Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Model-Based policy Search for Learning Multivariate PID Gain Scheduling Control

Maksym Lefarov

Course of Study:	Computer Science
Examiner:	Dr. Daniel Hennes
Supervisor:	Dipl. Ing. Andreas Doerr, Dr. Christian Daniel, Dr. Duy Nguyen-Tuong.
Commenced:	October 18, 2017
Completed:	April 18, 2018

Abstract

Due to its simplicity and demonstrated performance, Proportional Integral and Derivative (PID) controller remains one of the most widely-used closed-loop control mechanisms in industrial applications. For the unknown model of a system, however, a PID design can become a significantly complex task especially for a Multiple Input Multiple Output (MIMO) case.

For the efficient control of a nonlinear and non-stationary systems, a scheduled PID controller can be designed. The classical approach to gain scheduling is a system linearization and the design of controllers at different operating points with a subsequent application of interpolation.

This thesis continues on the recent advances in application of Reinforcement Learning (RL) to a multivariate PID tuning. In this work we extend the multivariate PID tuning framework based on the Probabilistic Inference for Learning Control (PILCO) algorithm to tune a scheduled PID controllers. The developed method does not require the linear model of a system dynamics and is not restricted to the low-order or Single Input Single Output (SISO) systems. The algorithm is evaluated using the Noisy Cart-Pole and Non-stationary Mass-Damper systems. Additionally, the proposed method is applied to the tuning of a scheduled PID controllers of autonomous Remote Control (RC) car.

Contents

List of Figures

List of Tables

List of Abbreviations

PID Proportional Integral and Derivative. 3

MIMO Multiple Input Multiple Output. 3

RL Reinforcement Learning. 3

PILCO Probabilistic Inference for Learning Control. 3

SISO Single Input Single Output. 3

RC Remote Control. 3

GP Gaussian Processes. 13

LTI Linear Time-Invariant. 15

IAE Intergated Absolute Error. 16

ISE Intergated square Error. 16

ZN Ziegler-Nichols method. 21

AMIGO Approximate M-constraint Integral Gain Optimization. 21

IMC Internal Model Control. 23

SOFC Static Output Feedback Controller. 23

LMI Linera Matrix Inequalities. 23

NARX Non-linear Auto-regressive Exogenous. 29

QLPV Quasi-Linear Parameter Varying. 30

GMV General Minimum Variance. 31

LQR Linear Quadratic Regulator. 43

BFGS Broyden — Fletcher — Goldfarb — Shanno. 44

UKF Unscented Kalman Filter. 52

1 Introduction

A simple structure and a demonstrated performance permitted PID to become one of the most popular closed-loop control mechanisms used among the industry [21]. MIMO systems can be controlled using the multivariate PID controllers [1].

Despite the clear structure of PID, it is usually designed via the application of a tedious heuristic tuning routines. The advanced PID design techniques such as Pole Placement [1], Loop shaping [4], D-Partitioning and Cancellation of Poles [11] require the linear model of a system. Moreover, some of the design methods, e.g. Tuning based on Gain and Phase margins [2] and λ-tuning [4], are applicable only to the low-order systems. For the multivariate PID tuning it is commonly assumed that a plant can be decoupled into SISO systems [39]. A design of controllers is then performed separately for every system via the standard techniques. The evolutionary based optimization methods for multivariate PID tuning do not require the decoupling and are not restricted to the low-order systems [20]. Nevertheless, the known or approximated model of a plant is still required for the evaluation of a fitness function.

For the unknown systems, auto-tuning methods such as Relay Auto-tuner [34] or Feature-Based tuning techniques [4] are used. Both perform the model estimation with subsequent application of a model-based tuning techniques.

Application of learning-based methods to the control tasks in general and PID tuning in particular is limited by their data inefficiency. A system rollouts of a real control scenarios can be rather costly. Addressing this limitation, Deisenroth et al. [14] developed a model-based policy search algorithm PILCO, which maintains the probabilistic model of a system dynamics learned from an agent-system interactions. Such setup permits efficient learning with relatively small amount of observed data. Doerr et al. [16] extended PILCO for multivariate PID tuning. The developed framework is applicable to a nonlinear MIMO systems and does not require prior knowledge about the system dynamics.

Gain scheduling is an efficient way to control a nonlinear and non-stationary systems, whose dynamics changes with different operating conditions [4]. The design of a scheduled controller, usually performed via the linearization of a plant and subsequent application of linear design techniques [29]. The scheduled controller is then obtained using the nearest-neighbor interpolation, e.g. controller switching, or linear interpolation, e.g. controller blending. Additionally, Fuzzy logic can be exploited to introduce the human expertise into the controllers interpolation [38]. In contrast, gain scheduling can be implemented in a form of a parametrized nonlinear function, which computes the gains directly [9]. The parameters of this function are obtained via the optimization procedures. For both approaches, a known model of system dynamics is required.

In this thesis we extend the multivariate PID tuning framework by Doerr et al. to learn the gain scheduling function for a nonlinear and non-stationary MIMO systems control. The proposed method does require neither prior knowledge of a dynamics model, nor its linearizion. The scheduling function is defined as a set of Gaussian Processes (GP) parameterized by a training targets with a uniformly distributed training inputs. All equations, necessary for the integration of this definition into PILCO, were derived and implemented.

Tests were conducted using the simulations of Noisy Cart-Pole and Non-stationary Mass-Damper systems, which were developed as a part of the thesis. In order to prove the applicability of the method to the real control task, it was exploited to tune the scheduled PID controllers of the autonomous RC car. Different approaches to the RC car dynamics modeling were proposed and evaluated. Additionally, modifications of the control mechanism, which can potentially improve the results of model learning were introduced and implemented.

The thesis continues with a presentation of a theoretical background (Chapter 2) required for the understanding of a further work. This background includes the basics of Control Design, PID controller, gain scheduling and GP. Next, Chapter 3 surveys the related research in PID tuning and gain scheduling. Additionally, it contains the detailed explanation of PILCO and its application to PID tuning. Following chapter (Chapter 4) introduces the PILCO modification for a scheduled PID learning. The chapter presents both, definition of gain scheduling function and all derivations required for its integration into PILCO. Next two chapters are devoted to the testing of the implemented PILCO modification. Test experiments with Noisy Cart-Pole and Non-stationary Mass-Damper are presented in Chapter 5. Chapter 6 summarizes the work, made in algorithm application to the autonomous RC car. It starts with the system description and control mechanism analysis. Then the approaches to the dynamics modeling are presented. Chapter 6 ends with a discussion of the proposed control modifications aimed to improve the model learning results. Finally, Chapter 7 concludes this work.

2 Background

In the following chapter the theoretical background, required for the further work is presented. This background includes basic information about Control Design and PID Controller as well as about Gaussian Processes.

2.1 Control Design

Åström and Murray [4] define control as exploitation of one dynamical system, called *controller* to influence the behavior of another one, called *plant*. Plant has a sate $\boldsymbol{x} \in \mathbb{R}^M$. $\boldsymbol{y} \in \mathbb{R}^M$ is a state observation (in some real scenarios, state $\boldsymbol{x}$ can be observed only partially). *Closed-loop control system* is a mechanism, where the output of a controller $\boldsymbol{u} \in \mathbb{R}^F$ depends on a *process variable* $\hat{\boldsymbol{y}} \in \mathbb{R}^{D \leq M}$, which is an observation of (the part of) a plant state $\boldsymbol{x}$ that have to be controlled. The output of a controller is calculated based on the *feedback error* or *control error*: $\boldsymbol{e} = \boldsymbol{r} - \hat{\boldsymbol{y}} \in \mathbb{R}^D$, where $\boldsymbol{r}$ is reference value or *set point*. From now onward for notation clarity we will assume that process variable is a complete observation $\hat{\boldsymbol{y}} = \boldsymbol{y}$.

In a control domain plants and controllers are commonly described in terms of a transfer functions, which define the relation between their inputs and outputs of a Linear Time-Invariant (LTI) system. A transfer function is defined as:

$$G_p(s) = \frac{Y(s)}{U(s)} = \frac{\mathcal{L}\{y(t)\}}{\mathcal{L}\{u(t)\}},$$

with $\mathcal{L}\{\cdot\}$ being a Laplace transformation. The transfer function can be rewritten in a pole-zero form:

$$G_p(s) = K\frac{(s - z_1)(s - z_2)...(s - z_m)}{(s - p_1)(s - p_2)...(s - p_n)},$$

where $z_1, ..., z_m$ are system zeros and $p_1, ..., p_m$ are system poles. Denominator of a pole-zero form of the transfer function is called *characteristic polynomial*. The number of roots of characteristic polynomial (system poles) defines the order of a system. Transfer function of MIMO plant has a form of a matrix:

$$G_p(s) = \begin{bmatrix} g_{11}(s) & \dots & g_{1F}(s) \\ \vdots & \ddots & \vdots \\ g_{D1}(s) & \dots & g_{DF}(s) \end{bmatrix} \in \mathbb{R}^{D \times F}. \tag{2.1}$$

One of the main advantage of a transfer function notation is that the complex system, which contains the number of different functional subsystems, can be represented as a product of their transfer functions. Thus, the transfer function of a closed-loop control system is given as:

$$G_l(s) = \frac{C(s)G_p(s)}{1 - C(s)G_p(s)},$$

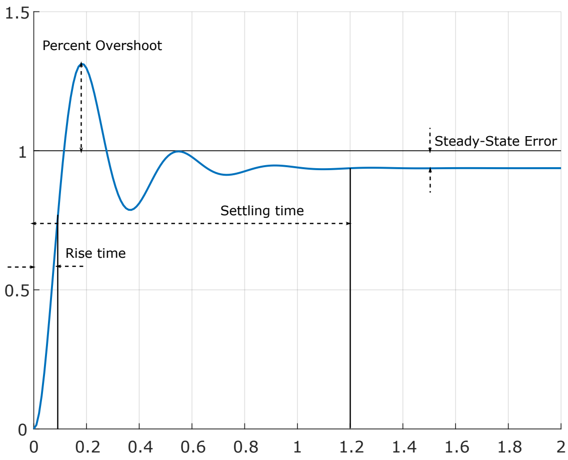


Figure 2.1: Quantities of a control system performance

where $C(s)$ and $G_p(s)$ are controller and plant transfer functions respectively.

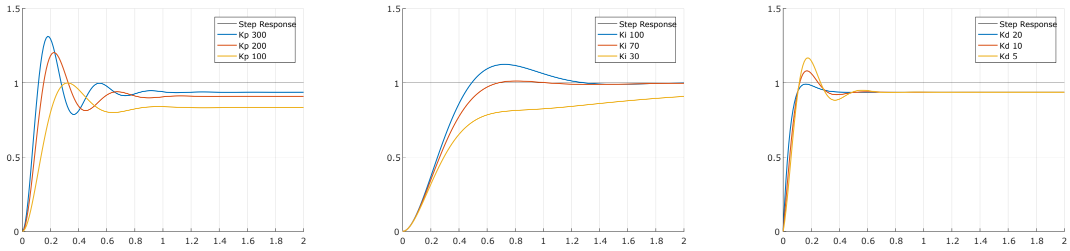
The control design has two common objectives, namely a set point following and a disturbance rejection [4]. The set point following objective is defined by the performance requirements. A performance of a SISO control system is often measured by applying a step function as r, and observing the response of y. Commonly, the response is quantified by waveform characteristics [23]. *Rise time* is the time, that system takes to reach the $80 - 90\%$ of final value after excitation with step r. *Percent overshoot* is the amount that y overshoots the r, expressed as a percentage of the final value of r. *Settling time* is the time required for the y to settle within a certain percentage (usually 5%) of the final value. *Steady-state error* is the final difference between the process variable y and set point r. Exact definition of these quantities can vary in industry and academia. For the MIMO systems these characteristics are measured for every output dimension. All described quantities are presented on Figure 2.1. The disturbance rejection objective is defined via the error, caused by the disturbance. Common quantities to characterize the disturbance attenuation are maximum error, time to maximum, decay ratio, Intergated Absolute Error (IAE) and Intergated square Error (ISE) [4].

2.2 PID controller

PID is a closed-loop feedback controller [2]. PID consists of three correcting terms, which are multiplications of gains with control error, its integral and derivative. Gains are the coefficients, which define the impact of a particular error type on PID output. Summation of terms gives the control output according to the equation:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{d(t)}. \tag{2.2}$$

In the above equation:



(a) Proportional gain **(b)** Integral gain **(c)** Derivative gain

Figure 2.2: The impact of PID gains on a waveform of a step response.

K_p is a *Proportional gain*
K_i is an *Integral gain*
K_d is a *Derivative gain*
$e(t)$ is a control error
t is a time
τ is an integration variable.

The proportional term $K_p e(t)$ accounts for the magnitude of current control error. The proportional gain determines the influence of the error on control output. As name suggests, high values of the gain result in the large change of control output for the given change in error. Excessively high values of the proportional gain could lead to the oscillation of the output and destabilization of the system, whereas low values could potentially make it insensitive to the error. As it demonstrated in Figure 2.2a, an increase in proportional gain reduced both the rise time and the steady-state error Additionally the overshoot is increased, and the settling time descries by a small amount.

The integral term $K_i \int_0^t e(\tau)d\tau$ accounts for both, magnitude of the error and its duration. It is a summation of all instantaneous errors over the time which corresponds to accumulated offset that should have been corrected previously. The integral gain defines the impact of this offset on the output of controller. An increase in integral gain tends to reduce the steady-state error since it is compensating for the collected errors from the past, however the overshooting (undershooting) is increased for the same reason (Figure 2.2b).

The derivative term $K_d \frac{de(t)}{d(t)}$ is set to compensate for the approximated future error behavior using the derivative of error with respect to time. Thus, tuning of derivative term improve the settling time and stability of the system (Figure 2.2c). Most implementations of PID include additional low-pass filter for the derivative to limit the influence of a noise on a system.

Transfer function of PID controller is defined as:

$$\begin{aligned} C(s) &= K_p \frac{s^2 T_i T_d + s T_i + 1}{s T_i}, \\ T_i &= \frac{K_p}{K_i}, \\ T_d &= \frac{K_d}{K_p}. \end{aligned} \tag{2.3}$$

2.2.1 Multivariate Case

In a general multidimensional case, equation (2.2) can be rewritten in a matrix notation form as following:

$$\boldsymbol{u}_t = \boldsymbol{K}_p \boldsymbol{e}_t + \boldsymbol{K}_i \int_0^t \boldsymbol{e}_t + \boldsymbol{K}_d \dot{\boldsymbol{e}}_t. \tag{2.4}$$

In the above equation $\boldsymbol{K}_p$, $\boldsymbol{K}_i$, and $\boldsymbol{K}_d$ are gain matrices of dimensionality $\mathbb{R}^{F \times D}$, where F is number of controller outputs and D is number of error states. Similarly, $\boldsymbol{e}_t$, $\int_0^t \boldsymbol{e}_t$ and $\dot{\boldsymbol{e}}_t$ are vectors of errors with dimensionality $\mathbb{R}^D$ at time step t and $\boldsymbol{u}_t \in \mathbb{R}^F$ is a control output at time t.

Two types of *Multivariate* PID controllers can be defined depending on the structure of gain matrices. The controller, which has the gain matrices of diagonal form is called *Decentralized* (*Decoupled*). Every output dimension of such controller depends only on one control error state. On the other hand, *Centralized* PIDs has gain matrices of arbitrary forms, meaning that every output of controller arbitrary depends on control errors. Transfer function of multivariate PID controller can be obtained similarly to (2.1). A transfer matrix is of diagonal form for a decoupled controller and arbitrary for a centralized. Centralized controllers have a better performance on systems with high interconnection between the states, however tuning of such controllers is significantly more difficult than decentralized ones, which is itself a tedious task. A detailed explanation on tuning of PID controllers is provided in Chapter 3.

2.2.2 Gain Scheduling

Gain scheduling is a changing of the controller's parameters (gains) depending on the measurements of operating conditions [4]. That measurements are called scheduling variables and could be systems states, process value, output of the controller or even external signals. Many design notions can fall under such broad definition - controller switching and controller blending fits the notion of gain scheduling as well as introduction of a nonlinear scheduling function [9, 25]. The gain scheduling is an efficient way to control nonlinear and non-stationary systems, whose dynamics changes with different operating conditions [4].

The most typical approach to design of gain scheduled controller for nonlinear or time varying system is a linearizion scheduling, which includes three steps [29]. The first step is computation of a linear parameter-varying model of the system. The most common method for such modeling is Jacobian Linearizion of a system around the equilibrium points ρ. This method results in a parametrized family of linearized systems, which form a basis for linearizion scheduling. The parametrization corresponds to the fixed values of the scheduling variables observed at corresponding equilibrium point.

Next step is to use the linear controller design methods for every linearized system in the family. This step results in a set of linear controllers at isolated values of scheduling variable. Final step is to obtain a linear parameter-varying controller. The simplest approach is controller switching, which is the nearest-neighbor interpolation of a controller parameters for the observed value of a scheduling variable. Even though such approach benefits form the simplicity, it produces discontinuous jumps in controller parameters which leads to performance decrease and probable destabilizations. Another way is to exploit linear interpolation, e.g. controller blending. Independent from linearizion and interpolation method, the more linear controllers are designed, the better performance is achieved [29]. More advanced method of gain scheduling will be reviewed in Chapter 3.

2.3 Gaussian Processes

A Gaussian process is a generalization of the Gaussian probability distribution, which describes the functions instead of scalars or vectors [26]. GP is a Bayesian nonparametric model which outputs the posterior distribution over the functions that describe the observed data $p(f|\mathcal{D})$. Non parametric stands for a model $M_{\boldsymbol{w}}$ parametrized by $\boldsymbol{w}$, which is selected from some family $\{M_{\boldsymbol{w}} : \boldsymbol{w} \in \mathcal{W}\}$, where $\mathcal{W}$ is infinite dimensional [31].

GP is completely specified by the prior distribution and observed data $\mathcal{D}$: n input vectors, $\boldsymbol{x} \in \mathbb{R}^D$ concatenated into matrix $\boldsymbol{X} \in \mathbb{R}^{D\times n}$, and targets $y \in \mathbb{R}$, concatenated into $\boldsymbol{y} \in \mathbb{R}^n$. Prior distribution represents the beliefs over the kind of functions, which one expects before the data is shown. The prior of GP is defined by mean function $m(\cdot)$ and kernel (positive semidefinite covariance function) $k(\cdot,\cdot)$. $m(\cdot)$ is commonly kept zero. Square Exponential is the usual choice for the kernel:

$$\kappa(\boldsymbol{x},\boldsymbol{x}') = \sigma_f^2 e^{\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}')^T\boldsymbol{\Lambda}^{-1}(\boldsymbol{x}-\boldsymbol{x}')\right)}. \tag{2.5}$$

In the above equation, $\boldsymbol{\Lambda}$ is $diag(\boldsymbol{l})^2$, where $\boldsymbol{l} \in \mathbb{R}^D$ is a vector of positive values called length-scales. The inverse of length-scale l_i determines the relevance of input dimension i. The hyper-parameters of GP, $\boldsymbol{\theta}$ are defined then as $(\boldsymbol{l},\sigma_f^2)^T$. For the realistic modeling scenario the noisy observations instead of a direct function values have to be assumed $y = f(x) + \epsilon$ [26]. For that purpose the additive independent identically distributed Gaussian noise ϵ with variance σ_w^2 is added to a kernel: $\kappa(\boldsymbol{x},\boldsymbol{x}') + \delta\sigma_w^2$, with δ being a Kronecker delta which returns one if $\boldsymbol{x} = \boldsymbol{x}'$ and zero otherwise. σ_w^2 is included into GP hyper-parameters $\boldsymbol{\theta} = (\boldsymbol{l},\sigma_f^2,\sigma_w^2)^T$. Covariance matrix for noisy observations $\mathbb{K}_y$ is defined as $(\mathbb{K} + \sigma_w^2\boldsymbol{I})$ with $\mathbb{K}_{ij} = \kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)$.

The hyper-parameters are obtained using the maximization of marginal likelihood:

$$p(\boldsymbol{y}|\boldsymbol{X};\boldsymbol{\theta}) = \int p(\boldsymbol{y}|f,\boldsymbol{X};\boldsymbol{\theta})p(f|\boldsymbol{X};\boldsymbol{\theta})df \tag{2.6}$$

The first two moments of a predictive posterior $p(y_*|\boldsymbol{x}_*,\boldsymbol{X},\boldsymbol{y};\boldsymbol{\theta})$ of the GP for the new input $\boldsymbol{x}_*$ are defined as following:

$$\mathrm{E}[y_*] = \kappa(\boldsymbol{X},\boldsymbol{x}_*)^T\mathbb{K}_y^{-1}\boldsymbol{y}, \tag{2.7}$$

$$\mathrm{var}[y_*] = \kappa(\boldsymbol{x}_*,\boldsymbol{x}_*) - \kappa(\boldsymbol{X},\boldsymbol{x}_*)^T\mathbb{K}_y^{-1}\kappa(\boldsymbol{X},\boldsymbol{x}_*), \tag{2.8}$$

The noticeable drawback of GP is its computational inefficiency - inversion of $\mathbb{K}_y$ is of $O(n^3)$ complexity. In order to reduce the complexity, sparse approximation of GP posterior can be used [12]. Snelson and Ghahramani [30] presents the Sparse Pseudo-Input Gaussian Processes. This model is parametrized by the pseudo (artificial) data set $\bar{\mathcal{D}}$, with the size $m \ll n$. Pseudo inputs $\bar{\boldsymbol{x}}$ are placed via the maximization of marginal likelihood (2.6). Pseudo targets $\bar{f}$ (without noise) are integrated out using the prior $\mathcal{N}(\bar{f}|0, \mathbb{K}_m)$ with $\mathbb{K}_{m_{ij}} = \kappa(\bar{\boldsymbol{x}}_i, \bar{\boldsymbol{x}}_j)$. The posterior of resulting sparse model is computed wilt complexity $O(m^2 n)$. After additional precomputations complexity can be improved even further to $O(m)$ for the mean of prediction and $O(m^2)$ for the prediction variance [30].

3 Related Work

This chapter gives the overview of the work related to the master thesis. First, the PID tuning research is surveyed. Then detailed explanation of the PILCO [14] and its application to optimization of PID gains [16] are presented. Chapter is concluded with the analysis of research in gain scheduling.

3.1 Research in PID Tuning

In the following section the review of relevant PID tuning techniques will be presented. For convenience all techniques are grouped into two artificial categories: classical techniques and techniques based on optimization.

3.1.1 Classical Techniques

Feature-based techniques are exploiting a features of system dynamics such as static gain K, velocity gain K_v, dominant time constant T and dominant dead time L [3], estimated via experiments, for the system approximation. Time-domain Ziegler-Nichols method (ZN) [4] is the most basic example of a feature-based tuning techniques, which is still used in industrial applications. This method approximates the system with a first-order plus dead-time model using the measurements obtained from the step response:

$$G_p(s) = \frac{K}{1 + sT} e^{-sL}. \tag{3.1}$$

The gains are given as functions of dominant dead time L and static gain K_p. Additionally, the Frequency-domain ZN [19] method defines gains as the functions of ultimate gain K_u (gain under which the system starts to oscillate) and ultimate period T_u (period of the oscillation).

