Training Gaussian Process Regression models using optimized trajectories

by

Sheran Wiratunga

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Author's Declaration

Abstract

Quadrotor helicopters and robot manipulators are used widely for both research and industrial applications. Both quadrotors and manipulators are difficult to model. Quadrotors have complex dynamic models, especially at high speeds. Obtaining an accurate model of manipulator dynamics is often difficult, due to inaccurate values for link parameters and dynamics such as friction which are difficult to model accurately.

Supervised learning methods such as Gaussian Process Regression (GPR) have been used to learn the inverse dynamics of a system. These methods can estimate a dynamic model from experimental data without requiring the structure of the model to be known, and can be used online to update the model if the system changes over time.

This approach has been used to learn the inverse dynamics of a manipulator, but has not yet been applied to quadrotors. In addition, collecting training data for supervised learning can be difficult and time consuming, and poor or inadequate training data may result in an inaccurate model. Another problem frequently encountered when using GPR to learn the model of a system is the large computational cost of using GPR. A number of sparse approximations of GPR exist to deal with this issue, but it is not clear which sparse approximation results in the best performance, particularly when training data is being added incrementally.

This thesis proposes a method for systematically collecting training data for a GPR model. The trajectory used to collect training data is parameterized, and the parameters are optimized to maximize the GPR variance over the trajectory. This approach is tested both in simulation and experimentally for a quadrotor, and in experiments on a 4-DOF manipulator. Optimizing the training trajectories is shown to reduce the amount of training data required to learn the model of a system.

The thesis also compares three sparse approximations of GPR: the dictionary approach, Sparse Spectrum GPR (SSGP) and simple downsampling of the training data to reduce the size of the training data set. Using a dictionary is found to provide the best performance, even when the dictionary contains a very small subset of the available data.

Finally, all GPR models have hyperparameters, which have a significant impact on the prediction made by the GP model. Training these hyperparameters is important for getting accurate predictions. This thesis evaluates different methods of hyperparameter training on a 4-DOF manipulator to determine the most effective method of training the hyperparameters. For SSGP, the best hyperparameter training strategy is to reinitialize and train the hyperparameters after each trajectory. SSGP is also observed to be highly sensitive to the number of iterations of gradient descent used in hyperparameter training;

too many iterations of gradient descent leads to overfitting and poor predictions. When using a dictionary, the best hyperparameter training method is to retrain the hyperparameters after each trajectory, using the previous hyperparameters as the initial starting point.

Acknowledgements

I would like to thank my supervisor, Dr. Dana Kulić, for her advice and guidance throughout my research. I am deeply grateful for the time and effort she has spent in meetings and reviewing my research results and draft papers.

I would also like to thank Dr. Steven Waslander, of the University of Waterloo WAVE Lab, for advice and assistance with the quadrotor platform, as well as providing equipment from the WAVE Lab. I would particularly like to thank Kevin Ling for his advice and collaboration in the quadrotor experiments.

Finally, I would like to thank my family for their support and encouragement throughout my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Complex robotic systems are widely used for many commercial and research applications. Quadrotor helicopters are a popular platform for unmanned aerial vehicle research and have proven to be useful for many inspection and surveillance based applications. Robot manipulators are frequently used in the manufacturing industry. Both these systems have complex dynamics, but the controllers used are often relatively simple and do not take the dynamic model into account. Using more advanced controllers based on the dynamic model of the system could result in improved tracking, the ability to perform more aggressive trajectories, and reduced energy consumption.

Many manipulators use simple PD or PID controllers to control each joint individually. Since there is often a significant amount of coupling between joints, this may result in increased energy consumption and tracking error, especially during high-speed motion. An alternative control strategy is to use a model-based controller, which includes a model of the manipulator dynamics. However, model-based controllers require an accurate model of the manipulator's dynamics. This requires having accurate data about the manipulator's physical parameters, such as the link masses, link lengths, and center of mass locations [42]. It also requires accurate modelling of highly nonlinear effects such as friction, which are often difficult to model. An alternative approach is to use machine learning to learn the dynamic model based on the input and output data alone. Gaussian Process Regression (GPR) and Locally Weighted Projection Regression (LWPR) are the most widely used regression methods in robotics. GPR usually provides more accurate predictions than LWPR, but has higher computational costs, particularly as the size of the training data set increases [29]. Controllers based on both LWPR and GPR have been shown to provide better performance for controlling robot manipulators compared to classical model-based controllers [41], [8], [27].

Quadrotors are frequently modelled as a rigid, rotating body with simple thrust and drag models of in-flight moments and forces acting on the vehicle [14]. However, both momentum and blade element theory point to more complex nonlinear dependence of the thrust produced by the rotors on the vehicle translational and rotational velocity [12]. Additional aerodynamic effects such as multi-rotor downwash interaction, vortex ring state and ground effect further complicate the vehicle dynamics and lead to large uncertainty in vehicle motion prediction. Models exist for each of these effects, but these models are both difficult to incorporate into vehicle controller design and are dependent on difficult to identify system parameters. As with manipulators, the difficulties with the lack of a high fidelity model can be overcome through the application of learning algorithms which learn the underlying model based on the input and output data alone.

When using machine learning to develop a model of a system, it is important to have training data which provides good coverage of the state space in which the system operates. If there is not enough training data, or the training data does not cover some parts of the state space, the learned model may give inaccurate results [41]. This thesis proposes a systematic method for collecting training data to learn a model of the robot dynamics. The trajectories are parameterized and the trajectory parameters are optimized to maximize the variance in the learned model. This will result in the training trajectories systematically exploring regions of the state space where the variance is large, and the learned model has the least confidence.

1.1 Thesis Contributions

1. This thesis develops an algorithm for systematically collecting training data to use in a learned model of a system. The algorithm is tested on a quadrotor and a 4-DOF manipulator, but could be applied to other systems, such as humanoid robots, which are modelled using learning methods.

2. GPR is used to learn the inverse dynamics model of a quadrotor in simulation and using experimental data. A GPR model could be used in the quadrotor controller to provide improved performance compared to existing control methods.

3. A number of GPR approximations (downsampling, using a dictionary, and Sparse Spectrum GPR) are compared. Each approach is used to model a 4-DOF manipulator, and the mean tracking error when using the GPR predictions in the control loop is compared.

4. Different approaches for training the GPR hyperparameters are compared to identify the best way of training the hyperparameters to get the most accurate predictions. Different hyperparameter training methods are tested in experiments on a 4-DOF manipulator, and the mean tracking error is compared.

1.2 Thesis Outline

Chapter 2 provides background on manipulator and quadrotor dynamics and GPR and GPR approximations. Previous work using GPR and other learning techniques to model manipulators and quadrotors is described.

Chapter 3 proposes a method for optimizing training trajectories to provide the most useful training data.

Chapter 4 describes the quadrotor and manipulator platforms used for experiments. This chapter also describes how GPR or a GPR approximation can be used to learn the model of a quadrotor, and how this model can be used in a controller.

Chapter 5 describes simulations and experiments performed on a quadrotor to test the GPR modelling and the optimization method described in Chapter 3.

Chapter 6 describes experiments performed using a 4-DOF manipulator to test the optimization method proposed in Chapter 3. Multiple GPR variants are compared to evaluate their performance. The best way of training the hyperparameters in order to improve performance is also examined.

Chapter 7 provides a summary of the contributions and results of this thesis, and proposes directions for future work.

Chapter 2

Related Work

2.1 Modelling Manipulators and Quadrotors

2.1.1 Manipulator Dynamics

The dynamics of a robot manipulator are given by the equation

$$\mathbf{D}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \tag{2.1}$$

where $\mathbf{q}$ is a vector of the robot joint angles, $\mathbf{D}(\mathbf{q})$ is the inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ represents the Coriolis and centripetal forces, $\mathbf{G}(\mathbf{q})$ represents the torque due to gravity and $\boldsymbol{\tau}$ is a vector of the torques applied to the robot joints [40]. $\mathbf{D}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{G}(\mathbf{q})$ all depend on the robot structure (i.e. link masses, inertia, etc.) [40]. Equation (2.1) does not include friction, which is difficult to model and is often neglected. Friction and other nonlinearities can be included as additional terms in (2.1).

2.1.2 Quadrotor Dynamics

A quadrotor has 4 rotors to provide thrust. Fig. 2.1 shows a freebody diagram illustrating the forces and moments on a quadrotor. Each rotor exerts a moment on the quadrotor's centre of mass; when all rotors are spinning at the same speed, these moments cancel out. The quadrotor attitude is controlled by varying the rotor speeds to result in a net moment on the quadrotor.

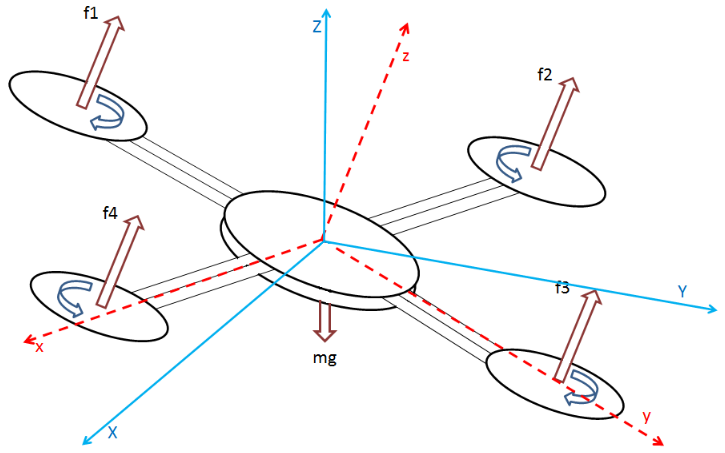


Figure 2.1: Quadrotor freebody diagram [25]

The quadrotor dynamics (in the body frame) can be defined as

$$\begin{aligned}\mathbf{F} &= m\mathbf{a} + \boldsymbol{\omega} \times m\mathbf{v} \\ \mathbf{M} &= \mathbf{J}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}\end{aligned} \tag{2.2}$$

where $\mathbf{F}$ is the total force on the quadrotor, $\mathbf{M}$ is the total moment, $\mathbf{a}$ is the total acceleration, $\mathbf{v}$ is the linear velocity, $\boldsymbol{\omega}$ is the angular velocity of the quadrotor, m is the mass of the quadrotor, and $\mathbf{J}$ is the inertia matrix of the quadrotor [13]. The main forces acting on a quadrotor include gravity, thrust and drag. Thrust is typically modeled as quadratically dependent on rotor speed, which can be controlled directly through the applied voltage or by requesting a reference rotor speed from a brushless DC motor speed controller [13], [36]. At low altitude, there is an additional force (ground effect) which effectively increases the thrust produced by each rotor [19]. Drag is typically modeled as linearly dependent on vehicle translational speed in the x-y plane of the body frame, while gravity applies a constant acceleration in the -z inertial direction. There are a number of nonlinear effects which affect the quadrotor dynamics. These include blade flapping [13], ground effect and airflow disruption by the quadrotor airframe [13]. Models exist for these effects, but these models are both difficult to incorporate into vehicle controller design and depend on difficult to identify system parameters [13].

In hover and at low speeds (airspeed <3 m/s), the above models work admirably and permit reliable reference command tracking in attitude and position using linear control techniques. More aggressive trajectories, such as those flown in indoor flight arenas, have required either learning refinements to reference commands [24] or nonlinear controller

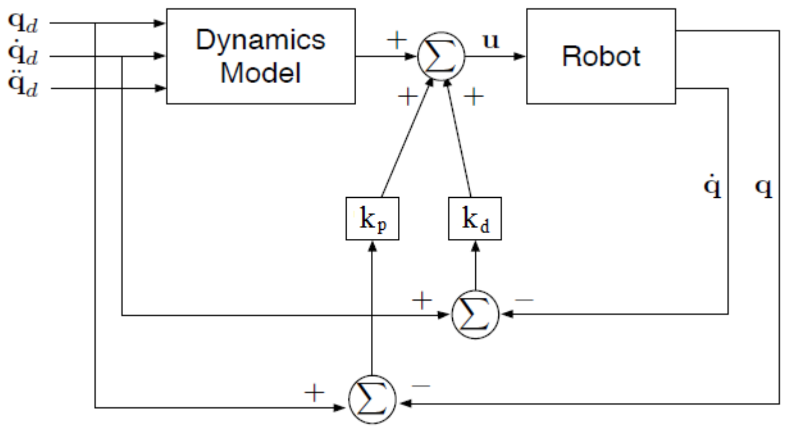


Figure 2.2: Feedforward controller [9]

design [18]. The limited volume available has restricted flight speeds to well below the maximum speeds demonstrated in outdoors flights, which can exceed 20 m/s (65 kph). At these higher speeds, it is unclear how reliable current modeling approaches are.

2.2 Manipulator Control

The simplest manipulator control design is to use PD or PID controllers to control each joint independently. This type of controller is simple to implement, but is likely to result in poor tracking, especially at high speeds, because the coupling between joints is neglected. One approach for improving tracking performance is to use the robot inverse dynamics in the controller [40]. Fig. 2.2 illustrates a feedforward controller which uses a model of the robot inverse dynamics. To track a desired joint trajectory $\mathbf{q_d}$, the required joint torques are computed by substituting $\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}, \ddot{\mathbf{q}}_\mathbf{d}$ into (2.1). Assuming the model is exact, the robot starts out on the desired trajectory, and there are no disturbances, the resulting joint torques will result in perfect tracking. The feedforward signal

$$\mathbf{u_{FF}} = \mathbf{D}(\mathbf{q_d})\ddot{\mathbf{q}}_\mathbf{d} + \mathbf{C}(\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}) + \mathbf{G}(\mathbf{q_d})$$

linearizes the system around the operating point $(\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}, \ddot{\mathbf{q}}_\mathbf{d})$ [2]. PD feedback can then be used to control the linearized system to correct any disturbances or model inaccuracies [2].

Computing the inverse dynamics using (2.1) requires an accurate model of the manipulator and accurate values for the manipulator parameters. Equation (2.1) also neglects nonlinearities such as friction, which is often difficult to accurately model and may have a significant impact on manipulator dynamics [4], [31].

Finding accurate values for the robot parameters is often difficult. Experimental parameter estimation is often performed to measure the parameters in the robot model [42], [33]. Equation (2.1) can be rearranged into the form

$$\boldsymbol{\phi}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\boldsymbol{\theta} = \boldsymbol{\tau} \tag{2.3}$$

where $\boldsymbol{\phi}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$ is the regressor matrix capturing the relationships between the kinematic variables $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$, and $\boldsymbol{\theta}$ is a vector of the unknown parameters. By exciting the manipulator and measuring $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ and $\boldsymbol{\tau}$, equation (2.3) can be solved for $\boldsymbol{\theta}$ using least squares [42], [3]. Measuring the joint torques $\boldsymbol{\tau}$ is often difficult, requiring the torques to be estimated from the motor current [3]. An alternative approach is to use a force plate to measure forces and torques at the base, instead of the torques at the joints [21], [5]. When performing parameter estimation, one major challenge is finding trajectories which will excite all the robot dynamics and minimize the error between the estimated parameters and the actual value. Swevers et al. [42] proposes parameterizing the trajectory in joint-space as a sum of sinusoidal functions. The trajectory parameters $\boldsymbol{\delta}$ are chosen by maximizing a function $f(\boldsymbol{\delta}, \bar{\boldsymbol{\theta}})$ which represents the amount of new information available from the trajectory defined by $\boldsymbol{\delta}$, for the current parameter estimate $\bar{\boldsymbol{\theta}}$. The parameters are estimated by iteratively computing a trajectory, then updating the estimated parameters $\bar{\boldsymbol{\theta}}$.

