Self-Learning Control of a Mechatronic System Using Gaussian Process

Antal Péter, BSc 7. félév

Konzulensek:

Dr. Péni Tamás
tudományos főmunkatárs
Számítástechnikai és Automatizálási Kutatóintézet

Dr. Tóth Roland
tudományos főmunkatárs
Számítástechnikai és Automatizálási Kutatóintézet

Dr. Sykora Henrik
tudományos segédmunkatárs
Műszaki Mechanikai Tanszék

Budapest, 2020 novemberén

Table of Contents

List of Figures

Abstract

Nowadays due to the increased computational power of modern computer architectures machine learning algorithms are becoming widely used in several fields of science and technology. Most of these learning algorithms are adaptive, i.e. they can adjust their behaviour based on the data collected during operation. Machine learning has been shown to give advantages in cases such as email filtering, computer vision or autonomous vehicle control [1], but also provides alternative methods for the mechanical design of beam structures [2], to mention a few application areas.

Due to tractability and solid theoretical background, linear models are widely used in the field of control engineering. However, most physical and mechanical systems are inherently non-linear, therefore a linear structure is not sufficient to describe their dynamical behaviour over the entire operating domain. Typical nonlinearities in mechatronic systems are non-linear friction effects, magnetic saturation, etc.

In this work, we propose to use the fusion of an augmented model based on Gaussian processes (GPs), and model predictive control (MPC), resulting in a self-learning MPC paradigm. This state-of-the-art approach uses collected data for compensating model uncertainties, complex non-linear effects and machine specific properties. It provides adaptability, safety guarantees, and has computationally efficient implementations (e.g. constrained non-linear multi-variable function minimizer) [3]. Two different self-learning methods are demonstrated. Firstly, an explicit MPC approach based on reinforcement learning. This method has the advantage of fewer run-time calculations due to off-line optimization, which makes it feasible with high sampling frequencies. However, not all necessary safety guarantees are provided during operation. The second approach, on-line GPMPC is able to calculate real-time optimal control inputs while satisfying all constraints. Although it has higher computational cost, the result is a self-learning advanced control method with safety guarantees.

The capabilities of the provided control methods are demonstrated on a self-designed cart-pole system built from of an inkjet printer mechanism [4]. Based on measurements, the actual system is significantly different from the corresponding mechanical model, which implies the use of data based, self-learning control methods.

Kivonat

Napjainkban a modern számítógép-architektúrák megnövekedett számítási teljesítménye lehetővé teszi a gépi tanulásra épülő eljárások egyre szélesebb körű alkalmazását. Ezen algoritmusok többsége adaptív, azaz működés közben, a gyűjtött adatok alapján folyamatosan fejlődik, tanul. Gépi tanulást előszeretettel alkalmaznak olyan problémákra, mint például elektronikus levelek szűrése, gépi látás, vagy autonóm járművek irányítása [1], de megjelenik rúdszerkezetek szilárdságtani tervezésében is [2].

Az irányítástechnikában elterjedt a lineáris modellek használata azok kezelhetősége és szilárd elméleti háttere miatt. A valóságban azonban számos fizikai, mechanikai rendszerben tapasztalunk nagy mértékű nemlinearitást, aminek következtében a lineáris modell nem képes a dinamika leírására a teljes működési tartományon. Mechanikai rendszerekre jellemző nemlinearitás a száraz súrlódás, a mágneses telítődés stb.

A dolgozat során Gauss-folyamattal kiegészített nominális modell és modell prediktív irányítás (MPC) egyesítésével létrehozott öntanuló irányítási módszert mutatunk be. Az így alkotott algoritmus mérési adatokat felhasználva képes kompenzálni a matematikai modell bizonytalanságait, komplex nemlineáris jelenségek hatásait, illetve gépspecifikus tulajdonságokat. Adaptív működéssel egyidejűleg képes biztonsági garanciákat biztosítani, flexibilis, és hatékonyan implementálható [3]. A dolgozatban két különböző öntanuló eljárás van bemutatva, elsőként egy megerősítéses tanuláson alapuló explicit MPC. A módszer előnye, hogy az off-line optimalizációból kifolyólag kisebb a valós idejű számítási igénye, így jól alkamazható magas mintavételezési frekvencia mellett. Hátránya azonban, hogy működés közben nincs minden biztonsági előírásra garancia biztosítva. A második öntanuló eljárás on-line optimalizációra épülő Gauss-folyamaton alapuló MPC, ami képes valós idejű optimális irányításra a szükséges fizikai kényszerek (vagyis biztonsági garanciák) betartásával egyidejűleg.

A tárgyalt irányítási stratégiák egy saját tervezésű, tintasugaras nyomtató mechanizmusából épített, vízszintesen megvezetett inverz inga egyensúlyozásán keresztül vannak bemutatva [4]. Korábbi mérési adatok alapján a megvalósított rendszer jelentősen eltér a levezetett mechanikai modelltől, így indokolt az adatalapú, öntanuló irányítás alkalmazása.

1 Introduction

The aim of this work is to demonstrate how joining machine learning and model predictive control results in a high performance self-learning control method through the example of a self-designed cart-pole dynamic system. Although a cart-pole system can be controlled using classical approaches (e.g. PID control [5]), with significant non-linearities and physical limitations such control design faces severe limitations [4]. Hence, we propose to use Gaussian Process Machine Learning (GPML) based advanced algorithms for learning those effects from data and compensating them adaptively at the same time.

Machine learning is a joint field of artificial intelligence and statistics, mainly concerned with the design of adaptive algorithms, which develop through learning from the collected data during operation. In a subset of these methods a statistical model results, which represents the data efficiently. Nowadays, machine learning appears at many fields of science and technology, because the available computational power makes it possible to process big amounts of data and carry out complicated calculations in a relatively short time. Early applications are such tasks as text and video recognition or e-mail filtering, where occasional errors can be tolerated and the uncertainty is not really significant while the algorithm works. However, when it comes to the control design of a mechatronic system, the accuracy and the measure of uncertainty are major factors as guarantees are required for safety and efficiency [6].

As a branch of machine learning, supervised learning is concerned with labelled input-output data pairs. This labels given by a teacher express what are good or wrong outputs for certain inputs. The goal of supervised learning is to make predictions after the analysis of training input-output pairs and generating a statistical model. These predictions are function estimations evaluated at new inputs, for which a sufficiently designed structure is essential to achieve an efficient algorithm. Artificial intelligence methods are often compared to human behaviour and learning. In that case, an example for supervised learning can be a child learning animal names. In storybooks there are many animal drawings, which are labels for the learning of children. If the drawing of a dog is not precise enough, it can happen, that the child points at the fox in the zoo and notes how shaggy tail that dog has. In this case, the mother tells him that it is a fox, thus the label is corrected and a new label is added to the collection at the same time.

Another type of ML is reinforcement learning, where only inputs are available without labelled outputs, and the reward of the actions coming from the inputs. The goal of these algorithms is to maximize the cumulative reward via iterations of trials and optimization. Although reinforcement learning is becoming more and more popular in the fields of autonomous vehicle and robotics control, safety concerns are often hard to consider within the framework of these algorithms. In terms of human analogy, reinforcement learning is somewhat similar to school education. If a student does a task correctly, the reward will be the maximal grade. In order to perform best, a studying strategy needs to be created and optimized to maximize performance and achieve maximal grades efficiently.

In the field of machine learning there are parametric and non-parametric approaches. Common examples for parametric models are neural networks, perceptron algorithms and logistic regression. These methods consist of a fixed form of mapping function along with fixed number of parameters, which are estimated during the learning process. In contrast, for non-parametric models, the number of parameters increases with the amount of collected data. Often such models are more flexible, which makes them able to capture the data represented dynamics, but their generalization capability is less.

Within the class of non-parametric modelling methods, empirical Bayesian modelling leads to data-driven generative models optimized to fit data under explicit assumptions of measurement uncertainty. Gaussian processes (GPs) are highlighted as an important class of Bayesian non-parametric models. A GP is a distribution over functions, from which an approximation can be formulated regarding the uncertainty of the prediction, often considered in terms of the mean. This is a great advantage over parametric approaches, where such uncertainty can be only formulated on the parameter level, and not in a functional sense [7, 8].

Nowadays a popular and widely researched field of interdisciplinary science is human balancing. Mathematicians, engineers and neuroscientists work together to understand the ways of sensing a part of the environment, processing the collected data and actuating a certain part of the body, aiming to help the elderly and people with balance disorders [9, 10]. Two common examples of the research area are human postural balance and stick balancing. An inverted pendulum on a horizontally driven cart (cart-pole dynamic system) is a simplified but accurate demonstration of the latter. For the modelling of human balancing, the classical PID controller is a widely used approach, because it can be matched to the human sensory functions, and has solid theoretical background [11]. However, if we try to balance a stick on our fingers, it will turn out that after some trials and learning, our skills get better, we can also balance a shorter stick or the same stick for a longer time. Our brain adapts to the senses and muscles of the body, learns how to react for a given state of the inverted pendulum.

Similarly to the human body, a cart-pole mechanism built from an inkjet printer has complex dynamics, significant nonlinearities, and sensor uncertainties [4]. The inner friction of the motor, the friction of the cart on the slider, and the effect of the belt drive are all very challenging modelling tasks. Instead of that data is collected from measurements during operation, and the less complex nominal model is augmented based on Gaussian Process Machine Learning. The augmented model is then fused with model predictive control, resulting in an efficient, data based advanced control strategy. This state-of-the-art approach is able to take advantage of the collected data by active learning, and provide safety guarantees by on-line optimization and adaptive control.

In order to investigate self-learning control of the mechatronic system using Gaussian process, the following topics will be presented. In **Section 2**, the cart-pole dynamic system is described based on both its standard model, and measurements carried out on

the implemented mechanism. The equations of motion are derived for the mechanical and electrical part separately, which are then joined resulting in a non-linear, continuous-time state space representation.

In **Section 3**, a theoretical overview is provided about Gaussian Process Regression, as a general function approximation method for a given data set. After the basic principles, the mathematical formulae are derived, which are then used for the machine learning part of control. The objective of the section is to formulate a viewpoint about the Bayesian approach as an efficient tool for regression and function approximation, not to fully cover the mathematical background of Gaussian processes and their applications.

In **Section 4**, probabilistic reinforcement learning control is discussed, with great emphasis on the 'Probabilistic Inference for Learning Control (PILCO)' framework [12], which is a fully developed, efficient tool for learning a probabilistic dynamics model and control policy fully from data, without using the nominal system model (except for simulating rollouts with a given control policy). The basic principles and algorithms are discussed in this section, based on which simulations can be carried out.

Section 5 presents linear-quadratic (LQ) control, as a basis for comparison with self-learning methods. It is a classical approach for the design of linear feedback control, with off-line optimization for a linear system model, and without considering any constraints. This method is chosen for comparison, because it uses similar cost function as MPC, and the calculated linear feedback gain can be used as a part of the MPC control input.

In **Section 6**, Gaussian Process MPC is discussed, for which a simulation framework is one of the main objectives of this work. Basic principles and parameters of the method are defined for deterministic MPC, which is then augmented with GPML to formulate a self-learning control method. This approach is capable of handling constraints, which is a great advantage over the other methods, but the on-line learning has higher computational cost.

Section 7 contains the results of simulations with all of the presented control methods, and the comparison of them based on control performance and computational time. The section includes the advantages and disadvantages of every strategy, backed up by quantitative results.