The ZN methods are easy to apply, however they result in a stable but not optimal in term of set point following and disturbances attenuation controller (Section 2.1). Thus many modifications were proposed, which aimed to retain the simplicity of ZN and improve the performance in the same time. Approximate M-constraint Integral Gain Optimization (AMIGO) [4] uses the model of a system dynamics based on velocity gain K_v:

$$G_p(s) = \frac{K_v}{s} e^{-sL}. \tag{3.2}$$

Additionally AMIGO uses different tuning rules (gains functions). The design goal of these rules was to maximize the integral gain while fulfilling the specified disturbance attenuation constraints [4].

Tuning Based on Gain g_m and Phase ϕ_m margins [11] is aiming to find a PI controller:

$$C(s) = K_p \frac{sT_i + 1}{sT_i}, \tag{3.3}$$

for the given approximation of a system model (3.1) so that closed-loop system has defined gain and phase margins. K_p and T_i are obtained via the solution of g_m and ϕ_m equations [22].

Analytical methods use the system dynamics model (approximated) to directly calculate the gains of PID controller. A Pole Placement [1] is a subset of an analytical method applied to the systems with a low-order transfer function. A common approach is to specify the desired damping ratio ζ and the natural frequency ω_0 [22] for the system and to position the poles such that fulfill the specified constraints and give the required closed-loop performance. For instance, pole placement can be applied to a first-order plant:

$$G_p(s) = \frac{K}{1 + sT}, \tag{3.4}$$

controlled by PI controller (3.3). The transfer function of such closed-loop system will have the characteristic polynomial of a form:

$$s^2 + 2\zeta\omega_0 s + \omega_0^2,$$

then controller parameters are derived from actual transfer function as:

$$\begin{aligned} K_p &= \frac{2\zeta\omega_0 T - 1}{K}, \\ T_i &= \frac{2\zeta\omega_0 T - 1}{\omega_0^2 T}. \end{aligned} \tag{3.5}$$

The pole placement can also be exploited to tune the PID (2.3) applied to the second-order system:

$$G_p(s) = \frac{K}{(1 + sT_1)(1 + sT_2)}. \tag{3.6}$$

The characteristic polynomial of a closed-loop system will have a form:

$$(s + \alpha\omega_0)(s^2 + 2\zeta\omega_0 s + \omega_0^2),$$

and PID gains are obtained similarly to (3.5). The Dominant Pole [1] design is a simplified pole placement technique used for high-order systems. Commonly, dominant dynamics of a high-order system can be approximated by two dominant poles p_1, p_2 that are poles with real part closest to zero [11, 23]. Therefore dominant poles can be placed in a complex plane such that result in a system with desired ζ and ω_0:

$$\begin{aligned} p_1 &= -\zeta\omega_0 + i\omega_0\sqrt{1 - \zeta^2}, \\ p_2 &= -\zeta\omega_0 - i\omega_0\sqrt{1 - \zeta^2}. \end{aligned}$$

The PID gains are determined such that p_1 and p_2 are the roots of characteristic polynomial of closed-loop system. Dominant pole placement can be applied to Multivariate PID tuning [21].

The λ-tuning method assumes first-order plus dead-time model (3.1) with a long dominant dead time L [4]. This method specified the desired closed-loop transfer function as:

$$G_l(s) = \frac{e^{-sL}}{1 + s\lambda T},$$

where λ is an additional tuning parameter which influences the response time of the resulting closed-loop system. $\lambda < 1$ leads to a faster response and decrease of the integrated absolute error, but also increases sensitivity of a controller to variations in a system dynamics. The controller transfer function is then obtained from a transfer function of closed-loop system with feedback error:

$$C(s) = \frac{1 + sT}{K(1 + \lambda sT - e^{-sL})},$$

which, for $L = 0$, becomes a PI controller with:

$$K_p = \frac{1}{\lambda K},$$
$$T_i = T.$$

Internal Model Control (IMC) is a general control system design method, when the controller contains the model of a system [4] . The corresponding controller transfer function looks as:

$$C(s) = \frac{G_f G_m^\dagger}{1 - G_f G_m^\dagger G_m},$$

where G_m is a model transfer function, $G_m^\dagger$ its inverse and G_f is a low-pass filter. The low-pass filter accounts for modeling errors. This method results in high-order controllers. However, for simple models such as (3.4) and (3.1), it is possible to obtain PI or PID controller. PID controller for (3.4) and a filter of form:

$$G_f(s) = \frac{1}{1 + sT_f},$$

obtained via IMC is defined as:

$$C(s) = \frac{(1 + {}^{sL}\!/\!{}_2)(1 + sT)}{Ks(L + T_f)},$$

with T_f being design parameter. The main advantage of IMC method is that the robustness is considered explicitly during the controller design. Skogestad [27] proposes the internal model control method to tune the PI controller for the first-order plus time-delay systems (3.1) using Taylor-series approximation of exponential term. Dong and Brosilow [17] propose the method to derive the Multivariate PID from the Multivariate IMC using the MacLaurin series expansion.

Zheng et al. [39] propose another approach, similar to the IMC in a way that PID is obtained via the transformation to another type of controller. In presented work PID is transformed to a Static Output Feedback Controller (SOFC). The stabilization problem of static feedback controller is solved using the Linera Matrix Inequalities (LMI) method. PID gains are then recovered from the obtained controller.

Loop-shaping techniques can also be applied to PID tuning problem [4]. Loop-shaping aims to obtain the desired transfer function of a closed-loop system choosing the right controller. First, the desired gain crossover frequency ω_{gc} is chosen (for instance based on requirement of attenuation of load disturbance). Additionally, target phase margin ϕ_m is specified. Then gains of PID are calculated such that closed-loop system meets given specifications [4].

Most of the reviewed design methods require the known linear model of a system dynamics, e.g. Analytical techniques [1] and Loop-shaping [4]. ZN and AMIGO [4] approximates the unknown system with a low-order models using the measurements from a relay or step response experiments. Some, techniques are applicable only to a low-order plants, e.g. Gain and Phase margin based design [11], λ-tuning [4]. For the high-order systems, either the low-order approximations have to be computed, e.g. Dominant Pole design [21], or transformation to a different type of controller has to be performed [17, 39]. Among the surveyed methods, only a Dominant Pole [21] design can be relatively easy extended to a MIMO settings.

3.1.2 Optimization Based Techniques

The optimization methods can also be exploited for PID tuning. Given a controller parametrized by its gains, one is optimizing the closed-loop performance defined as functions of controller parameters. Among the popular optimization criteria are IAE, ISE etc. The optimization methods are rather sensitive to the formulation of criteria, meaning that they can result in optimal, with respect to ISE, but unstable controller because of neglected constraints [4]. Local minimas of the optimization criteria can also deteriorate the performance of method [4]. Cancellation of Poles and D-partitioning [11] are optimization methods for PID tuning which require the linear model of a plant. Modulus Optimum and Symmetrical Optimum are limited to low-order systems [1].

Genetics algorithms and evolutionary based techniques are, similarly to optimization methods seeking for the controller parameters, which optimize some criteria. However these methods are not restricted to the low-order and SISO systems and demonstrated their ability to overcome local minima problems [20].

Chang [8] proposes to use a multi-crossover genetic algorithm to optimize the IAE of multivariate PID controller. All elements of PID gain matrices (2.4) are concatenated into one vector, used as a chromosome. Algorithm was tested on classical binary distillation column plant - first order MIMO system with strong interaction between input and output pair, and significant time delays. According to the results, genetic algorithm converged to the stable controller after 300 of generations.

Gaing [18] applies modified Particle Swarm Optimization algorithm for tuning of multivariate PID controller. Each PID gain was presented by a particle. Tests were conducted on two automatic voltage regulator systems. After 10 seconds of optimization in average algorithm was able to output stable controller gains.

Even though the above algorithms managed to demonstrate satisfactory results for a MIMO testing systems, evolutionary based approaches cannot guarantee convergence [28]. Moreover, both require the model of a plant for the evaluation of a fitness function.

3.2 PILCO

The data inefficiency of RL limits its application to control and robotics tasks, where the agent-environment interaction could become rather costly.

Deisenroth et al. [14] address this problem via modeling the observed dynamics of the environment with a flexible nonparametric approach. The model-based RL methods are more efficient in extracting valuable information from the data [14]. However, the performance of such methods degrades significantly if learned model is biased. That is why, in order to compensate for the bias in model, Disenroth et al. are exploit probabilistic model with the explicit notion of uncertainty, which is also incorporated into planning and policy evaluation.

Authors are presenting PILCO, a model based-policy search framework with nonparametric GP (Section 2.3) used as a model. Long term predictions and policy evaluation in PILCO based on deterministic approximation of GP inference. Policy updates use the analytic solution of policy gradients.

Here onwards the detailed description of PILCO is presented with all equations, since they will be used in further chapters during the derivation of the gain scheduling. Nonetheless, policy gradients are omitted, because of auto-derivation software exploitation.

PILCO considers the dynamic systems of form:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t) + \omega, \quad \boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_{\omega}), \tag{3.7}$$

where $\boldsymbol{x}$ and $\boldsymbol{u}$ is a continuous-valued system state $\mathbb{R}^D$ and control $\mathbb{R}^F$ respectively. f is unknown transition dynamics and $\boldsymbol{\omega}$ is a Gaussian system noise. Policy search is optimizing $\pi : \boldsymbol{x} \mapsto \pi(\boldsymbol{x}, \boldsymbol{\theta})$ with respect to expected long term cost:

$$J^{\pi}(\boldsymbol{\theta}) = \sum_{t=0}^{T} \mathrm{E}[c(\boldsymbol{x}_t)], \quad \boldsymbol{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0). \tag{3.8}$$

In the above equation $c(\boldsymbol{x}_t)$ is a cost of being in state $\boldsymbol{x}$ at time t. A policy π is a function parametrized by $\boldsymbol{\theta}$.

In order to find π^*, which minimizes (3.8) PILCO builds a probabilistic GP model and uses the deterministic approximation of inference to compute the long term predictions and evaluate policy, then optimizes it exploiting analytical gradients (here however omitted, due to the usage of auto-derivation software). PILCO steps are summarized in Algorithm 3.1.

As a probabilistic dynamics model, PILCO uses GP, with tuples $(\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathbb{R}^{D+F}$ as training inputs and differences $\boldsymbol{\Delta}_t = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t \in \mathbb{R}^D$ as training targets. As it was stated in Section 2.3, GP outputs the posterior distribution over the functions, thus it cannot have multidimensional output. That is why PILCO actually uses D conditionally independent GP, trained simultaneously for every target dimension.

Algorithm 3.1 PILCO

```
 1: procedure POLICY SEARCH
 2:     Initialize:
        Sample policy parameters $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
        Apply $\pi(\boldsymbol{\theta})$ and record data $\{\boldsymbol{x}_t, \boldsymbol{u}_t\}_{t=1...T}$.
 3:     repeat
 4:         Learn probabilistic GP dynamics model, using all data.
 5:         repeat
 6:             Approximate inference for policy evaluation $J^\pi(\boldsymbol{\theta})$.
 7:             Obtain gradients $\frac{J^\pi(\boldsymbol{\theta}}{d\boldsymbol{\theta}}$ (auto-derivation software).
 8:             Update parameters $\boldsymbol{\theta}$ (e.g., CG or L-BFGS).
 9:         until convergence; return $\boldsymbol{\theta}^* = argminJ(\boldsymbol{\theta})$
10:         Set $\pi^* \leftarrow \pi(\boldsymbol{\theta}^*)$.
11:         Apply $\pi^*$ to system and record new data $\{\boldsymbol{x}_t, \boldsymbol{u}_t\}_{t=1...T}$.
12:     until task learned; return $\pi^*$
13: end procedure
```

According to Section 2.3, GP is completely specified by prior mean function $m(\cdot)$ (which is usually taken as zero) and kernel $k(\cdot, \cdot)$. In a PILCO paper Disenroth et al. use square exponential kernel (2.5). Training points of GP are defined as $\tilde{\boldsymbol{X}} = [\tilde{\boldsymbol{x}}_1, ..., \tilde{\boldsymbol{x}}_n]$ and $\boldsymbol{y} = [\Delta_1, ..., \Delta_n]^T$. Hyper-parameters of GP posterior are obtained via marginal likelihood maximization (2.6).

GP posterior gives one-step prediction $\boldsymbol{x}_{t+1}$, which is Gaussian distributed:

$$\begin{aligned} p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) &= \mathcal{N}(\boldsymbol{x}_{t+1}|\boldsymbol{\mu}_{t+1}, \boldsymbol{\Sigma}_{t+1}), \\ \boldsymbol{\mu}_{t+1} &= \boldsymbol{x}_t + \mathrm{E}[\boldsymbol{\Delta}_t], \\ \boldsymbol{\Sigma}_{t+1} &= \mathrm{var}[\boldsymbol{\Delta}_t], \end{aligned} \tag{3.9}$$

In the above equation $\mathrm{E}[\boldsymbol{\Delta}_t]$ and $\mathrm{var}[\boldsymbol{\Delta}_t]$ are calculated according to (2.7) and (2.8) respectively.

In order to evaluate J^π and optimize π (steps 6, 7, and 8 at Algorithm 3.1) PILCO constructs the t-step-ahead marginal distribution $p(\boldsymbol{x}_1|\pi), ..., p(\boldsymbol{x}_T|\pi)$ (further on conditioning on π is omitted for simplicity) with the initial state distribution $p(\boldsymbol{x}_0) = \mathcal{N}(\boldsymbol{x}_0|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$, which is done by the recursive application of (3.9). For that, one needs to propagate the uncertain inputs through the GP.

For one propagation step, i.e. to obtain $\boldsymbol{x}_{t+1}$ given $p(\boldsymbol{x}_t)$, joint distribution $p(\tilde{\boldsymbol{x}}) = p(\boldsymbol{x}_t, \boldsymbol{u}_t)$ is required (3.7). This distribution is approximated by Gaussian $p(\tilde{\boldsymbol{x}}) = \mathcal{N}(\tilde{\boldsymbol{x}}_t|\tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$. Using the approximated joint input distribution, posterior for uncertain input is given by:

$$p(\boldsymbol{\Delta}_t) = \int\int p(f(\tilde{\boldsymbol{x}}_t)|\tilde{\boldsymbol{x}}_t)p(\tilde{\boldsymbol{x}}_t)df d\tilde{\boldsymbol{x}}, \tag{3.10}$$

with f being the posterior of GP. Exact computation of (3.10) is analytically convoluted, that is why it is also approximated by Gaussian. Next state prediction $p(\tilde{\boldsymbol{x}}_{t+1})$ is obtained then as:

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \boldsymbol{\mu}_{\boldsymbol{\Delta}}, \\ \boldsymbol{\Sigma}_{t+1} &= \boldsymbol{\Sigma}_t + \boldsymbol{\Sigma}_{\boldsymbol{\Delta}} + \mathrm{cov}[\boldsymbol{x}_t, \boldsymbol{\Delta}_t] + \mathrm{cov}[\boldsymbol{\Delta}_t, \boldsymbol{x}_t]. \end{aligned} \tag{3.11}$$

Above approximation can be obtained using either Moment Matching or Linearizion of GP posterior mean function [14]. Details of latter method is omitted here since the Moment Matching is used in a further work.

To construct t-step-ahead marginal distribution using Moment Matching, one needs to compute predictive mean $\boldsymbol{\mu}_{\Delta}$, predictive covariance matrix $\boldsymbol{\Sigma}_{\Delta}$, covariances $\text{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t]$, $\text{cov}[\boldsymbol{\Delta}_t, \tilde{\boldsymbol{x}}_t]$ (3.11) and cross-covariance $\text{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\mu}_{\Delta}]$. First two terms are needed to obtain the distribution $p(\boldsymbol{\Delta}_t) = \mathcal{N}(\boldsymbol{\mu}_{\Delta}, \boldsymbol{\Sigma}_{\Delta})$ (3.10), covariances are used to get next step state distribution and cross-covariance is used to get next step joint state-action distribution. Exact derivations of mean and variance [14] are not presented here for thesis length reasons. Covariance and cross-covariance terms, in contrast, listed with detailed derivation, since they will be referenced during the formulation of gain scheduling in the following chapters.

The cross-covariance is computed between the joint state-action distribution at time step t : $\tilde{\boldsymbol{x}}_t \in \mathcal{N}(\tilde{\boldsymbol{x}}_t | \tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)$ and corresponding predicted (approximated) state difference $\boldsymbol{x}_{t+1} - \boldsymbol{x}_t = \boldsymbol{\Delta}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\Delta}, \boldsymbol{\Sigma}_{\Delta})$ as following:

$$\text{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t] = \text{E}[\tilde{\boldsymbol{x}}_t \boldsymbol{\Delta}_t^T] - \tilde{\boldsymbol{\mu}}_t \boldsymbol{\mu}_{\Delta}^T, \tag{3.12}$$

where the right term is defined, $\boldsymbol{\mu}_{\Delta}$ is given [14] and $\text{E}[\tilde{\boldsymbol{x}}_t]$ defined previously. In order to compute the left term, for each state dimension $a = 1, ..., D$, low of iterated expectations is applied:

$$\begin{aligned} \text{E}[\tilde{\boldsymbol{x}}_t \Delta_t^a] &= \text{E}[\tilde{\boldsymbol{x}}_t \, \text{E}[\Delta_t^a | \tilde{\boldsymbol{x}}_t]] \\ &= \int \tilde{\boldsymbol{x}}_t m_f^a(\tilde{\boldsymbol{x}}_t) p(\tilde{\boldsymbol{x}}_t) d\tilde{\boldsymbol{x}}_t \\ &= \int \tilde{\boldsymbol{x}}_t \left(\sum_{i=1}^{n} \beta_{ai} k_f^a(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_i) \right) p(\tilde{\boldsymbol{x}}_t) d\tilde{\boldsymbol{x}}_t, \end{aligned} \tag{3.13}$$

with posterior mean function $m_f(\tilde{\boldsymbol{x}}_t)$ represented as a finite kernel expansion. Then summation order is changed and β_{ai} is pulled out of integral:

$$\text{E}[\tilde{\boldsymbol{x}}_t \boldsymbol{\Delta}_t^a] = \sum_{i=1}^{n} \beta_{ai} \int \tilde{\boldsymbol{x}}_t \, \underbrace{c_1 \mathcal{N}(\tilde{\boldsymbol{x}}_t | \tilde{\boldsymbol{x}}_i, \boldsymbol{\Lambda}_a)}_{\kappa_f^a(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_i)} \underbrace{\mathcal{N}(\tilde{\boldsymbol{x}}_t | \tilde{\boldsymbol{\mu}}_t, \tilde{\boldsymbol{\Sigma}}_t)}_{p(\tilde{\boldsymbol{x}}_t)} \, d\tilde{\boldsymbol{x}}_t, \tag{3.14}$$

where $c_1 := \sigma_{fa}^2 (2\pi)^{\frac{D+F}{2}} |\boldsymbol{\Lambda}_a|^{\frac{1}{2}}$ is introduced to represent the GP kernel as an unnormalized Gaussian for training inputs $\tilde{\boldsymbol{x}}_i, i = 1, ..., n$. The product of two Gaussian (3.14) also results in unnormalized Gaussian $c_2^{-1} \mathcal{N}(\tilde{\boldsymbol{x}}_t | \boldsymbol{\varphi}_i, \boldsymbol{\Psi})$ with moments defined as following (A.2):

$$\begin{aligned} c_2^{-1} &= \sigma_{fa}^2 (2\pi)^{-\frac{D+F}{2}} |\boldsymbol{\Lambda}_a + \tilde{\boldsymbol{\Sigma}}_t|^{-\frac{1}{2}} e^{\left(-\frac{1}{2} (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t)^T (\boldsymbol{\Lambda}_a + \tilde{\boldsymbol{\Sigma}}_t)^{-1} (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t) \right)}, \\ \boldsymbol{\varphi}_i &= (\boldsymbol{\Lambda}_a^{-1} \tilde{\boldsymbol{x}}_i + \tilde{\boldsymbol{\Sigma}}_t^{-1} \tilde{\boldsymbol{\mu}}_t), \\ \boldsymbol{\Psi} &= (\boldsymbol{\Lambda}_a^{-1} + \tilde{\boldsymbol{\Sigma}}_t^{-1})^{-1}. \end{aligned} \tag{3.15}$$

After pooling all variables, independent of $\tilde{\boldsymbol{x}}_t$, out of integral in (3.14), it results in expected value of product of two Gaussians $\boldsymbol{\varphi}_i$:

$$\text{E}[\tilde{\boldsymbol{x}}_t \Delta_t^a] = \sum_{i=1}^{n} c_1 c_2^{-1} \beta_{ai} \boldsymbol{\varphi}_i, \quad a = 1, ..., D. \tag{3.16}$$

After inserting it back into (3.12):

$$\mathrm{cov}[\tilde{\boldsymbol{x}}_t, \Delta_t^a] = \sum_{i=1}^{n} c_1 c_2^{-1} \beta_{ai} \boldsymbol{\varphi}_i - \tilde{\boldsymbol{\mu}}_t \mu_{\boldsymbol{\Delta}}^a. \tag{3.17}$$

Final simplifications leads to:

$$\mathrm{cov}[\tilde{\boldsymbol{x}}_t, \Delta_t^a] = \sum_{i=1}^{n} c_1 c_2^{-1} \beta_{ai} \tilde{\boldsymbol{\Sigma}}_t (\boldsymbol{\Lambda}_a + \tilde{\boldsymbol{\Sigma}}_t)^{-1} (\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{\mu}}_t), \quad a = 1, ..., D. \tag{3.18}$$

Desired covariance terms $\mathrm{cov}[\tilde{\boldsymbol{x}}_t, \boldsymbol{\Delta}_t]$ from (3.11) is a sub-matrix of size $\mathbb{R}^{D \times D}$ of matrix $\mathbb{R}^{(D+F) \times D}$ computed at above equation.

Now all terms required for the uncertainty propagation are defined, thus t-step-ahead marginal distribution can be obtained. Last step to evaluation of J^π (3.8) is computation of expected values:

$$\mathrm{E}[c(\boldsymbol{x}_t)] = \int c(\boldsymbol{x}_t) \mathcal{N}(\boldsymbol{x}_t | \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) d\boldsymbol{x}_t,$$

with c being a cost function. PILCO uses binary saturation cost function:

$$c(\boldsymbol{x}) = 1 - e^{\left(-\frac{1}{2\sigma_c^2} d(\boldsymbol{x}, \boldsymbol{x}_{\mathrm{tar}})^2\right)} \in [0, 1],$$

where d is euclidean distance and σ_c^2 is parameter, which controls the width of the cost function. For the further work this function will be kept, since it allows for a natural exploration [14].

PILCO was evaluated using Double-Pendulum swing-up and Cart-Pole problems. Main focus was set on learning speed and quality of approximated inference. PILCO was trained for 15 iterations (37.5 seconds of interaction) for Cart-Pole system and 30 iterations (75 seconds) for the Double-Pendulum swing-up. As a results, algorithm in 95% was able to learn the policy, which solves the Cart-Pole problem, after 15 - 20 seconds of experience. The same success rate for the Double-Pendulum system was achieved after 40 - 50 seconds of experience. Additionally, resulting t-step-ahead marginal distribution has a relatively small variance for the successfully learned policy. That indicates the ability of GPs to model the dynamics of both systems. Taking into account the difficulty of tasks and obtained results, one can conclude that PILCO fulfilled the data efficiency constraint.

3.3 PID Tuning using PILCO

Considering the demonstrated performance of PILCO and its applicability to the real control scenarios, i.e. data efficiency, Doerr et al. [16] extend the described framework for multivariate PID tuning.