The robot parameters can also be learned by an adaptive controller [15], [38]. In adaptive control, the mainpulator dynamics are assumed to have a known structure, but some of the model parameters are unknown. The adaptive controller estimates the unknown model parameters, and updates the parameter estimates while the controller is running [15], [38]. Adaptive controllers have also been used to learn the joint friction [10]. The main advantage of using an adaptive controller is that it can estimate unknown or changing parameter values; however, it still requires the structure of the model to be known in advance.

Another method used to control a manipulator is Iterative Learning Control (ILC). ILC is used to learn the feedforward control signal required to track a particular trajectory. The trajectory is tracked repeatedly, and after each iteration, the feedforward signal is updated based on data from the previous iteration [16], [43]. After repeated iterations of the trajectory, the tracking error decreases as the controller learns the dynamics for the trajectory [16], [43]. Compared to parameter estimation and adaptive control, ILC has the advantage that it does not assume the manipulator model in advance; however, it is restricted to learning a single trajectory, and does not provide any information for tracking a different trajectory.

Instead of computing the manipulator dynamics by estimating the manipulator parameters and using them to compute $\mathbf{D}(\mathbf{q})$, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{G}(\mathbf{q})$, the inverse dynamics can be learned by using regression methods to learn the mapping

$$(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}) \mapsto \boldsymbol{\tau} \tag{2.4}$$

The learned model can then be used instead of equation (2.1) to compute the feedforward signal $\mathbf{u_{FF}}$. Learning methods have a set of training data $\mathbf{X}$ and the corresponding training outputs $\mathbf{y}$, where $y_i = f(\mathbf{x_i})$. The learned model then provides the function $y_* = \bar{f}(\mathbf{x})$, allowing the output y to be estimated for any input $\mathbf{x}$.

Two commonly used regression methods for learning robot dynamics are Locally Weighted Projection Regression (LWPR) and Gaussian Process Regression (GPR). LWPR [37] uses multiple local linear models to model a function, and is explicitly designed to efficiently handle large amounts of training data. GPR [34] is described in more detail in Section 2.3, and is a global model of the system. Unlike LWPR, GPR cannot efficiently handle large amounts of training data. As a result, a number of sparse GPR approximations are often used instead of full GPR. These include Sparse Pseudo-Input Gaussian Processes (SPGP) [39], Sparse Online Gaussian Processes (SOGP) [6] and Sparse Spectrum Gaussian Processes (SSGP) [17]. SPGP approximates GPR by using pseudo-inputs, which do not correspond to actual training data points and are trained using gradient descent [39]. The pseudo-inputs can be thought of as an aggregate of the training data, which is optimized to provide the best predictions. SOGP is similar to SPGP, but uses basis vectors which are selected from the training data [6]. SSGP is described in more detail in Section 2.4, and approximates the GPR covariance function using its power spectrum. Multiple local GPR models can also been used to learn the manipulator dynamics [30]. It is also possible to use only a subset of the available data to train the GPR model [28]. This approach is described in more detail in Section 2.5.

Sun De la Cruz [9] studied LWPR, SPGP and SOGP, and proposed ways to incorporate existing knowledge about the system into the learned model. In simulation, SPGP and SOGP were found to have similar performance (SPGP performing slightly better than SOGP), and both SPGP and SOGP perform better than LWPR. Sparse Spectrum Gaussian Process Regression has also been used to learn manipulator dynamics using actual robot data [11], although this model was not used to control the robot. SSGP was found to perform similarly to or better than full GPR, and both SSGP and GPR outperformed LWPR.

2.3 Gaussian Process Regression

A Gaussian Process (GP) is a collection of random variables, any finite number of which have joint Gaussian distributions. A GP can be completely specified by its mean and covariance functions. For any set of training points $\mathbf{X}$ and a testing point $\mathbf{x}_*$, the joint distribution of the training outputs $\mathbf{y}$ and the (unknown) testing output y_* is also Gaussian, and the mean and variance can be computed from the mean and covariance functions evaluated at $\mathbf{X}$ and $\mathbf{x}_*$ [34]. The mean is the value predicted by GPR, while the variance indicates the degree of confidence in the result; a small variance indicates that the prediction is likely to be accurate, while a larger variance indicates some uncertainty in the prediction.

The joint distribution of the training outputs and the predicted mean is

$$\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} k(\mathbf{X},\mathbf{X}) + \sigma_n^2 I & k(\mathbf{X},\mathbf{x}_*) \\ k(\mathbf{x}_*,\mathbf{X}) & k(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}\right)$$

The predicted mean is given by

$$y_* = \mathbf{k}_*{}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} \tag{2.5}$$

The predicted variance is given by

$$\sigma_*^2 = k(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}_*{}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_* \tag{2.6}$$

The n-by-n covariance matrix $\mathbf{K} = k(\mathbf{X},\mathbf{X})$ is calculated by evaluating the covariance function between each of the training inputs ($k_{ij} = k(\mathbf{x_i},\mathbf{x_j})$). The vector $\mathbf{k}_* = k(\mathbf{X},\mathbf{x}_*)$ is calculated by evaluating the covariance function between the testing point and each of the training points.

Most covariance functions have hyperparameters which affect the shape of the function. The choice of covariance function and its hyperparameters has a major impact on the prediction made by GPR. The most common covariance function is the Squared Exponential (SE) function

$$k_{SE}(\mathbf{x_i},\mathbf{x_j}) = \sigma_0^2 \exp\left(-\frac{1}{2}(\mathbf{x_i} - \mathbf{x_j})^T \mathbf{\Lambda}^{-1}(\mathbf{x_i} - \mathbf{x_j})\right)$$

where $\mathbf{\Lambda}$ is a diagonal matrix determining how quickly the covariance decreases for each input dimension. The hyperparameters of the SE covariance function are the variance σ_0^2 and the lengthscales $\mathbf{\Lambda}$. The hyperparameters are trained by maximizing the log marginal likelihood

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

using gradient descent.

Training the hyperparameters requires the covariance matrix to be computed and inverted; this is a $O(n^3)$ operation (where n is the number of training points). Once the hyperparameters are known, the inverted covariance matrix can be stored, after which making further predictions is $O(n)$. When there are large amounts of training data, training (and possibly predictions) becomes too computationally intensive. In this case, various approximations can be used to reduce the size of the covariance matrix and make the GP less computationally intensive.

2.4 Sparse Spectrum Gaussian Process Regression

Sparse Spectrum Gaussian Process Regression (SSGP) [17] approximates the covariance function using its power spectrum (similar to a Fourier transform). A stationary covariance function (defined as a covariance function $k(\mathbf{x_1}, \mathbf{x_2})$ where $k(\mathbf{x_1}, \mathbf{x_2}) = k_0(\mathbf{x_1} - \mathbf{x_2})$) can be defined in terms of its power spectral density $S(\mathbf{s})$

$$k(\mathbf{x_1}, \mathbf{x_2}) = \int_{\mathbb{R}^D} \exp(2\pi j \mathbf{s}^T(\mathbf{x_1} - \mathbf{x_2})) S(\mathbf{s}) d\mathbf{s}$$

where the inputs $\mathbf{x_1}, \mathbf{x_2}$ have D elements. In SSGP, $S(\mathbf{s})$ is approximated by sampling at a set of m frequencies $\{\mathbf{s_r}, -\mathbf{s_r}\}$. These samples are called “spectral points”. The covariance function becomes

$$k(\mathbf{x_1}, \mathbf{x_2}) = \frac{\sigma_0^2}{m} \sum_{n=1}^{m} \cos(2\pi \mathbf{s_n}(\mathbf{x_1} - \mathbf{x_2}))$$

This can be used to approximate any stationary covariance function, with the quality of the approximation improving as the number of spectral points increases. This is shown in Figure 2.3. When m spectral points are used, computing the mean is $O(m)$ and computing the variance is $O(m^2)$ [17]. Computing the log marginal likelihood for hyperparameter training is $O(nm^2)$ [17]. The spectral points can be chosen to match a given covariance function (such as the Squared Exponential function), or trained along with the hyperparameters, instead of using a known covariance function. This is similar to learning the covariance function to fit the training data, although it does increase the risk of overfitting [11].

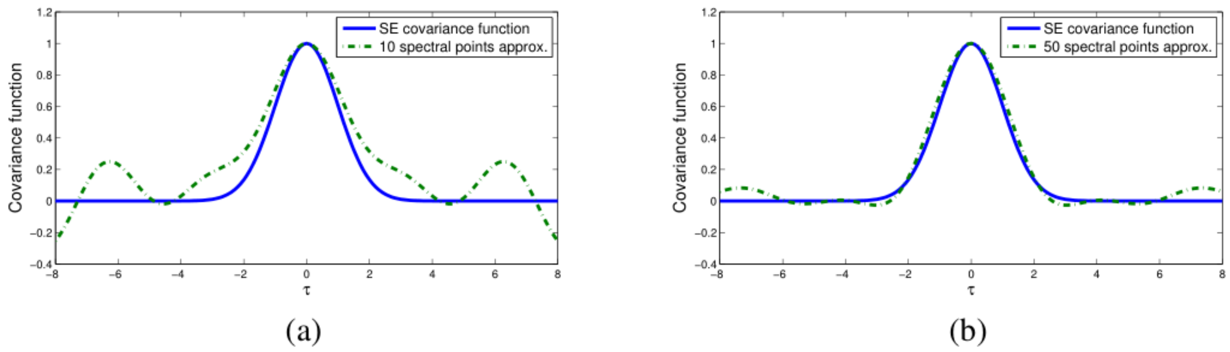


Figure 2.3: Approximation of Squared Exponential covariance function with (a) 10 and (b) 50 spectral points [17]

2.5 GPR Dictionary

An alternative approach for improving GP performance is to limit the size of the training data set by including only the most important data points. Nguyen-Tong and Peters propose selecting a "dictionary" $\mathbf{D}$ containing a fixed number of data points from the full training data set $\mathbf{X}$, and using this dictionary to make predictions using GPR [28]. A new point $\mathbf{d}$ is added to the dictionary if the value

$$\delta = k(\mathbf{d}, \mathbf{d}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \tag{2.7}$$

exceeds a threshold δ_t, where $\mathbf{K} = k(\mathbf{D}, \mathbf{D})$ is the covariance matrix for the dictionary, and $\mathbf{k} = k(\mathbf{D}, \mathbf{d})$ is the covariance between the new point $\mathbf{d}$ and the dictionary [28]. The new data point replaces the data point in the dictionary which has the lowest δ. Equation (2.7) is equivalent to computing the variance (2.6), for $\sigma_n^2 = 0$.

An alternative approach for selecting points to be included in the dictionary is to use the error between the GP prediction and the actual value. In this case, δ is calculated as

$$\delta = y_* - y \tag{2.8}$$

where y_* is the value at $\mathbf{d}$ predicted by (2.5), and y is the actual value at $\mathbf{d}$ from the training data.

2.6 Hyperparameter Training

GPR models and GPR approximations have hyperparameters that need to be set when learning the model. These hyperparameters have a significant impact on the model prediction, and a good choice of hyperparameters is key to making accurate predictions [34]. Often, the hyperparameters are trained using gradient descent on the initial training data set, and then kept constant [11], [27]. However, once additional training data is added, the hyperparameters may need to be updated to match the new training set.

This thesis tests different hyperparameter training approaches to determine the best way of updating the hyperparameters as new training data is added.

2.7 Training Data Collection

When learning the model of a system, one main challenge is collecting training data. In order for the learned model to make accurate predictions, the training data must cover the state space in which the system is operating. If training data is missing, the learned model will not be able to make accurate predictions. For manipulators, training data is often obtained through "motor babbling" - random movements of the arm [32]. Training data can also be obtained by tracking trajectories [37]. These approaches may not initially provide sufficient coverage of the state space, in which case more training data will have to be collected until the performance of the model improves.

Another approach is to perform online learning, in which training data is collected and added to the model while the arm is moving [9], [32], [28], [7]. Online learning ensures that the system collects training data for the region of the state-space in which the manipulator is operating, and allows the learned model to detect changes in the system (for example, due to wear) [7]. However, online learning still results in poor initial performance, because the system has not yet collected sufficient training data to learn the model for the current region of the state space. Also, online learning requires the system to be able to collect and process large amounts of data efficiently. This increases computational requirements, and prevents some learning methods from being used. For example, making predictions in GPR is $O(n)$, which means that the system will become slower as more training data is added.

A systematic method for collecting training data could make learning the system model faster, by reducing the time required to collect training data. It could also potentially improve performance by ensuring that there is sufficient training data for the entire state space. Also, systematically collecting training data could reduce the overall size of the training data set, improving performance.

Chapter 3

Optimization

To ensure good coverage of the state-space and minimize the amount of flight time required to acquire sufficient training data, it is helpful to have a systematic method of designing trajectories for collecting training data. This can be done by parameterizing the trajectory, and then optimizing the trajectory parameters to maximize some metric.

Since the goal of the optimization is to find trajectories which provide good coverage of the state space, the optimization metric should measure how close a trajectory is to the existing training data. The variance predicted by the GP indicates the degree of confidence in the prediction, with a small variance indicating high confidence in the prediction. One possible optimization metric is to make predictions for a trajectory and compute the average variance. If the result is small, it indicates that the GP has high confidence in its predictions and the trajectory is close to the existing training data; if the result is large, it indicates that the GP does not have confidence in its predictions and more training data is required. Optimizing the trajectory parameters to maximize the variance is expected to result in a trajectory that is far from the existing training data.

The trajectory can be parameterized as a sum of sine and cosine functions

$$r(t) = \sum_{i=1}^{N} \frac{a_i}{\omega_0 i} \sin(\omega_0 i t) - \sum_{i=1}^{N} \frac{b_i}{\omega_0 i} \cos(\omega_0 i t) + r_0 \tag{3.1}$$

with a_i and b_i specifying the amplitude of the sinusoidal functions and r_0 specifying a constant offset. This parameterization was originally proposed by Swevers et al. for performing parameter estimation [42]. This parameterization is chosen because it provides a wide range of possible trajectories. The frequencies in the trajectory can be selected by setting ω_0 and N. The choice of ω_0 also affects the trajectory length, since the trajectory should include at least one full period $\frac{2\pi}{\omega_0}$.

The trajectory is optimized by solving

$$\begin{gathered}\underset{a_1,\ldots,a_N,b_1,\ldots,b_N,r_0}{\operatorname{argmin}} \sum \sigma_*(a_1,\ldots,a_N,b_1,\ldots,b_N,r_0) \\ \text{subject to} \\ c_i(t) \leq C_i, \text{ for } i = 1\ldots n\end{gathered} \tag{3.2}$$

where $\sigma_*(a_1,\ldots,a_N,b_1,\ldots,b_N,r_0)$ computes the GP variance (equation (2.6)) at each point along the trajectory defined by $(a_1,\ldots,a_N,b_1,\ldots,b_N,r_0)$, and $c_i(t)$ and C_i represent constraints on the trajectory.