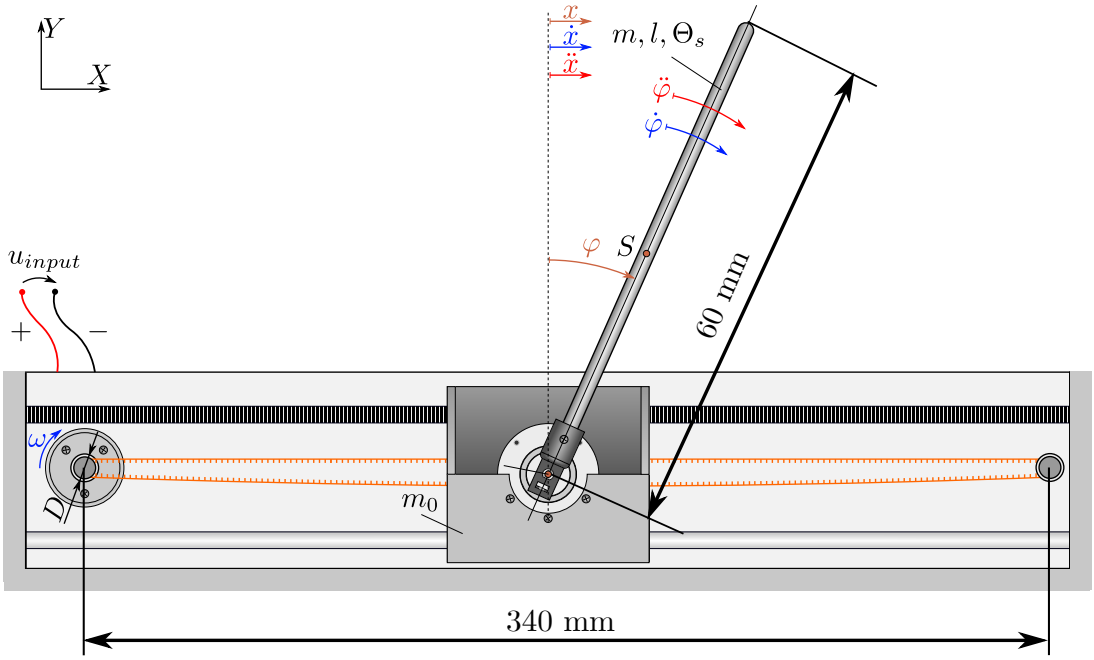


Figure 2.1: Assembly of the implemented cart-pole system

2 Mechatronic System and Control Task

First we introduce the mechatronic system and the control task. The main parts of the mechanism are the actuating DC servo motor, the carriage (cart) with the pendulum (pole), and the frame on which the carriage slides. The motor and the cart are connected by a plastic belt. The position of the carriage is measured by a linear incremental optical encoder with resolution of 0.0356 mm, and the angle of the pendulum with a rotary incremental optical encoder with resolution of 600 ppr (pulse per revolution), which is equal to approximately 0.15°. Except for the pendulum and the sensor of the angle, all mentioned parts are from the original Epson Stylus DX38503 inkjet printer. The parameters of the system can be found in Table 2.1.

2.1 Dynamics of the System

The equations of motion of the two degree-of-freedom cart-pole system can be derived from the Lagrangian equations using the notations from Figure 2.1, and then augmented with the electromechanical equations of the DC servo motor.

2.1.1 Mechanical Subsystem

The mechanical subsystem consists of a cart with mass m_0, actuated by a force Q provided by the DC motor through the belt drive, and a homogeneous stick (pendulum) with mass m and length l. The numerical values of the parameters can be found in Table 2.1. The

position of the cart is denoted by $q_1 = x$, and the angle of the pendulum is by $q_2 = \varphi$. Let $\mathbf{q} = [q_1\ q_2]^\top$ be the vector of general coordinates. The kinetic energy results from the velocity of the cart and the angular velocity of the pendulum, as

$$T = \frac{1}{2}m_0\dot{x}^2 + \frac{1}{2}m\left(\dot{x}^2 + l\dot{x}\dot{\varphi}\cos\varphi + \left(\frac{l}{2}\right)^2\dot{\varphi}^2\right) + \frac{1}{2}\Theta_s\dot{\varphi}^2, \tag{2.1}$$

where Θ_s is the moment of inertia of the pendulum. The potential energy is expressed as

$$U = mg\frac{l}{2}\cos\varphi. \tag{2.2}$$

Using (2.1), the motion of the cart is characterized by the derivatives

$$\frac{\partial T}{\partial \dot{x}} = m_0\dot{x} + m\dot{x} + m\frac{l}{2}\dot{\varphi}\cos\varphi, \tag{2.3}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{x}} = m_0\ddot{x} + m\ddot{x} + m\frac{l}{2}\ddot{\varphi}\cos\varphi - m\frac{l}{2}\dot{\varphi}^2\sin\varphi, \tag{2.4}$$

$$\frac{\partial T}{\partial x} = 0. \tag{2.5}$$

Similarly, the derivatives of (2.1) considering the motion of the pendulum,

$$\frac{\partial T}{\partial \dot{\varphi}} = m\frac{l}{2}\dot{x}\cos\varphi + \dot{\varphi}\left(m\left(\frac{l}{2}\right)^2 + \Theta_s\right), \tag{2.6}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{\varphi}} = m\frac{l}{2}\ddot{x}\cos\varphi - m\frac{l}{2}\dot{x}\dot{\varphi}\sin\varphi + \ddot{\varphi}\left(m\left(\frac{l}{2}\right)^2 + \Theta_s\right), \tag{2.7}$$

$$\frac{\partial T}{\partial \varphi} = -m\frac{l}{2}\dot{x}\dot{\varphi}\sin\varphi. \tag{2.8}$$

To complete the characterization, the derivatives of the potential function from (2.2):

$$\frac{\partial U}{\partial x} = 0, \tag{2.9}$$

$$\frac{\partial U}{\partial \varphi} = -mg\frac{l}{2}\sin\varphi. \tag{2.10}$$

The forces acting on the system can be calculated from the power, as

$$P = Q\dot{x} = Q_1\dot{\varphi} + Q_2\dot{x}, \quad \text{from which} \tag{2.11}$$

$$Q_1 = 0, \quad Q_2 = Q, \tag{2.12}$$

which is quite trivial, given the only actuation is the force Q acting on the cart. The general form of the governing equation is

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial T}{\partial \dot{q}_i} - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} = Q_i, \quad i = 1, 2, \tag{2.13}$$

from which the equations of motion, using $\Theta_s = \frac{1}{12}ml^2$ are

$$q_1 = \varphi: \ \frac{1}{2}ml\ddot{x}\cos\varphi + \frac{1}{3}ml^2\ddot{\varphi} - \frac{1}{2}mgl\sin\varphi = 0, \tag{2.14}$$

$$q_2 = x: \ (m + m_0)\ddot{x} + \frac{1}{2}ml\ddot{\varphi}\cos\varphi - \frac{1}{2}ml\dot{\varphi}^2\sin\varphi = Q. \tag{2.15}$$

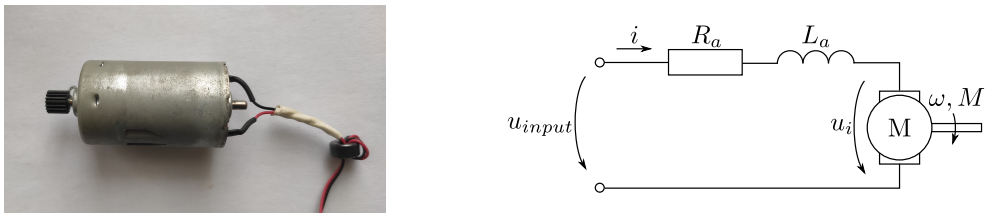


Figure 2.2: Photo of the DC motor (left), and electric circle model (right)

2.1.2 DC Motor

To actuate the mechanism a brushed DC motor is used, which can be modelled as a series R-L (resistance-inductance) circle. The mechanical speed and torque are characterized by the speed constant k_e and torque constant k_m, the numerical values of the parameters are in Table 2.1. The electrical equation is

$$u_{input} = L_a \frac{\mathrm{d}i}{\mathrm{d}t} + R_a i + u_i, \tag{2.16}$$

where u_{input} is the input (control) voltage, i is the armature current, and u_i is the induced voltage. The electrical and mechanical subsystems are coupled via two equations:

$$u_i = k_e\, \omega = \frac{k_e}{R}\dot{x}, \tag{2.17}$$

$$M = k_m i, \tag{2.18}$$

where ω is the angular velocity of the motor shaft, M is the torque provided by the motor and R is the radius of the motor shaft. Based on the torque M, the actuating force Q acting on the cart can be expressed as

$$Q = \frac{M}{R}. \tag{2.19}$$

However, the DC motor has much faster dynamics, than the mechanical subsystem. As a qualitative measure, the electric time constant of the motor is calculated as

$$T_e = \frac{L_a}{R_a} \approx 0.69 \text{ ms}. \tag{2.20}$$

Considering this, the inductance can be neglected, thus an algebraic equation describes the electric subsystem,

$$u_{input} = R_a i + u_i. \tag{2.21}$$

2.1.3 Modelling of Friction

Besides the standard model of a cart-pole system, the implemented mechanism contains additional dynamics, based on measurements. Firstly, the sensing and actuating contains sampling and quantization. The microcontroller is a digital device with certain sampling time, and the sensors measure discrete values, therefore contain threshold, which effects

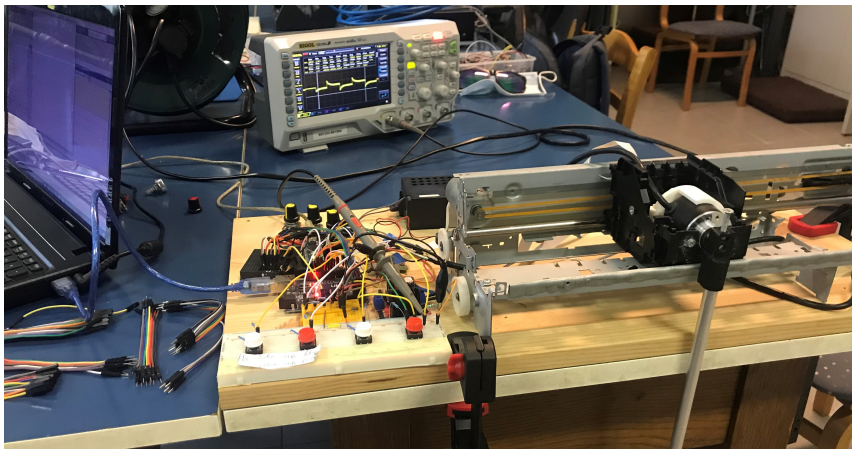


Figure 2.3: Measurement of the friction

can have a major impact on the dynamics (e.g. cause micro-chaos [10]). The elasticity of the plastic belt can also affect the dynamics, but the self-learning controllers are expected to suppress these two factors. Secondly, significant friction is observed between the cart and the frame of the structure, which can be quantified and built in the model after making measurements.

In the field of mechanical engineering, friction is still a widely researched topic today, because it often has major dependence on several factors (e.g. roughness of the contacting surfaces, speed, lubrication). Although self-slearning algorithms aim to balance model uncertainties, according to several measurements the friction in the implemented mechanism is too major to be considered negligible.

Friction is an additional force acting on the cart which needs to be balanced by the control force Q. It means that when the speed is constant (acceleration is zero), the friction force is equal to Q and this way proportional to the armature current, which can be measured directly. From (2.18) and (2.19) the friction force F_c can be expressed as

$$F_c = \frac{k_m}{R} i, \tag{2.22}$$

if $\ddot{x} = 0$.

The measurement of the current at constant velocity was carried out by measuring the voltage of an external, $R_{ext} = 1\ \Omega$ resistance in series connection with the motor, using an oscilloscope. The velocity was calculated as the discrete derivative of the position signal. The setup of the measurement is shown on Figure 2.3, and the results on Figure 2.4[1].

The form of the fitted function is

$$F_c = a \tanh(b\dot{x}), \tag{2.23}$$

with parameters $a = 3.43$ N and $b = 53.6$ s/m. Although there are only eight measured points, the correlation coefficient is $R^2 = 0.9959$, therefore the function properly describes

[1]Only one direction of the speed was measured, the negative values are their reflection in the origin.

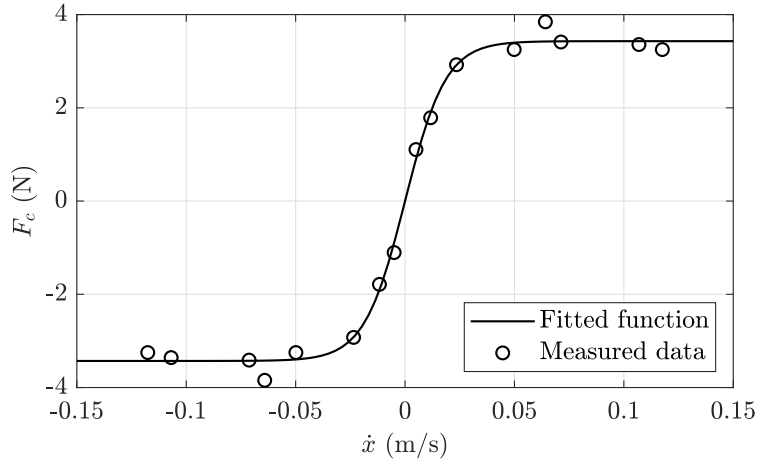


Figure 2.4: Friction force as a function of cart velocity

the measured connection. In order to include the friction in the mathematical model of the system, we need to complete the right hand side of equation (2.15) with the term $-F_c$.

From the measured friction force, the friction coefficient can be calculated as

$$\mu_c = \frac{F_c}{(m+m_0)g} \approx 2.87, \tag{2.24}$$

if $\dot{x} \to \infty$ (described by Coulomb friction). This is a large number considering the usual friction coefficients between engineering materials. However, this friction force contains the inner friction of the DC motor, the force needed to drive the belt, and the hitch of the slider on the cart.

