies on the manifold will satisfy the constraints associated with the desired behavior. A geodesic is the shortest trajectory connecting two points on the manifold, and is therefore the optimal path for the system.

6.3.1 Havoutis and Geodesic Path Planning

The idea for geodesic path planning is presented in Havoutis [15]. Havoutis seeks to improve the domain of motion planning for robotic systems that use training data to learn skills or behaviors. To improve the generality and flexibility of these learned skills, a "skill manifold" is defined as a low dimensional representation of the training data that specifies the structure and dynamics of the skill. Once the manifold is known, geodesic trajectories on the manifold correspond to execution of the desired skill. Havoutis uses Locally Smooth Manifold Learning [8], an approach that seeks to find the tangent space of manifold rather a coordinate mapping. This approach does not need to generate a latent space, instead associating a set of basis vectors for the tangent space at every point. This can be used to traverse the manifold by restricting motion to this tangent space, and can even be used to extrapolate the manifold

based on smoothness assumptions. This approach has many of the same downsides as most deterministic manifold learning methods as discussed in Section 5.1.3.

In Havoutis' work, the learned tangent space is used to produce geodesics on the manifold. Trajectories are generated by creating a segmented line from the start and end points of the desired motion and then mapping the points on the line to the manifold while also minimizing the length of the path. To make sure points on the line lie on the manifold, the points are iteratively adjusted to minimize the error between the direction of the line and the projection of the line to the tangent space of the manifold.

To demonstrate this method, Havoutis first uses a 3-link manipulator operation in a 2-dimensional plane. The skill manifold that Havoutis has chosen to learn is motion that minimizes the pose strain, or distance from a rest position. His results show very low error when interpolating on the manifold and moderately higher error when extrapolating trajectories off the learned portion of the manifold, but with shorter planning time than is required to compute the paths using redundancy resolution. Havoutis then performs a similar demonstration using a small humanoid robot, where the skill manifold is a walking motion. The constraints of the motion are defined using a cost function, and training trajectories are generated by initializing a simple path and optimizing it according to the cost function. Once the skill manifold was learned, new trajectories could be generated as before. Results showed that although the error of the geodesic trajectories compared to the cost function was relatively high due to the manifold's smoothness assumptions, the generated trajectory still resulted in stable walking motion. The time needed to plan the motion using the skill manifold was a significant decrease from the trajectories planned using the cost function, from almost 2 minutes down to 2 seconds.

This could allow for greater flexibility in adjusting trajectories in response to new information.

One problem with the approach used by Havoutis is that the manifold learning method has the same problems as other deterministic methods that were described in section 5.1.3. The approach works well with the largely flat manifolds it was tested on, but will begin to suffer when considering more complex manifolds due to the smoothness assumptions it makes. This was noticeable in the test case of the walking motion where the constraint error was significantly higher. It also suffers when a manifold has gaps that are part of the structure rather than missing data, as it will try to form paths through those gaps as if they were flat.

Another problem is the way that geodesics are computed. The method described by Havoutis produces only a locally optimal path, where the initial trajectory is the Euclidean shortest path. It can be shown that there exist cases where the geodesic found through Havoutis' method is not the globally shortest path, and due to the way the manifold is learned there is no coordinate space and therefore no way to adapt the manifold to solve the geodesic differential equations. When adding obstacles [16], Havoutis' method no longer produces geodesics and relies on a naive method to route around obstacles that does not work in all cases.

6.3.2 GP-Manifold Geodesics

While the principles of skill manifolds and geodesics for path planning introduced by Havoutis are still used, the way in which they are used by a Gaussian process differs significantly.

Instead of just learning the tangent space of a manifold, the coordinate mapping is found through a GP-LVM. The major downside to a tangent space approach is that it doesn't allow answering the question of whether or not a particular point belongs to the manifold or not. Havoutis demonstrates an approach for finding the projection of a point onto the manifold, and while it works in practice it relies on the existence of a point known to lie on the manifold in close proximity. Learning the coordinate space of the manifold through a GP-LVM eliminates this issue.

The second major way that the GP-Manifold approach differs is in the computation of geodesics. Havoutis uses an iterative method that gradually fits the straight line approach onto the manifold through use of the tangent space to ensure points only move along the manifold itself. When using a GP-Manifold, geodesics can instead be found through solutions to their governing differential equations as mentioned in section 5.3. There are several advantages to this approach, with the biggest advantage being the application of existing differential equation solvers to improve solution time rather than relying on a custom iterative method. Using a differential equation solver such as MATLAB's built-in bvp4c also means that step size can be handled dynamically, an important necessity when considering manifolds with varying curvature that would need to be considered and implemented in Havoutis' original method.

In some cases, the differential equation solver may not converge nicely due to numerical instabilities. These can occur when the determinant of the metric tensor is large, something that can happen for a GP-Manifold when the trajectory endpoints lie far from the trained portion of the manifold. In these situations, the curve shortening flow approach mentioned

in section 5.3 can be implemented instead. This approach shares some commonality with the iterative method used by Havoutis. Both methods iterate until the path is shortened sufficiently, but they operate in different spaces. Havoutis works in the tangent space of the manifold, represented by a restriction on the ambient space that the manifold is embedded in. The curve shortening flows work in the coordinate space, using the metric tensor to inform the notion of distance. Working in the coordinate space guarantees the final result will lie on the manifold, while working in the tangent space can result in error pushing the final path away from the manifold if step size is not considered properly.

6.3.3 Obstacle Avoidance

One benefit that Havoutis presents in his approach is the ability to dynamically plan paths around obstacles not present in the original training data. The same can be accomplished in a superior way by using GP-Manifolds and the geodesic path planning. In the case of a known obstacle introduced after training, a region on the manifold is designated as blocked by the obstacle. The distance metric can be modified to penalize trajectories that enter this region, resulting in geodesics that will avoid the obstacle. This results in a natural way to introduce dynamic obstacle avoidance.

The metric tensor must be modified in such a way that it only affects the region near the obstacle to be avoided. This can be done by adding a component $f(p)$ to the diagonal elements of the metric tensor that is a function of the position on the manifold, which is

large in the avoidance region and zero elsewhere. In order to be able to solve the geodesic differential equations, however, the metric tensor needs to be differentiable at all points, imposing additional constraints on the function added to the metric tensor. The derivative $f'(p)$ must also be zero along the boundary of the avoidance region to ensure that the derivative of the metric tensor has no discontinuities that would prevent finding solutions to the geodesic differential equations.

One such function that satisfies these requirements is the bump function, defined as:

$$f(r) = \begin{cases} \exp(-\frac{1}{1-(\frac{r}{R})^2}) & \text{for } r < R \\ 0 & \text{otherwise} \end{cases} \tag{6.9}$$

Here R is the radius of the avoidance region, and r is the distance from the center of the region in coordinate space. Note this assumes local linearity around the obstacle due to the presence of a Euclidean distance term that does not depend on the metric tensor. This makes the size of the avoidance region somewhat dependent on the curvature of the manifold near the obstacle. While this can be corrected for by implementing a more complex bump function that accounts for the manifold's geometry, adding this complexity drastically increases the time needed to compute the geodesic differential equations that incorporate this bump function.

6.3.4 3 Link Test Results

To test the performance of the GP-Manifold based geodesics, a similar test case to the one described in Havoutis was used. A 3-link planar manipulator simulation was used to produce training points for a region of a flat plane, with a constraint enforcing minimal joint angles. Each training point in the plane was recorded as a triple of the joint angles corresponding to the solution of the robotic manipulator at that point. A grid of points was produced, and then a random sample of those points was used to train the GP-Manifold. In Havoutis, extra information regarding the distance between points was added due to the method used to learn the tangent space; this is not needed for training the GP-Manifold.

Once trained, the capabilities of the geodesic path planning method were tested by generating start-end point pairs and computing the shortest path between them. The time needed to solve the geodesic differential equations was recorded and once the path was found, the constraint error at each point could be measured. The GP-Manifold predicted the joint angles at each point along the path, and the constraint function used to produce the original training points could be applied to the resulting point to find how much the geodesic trajectory point differed from the correct value.

Figure 6.12 shows the GP-Manifold in the joint angle space of the 3 link manipulator and the corresponding end effector positions. The black lines are the trajectories planned by the geodesic path planner, according to the constraint of the manifold. The original training points followed a constraint of minimal sum of squared joint angles, so the trajectories should

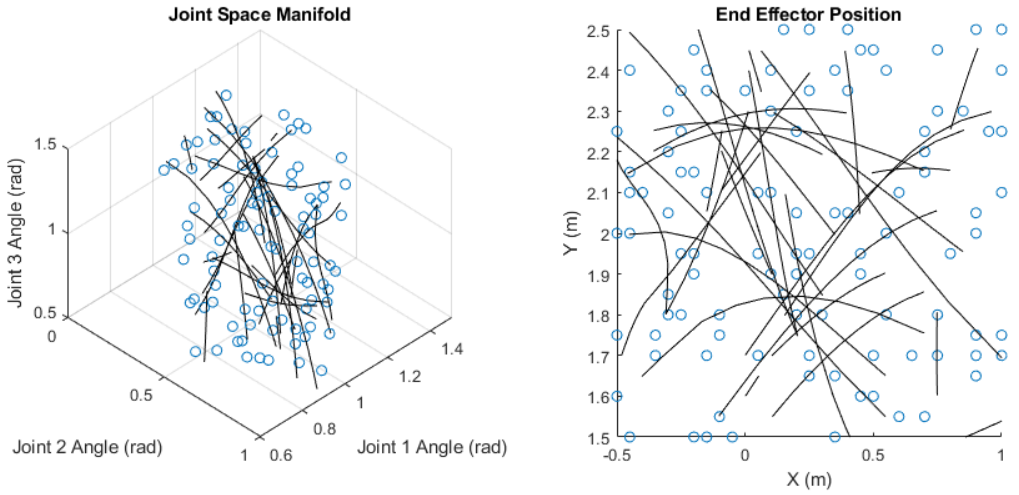

Figure 6.12: Plot showing the joint space manifold and end effector positions for the 3 link planar manipulator with the geodesic paths. Circles represent the training points, lines are generated geodesics Left is the joint space manifold, right is the end effector position.

also meet this constraint at each point along the path.

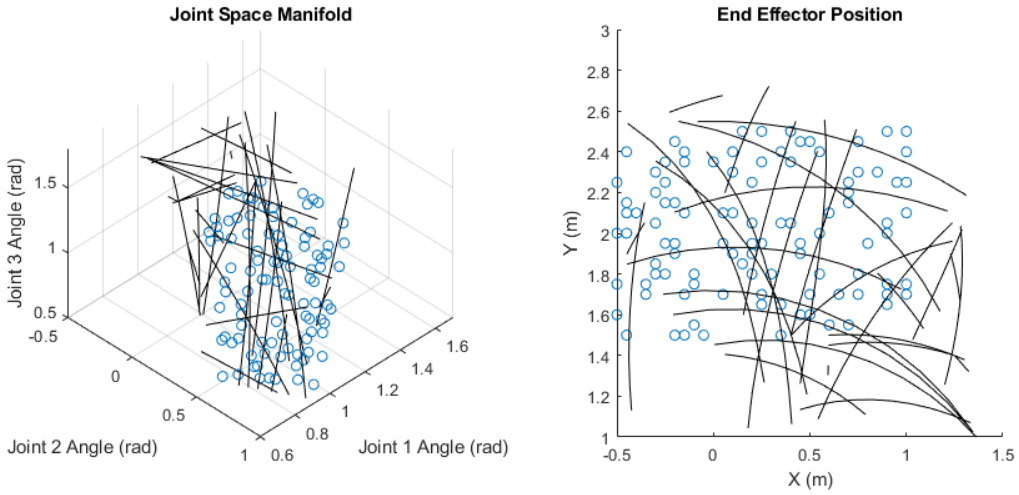

Figure 6.13: Plot showing the joint space manifold and end effector positions for the 3 link planar manipulator with the geodesic paths. Left is the joint space manifold, right is the end effector position.