To represent the arbitrary multivariate PID controller as a parametrized static policy, a state information has to be extended. In order to compute the controller output $\boldsymbol{u}_t$, all necessary error information from (2.4) has to be available at time step t. Additionally, a joint distribution for the extended state has to be calculated in order to propagate the uncertainty, compute the t-step-ahead marginal distribution and evaluate J^{PID}. New system state z_t was introduced. It contains the error at previous time step $\boldsymbol{e}_{t-1}$ and errors, accumulated until the previous time step $\sum_{\tau=0}^{t-1} \boldsymbol{e}_\tau$:

$$z_t := \left[\boldsymbol{e}_{t-1}, \Delta T \sum_{\tau=0}^{t-1} \boldsymbol{e}_\tau, \tilde{\boldsymbol{x}}_{t-1}\right],$$

with $[\cdot,\cdot]$ stating for concatenation of vectors and ΔT being a system sampling time. Then the GP posterior $p(\boldsymbol{x}_t)$, approximated via Moment Matching, is added to the state, resulting in $\tilde{z}_t = [z_t, \boldsymbol{x}_t]$.

After calculation of posterior expectation, variance and cross-covariance terms, described in Section 3.2, the joint distribution for the new state extension can be obtained. Assuming that the $\boldsymbol{x}_t$ is independent from the error part of z_t one gets:

$$\begin{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{e}}_{t-1} \\ \tilde{\boldsymbol{x}}_{t-1} \end{bmatrix} \\ \boldsymbol{x}_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_{z_t} \\ \boldsymbol{\mu}_{\boldsymbol{x}_t} \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{z_t} & \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{\Sigma}_{\boldsymbol{x}_t, \tilde{\boldsymbol{x}}_{t-1}} \end{bmatrix} \\ \begin{bmatrix} \boldsymbol{0} & \boldsymbol{\Sigma}_{\tilde{\boldsymbol{x}}_{t-1}, \boldsymbol{x}_t} \end{bmatrix} & \boldsymbol{\Sigma}_{\boldsymbol{x}_t} \end{bmatrix} \right), \tag{3.19}$$

where $\tilde{\boldsymbol{e}}_{t-1}$ is the errors part of extended state. $\boldsymbol{\Sigma}_{\boldsymbol{x}_t, \tilde{\boldsymbol{x}}_{t-1}}$ stands for the cross-covariance term $\text{cov}[\boldsymbol{x}_t, \tilde{\boldsymbol{x}}_{t-1}]$ and $\boldsymbol{\Sigma}_{\tilde{\boldsymbol{x}}_{t-1}, \boldsymbol{x}_t}$ for $\text{cov}[\tilde{\boldsymbol{x}}_{t-1}, \boldsymbol{x}_t]$.

Next, a desired trajectory information has to be added. The target trajectory $\boldsymbol{x}_{\text{tr}}$ is given as Gaussian Distribution $\mathcal{N}(\boldsymbol{x}_{\text{tr}} | \boldsymbol{\mu}_{\text{tr}}, \boldsymbol{\Sigma}_{\text{tr}})$ [15]. A new state extension is $\tilde{z}_t = [z_t, \boldsymbol{x}_t, \boldsymbol{x}_{\text{tr},t}]$. Since the target trajectory is independent from $\tilde{z}_t$ the joint distribution of the new state extension can be obtained similarly to (3.19), where the state prediction part $\boldsymbol{x}_t$ is independent from $\tilde{\boldsymbol{e}}_{t-1}$. A new mean of $\tilde{z}_t$ is just a concatenation of a previous mean and a target mean $\boldsymbol{\mu}_{\text{tr},t}$. A covariance matrix is a concatenation of $\boldsymbol{\Sigma}_{\tilde{z}}$ and $\boldsymbol{\Sigma}_{\text{tr},t}$ along the diagonal.

The current error is a difference between the predicted state $\boldsymbol{x}_t$ and desired $\boldsymbol{x}_{\text{tr}}$. For the current error a derivative and integrated error approximations are calculated as following:

$$\dot{\boldsymbol{e}}_t \approx \frac{\boldsymbol{e}_t - \boldsymbol{e}_{t-1}}{\Delta T},$$

$$\int_0^{t \cdot \Delta T} \boldsymbol{e}(\tau) d\tau \approx \Delta T \sum_{\tau=0}^{t-1} \boldsymbol{e}_\tau + \Delta T \boldsymbol{e}_t.$$

Defined above errors are linear transformations of $\tilde{z}$, thus the joint distribution of the new state extension $\tilde{z}_t = [z_t, \boldsymbol{x}_t, \boldsymbol{x}_{\text{tar},t}, \boldsymbol{e}_t, \Delta T \sum_{\tau=0}^{t} \boldsymbol{e}_\tau, \frac{\boldsymbol{e}_t - \boldsymbol{e}_{t-1}}{\Delta T}]$ remains Gaussian and can be obtained via the known formulas of linear Gaussian transformation (A.1).

At this point, $\tilde{z}_t$ contains the information required for the calculation of PID output $\boldsymbol{u}_t$ as a linear transformation of the augmented state. The final joint distribution $p(\tilde{z}_t, \boldsymbol{u}_t)$ is calculated according to (A.1). All values required for one PILCO iteration are defined.

Evaluation experiments were conducted using the inverted pendulum problem on the humanoid upper-body robot Apollo. State of the inverted pendulum system consists of four states: effector position and velocity, a pendulum angle and an angular velocity. In order to omit the modeling of the entire system state and its dynamics during the experiments, the Non-linear Auto-regressive Exogenous (NARX) model [7] for the effector position and pendulum angle was used. For the dynamics modeling Doerr et al. [16], use a sparse GP with the hyper-parameters learned by a marginal likelihood maximization.

According to the demonstrated results, the presented algorithm was able to learn accurate dynamics model and optimize the PID policy with 106 seconds of interaction with physical system in total. Hence, the algorithm preserves the data-efficiency property of PILCO and can be applied to real control scenarios. Therefore, this work was considered as a suitable starting point for the further research in data-driven controller tuning. For the scope of this thesis it was decided to extend the presented algorithm for learning of optimal gain scheduled PID, with the aim on improvement of performance for nonlinear systems and on application to non-stationary systems [4].

3.4 Gain Scheduling

Apart from Jacobian linearizion mentioned in Section 2.2.2, Quasi-Linear Parameter Varying (QLPV) [29] approach can be used to obtain a family of linear systems. The nonlinear system is rewritten in a way to replace the nonlinear terms with new time-varying parameters and use them later one as a scheduling variables. This method does not require the linearizion and equilibrium point determination. However, new definition of a system can introduce "the additional behavior beyond the original plant description" [29], which is complicating the design of a suitable controller. Not all systems can be redefined as QLPV which constraints the possible applications of this method. Moreover, the QLPV representation of a system is not unique and have to be additionally determined for a particular implementation of gain scheduling [29]. The extension of linearizion based scheduling (Jacobian and QLPV) to MIMO systems depends on the controller design method used for a linearized parameter varying system [6].

3.4.1 Fuzzy Logic Based Gain Scheduling

The Fuzzy Inference [37] is a common mechanism to incorporate the system knowledge into a scheduling step (interpolation of controllers) via the specification of a Fuzzy rules (membership functions) in order to increase the performance and robustness [38].

Visioli [33] presents the work where the gain scheduling was applied to build the two degrees of freedom controller with one part handling the attenuation of load disturbances and the second part being responsible for set point following. Two controllers were tuned separately using ZN. The resulting controller then was scheduled based on the current control error and its derivative using the Fuzzy Inference for interpolation.

Woo et al. [36] applies the Fuzzy inference to schedule a parallel PI and PD controllers. They also introduced the Parameter Regulator which adaptively change the membership functions of the Fuzzy module, based on the step response.

The requirement of a human expertise and absence of general tuning method makes the Fuzzy scheduling strongly application dependent [38]. Since the Fuzzy logic based scheduling modifies the interpolation step only, it does not influence the applicability to MIMO systems.

3.4.2 Artificial Neurons Based Gain Scheduling

Due to the capability of neural networks to accurately approximate the highly nonlinear function, they appear in gain scheduling research [24].

Chen and Huang [10] design the gain scheduled PID controller for a highly nonlinear systems. PID gains are scheduled using the separate outer loop, composed of a General Minimum Variance (GMV) controller [5] and a neural network. The neural network is trained off-line to model the dynamics of a system. The linear dynamics model is obtained at every control iteration via the linearizion of the neural net. The GMV computes then the optimal gains of PID for the given linear model of a system. The proposed method, however, accounts only for a SISO nonlinear systems.

Chang et al. [9] propose the method to schedule a multivariate PID using the Auto-tuning Neurons. Auto-tuning neuron is a sigmoid function with an adaptive shape applied to the thresholded input. Sigmoid is parametrized by the saturation level and its slope. Each PID gain of MIMO controller is defined as an output of such sigmoid. The gains are scheduled then based on the control error. In order to tune such scheduled controller a monotonically responding system is assumed, i.e. the output is monotonically increasing or decreasing with the increase in a control signal. Under such assumption the authors were able to derive the optimization algorithm, which outputs the optimal sigmoid parameters with respect to the IAE.

4 Gain Scheduling using PILCO

In this chapter the definition of gain scheduling function is presented as well as mathematical derivations, necessary for its implementation and integration into PILCO framework.

4.1 Formulation

As it was discussed in Section 2.2.2 and Section 3.4, a gain scheduling can vary from simple interpolation of linear controllers, built at system's equilibrium points, to the complex nonlinear function, which defines the gains given the scheduling variable. For the purpose of this master thesis it was decided to follow the latter approach and encapsulate the entire scheduling into one nonlinear parametrized function. Parameters $\boldsymbol{\theta}$ of this function then will be optimized as a policy parameters $\pi(\boldsymbol{\theta})$ during the PILCO iteration (Algorithm 3.1).

In order to stay in fully Bayesian setting and maintain the PILCO notion of uncertainty, the gain scheduling function was defined as a set of GPs parametrized by the training targets with uniformly distributed training inputs. Gains of the multivariate PID controller are given as:

$$\boldsymbol{K} \sim \mathcal{N}(\boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K) \in \mathbb{R}^{3FE}, \tag{4.1}$$

where F is the dimensionality of a control signal and E is the dimensionality of a control error. Each element $\mathrm{E}[K_a]$ of gains mean vector $\boldsymbol{\mu}_K$ and $\mathrm{var}[K_a]$ of covariance matrix $\boldsymbol{\Sigma}_K$ for $a = 1, \ldots, 3FE$ are defined according to (2.7) and (2.8) as:

$$\mathrm{E}[K_a] = \kappa(\boldsymbol{\Gamma}, \boldsymbol{x}_\mathrm{s})^T (\mathbb{K} + \sigma_w^2 \boldsymbol{I})^{-1} \boldsymbol{y}_a, \tag{4.2}$$

$$\mathrm{var}[K_a] = \kappa(\boldsymbol{x}_\mathrm{s}, \boldsymbol{x}_\mathrm{s}) - \kappa(\boldsymbol{\Gamma}, \boldsymbol{x}_\mathrm{s})^T (\mathbb{K} + \sigma_w^2 \boldsymbol{I})^{-1} \kappa(\boldsymbol{\Gamma}, \boldsymbol{x}_\mathrm{s}). \tag{4.3}$$

In the above equation, $\boldsymbol{x}_\mathrm{s} \in \mathbb{R}^S$ is a vector of scheduling variables and $\boldsymbol{\Gamma} \in \mathbb{R}^{N \times S}$ is a matrix of uniformly distributed training inputs with N being a number of inputs and S - dimensionality of scheduling vector. Elements of the matrix $\mathbb{K}$ are given by $\kappa(\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_j)$, where $\kappa(\cdot, \cdot)$ is a prior covariance and $\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_j \subset \boldsymbol{\Gamma}$. $\boldsymbol{y}_a \in \mathbb{R}^N$ is a training target vector for GP outputting K_a.

Vectors $\boldsymbol{y}_a$, concatenated into matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times 3FE}$, form the parameters $\boldsymbol{\theta}$ of a policy $\pi(\boldsymbol{\theta})$, which will be optimized by PILCO. Hyper-parameters such as $k(\cdot, \cdot)$ and $\boldsymbol{\Gamma}$ are shared among all GPs. They are not a part of $\boldsymbol{\theta}$, i.e. are not optimized by the algorithm. The gradient of GP with respect to its prior covariance is more unstable than with respect to targets, thus the convergence of a gradient based method can be compromised. That is why hyper-parameters are determined independently for every application using the experiments.

Joint distribution $p(\tilde{z}, \boldsymbol{K})$, where $\tilde{z}$ is the extended state from Section 3.3 is completely defined [14]. Nevertheless, the control signal $\boldsymbol{u}$ under such definition is a product of two correlated multidimensional random variables $p(\boldsymbol{K}|\boldsymbol{x}_{\mathrm{s}})$ and $p(\tilde{\boldsymbol{e}}|z_t, \boldsymbol{x}_t, \boldsymbol{x}_{\mathrm{tar},t})$, that is concatenation of all errors vectors. To integrate the defined above scheduling function into the PILCO framework, the control mean $\mathrm{E}[\boldsymbol{u}]$ and variance $\mathrm{var}[\boldsymbol{u}]$ have to be computed (approximated). Additionally, cross-covariance term $\mathrm{cov}[\boldsymbol{u}, \tilde{z}]$ have to be derived for the approximation of joint distribution $p(\tilde{z}, \boldsymbol{u})$, where $\tilde{z}$ is already concatenated with gains.

During the work on this thesis two cases were considered: $\boldsymbol{K}$ being dependent on and independent of $\tilde{\boldsymbol{e}}$. It is obvious that for independent case neglects the cross-covariance information between $\boldsymbol{K}$ and $\tilde{\boldsymbol{e}}$, which can lead to deterioration of optimization results. However, for the fully dependent case rough approximations for $\mathrm{var}[\boldsymbol{u}]$ are used, which can also turn into a problem. Rest of the chapter contains all necessary derivations for both cases.

4.2 Dependent Case

The multivariate PID controller's equation (2.4) from the Background section looks as following:

$$\boldsymbol{u}_t = \boldsymbol{K_p}\boldsymbol{e}_t + \boldsymbol{K_i}\int_0^t \boldsymbol{e}_t + \boldsymbol{K_d}\dot{\boldsymbol{e}}_t. \tag{4.4}$$

Since the above equation is a linear combination of three terms, in order to avoid redundancy, derivations only for the proportional $\boldsymbol{u}_p$ part are provided. Two others can be derived via the same computations. Furthermore, for the clarity of derivations, an input is assumed to be one dimensional. Extension to multidimensional case is trivial (separate application to every dimension).

4.2.1 Expected Value

Expected value of proportional term can be decomposed as:

$$\mathrm{E}[u_p] = \mathrm{E}[\boldsymbol{K}_p^T\boldsymbol{e}] = \mathrm{E}[\sum_i \boldsymbol{K}_{pi}\boldsymbol{e}_i] = \sum_i \mathrm{E}[\boldsymbol{K}_{pi}\boldsymbol{e}_i].$$

To the above equation total covariance low is applied:

$$\mathrm{cov}[\boldsymbol{X}, \boldsymbol{Y}] = \mathrm{E}[\boldsymbol{XY}] - \mathrm{E}[\boldsymbol{X}]\,\mathrm{E}[\boldsymbol{Y}], \tag{4.5}$$

which results in:

$$\begin{aligned}\mathrm{E}[u_p] &= \sum_i (\mathrm{cov}[\boldsymbol{K}_{pi}, \boldsymbol{e}_i] + \boldsymbol{\mu}_{ki}\boldsymbol{\mu}_{ei}) \\ &= \boldsymbol{\mu}_k^T\boldsymbol{\mu}_e + \sum_i \mathrm{cov}[\boldsymbol{K}_{pi}, \boldsymbol{e}_i].\end{aligned}$$

Covariances and mean vectors, required for the computation of above equation, can be obtained from the known joint distribution of the extended state $p(\tilde{z})$.

4.2.2 Variance

In order to compute the variance, as previously, u_p is first rewritten as a sum of products:

$$\mathrm{var}[u_p] = \mathrm{var}[\sum_i \boldsymbol{K}_{pi}\boldsymbol{e}_i].$$

Consider system state first being only two dimensional ($\boldsymbol{K}, \boldsymbol{e} \in \mathbb{R}^2$). Here onwards $\boldsymbol{K}$ stands for $\boldsymbol{K}_p$ to avoid indices redundancy:

$$\mathrm{var}[K_0e_0 + K_1e_1] = \mathrm{var}[K_0e_0] + \mathrm{var}[K_1e_1] + 2\,\mathrm{cov}[K_0e_0, K_1e_1],$$

which can be generalized to:

$$\mathrm{var}[\sum_i K_ie_i] = \sum_i \mathrm{var}[K_ie_i] + \sum_i^{n-1}\sum_{j=i+1}^{n} 2\mathrm{cov}[K_ie_i, K_je_j]. \tag{4.6}$$

First, blue term of the (4.6) is derived. Application of variance definition results in:

$$\begin{aligned}\mathrm{var}[K_ie_i] &= \mathrm{E}[(K_ie_i)^2] - \mathrm{E}[K_ie_i]^2\\ &= \mathrm{E}[K_i^2e_i^2] - \mathrm{E}[K_ie_i]^2.\end{aligned} \tag{4.7}$$

The second term of right side of above equation can be computed via the application of total covariance low (4.5) as following:

$$\begin{aligned}\mathrm{E}[K_ie_i]^2 &= (\mathrm{cov}[K_i, e_i] + \mathrm{E}[K_i]\,\mathrm{E}[e_i])^2\\ &= (\mathrm{cov}[K_i, e_i] + \mu_{ki}\mu_{ei})^2,\end{aligned}$$

where all required information is contained in the joint distribution of the extended state $p(\tilde{z})$. However, the $\mathrm{E}[K_i^2e_i^2]$ term of (4.7) turned out to be more difficult to compute.

Application of the low of iterated expectations to the $\mathrm{E}[K_i^2e_i^2]$ results in:

$$\mathrm{E}[K_i^2e_i^2] = \mathrm{E}[\mathrm{E}[K_i^2e_i^2|K_i]] = \mathrm{E}[K_i^2\,\mathrm{E}[e_i^2|K_i]], \tag{4.8}$$

where K_i and e_i are jointly normally distributed random variables with correlation coefficient ρ. Then, the conditional distribution of e_i on K_i is also normal with mean and variance defined as following:

$$\begin{aligned}\mathrm{E}[e_i|K_i] &= \mathrm{E}[e_i] + \rho\sqrt{\frac{\mathrm{var}[e_i]}{\mathrm{var}[K_i]}}(K_i - \mathrm{E}[K_i]),\\ \mathrm{var}[e_i|K_i] &= \mathrm{var}[e_i](1-\rho^2).\end{aligned} \tag{4.9}$$

Now, the expected value of error squared conditioned on gain $\mathrm{E}[e_i^2|K_i]$ is obtained via definition of conditioned variance:

$$\mathrm{E}[e_i^2|K_i] = \mathrm{var}[e_i|K_i] - \mathrm{E}[e_i|K_i]^2.$$

Substitution of the inner term of the right most expectation in (4.8):

$$K_i^2 \operatorname{E}[e_i^2|K_i] = K_i^2 \left[\operatorname{var}[e_i|K_i] + \operatorname{E}[e_i|K_i]^2 \right].$$

Then the above equation is rewritten with the definitions from (4.9):

$$K_i^2 \operatorname{E}[e_i^2|K_i] = K_i^2 \left[\operatorname{var}[e_i](1-\rho^2) + \left(\operatorname{E}[e_i] + \rho\sqrt{\frac{\operatorname{var}[e_i]}{\operatorname{var}[K_i]}} \Big(K_i - \operatorname{E}[K_i] \Big) \right)^2 \right].$$

The above expectation is a quartic function of K_i, named $g(K_i)$. Insertion of $g(K_i)$ back into (4.8) results in:

$$\operatorname{E}[K_i^2 e_i^2] = \operatorname{E}[g(K_i)].$$

At this point a first approximation is introduced. A performance of the algorithm will probably depend on a quality of this approximation, so, as a future work, higher order approximations for this term can be derived. For now $\operatorname{E}[g(K_i)]$ can be approximated using the Taylor Series, first two derivatives of $g(K_i)$ and first two moments of K_i as following:

$$\operatorname{E}[g(K_i)] = \operatorname{E}[g(\mu_{ki}) + g(\dot{\mu}_{ki})(K_i - \mu_{ki}) + \frac{1}{2} g(\ddot{\mu}_{ki})(K_i - \mu_{ki})^2] = g(\mu_{ki}) + \frac{1}{2} g(\ddot{\mu}_{ki})\sigma_x^2.$$

Below first two derivatives of $g(K_i)$ are computed. The $\rho\sqrt{\frac{\operatorname{var}[e_i]}{\operatorname{var}[K_i]}}$ term is substituted with A for readability reasons:

$$\begin{aligned} g(\dot{K}_i) = {} & 2K_i \left(\operatorname{var}[e_i](1-\rho^2) + \Big(\operatorname{E}[e_i] + A\big(K_i - \mu_{ki}\big) \Big)^2 \right) \\ & + K_i^2 \left(2\Big(\operatorname{E}[e_i] + A\big(K_i - \mu_{ki}\big) \Big) A \right). \end{aligned}$$

$$\begin{aligned} g(\ddot{K}_i) = {} & 2 \left(\operatorname{var}[e_i](1-\rho^2) + \Big(\operatorname{E}[e_i] + A\big(K_i - \mu_{ki}\big) \Big)^2 \right) \\ & + 4K_i \left(2\Big(\operatorname{E}[e_i] + A\big(K_i - \mu_{ki}\big) \Big) A \right) \\ & + K_i^2 A^2. \end{aligned}$$

All information, reacquired for computation of above equations and $\operatorname{E}[K_i^2 e_i^2]$ can be obtained from joint distribution of extended state $p(\tilde{z})$. Now all components of $\operatorname{var}[K_i e_i]$ in (4.6) are defined.

In order to compute $\operatorname{cov}[K_i e_i, K_j e_j]$ from (4.6) we are applying the total covariance low (4.5):

$$\operatorname{cov}[K_i e_i, K_j e_j] = \operatorname{E}[K_i e_i K_j e_j] - \operatorname{E}[K_i e_i] \operatorname{E}[K_j e_j]. \tag{4.10}$$

Second part $\operatorname{E}[K_i e_i] \operatorname{E}[K_j e_j]$ of the right side of the above equation is completely defined. Consecutive application of (4.5) to it results in terms contained at $p(\tilde{z})$. However, $\operatorname{E}[K_i e_i K_j e_j]$ revealed to be more difficult. The above term was derived with loss of information, which is introducing second source of possible deterioration in results and is an aim for improvement in further work. For now, K_i is assumed to be independent of e_j and K_j of e_i, which permits to rearranging the inner multiplication terms of expectation and rewrite it as $\mathrm{E}[K_iK_j]\,\mathrm{E}[e_ie_j]$. Then substitute it back to (4.10) and apply (4.5) to the right part. Rearrangement of terms results in:

$$\begin{aligned}
\operatorname{cov}[K_ie_i, K_je_j] &= \mathrm{E}[K_iK_j]\,\mathrm{E}[e_ie_j] \\
&- (\operatorname{cov}[K_i, e_i] + \mu_{ki}\mu_{ei})(\operatorname{cov}[K_j, e_j] + \mu_{kj}\mu_{ej}) \\
&= (\operatorname{cov}[K_i, K_j] + \mu_{ki}\mu_{kj})(\operatorname{cov}[e_i, e_j] + \mu_{ei}\mu_{ej}) \\
&- (\operatorname{cov}[K_i, e_i] + \mu_{ki}\mu_{ei})(\operatorname{cov}[K_j, e_j] + \mu_{kj}\mu_{ej}) \\
&= \operatorname{cov}[K_i, K_j]\operatorname{cov}[e_i, e_j] + \operatorname{cov}[K_i, K_j]\mu_{ei}\mu_{ej} + \operatorname{cov}[e_i, e_j]\mu_{ki}\mu_{kj} \\
&- \operatorname{cov}[K_i, e_i]\operatorname{cov}[K_j, e_j] - \operatorname{cov}[K_i, e_i]\mu_{kj}\mu_{ej} - \operatorname{cov}[K_j, e_j]\mu_{ki}\mu_{ei}.
\end{aligned}$$

The above equation concludes the derivation of all terms, required to compute the variance (4.6), even though currently we are neglecting $\operatorname{cov}[K_i, e_j]\operatorname{cov}[K_j, e_i]$ etc.