2.1.4 State Space Representation

After considering all significant components of the dynamics, a first order, four dimensional state space representation in the form $\dot{\mathbf{x}} = f(\mathbf{x}, u)$ can be arranged. From (2.14), (2.15) with the additional friction term, and (2.21), the equation is

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\varphi\\ \dot{\varphi}\\ x\\ \dot{x}\end{bmatrix} = \begin{bmatrix}\dot{\varphi}\\ \dfrac{(m+m_0)g\sin\varphi - \left(\frac{k_m}{RR_a}\left(u - \frac{k_e}{R}\dot{x}\right) - F_c\right)\cos\varphi - m\frac{l}{2}\dot{\varphi}^2\sin\varphi\cos\varphi}{\frac{2}{3}l(m+m_0) - m\frac{l}{2}\cos^2\varphi}\\ \dot{x}\\ \dfrac{\frac{k_m}{RR_a}\left(u - \frac{k_e}{R}\dot{x}\right) - F_c - \frac{3}{4}mg\sin\varphi\cos\varphi + m\frac{l}{2}\dot{\varphi}^2\sin\varphi}{m+m_0-\frac{3}{4}m\cos^2\varphi}\end{bmatrix}, \tag{2.25}$$

with state vector $\mathbf{x}$ and control input $u = u_{input}$.

2.2 Control Task

The control task is to balance the pendulum at the upper equilibrium position, while the cart is at the center of the horizontal guide, and both the velocity of the cart and the angular velocity of the pendulum are zero. Given that the angle and the position are both measured from this state, the target state is

$$\begin{bmatrix} \varphi \\ \dot{\varphi} \\ x \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{2.26}$$

During operation, the cart has to be inside the 340 mm long range as shown on Figure 2.1, and the voltage of the DC motor between the allowed minimum and maximum values, ±40 V. The swing-up of the pendulum is not required in this case, because at the beginning of the control procedure the pendulum is near the upper equilibrium point ($|\varphi| \leq 15°$), and the cart is at the center of the frame ($x = 0$).

Table 2.1: Physical parameters of the implemented cart-pole mechanism

Mass of cart	m_0	0.07 kg
Mass of pendulum	m	0.052 kg
Length of pendulum	l	0.6 m
Motor speed constant	k_e	0.0538 Vs/m
Motor torque constant	k_m	0.0417 Nm/A
Motor shaft radius	R	0.0035 m
Motor terminal resistance	R_a	29.1 Ω
Motor terminal inductance	L_a	20.1 mH
Motor maximal voltage	u_{max}	40 V
Friction coefficient	μ_c	2.87

3 Gaussian Process Regression

In order to apply a self-learning controller using machine learning algorithms, function approximation is necessary, for which regression is a widely used approach. Regression is a statistical method for estimating the function $f : \mathbb{R}^D \to \mathbb{R}$ describing the relationship between the input variables $\mathbf{x}_i$ and the observations y_i, where

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad i = 1 \dots N, \tag{3.1}$$

and ε_i is a noise term. A common application for regression is describing the transitions in a dynamic system. Due to the finite number of observations (measurements), and the noise term, the function f can be identified only with uncertainty. As mentioned in Section 1, the Gaussian Process Regression is a non-parametric method, which is flexible, robust, and gives an approximation for the measure of the uncertainty.

3.1 Definition and Main Principles

The intuitive idea behind using Gaussian processes is based on a distribution over functions. The modelling concept is that at each value of x (which is now a scalar for easier representation) we see the possible function value $f(x)$ in our model coming from a Gaussian distribution: $f(x) \sim \mathcal{N}(0, \sigma^2(x))$. After making measurements, we update the prior distribution based on the observed samples to reach a distribution over functions, which is based on the actual data set.

Definition 1 (Gaussian process). A Gaussian process (GP) is a collection of random variables, any finite number of which have a consistent joint Gaussian distribution [7].

For the GP regression model we have a training data set $\mathcal{D}_N := \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ from observations. The relationship between the inputs and outputs are described according to (3.1), with the noise term ε_i, which is assumed to be independently and identically distributed (i.i.d.) for each observation i, namely $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ for each $i = 1 \dots N$, and f being a Gaussian process. As the Gaussian distribution is fully specified by its mean and covariance, the GP is completely characterized by its mean *function* ($m(\mathbf{x})$), and covariance *function* ($K(\mathbf{x}, \tilde{\mathbf{x}})$, also called *kernel*), such as

$$m(\mathbf{x}) = \mathbb{E}\{f(\mathbf{x})\}, \tag{3.2}$$

$$K(\mathbf{x}, \tilde{\mathbf{x}}) = \mathbb{E}\{(f(\mathbf{x}) - m(\mathbf{x}))(f(\tilde{\mathbf{x}}) - m(\tilde{\mathbf{x}}))\}, \tag{3.3}$$

shortened as $f \sim \mathcal{GP}(m, K)$.

3.2 Bayesian Inference

In order to formulate a posterior distribution on the function f, we need to employ a Bayesian inference technique. This method consists of three main steps: first, the prior

distribution on f should be specified. Second, measurements should be made to obtain observations, and third, the posterior distribution should be calculated via conditioning based on the observed data.

3.2.1 Prior

When creating the GP model, the mean and covariance function of the prior distribution should be specified. The average can always be subtracted from the observations, therefore it is common to choose the mean function to zero. However, for the choice of the kernel there are many options, and it is an important step, because it describes the properties (smoothness, periodicity, etc.) of the GP model and contains the so called *hyper-parameters*, which are tuned during the training process. Without claiming completeness, some examples for the kernels are Matérn class, periodic, dot product, and impulse response [7]. In this work, the most common choice, the infinitely differentiable *squared exponential* covariance function will be used with automatic relevance determination:

$$K_{SE}(\mathbf{x}, \tilde{\mathbf{x}}) = \sigma_f^2 \exp\left(-\frac{1}{2}(\mathbf{x} - \tilde{\mathbf{x}})^\top \mathbf{\Lambda}^{-1}(\mathbf{x} - \tilde{\mathbf{x}})\right), \quad \mathbf{x}, \tilde{\mathbf{x}} \in \mathbb{R}^D, \tag{3.4}$$

where $\mathbf{\Lambda} = \text{diag}([\lambda_1^2, \ldots, \lambda_D^2])$ is the diagonal matrix of squared characteristic length-scales λ_i, $i = 1 \ldots D$, and σ_f^2 is the signal variance of f. The length-scales λ_i and the signal variance σ_f make up the array of hyper-parameters $\boldsymbol{\theta}$, which specifies the kernel function, and plays an important role as determining the properties of the distribution. From now on, the notation $K(\mathbf{x}, \tilde{\mathbf{x}})$ will be used for the squared exponential covariance function with automatic relevant determination.

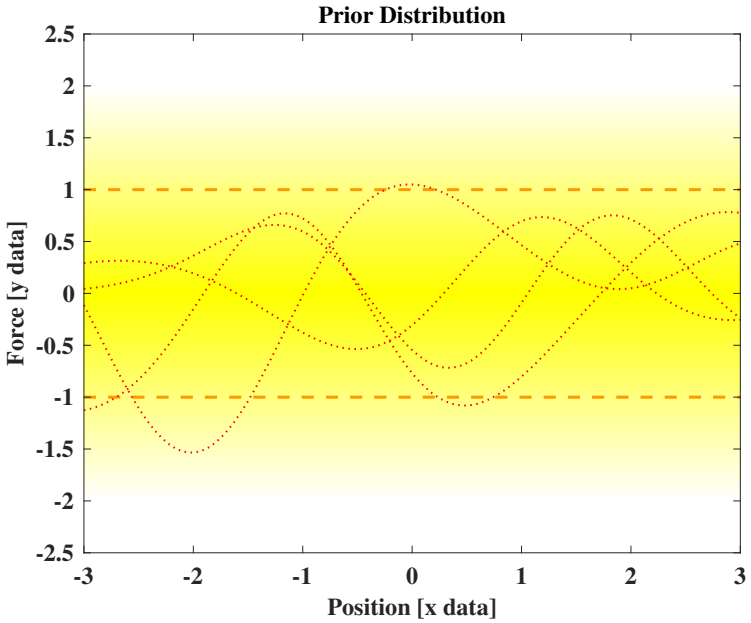


Figure 3.1: Example for drawn functions from a prior distribution

3.2.2 Posterior

After having observed function values $\mathbf{y}$ with $y_i = f(\mathbf{x}_i) + \varepsilon_i,\ i = 1 \dots N$, for a set of input vectors $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$, applying Bayes' theorem gives

$$p(f|\mathbf{X}, \mathbf{y}, \boldsymbol{\theta}) = \frac{p(\mathbf{y}|f, \mathbf{X}, \boldsymbol{\theta}) p(f|\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})}, \tag{3.5}$$

the posterior GP distribution over f.

The observations y_i can be assumed conditionally independent given $\mathbf{X}$. Therefore, the likelihood of f can be written as

$$p(\mathbf{y}|f, \mathbf{X}, \theta) = \prod_{i=1}^{N} p(y_i)|f(\mathbf{x}_i), \theta) = \prod_{i=1}^{N} \mathcal{N}(y_i|f(\mathbf{x}_i), \sigma_\varepsilon^2) = \mathcal{N}(\mathbf{y}|f(\mathbf{X}), \sigma_\varepsilon^2 \mathbf{I}), \tag{3.6}$$

where $\mathcal{N}(\mathbf{y}|f(\mathbf{X}), \sigma_\varepsilon^2 \mathbf{I})$ denotes the probability density function of multidimensional normal distribution for the random variable $\mathbf{y}$, with mean $f(\mathbf{X})$ and variance $\sigma_\varepsilon \mathbf{I}$. From (3.5) and (3.6), the mean and covariance function of the posterior distribution can be calculated as its expected value and covariance as [12]

$$\mathbb{E}\{f(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta)\} = \mathbf{k}(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} \mathbf{y}, \tag{3.7}$$

$$\operatorname{cov}\{f(\mathbf{x}_*), f(\tilde{\mathbf{x}}_*)|\mathbf{X}, \mathbf{y}, \theta)\} = K(\mathbf{x}_*, \tilde{\mathbf{x}}_*) - \mathbf{k}(\mathbf{x}_*, \mathbf{X})(\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I})^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}_*), \tag{3.8}$$

where $\mathbf{K} \in \mathbb{R}^{N \times N}$ is the kernel matrix with elements $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)\ i, j = 1 \dots N$, $\mathbf{k}(\mathbf{x}_*, \mathbf{X}) \in \mathbb{R}^{1 \times N}$ with $k_i = K(\mathbf{x}_*, \mathbf{x}_i)\ i = 1 \dots N$, and $\mathbf{x}_*, \tilde{\mathbf{x}}_* \in \mathbb{R}^D$ are the *test inputs* on which we evaluate the function f.

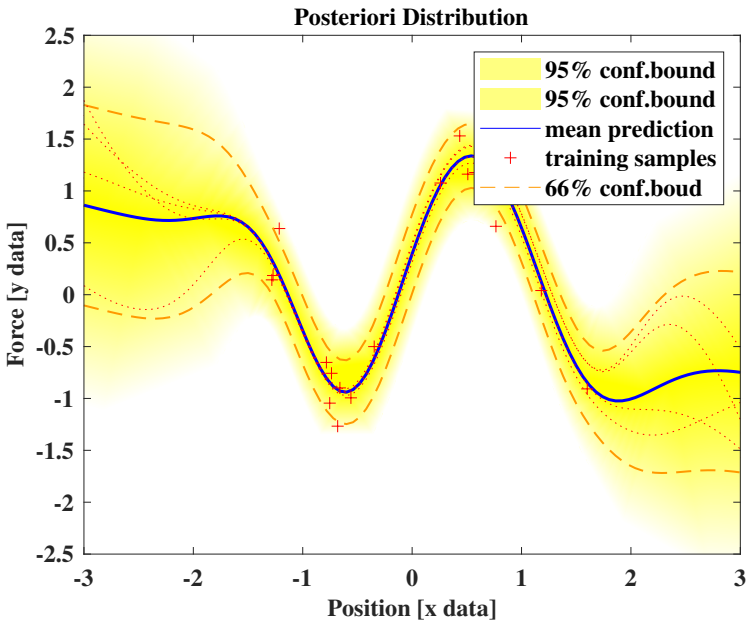


Figure 3.2: Example for a posterior distribution

3.2.3 Training of the Hyper-Parameters

The goal of training is to find the GP which is the most likely to have generated the observations (samples). In order to achieve that, the *Maximum a Posterior* (MAP) method is a common choice. The MAP estimate of the hyper-parameters $\boldsymbol{\theta}$ equals to maximizing the marginalized likelihood [7] of the output with respect to $\boldsymbol{\theta}$, often via maximizing the logarithm of it, which can be computed as

$$\begin{aligned}\log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}) &= \log \int_{-\infty}^{\infty} p(\mathbf{y}|f,\mathbf{X},\boldsymbol{\theta}) p(f|\boldsymbol{\theta}) \mathrm{d}f \\ &= \underbrace{-\frac{1}{2}\mathbf{y}^\top \left(\mathbf{K}+\sigma_\varepsilon^2\mathbf{I}\right)^{-1}\mathbf{y}}_{\text{data-fit term}} \underbrace{-\frac{1}{2}\log\left|\mathbf{K}+\sigma_\varepsilon^2\mathbf{I}\right|}_{\text{complexity term}} - \frac{D}{2}\log(2\pi). \end{aligned} \tag{3.9}$$

Estimation of $\boldsymbol{\theta}$ via marginalized likelihood has efficient implementations (analytic gradient) and leads to an automatic trade-off between data-fit and model complexity.