3.1 Quadrotor

To simplify tracking the trajectory, the trajectory is specified by the quadrotor position in the inertial reference frame. The position in each axis is parameterized as a sum of sine and cosine functions:

$$\begin{aligned}x(t) &= \sum_{i=1}^{N} \frac{a_i^x}{\omega_0 i}\sin(\omega_0 i t) - \sum_{i=1}^{N} \frac{b_i^x}{\omega_0 i}\cos(\omega_0 i t) \\ y(t) &= \sum_{i=1}^{N} \frac{a_i^y}{\omega_0 i}\sin(\omega_0 i t) - \sum_{i=1}^{N} \frac{b_i^y}{\omega_0 i}\cos(\omega_0 i t) \\ z(t) &= \sum_{i=1}^{N} \frac{a_i^z}{\omega_0 i}\sin(\omega_0 i t) - \sum_{i=1}^{N} \frac{b_i^z}{\omega_0 i}\cos(\omega_0 i t) + z_0\end{aligned} \tag{3.3}$$

with a_i and b_i specifying the amplitude of the sinusoidal functions and z_0 specifying the height.

The trajectory is optimized by solving

$$\begin{gathered}\underset{a^x,a^y,a^z,b^x,b^y,b^z,z_0}{\operatorname{argmin}} \sum \sigma_*(a^x,a^y,a^z,b^x,b^y,b^z,z_0) \\ \text{subject to} \\ r_{min} \leq r(t) \leq r_{max} \\ ||v(t)|| \leq v_{max}\end{gathered} \tag{3.4}$$

where $\sigma_*(a^x, a^y, a^z, b^x, b^y, b^z, z_0)$ computes the GP variance (equation (2.6)) at each point along the trajectory defined by $(a^x, a^y, a^z, b^x, b^y, b^z, z_0)$, $r(t)$ is the position of the quadrotor (in the inertial frame) at time t, $v(t)$ is the velocity along the trajectory, r_{min} and r_{max} are the position limits, and v_{max} is the maximum velocity.

The procedure for learning the quadrotor model is:

1. Fly an arbitrary initial trajectory and use the results as the initial training data set.
2. Compute a new, optimized training trajectory by maximizing the predicted variance along the new trajectory.
3. Fly the new trajectory and add the measured data to the training data set.
4. Repeat steps 2-3.

During trajectory optimization, the altitude, acceleration, angular velocity, angular acceleration of the trajectory are needed in order to calculate the variance along the new trajectory. Because the trajectory has not been flown yet, measured data is not available, and these values have to be calculated from the trajectory parameters $(a^x, a^y, a^z, b^x, b^y, b^z, z_0)$. The altitude is directly specified by the trajectory, and the acceleration can easily be calculated by differentiating the trajectory. The quadrotor attitude is not directly specified by the trajectory. To estimate the quadrotor attitude, the required thrust vector is computed as the desired acceleration plus a constant gravity term. The quadrotor attitude can then be calculated so the thrust vector is in the correct direction. The angular velocity and acceleration can be determined by differentiating the computed attitude. This method for determining the quadrotor attitude ignores drag. If the drag coefficient is known, the drag can also be computed and used in the target attitude calculations. An alternative method of determining the quadrotor attitude is to train a second Gaussian Process to learn the relationship between the quadrotor altitude, velocity and acceleration and the quadrotor attitude.

When performing the optimization, the optimization metric is computed using the following procedure:

1. Assuming the only forces acting on the quadrotor are gravity and thrust, compute the required attitude for the quadrotor to track the trajectory.
2. Compute the altitude, acceleration, angular velocity and angular acceleration required by the trajectory.

3. Compute the GP variance at each timestep of the trajectory.

4. Compute the average variance along the entire trajectory.

3.2 Manipulator

The trajectory of each joint is parameterized as a sum of sine and cosine functions

$$q(t) = \sum_{i=1}^{N} \frac{a_i}{\omega_0 i} \sin(\omega_0 i t) - \sum_{i=1}^{N} \frac{b_i}{\omega_0 i} \cos(\omega_0 i t) + q_0 \tag{3.5}$$

The frequency ω_0, amplitudes a and b and initial joint angle q_0 are optimized to maximize the variance of the model prediction along the trajectory.

The trajectory is optimized by solving

$$\begin{gathered} \underset{a,b,q_0,\omega_0}{\operatorname{argmin}} \sum \sigma_*(a, b, q_0, \omega_0) \\ \text{subject to} \\ q_{min} \leq q(t) \leq q_{max} \end{gathered} \tag{3.6}$$

where $\sigma_*(a, b, q_0, \omega_0)$ computes the model variance at each point along the trajectory defined by (a, b, q_0, ω_0) and q_{min} and q_{max} are the joint limits.

The algorithm for learning the robot model is:

1. Track an arbitrary initial trajectory and use the results as the initial training data set.

2. Compute a new, optimized training trajectory by maximizing the variance in predictions made on the new trajectory.

3. Track the new trajectory and add the results to the training data set.

4. Repeat steps 2-3.

3.3 Summary

When collecting data for a GPR model, the trajectory tracked to collect data can be optimized by finding a trajectory which will maximize the predicted GP variance over the trajectory. The GP variance acts as an indicator of the confidence in the prediction, so a large variance indicates that the prediction may be inaccurate, and more training data is required. Optimizing the trajectory requires the trajectory to be parameterized. The parameterization chosen is to use a sum of sines and cosines. The GP model can then be learned by iteratively computing a new trajectory by maximizing the variance, and updating the GP model with data collected by tracking the new trajectory.

Chapter 4

Experimental Platforms

4.1 Quadrotor

4.1.1 Simulation

The quadrotor model from [20] was used to simulate a quadrotor in Matlab [22]. The model simulates a small quadrotor, similar in size to the Parrot AR Drone. The simulation model includes drag, ground effect and models the DC motors, but does not simulate blade flapping and other nonlinearities. Gaussian noise is added to the acceleration values to simulate random disturbances.

The GP training inputs consisted of the altitude, linear acceleration, angular velocity and angular acceleration of the quadrotor. The GP training outputs consisted of the 4 motor voltages.

At the beginning of each simulation, the quadrotor is hovering at the start of the trajectory. A PID controller is used to convert position error to desired accelerations, which are then converted to attitude commands. Altitude is also controlled using a PID controller. The simulation lasts for the duration of the trajectory. The simulation ran at a 1000 Hz sampling rate. This generated too much training data for the GP model to process, so the data was downsampled to 10 Hz before being added to the GP training set.

4.1.2 Hardware

Experimental data was collected on an AscTec Pelican quadrotor helicopter, shown in Figure 4.1, equipped with a MaxBotix sonar sensor for improved height estimation. The vehicle was controlled using the `asctec_drivers` ROS stack released by CCNY [1]. Attitude control on the vehicle was implemented in a low-level processor by AscTec and came off-the-shelf with the vehicle. For position control, a simple PID control scheme was used to map position error to desired accelerations. Desired accelerations were then mapped to attitude commands through a dynamic model [23]. For altitude control, a PID,DD controller was implemented [12]. Position data in the horizontal plane was measured with an Optitrack motion capture system which provided the vehicle with position updates at 40 Hz. Onboard data from the quadrotor was recorded at 20Hz.

The GP training inputs consisted of the altitude, linear acceleration, angular velocity and angular acceleration of the quadrotor. The altitude of the quadrotor was measured using the Optitrack motion capture system. The linear acceleration and angular velocity were measured using an onboard IMU. The angular acceleration was calculated by numerically differentiating the angular velocity data. The GP training outputs were the pitch, yaw, roll and thrust commands.

4.2 Manipulator

A 4-DOF manipulator was used for experiments. Fig. 4.2 shows the manipulator used in experiments. The first (waist) joint of the manipulator rotates about the vertical axis; the remaining (shoulder, elbow and wrist) joints rotate around parallel horizontal axes and are controlled by linear actuators.

The arm was controlled using a feedforward controller, as in Fig. 2.2, running at 100 Hz. When collecting training data the robot was controlled using only PD feedback control and the feedforward signal was set to zero. The training inputs ($\mathbf{X}$) from each trajectory consisted of the desired joint position, velocity and acceleration, and the training outputs ($\mathbf{y}$) were the applied motor PWM. Ideally, the training data would have consisted of the measured joint position, velocity, and acceleration; however, only the joint position could be measured, and numerically differentiating the measured joint position led to very noisy velocity and acceleration values. To reduce the effect of noise on the GP, the desired joint position, velocity and acceleration were used instead of the measured values. The performance of the learned model was tested by using the model to compute a feedforward signal, which was then used in the control loop (along with the PD feedback).

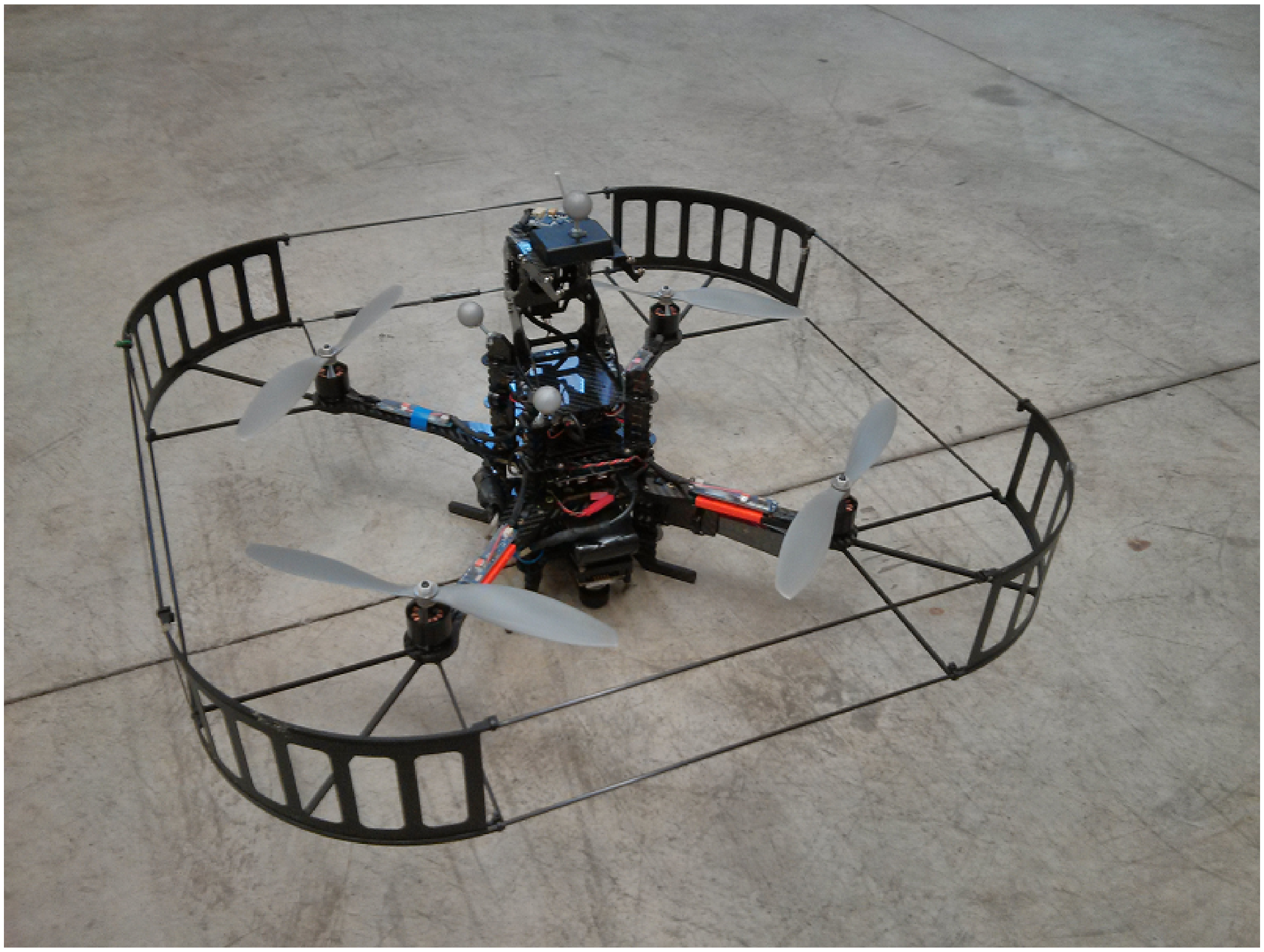


Figure 4.1: Pelican quadrotor used in experiments

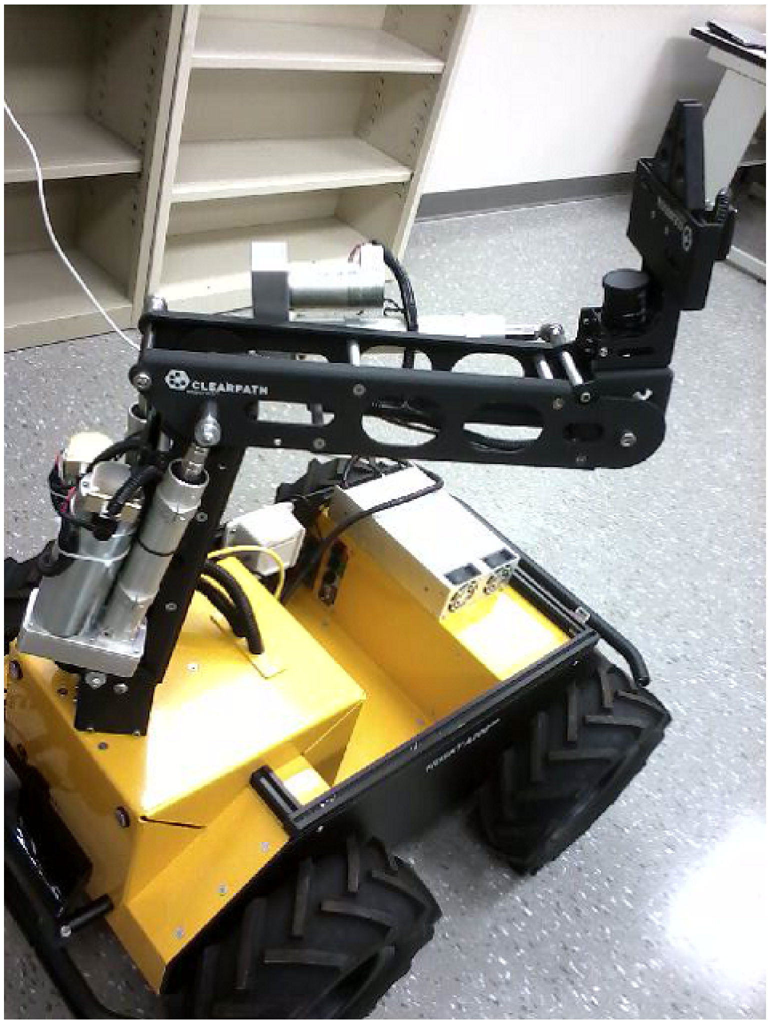


Figure 4.2: Manipulator used in experiments

Training trajectories were parameterized as in equation (3.3), using $N = 2$. Each training trajectory was 60 seconds long. The trajectories were generated by selecting random values for the parameters (a, b, q_0, ω_0). The trajectory parameter limits were chosen to ensure that the joint angles did not exceed the mechanical limits of the hardware. The waist joint angle was limited to a 90 degree range, which is significantly less than the mechanical limits of the waist joint. This was done to reduce the size of the state space, since the robot dynamics are not affected by the waist joint angle.

Chapter 5

Quadrotor Experiments

5.1 Modelling a Quadrotor using Gaussian Processes

From Section 2.1.2, the quadrotor angular and linear acceleration can be expressed as a mapping

$$(\mathbf{V}, h, \boldsymbol{\omega}) \mapsto (\mathbf{a}, \boldsymbol{\alpha}) \tag{5.1}$$

where $\mathbf{V}$ is a vector of the 4 control inputs (i.e. the 4 motor voltages), h represents the quadrotor height (required to account for ground effect), $\mathbf{a}$ is the linear acceleration and $\boldsymbol{\alpha} = \dot{\boldsymbol{\omega}}$ is the angular acceleration. To calculate the motor voltages $\mathbf{V}$ required to achieve a given linear acceleration $\mathbf{a}$ and angular acceleration $\boldsymbol{\alpha}$, equation (5.1) can be rearranged to the mapping

$$(h, \boldsymbol{\omega}, \mathbf{a}, \boldsymbol{\alpha}) \mapsto \mathbf{V} \tag{5.2}$$

This mapping represents the inverse dynamics of the quadrotor (including both the mechanical and the electrical system) and provides the motor voltage required to produce a given acceleration $\mathbf{a}$ and angular acceleration $\boldsymbol{\alpha}$ for a known quadrotor state $(h, \boldsymbol{\omega})$. GPR can be used to learn the mapping (5.2) for a specific quadrotor. Since there are 4 components in $\mathbf{V}$, a total of 4 separate GPs are needed (one for each component). It is usually convenient to use $\mathbf{V}$ to represent the 4 motor voltages, since $\mathbf{V}$ can then be used directly as a feedforward signal. However, it is possible to use other control inputs, such as the total thrust and the moment around each axis, or the four rotor speeds. The motor voltage is used here because it is the most convenient control signal for our experimental platform.

To learn the mapping (5.2), the training data should provide good coverage of the state space. If there is not enough training data, or if there are regions of the state space which do not have sufficient training data, the predictions made by GPR may be inaccurate [26]. Since the training data comes from actual flight data, the GP model can learn any state that the quadrotor achieves during flight by adding data from that state to the training data set.

5.2 Simulation Results

The trajectory optimization algorithm was initially tested in simulation, using the Matlab simulation system described in Section 4.1.1. The trajectories were parameterized as in equation (3.3), using $N = 2$. This value was chosen because it allows for a wide variety of trajectories without resulting in an excessive number of parameters to optimize. The trajectory was constrained to be within a 20m by 20m by 5m box. Each simulation was 30 seconds long ($T = 30$), and $\omega_0 = \frac{2\pi}{T}$ was used, so the simulation lasts for one period of the trajectory. After downsampling to 10 Hz, each trajectory generated 300 data points, which were added to the training data set. When computing the optimization metric, the trajectory was sampled at 10 Hz. The Gaussian Processes for Machine Learning Toolbox [35] was used to run GPR on the data. The trajectory was optimized using the `fmincon` function in Matlab [22]. The default active set optimization algorithm was used.

Gaussian Processes were trained using data from a number of simulated flights, and then tested on a different flight. The same testing flight was used throughout. The hyper-parameters were trained on the first training flight, and then kept constant. The training flights were parameterized as in equation (3.3); the parameters were either randomly generated or optimized as described above. The first training flight was the same for both the random trajectory and the optimized trajectory simulations. Fig. 5.1 shows the mean error (for the testing flight) between the predicted motor voltages and the actual motor voltages for varying numbers of training flights. For comparison, Fig. 5.1 also shows the mean prediction error for a mathematical model of the quadrotor inverse dynamics. This model has the exact values for the quadrotor parameters (mass, inertia, etc.) but ignores drag and ground effect, which are both included in the simulation. Fig. 5.2 shows the variance predicted by the GPs for varying numbers of training flights. It can be seen that using optimized trajectories for training results in reduced error and variance compared to using randomly generated trajectories. After enough training flights, the learned model also outperforms the conventional mathematical model of the inverse dynamics.

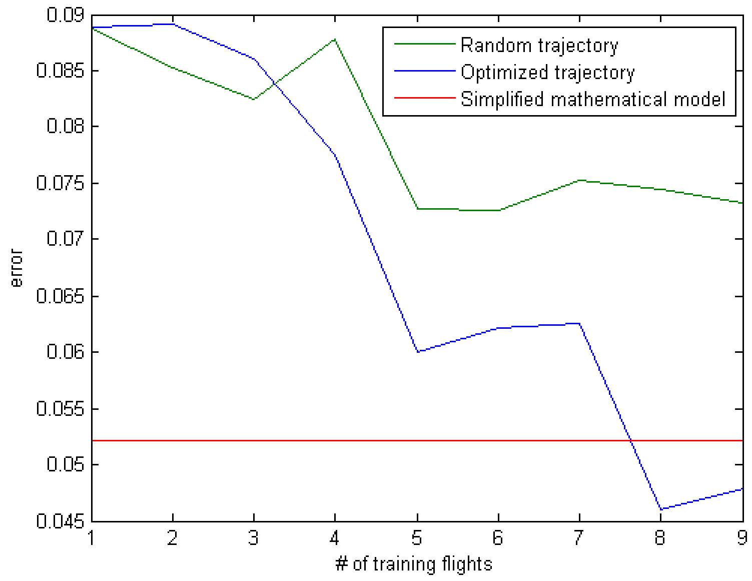


Figure 5.1: **Quadrotor**: Mean prediction error in simulation

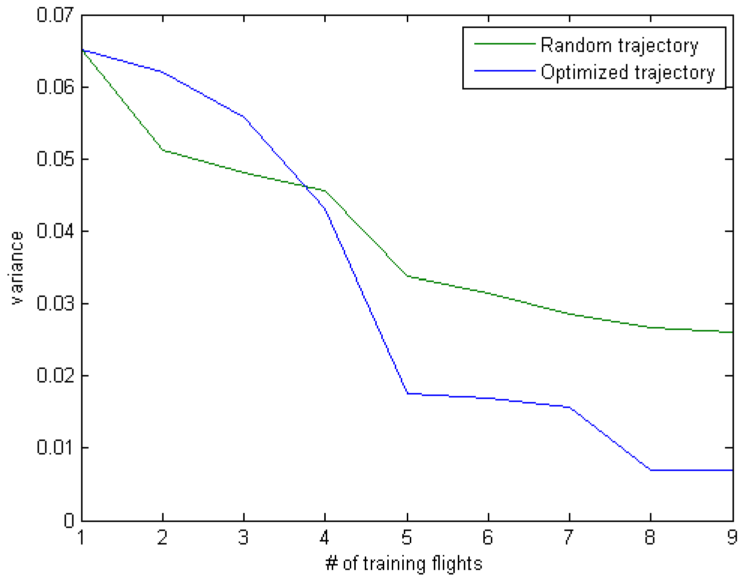


Figure 5.2: **Quadrotor**: Predicted variance in simulation

5.3 Experimental Results

The Pelican quadrotor described in Section 4.1.2 was used to test the trajectory optimization experimentally. The quadrotor flew both randomly generated trajectories and trajectories where the parameters were optimized as in equation ((3.4)). As in the simulations, $N = 2$ was used. Each trajectory was 30 seconds long, and $\omega_0 = \frac{2\pi}{T}$ was chosen, so one period of the trajectory corresponds to the length of the flight. The code used to run GPR and generate new trajectories was the same as the code used for the simulation data, with only a few small changes made to the constraints and to replace the simulation data with logged data from the Pelican. The `fmincon` function was used to optimize the trajectory parameters. The trajectory constraints were modified to restrict the trajectory to a cylinder with radius 1.5m and height 2m, and the quadrotor speed was limited to 1.5 m/s. These constraints were chosen to ensure that the quadrotor remained within the Optitrack capture space.

A number of flights were flown, using either randomly generated or optimized trajectories. Fig. 5.3 shows one of the trajectories flown and the desired trajectory. After each optimized trajectory was flown, data from that trajectory was added to the training set, and the entire training set was used to generate the next optimized trajectory. For the initial flight, the data from the entire flight (from takeoff to landing) was added to the training set. The hyperparameters were trained on this flight, and then kept constant. For subsequent flights, only data from the predefined trajectory was added to the training set; the takeoff and landing phases were discarded. This was done to reduce the amount of training data (to prevent GPR performance issues resulting from a large training set) and to ensure that the quality of the training set depended only on the trajectories flown by the quadrotor, and not on data from other phases of flight.

Figs. 5.4 and 5.5 show the error and variance, tested on the same flight, for varying numbers of training flights. The optimized trajectories consistently provide smaller error; the optimized trajectories initially have smaller variance as well, but the random trajectories provide similar variance when there are more than 3 training flights. Figs. 5.6 and 5.7 show the error and variance for varying numbers of training flights, tested on different flights for each iteration. Again, the optimized trajectories consistently provide smaller error; the optimized trajectories give smaller variance initially, but after multiple training flights, both the optimized and random trajectories give similar results. The difference between optimized and random trajectories is large for a small number of training flights, and shrinks as more training flights are added. This is expected; for a small number of training flights, the random trajectories are more likely to have 'gaps' where data is missing, which get filled in as more training data is added. When optimized training flights

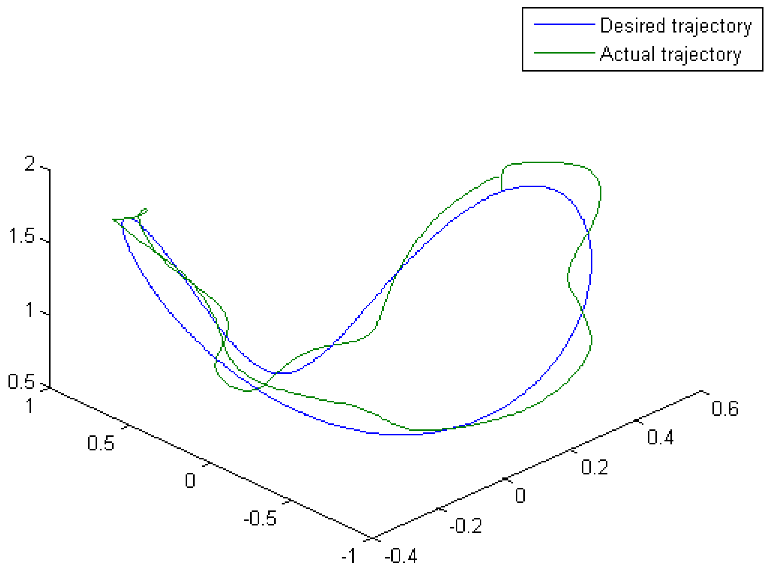


Figure 5.3: **Quadrotor**: Example of trajectory flown

are used, the training data will be more evenly distributed, even for a small number of training flights. Fig. 5.8 shows the mean and 1-σ confidence bounds predicted by the GP trained on 5 optimized training flights for a complete testing flight.

Due to limitations on the capture space and the quadrotor speed, the trajectories flown were quite conservative. If more aggressive trajectories were permitted, the state space would be larger and more training data would be needed, especially since many complex aerodynamic effects appear chiefly during high-speed flight. As a result, we would expect a bigger performance gain when using optimized training flights compared to random flights. The simulation results, which allowed more aggressive trajectories, showed a significant performance improvement when using optimized flights even after 9 training flights.

5.4 Summary

In experiments, using optimized trajectories resulted in improved predictions for a small number of training flights. After more training flights, the predictions made using a GP model trained on random trajectories provided similar performance compared to a GP model trained using optimized trajectories. In simulations, which allowed more aggressive

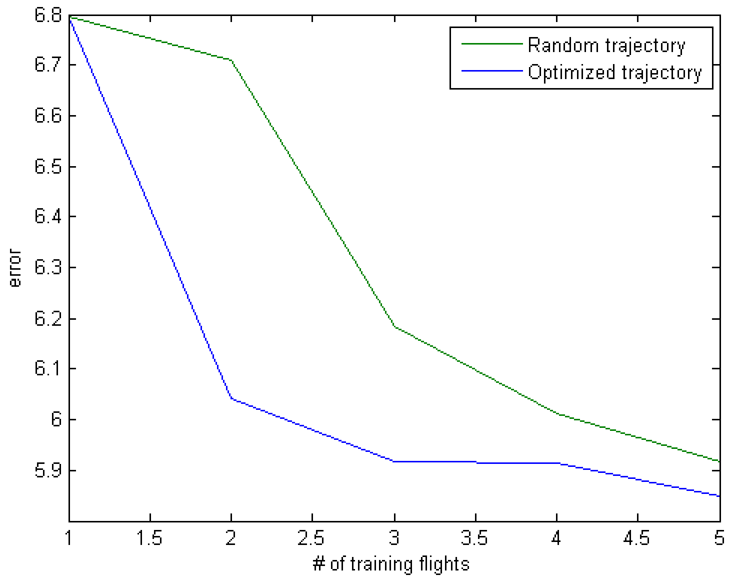


Figure 5.4: **Quadrotor**: Mean prediction error - same flight

trajectories, GP models trained on optimized trajectories provided better performance even after 9 training flights. A GP model trained on optimized trajectories also provided better performance than an inverse dynamics model of the quadrotor which used the exact values for the quadrotor parameters but neglected drag and ground effect.

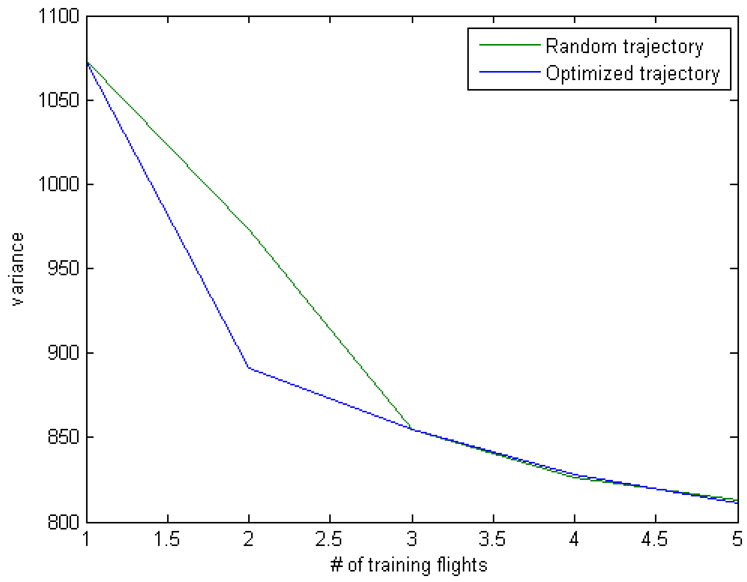


Figure 5.5: **Quadrotor**: Mean prediction variance - same flight

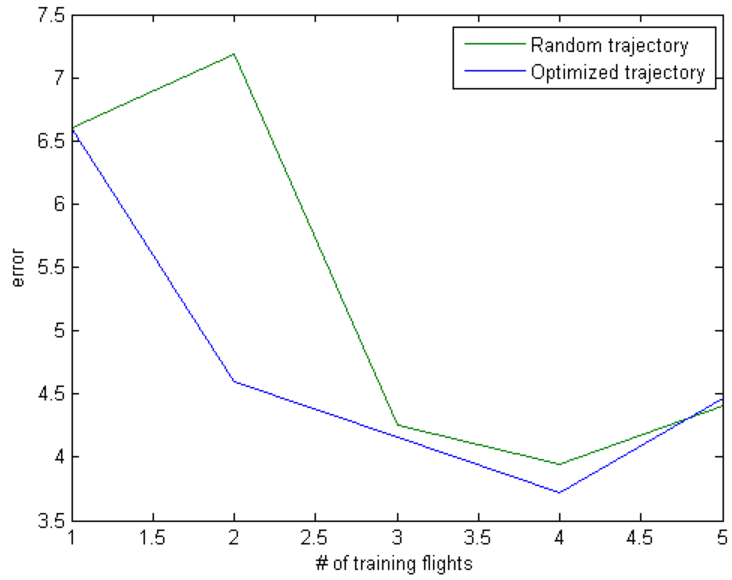


Figure 5.6: **Quadrotor**: Mean prediction error - random flights

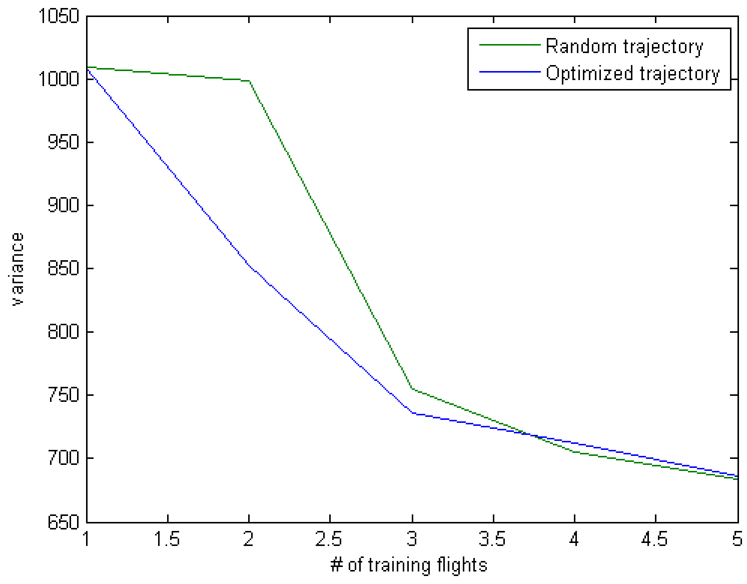


Figure 5.7: **Quadrotor**: Mean prediction variance - random flights

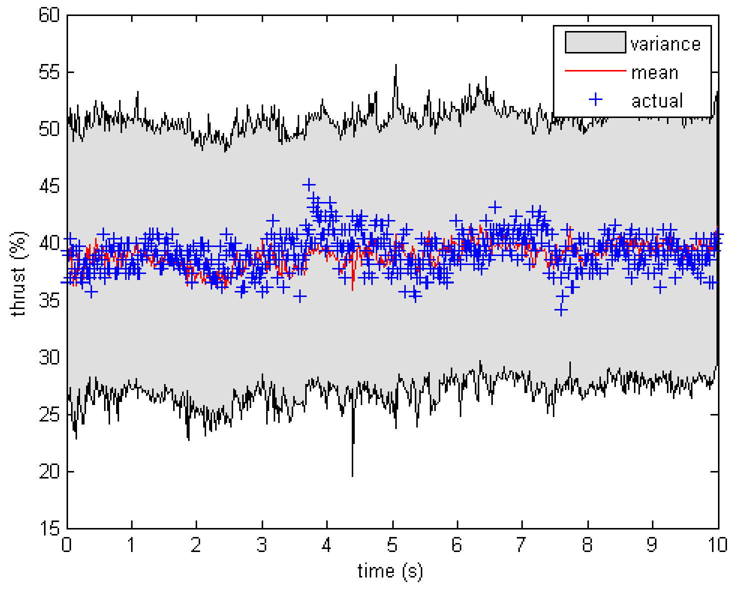


Figure 5.8: **Quadrotor**: GP thrust prediction and variance

Chapter 6

Manipulator Experiments

The trajectory optimization described in Chapter 3 was tested on the 4-DOF manipulator described in Section 4.2. The controller structure and trajectory parameterization were the same as in Chapter 3. The trajectory parameters (a, b, q_0, ω_0) were either chosen randomly or were optimized using equation (3.6). A separate trajectory, which was not used for training, was used to test the GP performance. The robot tracked the testing trajectory, using a PD feedforward controller; the commanded motor PWM was the sum of a feedforward signal computed by the learned model and a PD feedback signal. The tracking error between the desired and actual joint angles was used to evaluate the performance of the learned model. In Section 6.1, hyperparameter training for different GPR approximations is examined to determine the best way of training the hyperparameters. To test trajectory optimization, GPR using a dictionary based on prediction error and SSGP learning spectral points were used, as these were the two most accurate methods, based on the results from Sections 6.1 and 6.2. After each new trajectory, the hyperparameters were updated using the most effective method (based on the results from Section 6.1).

6.1 Hyperparameter Training

Hyperparameter training has a significant impact on the GP predictions, and it is important to train the hyperparameters well. A bad choice of hyperparameters can lead to the GP making inaccurate predictions, resulting in a poor model of the system and poor control. As suggested by Lázaro-Gredilla et al. [17], the lengthscales $\mathbf{\Lambda}$ are initialized to half the range of the corresponding input data, and the variance σ_0^2 is initialized to the variance of the output $\mathbf{y}$. The hyperparameters are then trained by using gradient descent to maximize the marginal likelihood. The number of steps of gradient descent is important; using more steps will increase the marginal likelihood, but may result in overfitting. When new data is added to the GP training set, there are 3 possible choices for the hyperparameters:

1. Keep the hyperparameters unchanged
2. Update the hyperparameters by using gradient descent, with the previous hyperparameters being used as the initial value
3. Reinitialize the hyperparameters and retrain them using gradient descent

6.1.1 GPR Dictionary

All three hyperparameter training approaches were tested by training a GPR Dictionary model on the 4-DOF manipulator. The training trajectories used were generated randomly by using random values for the parameterization in Equation (3.5). To evaluate the performance of the GPR Dictionary model, the model was used to control the manipulator while it tracked a testing trajectory. Figure 6.1 shows the mean error for all three hyperparameter training options. Table 6.1 shows the error after 4 training trajectories.

It is clear that the best option is to update the hyperparameters after each iteration, using the previous hyperparameters as the initial starting point. This result is somewhat counterintuitive, since it would be expected that reinitializing the hyperparameters would provide better results than using hyperparameters which were trained on a previous version of the dictionary. However, the dictionary is updated based on predictions made using the previous set of hyperparameters. Significantly altering the hyperparameters by reinitializing them will change these predictions and may result in poor predictions on new trajectories. Table 6.2 shows the some of the hyperparameters (the lengthscales $\mathbf{\Lambda}$) after 5 training trajectories. The original hyperparameters are the hyperparameters used when updating the dictionary; the updated and reinitialized hyperparameters were trained on the updated dictionary. This table shows that the updated hyperparameters remain close to the original hyperparameters, while the reinitialized hyperparameters are significantly different.

Figure 6.2 shows how changing the number of iterations of gradient descent used in hyperparameter training affects the results. In both cases, the hyperparameters are updated (using the previous hyperparameters as the initial point for gradient descent) after each new trajectory. Table 6.3 shows the tracking error on the testing trajectory after 6 training

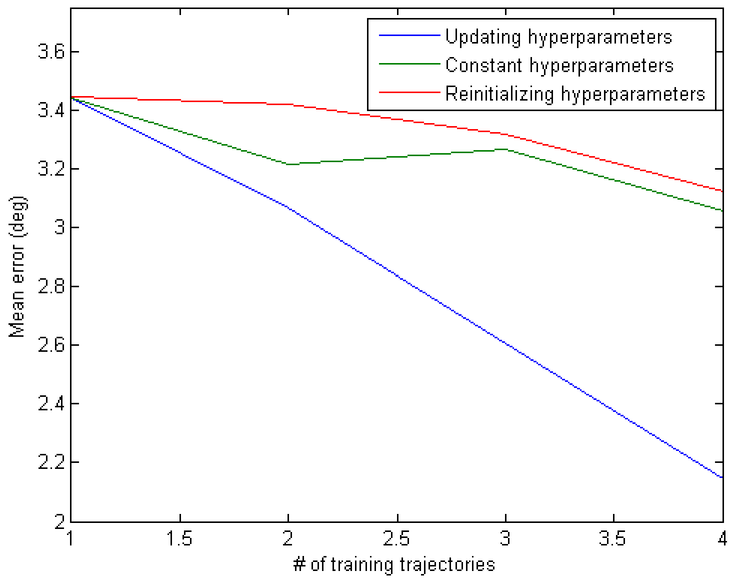


Figure 6.1: Hyperparameter Training - GPR Dictionary

	Waist	**Shoulder**	**Elbow**	**Wrist**
Constant Hyperparameters	3.0163	3.3312	2.3296	3.5595
Updating Hyperparameters	2.8870	1.1887	1.6083	2.8942
Reinitializing Hyperparameters	3.1144	2.3324	2.3498	4.6988

Table 6.1: Hyperparameter Training - GPR Dictionary - Error after 4 training trajectories

Original	8.0998	0.3663	2.0573	9.4783	1.6597	0.0454	0.6021	0.8885	0.4067	0.0147	0.3425	0.1622
Updated	9.1369	0.3650	1.8463	10.5568	1.5779	0.0466	0.5835	1.0052	0.4182	0.0146	0.3568	0.1693
Reinitialized	0.7281	0.4464	0.4617	0.4589	0.1393	0.1073	0.1150	0.0850	0.0463	0.0260	0.0380	0.0323

Table 6.2: Covariance function lengthscales

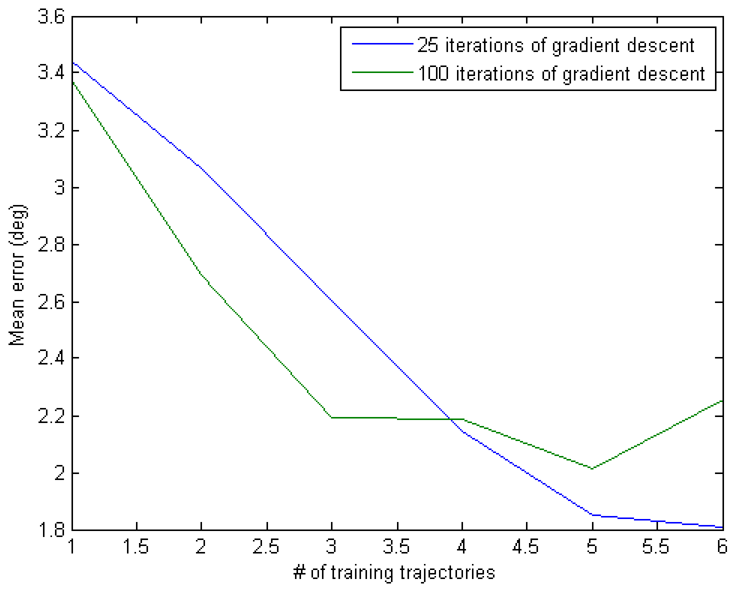


Figure 6.2: GPR Dictionary hyperparameter training - # of iterations of gradient descent

	Waist	**Shoulder**	**Elbow**	**Wrist**
25 steps of gradient descent	1.8864	0.7971	1.8133	2.7343
100 steps of gradient descent	3.1652	1.3563	1.8457	2.6520

Table 6.3: GPR Dictionary hyperparameter training - tracking error after 6 trajectories

trajectories. Using a large number of steps of gradient descent initially provides better results, but after 4 trajectories, using a smaller number of steps provides better performance. When data is only available for a few trajectories, using a large number of steps of gradient descent is expected to provide hyperparameters that more closely match the training data; however, after multiple trajectories (and multiple hyperparameter updates), using a large number of steps of gradient descent may result in overfitting. Also, using a large number of steps of gradient descent may result in hyperparameters that differ significantly from the original hyperparameters; as with reinitializing hyperparameters, this can result in poorer performance.

	Waist	**Shoulder**	**Elbow**	**Wrist**
5 steps of gradient descent	2.4109	1.4520	2.3456	2.7366
25 steps of gradient descent	2.9890	2.6489	2.5656	3.9166

Table 6.4: SSGP (learning spectral points) hyperparameter training - tracking error after 4 training trajectories

	Waist	**Shoulder**	**Elbow**	**Wrist**
Constant hyperparameters	3.3134	2.7900	2.3796	2.6331
Updating hyperparameters	3.3049	2.7791	2.4372	2.7643
Reinitializing hyperparameters	2.4109	1.4520	2.3456	2.7366

Table 6.5: SSGP (learning spectral points) hyperparameter training - tracking error after 4 training trajectories

6.1.2 SSGP

The SSGP implementation by Lázaro-Gredilla et al. [17] was used to train a SSGP model of the 4-DOF manipulator. SSGP was found to be highly sensitive to the number of steps of gradient descent used. Initially, 25 steps of gradient descent were used after each trajectory to retrain the hyperparameters; this was found to lead to overfitting. The motion when tracking a testing trajectory tended to be very jerky, as shown in Figure 6.3. Figure 6.4 shows the PWM predicted by the SSGP and the applied PWM (the sum of the SSGP prediction and the PD feedback signal). The SSGP prediction is clearly inaccurate. When only 5 steps of gradient descent were used for hyperparameter training, the performance improved significantly. Figure 6.5 shows the tracking error for both 5 and 25 steps of gradient descent. Table 6.4 shows the error after 4 training trajectories when using either 5 steps of gradient descent or 25 steps of gradient descent.

Figure 6.6 shows the mean error for the different hyperparameter training options. Table 6.5 shows the error after 4 training iterations. These results show that for SSGP, the best hyperparameter training approach is to reinitialize and retrain the hyperparameters after each new training trajectory. This result is expected, since retraining the hyperparameters from scratch is the best way to ensure that the hyperparameters reflect the training data.

SSGP using fixed spectral points shows similar results to learning spectral points. Figures 6.7 and 6.8 show how changing the number of iterations of gradient descent and changing the hyperparameter training method affect the performance of the model. Tables

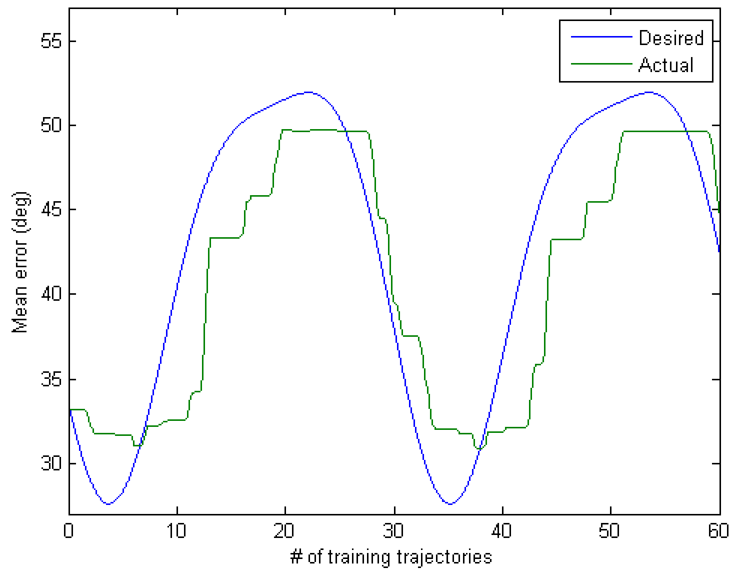


Figure 6.3: SSGP (learning spectral points) - Wrist angle for testing trajectory

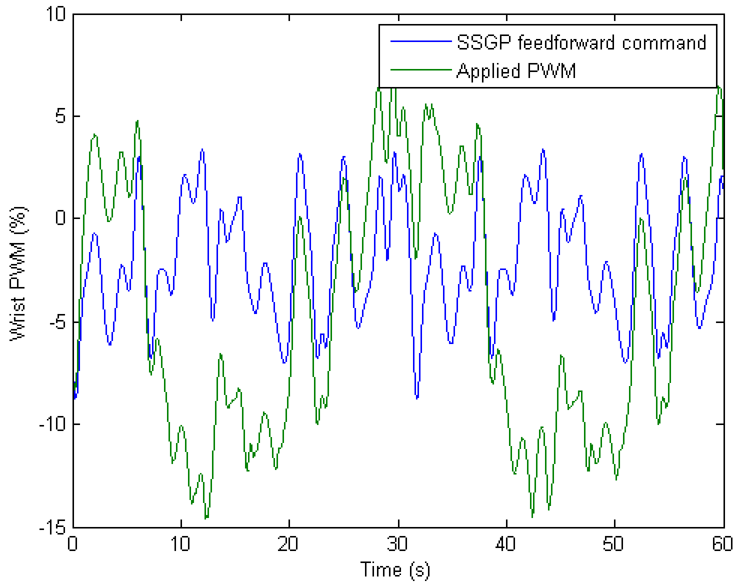


Figure 6.4: SSGP (learning spectral points) - Wrist PWM for testing trajectory

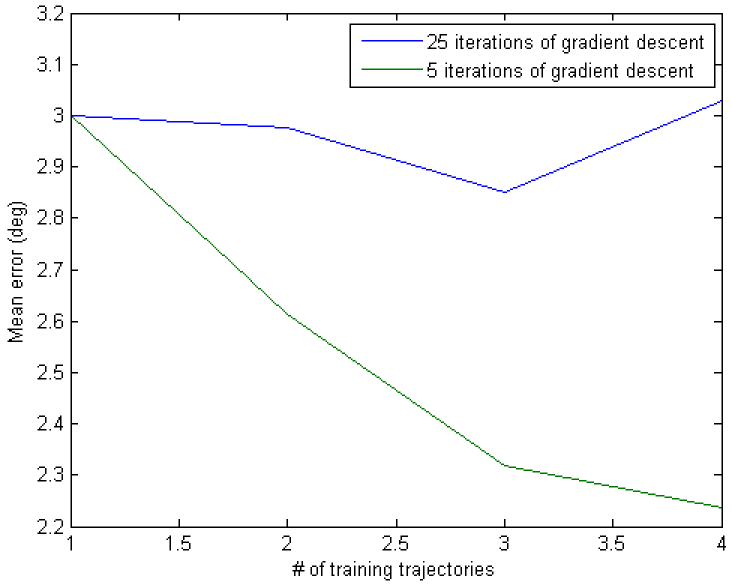


Figure 6.5: SSGP (learning spectral points) hyperparameter training - # of iterations of gradient descent

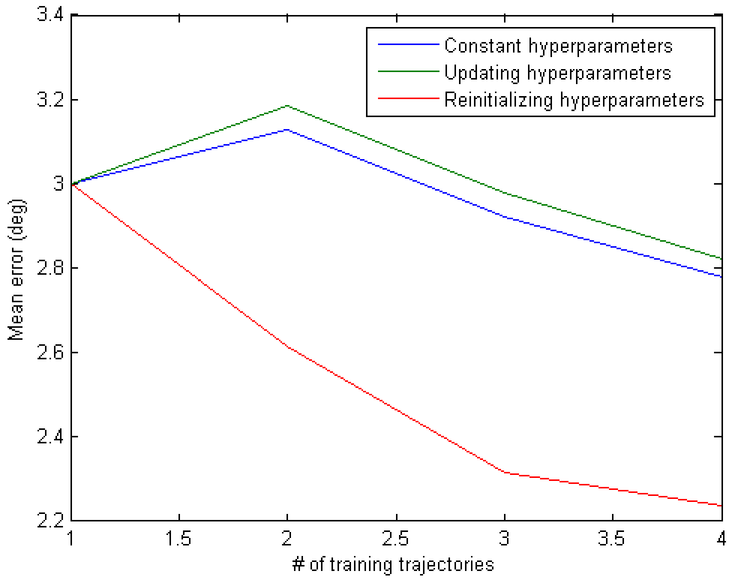


Figure 6.6: SSGP (learning spectral points) hyperparameter training

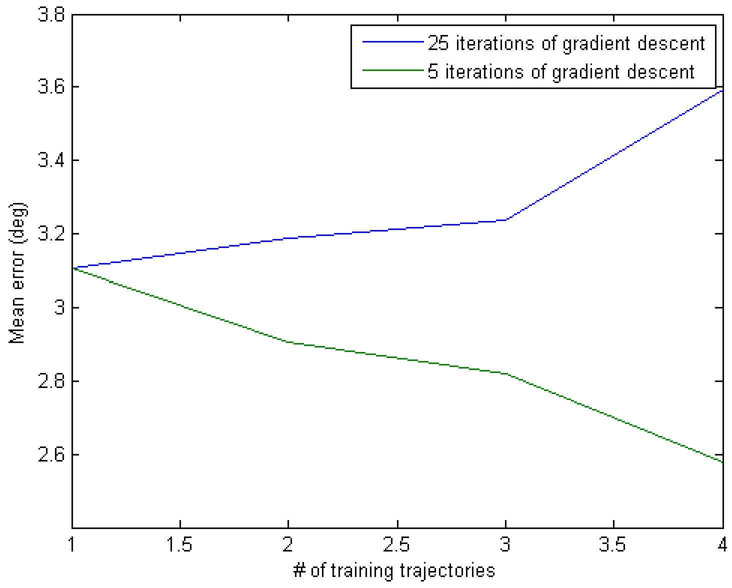


Figure 6.7: SSGP (fixed spectral points) hyperparameter training - # of iterations of gradient descent

	Waist	**Shoulder**	**Elbow**	**Wrist**
5 steps of gradient descent	2.2442	1.7233	2.2816	4.0551
25 steps of gradient descent	3.8244	2.5761	3.2124	4.7661

Table 6.6: SSGP (fixed spectral points) hyperparameter training - tracking error after 4 training trajectories

6.6 and 6.7 show the mean error after 4 training trajectories. As with SSGP learning spectral points, the performance is highly sensitive to the number of iterations of gradient descent used in hyperparameter training, and overtraining the hyperparameters leads to poor performance. The best results come from reinitializing and retraining the hyperparameters after each new trajectory.

Figure 6.9 compares learning and fixed spectral points. Learning the spectral points consistently provides better results. Table 6.8 shows the mean tracking error after 4 training trajectories. The main difference between learning and fixed spectral points comes when learning the wrist joint; after 4 training trajectories, learning the spectral points results in a mean tracking error of 2.7366 deg/s, while using fixed spectral points results in a mean

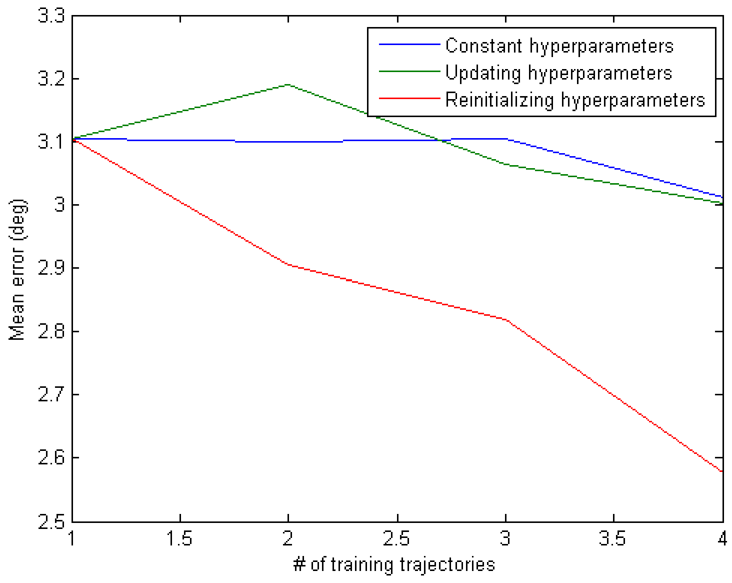


Figure 6.8: SSGP (fixed spectral points) hyperparameter training

	Waist	**Shoulder**	**Elbow**	**Wrist**
Constant hyperparameters	4.3234	2.4421	2.3767	2.9085
Updating hyperparameters	4.1632	2.5026	2.3425	3.0020
Reinitializing hyperparameters	2.2442	1.7233	2.2816	4.0551

Table 6.7: SSGP (fixed spectral points) hyperparameter training - tracking error after 4 training trajectories

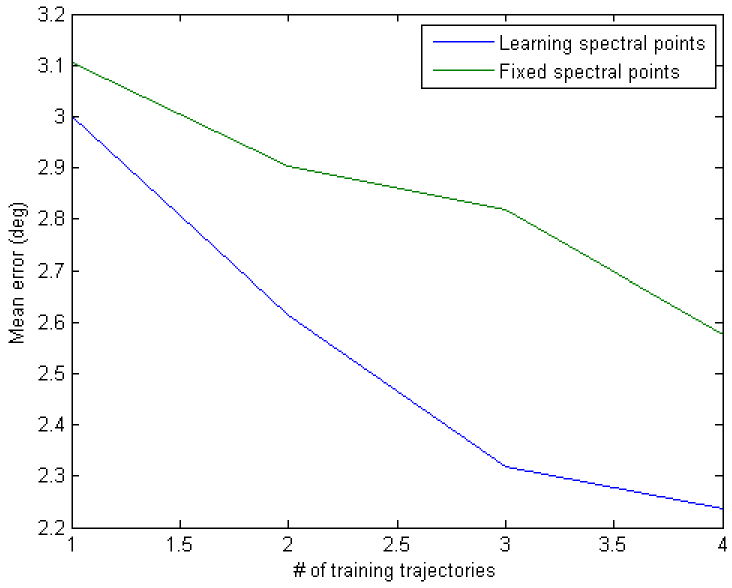


Figure 6.9: SSGP - comparison of learning and fixed spectral points

	Waist	**Shoulder**	**Elbow**	**Wrist**
Learning spectral points	2.4109	1.4520	2.3456	2.7366
Fixed spectral points	2.2442	1.7233	2.2816	4.0551

Table 6.8: SSGP - comparison of learning and fixed spectral points

tracking error of 4.0551 deg/s. The tracking error for the remaining joints is similar for both fixed and learned spectral points.

6.2 GPR Dictionary

The GPR dictionary approach described in Section 2.5 was compared to the simpler approach of downsampling the data from each trajectory and adding the downsampled data to the training set. Dictionaries using both variance and prediction error were compared to downsampling.

Each trajectory was generated randomly, and was 60 seconds long. The same random trajectories were used for all 3 GPR models. The controller ran at 100 Hz. Each trajec-

Joint angle error (deg)	**Waist**	**Shoulder**	**Elbow**	**Wrist**
Downsampling	0.9650	1.8155	1.3027	3.5596
Dictionary (variance)	1.1536	2.5569	1.5540	2.9129
Dictionary (error)	1.8864	0.7971	1.8133	2.7343

Table 6.9: Joint angle error (deg) after 6 training trajectories

tory generated 3000 data points. Running GPR using multiple trajectories for training is impractical, due to the amount of time required for both hyperparameter training and prediction. When downsampling was used, the data was downsampled by a factor of 12, resulting in an effective sampling rate of 8.333 Hz. This reduces the data set size to 500 points per trajectory, making it possible to run full GPR on the data. Even in this case, computational speed becomes an issue after multiple trajectories. For the dictionary, a dictionary size of 500 points was used. After each new trajectory, the dictionary was updated to minimize either the variance or the prediction error when making predictions on points from the new trajectory.

Table 6.9 shows the mean joint angle error after a total of 6 training trajectories. Figure 6.10 shows how the joint angle error changes as more training data is added.

Using a GPR dictionary based on minimizing error provides better results than a dictionary based on minimizing variance. The error-based dictionary provides results that are comparable to using full GPR (with downsampling), but the dictionary requires much less training data (500 data points vs. 3000 data points) and as a result is less computationally intensive. Adding additional data (for example, during online learning) to the full GPR model is difficult, because of computational issues resulting from the size of the training set. New data can easily be added to the dictionary, because the overall training data set size is fixed.

6.3 Trajectory Optimization

To test the trajectory optimization described in Chapter 3, GP models were trained using both optimized and random trajectories and used to compute feedforward signals for the 4-DOF manipulator. An error-based dictionary and SSGP were both used to evaluate the trajectory optimization.

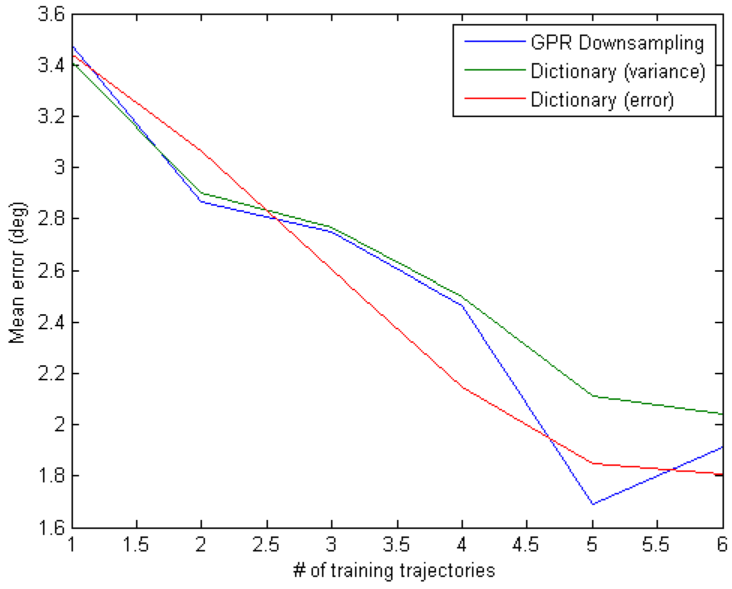


Figure 6.10: Comparison of joint angle error using GPR Downsampling, Dictionary-Error and Dictionary-Variance

Joint angle error (deg)	**Waist**	**Shoulder**	**Elbow**	**Wrist**
Optimizing trajectories	0.8072	1.1581	0.8136	1.3627
Random trajectories	1.0914	1.0061	0.9235	1.0972

Table 6.10: **Robot**: GPR Dictionary (error) with 500 points - Error

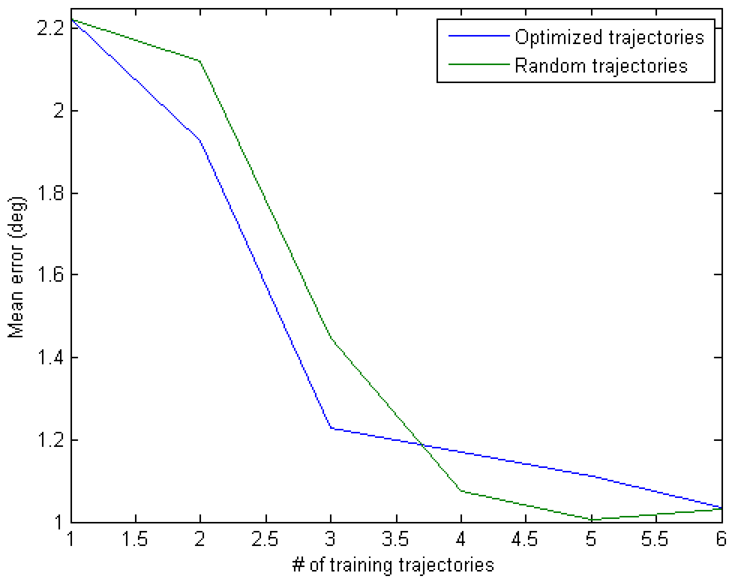


Figure 6.11: **Robot**: GPR Dictionary (error) with 500 points - Mean joint angle error

6.3.1 GPR Dictionary

A dictionary containing 500 points was used as the training data for GP predictions. The points in the dictionary were selected from the entire training set to minimize the error between the GP prediction and the actual motor PWM.

Table 6.10 shows the mean joint angle error over the testing trajectory after a total of 6 training trajectories. Fig. 6.11 shows how the mean joint angle error over the testing trajectory changes as more training trajectories are added. Table 6.11 shows the variance after 6 training trajectories when making predictions for the testing trajectory. Fig. 6.12 shows how the variance over the testing trajectory changes as more training data is added.

The results show that using optimized trajectories results in improved performance

GPR Variance	**Waist**	**Shoulder**	**Elbow**	**Wrist**
Optimizing trajectories	26.9634	18.2700	44.0714	17.1138
Random trajectories	33.5733	33.6396	27.4302	28.2919

Table 6.11: **Robot**: GPR Dictionary (error) with 500 points - variance

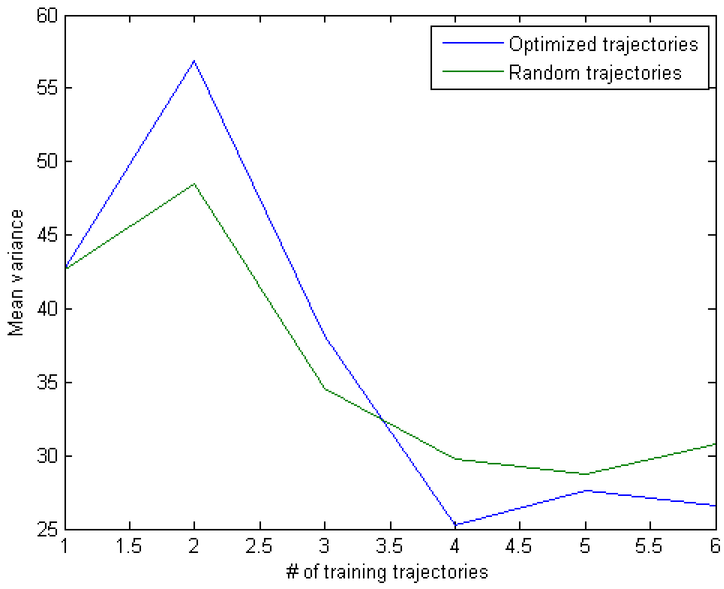
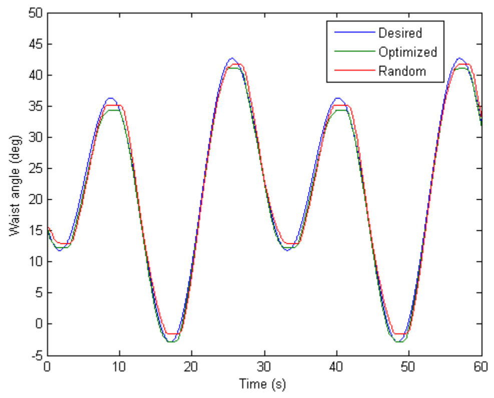
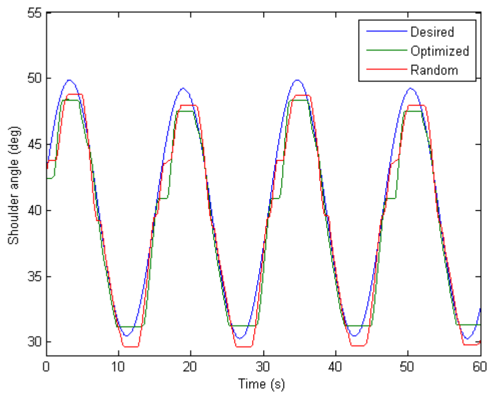


Figure 6.12: **Robot**: GPR Dictionary (error) with 500 points - variance

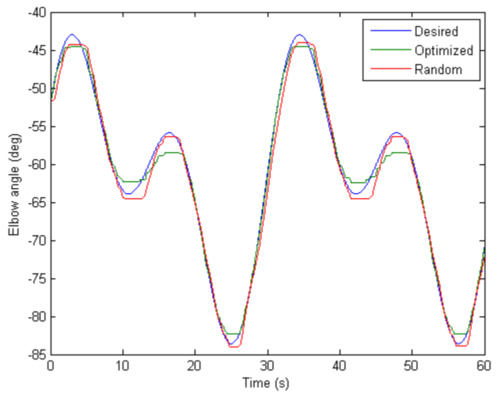
]



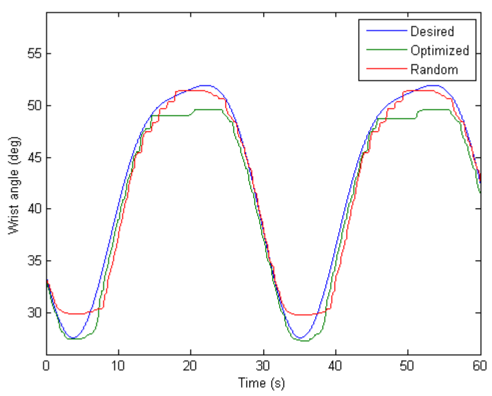
(a) Waist



(b) Shoulder



(c) Elbow

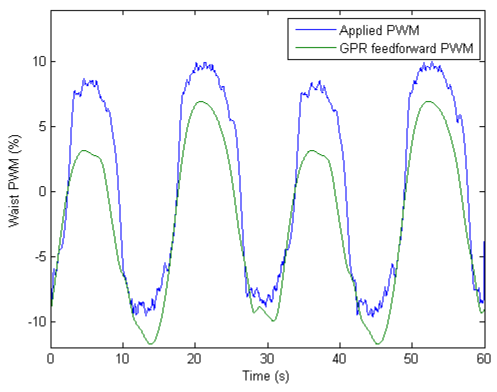


(d) Wrist

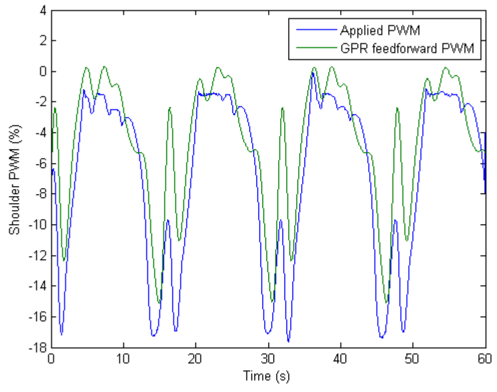
Figure 6.13: **Robot**: GPR Dictionary (error) with 500 points - joint angle trajectory

initially; after 4 trajectories, both random and optimized trajectories give similar performance. Also, after 4 trajectories, the mean tracking error decreases relatively slowly, and it would appear that adding further training data will not result in significant performance improvements.

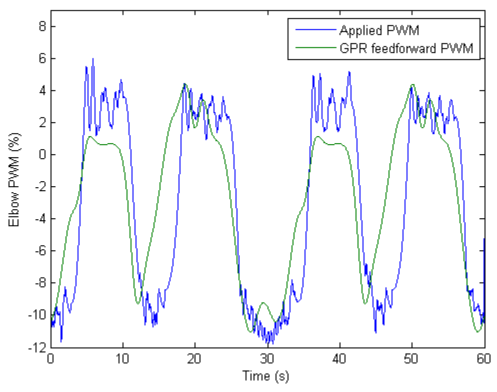
Figure 6.13 shows the desired joint position trajectory and the actual joint positions after training on a total of 6 trajectories. The performance is similar for the shoulder and elbow joints; for the waist joint, the optimized trajectories provide slightly better results, and the random trajectories provide better results for the wrist joint. For all joints, the error occurs mainly when the joint reverses direction. For the wrist joint, there is also a significant amount of undershoot at the maxima and minima. The optimized trajectories



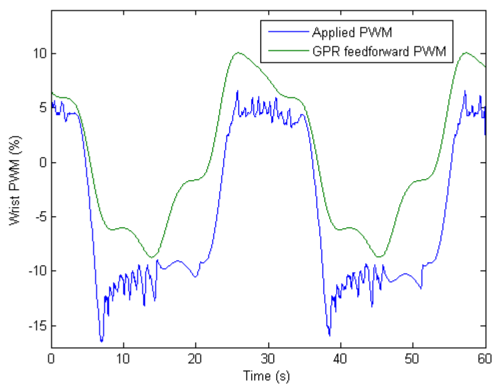
(a) Waist



(b) Shoulder



(c) Elbow



(d) Wrist

Figure 6.14: **Robot**: GPR Dictionary (error) with 500 points - motor PWM

Joint angle error (deg)	**Waist**	**Shoulder**	**Elbow**	**Wrist**
Optimizing trajectories	1.1818	1.9723	1.4067	1.6444
Random trajectories	1.1753	2.4789	0.8647	1.6112

Table 6.12: **Robot**: SSGP (learning spectral points) with 200 spectral points

provide good tracking at the minima, but significant undershoot at the maxima, while the random trajectories provide good tracking at the maxima, but undershoot at the minima. Figure 6.14 shows the feedforward PWM command from the GP model (trained on optimized trajectories), and the actual PWM applied (the sum of the feedforward PWM command and the feedback PD signal), for the optimized training trajectories. For the shoulder joint, which has the smallest error, the feedforward command is close to the actual applied PWM (i.e. the feedback signal is small). For the waist and elbow joints, the feedback signal is usually small, but there are points where the feedback signal is significant. For the wrist joint, which has the largest error, the applied PWM is significantly different from the GPR prediction; this indicates that the feedback signal for this joint is quite large compared to the GPR prediction.

6.3.2 SSGP

SSGP, using 200 spectral points, was used to learn a model of the manipulator. After each iteration, both the spectral points and hyperparameters were trained to fit the training data. Table 6.12 shows the mean error after 6 training trajectories. Figures 6.15 and 6.16 plot the mean joint angle error and variance vs. the number of training trajectories.

Using optimized trajectories results in better performance than using random trajectories for training. For both random and optimized trajectories, the tracking error sometimes increases after new training data is added. This results from reinitializing the hyperparameters after each trajectory; this can cause significant changes in the hyperparameters, which in turn will affect the predictions made by the SSGP model. Using constant hyperparameters or using the previous hyperparameters as the initial starting point for hyperparameter training will result in the error decreasing more smoothly, but at a slower rate, as was observed in Section 6.1.

Figure 6.17 shows the desired joint position trajectory and the actual joint positions after training on a total of 6 trajectories. As before, the error occurs mainly when the joint reverses direction. For the waist and elbow joints, the performance is similar for both SSGP models, although the random model seems to provide slightly better performance

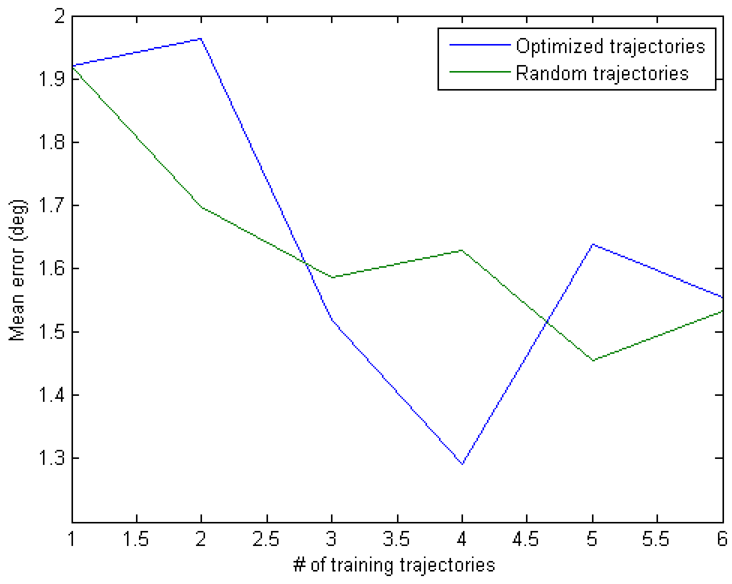


Figure 6.15: **Robot**: SSGP (learning spectral points) with 200 spectral points - Mean joint angle error

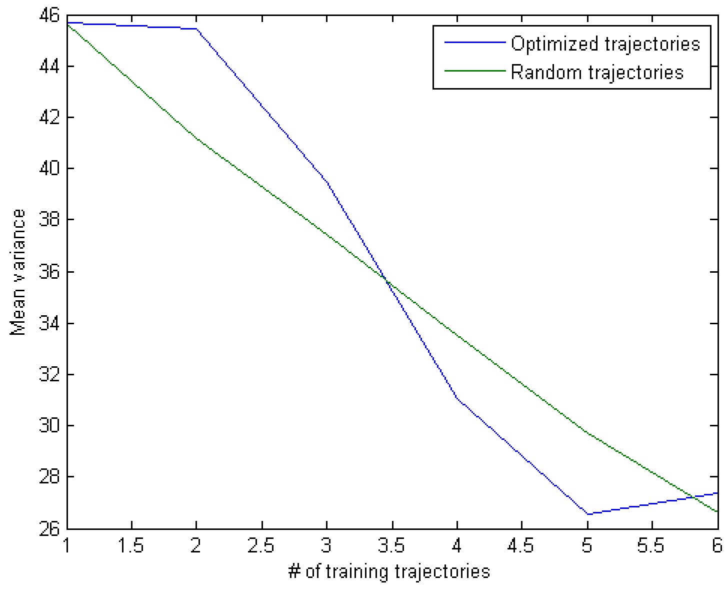
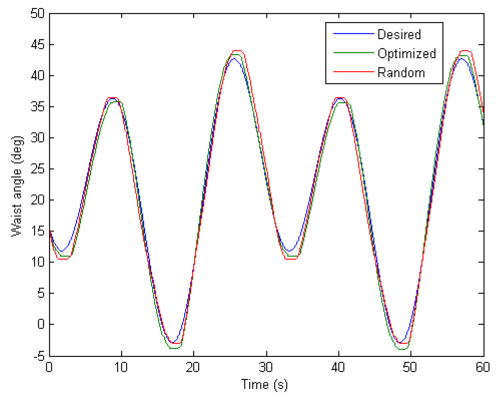
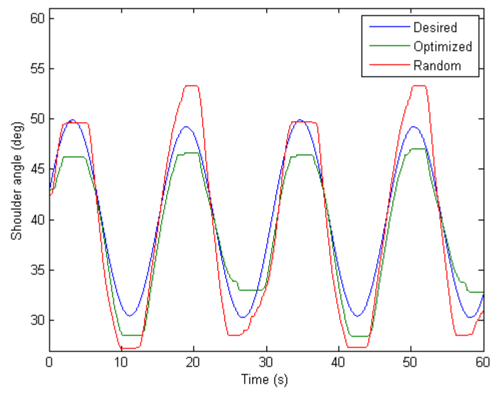


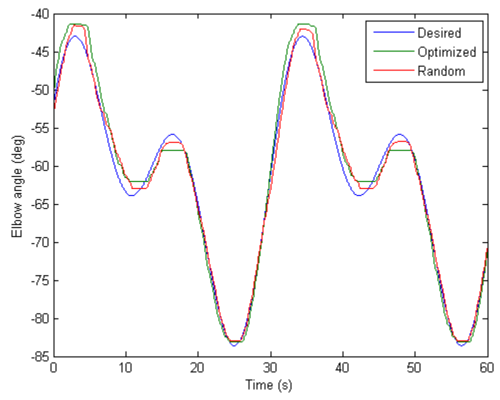
Figure 6.16: **Robot**: SSGP (learning spectral points) with 200 spectral points - variance



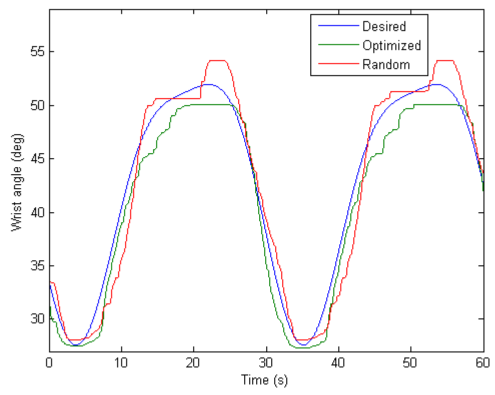
(a) Waist



(b) Shoulder



(c) Elbow



(d) Wrist

Figure 6.17: **Robot**: SSGP (learning spectral points) with 200 spectral points - joint angle trajectory

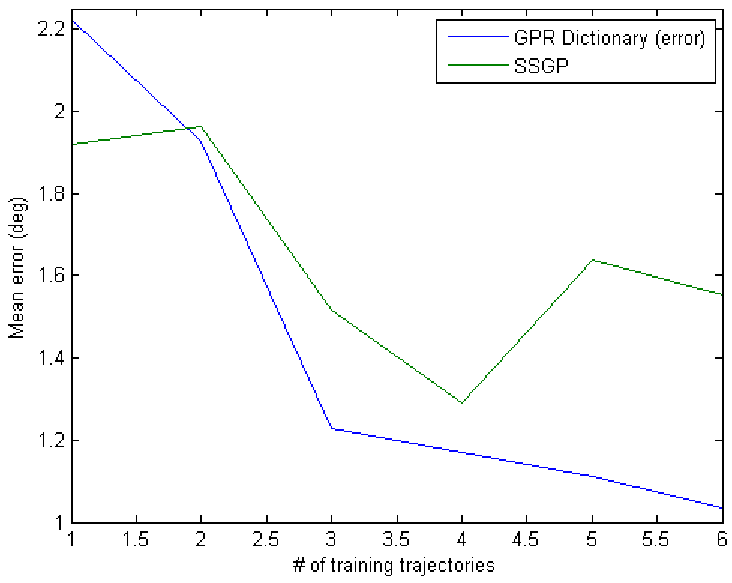


Figure 6.18: Mean joint angle error for GPR Dictionary and SSGP

for the elbow joint. For the shoulder and waist joints, there is significant error, as well as noticeable differences between the optimized and random models. For the shoulder joint, the random model has significant overshoot at the maxima and minima; the optimized model has undershoot at the maxima, and displays both overshoot and undershoot at different minima. For the wrist joint, the random trajectory has a noticeable spike at both maxima. This spike occurs because the applied voltage is initially too low to overcome friction; the applied voltage then increases, causing the joint to start moving.

Figure 6.18 compares the results for the GPR Dictionary and SSGP. Optimized trajectories are used to train both the SSGP and dictionary models. SSGP provides better initial performance (after only 1 training trajectory), but after this, the dictionary consistently outperforms SSGP. For the dictionary model, the tracking error decreases after each new trajectory; for the SSGP model, the tracking error increases after some of the trajectories. This is due to the difference in how the hyperparameters are trained; as described above, the SSGP hyperparameters were reinitialized and retrained after each trajectory, which has a significant impact on the SSGP predictions.

6.4 Summary

Dictionaries selected using both variance and error were compared to using downsampling to reduce the size of the training set. A dictionary based on error was found to provide better performance than a dictionary based on variance, and performance comparable to downsampling. The dictionary has the additional advantages of having a much smaller training set, and being able to perform online learning.

The hyperparameters used were found to have a significant impact on the predictions made by the GP models, and the method used for retraining the hyperparameters when new data is added was found to have a significant impact on performance. For the dictionary approach, updating the hyperparameters by using gradient descent starting from the previous hyperparameters was found to be the best approach. For SSGP, the best approach for hyperparameter training was to reinitialize the hyperparameters and retrain them using gradient descent. SSGP was also found to be extremely sensitive to overtraining the hyperparameters, and using too many iterations of gradient descent resulted in very bad performance.

For both the dictionary approach and SSGP, models trained using optimized trajectories were compared to models trained using random trajectories. In both cases, using optimized trajectories provided better performance when there was limited training data, and as more training data was collected the performance of both the optimized and random models was similar. The dictionary approach provided better performance than SSGP, and also provided a smoother decrease in the tracking error. When using the SSGP model, the error sometimes increased after the model was updated; this did not occur with the dictionary model.

Chapter 7

Conclusions & Recommendations

This thesis proposed a method for systematically collecting training data to learn the model of a system using GPR. Using full GPR (with downsampling to reduce the amount of training data) was compared to various GPR approximations. Different hyperparameter training approaches were tested to find the best way of training hyperparameters. Simulations were performed using a model of a quadrotor. Experiments were carried out on both a quadrotor and a 4-DOF manipulator.

For both the quadrotor and manipulator, collecting training data by optimizing trajectories to maximize variance was found to provide better results than using random trajectories. Using optimized trajectories caused the error to decrease at a faster rate and, for the quadrotor, resulted in smaller error even after a number of training trajectories. In simulation, the learned GPR model was found to provide more accurate predictions of the quadrotor motor voltage than a simplified mathematical model of the quadrotor.

Using a dictionary containing a subset of the available training data was found to provide similar performance to using all the available training data. The dictionary has the advantage of significantly reducing the computation time required to train the hyperparameters and to make predictions. The best method for selecting points to add to the dictionary is to choose points that have a large prediction error. An alternative method is to select points with a large variance; however, this method results in poor performance compared to using prediction error. The dictionary also provides better performance than SSGP. When using SSGP, learning the spectral points provides slightly better performance than using fixed spectral points.

SSGP was observed to be highly sensitive to the number of iterations of hyperparameter training. When training an SSGP model, care has to be taken to avoid overfitting as a result of overtraining the hyperparameters. For SSGP, the most effective method of training the hyperparameters was found to be reinitializing and retraining the hyperparameters after each trajectory. When using a dictionary, the most effective hyperparameter training method is to retrain the hyperparameters using the previous hyperparameters as the initial starting point.

Hyperparameter training was observed to have a significant impact on the predictions made by the GP models. In particular, retraining or updating the hyperparameters after new training data was incorporated resulted in better performance than keeping the hyperparameters constant.

7.1 Future Work

7.1.1 Further Experiments

More experimental work is needed to study using GPR models to control a quadrotor. In this thesis, the GPR model was shown to accurately predict the quadrotor control signal, but the model was never used in the quadrotor control loop. The learned GPR model should be used in the control loop to compute the quadrotor control signal; this would be expected to reduce the tracking error between the quadrotor position and the desired position. Additionally, different GPR approximations (such as Dictionary and SSGP) should be compared to see which works best when learning the quadrotor model. The manipulator results from Chapter 6 suggest that using a dictionary is the best approach, but this needs to be verified on the quadrotor.

7.1.2 Other GPR approximations

There are a number of other approximations of GPR which were not tested in this thesis. In particular, Sparse Pseudo-Input Gaussian Processes (SPGP) [39] and Sparse Online Gaussian Processes (SOGP) [6] have been used to model and control manipulators [9].

7.1.3 Other platforms

In this thesis, GPR was used to model a quadrotor and a 4-DOF manipulator. Since GPR can be used to learn the model of a wide range of systems, the results from this thesis can be extended to other systems, such as humanoid robots. The 4-DOF manipulator used in this thesis also has a wheeled base. This base was held stationary, but it would be possible to learn a combined model of the mobile maniplator which includes both the manipulator and the wheeled base.

7.1.4 Online Learning

Another direction for future work is to perform learning online, while the system is active. The work in this thesis focused on tracking a trajectory for a fixed period of time, then updating the learned model with data from the latest trajectory. An alternate approach is to incrementally update the model with data from the last timestep. For GPR-based models, online learning involves updating the training set with new data points representing the current system state and control inputs. The advantage of online learning is that the learned model is continuously updated, allowing it to adjust to changes in the system (for example, due to wear), as well as allowing the model to learn about areas of the state space which were not adequately covered during the initial training. Of the learning methods discussed in this thesis, online learning cannot be performed when using full GPR, because the training data set will eventually become too big for computations to be made in a reasonable amount of time. Both the dictionary approach and SSGP can be used for online learning. The dictionary approach has a constant training set size, and the computational complexity of the sparse approximations does not depend on the number of training points.

References

[1] asctec_drivers - ros wiki. http://wiki.ros.org/asctec_drivers.

[2] C.H. An, C.G. Atkeson, J. Griffiths, and J.M. Hollerbach. Experimental evaluation of feedforward and computed torque control. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 165–168, 1987.

[3] C.H. An, C.G. Atkeson, and J.M. Hollerbach. Estimation of inertial parameters of rigid body links of manipulators. In *Decision and Control, 1985 24th IEEE Conference on*, pages 990–995, Dec 1985.

[4] Brian Armstrong-Hélouvry, Pierre Dupont, and Carlos Canudas De Wit. A survey of models, analysis tools and compensation methods for the control of machines with friction. *Automatica*, 30(7):1083 – 1138, 1994.

[5] Ko Ayusawa, Gentiane Venture, and Yoshihiko Nakamura. Identifiability and identification of inertial parameters using the underactuated base-link dynamics for legged multibody systems. *The International Journal of Robotics Research*, 33(3):446–468, 2014.

[6] Lehel Csato and Manfred Opper. Sparse online gaussian processes. *Neural Computation*, 14:641–668, 2002.

[7] J. Peters D. Nguyen-Tong. Model learning in robotics: a survey. *Cognitive Processing*, 12(4), 2011.

[8] J. Sun de la Cruz, W. Owen, and D. Kulić. Online learning of inverse dynamics via gaussian process regression. In *IEEE International Conference on Intelligent Robots and Systems*, pages 3583–3590, 2012.

[9] Joseph Sun de La Cruz. Learning inverse dynamics for robot manipulator control. Master's thesis, University of Waterloo, 2011.

[10] C.C. de Wit, P. Noel, A Aubin, B. Brogliato, and P. Drevet. Adaptive friction compensation in robot manipulators: low-velocities. In *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on*, pages 1352–1357 vol.3, May 1989.

[11] Arjan Gijsberts and Giorgio Metta. Real-time model learning using incremental sparse spectrum gaussian process regression. *Neural Networks*, 41(0):59 – 69, 2013. Special Issue on Autonomous Learning.

[12] Gabriel M. Hoffman, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Precision flight control for a multi-vehicle quadrotor helicopter testbed. *Control Engineering Practice*, 19(9):1023–1036, September 2011.

[13] Gabriel M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, 2007.

[14] Haomiao Huang, Gabriel M. Hoffmann, Steven L. Waslander, and Claire J. Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009.

[15] S. Shankar Sastry John J. Craig, Ping Hsu. Adaptive control of mechanical manipulators. *The International Journal of Robtics Research*, 1987.

[16] Tae-Yong Kuc, Kwanghee Nam, and J.S. Lee. An iterative learning control of robot manipulators. *Robotics and Automation, IEEE Transactions on*, 7(6):835–842, Dec 1991.

[17] Miguel Lázaro-Gredilla, Joaquin Quiñonero Candela, Carl Edward Rasmussen, and Aníbal R. Figueiras-Vidal. Sparse spectrum gaussian process regression. *J. Mach. Learn. Res.*, 99:1865–1881, August 2010.

[18] Daewon Lee, H. Jin Kim, and Shankar Sastry. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of Control, Automation, and Systems*, 7(3):419–428, 2009.

[19] J. Gordon Leishman. *Principles of Helicopter Aerodynamics.* Cambridge University Press, 2nd edition, March 2006.

[20] K. Ling, D. Chow, A Das, and S.L. Waslander. Autonomous maritime landings for low-cost vtol aerial vehicles. In *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, pages 32–39, May 2014.

[21] Guangjun Liu, K. Iagnemma, S. Dubowsky, and G. Morel. A base force/torque sensor approach to robot manipulator inertial parameter estimation. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 3316–3321 vol.4, May 1998.

[22] MATLAB. *Version 8.0.0.783 (R2012b)*. The MathWorks Inc., Natick, Massachusetts, 2012.

[23] Abdellah Mokhtari and A. Benallegue. Dynamic feedback controller of euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle. In *Proceedings of the 2004 IEEE International Converence on Robotics & Automation*, New Orleans, LA, April 2004. IEEE.

[24] F.L. Mueller, A.P. Schoellig, and R. D'Andrea. Iterative learning of feed-forward corrections for high-performance tracking. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3276–3281, 2012.

[25] Prasenjit Mukherjee and Steven Waslander. Direct adaptive feedback linearization for quadrotor control. In *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, 2012.

[26] D. Nguyen-Tong and J. Peters. Using model knowledge for learning inverse dynamics. In *IEEE International Conference on Robotics and Automation*, 2010.

[27] D. Nguyen-Tong, M. Seeger, and J. Peters. Real-time local gp model learning. In J. Peters O. Sigaud, editor, *From Motor Learning to Interation Learning in Robots*. Springer Verlag, 2010.

[28] D. Nguyen-Tuong and J. Peters. Incremental online sparsification for model learning in real-time robot control. *Neurocomputing*, 74(11):1859–1867, 2011.

[29] D. Nguyen-Tuong, J. Peters, M. Seeger, and B. Schoelkopf. Learning inverse dynamics: a comparison. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2008)*, 2008.

[30] D. Nguyen-Tuong, M. Seeger, and J. Peters. Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034, 2009.

[31] H. Olsson, K.J. Åström, C. Canudas de Wit, M. Gäfvert, and P. Lischinsky. Friction models and friction compensation. *European Journal of Control*, 4(3):176 – 195, 1998.

[32] Jan Peters and Stefan Schaal. Learning to control in operational space. *The International Journal of Robotics Research*, 27(2):197–212, 2008.

[33] K. Radkhah, D. Kulic, and E. Croft. Dynamic parameter identification for the crs a460 robot. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3842–3847, Oct 2007.

[34] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[35] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, December 2010.

[36] Roland Siegwart Samir Bouabdallah, Pierpaolo Murrieri. Design and control of an indoor micro quadrotor. In *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, New Orleans, LA, April 2004.

[37] Stefan Schaal Sethu Vijayakumar, Aaron D'Souza. Incremental online learning in high dimensions. *Neural Computation*, 2005.

[38] Jean-Jacques E. Slotine and Weiping Li. On the adaptive control of robot manipulators. *The International Journal of Robotics Research*, 6(3):49–59, 1987.

[39] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT press, 2006.

[40] Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot Modeling and Control.* John Wiley & Sons, Inc., 2006.

[41] J. Sun de la Cruz, D. Kulić, and W. Owen. Learning inverse dynamics for redundant manipulator control. In *Autonomous and Intelligent Systems (AIS), 2010 International Conference on*, pages 1–6, 2010.

[42] Jan Swevers, Chris Ganseman, D Bilgin Tukel, Joris De Schutter, and Hendrik Van Brussel. Optimal robot excitation and identification. *Robotics and Automation, IEEE Transactions on*, 13(5):730–740, 1997.

[43] Abdelhamid Tayebi. Adaptive iterative learning control for robot manipulators. *Automatica*, 40(7):1195 – 1203, 2004.