4.3 Independent Case

As it was mentioned above, assuming the independence of $\boldsymbol{K}$ and $\tilde{\boldsymbol{e}}$ results in loss of valuable information. Nevertheless, for the independent case computations are significantly simpler than for the dependent one, which allows to avoid approximations

4.3.1 Expected Value

Since $\operatorname{cov}[K_{pi}, e_i]$ is equal to 0, the expected value becomes:

$$\begin{aligned}
\mathrm{E}[u_p] &= \sum_i (\operatorname{cov}[K_{pi}, e_i] + \mu_{ki}\mu_{ei}) \\
&= \mu_k^T\mu_e.
\end{aligned}$$

4.3.2 Variance

Instead of approximating $\mathrm{E}[K_i^2e_i^2]$ in (4.7), the covariance low (4.5) is applied repeatedly:

$$\begin{aligned}
\mathrm{E}[K_i^2e_i^2] &= \operatorname{cov}[K_i^2, e_i^2] - \mathrm{E}[K_i^2]\,\mathrm{E}[e_i^2] \\
&= \operatorname{cov}[K_i^2, e_i^2] - (\operatorname{var}[K_i] + \mathrm{E}[K_i]^2)(\operatorname{var}[e_i] + \mathrm{E}[e_i]^2).
\end{aligned}$$

Substitution of the above equations to (4.7) results in:

$$\begin{aligned}
\operatorname{var}[K_ie_i] &= \operatorname{cov}[K_i^2, e_i^2] \\
&- (\operatorname{var}[K_i] + \mathrm{E}[K_i]^2)(\operatorname{var}[e_i] + \mathrm{E}[e_i]^2) \\
&- (\operatorname{cov}[K_ie_i] + \mu_{ki}\mu_{ei})^2.
\end{aligned}$$

Now $\mathrm{cov}[K_i^2, e_i^2]$ and $\mathrm{cov}[K_i, e_i]$ can be replaced with 0. Further simplification leads to:

$$\mathrm{var}[K_i e_i] = \mathrm{var}[K_i]\,\mathrm{var}[e_i] + \mathrm{var}[K_i]\mu_{ei}^2 + \mathrm{var}[e_i]\mu_{ki}^2.$$

And $\mathrm{cov}[K_i e_i, K_j e_j]$ from (4.6) simplifies to:

$$\mathrm{cov}[K_i e_i, K_j e_j] = \mathrm{cov}[K_i, K_j]\,\mathrm{cov}[e_i, e_j] + \mathrm{cov}[K_i, K_j]\mu_{ei}\mu_{ej} + \mathrm{cov}[e_i, e_j]\mu_{ki}\mu_{kj}$$

4.4 Cross Covariance

Cross covariance term resembles one given at [14]. However, the computation becomes significantly more complex for the covariance between the $\boldsymbol{u}$ and the entire extended state. $\tilde{z}$ stands for the augmented state, which includes all information:

$$\mathrm{cov}[\tilde{z}, \boldsymbol{u}] = \mathrm{E}[\tilde{z}\boldsymbol{u}] - \boldsymbol{\mu}_{\tilde{z}}\boldsymbol{\mu}_{\boldsymbol{u}}^T, \tag{4.11}$$

with $\mathrm{E}[\tilde{z}\boldsymbol{u}]$ being a vector of $[\mathrm{E}[\tilde{z}u^0], \mathrm{E}[\tilde{z}u^1], .., \mathrm{E}[\tilde{z}u^a], ..]$. Using the low of iterated expectations and given that all GPs of scheduling function share hyper-parameters similarly to (3.13):

$$\begin{aligned}
\mathrm{E}[\tilde{z}u^a] &= \mathrm{E}[\tilde{z}\,\mathrm{E}[u^a|\tilde{z}]] \\
&= \int \tilde{z}\big(\boldsymbol{\mu}_a^{kp}\boldsymbol{e} + \boldsymbol{\mu}_a^{ki}\int_0^t \boldsymbol{e} + \boldsymbol{\mu}_a^{kd}\dot{\boldsymbol{e}}\big)p(\tilde{z})d\tilde{z} \\
&= \int \tilde{z}\sum_i \kappa(\boldsymbol{x}_\mathrm{s}, \boldsymbol{\gamma}_i)\big(\boldsymbol{\beta}_{ai}^{kp}\boldsymbol{e} + \boldsymbol{\beta}_{ai}^{ki}\int_0^t \boldsymbol{e} + \boldsymbol{\beta}_{ai}^{kd}\dot{\boldsymbol{e}}\big)p(\tilde{z})d\tilde{z},
\end{aligned} \tag{4.12}$$

where $\boldsymbol{x}_\mathrm{s}$ is a vector of scheduling variables, $\boldsymbol{\gamma}_i$ is a training input and $\kappa(\cdot,\cdot)$ is a prior covariance of the scheduling function described at Section 4.1. $\boldsymbol{\beta}_a^{kp}$ is obtained from (4.2) as:

$$\boldsymbol{\beta}_a^{kp} = (\mathbb{K} + \sigma_w^2\boldsymbol{I})^{-1}\boldsymbol{y}_a,$$

for GP outputting the a dimension of proportional gain vector $\boldsymbol{K}_p$ etc. Now the linear transformation matrices are introduced:

$\boldsymbol{S}$ returns the scheduling variable $\boldsymbol{x}_\mathrm{s}$ from the extended state $\tilde{z}$
$\boldsymbol{P}$ returns current error
$\boldsymbol{I}$ returns sum of errors and
$\boldsymbol{D}$ returns error derivative.

The insertion of the above matrices into (4.12) results in:

$$
\begin{aligned}
\mathrm{E}[\tilde{z}u^a] &= \int \tilde{z} \sum_i k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)\big(\beta^{kp}_{ai}(\boldsymbol{P}_a\tilde{z}) + \beta^{ki}_{ai}(\boldsymbol{I}_a\tilde{z}) + \beta^{kd}_{ai}(\boldsymbol{D}_a\tilde{z})\big)p(\tilde{z})d\tilde{z} \\
&= \int \tilde{z} \sum_i k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)\beta^{kp}_{ai}(\boldsymbol{P}_a\tilde{z})p(\tilde{z})d\tilde{z} \\
&+ \int \tilde{z} \sum_i k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)\beta^{ki}_{ai}(\boldsymbol{I}_a\tilde{z})p(\tilde{z})d\tilde{z} \\
&+ \int \tilde{z} \sum_i k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)\beta^{kd}_{ai}(\boldsymbol{D}_a\tilde{z})p(\tilde{z})d\tilde{z}.
\end{aligned}
$$

Then only the first summation term is considered. After pulling out the independent variables and change the order of summation and integration:

$$
\mathrm{E}[\tilde{z}u^a] = \sum_i \beta^{kp}_{ai} \int \tilde{z}k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)(\boldsymbol{P}_a\tilde{z})p(\tilde{z})d\tilde{z}.
$$

Next, $k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)$, in a blue term above, is transformed to a Normal distribution, using the multiplication with constant c_1 as in (3.14). $(\boldsymbol{P}_a\tilde{z})$ is also normal according to (A.1):

$$
\int \tilde{z}k(\boldsymbol{S}\tilde{z}, \boldsymbol{\gamma}_i)(\boldsymbol{P}_a\tilde{z})p(\tilde{z})d\tilde{z} = \int \tilde{z}c_1\mathcal{N}(\boldsymbol{S}\tilde{z}|\boldsymbol{\gamma}_i, \boldsymbol{\Lambda})\mathcal{N}(\boldsymbol{P}_a\tilde{z}|\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)\mathcal{N}(\tilde{z}|\boldsymbol{\mu}_{\tilde{z}}, \boldsymbol{\Sigma}_{\tilde{z}})d\tilde{z}.
$$

Now, as in [14], Gaussians product rule (A.2) can be applied. However, for that $\mathcal{N}(\boldsymbol{S}\tilde{z}|\boldsymbol{\gamma}_i, \boldsymbol{\Lambda})$ has to be transformed to $\mathcal{N}(\tilde{z}|.,.)$ and similarly $\mathcal{N}(\boldsymbol{P}_a\tilde{z}|\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$. Using the (A.1):

$$
\begin{aligned}
\boldsymbol{S}\tilde{z} &\sim \mathcal{N}(\boldsymbol{\gamma}_i, \boldsymbol{\Lambda}), \\
\tilde{z} &\sim \mathcal{N}(\boldsymbol{S}^\dagger\boldsymbol{\gamma}_i, \boldsymbol{S}^\dagger\boldsymbol{\Lambda}\boldsymbol{S}^{\dagger T}),
\end{aligned}
$$

where $\boldsymbol{S}^\dagger$ is a pseudo inverse of $\boldsymbol{S}$. At this point it was decided to stop computation of cross-covariance between the action and entire extended state, since there was no confidence that it will lead to the right result and not just over-complication. Instead the cross-covariance term only for scheduling vector $\boldsymbol{x}_\mathrm{s}$ of extended state $\tilde{z}$ was calculated:

$$
\mathrm{cov}[\boldsymbol{x}_\mathrm{s}, \boldsymbol{u}] = \mathrm{E}[\boldsymbol{x}_\mathrm{s}\boldsymbol{u}] - \boldsymbol{\mu}_{\boldsymbol{x}_\mathrm{s}}\boldsymbol{\mu}_{\boldsymbol{u}}^T.
$$

All errors terms are assumed to be given as the constants and independent from $\boldsymbol{x}_\mathrm{s}$. As previously, the low of iterated expectations is applied to every element $\mathrm{E}[\boldsymbol{x}_\mathrm{s}u^a]$ of expectation of product in above equation:

$$
\mathrm{E}[\boldsymbol{x}_\mathrm{s}u^a] = \int \boldsymbol{x}_\mathrm{s} \sum_i k(\boldsymbol{x}_\mathrm{s}, \boldsymbol{\gamma}_i)\big(\boldsymbol{\beta}^{kp}_{ai}\boldsymbol{e} + \boldsymbol{\beta}^{ki}_{ai}\int_0^t \boldsymbol{e} + \boldsymbol{\beta}^{kd}_{ai}\dot{\boldsymbol{e}}\big)p(\boldsymbol{x}_\mathrm{s})d\boldsymbol{x}_\mathrm{s}.
$$

	Dependent Approximation		Independent Approximation	
Experiment	μ Error	σ Error	μ Error	σ Error
1	0.214	1.727	0.211	0.531
2	0.061	0.413	0.069	1.342
3	0.005	0.021	0.005	9.506

Table 4.1: The moments errors for the subset of numerical test experiments.

Next, by pulling out the independent terms and switching the order of summation and integration:

$$\begin{aligned}\mathrm{E}[\boldsymbol{x}_\mathrm{s} u^a] &= \sum_i \big(\boldsymbol{\beta}_{ai}^{kp}\boldsymbol{e} + \boldsymbol{\beta}_{ai}^{ki}\int_0^t \boldsymbol{e} + \boldsymbol{\beta}_{ai}^{kd}\dot{\boldsymbol{e}}\big)\Big(\int \boldsymbol{x}_\mathrm{s} k(\boldsymbol{x}_\mathrm{s}, \boldsymbol{\gamma}_i) p(\boldsymbol{x}_\mathrm{s}) d\boldsymbol{x}_\mathrm{s}\Big)\\ &= \sum_i \boldsymbol{\beta}_{ai}^{kp}\boldsymbol{e}\Big(\int \boldsymbol{x}_\mathrm{s} k(\boldsymbol{x}_\mathrm{s}, \boldsymbol{\gamma}_i) p(\boldsymbol{x}_\mathrm{s}) d\boldsymbol{x}_\mathrm{s}\Big) + \ldots + \\ &= \boldsymbol{e}{\color{red}\sum_i \boldsymbol{\beta}_{ai}^{kp}\Big(\int \boldsymbol{x}_\mathrm{s} k(\boldsymbol{x}_\mathrm{s}, \boldsymbol{\gamma}_i) p(\boldsymbol{x}_\mathrm{s}) d\boldsymbol{x}_\mathrm{s}\Big)} + \ldots + .\end{aligned}$$

In above equation the red term is $\mathrm{E}[\boldsymbol{x}_\mathrm{s}\boldsymbol{K}_p^a]$ defined in (3.16). Furthermore, for the implementation simplicity it can be obtained from the covariance $\mathrm{cov}[\boldsymbol{x}_\mathrm{s}, \boldsymbol{K}_p^a]$, which is contained at the joint distribution of extended state $p(\tilde{z})$ using (3.17). Now the final equation for cross-covariance computation look like:

$$\begin{aligned}\mathrm{cov}[\boldsymbol{x}_\mathrm{s}, u^a] = \mathrm{cov}&[\boldsymbol{x}_\mathrm{s}\boldsymbol{K}_p^a]\boldsymbol{e} + \boldsymbol{\mu}_{\boldsymbol{x}_\mathrm{s}}\boldsymbol{\mu}_{\boldsymbol{K}_p^a}^T\\ &+ \mathrm{cov}[\boldsymbol{x}_\mathrm{s}\boldsymbol{K}_i^a]\int_0^t \boldsymbol{e} + \boldsymbol{\mu}_{\boldsymbol{x}_\mathrm{s}}\boldsymbol{\mu}_{\boldsymbol{K}_i^a}^T\\ &+ \mathrm{cov}[\boldsymbol{x}_\mathrm{s}\boldsymbol{K}_d^a]\dot{\boldsymbol{e}} + \boldsymbol{\mu}_{\boldsymbol{x}_\mathrm{s}}\boldsymbol{\mu}_{\boldsymbol{K}_d^a}^T\\ &+ \boldsymbol{\mu}_{\boldsymbol{x}_\mathrm{s}}\mu_{u^a} .\end{aligned}$$

Thus, all equations, required for integration of our scheduling function, defined at Section 4.1, into the PILCO framework are derived.

4.5 Numerical Test

In order to validate the derivations provided in Section 4.2 and Section 4.4, several numerical test experiments were conducted. Subset of test experiments is presented at Figure 4.1. Left side of the figure demonstrates initial correlated Gaussian distributions (green and orange) as well as real distribution of a product (gray dashed) according to [13]. Plots on the right present the approximations of product with Normal distributions using the equation for dependent (blue) and independent (red) cases derived above.

Additionally, for every experiment, we sampled the exact product distribution [13] and calculated its mean and variance. Errors between the first two moments of approximated distributions and exact one for experiments at Figure 4.1 can be found in Table 4.1. As one can infer, mean error

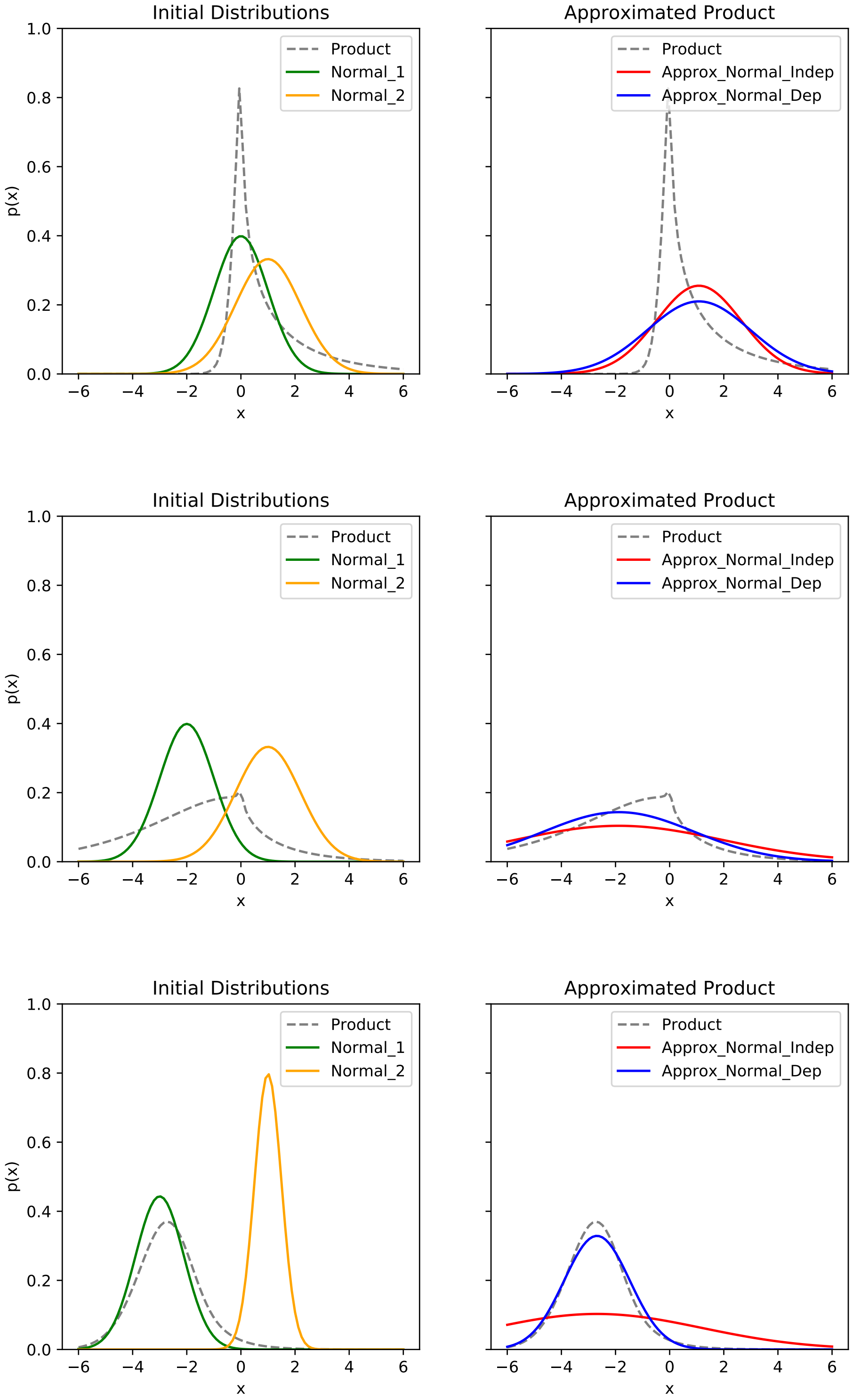


Figure 4.1: Subset of numerical test experiments

does not heavily depend on way of approximation, at least for the considered experiments. Whereas the variance changes a lot for dependent and independent cases. For variance there is no clear superiority of one approximation comparing to another - for some experiments dependent variance is outperforming independent, for others - vice versa. Decision on which approximation to use should be made separately for every application based on the experiments which permit to measure the quality of t-step-ahead marginal state distribution.

According to the demonstration, the provided derivations result in reasonable approximation. However, considering that during the PILCO iteration input distributions will be already approximated using the Moment Matching, it is rather difficult to estimate the quality until the tests of entire algorithm on real (testing) system.

5 Testing Systems

In this chapter the developed algorithm is applied to the testing systems. First, PILCO is used to tune the gains of a PID controller applied to the noisy Cart-Pole, which is the modification of OpenAI Gym Cart-Pole environment. Then, the modified PILCO will optimize the scheduled PID, which is controlling the non-stationary Mass-Damper system implemented for testing purposes.

5.1 Noisy Cart-Pole

In order to test the algorithm developed by Doerr et al. [16] and create a baseline for further work, PILCO was exploited to find the optimal gains of PID controlling the noisy Cart-Pole system. For that noisy modification of OpenAI Cart-Pole environment was developed. The standard OpenAI Gym Cart-Pole environment neglects a friction coefficient of a cart and a mass moment of inertia of the pole. The state-space representation of a system was recovered from the source code and the additive noise for the observations was included:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & mlg & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{g(m+M)}{-\frac{4}{3}l(m+M)+lm} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\frac{4}{3}l(m+M)}{(m+M)} \\ 0 \\ -\frac{1}{\frac{4}{3}l(m+M)+lm} \end{bmatrix} u,$$

$$\tilde{\boldsymbol{y}} = \begin{bmatrix} x \\ \phi \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u + \boldsymbol{\sigma}_y.$$

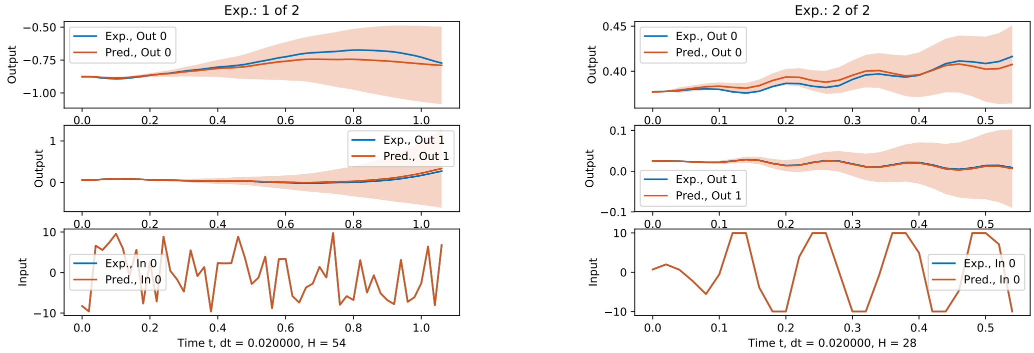
In the above equation:

x stands for a cart's position
θ is a pendulum angle from vertical
u is a force applied to the cart.

System parameters and their values used during the experiments are given in Table 5.1. Additionally, the PID controller was implemented to follow the desired trajectory of the cart position and the pole angle. The recovered state-space representation was discretized using Matlab 2016b. Then a Linear Quadratic Regulator (LQR) was computed as a benchmark for PILCO tuning. We put additional threshold to the pole angle in the modified environment. With the pole angle bounded at $+/-\frac{\pi}{6}$ the Cart-Pole dynamics is approximately linear. Target trajectory was set to zero for both, position and angle.

Parameter		Value	Unit
m	mass of pole	0.1	kg
M	mass of cart	1.0	kg
l	length to the pole center of mass	0.5	m
dt	system's sampling time	0.02	s
σ_y	spring characteristic	$\sim \mathcal{U}(1e^{-2}, 1e^{-3})$	m, rad

Table 5.1: Noisy Cart-Pole parameters.