3.3 Predictions

As mentioned before, the aim of the regression is to make predictions which can estimate the behaviour of the controlled dynamical system. These predictions are calculated as the posterior distribution of $f(\mathbf{x}_*)$ at any test point $\mathbf{x}_*$. In the following, it is assumed that there is training data available and the hyper-parameters are trained using the maximization of marginalized likelihood. Calculation of predictions at deterministic and uncertain inputs will both be mentioned in this subsection. Always univariate predictions (i.e. scalar outputs) will be considered as only this case is important in the present application, but the formulae can be simply augmented to multivariate predictions as detailed in [12].

3.3.1 Predictions at Deterministic Inputs

Following Definition 1 the function values for test and training inputs are jointly Gaussian, therefore

$$p(f,f_*|\mathbf{X},\mathbf{x}_*) = \mathcal{N}\left(\begin{bmatrix} m(\mathbf{X}) \\ m(\mathbf{x}_*) \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k}(\mathbf{X},\mathbf{x}_*) \\ \mathbf{k}(\mathbf{x}_*,\mathbf{X}) & K(\mathbf{x}_*,\mathbf{x}_*) \end{bmatrix}\right). \tag{3.10}$$

From these, the predictive marginal distribution $p(f(\mathbf{x}_*)|\mathcal{D},\mathbf{x}_*)$ of the function value $f(\mathbf{x}_*)$ is Gaussian, and its mean and variance can be computed as

$$\begin{aligned}\mu_* := m(\mathbf{x}_*) :=& \mathbb{E}\{f(\mathbf{x}_*)|\mathbf{X},\mathbf{y}\} = \mathbf{k}(\mathbf{x}_*,\mathbf{X})(\mathbf{K}+\sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y} \\ =& \mathbf{k}(\mathbf{x}_*,\mathbf{X})\boldsymbol{\alpha} = \sum_{i=1}^{N}\alpha_i K(\mathbf{x}_i,\mathbf{x}_*), \end{aligned} \tag{3.11}$$

$$\sigma_*^2 := \sigma^2(\mathbf{x}_*) := \mathrm{var}\{f(\mathbf{x}_*)|\mathbf{X},\mathbf{y}\} = K(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*,\mathbf{X})(\mathbf{K}+\sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{k}(\mathbf{X},\mathbf{x}_*) \tag{3.12}$$

respectively, where

$$\boldsymbol{\alpha} := (\mathbf{K}+\sigma_\varepsilon^2\mathbf{I})^{-1}\mathbf{y}. \tag{3.13}$$

Note, that $\mathbf{K} + \sigma_\varepsilon^2 \mathbf{I}$ is a Hermitian, positive definite matrix, thus (3.13) can be calculated utilizing the computationally efficient Cholesky factorization, which is important for online learning.

3.3.2 Predictions at Uncertain Inputs

For time forecasting tasks – such as probabilistic MPC formulation – it is necessary to predict multiple steps ahead in time. As the current state is given (can be measured or observed), the first prediction can be calculated at deterministic input, but the output of it, which is the input of the next prediction will be uncertain. Although this uncertainty is usually not normally distributed, it is approximated with a Gaussian distribution, which gives a tractable computation formula for predictions at uncertain inputs.

For a Gaussian distributed input $\mathbf{x}_* \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the exact predictive distribution is

$$p(f(\mathbf{x}_*)|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \int p(f(\mathbf{x}_*)|\mathbf{x}_*) p(\mathbf{x}_*|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \mathrm{d}\mathbf{x}_*. \tag{3.14}$$

Approximating this with Gaussian distribution, the mean μ_* and variance σ_*^2 of the predictive distribution can be computed using the law of total expectation and law of total variance, respectively. The details of these computations can be found in [12, 13].

3.4 Sparse Gaussian Processes

Although Gaussian Process Regression is an efficient and flexible machine learning approach, the computational cost of training and evaluation is a major limitation, when there is an increased amount of collected data. As the computational cost of training scales at $\mathcal{O}(n^3)$ (with n data points), a few thousand points can result in large computation times [7]. In case of control design, where real-time evaluation is required with fast sampling times, sparse approximations are necessary to be computationally tractable [3].

There exist several approaches for sparse GP approximations, which are based on a subset of data points, that represent the collected data in a way that the quality of the regression remains unchanged. The elements of this certain subset are called *inducing* inputs and targets. From the many approximation methods investigated in [14], Fully Independent Training Conditional (FITC) [15] is widely used in control applications [3, 12]. This approach is based on a *pseudo-training set* with M data points, such that the predictive distribution of the GP on $\hat{\mathcal{D}}_M$ is equivalent with on $\mathcal{D}_N$, where $\mathcal{D}_N$ is the original (full) data set.

With defining the set of inducing inputs as $\mathbf{X}_{ind} = \{\mathbf{x}_{ind,i}\}_{i=1}^M$. The mean and variance of

the prediction using FITC sparse approximation can be calculated in the form

$$\tilde{\mu}_* = \mathbf{Q}(\mathbf{x}_*, \mathbf{X})(\mathbf{Q}(\mathbf{X}, \mathbf{X}) + \mathbf{L})^{-1}\mathbf{y}, \tag{3.15}$$

$$\tilde{\sigma}_* = K(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{Q}(\mathbf{x}_*, \mathbf{X})(\mathbf{Q}(\mathbf{X}, \mathbf{X}) + \mathbf{L})^{-1}\mathbf{Q}(\mathbf{X}, \mathbf{x}_*), \text{ with} \tag{3.16}$$

$$\mathbf{Q}(\mathbf{x}, \tilde{\mathbf{x}}) := \mathbf{K}(\mathbf{x}, \mathbf{X}_{ind})(\mathbf{K}(\mathbf{X}_{ind}, \mathbf{X}_{ind})^{-1}\mathbf{K}(\mathbf{X}_{ind}, \tilde{\mathbf{x}}), \tag{3.17}$$

$$\mathbf{L} := \text{diag}(\mathbf{K}(\mathbf{X}, \mathbf{X}) - \mathbf{Q}(\mathbf{X}, \mathbf{X}) + \sigma_\varepsilon^2\mathbf{I}). \tag{3.18}$$

For the selection of inducing inputs there are also multiple methods. In [15] the selection is based on evidence maximization, while in [3] a more heuristic, trajectory prediction-based method is described. The former is the optimal solution regarding the representation of the full set, but the latter requires far less computational cost, which makes it tractable at high sampling frequency real-time applications (as balancing the inverted pendulum).

4 Probabilistic Reinforcement Learning Control

First of the two machine learning approaches covered in the present work is probabilistic reinforcement learning. In the recent years, research showed that this kind of model-based method is efficient regarding the required learning time, representation of uncertainty and applicability for various dynamical systems [12, 16]. This part of the present work relies highly on Probabilistic Inference for Learning Control (PILCO), which is a Bayesian approach for data-efficient reinforcement learning [12]. PILCO contains a fully developed framework for implementing scenarios in simulation and also for real control systems.

4.1 Basic Principles

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. It is distinguished from other computational approaches by its emphasis on learning by an agent from direct interaction with its environment, without relying on exemplary supervision or complete models of the environment [17]. In contrast to supervised learning, correct outputs are not available, only rewards, therefore the way of learning is via finding an optimal behaviour, which is in the case of control an optimal control policy.

Unlike the standard approach of fitting a deterministic function to the data using neural networks or radial basis function networks, in this case a probabilistic dynamics model is considered. A probabilistic model is able to describe and quantify uncertainty regarding the learned model, which allows for the proper extension of available experience to unknown situations. That is, that at the beginning of a learning scenario only the structure of the dynamics model is available with user defined initial parameters. After collecting data via simulations, these parameters are tuned to describe the learned model properly. As more observations are made, the precision of the model increases and the uncertainty decreases gradually.

4.2 Policy Learning using PILCO

In this subsection the high-level steps of control policy learning will be discussed. The main parts of the process are described in Algorithm 1.

4.2.1 Settings

As the first step of the process, the settings of the learning have to be specified. These settings contain the plant structure, the policy structure, the cost function, the dynamics GP model structure, and the main properties of the scenario.

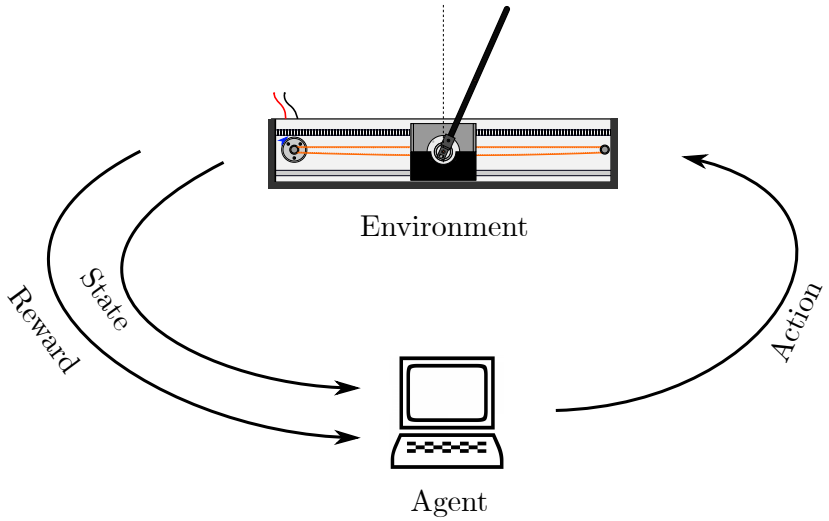


Figure 4.1: Typical framing of a reinforcement learning scenario

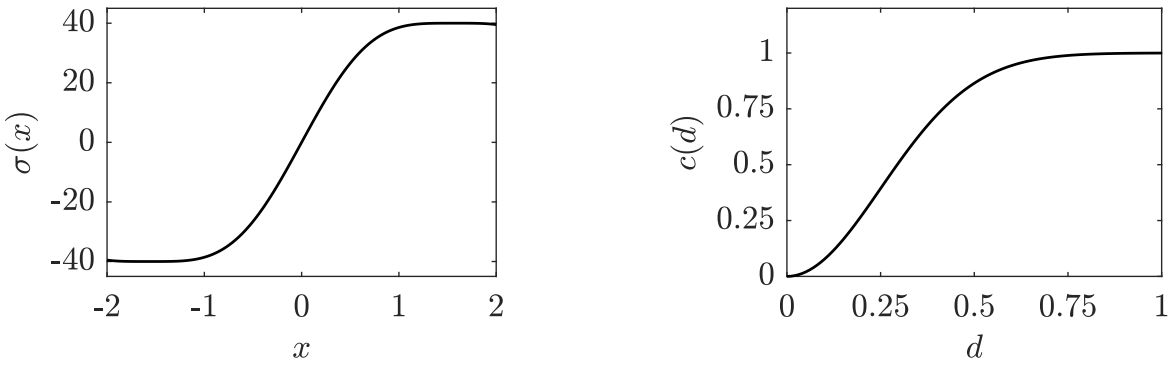
Algorithm 1 High-level steps of policy learning

```
Program and model settings
Initialize policy to random
loop
    Apply random policy
    Save observations
    Learn probabilistic dynamics model
    loop
        Apply policy π
        Save observations
        Compute expected long-term cost
        Optimize policy
    end loop
end loop
```

Plant structure

The plant structure needs to be specified to carry out simulations with different control policies. The deterministic dynamics model of the system has to be given in the form of ODEs in order to pass the function to a built-in solver, and simulate layouts. The model is given based on the state space representation in (2.25). However, it is somewhat modified, because of two reasons. Firstly, in PILCO the angle of the pendulum is measured anti-clockwise from hanging down, and secondly, an augmented state is used with the sine and cosine of the angle. As it is mapped to the unit circle, it is more tractable to consider the pendulum swinging multiple times. The augmented state vector used by the framework is

$$\mathbf{x} = [x\ \dot{x}\ \dot{\varphi}\ \sin\varphi\ \cos\varphi]^\top. \tag{4.1}$$



(a) Squashing function from (4.2)

(b) Saturating cost function from (4.5)

Figure 4.2: Visualization of applied squashing function and cost function

Policy structure

The control policy used during this work is based on the combination of a GP model, and a squashing function to model saturation. The GP model is a deterministic GP with squared exponential covariance function, which equals to the linear combination of radial basis functions (RBFs) for which the number of parameters, i.e. the number of basis functions and their values should be specified. The squashing function used to model saturation is the third-order Fourier series expansion of a trapezoidal wave, normalized to the interval $[-u_{max};\ u_{max}]$,

$$\sigma(x) = u_{max}\frac{9\sin x + \sin(3x)}{8}. \tag{4.2}$$

The maximum of the control signal, u_{max} also needs to be specified with the other parameters. As discussed in Section 3, a GP with squared exponential kernel has signal variance and length-scales as hyper-parameters. During the optimization process, these hyper-parameter values are tuned.