Figure 6.13 shows similar results to Figure 6.12 but for trajectories outside the training data. When trajectories must be computed for points outside the training data, the large increase in the distance metric due to the high uncertainty can result in numerical instabilities

that cause problems when solving the geodesic differential equations. It is possible to omit the term from the metric tensor corresponding to the uncertainty, and solve the differential equations while ignoring the stochastic nature. This was necessary to produce some of the paths shown in Figure 6.13.

The RMSE is computed at each point along each trajectory, resulting in an RMSE of $1.15 \times 10^{-4} \pm 1.06 \times 10^{-4}$ for trajectories within the training region and $2.00 \times 10^{-3} \pm 2.40 \times 10^{-3}$ for points outside of it. This is compared to Havoutis' RMSE for a similar region of $1.89 \times 10^{-4} \pm 3.60 \times 10^{-5}$ for trajectories within the training data and $6.84 \times 10^{-2} \pm 2.19 \times 10^{-2}$ outside of it. The GP-Manifold shows slightly better performance inside the training region, and much better outside of it.

6.3.5 Robonaut Example Learning

Gaussian process skill manifolds can be used to learn by demonstration from example data without explicitly knowing the constraints. To demonstrate the ability to learn by demonstration, raw data from the Robonaut experiments[1] is used to train the GP-manifold. Once the manifold is learned, new trajectories can be generated that mimic the original behavior from the Robonaut experiments.

The original Robonaut data comes from experiments where the motion of one of the Robonaut arms was manually moved to pick up an object and drop it in a container and

Figure 6.14: The NASA Robonaut platform. Photo credit: Dr. Alan Peters

then back to the starting position. All of the sensor data for multiple trials from the original experiments was available, divided into two categories for success and failure. Successful examples are where the object was picked up and placed properly, and failed examples are where the object was not picked up properly or was dropped prematurely. For training the GP-manifold, only the 5 successful examples were used for learning.

For the purposes of generating new trajectories, the useful data are the 7 joint angles for the arm. Since the behavior that is being learned is a motion in 3D space, the GP-manifold approximation is a 3 dimensional one. To generate trajectories, the starting point on the manifold is taken from one of the successful trajectories while the end point is selected from a region where all 5 trajectories would have reached the object to be picked up. The geodesic differential equations from (5.16) were then solved with the endpoint boundary conditions to

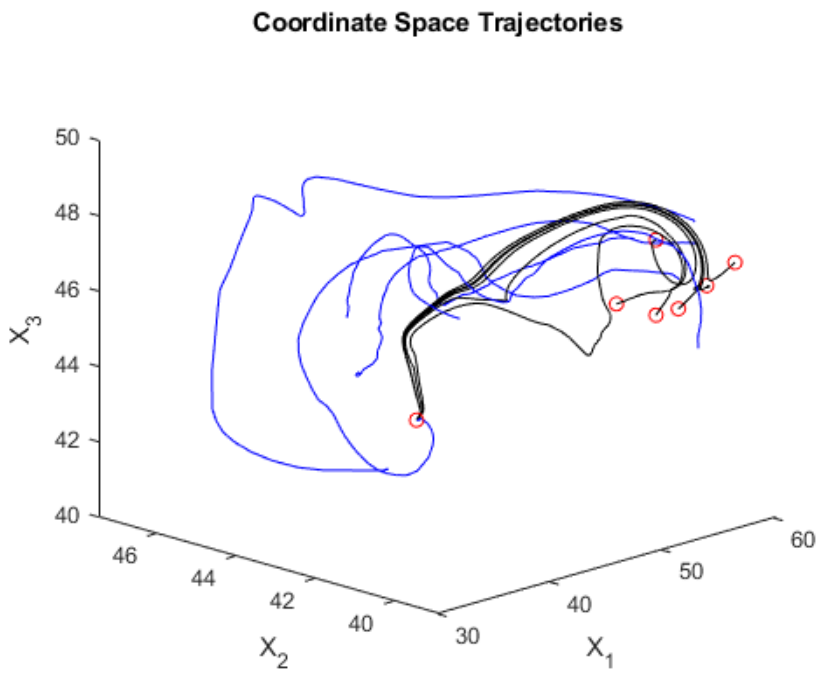

Figure 6.15: Plot showing the manifold coordinate space geodesics. Training data is blue, start and end points are in red, geodesics are in black.

get the final trajectory.

Figure 6.15 shows the coordinate space of the manifold. The geodesics are shown in black, and the training data is shown in blue. Note that these trajectories are in the coordinate space; the manifold points are the 7 dimensional joint angles and so it is difficult to visualize those directly. Instead, it is possible to use the joint angles to compute the forward kinematics and produce a visualization of the trajectory of the end effector.

One difficulty of learning by demonstration is that there are not quantifiable metrics for success beyond a binary success or failure of the task. Instead, the performance of the method must be judged subjectively by observing successful examples and qualitatively describing

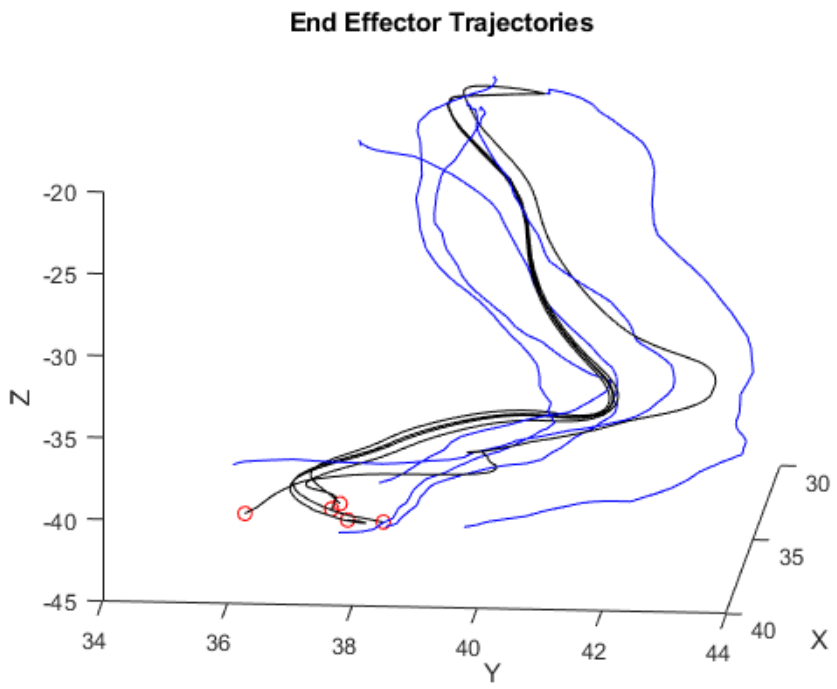

Figure 6.16: Plot showing the manifold coordinate space geodesics. Training data is blue, start and end points are in red, geodesics are in black.

how well the new trajectories match the original demonstration behavior.

Figure 6.16 shows the trajectory of the end effector for both the training paths in blue and the generated geodesics in black. Note that the generated trajectories follow the shape of the training paths rather than cutting across straight to the endpoint. This is confirmation that the geodesics are remaining on the manifold and staying close to the training data.

6.3.6 Tiago Skill Manifold

As further demonstration of the capability of geodesic trajectory planning on GP-Manifolds, tests were done using a ROS simulation of the TIAGo robot. TIAGo is a mobile robot

produced by PAL Robotics with a 7DoF manipulator that can be equipped with several different end effectors.

Figure 6.17: TIAGo robot, produced by PAL Robotics.

The 7DoF manipulator on the TIAGo robot is a good example for the utility of geodesic path planning. The task under consideration is a constrained motion, where the end effector remains in perpendicular contact with a surface. This might be the case when the robot is used to write on a surface. The constraints for this type of motion are relatively simple when expressed in Cartesian space, but become more complex when converting them into joint angles and velocities. These constraints in joint space can be represented by a GP-Manifold, with geodesics representing the shortest paths that satisfy the constrained motion.

To produce the training data that the GP-Manifold is built from, a set of points covering the constraint surface in Cartesian space along with the normal vector at each point are generated. The inverse kinematics are then solved through ROS to obtain the joint space configuration of the robot for each point. These joint space configurations are then the points on the GP-Manifold, and the manifold coordinates can either be taken from the Cartesian space points or learned through a GPLVM. Since the surface in Cartesian space is a 2-dimensional shape, a 2-dimensional coordinate space works well.

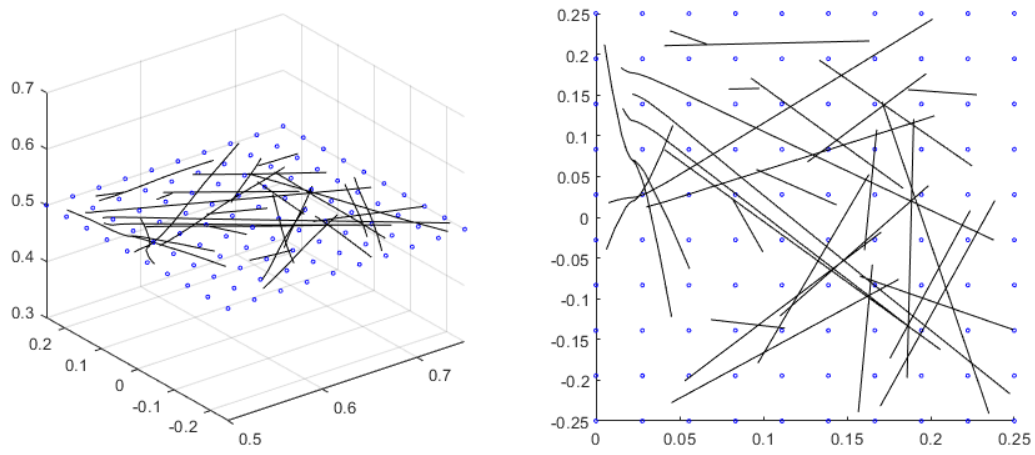

Figure 6.18: End effector(left) and coordinate(right) trajectories. Trajectories are shown in black, training points are in blue.

Figure 6.18 shows the end effector trajectories along with their corresponding manifold coordinates. In this figure, the trajectories shown are geodesics for point-to-point motion. More complex trajectories can be planned by simply drawing any path in the coordinate space. These coordinate trajectories are mapped through the GP-Manifold to produce the series of joint angles for the robot to follow. The trajectories can then be evaluated in terms

of how well they meet the original constraints for the motion in terms of their deviation from the intended surface. For this set of 30 geodesics, the RMSE is $5.244 \times 10^{-2} \pm 2.682 \times 10^{-2}$ cm when looking at the deviation from the plane of motion.

An important concern when talking about constrained motion is the time it takes to produce a solution. Solutions already exist in the form of inverse kinematics, so it is important to compare the time needed. For these geodesics, the solution time averages 0.06 seconds. Using inverse kinematics to solve these trajectories for the joint configuration averages 0.2 seconds, almost four times as long. It is important to note that this is only once a trajectory in Cartesian space has been found, while the geodesic solution incorporates this additional step. This is a noticeable speed benefit that makes the GP-Manifold viable despite the existence of inverse kinematic methods to find solutions.

Chapter 7

Conclusion

Manifold learning is a branch of structured machine learning that can provide a compact, tangible representation of a complex, high dimensional system. A manifold can be derived from a data set that comprises the inputs and outputs of a high dimensional system. The result is a closed lower-dimensional subset of the system's state space – a manifold – that not only represents the training data but also can be used to control the system in novel situations. Probabilistic manifolds can be learned as Gaussian Processes that enable extrapolation to inputs that do not conform precisely to the learned manifold thereby expanding their use to systems with noisy and uncertain inputs.

7.1 Contributions

This dissertation has covered the fundamentals of Gaussian processes in Section 3, a probabilistic approach to functional regression that has been used for dimensionality reduction in the form of GP-LVMs. Section 4 covered the use of Gaussian processes in GP-Bayes filters to model and track complex systems, ultimately showing that the assumptions needed for a GP-Bayes filter imply the existence of a manifold. Section 5 defined and described GP-manifolds as a probabilistic approach to manifold learning and developed new methods

for working with a stochastic manifold. Section 6 showed the application of these topics to practical situations.

The existing work in this area was covered in Section 1.2. Manifold learning in literature falls into two main categories. The first is where manifold structure is used to solve problems, but the manifold itself is already known. The second is where the manifold is learned from a set of data, but in these cases the manifold structure is not used to solve problems beyond simple regression. Combining these two categories into cases where the manifold is both learned and then used for more complex problems such as path planning is demonstrated in Havoutis [15],[16]. The weaknesses of that particular approach lie in the incomplete nature of the manifold learned and the naive approach to finding geodesics. To successfully combine both the learning and usage of manifold structure requires that a new approach to both be taken. Hauberg [14] demonstrates the necessity of stochastic manifolds if the structure of a learned manifold is to be used properly.

This dissertation addresses the need of a stochastic manifold by developing a framework for GP-manifolds. Section 5 contains the contributions to manifold learning through the definition of Gaussian process manifolds. The derivations for working with stochastic manifolds in the particular case of a Gaussian process manifold are also shown here. Section 5.2 explains how the metric tensor differs in the case of a stochastic manifold and then derives the expected value of the metric tensor in equation (5.10). The code showing how this is implemented is shown in Appendix 8.2.1. Section 5.3 gives the definition of geodesics through the geodesic differential equations. The Christoffel symbols needed for these equations are derived from the expected value of the metric tensor in equation (5.29). The code implementing these deriva-

tions is shown in Appendix 8.2.2, and the geodesic equations and solutions are found through the code shown in Appendix 8.2.3. Section 5.4 explains the importance of the curvature tensor in regards to geometry on manifolds. The curvature tensor for a GP-manifold is derived in equation (5.31), with the code implementing these equations shown in Appendix 8.2.6.

The benefits of using GP-manifolds can best be understood by their application to practical problems. Section 6 covers these applications and how they improve upon existing work. Section 6.1 demonstrates a novel method of GP-LVM compression through re-sampling. Section 6.2 shows the application of a GP-Bayes filter to a problem involving sensor failure. Section 6.3 covers multiple experiments where path planning for robotic systems is performed using geodesics on learned manifolds. The first experiment in Section 6.3.4 mirrors the experimental setup in Havoutis [15], and demonstrates superior accuracy when performing geodesic path planning. The second experiment in Section 6.3.5 demonstrates the ability to take data from manual operation and use it to learn a manifold and generate new trajectories that mimic the original operation. The final experiment in Section 6.3.6 takes a more complex robotic manipulator and shows significant speed improvement for constrained path planning by using geodesics on a learned manifold when compared to a method that explicitly satisfies the motion constraints.

7.2 Manifold Improvements

One of the ways that GP-Manifold learning could be improved is to find ways to improve the manifold learning portion. Looking at different kernel functions or additional likelihood terms could result in finding coordinates for the manifold that improve the quality of the mapping. One likelihood term that may be interesting to consider is one that is based on information regarding the curvature of the manifold, if that information is available.

Adding a curvature likelihood term may improve the preservation of manifold structure in the mapping that a GP-Manifold provides. To add this term, the curvature tensor or scalar would need to be described in terms of the coordinates. Next, the probability distribution of the curvature would need to be found in order to set up the likelihood term properly. One aspect of this approach that may present particular difficulty is the dependence of the metric tensor on the variance of the original Gaussian Process.

Other improvements could be made to the re-sampling methods, looking into other types of sampling that may give similar or better performance when compared to the Mitchell's best candidate method discussed in section 6.1.3. It might also be possible to perform some manner of sampling before the optimization process that could speed up the learning, although finding a way to measure distance without any manifold structure could prove problematic. Instead, the sampling methods discussed may have some use in training the GP-Manifold through batches.

7.3 Manifold Usage

While the contents of sections 6 shows some use cases for GP-Manifolds, particularly the use of geodesics on manifolds for path planning. There are still more use cases for manifolds that GP-Manifolds enable by allowing the use of the metric and curvature tensors. One more obvious use case is considering control systems on manifolds.

In the discussion on the curvature tensor in section 5.4, it was briefly mentioned that the interpretation of the curvature has implications in the case of commutativity of vector transport. In typical control systems the Lie bracket is used to describe the commutativity, allowing for new control strategies to be understood as the result of switching between existing ones in a particular order. When applying this to manifolds, the curvature tensor is needed to completely describe the commutativity in addition to the Lie bracket. GP-Manifolds allow for the curvature tensor to be used in this way, opening the possibility of exploring control systems restricted to manifolds.

Chapter 8

Appendix

8.1 Likelihoods and Gradients for Latent Variable Optimization

8.1.1 Probability of the Data

Given an N-sample vector time series $\{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$, of data let $\mathbf{Y}$ be the data matrix,

$$\mathbf{Y}_{N\times D} = \begin{bmatrix} \mathbf{y}_1 & \cdots & \mathbf{y}_N \end{bmatrix}^\mathsf{T} = \begin{bmatrix} \mathbf{y}_1^c & \cdots & \mathbf{y}_D^c \end{bmatrix} \text{ where } \mathbf{y}_m \in \mathbb{R}^D \text{ and } \mathbf{y}_\ell^c \in \mathbb{R}^N, \tag{8.1}$$

and let $\mathbf{X}$ be the matrix of the corresponding time series of latent vectors,

$$\mathbf{X}_{N\times d} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_N \end{bmatrix}^\mathsf{T} = \begin{bmatrix} \mathbf{x}_1^c & \cdots & \mathbf{x}_d^c \end{bmatrix} \text{ where } \mathbf{x}_m \in \mathbb{R}^d \text{ and } \mathbf{x}_\ell^c \in \mathbb{R}^N. \tag{8.2}$$

Define a weight matrix for the data vectors as

$$\mathbf{W} = \operatorname{diag}(\mathbf{w}), \text{ where } \mathbf{w} = \begin{bmatrix} w_1 \cdots & w_D \end{bmatrix}^\mathsf{T}, \text{ such that } w_\ell > 0, \tag{8.3}$$

and write the Gaussian process hyperparameters as the vector,

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix} \tag{8.4}$$

Then the conditional probability of the data given the latent variables, the hyperparameters, and the weights is

$$p(\mathbf{Y}|\,\mathbf{X}, \boldsymbol{\alpha}, \mathbf{W}) = \frac{|\mathbf{W}|^N}{\sqrt{(2\pi)^{ND}\,|\mathbf{K}_\mathbf{X}|^D}} \exp\left(-\frac{1}{2}\,\mathrm{trace}\left(\mathbf{K}_\mathbf{X}^{-1}\mathbf{Y}\mathbf{W}^2\mathbf{Y}^\intercal\right)\right), \tag{8.5}$$

where the kernel matrix

$$\mathbf{K}_\mathbf{X} = \left[k\left(\mathbf{x}_m, \mathbf{x}_n\right)\right]_{N\times N} \tag{8.6}$$

is a radial basis function matrix with elements,

$$k\left(\mathbf{x}_m, \mathbf{x}_n\right) = \alpha_1 \exp\left(-\frac{\alpha_2}{2}\left\|\mathbf{x}_m - \mathbf{x}_n\right\|^2\right) + \frac{1}{\alpha_3}\delta_{m,n}, \tag{8.7}$$

for latent vectors $\mathbf{x}_m$ and $\mathbf{x}_n$ from the time series with

$$\delta_{m,n} = \begin{cases} 1 & \text{for} \quad m = n \\ \\ 0 & \text{for} \quad m \neq n \end{cases}. \tag{8.8}$$

8.1.2 Log Likelihood of the Data

The log likelihood of the conditional distribution (8.5) is the scalar value,

$$\mathcal{L}_{\mathbf{Y}} = N \log |\mathbf{W}| - \tfrac{ND}{2} \log (2\pi) - \tfrac{D}{2} \log |\mathbf{K}_{\mathbf{X}}| - \tfrac{1}{2} \operatorname{trace} \left(\mathbf{K}_{\mathbf{X}}^{-1} \mathbf{Y} \mathbf{W}^2 \mathbf{Y}^{\mathsf{T}} \right). \tag{8.9}$$

This can be simplified for computation. The logarithm of the determinant of a diagonal matrix is

$$\log (|\mathbf{W}|) = \log \left(\prod_{\ell=1}^{D} w_\ell \right) = \sum_{\ell=1}^{D} \log (w_\ell). \tag{8.10}$$

RBF kernel $\mathbf{K}_{\mathbf{X}}$ is symmetric positive definite and therefore has a Cholesky decomposition:

$$\mathbf{K}_{\mathbf{X}} = \mathbf{L} \overline{\mathbf{L}}^{\mathsf{T}}, \tag{8.11}$$

where $\mathbf{L}$ is lower triangular. Therefore, the determinant of $\mathbf{K}_{\mathbf{X}}$ is

$$|\mathbf{K}_{\mathbf{X}}| = |\mathbf{L}| \left| \overline{\mathbf{L}}^{\mathsf{T}} \right| = \prod_{n=1}^{N} \operatorname{diag} [\mathbf{L}] \prod_{n=1}^{N} \operatorname{diag} \left[\overline{\mathbf{L}}^{\mathsf{T}} \right] = \left(\prod_{n=1}^{N} \operatorname{diag} [\mathbf{L}] \right)^2, \tag{8.12}$$

and its logarithm is

$$\log (|\mathbf{K}_{\mathbf{X}}|) = 2 \sum_{n=1}^{N} \log (\operatorname{diag} [\mathbf{L}]). \tag{8.13}$$

The last term of (8.9) can be simplified since $\mathbf{W}$ is a $D \times D$ diagonal matrix:

$$\mathbf{Y} \mathbf{W}^2 \mathbf{Y}^{\mathsf{T}} = \mathbf{Y} \mathbf{W} [\mathbf{Y} \mathbf{W}]^{\mathsf{T}} = \begin{bmatrix} w_1 \mathbf{y}_1^c & \cdots & w_D \mathbf{y}_D^c \end{bmatrix} \begin{bmatrix} w_1 \mathbf{y}_1^c & \cdots & w_D \mathbf{y}_D^c \end{bmatrix}^{\mathsf{T}}, \tag{8.14}$$

where $\mathbf{y}_\ell^c$ is the ℓ^{th} column of $\mathbf{Y}$ – the ℓ^{th} data stream for times 1 to N. Since the trace is linear, and since $\text{trace}(AB) = \text{trace}(BA)$ whenever A is $m \times n$ and B is $n \times m$,

$$\begin{aligned}
\text{trace}\left(\mathbf{K}_\mathbf{X}^{-1}\mathbf{Y}\mathbf{W}^2\mathbf{Y}^\mathsf{T}\right) &= \text{trace}\left[\mathbf{K}_\mathbf{X}^{-1}\sum_{\ell=1}^{D} w_\ell^2 \mathbf{y}_\ell^c \mathbf{y}_\ell^{c\mathsf{T}}\right] && (8.15)\\
&= \text{trace}\left[\sum_{\ell=1}^{D} w_\ell^2 \left[\mathbf{K}_\mathbf{X}^{-1}\mathbf{y}_\ell^c\right] \mathbf{y}_\ell^{c\mathsf{T}}\right] && (8.16)\\
&= \sum_{\ell=1}^{D} w_\ell^2 \,\text{trace}\left[\left[\mathbf{K}_\mathbf{X}^{-1}\mathbf{y}_\ell^c\right] \mathbf{y}_\ell^{c\mathsf{T}}\right] && (8.17)\\
&= \sum_{\ell=1}^{D} w_\ell^2 \,\text{trace}\left[\mathbf{y}_\ell^{c\mathsf{T}}\left[\mathbf{K}_\mathbf{X}^{-1}\mathbf{y}_\ell^c\right]\right] && (8.18)\\
&= \sum_{\ell=1}^{D} w_\ell^2 \mathbf{y}_\ell^{c\mathsf{T}}\left[\mathbf{K}_\mathbf{X}^{-1}\mathbf{y}_\ell^c\right]. && (8.19)
\end{aligned}$$

Then the log likelihood of the conditional distribution (8.5) can be written

$$\mathcal{L}_\mathbf{Y} = N\sum_{\ell=1}^{D}\log\left(w_\ell\right) - \tfrac{ND}{2}\log\left(2\pi\right) - D\sum_{i=1}^{N}\log\left(\text{diag}\left[\mathbf{L}\right]\right) - \tfrac{1}{2}\sum_{\ell=1}^{D} w_\ell^2 \mathbf{y}_\ell^{c\mathsf{T}}\left[\mathbf{K}_\mathbf{X}^{-1}\mathbf{y}_\ell^c\right]. \quad (8.20)$$

8.1.3 Gradient of the Log Likelihood of the Data

We need to find the values of $\mathbf{X}$, $\mathbf{W}$, and $\boldsymbol{\alpha}$ that will minimize the negative log likelihood. That requires the gradient of $\mathcal{L}_\mathbf{Y}$ with respect to $\mathbf{X}$, $\mathbf{W}$, and $\boldsymbol{\alpha}$. First consider the gradient with respect to $\mathbf{X}$. It is the Nd-dimensional row vector constructed via the column-wise catenation of the $N \times d$ matrix,

$$\nabla_\mathbf{X}\mathcal{L}_\mathbf{Y} = \text{vec}\left(\left[\begin{array}{ccc} \nabla_{\mathbf{X},1}\mathcal{L}_\mathbf{Y} & \cdots & \nabla_{\mathbf{X},d}\mathcal{L}_\mathbf{Y} \end{array}\right]\right)^\mathsf{T}, \quad (8.21)$$

where $[\nabla_{\mathbf{X},\ell}\mathcal{L}_{\mathbf{Y}}]^{\mathsf{T}}$ is the gradient along the ℓ^{th} dimension of $\mathbf{X}$. Each column of the matrix is the vector of partial derivatives $\partial\mathcal{L}_{\mathbf{Y}}/\partial x_{i,\ell}$ for $i = 1, \ldots, N$. Each of those can be found using the chain rule for matrix calculus[1]:

$$\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial x_{i,\ell}} = \text{trace}\left\{\left[\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial\mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}}\frac{\partial\mathbf{K}_{\mathbf{X}}}{\partial x_{i,\ell}}\right\} = \sum_{\text{cols}}\sum_{\text{rows}}\left\{\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial\mathbf{K}_{\mathbf{X}}} \odot \frac{\partial\mathbf{K}_{\mathbf{X}}}{\partial x_{i,\ell}}\right\}, \tag{8.22}$$

where "$\odot$" is a Hadamard product – an element-wise multiplication. Referring to equ. (8.9) the first factor in (8.22) is the $N\times N$ **symmetric** matrix,[2]

$$\begin{aligned}
\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial\mathbf{K}_{\mathbf{X}}} &= -\tfrac{D}{2}\frac{\partial}{\partial\mathbf{K}_{\mathbf{X}}}\log|\mathbf{K}_{\mathbf{X}}| - \tfrac{1}{2}\frac{\partial}{\partial\mathbf{K}_{\mathbf{X}}}\text{trace}\left(\mathbf{K}_{\mathbf{X}}^{-1}\mathbf{Y}\mathbf{W}^2\mathbf{Y}^{\mathsf{T}}\right) \\
&= -\tfrac{D}{2}\mathbf{K}_{\mathbf{X}}^{-1}\frac{\partial}{\partial\mathbf{K}_{\mathbf{X}}}|\mathbf{K}_{\mathbf{X}}| - \tfrac{1}{2}\frac{\partial}{\partial\mathbf{K}_{\mathbf{X}}}\text{trace}\left(\mathbf{W}^{\mathsf{T}}\mathbf{Y}^{\mathsf{T}}\mathbf{K}_{\mathbf{X}}^{-1}\mathbf{Y}\mathbf{W}\right) \\
&= -\tfrac{D}{2}\mathbf{K}_{\mathbf{X}}^{-1} + \tfrac{1}{2}\mathbf{K}_{\mathbf{X}}^{-1}\mathbf{Y}\mathbf{W}\mathbf{W}\mathbf{Y}^{\mathsf{T}}\mathbf{K}_{\mathbf{X}}^{-1}.
\end{aligned} \tag{8.23}$$

We derive the second factor in (8.22) one kernel matrix element at a time:

$$\begin{aligned}
\frac{\partial k}{\partial x_{i,\ell}}(\mathbf{x}_m, \mathbf{x}_n) &= \frac{\partial}{\partial x_{i,\ell}}\left[\alpha_1\exp\left(-\frac{\alpha_2}{2}\|\mathbf{x}_m - \mathbf{x}_n\|^2\right) + \frac{1}{\alpha_3}\delta_{ij}\right] \\
&= -\frac{\alpha_2}{2}\frac{\partial}{\partial x_{i,\ell}}\|\mathbf{x}_m - \mathbf{x}_n\|^2 k(\mathbf{x}_m, \mathbf{x}_n) \\
&= -\frac{\alpha_2}{2}\frac{\partial}{\partial x_{i,\ell}}\langle\mathbf{x}_m - \mathbf{x}_n, \mathbf{x}_m - \mathbf{x}_n\rangle k(\mathbf{x}_m, \mathbf{x}_n).
\end{aligned} \tag{8.24}$$

[1] See `https://en.wikipedia.org/wiki/Matrix_calculus`

[2] For the second term see Peterson and Pederson, *The Matrix Cookbook*, eqn. (113).

At this point, note that all the components of $\mathbf{x}_m$ or $\mathbf{x}_n$ that are not the ℓ^{th} will result in 0. Therefore the inner product simplifies to

$$= -\frac{\alpha_2}{2}\frac{\partial}{\partial x_{i,\ell}}\langle x_{m,\ell} - x_{n,\ell}, x_{m,\ell} - x_{n,\ell}\rangle k\left(\mathbf{x}_m, \mathbf{x}_n\right). \tag{8.25}$$

Now note that if neither $m = i$ nor $n = i$, or if both $m = i$ and $n = i$ then the result is zero. Assume that $m = i$, then the derivation continues:

$$\begin{aligned} &= -\frac{\alpha_2}{2}\left(\langle 1, x_{i,\ell} - x_{n,\ell}\rangle + \langle x_{i,\ell} - x_{n,\ell}, 1\rangle\right) k\left(\mathbf{x}_i, \mathbf{x}_n\right) && (8.26)\\ &= -\frac{\alpha_2}{2}\left(x_{i,\ell} - x_{n,\ell} + x_{i,\ell} - x_{n,\ell}\right) k\left(\mathbf{x}_i, \mathbf{x}_n\right) && (8.27)\\ &= -\alpha_2\left(x_{i,\ell} - x_{n,\ell}\right) k\left(\mathbf{x}_i, \mathbf{x}_n\right) \quad [m = i, n \neq i]. && (8.28)\end{aligned}$$

Finally, if $n = i$ the derivation would continue from (8.25) with

$$\begin{aligned} \frac{\partial k}{\partial x_{i,\ell}}(\mathbf{x}_m, \mathbf{x}_n) &= -\frac{\alpha_2}{2}\left(\langle -1, x_{m,\ell} - x_{i,\ell}\rangle + \langle x_{m,\ell} - x_{i,\ell}, -1\rangle\right) k\left(\mathbf{x}_m, \mathbf{x}_i\right) && (8.29)\\ &= -\alpha_2\left(-x_{m,\ell} + x_{i,\ell}\right) k\left(\mathbf{x}_m, \mathbf{x}_i\right) && (8.30)\\ &= \alpha_2\left(x_{m,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_m, \mathbf{x}_i\right) \quad [m \neq i, n = i]. && (8.31)\end{aligned}$$

There are $N\times N\times d$ of these values over all $\mathbf{X}$. Combine (8.28) and (8.31) to get the partial derivative of the entire kernel matrix, $\mathbf{K}_{\mathbf{X}}$, with respect to scalar $x_{i,\ell}$, the $N\times N$ matrix:

$$\left[\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{i,\ell}}\right]_{m,n} = \begin{cases} \alpha_2 \left(x_{n,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_i, \mathbf{x}_n\right), & \text{for} \quad m = i, \\ \alpha_2 \left(x_{m,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_m, \mathbf{x}_i\right), & \text{for} \quad n = i, \quad . \\ 0, \text{ otherwise} \end{cases} \tag{8.32}$$

Consider $x_{i,\ell}$ – component ℓ of latent vector, $\mathbf{x}_i$. If we fill in an $N\times N$ matrix by collecting the values of (8.32) over $m, n \in \{1, \ldots, N\}$, then the matrix has one non zero row, the i^{th}, and one non zero column, also the i^{th}. Moreover this sparse matrix is symmetric and its main diagonal is 0. That is, the matrix is of the form

$$\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{i,\ell}}\left(\mathbf{x}_m, \mathbf{x}_n\right) = \alpha_2 \begin{bmatrix} \mathbf{0}_{i-1\times i-1} & \mathbf{u}_i & \mathbf{0}_{i-1\times N-i} \\ \mathbf{u}_i^{\mathsf{T}} & 0 & \mathbf{y}_i^{\mathsf{T}} \\ \mathbf{0}_{N-i\times i-1} & \mathbf{y}_i & \mathbf{0}_{N-i\times N-i} \end{bmatrix}, \tag{8.33}$$

where $\mathbf{0}_{r\times s}$ is an $r\times s$ zero matrix and vectors and

$$\mathbf{u}_{\text{row}=i}^{\mathsf{T}} = \begin{bmatrix} \left(x_{1,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_i, \mathbf{x}_1\right) & \cdots & \left(x_{i-1,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_{i-1}, \mathbf{x}_i\right) \end{bmatrix}, \tag{8.34}$$

$$\mathbf{y}_{\text{row}=i}^{\mathsf{T}} = \begin{bmatrix} \left(x_{i+1,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_{i+1}, \mathbf{x}_i\right) & \ldots & \left(x_{N,\ell} - x_{i,\ell}\right) k\left(\mathbf{x}_N, \mathbf{x}_i\right) \end{bmatrix}. \tag{8.35}$$

For each dimension ℓ of latent vector $\mathbf{x}_i$, $\partial \mathcal{L}_{\mathbf{Y}}/\partial x_{i,\ell}$ is the scalar given by eqn. (8.22). To obtain the full gradient, $\nabla_{\mathbf{X}}\mathcal{L}_{\mathbf{Y}}$, requires that we compute (8.22) $N\times d$ times. There is, however, much redundancy in that calculation. There exists another way to compute $\nabla_{\mathbf{X}}\mathcal{L}_{\mathbf{Y}}$

that is much more efficient.

Recall that $\partial \mathcal{L}_{\mathbf{Y}} / \partial \mathbf{K}_{\mathbf{X}}$ is a symmetric matrix. Write its 3×3 submatrix in the upper left as:

$$\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{K}_{\mathbf{X}}} = \begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix}. \tag{8.36}$$

Consider the upper left 3×3 submatrices of the first three $\partial \mathbf{K}_{\mathbf{X}}/\partial x_{i,\ell}$s.

$$\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{1,\ell}} = \alpha_2 \begin{bmatrix} 0 & (x_{2,\ell} - x_{1,\ell})k(\mathbf{x}_1, \mathbf{x}_2) & (x_{3,\ell} - x_{1,\ell})k(\mathbf{x}_1, \mathbf{x}_3) \\ (x_{2,\ell} - x_{1,\ell})k(\mathbf{x}_2, \mathbf{x}_1) & 0 & 0 \\ (x_{3,\ell} - x_{1,\ell})k(\mathbf{x}_3, \mathbf{x}_1) & 0 & 0 \end{bmatrix} \tag{8.37}$$

$$\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{2,\ell}} = \alpha_2 \begin{bmatrix} 0 & (x_{1,\ell} - x_{2,\ell})k(\mathbf{x}_2, \mathbf{x}_1) & 0 \\ (x_{1,\ell} - x_{2,\ell})k(\mathbf{x}_2, \mathbf{x}_1) & 0 & (x_{3,\ell} - x_{2,\ell})k(\mathbf{x}_2, \mathbf{x}_3) \\ 0 & (x_{3,\ell} - x_{2,\ell})k(\mathbf{x}_3, \mathbf{x}_2) & 0 \end{bmatrix} \tag{8.38}$$

$$\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{3,\ell}} = \alpha_2 \begin{bmatrix} 0 & 0 & (x_{1,\ell} - x_{3,\ell})k(\mathbf{x}_1, \mathbf{x}_3) \\ 0 & 0 & (x_{2,\ell} - x_{3,\ell})k(\mathbf{x}_2, \mathbf{x}_3) \\ (x_{1,\ell} - x_{3,\ell})k(\mathbf{x}_3, \mathbf{x}_1) & (x_{2,\ell} - x_{3,\ell})k(\mathbf{x}_3, \mathbf{x}_2) & 0 \end{bmatrix} \tag{8.39}$$

Note that they sum to 0. Through induction one can deduce that is true for any N.

$$\sum_{i=1}^{N} \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{i,\ell}} = 0. \tag{8.40}$$

Using that information rewrite the three matrices as

$$\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{1,\ell}} = \begin{bmatrix} 0 & r & s \\ r & 0 & 0 \\ s & 0 & 0 \end{bmatrix}, \quad \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{2,\ell}} = \begin{bmatrix} 0 & -r & 0 \\ -r & 0 & t \\ 0 & t & 0 \end{bmatrix}, \quad \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial x_{3,\ell}} = \begin{bmatrix} 0 & 0 & -s \\ 0 & 0 & -t \\ -s & -t & 0 \end{bmatrix}. \tag{8.41}$$

Then, the first three $\partial\mathcal{L}_{\mathbf{Y}}/\partial x_{i,\ell}$s are

$$\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial x_{1,\ell}} = \text{trace}\left\{\left[\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial\mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}}\frac{\partial\mathbf{K}_{\mathbf{X}}}{\partial x_{1,\ell}}\right\} = \text{trace}\left\{\begin{bmatrix} br+cs & \cdot & \cdot \\ \cdot & br & \cdot \\ \cdot & \cdot & cs \end{bmatrix}\right\} = 2(br+cs), \tag{8.42}$$

$$\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial x_{2,\ell}} = \text{trace}\left\{\left[\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial\mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}}\frac{\partial\mathbf{K}_{\mathbf{X}}}{\partial x_{2,\ell}}\right\} = \text{trace}\left\{\begin{bmatrix} -br & \cdot & \cdot \\ \cdot & -br+et & \cdot \\ \cdot & \cdot & et \end{bmatrix}\right\} = 2(-br+et), \tag{8.43}$$

$$\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial x_{3,\ell}} = \text{trace}\left\{\left[\frac{\partial\mathcal{L}_{\mathbf{Y}}}{\partial\mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}}\frac{\partial\mathbf{K}_{\mathbf{X}}}{\partial x_{3,\ell}}\right\} = \text{trace}\left\{\begin{bmatrix} -cs & \cdot & \cdot \\ \cdot & -et & \cdot \\ \cdot & \cdot & -cs-et \end{bmatrix}\right\} = 2(-cs-et). \tag{8.44}$$

Now note that,

$$2\sum_{\text{cols}}\left\{\begin{bmatrix} a & b & c \\ b & d & e \\ c & e & f \end{bmatrix} \odot \begin{bmatrix} 0 & r & s \\ -r & 0 & t \\ -s & -t & 0 \end{bmatrix}\right\} = 2\begin{bmatrix} br+cs \\ -br+et \\ -cs-et \end{bmatrix} = \begin{bmatrix} \partial\mathcal{L}_{\mathbf{Y}}/\partial x_{1,\ell} \\ \partial\mathcal{L}_{\mathbf{Y}}/\partial x_{2,\ell} \\ \partial\mathcal{L}_{\mathbf{Y}}/\partial x_{3,\ell} \end{bmatrix}, \tag{8.45}$$

where the second matrix is

$$\sum_{i=1}^{N}\Phi_i\left[\frac{\partial\mathbf{K}_{\mathbf{X}}}{\partial x_{i,\ell}}\right] \quad \text{with} \quad \Phi_i[A] = \begin{cases} A_{in} & \text{for } n=1,\ldots,N, \text{ if } m=i \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{8.46}$$

Also note that although each $\partial\mathbf{K}_{\mathbf{X}}/\partial x_{i,\ell}$ is symmetric, the second matrix in (8.45) is antisymmetric. The result extends through induction from 3 to N to yield the partial derivative of

the log likelihood with respect to dimension ℓ of $\mathbf{X}$. It is the N-vector,

$$\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{x}_\ell} = -\alpha_2 \sum_{\text{cols}} \left[\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{K}_{\mathbf{X}}} \odot 2\boldsymbol{\chi}_\ell \odot \mathbf{K}_{\mathbf{X}} \right] \tag{8.47}$$

$$= -2\alpha_2 \sum_{\text{cols}} \left[\left(\frac{1}{2} \mathbf{K}_{\mathbf{X}}^{-1} \mathbf{Y}\mathbf{W}\mathbf{W}\mathbf{Y}^{\mathsf{T}} \mathbf{K}_{\mathbf{X}}^{-1} - \frac{D}{2} \mathbf{K}_{\mathbf{X}}^{-1} \right) \odot \boldsymbol{\chi}_\ell \odot \mathbf{K}_{\mathbf{X}} \right]. \tag{8.48}$$

where $N \times N$ matrix $\boldsymbol{\chi}_\ell$ has elements

$$[\boldsymbol{\chi}_\ell]_{m,n} = x_{m,\ell} - x_{n,\ell}. \tag{8.49}$$

Note that it is antisymmetric. $\mathbf{K}_{\mathbf{x}}$ is the kernel matrix (8.6) Matrix $\boldsymbol{\chi}_\ell \odot \mathbf{K}_{\mathbf{x}}$ is the antisymmetric aggregated kernel constructed from all the $\partial \mathbf{K}_{\mathbf{X}} / \partial x_{i,\ell}$ matrices.[3] If we collect those for all $\ell = 1, \ldots, d$ we get the $N \times d$ matrix of partial derivatives of $\mathcal{L}_{\mathbf{Y}}$ with respect to $\mathbf{X}$, the Jacobian matrix,

$$\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{X}} = \left[\begin{array}{ccc} \frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{x}_d} \end{array} \right]. \tag{8.50}$$

Then the gradient of $\mathcal{L}_{\mathbf{Y}}$ with respect to $\mathbf{X}$ is the Nd-dimensional row vector,

$$\nabla_{\mathbf{X}} \mathcal{L}_{\mathbf{Y}} = \left[\text{vec} \left(\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{X}} \right) \right]^{\mathsf{T}}, \tag{8.51}$$

where vec is the vectorization operator that makes a column vector from a matrix by stacking its columns from left to right.

The gradient of $\mathcal{L}_{\mathbf{Y}}$ with respect to the $D \times D$ diagonal weight matrix is found from its partial derivatives with respect to the weights w_i for $i = 1, \ldots, D$. Referring to eqn. (8.20)

[3] To get the same result as eqn. (8.48) trace eqn. (8.22) would have to be executed N times – at considerably more computational expense than (8.48) computed once.

we find:

$$\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial w_i} = \frac{N}{w_i} - w_i \mathbf{y}_i^{\mathsf{T}} \left[\mathbf{K}_{\mathbf{X}}^{-1} \mathbf{y}_i \right]. \tag{8.52}$$

Then the gradient is the D-dimensional row vector,

$$\nabla_{\mathbf{W}} \mathcal{L}_{\mathbf{Y}} = \left[\begin{array}{ccc} \frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial w_1} & \cdots & \frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial w_D} \end{array} \right]. \tag{8.53}$$

Since the $\boldsymbol{\alpha}$ parameters are hidden inside the RBF kernel, we need to use the chain rule to compute the gradient of $\mathcal{L}_{\mathbf{Y}}$ with respect to them. The partial derivatives of the kernel values with respect the alphas are,

$$\begin{aligned} \frac{\partial k}{\partial \alpha_1}(\mathbf{x}_m, \mathbf{x}_n) &= \exp\left(-\tfrac{\alpha_2}{2} \left\| \mathbf{x}_m - \mathbf{x}_n \right\|^2\right), & (8.54) \\ \frac{\partial k}{\partial \alpha_2}(\mathbf{x}_m, \mathbf{x}_n) &= -\tfrac{1}{2} \left\| \mathbf{x}_m - \mathbf{x}_n \right\|^2 \left[k(\mathbf{x}_m, \mathbf{x}_n) - \tfrac{1}{\alpha_3} \delta_{m,n} \right], \quad \text{and} & (8.55) \\ \frac{\partial k}{\partial \alpha_3}(\mathbf{x}_m, \mathbf{x}_n) &= -\frac{1}{\alpha_3^2} \delta_{m,n}. & (8.56) \end{aligned}$$

Over all the latent variables they form the $N \times N$ matrices,

$$\begin{aligned} \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_1} &= \tfrac{1}{\alpha_1} \left[\mathbf{K}_{\mathbf{X}} - \tfrac{1}{\alpha_3} \mathbf{I}_{N \times N} \right], & (8.57) \\ \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_2} &= -\tfrac{1}{2} \operatorname{dist}^2(\mathbf{X}) \odot \mathbf{K}_{\mathbf{X}}, \quad \text{and} & (8.58) \\ \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_3} &= -\frac{1}{\alpha_3^2} \mathbf{I}_{N \times N}. & (8.59) \end{aligned}$$

$\operatorname{dist}^2(\mathbf{X})$ is the $N \times N$ matrix of squared distances between all the latent vectors. Then the

derivative of $\mathcal{L}_{\mathbf{Y}}$ with respect to α_ℓ is

$$\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \alpha_j} = \text{trace}\left[\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{K}_{\mathbf{X}}}\frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_j}\right] = \sum_{\text{cols}}\sum_{\text{rows}}\left[\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}} \odot \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_j}, \tag{8.60}$$

for $j = 1, 2, 3$. Note that $\partial \mathcal{L}_{\mathbf{Y}} / \partial \mathbf{K}_{\mathbf{X}}$ is symmetric so the transpose is unnecessary. Then the gradient is the row vector

$$\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{\mathbf{Y}} = \left[\begin{array}{ccc} \dfrac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \alpha_1} & \dfrac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \alpha_2} & \dfrac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \alpha_3} \end{array}\right]. \tag{8.61}$$

The gradient of the log likelihood of the conditional probability of the data, $\mathbf{Y}$, is therefore the set

$$\nabla_{\mathbf{X},\mathbf{W},\boldsymbol{\alpha}} \mathcal{L}_{\mathbf{Y}} = \left\{\nabla_{\mathbf{X}} \mathcal{L}_{\mathbf{Y}}, \nabla_{\mathbf{W}} \mathcal{L}_{\mathbf{Y}}, \nabla_{\alpha} \mathcal{L}_{\mathbf{Y}}\right\}. \tag{8.62}$$

given by eqns. (8.21), (8.53), and (8.61).

Equation for MAP estimation of the latent variables.

In Wang et al. [] the authors minimize the joint negative log-posterior of the unknowns, eqn. (8.20), plus the sum of the logs of hyperparameters, $\boldsymbol{\alpha}$. That is equivalent to maximizing

$$\mathcal{L}_{\mathbf{Y}} = N\sum_{\ell=1}^{D}\log\left(w_\ell\right) - D\sum_{i=1}^{N}\log\left(\text{diag}\left[\mathbf{L}\right]\right) - \tfrac{1}{2}\sum_{\ell=1}^{D} w_\ell^2 \mathbf{y}_\ell^{c\mathsf{T}}\left[\mathbf{K}_{\mathbf{X}}^{-1}\mathbf{y}_\ell^c\right] - \sum_{j=1}^{3}\log(\alpha_j). \tag{8.63}$$

The constant second term in (8.20) is unnecessary for optimization.

Since a term that is a function of $\boldsymbol{\alpha}$ has been added to (8.20) its gradient must be modified.

In particular (8.61) becomes

$$\begin{aligned}\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \alpha_j} &= \sum_{\text{cols}}\sum_{\text{rows}}\left[\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}} \odot \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_j} - \frac{\partial}{\partial \alpha_j}\sum_{j=1}^{3}\log(\alpha_j) \\ &= \sum_{\text{cols}}\sum_{\text{rows}}\left[\frac{\partial \mathcal{L}_{\mathbf{Y}}}{\partial \mathbf{K}_{\mathbf{X}}}\right]^{\mathsf{T}} \odot \frac{\partial \mathbf{K}_{\mathbf{X}}}{\partial \alpha_j} - \frac{1}{\alpha_j}. \end{aligned} \tag{8.64}$$

Equation (8.62) holds with that modification of $\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{\mathbf{Y}}$.[4]

8.2 Gaussian Process Matlab Code

8.2.1 Metric Tensor

Code to compute the metric tensor at a given point. See Section 5.2

```
function [g] = MetricTensor_GP( p,sampleP,sampleY,theta,varianceTerm,
    optionalGFunc )
%METRICTENSOR_GP Calculates the Metric Tensor for a GP Manifold
%   p       -- Manifold coordinates for point on manifold to compute symbols
%   sampleP -- Prior coordinate points for GP
%   sampleY -- Prior manifold points for GP
%   theta   -- GP hyperparameters
%   varianceTerm -- Flag to determine whether or not to include variance in
%                   calculations
%   optionalGFunc -- Function to modify metric tensor after calculations.
%                    Used primarily for obstacle avoidance

%Number of GP prior points
numSamples = size(sampleY,1);
%Dimension of GP Manifold
numGPDims = size(p,2);
%Dimension of embedding space for the GP Manifold
numDims = size(sampleY,2);
%Number of points to compute metric tensor for
numPoints = size(p,1);

% Legacy support for calls with no varianceTerm defined
if nargin < 5
varianceTerm = false;
end


%RBF Kernel matrix for GP Manifold
```

[4] 2016-05-13 This has been checked numerically and found to be accurate.

```
K_sample = GPrbfKernel(sampleP,theta);



%Initialize vectors
Xdot = zeros(numPoints,numDims,numGPDims);
Sigma = zeros(numPoints,numGPDims,numGPDims);
kdotvec = zeros(numPoints,numSamples,numGPDims);

%First derivative and variance terms
for i=1:numGPDims
%Covariance vector derivative
kdotvec(:,:,i) = GPrbfPointDer(p,sampleP,theta,i);
%Manifold embedded space derivative
Xdot(:,:,i) = (kdotvec(:,:,i)/K_sample)*sampleY;
if varianceTerm
%calculate variance term
for j=1:i
k_j = GPrbfPointDer(p,sampleP,theta,j);
if i ~= j
%Baseline variance for every point
Sigma(:,i,j) = (theta(2)^2 * p(:,i).*p(:,j).*exp(-theta(2)*(p(:,i).^2 + p(:,j)
    .^2)/2));
end
%Reduce variance based on covariance with existing
Sigma(:,i,j) = Sigma(:,i,j)-diag((kdotvec(:,:,i)/K_sample)*k_j');
Sigma(:,j,i) = Sigma(:,i,j);
end
end
end

%Metric tensor
g = zeros(numPoints,numGPDims,numGPDims);
for i=1:numGPDims
for j=1:i
%Compute each entry of the metric tensor by taking the dot product
%of the manifold embedding space vectors
g(:,i,j)=dot(Xdot(:,:,i),Xdot(:,:,j),2);
%If the variance term is included, add it to the metric tensor
if varianceTerm
g(:,i,j) = g(:,i,j) + Sigma(:,i,j);
end
%Metric tensor is symmetric
g(:,j,i)=g(:,i,j);
end
end

%If the extra metric tensor modification is included, evaluate it at the
%point(s) being considered
if nargin < 6
optionalG = zeros(numPoints,numGPDims,numGPDims);
else
[optionalG,~] = optionalGFunc(p,g);
```

```
end

%Add the extra modification (zero if not included)
g = g + optionalG;
end
```

8.2.2 Christoffel Symbols

Code to compute the Christoffel Symbols at a given point. See Section 5.3

```
function [ symbols ] = ChristoffelSymbols_GP( p,sampleP,sampleY,theta,
    varianceTerm,optionalGFunc )
%CHRISTOFFELSYMBOLS_GP Calculates the Christoffel Symbols for a GP Manifold
%   p       -- Manifold coordinates for point on manifold to compute metric
%   sampleP -- Prior coordinate points for GP
%   sampleY -- Prior manifold points for GP
%   theta   -- GP hyperparameters
%   varianceTerm -- Flag to determine whether or not to include variance in
%                   calculations
%   optionalGFunc -- Function to modify metric tensor after calculations.
%                    Used primarily for obstacle avoidance

%Number of GP prior points
numSamples = size(sampleY,1);
%Dimension of GP Manifold
numGPDims = size(p,2);
%Dimension of embedding space for the GP Manifold
numDims = size(sampleY,2);
%Number of points to compute Christoffel symbols for
numPoints = size(p,1);

% Legacy support for calls with no varianceTerm defined
if nargin < 5
varianceTerm = false;
end


%RBF Kernel matrix for GP Manifold
K_sample = GPrbfKernel(sampleP,theta);




%Initialize vectors
Xdot = zeros(numPoints,numDims,numGPDims);
Sigma = zeros(numPoints,numGPDims,numGPDims);
kdotvec = zeros(numPoints,numSamples,numGPDims);

%First derivative and variance terms
for i=1:numGPDims
%Covariance vector derivative
kdotvec(:,:,i) = GPrbfPointDer(p,sampleP,theta,i);
%Manifold embedded space derivative
```

```
Xdot(:,:,i) = (kdotvec(:,:,i)/K_sample)*sampleY;
if varianceTerm
%calculate variance term
for j=1:i
k_j = GPrbfPointDer(p,sampleP,theta,j);
if i ~= j
%Baseline variance for every point
Sigma(:,i,j) = (theta(2)^2 * p(:,i).*p(:,j).*exp(-theta(2)*(p(:,i).^2 + p(:,j)
    .^2)/2));
end
%Reduce variance based on covariance with existing
Sigma(:,i,j) = Sigma(:,i,j)-diag((kdotvec(:,:,i)/K_sample)*k_j');
Sigma(:,j,i) = Sigma(:,i,j);
end
end
end

%Metric tensor
g = zeros(numPoints,numGPDims,numGPDims);
for i=1:numGPDims
for j=1:i
%Compute each entry of the metric tensor by taking the dot product
%of the manifold embedding space vectors
g(:,i,j)=dot(Xdot(:,:,i),Xdot(:,:,j),2);
%If the variance term is included, add it to the metric tensor
if varianceTerm
g(:,i,j) = g(:,i,j) + Sigma(:,i,j);
end
%Metric tensor is symmetric
g(:,j,i)=g(:,i,j);
end
end

%If the extra metric tensor modification is included, evaluate it at the
%point(s) being considered
if nargin < 6
optionalG = zeros(numPoints,numGPDims,numGPDims);
optionalGDer = zeros(numPoints,numGPDims,numGPDims,numGPDims);
else
[optionalG,optionalGDer] = optionalGFunc(p,g);
end

%Add the extra modification (zero if not included)
g = g + optionalG;

%Initialize the second derivatives
Xddot = zeros(numPoints,numDims,numGPDims,numGPDims);
kddotvec = zeros(numPoints,numSamples,numGPDims,numGPDims);
for i=1:numGPDims
for j=1:i
%Second derivative of the covariance vector
kddotvec(:,:,i,j) = GPrbfPointDDer(p,sampleP,theta,[i,j]);
kddotvec(:,:,j,i) = kddotvec(:,:,i,j);
%Second derivative of the embedded space vector
```

```
96  Xddot(:,:,i,j) = (kddotvec(:,:,i,j)/K_sample)*sampleY;
97  Xddot(:,:,j,i) = Xddot(:,:,i,j);
98  end
99  end
100
101 %Initialize Christoffel symbols variable
102 symbols = zeros(numPoints,numGPDims,numGPDims,numGPDims);
103
104 for j=1:numGPDims
105 for k=1:numGPDims
106 %Initialize vector of derivatives
107 gDerivativeVector = zeros(numPoints,numGPDims);
108 for l=1:numGPDims
109 %Derivatives of extra modification to metric tensor
110 optionalGDer_l=optionalGDer(:,l,j,k)+optionalGDer(:,l,k,j)-optionalGDer(:,j,k,
        l);
111 %Derivatives of GP metric tensor
112 gDerivativeVector(:,l) = (dot(Xddot(:,:,j,k),Xdot(:,:,l),2)+0.5*optionalGDer_l
        );
113 %Derivatives of variance terms
114 if varianceTerm
115 varDerivative = -diag((kdotvec(:,:,l)/K_sample)*kddotvec(:,:,j,k)');
116 if j == k && j ~= l
117 varDerivative = varDerivative + (theta(2)^2 * p(:,l) - theta(2)^3 * p(:,l) .*
        (p(:,j).^2))...
118 .*exp(-theta(2)*(p(:,l).^2 + p(:,j).^2)/2);
119 end
120 gDerivativeVector(:,l) = gDerivativeVector(:,l) + varDerivative;
121 end
122 end
123 for n = 1:numPoints
124 g_n = reshape(g(n,:,:),numGPDims,numGPDims);
125 %Multiply by inverse metric tensor (slow step)
126 %Cannot pre-compute inverse, due to numerical instability
127 %Additionally, inverse is used only once so pre-computing does
128 %not save time.
129 symbols(n,:,j,k) = g_n\gDerivativeVector(n,:)';
130 end
131 end
132 end
133
134
135 end
```

8.2.3 Geodesic Differential Equations

Code to evaluate the geodesic differential equations at a given point. See Section 5, Equation (5.16).

```
1  function [ yprime ] = geodesicDiffEq_GP(t,y,X,Y,theta,varianceTerm,optionalG)
2  %GEODESICDIFFEQ Geodesic Differential Equations for a GP Manifold
3  %   Used by boundary value solvers (bvp4c) to find geodesics on a GP
4  %   Manifold. Optionally include variance and additional metric modifiers.
```

```
5  %   y           -- Trajectory and derivative. y(1:Dims,:) is the trajectory,
6  %                  y(Dims:end,:) is the derivative (with respect to time)
7  %   X           -- Prior manifold coordinates
8  %   Y           -- Prior manifold points
9  %   theta       -- RBF covariance hyperparameters
10 %   varianceTerm-- Flag to include variance terms
11 %   optionalG   -- Function containing metric tensor modifications
12
13 %Coordinate/manifold dimension
14 numGPDims = size(X,2);
15 %number of trajectory points
16 numPoints = size(y,2);
17
18 %Trajectory points
19 Y_in = zeros(numPoints,numGPDims);
20
21
22 for i=1:numGPDims
23 Y_in(:,i)=y(i,:);
24 end
25
26 %Compute Christoffel Symbols
27 if nargin < 6
28 %Compatability (deprecated
29 symbols = ChristoffelSymbols_GP(Y_in,X,Y,theta);
30 elseif nargin < 7
31 %Variance term, no metric tensor modifications
32 symbols = ChristoffelSymbols_GP(Y_in,X,Y,theta,varianceTerm);
33 else
34 %Variance term and metric tensor modifications
35 symbols = ChristoffelSymbols_GP(Y_in,X,Y,theta,varianceTerm,optionalG);
36 end
37
38
39 %initialize time derivatives (for solvers)
40 yprime = zeros(2*numGPDims,numPoints);
41
42 for i=1:numGPDims
43 %2nd order equation, so pass derivatives back as derivatives for
44 %trajectory
45 yprime(i,:) = y(i+numGPDims,:);
46
47 %2nd order derivatives
48 diffEQ_sum = zeros(1,numPoints);
49 for j=1:numGPDims
50 for k=1:numGPDims
51 diffEQ_sum = diffEQ_sum -symbols(:,i,j,k)'.*(y(j+numGPDims,:).*y(k+numGPDims
       ,:));
52 end
53 end
54 yprime(i+numGPDims,:) = diffEQ_sum;
55 end
56
57
```

```
end
```

8.2.4 Geodesic Solver

Code to find a geodesic between two points. Uses the Geodesic Differential Equations.

```
function [path,time, sol] = generateGeodesic(startPoint,endPoint,GP_X,GP_Y,
    hyperParams,varianceTerm,initialSol,optionalG)
%GENERATEGEODESIC Generate a geodesic path for GP Manifolds
%   Initializes and solves the geodesic differential equations using bvp4c
%   startPoint      -- Start point of the geodesic
%   endPoint        -- End point of the geodesic
%   GP_X            -- Prior manifold coordinates
%   GP_Y            -- Prior manifold embedded space points
%   hyperParams     -- RBF Covariance function hyperparameters
%   varianceTerm    -- Flag to include variance terms in calculations
%   initialSol      -- Optional initial guess
%   optionalG       -- Metric tensor modifications

%Dimension of GP Manifold coordinates
numGPDims = size(startPoint,2);

%Residue function to ensure boundary conditions (start and end points)
residue = @(y0,y1,~,~,~,~)[y0(1:numGPDims)'-startPoint,y1(1:numGPDims)'-
    endPoint];

%Initial guess
tGuess=linspace(0,1,10);
yGuess=@(t)[((1-t)*startPoint+t*endPoint),zeros(1,numGPDims)];

%Options for diff eq solver
options = bvpset('RelTol',0.1,'AbsTol',0.1,'Stats','on','Vectorized','on');

tic
%Initialize solution for solver
solinit = bvpinit(tGuess,yGuess);
if nargin > 7
% Full solution, including metric tensor modifications
sol = bvp4c(@geodesicDiffEq_GP,residue,solinit,options,GP_X,GP_Y,hyperParams,
    varianceTerm,optionalG);
elseif nargin > 6
% Solution with optional initial guess
sol = bvp4c(@geodesicDiffEq_GP,residue,initialSol,options,GP_X,GP_Y,
    hyperParams,varianceTerm);
elseif varianceTerm
%Solution including variance term
options = bvpset('RelTol',0.2,'AbsTol',0.2,'Stats','on','Vectorized','on');
sol = bvp4c(@geodesicDiffEq_GP,residue,solinit,options,GP_X,GP_Y,hyperParams,
    varianceTerm);
else
%Default solution
sol = bvp4c(@geodesicDiffEq_GP,residue,solinit,options,GP_X,GP_Y,hyperParams,
    false);
```

```
end
time = toc;

path = sol.y';
end
```

8.2.5 Curve Shortening Flows

Code to find a geodesic through Curve Shortening Flows. See Section 5.3.2

```
function [shortPath,stepError,iterations] = curveShorteningFlow_GP(path,
    christoffelSymbolFunc)
%CURVESHORTENINGFLOW Finds geodesics through curve shortening flows
%   Uses the curvature of a path to shorten the overal length of the path
%   path    -- Original trajectory to shorten
%   christoffelSymbolFunc   -- Function to generate the Christoffel symbols
%                              for the manifold

%Error tolerances
stepSize = 0.05;
tolerance = 0.0001;
stepTolerance = 1e-8;
% paramError = 0.01;

%initialize t, the time parameter for the path
t = linspace(0,10,size(path,1))';
%default hyperparameters for GP
theta = [1,0.1,1];


curPath = path;


%loop conditions
stepError = 1;
stepErrorPrev = 100;
stepChange = 1;
iterations = 1;
%Repeat until tolerance
while stepError > tolerance && iterations < 200% && abs(stepChange) >
    stepTolerance%

%Approximate path with a Gaussian Process
theta = OptimizeRBFHyperparameters( t,curPath,theta );
[K,~,~,~] = GPrbfKernel(t,theta);
K_dt = GPrbfPointDer(t,t,theta,1)/K;
K_dt2 = GPrbfPointDDer(t,t,theta,[1,1])/K;

%Compute Christoffel symbols along the path
chiSym = christoffelSymbolFunc(curPath);

%Find the curvature and tangent space normal of the path at each point
path_dt = K_dt * curPath;
```

```
unitT = path_dt./sqrt(dot(path_dt,path_dt,2));
path_dt2 = K_dt2*curPath;
normalVec = (path_dt2 - dot(path_dt2,unitT,2).*unitT)./sqrt(dot(path_dt,
    path_dt,2));
%unitT = unitT(1:end-1);
for i = 1:size(chiSym,2)
for j = 1:size(chiSym,3)
for k = 1:size(chiSym,4)
normalVec(:,i) = normalVec(:,i) + unitT(:,j).*unitT(:,k).*chiSym(:,i,j,k);
end
end
end

%adjust points of path to reduce curvature
adjust = normalVec;
stepError = sum(dot(adjust,adjust,2));
stepChange = stepError - stepErrorPrev;
stepErrorPrev = stepError;
curPath(2:end-1,:) = curPath(2:end-1,:) + adjust(2:end-1,:).*stepSize;

iterations = iterations + 1;
end
shortPath = curPath;
end
```

8.2.6 Curvature Tensor

Code to compute the curvature tensor at a given point. See Section 5.4.1

```
function [curvature] = CurvatureTensor_GP(p,sampleP,sampleY,theta,type)
%CURVATURETENSOR_GP Computes the curvature tensor for a GP-Manifold
%   p       -- Point(s) where the curvature is to be computed
%   sampleP -- Prior manifold coordinates
%   sampleY -- Prior manifold embedded space points
%   theta   -- RBF covariance hyperparameters
%   type    -- Type of curvature to compute (scalar or full tensor)

%Number of points to compute at
numPoints = size(p,1);
%Dimension of manifold coordinates
numGPDims = size(p,2);

%Default to scalar curvature
if nargin < 5
type = 'scalar';
end

%RBF Covariance kernel
K_sample = GPrbfKernel(sampleP,theta);

for i=1:numGPDims
%First derivative of points with respect to coordinates
Xdot(:,:,i) = GPrbfPointDer(p,sampleP,theta,i)/K_sample*sampleY;
```

```
%Second derivative of points with respect to coordinates
for j=1:i
Xddot(:,:,i,j) = GPrbfPointDDer(p,sampleP,theta,[i,j])/K_sample*sampleY;
Xddot(:,:,j,i) = Xddot(:,:,i,j);
%Third derivative
for k=1:j
Xdddot(:,:,i,j,k) = GPrbfPointDDDer(p,sampleP,theta,[i,j,k])/K_sample*sampleY;
Xdddot(:,:,i,k,j) = Xdddot(:,:,i,j,k);
Xdddot(:,:,j,i,k) = Xdddot(:,:,i,j,k);
Xdddot(:,:,j,k,i) = Xdddot(:,:,i,j,k);
Xdddot(:,:,k,j,i) = Xdddot(:,:,i,j,k);
Xdddot(:,:,k,i,j) = Xdddot(:,:,i,j,k);
end
end
end

%Metric tensor
g = zeros(numPoints,numGPDims,numGPDims);
for i=1:numGPDims
for j=1:i
g(:,i,j)=dot(Xdot(:,:,i),Xdot(:,:,j),2);
g(:,j,i)=g(:,i,j);
end
end


%Inverse metric tensor pre-computation
gInv = zeros(numPoints,numGPDims,numGPDims);
for n=1:numPoints
gInv(n,:,:) = inv(reshape(g(n,:,:),numGPDims,numGPDims));
end


%Christoffel symbols
symbols = zeros(numPoints,numGPDims,numGPDims,numGPDims);
for i=1:numGPDims
for j=1:numGPDims
for k=1:numGPDims
for l=1:numGPDims
symbols(:,i,j,k) = symbols(:,i,j,k) + gInv(:,i,l).*(dot(Xddot(:,:,j,k),Xdot
    (:,:,l),2));
end
end
end
end

%Final Riemannian curvature tensor computation
riemann_curvature = zeros(numPoints,numGPDims,numGPDims,numGPDims,numGPDims);
for i=1:numGPDims
for j=1:numGPDims
for k=1:numGPDims
for l=1:numGPDims
riemann_curvature(:,i,j,k,l) = 0.5 *( dot(Xddot(:,:,i,k),Xddot(:,:,l,j),2) ...
    iljk
```

```
+dot(Xddot(:,:,j,l),Xddot(:,:,k,i),2)...
-dot(Xddot(:,:,i,l),Xddot(:,:,k,j),2)...
-dot(Xddot(:,:,j,k),Xddot(:,:,l,i),2));
symbol_term = zeros(numPoints,1);
for n=1:numGPDims
for p=1:numGPDims
symbol_term = symbol_term + g(:,n,p).*(symbols(:,n,j,k).*symbols(:,p,i,l)...
-symbols(:,n,j,l).*symbols(:,p,i,k));
end
end
riemann_curvature(:,i,j,k,l) = riemann_curvature(:,i,j,k,l) + symbol_term;
end
end
end
end

%If scalar is required, contract tensor using inverse metric tensor.
if strcmp(type,'scalar')
curvature = zeros(numPoints,1);
for i=1:numGPDims
for j=1:numGPDims
for k=1:numGPDims
for l=1:numGPDims
curvature = curvature + gInv(:,i,j).*gInv(:,k,l).*riemann_curvature(:,i,k,j,l)
    ;
end
end
end
end
end

end
```

8.2.7 Gaussian Process Likelihood Wrapper

Code to unpack GPLVM parameters and call the likelihood and gradient functions. Used with Matlab's built-in optimizers to maximize the likelihood.

```
function [L,G] = LikelihoodRBF(params,Y)
%LIKELIHOODRBF Likelihood wrapper for RBF GP
%   params  -- Input from optimization function
%   Y       -- Output Training Data
%   Unpacks params into the inputs for the various likelihood functions,
%   then returns likelihoods and gradients.


%Get number of training points
N = size(Y, 1);
%Dimension of latent variables based on number of points
q = round((length(params)-3)/N);

Nq = N*q;
% latent variables
```

```
X = reshape(params(1:Nq), N, q);
% hyperparameters (exponentials for the purpose of optimization)
alpha = exp(params(end-2:end));

%Gradient computed only if needed
if nargout > 1
%Likelihood and gradient
[L,G] = GPrbfLikelihood(X,alpha,Y);

%Hyper parameters need to be adjusted for the log/exp required by
%optimization
G((end-2):end) = alpha.*G((end-2):end);
L = -L;
G = -G;

else
L = -GPrbfLikelihood(X,alpha,Y);
end
end
```

8.2.8 Gaussian Process Likelihood

Code to compute the likelihood and gradient of a GPLVM. See section 3.2.2 for details on the equations behind this code.

```
function [L,G] = GPrbfLikelihood(X, alpha, Y)
%GPRBFLIKELIHOOD Likelihood calculation for the RBF Gaussian Process
%   Compute the likelihood and gradient for the RBF Gaussian process with
%   X   -- Prior Input points
%   Y   -- Prior Output Points
%   alpha -- RBF Kernel Hyperparameters
%   Gradient is computed with respect to X and alpha

N = size(X, 1); % number of time sample vectors
D = size(Y, 2); % dimension of each sample vector
q = size(X, 2); % dimension of each latent variable

[K, invK, KlessI, DM2] = GPrbfKernel(X, alpha); % compute RBF kernel

% First compute the log determinant of K and sqrt(2pi^(ND))
L = -D*( sum(log(diag(chol(K)))) + ((N/2)*log(2*pi)) );

% Then compute the log of the exponential term.
% Is -(1/2)*trace of invK*Y*diag(W)*diag(W)*Y'
Lex = zeros(D,1);
for d = 1:D
Lex(d) = -(1/2)*Y(:, d)'*invK*Y(:, d);
end
L = L + sum(Lex);


%Gradient computation
if nargout > 1
```

```
%Gradient of the likelihood with respect to the rbf kernel
dL_dK = -(D/2)*invK + .5*invK*(Y*Y')*invK;

% Compute dK/db_i, derivatives of the std rbf kernel K w.r.t. hyper parameters
    alpha
% eqns (56)-(58)
dK = cell(1,3);
% dK{1} is the drvtv of K w.r.t. alpha(1)
% dK{2} is the drvtv of K w.r.t. alpha(2)
% whereas K = alpha(1)*exp(-(alpha(2)*dist2(X,X'))) + I/alpha(3),
% we use KlessI = alpha(1)*exp(-(alpha(2)*dist2(X,X')))
dK{1} =  KlessI/alpha(1);        % eqn (56)
dK{2} = -0.5*DM2.*KlessI;        % eqn (57)
dK{3} = -1/(alpha(3)*alpha(3)); % eqn (58)

% chain rule: dL/da_i = dL/dK x dK/da_i eqn (59)
dL_dAlpha = zeros(1, 3);
for i = 1:2 % alpha(2):alpha(1)
dL_dAlpha(i) = sum(sum(dL_dK.*dK{i}));
end
dL_dAlpha(3) = -sum(diag(dL_dK)/(alpha(3).^2));

for d = 1:q
% compute dKy/dX_d  (dth col of X)
% -alpha(2)*(X_d - X_d').*K;  K = alpha(2)*exp(-(alpha(1)*dist2(X,X')))
dK_dX = GPrbfDer(X, alpha(2),KlessI, d);
% chain rule: dL_dX = dL/dK .* dK/dX
% the second term of the line below makes no sense because
% diag(dK_dX) == 0 always (unless K KlessI+I/alpha(3) is used)
dL_dX(:, d) = sum(dL_dK.*dK_dX, 2) - diag(dL_dK).*diag(dK_dX);
end

G = [dL_dX(:)',dL_dAlpha];
end
end
```

8.2.9 Sample GP Script

Example script using a GP-manifold to find geodesics. Script below used for the TIAGo robot in Section 6.3.6

```

        %Initialize Robot/Solver



        tiagoRB = importrobot('tiago.urdf');
        torsoJoint = tiagoRB.getBody('torso_lift_link').Joint;
        torsoJoint.HomePosition = 0;
        torsoJoint.PositionLimits = [0,1e-6];
        ikSolver = robotics.InverseKinematics('RigidBodyTree',tiagoRB);

        %Define motion plane
```

```
13          %Using the x+y-z = 0, then end effector orientation is constant
14          nVec = [1;1;-1]/sqrt(3);
15          Rn = [1,0,0;
16          0,1,0;
17          0,0,-1];
18          %Pick a grid of points to train with
19          u = linspace(0,0.25,10);
20          v = linspace(-0.25,0.25,10);
21          [U,V] = meshgrid(u,v);
22          coordinates = [U(:),V(:)];
23
24          %Convert coordinates to points
25          basePoint = [0.5,0.0,0.5];
26          orthoVec1 = [1,0,0];
27          orthoVec2 = [0,1,0];
28          pointX = basePoint(1) + coordinates(:,1)*orthoVec1(1) + coordinates
                (:,2)*orthoVec2(1);
29          pointY = basePoint(2) + coordinates(:,1)*orthoVec1(2) + coordinates
                (:,2)*orthoVec2(2);
30          pointZ = basePoint(3) + coordinates(:,1)*0;% + coordinates(:,1).^2 +
                coordinates(:,2).^2;
31          points = [pointX,pointY,pointZ];
32
33
34          %IK solve all points
35          initialGuess = tiagoRB.homeConfiguration;
36          ikSolver.SolverParameters.AllowRandomRestart = false;
37          endEffector = 'arm_tool_link';
38          weights = [0.25,0.25,0.25,1,1,1];
39          jointAngles = zeros(size(points,1),8);
40          for i = 1:size(points,1)
41          p = points(i,:);
42          pose = trvec2tform(p)*eul2tform([0,pi/2,0]);
43          jointSolution = ikSolver(endEffector, pose, weights, initialGuess);
44          initialGuess = jointSolution;
45          jointAngles(i,1) = jointSolution(13).JointPosition;
46          jointAngles(i,2) = jointSolution(14).JointPosition;
47          jointAngles(i,3) = jointSolution(15).JointPosition;
48          jointAngles(i,4) = jointSolution(16).JointPosition;
49          jointAngles(i,5) = jointSolution(17).JointPosition;
50          jointAngles(i,6) = jointSolution(18).JointPosition;
51          jointAngles(i,7) = jointSolution(19).JointPosition;
52          jointAngles(i,8) = jointSolution(20).JointPosition;
53          end
54          %%
55          %Build a GP-Manifold, draw some trajectories
56          X = coordinates;
57          Y = jointAngles;
58          numPoints = size(X,1);
59          alpha = [1,1,1];
60          f = @(x)LikelihoodRBFParam(x,X,Y);
61          %Xflat = X0(:)';
62
```

```
63          optionFwd = optimoptions('fminunc','Algorithm','trust-region','
                SpecifyObjectiveGradient',true,...
64          'MaxIterations',100,'Display','iter');
65
66
67          alpha = fminunc(f,alpha,optionFwd);
68          alpha = exp(alpha);
69          [~,invK,~,~]= GPrbfKernel(X,alpha);
70          [toManifold,manifoldDer] = wrapGPrbf(X,Y,invK,alpha);
71
72          %Solve some trajectories
73          geodesics = cell(30);
74          jointPaths = cell(30);
75          planTime = zeros(30,1);
76          figure(1)
77          hold off
78          scatter(X(:,1),X(:,2),5,'b');
79          hold on
80          for i = 1:30
81          startCoord = [0.25*rand(), 0.5*rand()-0.25];
82          startPoint = startCoord;%basePoint+orthoVec1*startCoord(1)+orthoVec2*
                startCoord(2);
83          endCoord = [0.25*rand(), 0.5*rand()-0.25];
84          endPoint = endCoord;%basePoint+orthoVec1*endCoord(1)+orthoVec2*
                endCoord(2);
85          [geoPath,time] = generateGeodesic(startPoint,endPoint,X,Y,alpha,true);
86          planTime(i) = time;
87          geodesics{i} = geoPath(:,1:2);
88          plot(geoPath(:,1),geoPath(:,2),'k');
89          jointPaths{i} = toManifold(geoPath(:,1:2));
90          end
```

Bibliography

[1] Robert O Ambrose, Hal Aldridge, R Scott Askew, Robert R Burridge, William Bluethmann, Myron Diftler, Chris Lovchik, Darby Magruder, and Fredrik Rehnmark. Robonaut: Nasa's space humanoid. *IEEE Intelligent Systems and Their Applications*, 15(4):57–63, 2000.

[2] Mukund Balasubramanian and Eric L Schwartz. The isomap algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

[3] Oren Barkan, Jonathan Weill, and Amir Averbuch. Gaussian process regression for out-of-sample extension. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2016.

[4] Christopher M Bishop. Model-based machine learning. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20120222, 2013.

[5] Roberto Calandra, Jan Peters, Carl Edward Rasmussen, and Marc Peter Deisenroth. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE, 2016.

[6] Zhe Chen et al. Bayesian filtering: From kalman filters to particle filters, and beyond. *Statistics*, 182(1):1–69, 2003.

[7] Sam Coveney, Cesare Corrado, Caroline H Roney, Daniel O'Hare, Steven E Williams, Mark D O'Neill, Steven A Niederer, Richard H Clayton, Jeremy E Oakley, and Richard D Wilkinson. Gaussian process manifold interpolation for probabilistic atrial activation maps and uncertain conduction velocity. *Philosophical Transactions of the Royal Society A*, 378(2173):20190345, 2020.

[8] Piotr Dollár, Vincent Rabaud, and Serge Belongie. Non-isometric manifold learning: Analysis and an algorithm. In *Proceedings of the 24th international conference on Machine learning*, pages 241–248, 2007.

[9] David Eklund and Søren Hauberg. Expected path length on random manifolds. *arXiv preprint arXiv:1908.07377*, 2019.

[10] Peter Englert, Isabel M Rayas Fernández, Ragesh K Ramachandran, and Gaurav S Sukhatme. Sampling-based motion planning on manifold sequences. *arXiv preprint arXiv:2006.02027*, 2020.

[11] CL Epstein and Michael Gage. The curve shortening flow. In *Wave motion: theory, modelling, and computation*, pages 15–59. Springer, 1987.

[12] Tingran Gao, Shahar Z Kovalsky, and Ingrid Daubechies. Gaussian process landmarking on manifolds. *SIAM Journal on Mathematics of Data Science*, 1(1):208–236, 2019.

[13] Rajarshi Guhaniyogi and David B Dunson. Compressed gaussian process for manifold regression. *The Journal of Machine Learning Research*, 17(1):2472–2497, 2016.

[14] Søren Hauberg. Only bayes should learn a manifold (on the estimation of differential geometric structure from data). *arXiv preprint arXiv:1806.04994*, 2018.

[15] Ioannis Havoutis and Subramanian Ramamoorthy. Geodesic trajectory generation on learnt skill manifolds. In *2010 IEEE International Conference on Robotics and Automation*, pages 2946–2952. IEEE, 2010.

[16] Ioannis Havoutis and Subramanian Ramamoorthy. Motion planning and reactive control on learnt skill manifolds. *The International Journal of Robotics Research*, 32(9-10):1120–1150, 2013.

[17] Martin Jørgensen and Søren Hauberg. Isometric gaussian process latent variable model for dissimilarity data. *arXiv preprint arXiv:2006.11741*, 2020.

[18] Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

[19] Jonathan Ko and Dieter Fox. Learning gp-bayesfilters via gaussian process latent variable models. *Autonomous Robots*, 30(1):3–23, 2011.

[20] Neil Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of machine learning research*, 6(Nov):1783–1816, 2005.

[21] Neil D Lawrence. Learning for larger datasets with the gaussian process latent variable model. In *Artificial intelligence and statistics*, pages 243–250, 2007.

[22] Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (TOG)*, 31(4):28, 2012.

[23] Miao Li, Kenji Tahara, and Aude Billard. Learning task manifolds for constrained object manipulation. *Autonomous Robots*, 42(1):159–174, 2018.

[24] Dawen Liang and John Paisley. Landmarking manifolds with gaussian processes. In *International Conference on Machine Learning*, pages 466–474, 2015.

[25] Lizhen Lin, Niu Mu, Pokman Cheung, David Dunson, et al. Extrinsic gaussian processes for regression and classification on manifolds. *Bayesian Analysis*, 14(3):887–906, 2019.

[26] Stephen Lovett. *Differential Geometry of Manifolds*. A K Peters Ltd, 2020.

[27] Anton Mallasto, Søren Hauberg, and Aasa Feragen. Probabilistic riemannian submanifold learning with wrapped gaussian process latent variable models. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2368–2377. PMLR, 2019.

[28] Souma Mazumdar. Path planning in a riemannian manifold using optimal control. *arXiv preprint arXiv:2006.11205*, 2020.

[29] Al Mead. Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39, 1992.

[30] Don P Mitchell. Spectrally optimal sampling for distribution ray tracing. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 157–164, 1991.

[31] Anthony O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(1):1–24, 1978.

[32] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[33] Chafik Samir, Jean-Michel Loubes, Anne-Françoise Yao, and François Bachoc. Learning a gaussian process model on the riemannian manifold of non-decreasing distribution functions. In *Pacific Rim International Conference on Artificial Intelligence*, pages 107–120. Springer, 2019.

[34] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[35] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, USA, 2004.

[36] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[37] Lindsay I Smith. A tutorial on principal components analysis. Technical report, 2002.

[38] Hector J Sussmann. Lie brackets and local controllability: a sufficient condition for scalar-input systems. *SIAM Journal on Control and Optimization*, 21(5):686–713, 1983.

[39] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[40] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[41] Charanraj A Thimmisetty, Roger G Ghanem, Joshua A White, and Xiao Chen. High-dimensional intrinsic interpolation using gaussian process regression and diffusion maps. *Mathematical Geosciences*, 50(1):77–96, 2018.

[42] F-Y Tzeng and K-L Ma. *Opening the black box-data driven visualization of neural networks*. IEEE, 2005.

[43] Vladimir Vapnik, Steven E Golowich, and Alex J Smola. Support vector method for function approximation, regression estimation and signal processing. In *Advances in neural information processing systems*, pages 281–287, 1997.

[44] Jack M Wang, David J Fleet, and Aaron Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2008.

[45] Norbert Wiener. Extrapolation, interpolation, and smoothing of stationary time series, vol. 2, 1949.

[46] Yun Yang, David B Dunson, et al. Bayesian manifold regression. *The Annals of Statistics*, 44(2):876–905, 2016.

[47] Martijn JA Zeestraten, Ioannis Havoutis, Joao Silvério, Sylvain Calinon, and Darwin G Caldwell. An approach for imitation learning on riemannian manifolds. *IEEE Robotics and Automation Letters*, 2(3):1240–1247, 2017.