(a) Test experiment 1

(b) Test experiment 2

Figure 5.1: Tests of Noisy Cart-Pole dynamics model

As a first step of PILCO, the initial dynamics model have to be learned (Algorithm 3.1). For that a sparse GP with 150 inducing points was trained on two experiments with the random policy (0.84 and 0.62 seconds) and one experiment with suboptimal, manually tuned, gains (2.0 seconds) in order to collect additional valuable information around equilibrium point. NARX state included 2 states of the history for position, angle and control in order to learn the derivatives.

Three experiments with the random policy were used for testing. The simulated feed-forward rollouts with initial dynamics model for two of them are presented in Figure 5.1. Both plots include the blue line which stands for the data collected from the real rollout with the given policy, the orange line, which presents the mean of t-step-ahead marginal distribution obtained from the model, and the shaded region which demonstrates the variance of distribution (model uncertainty). This description holds for all plots which demonstrate the comparison of simulated and real rollouts from now onwards. Output 0 and 1 are cart position and pole angle respectively, and Input is a control signal. Since the results were obtained from the feed-forward simulation, Input does not have a variance. As one can see, an initial GP rather accurately models the dynamics of a system but has a relatively big posterior variance.

With initial dynamics model, PILCO was applied to tune the gains of a PID controller. The concatenated optimization progress of four PILCO iterations is presented in Figure 5.2. The Broyden — Fletcher — Goldfarb — Shanno (BFGS) was used for optimization. BFGS iterations from 0 to 32 corresponds to the first PILCO iteration. Due to the good precision of the posterior mean of our initial model, the cost obtained from the simulated rollout (orange dashed line) reassembles the cost from the system rollout (blue dashed). The persistent margin between two lines corresponds

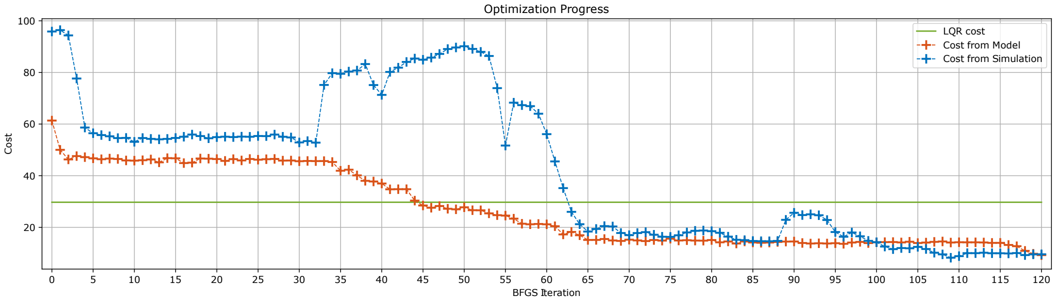
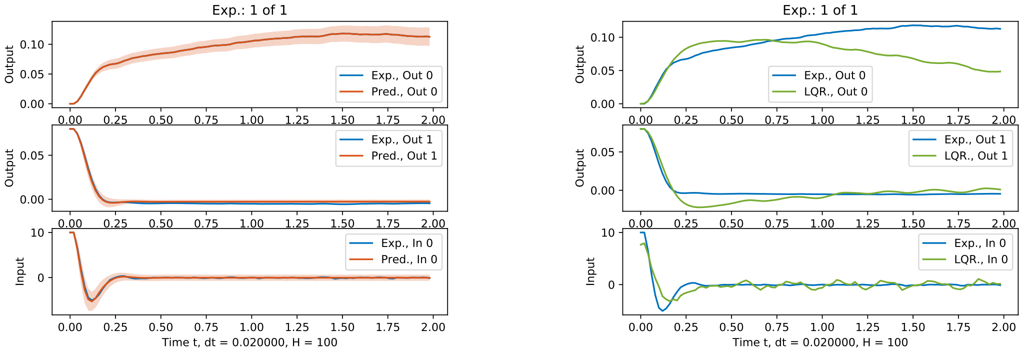


Figure 5.2: Concatenated optimization progress for the Noisy Cart-Pole PID.



(a) Simulated feed-back rollout for optimized PID

(b) The optimized PID versus LQR

Figure 5.3: The optimized PID for the Noisy Cart-Pole.

to the variance pf the initial model. According to Algorithm 3.1, at the end of the first iteration the model is retrained using a new data recorder with the optimized policy (one experiment, 1.42 seconds). However, since the number of inducing points of GP was not increased, retraining results in a degradation of the model precision. This effect can be observed during the second iteration (BFGS steps 33 – 63). Despite this fact, at the end of second iteration, the optimized policy was able to outperform the LQR (green line). Further concatenation of data and retraining of the model improved the precision and decreased the variance (BFGS steps 64 – 120).

The results of 4 iterations of PILCO are demonstrated in Figure 5.3. Figure 5.3a presents the simulated rollout for the optimized policy. As one can see, the resulting GP precisely models the dynamics and has relatively small variance. Figure 5.3b compares PID with the optimized gains (blue line) and LQR (green line). Since the cost function was set in a way to prioritize the pole angle error minimization, which leads to the faster system stabilization, LQR results in better tracking of the cart position (Output 0). PID optimized by PILCO, is clearly outperforming the LQR in second Output (pole angle), thus is more optimal with respect to the specified cost.

The conducted experiments prove that PILCO modification [16] is able to optimally tune the gains of PID with being data-efficient (9.46 seconds of system interaction was used in total to obtain the results from Figure 5.3). In the following section experimental setup used for testing of the scheduled PID tuning with PILCO (Chapter 4) is described and obtained results are analyzed.

Parameter		Value	Unit
m	mass	1.0	kg
D	damping coefficient	1.5	$^{Ns}/_{m}$
k	spring characteristic	~ Figure 5.4a	-
dt	system's sampling time	0.05	s

Table 5.2: Non-Stationary Mass-Damper parameters.

5.2 Non-stationary Mass-Damper

In order to test the derived integration of gain scheduling into PILCO the non-stationary Mass-Damper system was developed. The spring characteristic (k) is changing according to the sigmoid function with respect to the position of a center of mass (Figure 5.4a). Twenty continuous Mass-Damper systems with the following state-space representation were calculated:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{D}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u,$$

$$\tilde{\boldsymbol{y}} = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u,$$

where:

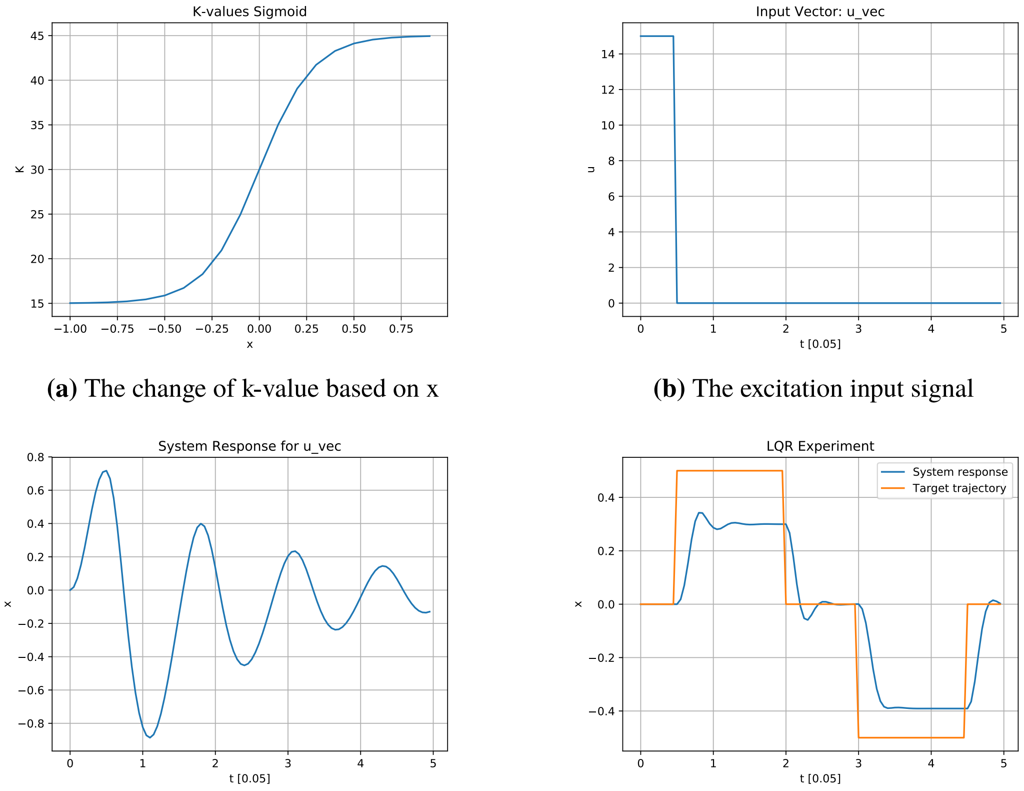
x stands for the position of a center of mass
u is a force applied to the a center of mass.

In the above differential equation, the parameters were set according to Table 5.2. Then, using Matlab 2016b these systems were converted to the discrete form with sampling time of 0.05 seconds. For the final non-stationary system $-\frac{k}{m}$ entry is linearly interpolated from discrete forms based on x.

Next, the PID controller for tracking the desired x was implemented. In order to define the controller output boundaries and feasible desired trajectory, the excitation signal (Figure 5.4b) was applied. The system's response (Figure 5.4c) demonstrates that with control bounded to $+/-15$, it is possible to follow the desired trajectory with the boundaries $+/-0.5m$.

Additionally LQR was computed in Matlab 2016b for the system with k equal to 15 (dynamics for negative x) for the benchmarking. Figure 5.4d presents the trajectory obtained with LQR (blue) versus the desired trajectory (orange). For the LQR trajectory one can observe persistent steady-state error in both negative and positive positions. Nonetheless, the error is lower for the negative x. The difference in steady-state errors and overshooting in positive and zero x shows that controller was tuned for dynamics in negative positions.

Then PILCO was applied to tune the PID gains. According to Algorithm 3.1, the initial probabilistic dynamics model have to be learned first. Sparse GPs with 100 inducing points and the NARX state with the history length of 4 for both, state and action were trained. The hyper-parameters were optimized using marginal likelihood. For the train experiments we use three rollouts with gains sampled uniformly from -10 to 10. The test experiments use similar rollouts.



(a) The change of k-value based on x

(b) The excitation input signal

(c) The excitation system's response

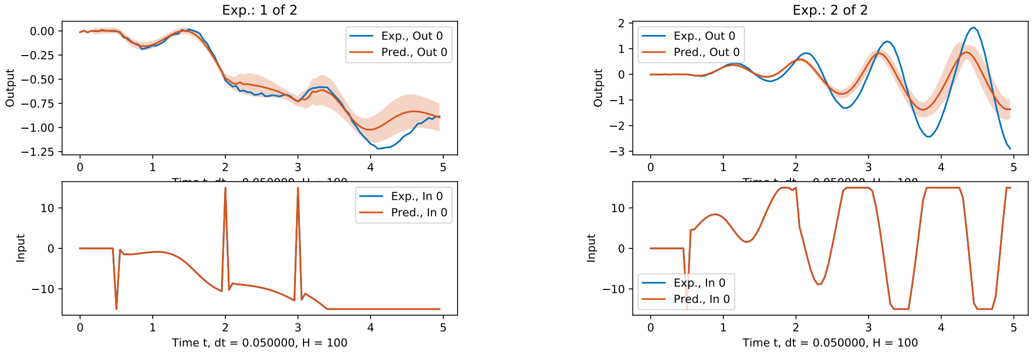
(d) LQR controller

Figure 5.4: Non-stationary Mass-Damper properties.

The results of test experiments are demonstrated in Figure 5.5. The Output stands for the x trajectory and the Input for a control command. Due to the simplicity of the system and enough NARX history GPs were able to learn the dynamics from three train experiments (5 seconds each).

The PID controller obtained after the first PILCO iteration is demonstrated in Figure 5.6. Figure 5.6a presents the simulated feedback rollout for the optimized controller. The predicted trajectory of x has a low variance, but deviates from the real one in the areas where the set point has a steep change. The reason is that the dynamics model predicts the non-truncated control commands and has no information about the boundaries of a controller output. This issue can be resolved via the concatenation of a new training data which contains a non-truncated control at the end of PILCO iteration. Despite this deviation, optimized controller outperforms LQR with respect to the specified cost function already after the first PILCO iteration (Figure 5.6d). Therefore, it was decided not to continue with a further iterations but proceed with a tuning of scheduled PID. Figure 5.6b demonstrates the comparison of the rollouts with optimized PID and LQR. Figure 5.6c plots the optimized PID rollout and the reference trajectory.

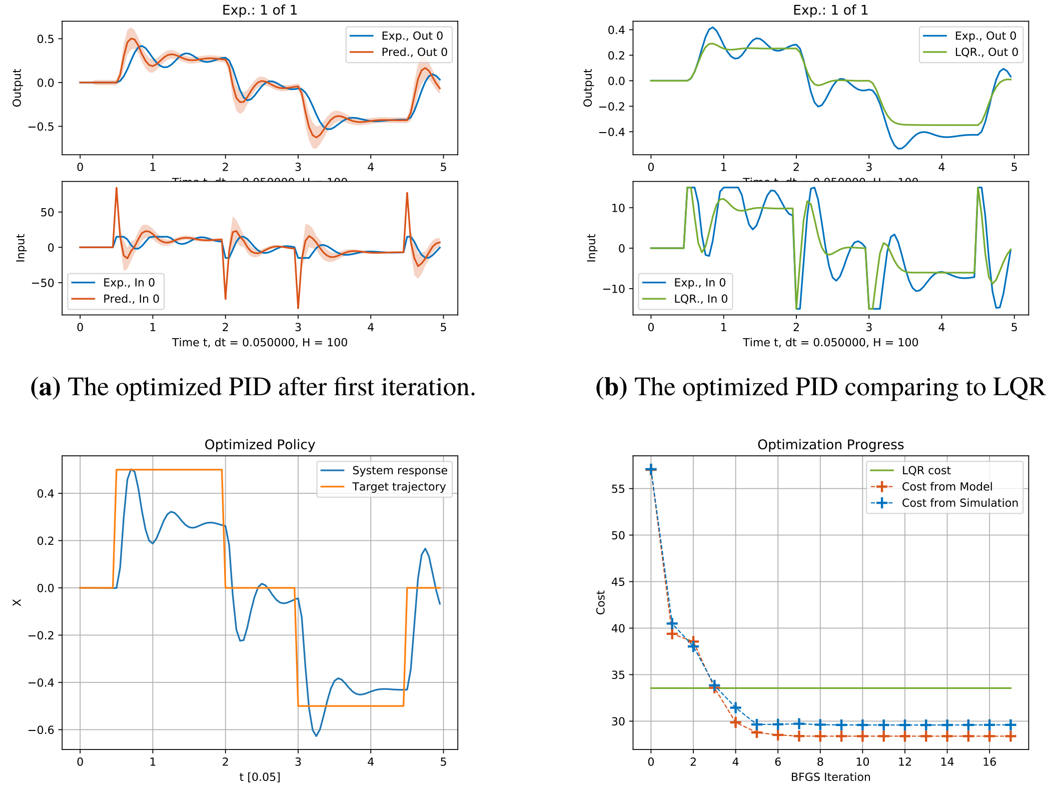
Next, the modified PILCO was applied to tuning of a gain scheduled PID which controls the Non-Stationary Mass-Damper system. As it was described in Section 4.1, the gain scheduling function is defined as a GPs with a uniformly distributed training inputs parametrized by the training



(a) Test experiment 1

(b) Test experiment 2

Figure 5.5: Tests of Non-stationary Mass-Damper dynamics model.



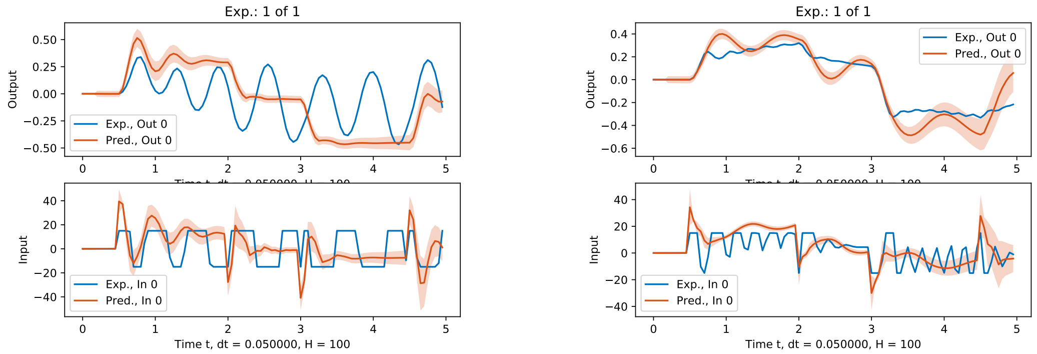
(a) The optimized PID after first iteration.

(b) The optimized PID comparing to LQR.

(c) The optimized PID comparing to target.

(d) The optimization progress.

Figure 5.6: The optimized nonscheduled PID for the Non-stationary Mass-Damper.



(a) Test experiment 1

(b) Test experiment 2

Figure 5.7: The optimized scheduled PID for the Non-Stationary Mass-Damper.

targets. Since the system dynamics depends on x, it was taken as a scheduling variable. The taring inputs were placed at $-0.5, 0, 0.5$. It was assumed that such inputs setup permits the optimized scheduling function to approximate the change of k and adjust the gains accordingly.

The scheduled controller optimized by 3 iterations of the modified PILCO is demonstrated in Figure 5.7a. The resulting controller has a reasonable performance with respect to the model prediction, however, the real rollout results in an oscillating trajectory. The observed behavior is caused by a steep scheduling function, which results in an extreme change of gains for the small variations in x. The order of magnitude is equal to 1 for the training targets and to -1 for x. Additionally the optimization procedure tends to place the targets in a relatively big range, e.g. from 20 to 80. The combination of this two points leads to the scheduling function with an extreme slope. Therefore during the feed-back simulation for the predicted x with a relatively small deviation from the real trajectory the gains are calculated with a strong bias.

In order to address this problem it was decided to put the additional cost on the training targets to prevent the learning of an extreme values. However, the adjustment of a cost coefficients of the binary saturation loss function (Section 3.2) revealed to be a rather complex problem. The small cost coefficients which punish the extreme values result in inability of auto-derivation software to compute the gradients. The coefficients were set to the smallest values that does not compromise a gradients calculation.

The result of a scheduled PID learning with the additional cost for the predicted gains are presented in Figure 5.7b. The difference between the predicted and simulated x trajectories is not as significant as in Figure 5.7a. However, the second plot of the figure shows that the predicted control signal still deviates from the real one. Nevertheless, the introduction of the additional cost permits to obtain the controller which outperform both, LQR and a nonscheduled PID. The concatenated optimization progress of 4 modified PILCO iterations is demonstrated in Figure 5.8. As one can see, due to the problem of the steep change in a gains the cost obtained from a model simulation and the cost of a real rollout does not converge to the perfect match. Nonetheless, after the 110 BFGS iterations (4 PILCO iterations) the optimized scheduled controller outperforms the LQR and a plain PID. The costs of LQR and simple PID rollouts are different from Figure 5.6d due to the additional cost terms.

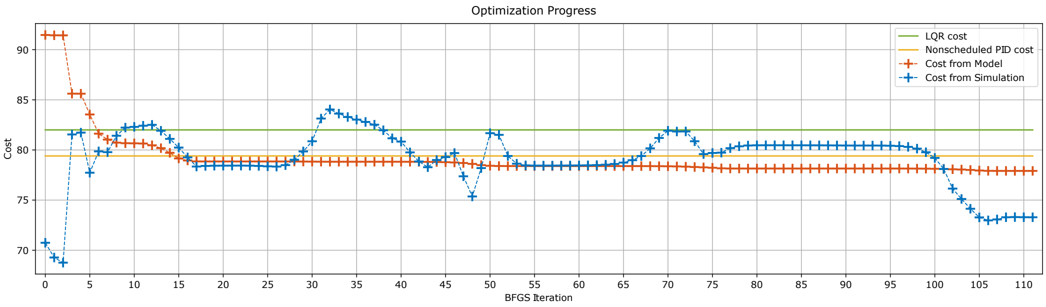


Figure 5.8: Concatenated optimization process for the scheduled PID.

Considering the demonstrated results and assuming that described problem of an extreme gains change is application specific, it was decided to apply the developed method to the real control system for further testing and evaluation.

6 Application to Autonomous RC Car

In order to prove the applicability of the designed algorithm to the real physical systems, it was exploited to tune the scheduled PID controllers of the autonomous RC car, which was developed as part of master thesis of Alexander Wischnewski [35]. This chapter presents the description of a system with all modifications which were made during the method integration, as well as the approaches to the modeling of system dynamics.

6.1 System Description

6.1.1 Physical Prototype

The physical prototype of the autonomous system is based on a commercial RC car of scale 1:6 to a real vehicle. This car is controlled by the steering and throttle input commands. Four-wheel system, which transmits the traction forces, is powered by a Brushed DC Motor. The front-wheel steering is performed by two separate but mechanically linked servo-motors. The frame is mounted on a single-wheel suspension with a spring-damper combination. Several parameters such as camber, toe and spring-preload are available for tuning.

The autonomous driving functionality on a physical prototype is implemented using the Pixhawk PX4 micro-controller, however this choice is not essential. The Pixhawk PX4 directly provides the measurements of 3D-accelerometer, 3D-magnetometer, 3D-gyroscope and GPS. Additionally, angular positions of the steering servo-motors are obtained from built-in rotary potentiometers and the Pixhawk's AD-converter interface. The sensors have a different sample rates which are lower than the minimal limit set for runtime of control software. The oversampling approach was implemented. The Sensors output the raw physical quantities and have to be calibrated, however this part is not the scope of this thesis.

Further details about the physical prototype [35] are irrelevant for this thesis. Taking the results from Wischnewski as a proof by demonstration, it was concluded that the system can be optimally controlled by the manually tuned PID scheduled on the vehicle velocity, thus is appropriate for application of our method.

6.1.2 Simulink Model

The controller for autonomous driving was developed in Simulink, using the precise RC car model provided by Bosch. Additionally, a software package which configures MATLAB Coder to work with the Pixhawk PX4 and provides an actuator and SD-card interfaces were installed. The resulting Simulink model was provided as well as the physical prototype and includes the following functional modules:

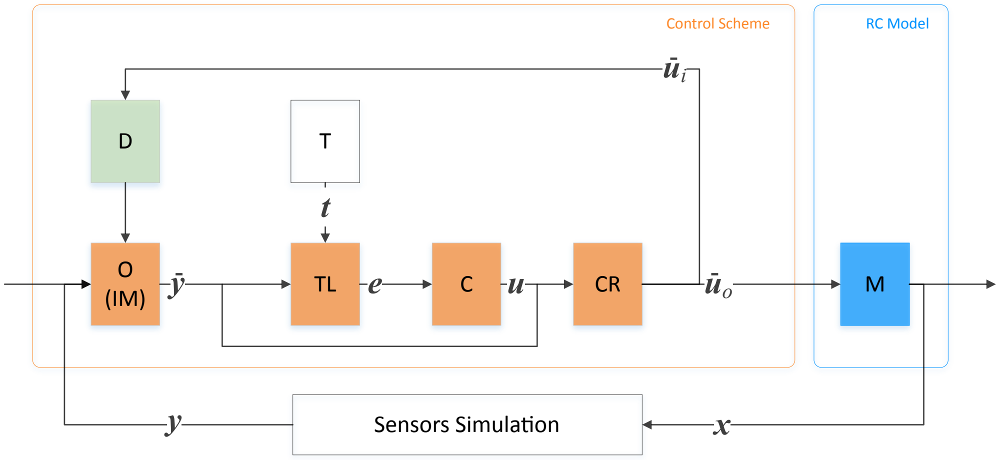


Figure 6.1: Simplified control scheme of the autonomous RC car.