While the dynamics of the system is based on a continuous model, the control input changes only at sampling times. Although PILCO is able to consider multiple approaches, a zero-order-hold is set in the scenario, i.e. the control input is constant during a sampling interval.

Cost function

In this approach, the objective of the cost function is to penalize the Euclidean distance of the tip of the pendulum from its target state, i.e.

$$d(\mathbf{x}, \mathbf{x}_{target})^2 = x^2 + 2xl\sin\varphi + 2l^2 + 2l^2\cos\varphi. \tag{4.3}$$

As d only depends on the position and the trigonometric functions of the angle, a reduced state can be introduced as

$$\mathbf{j} := [x\ \ \sin\varphi\ \ \cos\varphi]^\top. \tag{4.4}$$

The target state vector is this way $\mathbf{j}_{target} = [0, 0, -1]^\top$. For the description of the cost, a saturating immediate cost function is used with the formula

$$c(\mathbf{x}) = c(\mathbf{j}) = 1 - \exp\left(-\frac{1}{2}(\mathbf{j} - \mathbf{j}_{target})^\top \mathbf{T}^{-1}(\mathbf{j} - \mathbf{j}_{target})\right) \quad \in [0, 1], \tag{4.5}$$

where the calculation of the new variables are

$$\mathbf{T}^{-1} := a^{-2} \begin{bmatrix} 1 & l & 0 \\ l & l^2 & 0 \\ 0 & 0 & l^2 \end{bmatrix} = a^{-2}\mathbf{C}^\top\mathbf{C}, \quad \text{with} \quad \mathbf{C} := \begin{bmatrix} 1 & l & 0 \\ 0 & 0 & l \end{bmatrix}, \tag{4.6}$$

where a is the length-scale of the cost function, set to a fixed value during the learning and simulation. It is worth noting that by multiplying $(\mathbf{j} - \mathbf{j}_{target})$ from left and right with $\mathbf{C}^\top\mathbf{C}$, d^2 is recovered.

Dynamics GP model structure

The dynamics GP model is a major element of the optimization. Initially it contains practically no information about the real system, all of it is learned from the collected data. After every iteration of making observations from simulations, the hyper-parameters are refined and optimized. In the structure of the dynamics model, functions are specified for the training and prediction, besides optimization parameters (like number of line searches for training). Training is evaluated using MAP estimation, as discussed in Section 3. The prediction is at uncertain input, which significantly increases calculation time. It can be evaluated using both full and sparse GP methods.

4.2.2 Learning Dynamics

After the initialization, a random policy is applied to the system in order to collect data as observations, and use them for the training of the probabilistic dynamics model. This procedure can be evaluated multiple times, but in the present application one is enough. As the goal of the control is to balance the pendulum without swinging around, it would be enough in this application to save the data in which the pendulum angle is within a certain range (e.g. $-15° < \varphi < 15°$), but considering the relatively small number of observations, it is tractable to compute with all of them.

4.2.3 Policy Optimization

The policy optimization is executed in the inner loop of Algorithm 1. First, the current policy is applied, and the behaviour of the system is observed. From the state values the expected long-term cost is calculated, based on which the policy hyper-parameters are optimized. This loop runs as many times as it is necessary to tune the policy parameters, it is up to the user to determine, whether the policy satisfies the expectations about control performance.

4.3 Application for Real Mechanism

The demonstrated framework can be used to design control for the implemented cart-pole system. In order to make it work, the policy parameters need to be passed to the microcontroller, and measurements back to the optimizing algorithm. This way the rollouts can be executed directly on the mechanism, simulation is not needed as a part of the process.

By the time of this document, the Arduino code of the microcontroller is ready for GP evaluation and data collection, but with such physical properties the learning is not successful. The properties causing difficulties include among others the limitation of the cart position and the resolution of the rotary encoder. The main issue during the learning process was that the cart is not able to stay within the available range (34 cm) for the needed 2.5 seconds. The development of the mechanism is an ongoing project, which means that after refining these parts, probabilistic RL control of the implemented cart-pole system can become possible to achieve.

5 Linear-Quadratic Control

5.1 Basic Principles

A feedback control using linear gains calculated with linear-quadratic (LQ) method is a quite simple and common way to systematically design a control with relatively good performance for the linearised system. This approach is a field of optimal control, in which the objective is to minimize a cost function with pre-specified weighting factors. In the present case, a discrete control input is specified as

$$u_k = -\mathbf{K_d}\mathbf{x}_k, \quad k \in \mathbb{Z}^+, \tag{5.1}$$

where $\mathbf{K_d}$ is the feedback gain vector. The dynamics are described in (2.25) by a continuous-time, non-linear model, which needs to be linearised and discretised in order to calculate the feedback gain.

5.2 Linearisation and Discretisation

The linear model is a first order Taylor approximation of the original system around the working point. As the target equilibrium state (working point) is $\mathbf{x} = \mathbf{0},\ u = 0$, the linear model has the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u, \tag{5.2}$$

where the state matrix $\mathbf{A}$ and control matrix $\mathbf{B}$ are calculated as the Jacobian of $f(\mathbf{x}, u)$ evaluated at the working point,

$$\mathbf{A} = \left.\frac{\partial f(\mathbf{x},u)}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\mathbf{0}\\u=0}}, \qquad \mathbf{B} = \left.\frac{\partial f(\mathbf{x},u)}{\partial u}\right|_{\substack{\mathbf{x}=\mathbf{0}\\u=0}}, \tag{5.3}$$

respectively. In order to calculate the discrete state and control matrices $\mathbf{A}_d$ and $\mathbf{B}_d$, the first-order Euler-method is applied, giving

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_s(\mathbf{A}\mathbf{x}_k + \mathbf{B}u_k) = \underbrace{(\mathbf{I} + T_s\mathbf{A})}_{\mathbf{A}_d}\mathbf{x}_k + \underbrace{T_s\mathbf{B}}_{\mathbf{B}_d}u_k, \quad k \in \mathbb{Z}^+. \tag{5.4}$$

5.3 Cost Function

The resulting optimal control gain matrix is reached by minimizing a discrete linear quadratic cost function in the form

$$J = \sum_{k=0}^{\infty}(\mathbf{x}_k^\top \mathbf{Q}\mathbf{x}_k + u_k R u_k), \tag{5.5}$$

where $\mathbf{Q}$ and R contain the weights. The optimal $\mathbf{K_d}$ vector is calculated as the solution of the discrete Riccati equation [18].

It is worth noting that the LQ control in the form of

$$u_k = -\begin{bmatrix} K_1 & K_2 & K_3 & K_4 \end{bmatrix} \begin{bmatrix} \varphi \\ \dot{\varphi} \\ x \\ \dot{x} \end{bmatrix} \tag{5.6}$$

is equivalent to a PD control for the angle and the position, if we consider these two states only and $\mathbf{K_d} = [P_\varphi \ D_\varphi \ P_x \ D_x]$. The advantage of the LQ approach over classical PD controller is that the weight matrices can be used for the MPC as well, and there is a systematic way of determining an optimal feedback gain. However, a major disadvantage over MPC is that this method is not able to handle any kind of constraints either for the control input, or the states.

6 Probabilistic Model Predictive Control

Model predictive control is used to control a dynamic system while satisfying a set of state and input constraints. It is based on a mathematical model of the system or process, often obtained by system identification. Using this model predictions are made on a finite time horizon, from which an optimal control input can be calculated by the minimization of a specific cost function. Compared to LQ control, MPC optimizes over a finite horizon, and is capable of non-linear control design, such as the swing-up of an inverted pendulum, while satisfying state and control input constraints.

6.1 Deterministic MPC

In order to design GPMPC, it is useful to start with the deterministic (nominal) approach. After the development of a working framework for nominal MPC, the augmentation with self-learning is much easier. Deterministic MPC can be designed both in a linear and non-linear way. As GPMPC augments the linear model, only this approach is discussed in the present work. Non-linear deterministic MPC behaves similar near the working point around which linear MPC is designed, but has far more computational cost because of the non-linear optimization, therefore it is not discussed in the present work. However, the implementation of the resulting simulation is simple to augment for the non-linear case, only few modifications are needed. The high-level steps of linear MPC process are described in Algorithm 2.

Algorithm 2 High-level steps of deterministic linear MPC

1: Formulating discrete, linear system model
2: MPC and simulation parameters
3: **loop**
4: Prediction along the horizon
5: Calculating control input via cost function minimization
6: Applying control input, simulation of dynamics
7: **end loop**

The linear discrete dynamics model is from (5.4), the same model can be applied as in linear-quadratic control. The MPC algorithm can optimize the control input u directly, but it is common to seek u in the form

$$u = -\mathbf{K}\mathbf{x} + v, \tag{6.1}$$

where $\mathbf{K}$ is a linear feedback gain and v is the additional input, which is then optimized. Using (6.1), the MPC algorithm can be designed to a stable system, which makes it easier to design and compare with LQ approach (if using feedback gain designed with LQ method).

The discrete cost function is similar to (5.5), but for v on a finite horizon,

$$J = \sum_{k=0}^{H} (\mathbf{x}_k^\top \mathbf{Q} \mathbf{x}_k + v_k R v_k), \tag{6.2}$$

where H is the length of the prediction horizon (number of predicted time steps). The weight matrices are the same as in the LQ case.

As in every time step of the scenario an optimal control problem is solved, constraints can be formulated for the control input and the state variables. For example, the mathematical shaping for the voltage range of the DC motor is

$$\begin{aligned} u = -\mathbf{K}\mathbf{x} + v < u_{max} \;&\Leftrightarrow v < u_{max} + \mathbf{K}\mathbf{x}, \\ -\mathbf{K}\mathbf{x} + v > -u_{max} \;&\Leftrightarrow v > -u_{max} + \mathbf{K}\mathbf{x}. \end{aligned} \tag{6.3}$$

After calculating the control inputs along the horizon, only the first input is applied to the system, with which a time step is simulated, using the original continuous-time, nonlinear model. The same procedure is done in every sampling time instant, until the end of the scenario.

6.2 Gaussian Process MPC

Gaussian process MPC has the advantage over deterministic MPC, that it can calculate with the uncertainty of the predictions, and the collected data during operation. The high-level steps of the control process are described in Algorithm 3.

Algorithm 3 High-level steps of GPMPC

1: Linear, discrete system model
2: GP initialization and training
3: MPC and simulation parameters
4: **loop**
5: Prediction using GP evaluation and nominal model
6: Calculating control input via cost function minimization
7: Applying control input, simulation of dynamics
8: Saving observations for training points
9: **end loop**

6.2.1 Dynamics Model and Cost Function

The structure is very similar to linear MPC, but the state $\mathbf{x}$ in this case is a random variable with mean $\mathbf{m}$ and variance $\mathbf{\Sigma}$. The linear dynamics model is completed with an additional term,

$$\mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d u_k + \Delta_{GP}(\mathbf{x}_k), \tag{6.4}$$

where $\Delta_{GP}(\mathbf{x}_k)$ is a Gaussian process describing the difference between the behaviour of the real system and the linear model. That is, the input of this GP is the current state, and the output is the approximation of the difference between the observed next state and the calculated one.

As mentioned, the state is a random variable in this case, therefore predictions at uncertain inputs have to be made in order to evaluate the GP. Using the method described in Section 3, every state is approximated by a Gaussian distribution in the form $\mathbf{x} \sim \mathcal{N}(\mathbf{m}, \mathbf{\Sigma})$. The formula of the quadratic cost function is similar to (6.2), but it is now a linear transformation of a Gaussian distributed random variable, i.e., another Gaussian distributed random variable. The objective of probabilistic MPC is the minimization of the expected cost with respect to the control input. It is proved in [19] that

$$\mathbb{E}\{\mathbf{x}^\top \mathbf{Q}\mathbf{x}\} = \mathrm{Tr}(\mathbf{Q}\mathbf{\Sigma}) + \mathbf{m}^\top \mathbf{Q}\mathbf{m}, \tag{6.5}$$

where $\mathrm{Tr}(\cdot)$ is the trace of the corresponding matrix. Using this formula, the expected cost is

$$J = \sum_{k=0}^{H} (\mathrm{Tr}(\mathbf{Q}\mathbf{\Sigma}_k) + \mathbf{m}_k^\top \mathbf{Q}\mathbf{m}_k + v_k R v_k), \tag{6.6}$$

with $u_k = -\mathbf{K}\mathbf{m}_k + v_k$, as in the linear case.