1. High-level state-flow control
2. The Sensor and actuator simulators
3. The control mechanism for autonomous driving
4. Precise RC car model.
5. Data logging and post-processing routines
6. Controller parametrization via SD-card interface.

The following sections are focused on the control mechanism for autonomous driving module, other modules [35] are not in the scope of this thesis.

6.2 Control Scheme Analysis

As a part the master thesis Wischnewski [35] introduced the control mechanism for the RC car, which is able to follow the predefined discrete target trajectory. The control mechanism includes two parallel controllers, which can be switched during the simulation and execution. The controllers are nonlinear IMC controller, and multivariate scheduled PID. IMC was developed as a primary controller and PID was used for the benchmarking. However, since the nonlinear IMC is out of scope for this work, only the multivariate PID is considered.

A simplified control scheme is presented in Figure 6.1. The block M stands for the precise RC car model, which has to be controlled. Inputs $\bar{\boldsymbol{u}}_o$ of this model are throttle command u_t in percents $+/-100$, and steering command u_s in radians $+/-0.2$ (Table 6.1).

The model output state $\boldsymbol{x}$ with the information provided in Table 6.1, is transmitted through the outer loop and enters the Sensors Simulation. $\boldsymbol{x}$ is post-processed with Sensors Simulators (noise and delays are added) in order to account for the real hardware behavior. The resulting state observation $\boldsymbol{y}$ obtained from the sensors is transmitted to the State Observer(O). All sensor measurements

State	Information		Range	Unit
$\boldsymbol{u}$	u_{lat}	Lateral PID output	-	$^{rad}/_{s}$
	u_{lon}	Longitudinal PID output	-	$^{m}/_{s^2}$
$\bar{\boldsymbol{u}}_o$	u_{s}	Steering angle	$+/-0.2$	rad
	u_{t}	Throttle	$+/-100$	$perc$
$\bar{\boldsymbol{u}}_i$	$\dot{\theta}_*$	Requested yaw-rate	-	$^{rad}/_{s}$
	a_*	Requested acceleration	-	$^{m}/_{s^2}$
	γ	Estimated side-slip angle	-	rad
$\boldsymbol{x}$	x, y	Vehicle position	-	m
	θ	Vehicle orientation	-	rad
	v	Vehicle velocity	-	$^{m}/_{s}$
	$\dot{\theta}$	Vehicle yaw rate	-	$^{rad}/_{s}$
	a	Vehicle acceleration	-	$^{m}/_{s^2}$
$\bar{\boldsymbol{y}}$	$x_{\text{k}}, y_{\text{k}}$	Filtered position	-	m
	θ_{k}	Filtered orientation	$+/-3.14$	rad
	v_{k}	Filtered velocity	-	$^{m}/_{s}$
	$\dot{\theta}_{\text{k}}$	Filtered yaw-rate	-	$^{rad}/_{s}$
	γ_{k}	Filtered side-slip angle	-	rad
$\boldsymbol{t}$	s	Path along the trajectory	-	m
	$x_{\text{tr}}, y_{\text{tr}}$	Position in local coordinates	-	m
	k	Curvature	-	$^{rad}/_{s}$
	v_{tr}	Target velocity	-	$^{m}/_{s}$
	θ_{tr}	Target orientation	-	rad
	$\dot{\theta}_{\text{tr}}$	Target yaw-rate	-	$^{rad}/_{s}$
$\boldsymbol{e}$	e_{lat}	Lateral error	-	m
	e_{lon}	Longitudinal error	-	m

Table 6.1: The data transmitted within the control scheme of the autonomous RC car.

have different delays and sampling times. Therefore, the State Observer (O in Figure 6.1) has a convoluted functionality of a sensors fusion which includes an Unscented Kalman Filter (UKF) and the internal process model.

UKF outputs the state estimation based on the information measured by sensors. Nonetheless, the complete information of a system state required for UKF to compute an estimation is available with a delay of $3t$ where t is a system sampling time equal to 0.02. That is why the State Observer does also include the internal simplified model of RC car dynamics. The internal model is used to correct the chain of UKF observations made with an outdated sensor measurements. The internal model inputs $\bar{\boldsymbol{u}}_i$ are Requested yaw-rate $\dot{\theta}_*$, Requested acceleration a_*, and Estimated side-slip angle γ. They are transmitted via the inner loop (Figure 6.1) from the Control Router (CR) and delayed for three time steps using the delay stack (D). The internal model then predicts, using the new sensed information related to time-step (t-3) and all control outputs which were commanded since that

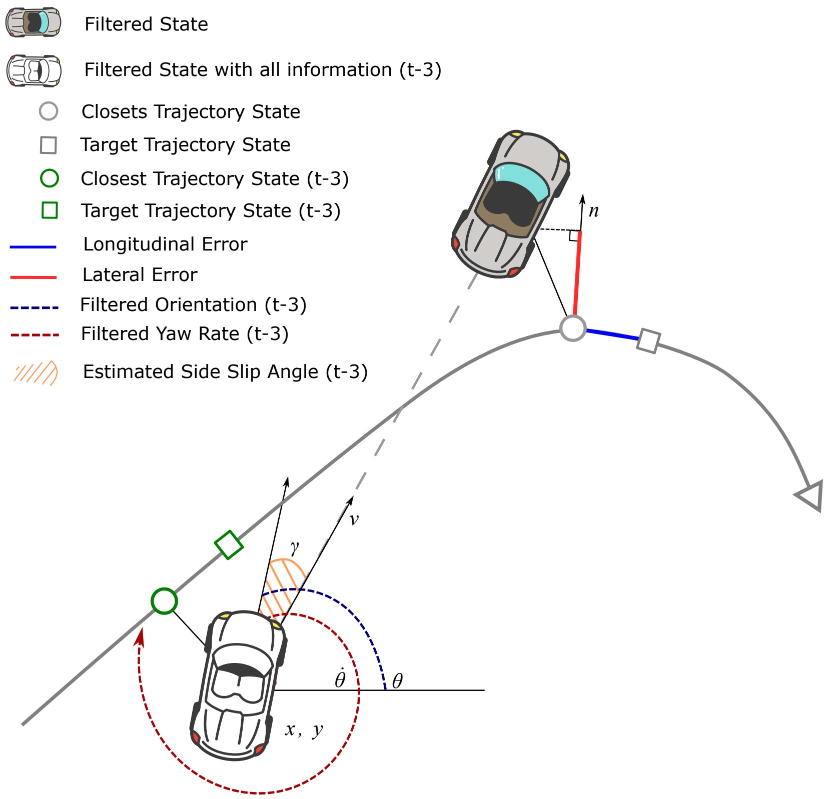
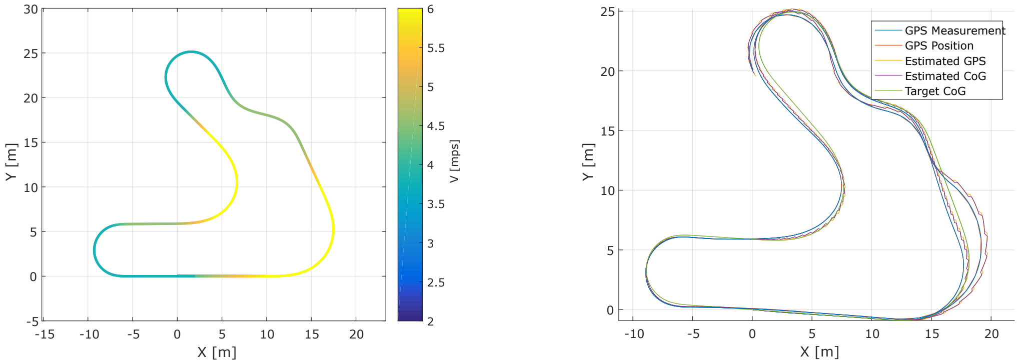

Filtered State
Filtered State with all information (t-3)
Closets Trajectory State
Target Trajectory State
Closest Trajectory State (t-3)
Target Trajectory State (t-3)
Longitudinal Error
Lateral Error
Filtered Orientation (t-3)
Filtered Yaw Rate (t-3)
Estimated Side Slip Angle (t-3)
n
γ
v
θ̇
θ
x, y

Figure 6.2: The definition of longitudinal and lateral error used for PID control.

point (stored in delay stack), what will be the correct estimation at the current step. The Kalman state estimate $\bar{\boldsymbol{y}}$ contains Filtered vehicle position x_k and y_k, heading θ_k, velocity v_k, yaw-rate $\dot{\theta}_\mathrm{k}$ and side-slip angle γ_k. The details are provided in Table 6.1.

Next, the estimated Kalman state is transmitted to the Trajectory Localization module (TL at Figure 6.1). Another input of TL is the trajectory $\boldsymbol{t}$, provided as a set of discrete points with the information given in Table 6.1. The trajectory is stored as a parameter in a Matlab Workspace during the simulation, but loaded from the SD card during the run on the physical system. First, the coordinates of the trajectory points are transformed to the coordinate frame fixed at initial vehicle position with Y axes aligned to vehicle orientation θ_k. This coordinate frame holds for both, Kalman filtered and target positions, until the system finishes a lap.

Afterwards, TL calculates the lateral and longitudinal errors are later used as an inputs to PID controllers. The Definition of these two errors is schematically presented in Figure 6.2. The transparent car represents the state at time-step $t-3$ when system had last sensor information update. The colored car stands for the filtered state at the current time t, corrected by the information from $t-3$. In order to compute the errors, the closest point to the filtered vehicle state on a target trajectory is calculated (gray circle in Figure 6.2). The Closest point is obtained in three steps. First, the path coordinate of the anticipated closest point s_c is integrated from $s_{\mathrm{c},t-1}$ (previously closest



(a) The target trajectory with velocity profile

(b) RC car trajectory

Figure 6.3: The target and actual trajectories of the autonomous RC car simulation (taken from [35]).

one) and filtered vehicle velocity v_k. At the first iteration s_c is set to 0. Then,the set of temporary points $s_\diamond$ is sampled from a region around s_c plus some safety margin. For every $s_\diamond$ position is interpolated from the discrete trajectory points. The euclidean distance is calculated then between every sampled point and the filtered position of the vehicle x_k, y_k. According to the euclidean distance, the closest point on a trajectory is selected. Once the closest point is defined, a difference between the filtered position and the closest point is projected to the normal at s_c. This projection is taken as the lateral error (red bar at Figure 6.1). The longitudinal error (blue bar) is calculated as a difference in s coordinate between the closest point and the target one, which is obtained via the integration of the target velocity.

The calculated errors are transmitted to The Controller (C) module, which contains two parallel PID controllers with the custom derivate terms scheduled on the velocity. As a derivative term of the lateral PID a difference between the filtered vehicle orientation and orientation in closest trajectory point (interpolated) is taken. The derivative term of the longitudinal controller is calculated as a difference in target and filtered velocities.

Last module in presented scheme is the Control Router (CR). CR takes a filtered Kalman state as an additional input. First, CR computes requested yaw-rate $\dot{\theta}_*$, acceleration a_* and estimated side-slip angle γ using the PID outputs $\boldsymbol{u}$ and the Kalman state $\bar{\boldsymbol{y}}$. Then this information is transmitted to the Delay stack (D) via the internal loop for further use in the State Observer (O). Then the Router performs the post-processing of a PIDs outputs. CR does include the inverse dynamics model of steering actuator, which is used to transform the lateral PID output and speedup the steering response [35]. Additionally the output of the longitudinal PID is transformed using a special look-up table to compensate for different acceleration dynamics of the RC car at different velocities [35].

Considering that the original scheduling mechanism ought to be replaced with the parametrized GPs and is not required for the method application, it was removed from the control mechanism. The gains of the remained plain PID controllers were slightly adjusted to give the performance comparable to the scheduled PID. The resulting trajectory of the autonomous RC car with the manually tuned PID controllers is presented in Figure 6.3b. Figure 6.3a demonstrates the target trajectory with the velocity profile which is used during the simulations. The trajectory starts at $(0, 0)$ and goes to the positive X direction. During the first lap the trajectory is followed more precisely since the target velocity is reduced for the sensor calibration purposes [35]. Starting from the second lap, one can observe a persistent error in the bottom right corner of a trajectory. However, the manually tuned PIDs are still able to follow the defined target trajectory with relatively small accumulated longitudinal and lateral errors. That is why this system was considered to be an appropriate application base for the developed algorithm. The results of the PID tuned by Wischnewski are taken as a benchmark. All experiments conducted from now on use the target trajectory from Figure 6.3a.

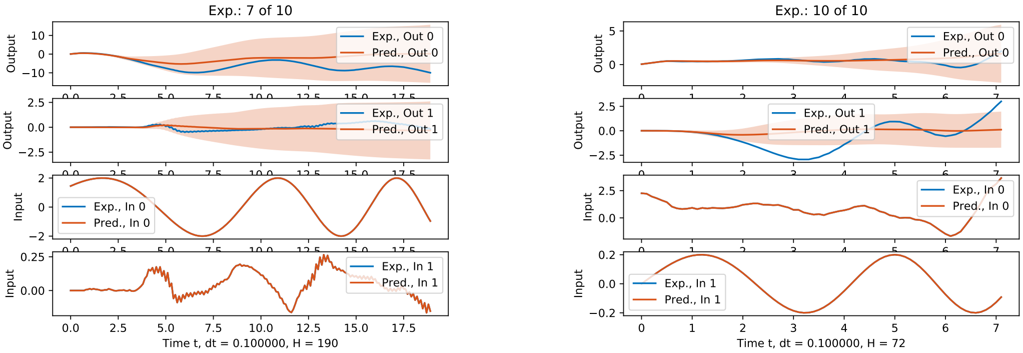
6.3 Dynamics Modeling

This section addresses learning of initial probabilistic dynamics model which corresponds to step 4 in Algorithm 3.1. The errors definition do not correspond to the linear transformation of a vehicle state and desired trajectory anymore. Thus, the integration of error calculation into the extended state $\tilde{z}$ requires additional approximations for the joint distribution $p(\tilde{z})$. Therefore, it was decided to model the error dynamics directly. Initial GP has to output $\boldsymbol{e}_t$ instead of $\boldsymbol{x}_t$ (Section 3.3).

Initially it was decided to encapsulate the entire control scheme from Figure 6.1 into a probabilistic model, which computes $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t)$. Here, however, two points have to be stressed out. First of all, $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t)$ defines a trajectory specific model, i.e. model of the dynamics for the target trajectory which was used during the training only. Instead of learning separate models for every trajectory, probabilistic model of error dynamics has to be trajectory dependent. Thus, a trajectory information has to be included as an input of a model and $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t, \boldsymbol{t}_t)$ has to be learned. However, addition of the trajectory information entails the filtered vehicle state $\bar{\boldsymbol{y}}$, which has to be included into the model due to the errors causality. The trajectory dependent probabilistic model of error dynamics looks as $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t, \boldsymbol{t}_t, \bar{\boldsymbol{y}})$. Moreover, the model has to be spatially invariant, since for the given trajectory, error dynamics does not depend on global location of a vehicle and current target point, but only on their relative positions. Taking into account the above assumption, all spatially-dependent information of state $\bar{\boldsymbol{y}}$ and trajectory $\boldsymbol{t}$ has to be avoided.

To obtain the training data we use the rollouts an optimal PID tuned by Wischnewski and a suboptimal PID with slightly different integral and derivative gains. Additionally, the rollouts with different chirp signals as an input to excite the system and the random walk were used. All rollouts were cut to the first 50 seconds. This time horizon covers the first two laps of a target trajectory which have different velocity profile. That is why, this information should be sufficient to learn the model of error dynamics. To reduce the amount of collected data and improve the sparse GP performance, experiments were downsampled to $10Hz$. The training used 274.68 seconds of system interaction in total. Such big amount of training data is required due to the system's complexity. Test experiments are build from a similar rollouts. During the training a different NARX history length and amount of sparse GP inducing points were tried out.

The results of test experiments for trajectory specific model $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t)$ are presented in Figure 6.4, where Input 0 and 1 stand for the longitudinal and lateral PID outputs u_{lon}, u_{lat} respectively. Output 0 is a longitudinal error and 1 is a lateral error. Figure 6.4a shows that the model is able to predict the errors dynamics for the changing longitudinal controller output and the constant lateral controller output (approximately the first 3 seconds). When the lateral output starts to deviate (after the first 3 seconds), the model gradually falls to the prior variance. Figure 6.4b proves that GPs are unable to learn the error dynamics for a change in output signal of the lateral controller. This brings us to



(a) Test experiment 1

(b) Test experiment 2

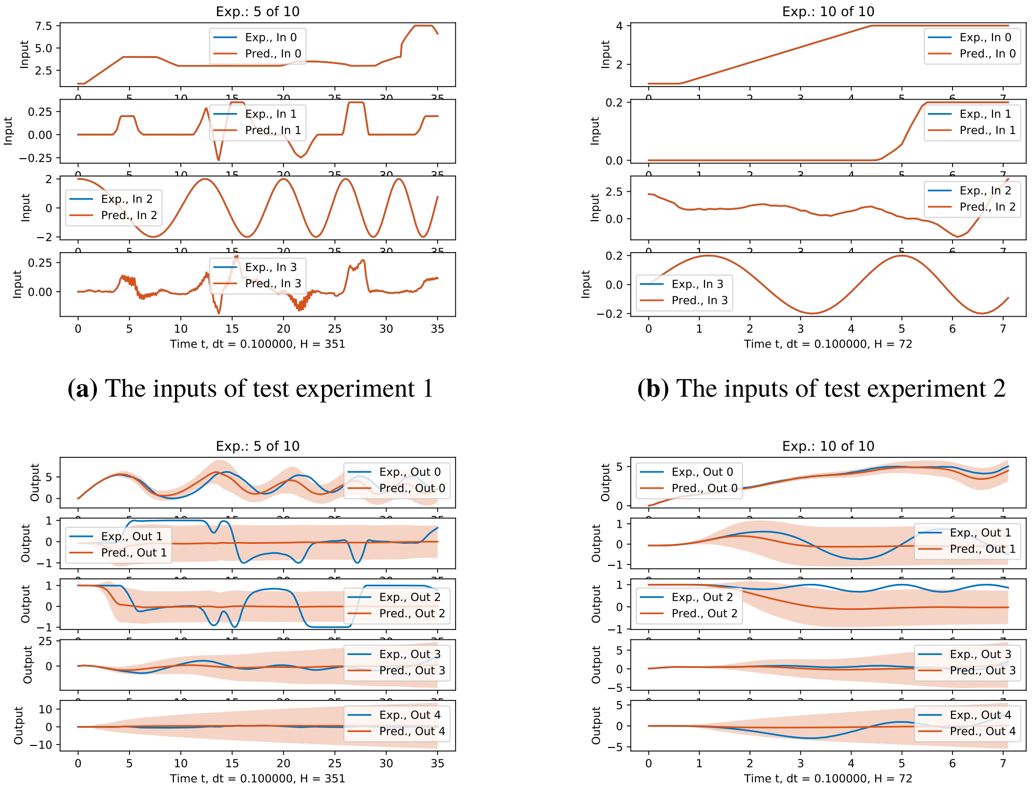
Figure 6.4: Tests of trajectory specific model of the errors dynamics.

the assumption that the steering control command has a more convoluted impact on system and error dynamics. For this particular experiments the NARX structure includes 6 history states for the inputs and outputs. The sparse GP uses 400 inducing points. However, the similar results are obtained for all configurations of NARX state and inducing points which were tried during the training.

As inputs to the trajectory dependent model $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t, \boldsymbol{t}_t)$, target velocity v_{tr} and trajectory curvature k, an outputs the longitudinal and lateral PID controllers were used (Table 6.1). Filtered vehicle velocity v_{k} and filtered orientation θ_{k} were used as model states. Vehicle orientation θ_{k} is framed from $-\pi$ to π. However, original non-framed orientation signal was recorded. Then *cos* and *sin* were applied to it.

The test experiments of trajectory dependent, model are given in Figure 6.5. For readability, the outputs and inputs are separated into different plots. The inputs are presented in the following order: target velocity v_{tr}, trajectory curvature k, u_{lon} and u_{lat}. The outputs are filtered velocity, *sin* of orientation, *cos* of orientation, longitudinal and lateral errors. Similarly to the trajectory specific model, $p(\boldsymbol{e}_{t+1}|\boldsymbol{e}_t, \boldsymbol{u}_t, \boldsymbol{t}_t)$ cannot learn the dynamics of errors for a changing output of the lateral controller. However, Figure 6.5a and Figure 6.5a demonstrate that the model does learn the dependency between the control outputs and the filtered vehicle velocity. From that one can assume that the velocity and errors dynamics of the learned model is not strongly coupled. The presented experiments use the NARX structure with 2 history states for the vehicle velocity, *sin* and *cos* of the vehicle orientation, the target velocity and the trajectory curvature. The history length for the errors and the control outputs is equal to 1. GP uses 400 inducing points. Such configuration demonstrated the best performance among the all which we tried during the training. Additionally, different spatially-independent information such as target and filtered yaw-rates, filtered side-slip angles etc. was included during the training. Nonetheless, none of these attempts resulted in accurate model.

In order to validate the assumption about the vehicle velocity and error dynamics coupling, the optimized GP hyper-parameters were investigated. Hyper-parameters are presented in Figure 6.6. In the left table length-scales are printed. Each row corresponds to the trained GP for a particular model output. Every column shows length-scales of all trained GPs for a corresponding model input. The large length-scale means that the input information is irrelevant for the model's output.



(a) The inputs of test experiment 1

(b) The inputs of test experiment 2

(c) The outputs of test experiment 2

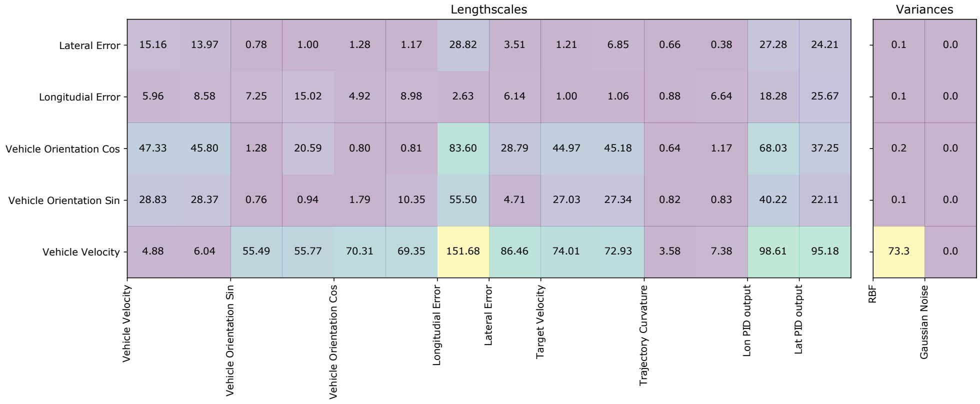
(d) The outputs of test experiment 1

Figure 6.5: Tests of trajectory dependent model of the errors dynamics.

According to the length-scales, the lateral error and the vehicle velocity is indeed loosely coupled. However, the length-scales of the longitudinal error for the vehicle velocity has a reasonable value. The upper right corner of this table shows the length-scales of a lateral and longitudinal errors for the PID outputs. According to our expectations, the length-scales of upper right corner should be about the same size or smaller comparing to all others, which did not hold true. Hyper-parameters from Figure 6.6 indicates that trained model of error dynamics is relatively insensible to the control commands, which seems incorrect.