6.2.2 State Constraints

For the probabilistic state $\mathbf{x}$, linear constraints are considered [3] in the form

$$\mathbf{H}\mathbf{x} \leq \mathbf{h}, \quad \mathbf{H}_j^\top \mathbf{x} \leq h_j \quad j = 1 \dots m,$$

where $\mathbf{H} \in \mathbb{R}^{m \times D}$ and $\mathbf{h} \in \mathbb{R}^m$ are the constraint matrix and vector, respectively, and m is the number of linear constraints. As $\mathbf{x}$ is a random variable, the constraints are to satisfied in probabilistic sense, i.e.

$$P(\mathbf{H}_j^\top \mathbf{x} \leq h_j) > \overline{p}, \tag{6.7}$$

which is called a chance constraint, where $\overline{p}$ is the probability of satisfying the constraint. Let a new variable z_j be

$$z_j := \mathbf{H}_j^\top \mathbf{x} - h_j \quad \rightarrow \quad z_j \sim \mathcal{N}(\underbrace{\mathbf{H}_j^\top \mathbf{m} - h_j}_{m_j}, \underbrace{\mathbf{H}_j^\top \mathbf{\Sigma} \mathbf{H}_j}_{\sigma_j}).$$

Let F be the distribution function of z_j and Q be the quantile function (inverse of F). The quantile function satisfies the Galois inequality:

$$F(\alpha) = P(z_j \leq \alpha) \geq \overline{p} \quad \Leftrightarrow \quad Q(\overline{p}) \leq \alpha. \tag{6.8}$$

In our case, $\alpha = 0$, therefore

$$F(0) = P(z_j \leq 0) \geq \overline{p} \quad \Leftrightarrow \quad Q(\overline{p}) \leq 0, \tag{6.9}$$

which is a *deterministic constraint* for

$$Q(\overline{p}) = m_j + \sigma_j \sqrt{2} \mathrm{erf}^{-1}(2\overline{p} - 1), \tag{6.10}$$

where $\mathrm{erf}(\cdot)$ is the Gauss error function. Since $\overline{p}$ is a pre-specified constant, $Q(\overline{p})$ is linear both in m_j and σ_j, and thus in $\mathbf{m}$ and $\mathbf{\Sigma}$ as well.

As it is derived, a tractable formulation can be given for linear chance constraints, as they can be transformed into deterministic, linear constraints on the mean and variance of the Gaussian distributed state. Further different constraints are discussed in [3].

7 Simulation Results

In this section the most important results of this work, the simulations with the two different self-learning control approaches along with the classical LQ approach, and their comparison are presented. The goal at every case is the balancing of the pendulum at the upper equilibrium, with the cart position being zero. The initial conditions are always close to the target state, therefore there is no need for swing-up. However, it is important to note that non-linear controllers, as the probabilistic RL control and MPC, GPMPC algorithms are able to learn and handle non-linearities, such as the swing-up of an inverted pendulum. In contrast, PD and LQ are linear controllers, therefore they can only control the system within a certain limited range around the target state.

In order to become closer to the theoretical cart-pole dynamic system, it is assumed that the implemented mechanism can be developed to reduce the friction coefficient to $\mu_c = 0.17$, which is realistic between lubricated plastic and aluminium or steel [20]. There could be several ways to reduce friction, e.g. a linear rail with roller bearings, or proper lubrication. The characteristics of the friction remain the same based on Section 2, only with $a = 0.2$ in (2.23).

Regarding the initial conditions of the simulations, the angle is $\varphi = 0.2$ rad $\approx 11.46°$, while all other states are zero. The sampling time is $T_s = 10$ ms in case of the LQ and MPC controllers, and $T_s = 50$ ms in case of probabilistic RL control. Note, that an important advantage of control design using PILCO is that lower sampling frequency can be used than the classical controllers [21].

The constraints for the state and control input are

$$-0.17 \leq x \leq 0.17, \qquad -40 \leq u \leq 40. \tag{7.1}$$

7.1 Linear-Quadratic Controller

For the calculation of the quadratic cost, the weight of the states ($\mathbf{Q}$), and the control input (R) are specified as

$$\mathbf{Q} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad R = 0.0001, \tag{7.2}$$

with $Q_{11} = 100$ being the weight of pendulum angle, and $Q_{33} = 1$ the weight of cart position. After linearisation and discretisation, the discrete state and control matrices

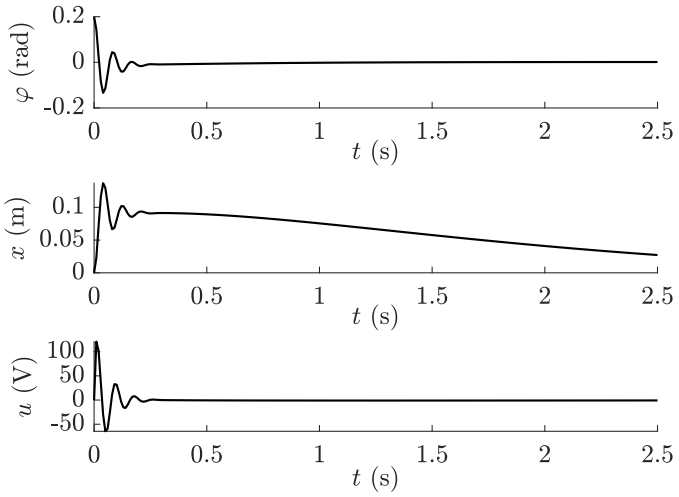


Figure 7.1: Trajectories using LQ feedback controller. The control input is out of the allowed range, but the controller is able to balance the pendulum

are numerically

$$\mathbf{A}_d = \begin{bmatrix} 1 & 0.01 & 0 & 0 \\ 0.36 & 1 & 0 & 5.12 \\ 0 & 0 & 1 & 0.01 \\ -0.046 & 0 & 0 & -1.05 \end{bmatrix}, \qquad \mathbf{B}_d = \begin{bmatrix} 0 \\ -0.1232 \\ 0 \\ 0.0493 \end{bmatrix}, \tag{7.3}$$

calculated from (5.3) and (5.4). The linear feedback gain matrix is

$$\mathbf{K_d} = \begin{bmatrix} -603.760 & -50.449 & -54.667 & -126.792, \end{bmatrix} \tag{7.4}$$

calculated with the `dlqr` function of Matlab. The trajectories of angle, position and control voltage are shown on Figure 7.1.

The simple linear feedback control does not contain any constraints, therefore at the beginning of the scenario, the control voltage is allowed to be $u = 120.75$ V, which is more than 3 times the maximum applicable voltage, 40 V. If this scenario was carried out on the real mechanism, two issues are possible. Firstly, if there is no protection from overvoltage, either the motor or the motor driver circuit would suffer damage. Secondly, if there is a software or hardware to prevent higher voltage than the allowed, the control would fail.

Although the cart position is inside the allowed range in this simulation, with different initial conditions or higher friction coefficient, it could also violate the corresponding constraint.

The conclusion of the LQ control simulation is that it works well around the target equilibrium state, but it is unable to handle constraints, which is a problem regarding the maximum applicable voltage of the DC motor. Another disadvantage over the probabilistic control approaches that it can not consider uncertainties. Unmodelled factors such as the threshold and noise of the sensors, or neglected dynamics of the system can affect this scenario in a much significant way than self-learning approaches.

7.2 Probabilistic RL Controller

Control design using probabilistic reinforcement learning is fully data-based, both the dynamics model and the control policy are learned from observed data during operation. The cost, the states and the control input are all random variables, the variances of which describe the measure of uncertainties.

After one trial the control fails, but with only 5 seconds of experience and 2 policy searches, it manages to balance the pendulum. With further observation and optimization steps, the uncertainty of the cost and the trajectories decreases, the prediction becomes more accurate. As Table 7.1 shows, both the expected value and the variance of the immediate cost reduces with the number of iterations.

Even with 25 seconds of experience and ten policy optimization steps, significant uncertainties appear on the trajectories of the controlled system as shown on Figure 7.3, and they do not converge to zero. This comes mainly from the white noise that is added to the measurements at every time step, in order to represent the real sensor noise and threshold.

Regarding the constraints, the control input is kept between its minimum and maximum via the squashing function in (4.2). However, state constraints are not included in PILCO, therefore the cart can move freely without boundaries. This would be a serious issue, if the learning process would be executed on the real system, because at the early stages of learning, the position reaches even $|x| = 0.39$ m, which is more than double of the allowed displacement. This is a major drawback of RL control, because constraints often mean safety guarantees as well. For example in case of autonomous vehicle control, safe operation is essential, therefore MPC is a more relevant approach.

Although this approach does not require on-line learning, the computational cost of model and policy optimization is quite high. Many optimization steps of non-linear functions are necessary, and GP evaluations with great number of training points take also lots of time. However, as a result the algorithm learns the dynamics fully from collected data, and gives an optimal control policy with the uncertainties taken into account.

Another drawback of probabilistic RL control is that it is not capable of trajectory tracking control. In this project, the only objective is balancing the inverted pendulum, but for other purposes as autonomous vehicle control or collaborative robots, trajectory tracking is essential and all the other presented control methods (LQ, MPC) are capable of it. On the other hand, PILCO makes it possible to learn one policy for multiple targets [22], which is a strong asset, as only one learning scenario is enough to design a control for multiple targets. Classical control methods (LQ, deterministic MPC) require expert knowledge to tune weights, and the weights and gains are scenario-specific, i.e. they need to be redesigned for a different control target.

[2]Standard Deviation

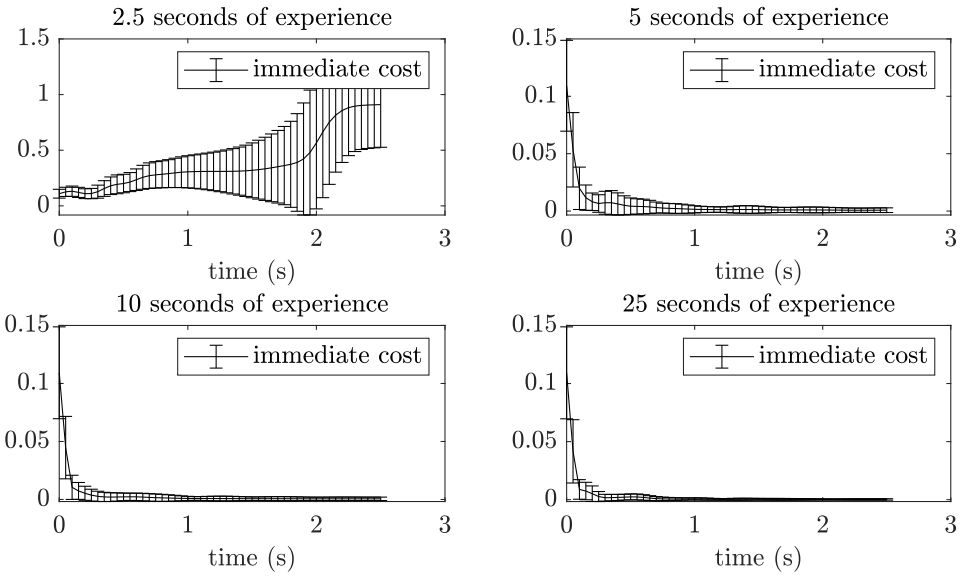


Figure 7.2: Immediate cost of states after 1, 2, 4, and 10 policy searches, calculated from (4.5)

Table 7.1: Total cost of simulations at different steps of learning process (sum of immediate costs). Experience here means the total simulation time, during which data is collected

Experience (s)	2.5	5	7.5	10	12.5	15	17.5	20	22.5	25
Mean	19.848	0.297	0.234	0.228	0.231	0.232	0.232	0.229	0.216	0.204
STD[2]	6.139	0.141	0.096	0.095	0.095	0.096	0.097	0.097	0.082	0.072

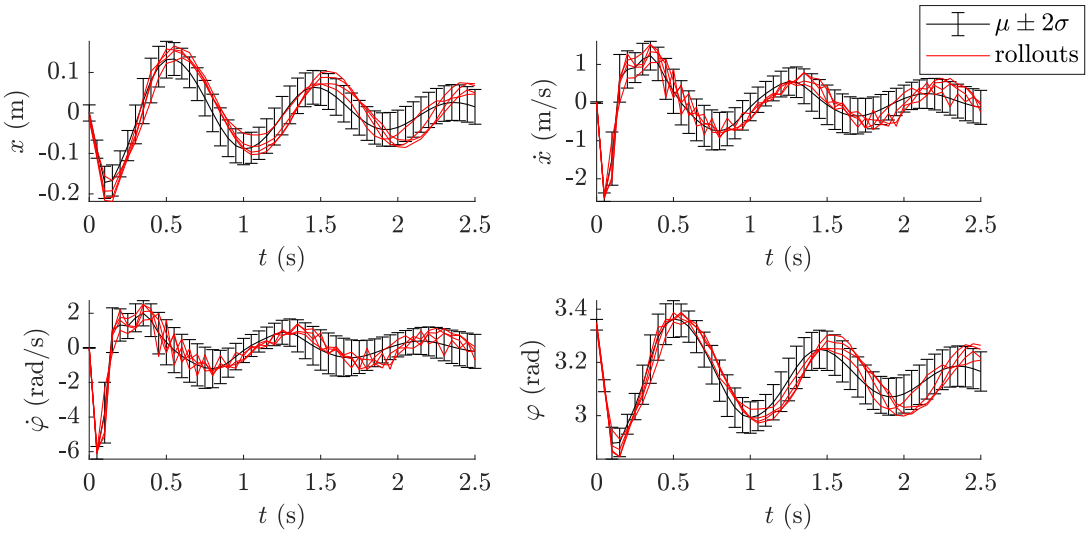


Figure 7.3: Trajectory rollouts using optimized policy, after 10 policy searches and 25 seconds of experience

7.3 Linear Deterministic MPC

The concept of linear deterministic MPC is very similar to that of the LQ controller, except that in this case the optimization is carried out at every sampling time instant, for a finite horizon. This way, a set of inputs $\{u_i\}_{i=k}^{k+H}$ is calculated, where k is the discrete time instant, and H is the length of prediction horizon, specified in this case to $H = 25$. The optimization is carried out using `fmincon`, a built-in non-linear optimizer function of Matlab, which is able to handle both linear and non-linear constraints. From the set of calculated control inputs, only the first one (u_k in time instant k) is applied to the system for a sampling interval, the others are used to initialize the next optimization process at the next sampling time instant $k + 1$.

As described in (6.1), a linear feedback term is added to the optimized MPC input v, resulting in $u = -\mathbf{K}\mathbf{x} + v$. The gain matrix $\mathbf{K}$ is calculated using discrete LQ method, it is the same matrix as (7.4), and v is the on-line optimized term.

At the current state of the developed framework, `fmincon` contains constraints only for the optimized control input, and not for the state. The form of these constraints is as described in (6.3). Although between the circumstances of the presented simulation the cart position is within the allowed range, the program will be completed with all necessary constraints for state variables in the near future.

The results of simulation using linear MPC is very similar to LQ control, but it is an important property that the constraints are satisfied in this case. It can be seen on Figure 7.4, that the control input saturates ate the beginning of the simulation, and after some oscillations it reduces to a small value, while the angle is very close to zero, and the position slowly converges as well. Besides being the base of the developed probabilistic MPC framework, this approach is also useful for collecting observations as initial training data of the GPMPC scenario. State action pairs are saved with the same parameters (sampling time, weight and gain matrices, horizon length) as in the GPMPC simulation.

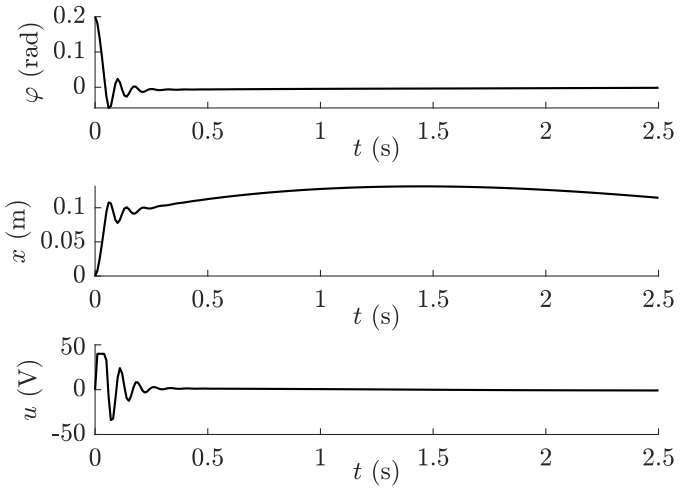


Figure 7.4: Trajectories using linear deterministic MPC control. Constraint for the control input is satisfied, and the balancing is successful

7.4 Gaussian Process MPC

From the point of view of application and further improvement, the most important part of this work is the design of a GPMPC framework. In the current state of development, it has major deficiencies and limitations, but is already capable of a better control than the deterministic approaches. There are two important limitations of the current version. Firstly, the training inputs and targets are based on the results of linear MPC simulation with equivalent parameters, i.e. no active learning is included. This implicates, that there is no need for sparse approximation, thus FITC is also not implemented yet. Secondly, the GP at uncertain inputs is evaluated with mean equivalent approximation [3], i.e., by assuming deterministic inputs. This way the uncertainty is not propagated over the prediction horizon, but it is computationally the cheapest approach. A third limitation similarly to linear MPC is that the state constraints – discussed in Section 6.2.2 – are also not implemented yet.

Algorithm 4 High-level steps of implemented GPMPC simulation

1: Calculating linear, discrete system model, linear feedback gain
2: GP initialization and training based on linear MPC simulation observations
3: MPC and simulation parameters – weights, horizon length
4: **loop**
5: Prediction using linear model with mean equivalent GP approximation
6: Calculating control input via quadratic cost function minimization
7: Applying control input, simulation of dynamics
8: **end loop**

7.4.1 Training and Hyper-Parameters of GP

Based on (6.4), the Gaussian process Δ_{GP} has inputs $\mathbf{x}_k$ and corresponding targets

$$\mathbf{y}_k = \mathbf{x}_{k+1} - \mathbf{A}_d\mathbf{x}_k - \mathbf{B}_d u_k, \tag{7.5}$$

with k being the time instant of the collected observations. As there are four inputs and four outputs, Δ_{GP} is described by four independent Gaussian processes, each with four different length-scales. From the linear MPC simulation discussed in Section 7.3, 15 data pairs are collected per dimension, with time indices $k = 2, 4, \ldots 30$, in total 60 training inputs and targets. The selection of these data pairs is purely heuristic, the systematic data collection is subject of further development.

The training of the hyper-parameters is carried out with `train` function of PILCO, which uses MAP estimation, described in Section 3.2.3. The signal and noise variances used in the simulation are $\sigma_f = 0.1$ and $\sigma_\varepsilon = 0.01$, respectively. The length-scales of the trained GPs can be found in Table 7.2.

7.4.2 Parameters and Results

Regarding the parameters of the simulation, same sampling time, weight and feedback gain matrices, and horizon length are used as in the linear MPC case. Although only 15 training inputs and targets are used per state, the difference compared to linear MPC is significant, there are less oscillations, and the settling time is lower, which both leads to less energy requirement.

The computational time is high compared to linear MPC, because even with this simple GP prediction method and 60 training data-pairs, the evaluation of (3.11) and (3.12) contain full matrix-vector multiplications which have to be calculated for all states independently.

The result of one prediction at time instant $t = 0.12$ s is demonstrated on Figure 7.6. At this point, the time interval of the prediction is $t \in [t_1; t_2]$, where $t_1 = 0.12$ s, and $t_2 = t_1 + T_s H = 0.37$ s, with sampling time $T_s = 0.01$ s, and horizon length $H = 25$. The errorbars present the prediction using the control input calculated at t_1, and the linear model augmented with the GP from (6.4). The errorbars demonstrate the mean and two times the standard deviation for each sampling time instant. The red lines are the simulation results within the time interval $[t_1; t_2]$, the same as on Figure 7.5.

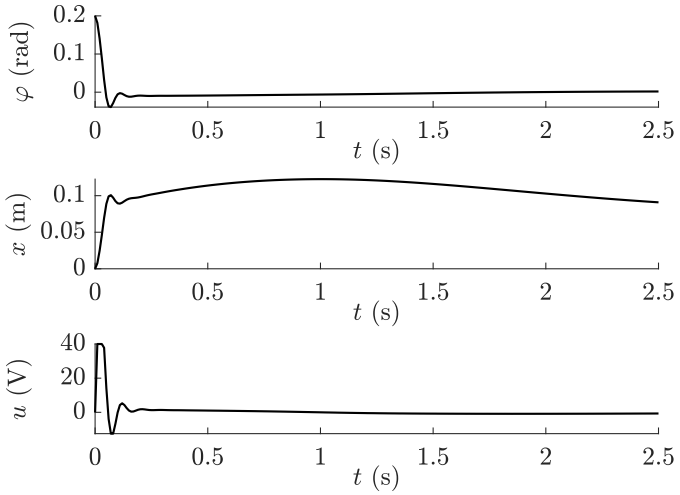


Figure 7.5: Trajectories using Gaussian process MPC control. Constraint for the control input is satisfied, and the balancing is successful with better performance than deterministic approaches

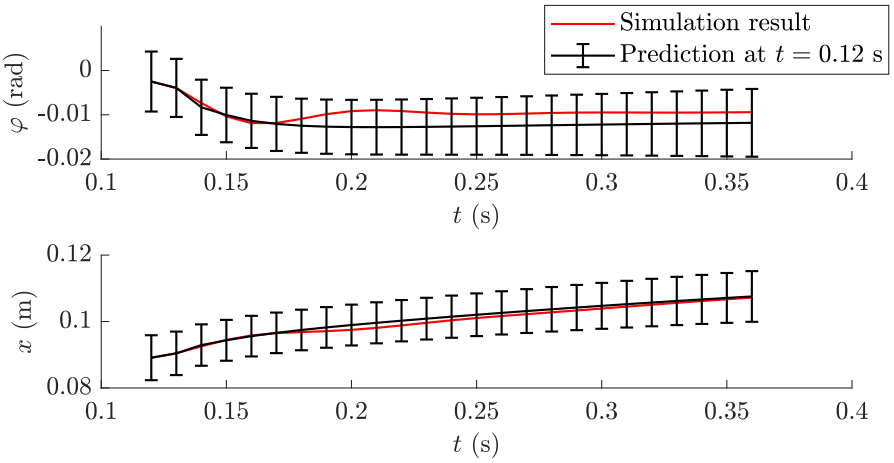


Figure 7.6: The predicted layout at time instant $t = 0.12$ s, and the results of the simulation at the prediction time interval

Table 7.2: Length-scales of trained Gaussian process with corresponding input at the left column, and output at the top row

	φ		$\dot{\varphi}$		x		$\dot{x}$	
φ	λ_1	0.1091	λ_1	3.3321	λ_1	0.1093	λ_1	3.3332
$\dot{\varphi}$	λ_2	21.3074	λ_2	4.9289	λ_2	21.4316	λ_2	4.9284
x	λ_3	0.4071	λ_3	0.0831	λ_3	0.4100	λ_3	0.0832
$\dot{x}$	λ_4	38.0903	λ_4	30.7287	λ_4	38.1277	λ_4	30.7344

7.5 Summary of Simulation Results

The simulation results are summarized and compared in this subsection based on their control performance, for which the error from the target pendulum angle and cart position are considered. From Figure 7.7 it can be seen, that the probabilistic RL control (PILCO) scenario is significantly different from the others. It can be explained partly with the higher sampling time, but it is a fully different approach. However, in order to compare the results quantitatively, Table 7.3 contains the root mean square error of the angle, position and control input signals. The calculation of root mean square error for a given discrete time error signal $w(k)$ $k = 1 \dots N$ is

$$RMS(w) = \sqrt{\frac{1}{N} \sum_{k=1}^{N} |w(k)|^2}, \tag{7.6}$$

which gives a quantitative measure for the deviation from zero. The error for the pendulum angle is similar for the LQ and GPMPC controllers, but an order of magnitude higher for PILCO. The RL controller is a lot slower to converge, and has large amplitude oscillations. The error of the position is quite similar for all simulations, but due to different causes. LQ and GPMPC have an almost constant error, very slowly converging to zero, and the PILCO signal has multiple oscillations, with same frequency as the pendulum angle signal. The RMS value of the control input is very significant, because it is related to the energy needed for the control process. The value is similar for the LQ and PILCO simulations, and less than half for the GPMPC. It means, that GPMPC is the most power-efficient approach, while it was the only method to satisfy both constraints.

Table 7.3: Root mean square errors and constraint satisfactions of the control approaches

	$e_{\varphi,RMS}$	$e_{x,RMS}$	$e_{u,RMS}$	Constraint for $\|u\|_{max}$	Constraint for $\|x\|_{max}$
LQ	0.0214	0.0682	12.438	✗	✓
GPMPC	0.0194	0.1060	4.837	✓	✓
PILCO	0.1536	0.1000	13.066	✓	✗

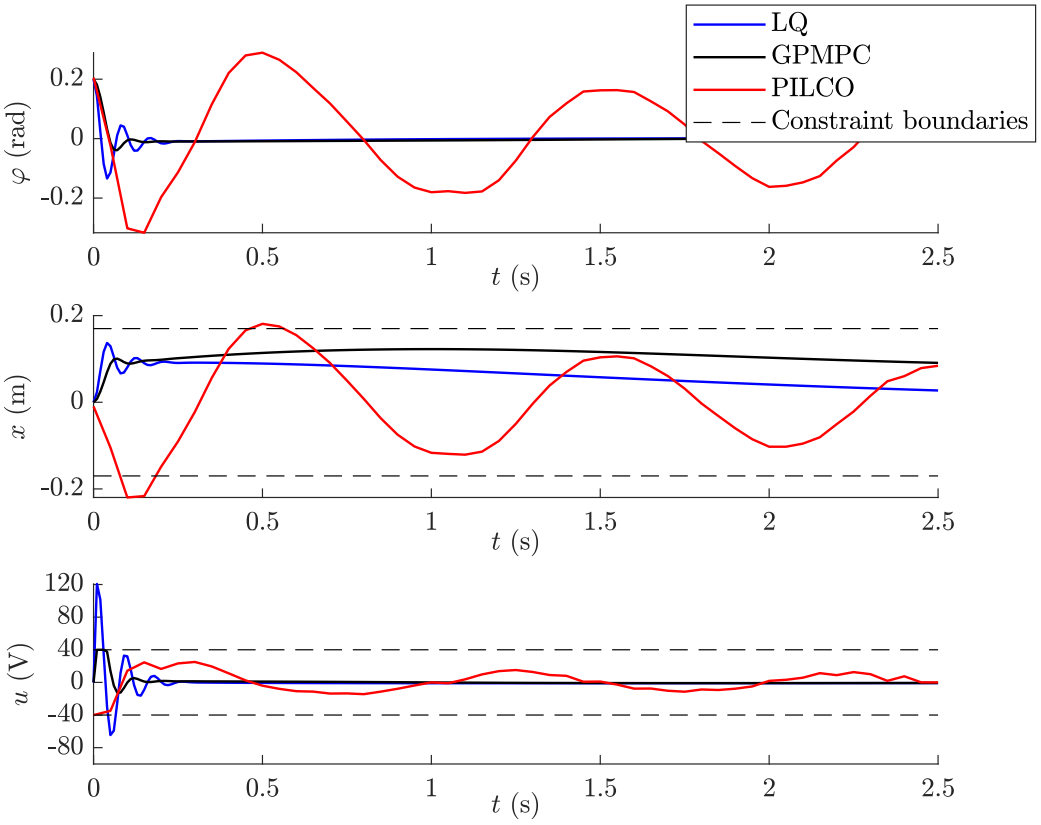


Figure 7.7: The trajectories of the angle and position with the presented control approaches

8 Summary and Conclusions

The goal of the work is achieved, the demonstration of the joint GPML and MPC approach was successful in simulation domain. In this section, firstly the advantages and disadvantages of the control strategies are covered from different aspects. Secondly, conclusion is drawn about the results, and as a closing, opportunities for further development is discussed.

8.1 Comparison of Control Approaches

In this subsection, the presented control approaches are compared based on four aspects: computational cost, complexity of application, handling constraints, and control performance.

Computational cost can be divided to on-line and off-line computations. Off-line computation means a problem only when there is limited time available to achieve results. However, on-line computations can make it impossible to operate with high sampling frequencies, which can lead to unsolvable control problems. The simplest approach, LQ control needs only off-line optimization, and it is negligible compared to the other methods. By probabilistic RL control the optimization is also off-line, but it requires a significant computation time, for the presented scenario it took about 40 minutes to learn the policy. GPMPC uses on-line optimization and learning, which is a hard task, considering a few tens of milliseconds sampling times, during which the GP predictions need to be made iteratively for optimizing the control input.

From application point of view, LQ and probabilistic RL are equally simple to use. Both need only a system model, and few parameters to adjust manually. For the LQ, these parameters are the weight matrices, and for PILCO, e.g. the cost function width, and prediction horizon length. GPMPC is less automatic, a training set has to be provided along with other parameters, and the results are sensible to the user-defined properties.

Regarding constraints, LQ is unable to handle any of them. PILCO has control input limitation, but no state constraints, and GPMPC can handle all kinds of constraints, providing safety guarantees.

The control performance is difficult to compare because of the different sampling rates, but it is clear that GPMPC controlled the system in the shortest time using least energy to the immediate vicinity of the target position, while satisfying the constraint of the control input.

As a general conclusion, PILCO is a very efficient, simply applicable black box control policy designer, which has limitations due to its lack of handling constraints. GPMPC has the best properties in most relevant areas, but the efficient implementation is important to be applicable in high sampling time control applications.

8.2 Opportunities for Further Improvement

So far only simulation results are available, none of the control methods works fully on the real system. However, the control approaches were selected in a way that they are also applicable for the implemented mechanism, therefore the *short-term* goal is to implement the presented control algorithms in Arduino language, and investigate their performance on the real system. In order to achieve better physical properties (reduce friction and sensor noise), the development of the cart-pole is ongoing, currently with the acquisition of a higher resolution rotary encoder, and a linear rail with roller bearings.

In parallel with this process, the GPMPC approach will be investigated in further details, and the framework completed with on-line learning, exact moment matching prediction [3], state constraints, and sparse approximation. The development of this framework leads to the *long-term* goal, which is the application of GPMPC to Unmanned Autonomous Vehicles (UAVs) in real-life test environment. For such safety critical systems safe learning is a key concern, for which probabilistic MPC formulation gives a solid and useful basis.

A Simulation Program Codes

In this part, the simulation Matlab codes are demonstrated. All files can be found in the GitHub repository `https://github.com/antalpeter1999/TDK_2020`.

A.1 Simulation Using PILCO

PILCO framework is publicly available at `http://mlg.eng.cam.ac.uk/pilco/`, where the current version is pilcoV0.9, this is used for the presented simulation. The `scenarios/cartPole` files are reworked in order to simulate the implemented cart-pole system (match the dynamics model and the parameters).

The main script of the probabilistic RL simulation is the `cp.m` script. It is the only file that has to be ran by the user. The high-level steps of the script are demonstrated in Algorithm 1. In this script, the verbosity of printing and plotting data during the simulation can be adjusted by giving values to the variables `flag.plotting` and `flag.printing`. The number of initial trials with random policy is stored in `N_initial`, and the number of episodes (number of policy searches) is given in `N_episodes`.

The other scripts redesigned in this work are `cp_settings.m`, and `cp_dynamics.m`. In the settings file, parameters of the simulation can be adjusted, e.g. sampling time, horizon length, cost function width, or constraint for the control input. The dynamics script contains the parameters of the cart-pole system and the state space representation in (2.25), adjusted to the modified states, which PILCO uses.

The simulation data is saved automatically with name specified in the `basename` variable of `cp.settings`. The data presented in Section 7.2 can be found in the repository `pilco_simuresults`.

A.2 Linear-Quadratic Regulator

The linear-quadratic control simulation is executed by the script `cp_LQR.m`. The script starts by loading the parameters of the cart-pole system from `cp_params.mat` Matlab data file. These parameters include the ones specified in Table 2.1, along with the a and b parameters of the friction function (2.23). The discrete, linear system model is formulated using the linearisation and discretisation in (5.3) and (5.4), based on the state space representation (2.25) by the function `cp_dynmodel.m`. The inputs of this function are the current state, and the control input, from which it calculates the derivative of the state as an output.

For calculating the linear feedback gain, the `dlqr` Matlab function is used, with the discrete, linear state space model, and the states as outputs. Weight matrices are specified in variables `Q` and `R`, as discussed in Section 7.1. The control scenario is simulated in a loop, where in every sampling time instant the new control input is calculated from (5.1),

and the motion is simulated using the `ode45` integrator of Matlab.

At the end of the script, plots are made by calling the `plotTraj.m` function with the results of the simulation and the sampling time. The function plots the first, third, and fifth rows of the input data array, because the angle, position, and control input signals are stored in these positions. The result is a figure with three subplots, axis labels, and LaTeX interpreted fonts.

A.3 Linear Deterministic MPC

The first part of the linear MPC script `cp_linMPC.m` is the same as LQR. The linear, discrete model is constructed, the weights are specified, and the linear feedback gain is computed. For the solution of the optimal control problem described in Section 6.1, the constrained non-linear function minimizer `fmincon` function of Matlab is used. The objective function is constructed in `cost_func_lin.m`, where predictions are made on the horizon, and the cost function `J` is calculated, as the output of `cost_func_lin.m`. The first element of the computed control input vector is used to simulate one sample time interval, and the loop starts over.

The results of the simulation are saved to `cp_linMPC_simuresult.mat`, and visualized by `plotTraj.m`. At the end of the script, training points are saved for the GPMPC simulation.

A.4 Gaussian Process MPC

The main script of GPMPC simulation is `cp_GPMPC.mat`. It loads training data from `cp_trainingpoints.mat`, which contains `xtrain` and `ytrain`, as the inputs and outputs of the GP discussed in Section 7.4. The training of the hyper-parameters are executed using PILCO's `train` function, with specified number of minimization steps. The function uses Maximum a Posteriori estimation, discussed in Section 3.2.3. Some matrices and vectors from (3.11) and (3.12) can be precomputed based only on the training data.

The calculation of optimal control input is similar to the linear MPC case, except that here the GP needs to be evaluated at all states, which is carried out in `cost_func_GP.m`, using (3.11) and (3.12), due to the mean equivalent approximation [3]. The cost function is calculated from (6.6), and minimized by `fmincon` satisfying constraints for the control input.

The results of the simulation are saved to `cp_GPMPC_simuresult.mat`, and visualized by `plotTraj.m`. At the end of the script, the code can be found for Figure 7.6, with the starting time instant k of the prediction, stored in the `start` variable.

References

[1] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger, "Learning-based model predictive control for autonomous racing," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3363–3370, 2019.

[2] H. S. Rao and B. R. Babu, "Hybrid neural network model for the design of beam subjected to bending and shear," *Sadhana*, vol. 32, pp. 577–586, 2007.

[3] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[4] D. Csuzdi and P. Antal, "Inverz inga egyensúlyozása tintasugaras nyomtató mehanizmusával," *BME Tudományos Diákköri Konferencia*, 2019.

[5] R. Sbresny, A. Getler, N. Felker, and C. Frederickson, "Implementation of an inverted pendulum pid control system using a stepper motor," 12 2016.

[6] G. Cao, E. M. Lai, and F. Alam, "Gaussian process model predictive control of unmanned quadrotors," in *2016 2nd International Conference on Control, Automation and Robotics (ICCAR)*, 2016, pp. 200–206.

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

[8] M. Liu, G. Chowdhary, B. Castra da Silva, S. Liu, and J. P. How, "Gaussian processes for learning and control: A tutorial with examples," *IEEE Control Systems Magazine*, vol. 38, no. 5, pp. 53–86, 2018.

[9] C. Molnar, A. Zelei, and T. Insperger, "Mechanical model for human balancing on rolling balance board," *Acta Polytechnica CTU Proceedings*, vol. 18, p. 32, 10 2018.

[10] J. Milton, T. Insperger, and G. Stépán, *Human Balance Control: Dead Zones, Intermittency, and Micro-chaos*, 02 2015, pp. 1–28.

[11] G. Stépán, "Delay effects in the human sensory system during balancing," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, pp. 1195 – 1212, 2009.

[12] M. Deisenroth, "Efficient reinforcement learning using gaussian processes," 11 2010.

[13] A. Girard and C. Rasmussen, "Prediction at an uncertain input for gaussian processes and relevance vector machines application to multiple-step ahead time-series forecasting," 07 2003.

[14] J. Quinonero-Candela and C. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research, v.6, 1935-1959 (2005)*, vol. 6, 12 2005.

[15] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs." 01 2005.

[16] M. Deisenroth, C. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 06 2011.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[18] G. F. Franklin, D. J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 4th ed. USA: Prentice Hall PTR, 2001.

[19] D. Bates, "Quadratic forms of random variables," 2010 (accessed October 25, 2020. [Online]. Available: http://pages.stat.wisc.edu/~st849-1/lectures/

[20] *WikiPedia: Friction*, 2020 (accessed October 25, 2020). [Online]. Available: https://en.wikipedia.org/wiki/Friction#Coefficient_of_friction

[21] D. Lemmers, "Comparison classical and reinforcement learning control based on pilco."

[22] M. P. Deisenroth and D. Fox, "Multiple-target reinforcement learning with a single policy."