After analysis of the obtained results it was assumed that, most probably, the internal model of the Observer obscures the dependency between the control commands and errors change. It was also possible that some important information was missed, which prevents GP from accurate modeling of the error dynamics. Therefore, the information from the inner loop $\bar{\boldsymbol{u}}_i$ and the filtered state at time step $t-3$ was added to help GP approximate the internal model's behavior.

The results of test experiments are provided in Figure 6.7. It contains outputs only, since the inputs of these experiments are the same as we used previously. The model inputs are provided at Figure 6.5a and Figure 6.5b. The outputs are going in following order: the filtered velocity v_k, sin , cos of the non-framed filtered orientation θ_k, the requested acceleration a_*, the requested yaw-rate $\dot{\theta}_*$, the longitudinal and lateral errors. Similarly to the previous results, Figure 6.7a and Figure 6.7a show the precise dynamics model for the velocity and acceleration, but not for the errors. For the



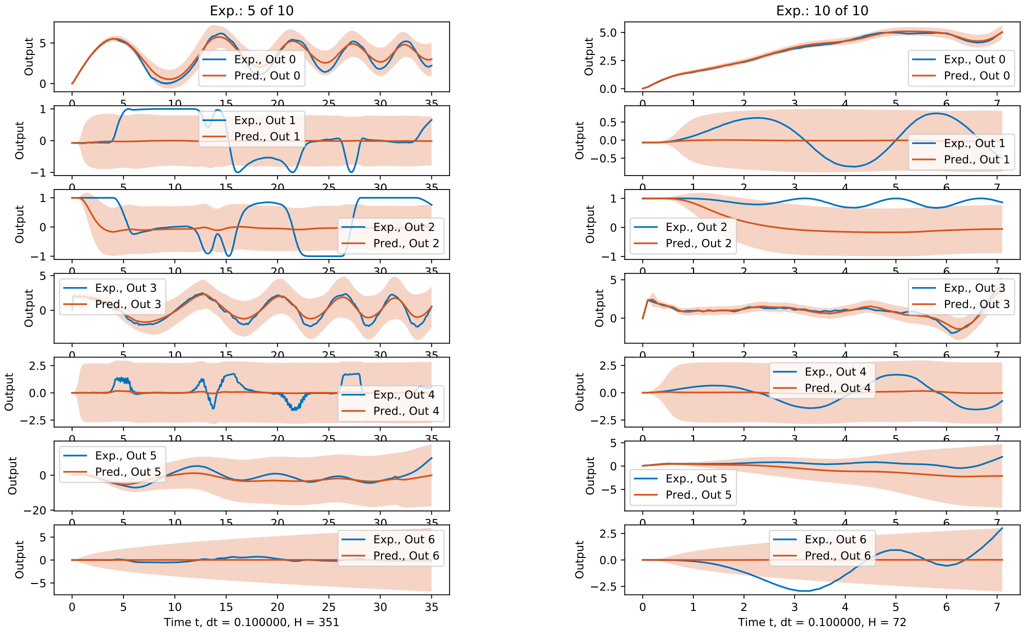
Lengthscales

	Vehicle Velocity		Vehicle Orientation Sin		Vehicle Orientation Cos		Longitudial Error	Lateral Error	Target Velocity		Trajectory Curvature		Lon PID output	Lat PID output
Lateral Error	15.16	13.97	0.78	1.00	1.28	1.17	28.82	3.51	1.21	6.85	0.66	0.38	27.28	24.21
Longitudial Error	5.96	8.58	7.25	15.02	4.92	8.98	2.63	6.14	1.00	1.06	0.88	6.64	18.28	25.67
Vehicle Orientation Cos	47.33	45.80	1.28	20.59	0.80	0.81	83.60	28.79	44.97	45.18	0.64	1.17	68.03	37.25
Vehicle Orientation Sin	28.83	28.37	0.76	0.94	1.79	10.35	55.50	4.71	27.03	27.34	0.82	0.83	40.22	22.11
Vehicle Velocity	4.88	6.04	55.49	55.77	70.31	69.35	151.68	86.46	74.01	72.93	3.58	7.38	98.61	95.18

Variances

	RBF	Gaussian Noise
Lateral Error	0.1	0.0
Longitudial Error	0.1	0.0
Vehicle Orientation Cos	0.2	0.0
Vehicle Orientation Sin	0.1	0.0
Vehicle Velocity	73.3	0.0

Figure 6.6: The hyper-parameters of the trajectory dependent model.



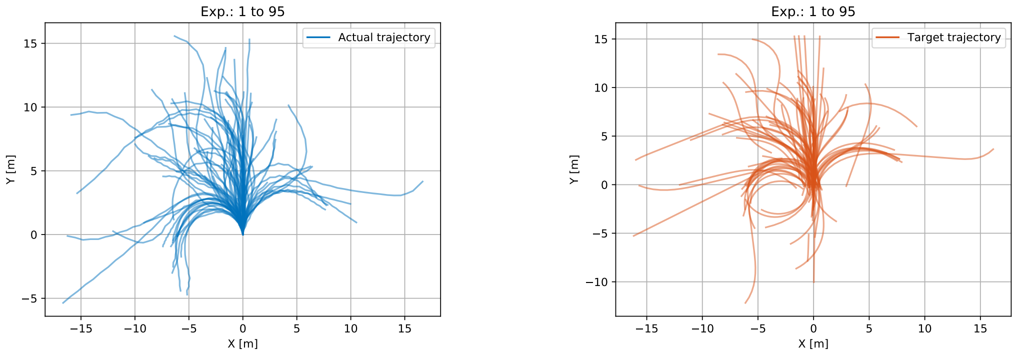
(a) Test experiment 1

(b) Test experiment 2

Figure 6.7: Tests of trajectory dependent model of the errors dynamics with additional states.

presented experiments, the NARX structure with 4 history states for every input and output is used. The number of inducing points is 500 due to the big amount of additional training data. The learned hyper-parameters are not presented here, since the pattern is similar to the previous model.

After the failure of the previous approach the assumption was made that relations between the spatial information and errors are too strong to neglect. That is why a ways to include some spatial information into our model but preserve the spatial invariance of dynamics were required. First,



(a) Actual trajectory spans

(b) Target trajectory spans

Figure 6.8: The RC car experiment subdivision.

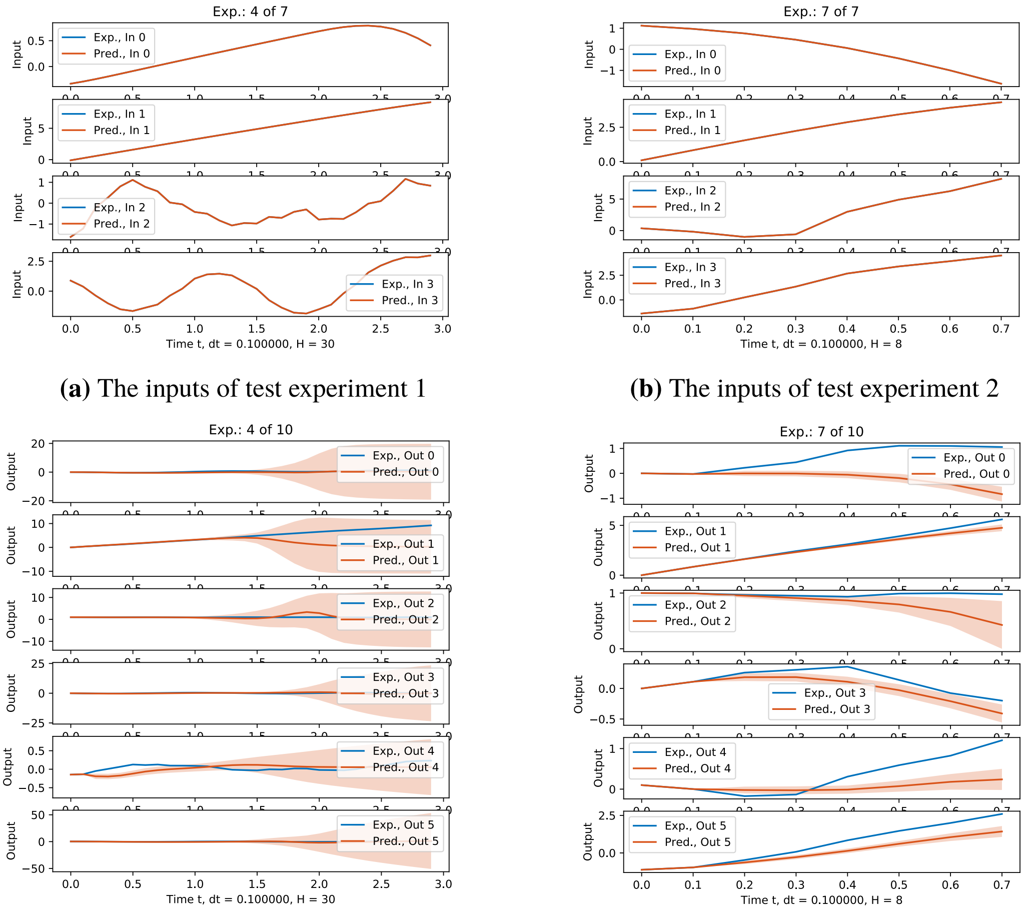
it was considered to use the location of one or several target trajectory points in a local vehicle coordinate frame as an input to a model. However, these points are obtained via the nonlinear transformation of a state and trajectory information, thus additional derivations are required to approximate the extended joint state distribution $p(\tilde{z})$.

Next, it was decided to train a local model with a relatively short horizon for different segments of target trajectory. With this approach, spatially-dependent data such as position can be used while preserving the spatial invariance for the global error dynamics. One experiment can be subdivided into the number of spans with time the horizon equal to 3 seconds. Vehicle and target positions then transformed to a coordinate frame fixed at initial filtered position $x^i_{\mathrm{k},0}$, $y^i_{\mathrm{k},0}$ for the span i with Y axes aligned to an initial orientation $\theta^i_{\mathrm{k},0}$. As an example, the subdivision of a train experiments into 95 spans is presented in Figure 6.8. Since the coordinates are transformed to the initial filtered position, the actual sub-trajectories in Figure 6.8a always start from $(0, 0)$, which is not true for the target ones in Figure 6.8b.

According to the Bellman principle of optimality [32], every part of an optimal path is also optimal. Therefore, PID gains can be computed such that RC car follows optimally every target trajectory span obtained via the subdivision of experiments. Given that the set of sub-trajectories is reach enough one can optimize PID for the entire trajectory.

Test experiments for a local model with spatial information are presented in Figure 6.9. The target position in a local span coordinates (Input 0, 1) and outputs of PID (Input 2, 3) were taken as an inputs of the model. The output information consists of filtered position (Output 0, 1), *sin* and *cos* of orientation (Output 2, 3), longitudinal and lateral errors (Output 4, 5) respectively.

The test experiments are demonstrated in Figure 6.9. According to Figure 6.9d the local model has a better performance for the monotonic input commands. The predicted errors dynamics (last two outputs) has a relatively small variance comparing to the previous approaches. Even though it does have some bias, it can be mitigated with longer history in NARX state or additional input information. However, for the periodic inputs the leaned model falls to the prior variance after the half of the rollout. This observation holds for all test experiments independently from the configuration of the local dynamics model. For the plotted test experiments we used the NARX structure with the history length of 3 for all inputs and outputs GPs have a 200 inducing points.



(a) The inputs of test experiment 1

(b) The inputs of test experiment 2

(c) The outputs of test experiment 2

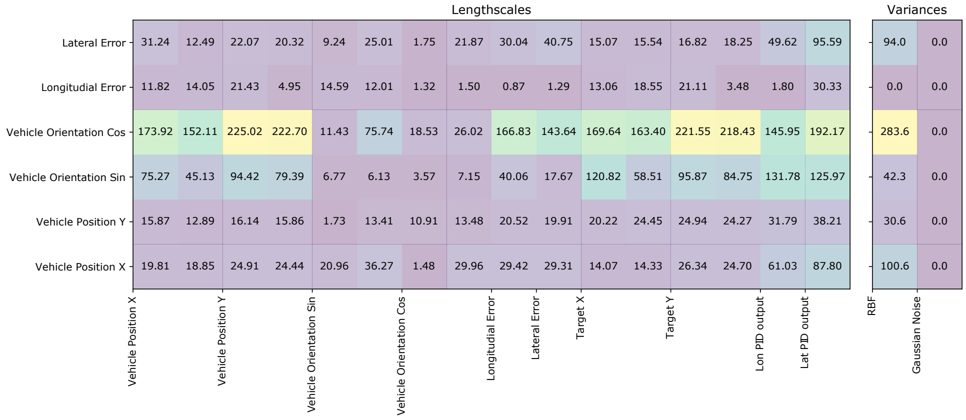
(d) The outputs of test experiment 1

Figure 6.9: Tests of the local model of the errors dynamics.

Optimized GP hyper-parameters are plotted in Figure 6.10. According to the length-scales, local model learns the dependency between the longitudinal error and the corresponding controller output. Nevertheless, the lateral error is still relatively insensible to steering command, i.e has the high length-scale. It is clear that such performance of the initial model is not sufficient for PILCO application.

6.4 Control Redesign

As it was demonstrated in Section 6.3, none of the proposed models was able to learn neither the direct errors dynamics nor filtered states dynamics. After conduction of several tests and further analysis, three possible causes of obtained poor results were proposed. First of all, sophisticated definition of errors, which involves the interpolation of a closest point on a target trajectory, can obscure the correlation between the errors and control commands. This assumption most probably holds for situations, where the closest point on a trajectory is not changing with respect to change in



Lateral Error	31.24	12.49	22.07	20.32	9.24	25.01	1.75	21.87	30.04	40.75	15.07	15.54	16.82	18.25	49.62	95.59	94.0	0.0
Longitudial Error	11.82	14.05	21.43	4.95	14.59	12.01	1.32	1.50	0.87	1.29	13.06	18.55	21.11	3.48	1.80	30.33	0.0	0.0
Vehicle Orientation Cos	173.92	152.11	225.02	222.70	11.43	75.74	18.53	26.02	166.83	143.64	169.64	163.40	221.55	218.43	145.95	192.17	283.6	0.0
Vehicle Orientation Sin	75.27	45.13	94.42	79.39	6.77	6.13	3.57	7.15	40.06	17.67	120.82	58.51	95.87	84.75	131.78	125.97	42.3	0.0
Vehicle Position Y	15.87	12.89	16.14	15.86	1.73	13.41	10.91	13.48	20.52	19.91	20.22	24.45	24.94	24.27	31.79	38.21	30.6	0.0
Vehicle Position X	19.81	18.85	24.91	24.44	20.96	36.27	1.48	29.96	29.42	29.31	14.07	14.33	26.34	24.70	61.03	87.80	100.6	0.0

Figure 6.10: The hyper-parameters of the local model of the errors dynamics.

vehicle position, for instance at some steep turns. As an addition, UKF can also disturb the dynamics of state observations and make GP unable to learn it. Finally, the correction of an estimated states using the internal model, is over complicating the problem and apparently deteriorates the learning results.

6.5 Validation Experiment

In order to validate the proposed assumptions, it was tested if GP can learn the dynamics of precise the model (block M at Figure 6.2) directly, before the application of UKF. Additional modules, which record the full output state $\boldsymbol{x}$ of the model and post processed control commands $\bar{\boldsymbol{u}}_o$ (throttle and steering), were included to the Simulink model. For data acquisition same experiments as described in Section 6.3 were used.

It was decided to follow the latter setup from the Section 6.3, where the local, position dependent, model of RC is learned by means of subdividing experiments to the 3 second intervals. As previously, the local coordinate frame of each subdivision is centered at initial vehicle position and Y axis coincides with vehicle heading.

For the training, first 50 seconds of each experiment are taken. Since this time horizon covers first two laps which have different velocity profile, one can guarantee that such data set is reach enough to learn the RC car dynamics. As previously, this experiments were subdivided into 3 sec intervals, which resulted in 71 training experiments.

The throttle and steering commands are taken as an inputs with the history length of 2 in order to give the model a possibility to learn a derivatives of inputs. The outputs are the vehicle coordinates x and y. Since the orientation and velocity are not used, it is necessary to give the model a history, so it can learn this information from differences in positions. After several experiments with the history length, it was defined that the length of 3 is sufficient and results in high accuracy.

Subset of four test experiments is presented in Figure 6.11. On the left hand side one can see the model inputs, throttle and steering respectively. On the right hand side GP outputs are presented. The first output is x coordinate and the second one is y. As one can see from these plots, the trained GP is able to learn the dynamics of RC model quite precisely, which proves the assumption regarding the state observer and complexity of errors definition.

Obtained results confirm that autonomous RC car system is appropriate for application of developed method, however the observer and control logic have to be redefined and reimplemented.

6.6 New Errors Definition

As it was stated in Section 6.5, in order to apply the developed method the redefinition of observer and control scheme is required. Since it is possible to test the derived PILCO extension completely in a simulation setup, the observer is omitted for now, main focus is put on a new control mechanism. As previously, the control scheme will contain two disjoint PID controllers: one for the lateral error and another one for the longitudinal. However, derivative gains will be calculated using the numerical differentiation methods. Moreover, the post-processing with Control Router is also skipped, in order to simplify the following modeling of error dynamics. Therefore the control design boils down to the correct errors definition. Nonetheless, even this problem is not trivial. The errors dynamics have to be simple enough, i.e interpolation of closest trajectory point has to be omitted, in order to be learned by GP. In the meantime, the errors definition has to be complete enough to ensure the possibility of the stable control. Since the Observer is not reimplemented, all errors are calculated with respect to the output state $\boldsymbol{x}$ of the RC car model directly. The integrated target points x_{tr}, y_{tr} are calculated by the Trajectory Localization

6.6.1 Lateral Error

Two options for lateral errors calculation are proposed: based on projection and based on angle. The first option, the lateral error based on a projection (Figure 6.12a) follows similar principle as the original lateral error. Instead of a closest point on a target trajectory, the difference vector is calculated between the vehicle position and integrated target point. Additionally, since the PID controllers without custom derivative terms are used, an error is calculated between the predicted state and target point at time step $t + 3$. This is aimed to avoid extreme steering commands. State prediction is obtained directly from the current velocity and orientation using the following equation:

$$\begin{bmatrix} x_{t+3} \\ y_{t+3} \end{bmatrix} = \begin{bmatrix} 3dt & 0 \\ 0 & 3dt \end{bmatrix} \begin{bmatrix} \cos(\theta)v \\ \sin(\theta)v \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \end{bmatrix}, \tag{6.1}$$

where θ is a vehicle orientation, v is vehicle velocity, x, y are a position coordinates and dt is sampling time. In contrast to the internal model of the Observer, this prediction does not accounting for the acceleration, side-slip angle and the change in a control commands. It goes without saying that the precision for a prediction is much lower then. Nevertheless, with such a short prediction horizon it should be accurate enough to build a stable controller. The difference vector between the prediction and the target is projected to the normal of the trajectory at an integrated point.

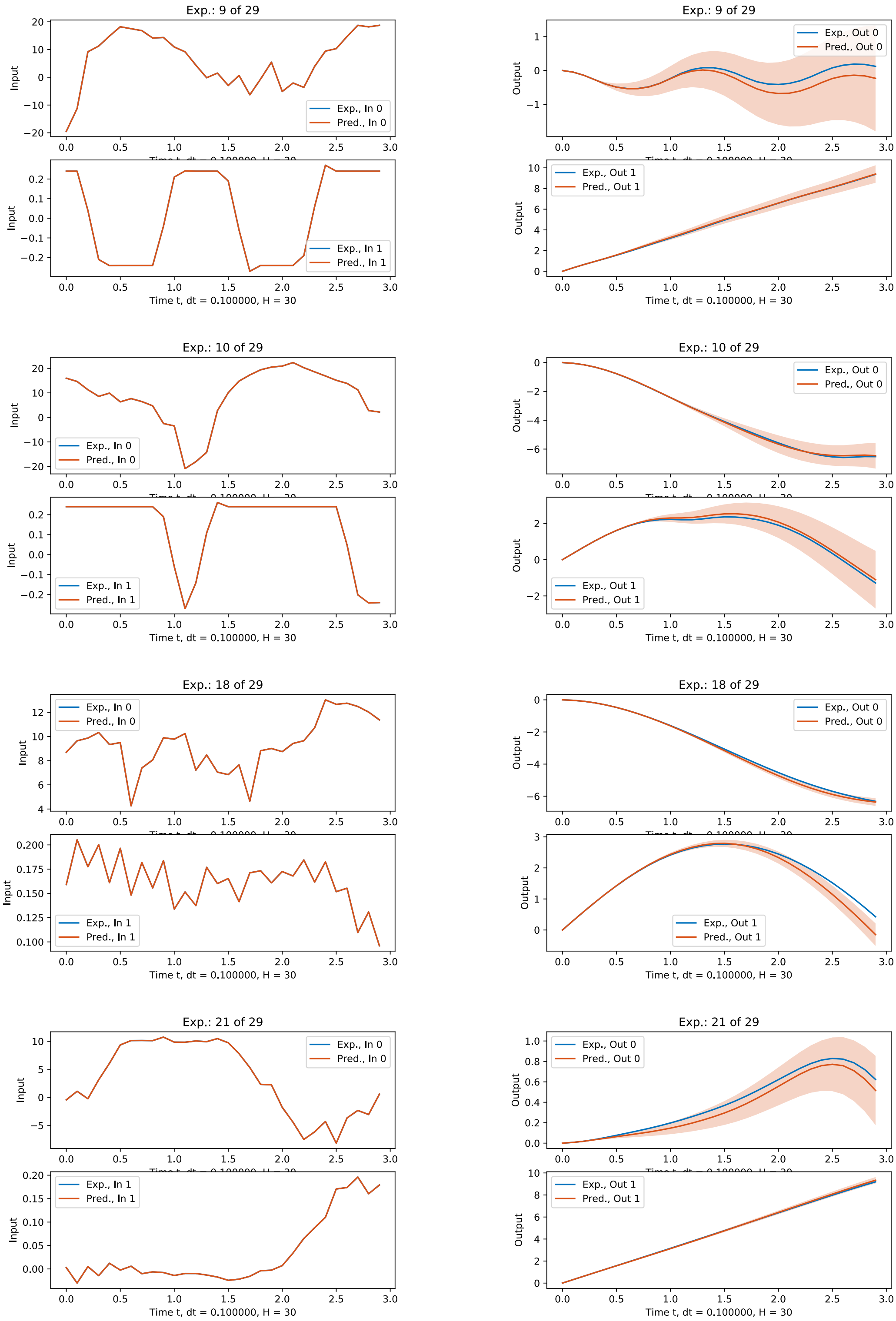
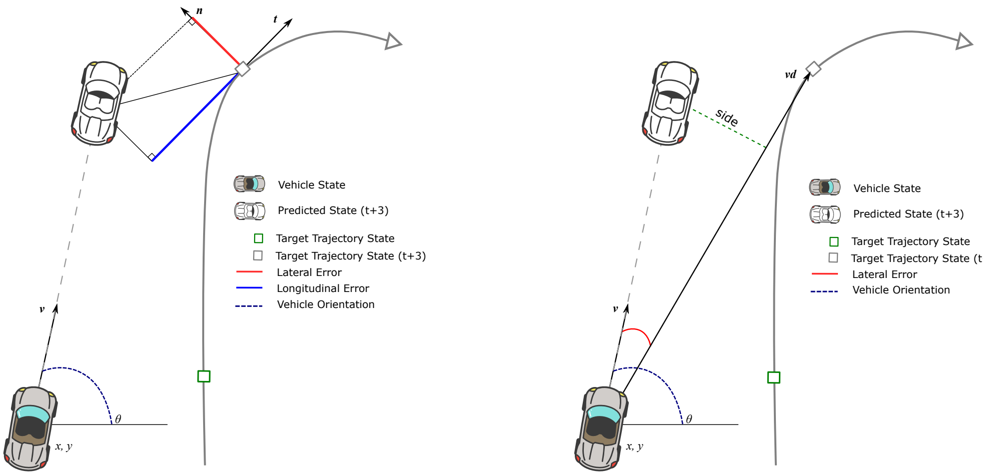


Figure 6.11: Validation experiment.



(a) The new errors based on the projections **(b)** The new lateral error based on the angle

Figure 6.12: The new error definitions.

The Second option is to define the lateral error as an angle between velocity vector $\boldsymbol{v}$ and $\boldsymbol{vd}$. $\boldsymbol{vd}$ is a difference vector among the current position x_t, y_t and the next integrated target $x_{\text{tr},t+3}$, $y_{\text{tr},t+3}$. The angle-based lateral error is demonstrated in Figure 6.12b. The angle is calculated using the dot product definition. However, since the codomain of $acos()$ is 0 to 2π, such definition is independent from the vehicle position relative to the target trajectory, which is insufficient to control the steering command. That is why the angle is additionally multiplied with the sign of determinant of following matrix:

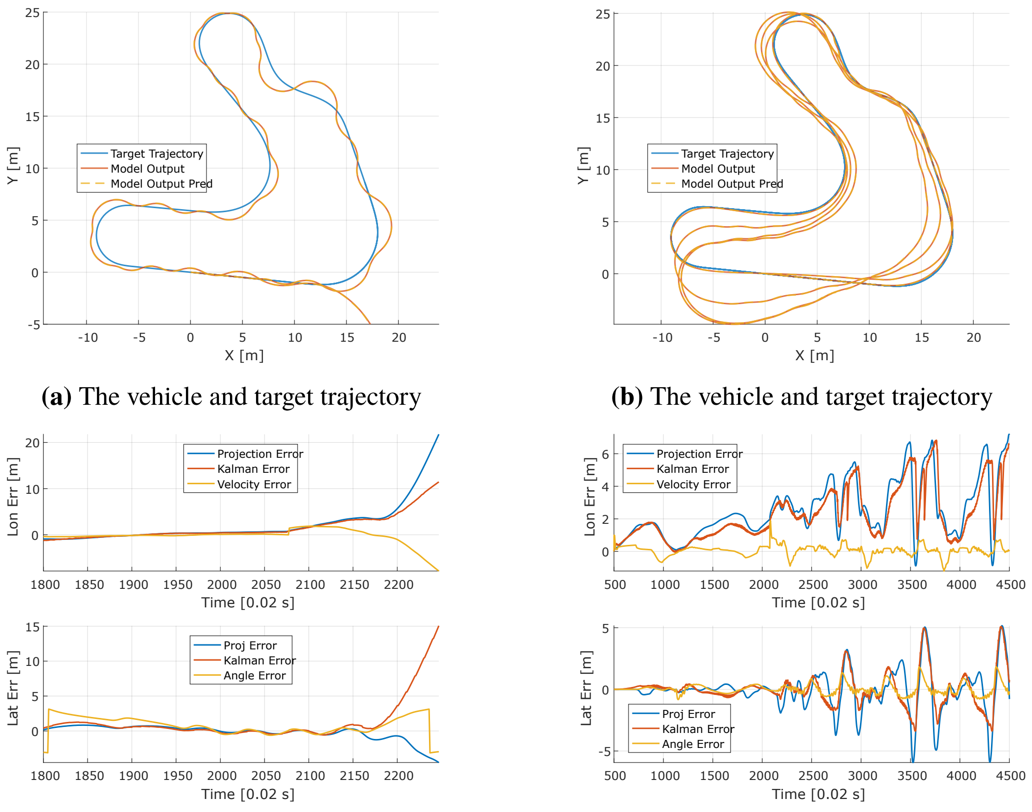
$$\begin{bmatrix} x_t - x_{\text{tr},t+3} & y_t - y_{\text{tr},t+3} \\ x_{t+3} - x_{\text{tr},t+3} & y_{t+3} - y_{\text{tr},t+3} \end{bmatrix},$$

which takes -1 or 1 dependent on the side of a predicted position with respect to $\boldsymbol{vd}$ vector and 0 if prediction lies on $\boldsymbol{vd}$.

6.6.2 Longitudinal Error

Similarly to the projection-based lateral error definition, the longitudinal error can be defined as a difference vector between the prediction and target projected to the tangent of $\boldsymbol{vd}$. However, such error definition permits for the undesirable effect for the steep turns. When the integrated target will be on the opposite side of a turn, longitudinal error can become either small or negative. The lateral and longitudinal projection-based errors are demonstrated in Figure 6.12a.

In addition, it was decided to use the simplified definition. The longitudinal error is obtained as a difference in vehicle and target velocities. One obvious drawback of such approach is that the controller will not compensate for the distance in path between current vehicle position and integrated target point. Nevertheless it can be assumed that in case when both controllers are tuned optimally such difference will be negligible and will not influence the final performance.



(a) The vehicle and target trajectory

(b) The vehicle and target trajectory

(c) The difference in errors

(d) The difference in errors

Figure 6.13: Test of the redefined errors.

6.7 New Errors Test

Since the mathematical proofs of convergence and stability are out of scope for this thesis project, values of newly defined errors, recorded during the simulation, have to resemble the original ones as much as possible.

For the first test experiment projection based lateral and longitudinal error definitions were used (left side of Figure 6.13). The manual tuning of PID gains for these errors did not result into a stable controller. That is why gains were selected such that the resulting simulation rollout clearly demonstrates the reasons of system instability. Figure 6.13a displays the target and actual trajectory of the vehicle. It also contains the predicted trajectory, which is really difficult to distinguish because of the overlap. This overlap indicates that for such short horizon, our prediction approximates next vehicle position with sufficient accuracy.

As one can see, the controller is oscillating even for the first lap with the reduced velocity. This osculation was present in all simulated rollouts during the tuning, which leads to a conclusion that it cannot be mitigated for such error definition without post-processing. However, the main cause of system destabilization is that vehicle went perpendicularly to the trajectory in wrong direction at the first turn of the second lap. This situation was frequently observed during the tuning procedure. The reason of such behavior can be found at the plot with errors comparison (Figure 6.13b). The axes were zoomed into the end of rollout in order to present the errors behavior clearly. The lateral error based on the projection starts to deviate a lot from the original error definition obtained via the closest point interpolation (here named Kalman error). The problem is that the integrated target point is continuously moving, and its normal (to which the difference vector is projected) is rapidly rotating because of the steep turn. This leads to the incorrect lateral error, forcing PID controller to command a steering, which results into the driving in the opposite direction. In contrast, under the original error definition, difference is projected to the normal of a closest point which is remaining the same when the vehicle is moving perpendicularly to the target trajectory. The original lateral error definition accounts for these situations, thus it is much easier to obtain the stable controller.

Additionally, in the same plot of Figure 6.13b the undesirable behavior of the lateral error based on the angle can be observed. The discontinuity closer to the end of rollout appears when the vehicle is overtaking the target integrated point, which results into the change of sign of $\boldsymbol{vd}$ vector, discontinuous jump in the lateral error and unstable system. This situation can be omitted though, when the longitudinal controller is tuned to be more error-tolerant.

The results of the test experiment for the angle based lateral and the velocity based longitudinal errors are given on the right side of Figure 6.13. Using these errors definitions, a stable PID controller was obtained. Nevertheless, as it demonstrated on Figure 6.13d, velocity based longitudinal error results in a constant margin between the vehicle s_t and target $s_{\mathrm{tr},t+3}$ coordinates. The Kalman longitudinal error, i.e. a difference in s coordinate between a closets and target point, has a persistent period and does not cross the zero. In combination with angular lateral error this margin leads to the constant deviation from the target trajectory (Figure 6.13b). This margin can be minimized with a more aggressive longitudinal PID. However, aggressive velocity based PID controller tends to overtake the target position especially during the first lap, when the velocity is reduced. Such overtaking leads to the change in sign for the lateral error based on angle and unstable behavior. Even though we were not concerned about the performance so far, the resulting controller from Figure 6.13b is clearly not an appropriate choice of the benchmark. Depending on the margin between s_t and $s_{\mathrm{tr},t+3}$ the resulting vehicle trajectory can have an arbitrary form

The critical situations for proposed error definitions were identified and analyzed. None of the setups, presented above, achieved the performance of the original controller [35]. However, due to the limit of time, it was not tested if the system can be optimally controlled with original error definitions but without a post-processing with the Control Router. Such test permits to conclude if the main reason of instability are the new errors or the absence of a post-processing, If latter, the CR can be included to the control mechanism with a preliminary testing similar to Section 6.5.

7 Conclusion and Outlook

In the presented thesis we developed the method for scheduled PID tuning based on PILCO. The application of the developed method does not require the prior knowledge of a system dynamics and is not limited to SISO systems. The gain scheduling function is defined as a set of GPs parametrized by training targets. Under such definition, the output of PID controller becomes the product of two correlated multidimensional random variables. In order to integrate the function into PILCO iteration, the joint distribution between the extended PILCO state $\tilde{z}$ and the predicted gains was approximated. During the derivation, two cases were considered, the scheduled gains being dependent on and independent of the control errors. The independent case neglects the cross-covariance between the gains and errors, whereas for the dependent case second order Taylor Series approximation is used to compute the variance of a control signal $\boldsymbol{u}$. The cross-covariance term for $\boldsymbol{u}$ was derived only with respect to the scheduling variable, which can undermine the optimization results. The simple numeric test for the approximation quality using the precise distribution of a product of two correlated Gaussians was performed. Even though the results look reasonable, one cannot estimate the impact of made approximations on the algorithm performance based on the conducted numeric test. The reason is that the gains and errors are only approximated with a Gaussian during PILCO iteration.

For the testing purposes two systems were developed: Noisy Cart-Pole and Non-stationary Mass-Damper. The first system was used for the plain PID tuning to demonstrate the performance of the method developed by Doerr et al. [16] and to create a baseline for the further work. The resulting PID outperformed the LQR already after the second iteration of the algorithm. Then, the GPs trained using 9 seconds of system interaction precisely model the dynamics. Then proposed modification of this method was applied to the scheduled PID tuning which controls the Non-stationary Mass-Damper system. The gains of the PID were scheduled based on the position of a center of mass. After the introduction of an additional cost terms for the scheduled gains, the optimized controller outperforms the plain PID and LQR.

Since the main aim of a method is application to unknown nonlinear MIMO systems, it was exploited to tune the scheduled PID controller of autonomous RC car. Even though, the final results were not achieved due to the time constraints and complexity of the problem, the control mechanism of the autonomous RC car was described and analyzed. Moreover, a different approaches to the system dynamics modeling were proposed and evaluated. None of them resulted in an accurate model. The analysis of the obtained results and further investigation of the control scheme allowed to identify a three possible reasons of the failure. Furthermore, first approaches to the solution such as control mechanism simplification and errors redefinition were proposed. The simplified control mechanism with redefined errors which operates directly on the RC car dynamics model output was implemented and tested. However, a stable sufficiently optimal controller with the new error definitions was not obtained by the end of this master thesis project. The reasons of instability were described and analyzed.

Considering the potential advantages, i.e. applicability to unknown MIMO systems and data-efficiency inherited from PILCO, the theoretical basis of the method and its application to autonomous RC car worth further research and investigation. As a prospective theoretical research higher order approximations for a variance of a scheduled PID controller output $\boldsymbol{u}$ and cross-covariance for the entire extended state $\tilde{z}$ can be derived. Additionally, the evaluation of the dependency between the algorithm's performance and the quality of these approximation can be conducted.

The autonomous RC car being a complicated nonlinear system is exploring the limits of method applicability. Further work in the method application to the RC car should begin with a design of a stable controller with a performance comparable to the original control scheme in order to create a new benchmark. Potentially it can be achieved using the post-processing of control output similar to the one used by Wischnewski [35]. Nevertheless, before the insertion of Control Router to the simplified mechanism, additional validation experiment has to be conducted for the precise RC car model plus Control Router. Once a stable, manually-tuned controller which results in a trajectory similar to Figure 6.3b is obtained, the developed method can be applied. With a positive results one can proceed to the reimplementation of Observer in order to apply the method to the physical system. However, the important point is that the resulting dynamics should remain feasible for GP.

A Appendix

A.1 Linear transformation of Gaussian distribution

Given two Gaussian distributions, where one is a linear transformation of another:

$$\begin{aligned}\boldsymbol{X} &\sim \mathcal{N}(\boldsymbol{X}|\boldsymbol{\mu}_X, \boldsymbol{\Sigma}_X) \in \mathbb{R}^D,\\ \boldsymbol{Y} &= \boldsymbol{A}\boldsymbol{X} + \boldsymbol{b} = \mathcal{N}(\boldsymbol{Y}|\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y) \in \mathbb{R}^F,\end{aligned}$$

with $\boldsymbol{A} \in \mathbb{R}^{F\times D}$ and $\boldsymbol{b} \in \mathbb{R}^F$. Joint distribution $p(\boldsymbol{X}, \boldsymbol{Y})$ is given by:

$$\begin{bmatrix}\boldsymbol{X}\\ \boldsymbol{Y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_X\\ \boldsymbol{\mu}_Y\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Sigma}_X & \boldsymbol{\Sigma}_X\boldsymbol{C}\\ \boldsymbol{C}^T\boldsymbol{\Sigma}_X^T & \boldsymbol{\Sigma}_Y\end{bmatrix}\right), \tag{A.1}$$

where:

$$\begin{aligned}\boldsymbol{\mu}_Y &= \boldsymbol{A}\boldsymbol{\mu}_X + \boldsymbol{b},\\ \boldsymbol{\Sigma}_Y &= \boldsymbol{A}\boldsymbol{\Sigma}_X\boldsymbol{A}^T,\\ \boldsymbol{C} &= \boldsymbol{A}^T.\end{aligned}$$

A.2 Product of two independent Gaussian distributions

Given two Gaussian distributions:

$$\begin{aligned}\boldsymbol{X} &= \mathcal{N}(\boldsymbol{\mu}_X, \boldsymbol{\Sigma}_X) \in \mathbb{R}^D,\\ \boldsymbol{Y} &= \mathcal{N}(\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y) \in \mathbb{R}^D.\end{aligned}$$

Their product results into unnormalized Gaussian:

$$\boldsymbol{X}\boldsymbol{Y} = c\mathcal{N}(\boldsymbol{\mu}_Z, \boldsymbol{\Sigma}_Z). \tag{A.2}$$

In the above equation:

$$\begin{aligned}c_c &= \mathcal{N}(\boldsymbol{\mu}_X|\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_X + \boldsymbol{\Sigma}_Y),\\ \boldsymbol{\mu}_Z &= (\boldsymbol{\Sigma}_X^{-1} + \boldsymbol{\Sigma}_Y^{-1})^{-1}(\boldsymbol{\Sigma}_X^{-1}\boldsymbol{\mu}_X + \boldsymbol{\Sigma}_Y^{-1}\boldsymbol{\mu}_Y),\\ \boldsymbol{\Sigma}_Z &= (\boldsymbol{\Sigma}_X^{-1} + \boldsymbol{\Sigma}_Y^{-1})^{-1}.\end{aligned}$$

Bibliography

[1] K. J. Åström, T. Hägglund. “Advanced PID control.” In: *The Instrumentation, Systems, and Automation Society*. Citeseer. 2006 (cit. on pp. 13, 22, 24).

[2] K. J. Åström, T. Hägglund. *PID controllers: theory, design, and tuning*. Vol. 2. Instrument society of America Research Triangle Park, NC, 1995 (cit. on pp. 13, 16).

[3] K. J. Åström, T. Hägglund. “Revisiting the Ziegler–Nichols step response method for PID control.” In: *Journal of process control* 14.6 (2004), pp. 635–650 (cit. on p. 21).

[4] K. J. Åström, R. M. Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010 (cit. on pp. 13, 15, 16, 18, 21, 23, 24, 30).

[5] K. J. Åström, B. Wittenmark. “On self tuning regulators.” In: *Automatica* 9.2 (1973), pp. 185–199 (cit. on p. 31).

[6] F. D. Bianchi, R. J. Mantz, C. F. Christiansen. “Gain scheduling control of variable-speed wind energy conversion systems using quasi-LPV models.” In: *Control Engineering Practice* 13.2 (2005), pp. 247–255 (cit. on p. 30).

[7] S. A. Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013 (cit. on p. 30).

[8] W.-D. Chang. “A multi-crossover genetic approach to multivariable PID controllers tuning.” In: *Expert Systems with Applications* 33.3 (2007), pp. 620–626 (cit. on p. 24).

[9] W.-D. Chang, R.-C. Hwang, J.-G. Hsieh. “A multivariable on-line adaptive PID controller using auto-tuning neurons.” In: *Engineering Applications of Artificial Intelligence* 16.1 (2003), pp. 57–63 (cit. on pp. 13, 18, 31).

[10] J. Chen, T.-C. Huang. “Applying neural networks to on-line updated PID controllers for nonlinear process control.” In: *Journal of process control* 14.2 (2004), pp. 211–230 (cit. on p. 31).

[11] P. Cominos, N. Munro. “PID controllers: recent tuning methods and design to specification.” In: *IEE Proceedings-Control Theory and Applications* 149.1 (2002), pp. 46–53 (cit. on pp. 13, 22, 24).

[12] L. Csató, M. Opper. “Sparse on-line Gaussian processes.” In: *Neural computation* 14.3 (2002), pp. 641–668 (cit. on p. 19).

[13] G. Cui, X. Yu, S. Iommelli, L. Kong. “Exact Distribution for the Product of Two Correlated Gaussian Random Variables.” In: *IEEE signal processing letters* 23.11 (2016), pp. 1662–1666 (cit. on p. 40).

[14] M. P. Deisenroth, D. Fox, C. E. Rasmussen. “Gaussian processes for data-efficient learning in robotics and control.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (2015), pp. 408–423 (cit. on pp. 13, 21, 25, 27, 28, 34, 38, 39).

[15] M. P. Deisenroth, C. E. Rasmussen, D. Fox. “Learning to control a low-cost manipulator using data-efficient reinforcement learning.” In: (2011) (cit. on p. 29).

[16] A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, S. Trimpe. “Model-based policy search for automatic tuning of multivariate PID controllers.” In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE. 2017, pp. 5295–5301 (cit. on pp. 13, 21, 28, 30, 43, 45, 69).

[17] J. Dong, C. B. Brosilow. “Design of robust multivariable PID controllers via IMC.” In: *American Control Conference, 1997. Proceedings of the 1997*. Vol. 5. IEEE. 1997, pp. 3380–3384 (cit. on pp. 23, 24).

[18] Z.-L. Gaing. “A particle swarm optimization approach for optimum design of PID controller in AVR system.” In: *IEEE transactions on energy conversion* 19.2 (2004), pp. 384–391 (cit. on p. 24).

[19] C. C. Hang, K. J. Åström, W. K. Ho. “Refinements of the Ziegler–Nichols tuning formula.” In: *IEE Proceedings D (Control Theory and Applications)*. Vol. 138. 2. IET. 1991, pp. 111–118 (cit. on p. 21).

[20] M. W. Iruthayarajan, S. Baskar. “Evolutionary algorithms based design of multivariable PID controller.” In: *Expert Systems with applications* 36.5 (2009), pp. 9159–9167 (cit. on pp. 13, 24).

[21] M. A. Johnson, M. H. Moradi. *PID control: new identification and design methods*. Springer Science & Business Media, 2006 (cit. on pp. 13, 23, 24).

[22] E. B. Lee, L. Markus. *Foundations of optimal control theory*. Tech. rep. MINNESOTA UNIV MINNEAPOLIS CENTER FOR CONTROL SCIENCES, 1967 (cit. on p. 22).

[23] W. S. Levine. *The control handbook*. CRC press, 1996 (cit. on pp. 16, 22).

[24] K. S. Narendra, K. Parthasarathy. “Identification and control of dynamical systems using neural networks.” In: *IEEE Transactions on neural networks* 1.1 (1990), pp. 4–27 (cit. on p. 31).

[25] A. Packard. “Gain scheduling via linear fractional transformations.” In: *Systems & control letters* 22.2 (1994), pp. 79–92 (cit. on p. 18).

[26] C. E. Rasmussen. “Gaussian processes in machine learning.” In: *Advanced lectures on machine learning*. Springer, 2004, pp. 63–71 (cit. on p. 19).

[27] D. E. Rivera, M. Morari, S. Skogestad. “Internal model control: PID controller design.” In: *Industrial & engineering chemistry process design and development* 25.1 (1986), pp. 252–265 (cit. on p. 23).

[28] G. Rudolph. “Convergence of evolutionary algorithms in general search spaces.” In: *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*. IEEE. 1996, pp. 50–54 (cit. on p. 25).

[29] W. J. Rugh, J. S. Shamma. “Research on gain scheduling.” In: *Automatica* 36.10 (2000), pp. 1401–1425 (cit. on pp. 13, 18, 19, 30).

[30] E. Snelson, Z. Ghahramani. “Sparse Gaussian processes using pseudo-inputs.” In: *Advances in neural information processing systems*. 2006, pp. 1257–1264 (cit. on pp. 19, 20).

[31] C. J. Stone. “Additive regression and other nonparametric models.” In: *The annals of Statistics* (1985), pp. 689–705 (cit. on p. 19).

[32] R. S. Sutton, A. G. Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998 (cit. on p. 60).

[33] A. Visioli. “Fuzzy logic based set-point weight tuning of PID controllers.” In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: systems and humans* 29.6 (1999), pp. 587–592 (cit. on p. 30).

[34] Q.-G. Wang, B. Zou, T.-H. Lee, Q. Bi. “Auto-tuning of multivariable PID controllers from decentralized relay feedback.” In: *Automatica* 33.3 (1997), pp. 319–330 (cit. on p. 13).

[35] A. Wischnewski. “Control of highly automated and autonomous vehicles in critical driving situations.” Master Thesis. 2017 (cit. on pp. 51, 52, 55, 56, 67, 70).

[36] Z.-W. Woo, H.-Y. Chung, J.-J. Lin. “A PID type fuzzy controller with self-tuning scaling factors.” In: *Fuzzy sets and systems* 115.2 (2000), pp. 321–326 (cit. on p. 30).

[37] J. Yen, R. Langari. *Fuzzy logic: intelligence, control, and information*. Vol. 1. Prentice Hall Upper Saddle River, NJ, 1999 (cit. on p. 30).

[38] Z.-Y. Zhao, M. Tomizuka, S. Isaka. “Fuzzy gain scheduling of PID controllers.” In: *IEEE transactions on systems, man, and cybernetics* 23.5 (1993), pp. 1392–1398 (cit. on pp. 13, 30).

[39] F. Zheng, Q.-G. Wang, T. H. Lee. “On the design of multivariable PID controllers via LMI approach.” In: *Automatica* 38.3 (2002), pp. 517–526 (cit. on pp. 13, 24).

All links were last followed on March 17, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature