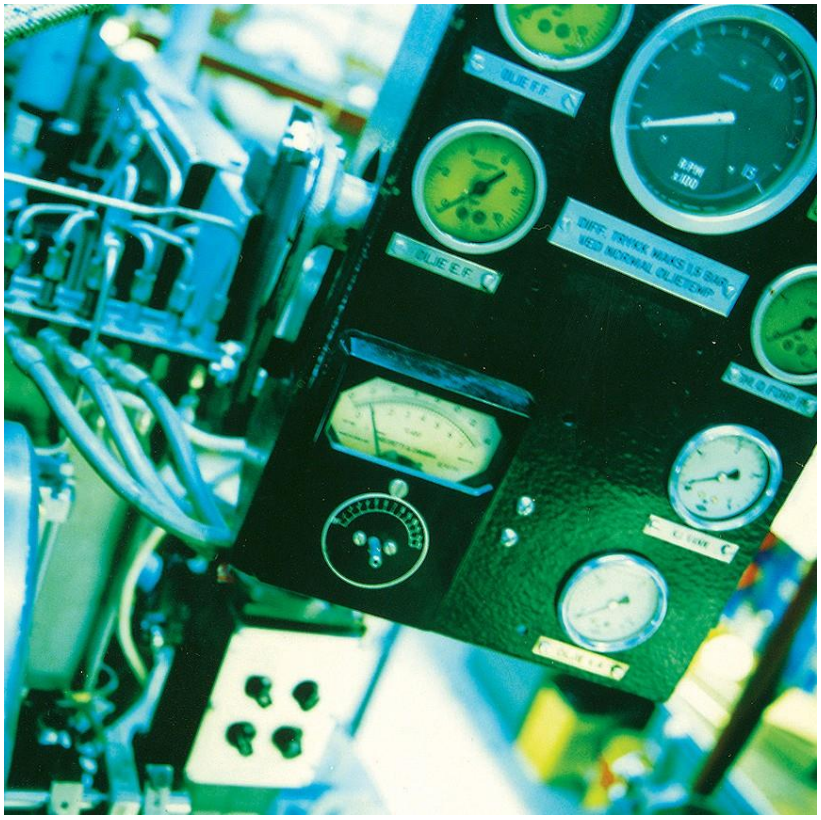
Vilius Ciuzelis

Gaussian Processes in non-linear regression

TTK4550 Fordypningsprosjekt

Trondheim 17. januar 2019



NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU
Kunnskap for en bedre verden

THIS PAGE LEFT INTENTIONALLY BLANK

Summary

Model estimation and, especially nonlinear, regression has been extensively studied in the past. The research is heavily used in various industries: manufacturing, process control, and, in the later decades, artificial intelligence (AI). This paper is looking into the possibility to use Gaussian Processes regression on an industrial level – modelling and control of a robot manipulator arm. The advantages of this method are that the modelling is non-parametric and has guaranteed convergence. No prior model of the system needs to exist to be able to model the desired system. However, the knowledge of some prior information of the system dynamics beforehand is not of no use. This information can be used in construction of *covariance function,* the heart of Gaussian Processes, and help speed up the regression process.

Gaussian Process has showed great potential in ability to interpret highly non-linear models extremely well with minimal tuning. Even though the hyperparameter optimization was not implemented fully, the resulting research showcases the potential of the method. Equally, it uncovers maybe the biggest problem with the method – its complexity. Complex derivation and implementation leads to confusion and frustration when trying to wrap one's head around the basics of Bayesian Inference.

THIS PAGE LEFT INTENTIONALLY BLANK

Contents

Summary i
List of Tables v
List of Figures v
Nomenclature v
I Introduction 1
Previous work 1
II Background theory 3
Robot model 3
Equations of Motion 3
Kinematics 4
Gaussian Processes 5
Mean function 6
Covariance function 6
Prior 7
Posterior 8
Prediction 8
Hyperparameters 9
III Simulation 12
Robot arm model 12
Controller 12
IV Results 14
V Discussion 16
VI Future Work 17
Appendix A 20
Derivation of the EOM for 2-link planer manipulator 20
Inverse kinematics for 2-link planar manipulator 21
Appendix B 24
MATLAB code 24
Manipulator.m 24
startup.m 30
GP_regression.m 30

THIS PAGE LEFT INTENTIONALLY BLANK

List of Tables

Table 1. Manipulator arm model parameters 12
Table 2. Controller gains 12

List of Figures

Figure 1. Example of overfitting 1
Figure 2. 2-link planar robot manipulator arm model 3
Figure 3. Example of multiple sample functions from the prior distribution. 7
Figure 4. Posterior distribution of a rather noisy signal 8
Figure 5. Algorithm to compute log marginal likelihood and its partial derivatives. 10
Figure 6. Motion trajectory, simulated motion and error dynamics 13
Figure 7. Prior and posterior for angular velocity of joint $q2$ 14
Figure 8. GP regression for joint $q2$ 14
Figure 9. Prior and posterior for joint angle $q2$ 15
Figure 10. GP regression for the angular velocity of joint angle $q1$ 15
Figure 11. Latin Hypercube with two variables. 17

Nomenclature

$\text{cholesky}(A)$	-	cholesky factorization. Returns L where $LL\backslash$
δ_{pq}	-	Kronecker delta function which equals to 1 iff $p = q$ and 0 otherwise
$\mathbf{f}_*$ *or* f_*	-	Gaussian process posterior prediction
$\bar{\mathbf{f}}_*$ *or* $\bar{\mathrm{f}}_*$	-	Gaussian process predictive mean
$\mathcal{GP}$	-	Gaussian process with a mean and a covariance function
J	-	moment of inertia
μ	-	mean value
m_i	-	mass of link i
$\mathcal{N}(\mu, \sigma^2)$	-	Normal (or Gaussian) distribution
n	-	number of training inputs
n_*	-	number of test inputs
$k(\mathbf{x}, \mathbf{x}')$	-	kernel (or similarity) function evaluated at $\mathbf{x}$ and $\mathbf{x}'$
$K(X, X)$	-	$n \times n$ covariance matrix
$\mathcal{L}$	-	Lagrangian
l	-	characteristic length-scale parameter
$m(\mathbf{x})$	-	mean function
σ_f^2	-	variance of the noise-free signal
σ_n^2	-	variance of the noise
T_*	-	kinetic co-energy
$\text{tr}(A)$	-	trace is the sum of the elements on the main diagonal of a square matrix. $\sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn}$
θ	-	joint angle
v_i	-	velocity of link i

V	-	potential energy
$\mathbb{V}$	-	variance in (C. E. Rasmussen, 2006)
X	-	matrix of training inputs
X_*	-	matrix of test inputs
$\mathbf{x}_i$	-	training input i
$\mathbf{y}$	-	training targets

I Introduction

Previous work

Gaussan Processes have received a lot of attention in the last decades, the leading cause being machine learning and hype around the Artificial Intelligence or AI (Hopgood 2003). The truth is, the first GP works with GP date back to as far as 1940's (Wiener, 1949). It has been used in geology (Cressie, 1990) and is extensively described by (C. E. Rasmussen, 2006), which has later become as a go-to work on GP. This particular book has therefore been used as basis for this paper.

GP is a stochastic process used in statistics and probability theory as a method for *regression* or *classification* when some data about the model is known. In terms of machine learning – *supervised learning* method. The main difficulty with finding the *correct* model is therefore the trade-off between *complexity* and *overfitting*. The model should be complex enough to model the main dynamics in the system, yet simple enough as to not model the noise, see Figure 1 below.

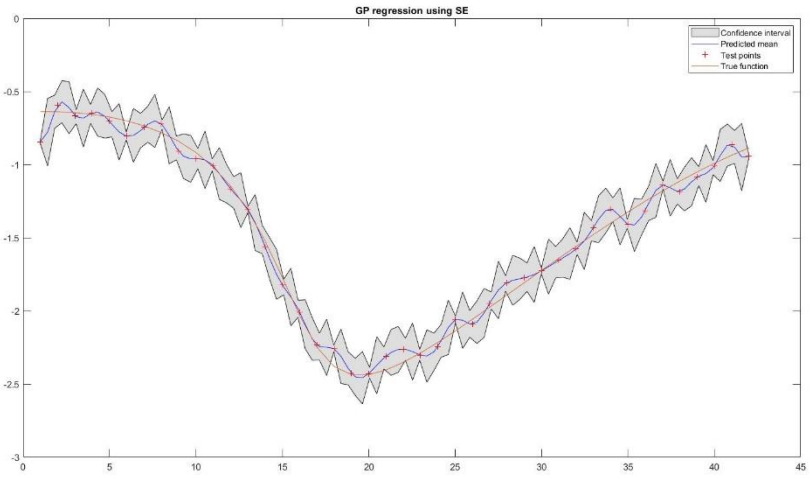


Figure 1. Example of overfitting

The fit is controlled by some free parameters, called the hyperparameters, which are then to be optimized to ensure best fit to the data at hand. There exists several other data-fitting methods in both regression and classification (Harvey Motulsky, 2004). Gaussian Processes has proven itself to be a generally strong method of data-fitting extremely nonlinear models and is easily scalable to different size models (J. Hensman 2015). It can even be combined with other control and modelling methods to enhance the method further (J. Kocijan 2004). The downsides of the method is its complexity when implementing and computational needs. The numerous inverses

of covariance matrix in GP are computationally expensive (C. E. Rasmussen, 2006).

As a test case, basic control of a 2-link planar robot manipulator arm has been chosen. The dynamics of manipulators are highly nonlinear in the sense that joints might have non-linear friction forces, elasticity and rigidity (or lack thereof) in the links is also highly non-linear (Cai 2002). Let’s take a look into the robot model itself.

II Background theory

Robot model

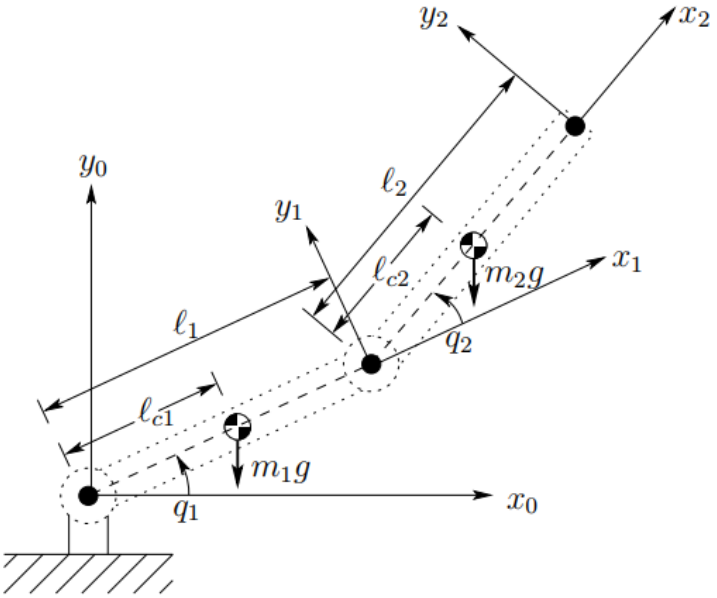


Figure 2. 2-link planar robot manipulator arm model

Physical manipulator parameters are (loosely) based on Kuka's own LBR iiwa 14 R280 (GmbH) robot. The mass of the robot was assumed to be uniform such that the masses of the respective links were simply obtained by linear combination of the lengths of the robot. To calculate moments of inertia, the arms of the manipulator are modelled as thin, uniform beams. The inertia about the center of mass is then given by

$$I_{center} = \frac{ml^2}{12}$$

The complete overview of the parameters used in the simulation is shown in Table 1 under III Simulation on page 12. Next, analytical robot model is derived.

Equations of Motion

In order to gather training data for the GP model of the robot, one needs to describe the model analytically. For that purpose, an energy-based - Lagrange method was used. Let the total energy in the system be defined as

$$\mathcal{L} = T^* - V$$

where T_* is the so-called *kinetic co-energy* and V is the potential energy in the system. Define these energies as

$$T^* = \sum_i \frac{1}{2} m_i v_i^2$$

$$V_{gravity} = mgh$$

Equations of motion (EOM) are then found using Lagrangian (Egeland, et al., 2002)

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i$$

For the extensive derivation of the equations of motion, see Appendix A. For completeness, fully-derived EOM are given below

$$\begin{aligned}\tau_1 = {} & \left(I_1 + I_2 + m_1 L_{c1}^2 + m_2(L_1^2 + L_{c2}^2 + 2L_1 L_{c2}\cos q_2)\right)\ddot{q}_1 \\ & + (I_2 + m_2 L_{c2}^2 + m_2 L_1 L_{c2}\cos q_2)\ddot{q}_2 - m_2 L_1 L_{c2}\sin q_2\,(2\dot{q}_1\dot{q}_2 + \dot{q}_2^2) \\ & + (m_1 L_{c1} + m_2 L_1) g\cos q_1 + m_2 L_{c2} g\cos(q_1 + q_2) \\ \tau_2 = {} & (I_2 + m_2 L_{c2}^2 + m_2 L_1 L_{c2}\cos q_2)\ddot{q}_1 + (I_2 + m_2 L_{c2}^2)\ddot{q}_2 + m_2 L_1 L_{c2}\dot{q}_1^2 sin\, q_2 \\ & + m_2 L_{c2} g\cos(q_1 + q_2)\end{aligned}$$

These equations can be written in a more compact form, just as in (Egeland, et al., 2002)

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau}$$

where $\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{M}^{\mathsf{T}}(\boldsymbol{q})$ is a positive definite matrix and $\boldsymbol{G}(\boldsymbol{q})$ is the gradient of the gravity potential. The matrix $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ can be selected to be

$$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \{\boldsymbol{c}_{jk}\} = \left\{\sum_{i=1}^{n} \boldsymbol{c}_{ijk}\dot{\boldsymbol{q}}_i\right\}$$

where

$$\boldsymbol{c}_{ijk} \coloneqq \frac{\boldsymbol{1}}{\boldsymbol{2}}\left(\frac{\partial \boldsymbol{m}_{jk}}{\partial \boldsymbol{q}_i} + \frac{\partial \boldsymbol{m}_{ik}}{\partial \boldsymbol{q}_j} + \frac{\partial \boldsymbol{m}_{ij}}{\partial \boldsymbol{q}_k}\right)$$

are *the Christoffel* symbols of the first kind. In this case the matrix $\dot{\boldsymbol{M}} - \boldsymbol{2C}$ is skew symmetric. When it comes to the controller, a simple PD regulator is used

$$\boldsymbol{\tau} = \boldsymbol{k}_d(-\dot{\boldsymbol{q}}) + \boldsymbol{k}_p(\boldsymbol{q}_r - \boldsymbol{q})$$

where $\boldsymbol{q}_r$ is the reference trajectory and $\boldsymbol{k}_d$ and $\boldsymbol{k}_d$ are control constants. Now that the equation of motion are described, one needs to describe the input out relationship of the system – kinematics.

Kinematics

Forward kinematics expresses the position of the end effector as the coordinates in the xy-plane as a function of joint angles and geometry of the links. These formulas are trivial to find and can be proven to have the following form

$$\begin{pmatrix} x_e \\ y_e \end{pmatrix} = \begin{pmatrix} l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \\ l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \end{pmatrix}$$

Inverse kinematics problem, or IKP for short, is sort-of an inverse of the forward kinematics problem, as the name suggests. The formulation of the problem is as follows: given the position of the end effector in xy-plane, compute *all* possible joint angles and link geometries which correspond to that particular end effector position. Several forms of the solution exist, each one with different characteristics when it comes to computation complexity and orientation of the links. The fully-derived formula is given below. For complete derivation of the formula see Appendix A.

$$q(x_e, y_e) = \begin{bmatrix} \operatorname{atan2}(y_e, x_e) \pm \operatorname{atan} 2\left(\frac{l_2 \sin q_2}{l_1 + l_2 \cos q_2}\right) \\ \operatorname{atan2}\left(\pm\sqrt{1 - \left(\frac{x_e^2 + y_e^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)^2}, \frac{x_e^2 + y_e^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \end{bmatrix}$$

where $k_1 = l_1 + l_2 \cos(q_2)$ and $k_2 = l_2 \sin(q_2)$. The analytical model of the robot arm is now fully derived. Let's take a look into the regression process.

Gaussian Processes

GP is used for making understandings about the relationships between the training and target data. Said in other words, input-output relationship or the conditional distribution of the targets, given the inputs (C. E. Rasmussen, 2006). By definition in the same book, a Gaussian Process "is a collection of random variables, any finite number of which have a joint Gaussian distribution". Given a dataset of n observations, which we call $\mathcal{D} = (\mathbf{X}, \mathbf{y})$, a GP is fully expressed by its mean and covariance functions $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$. These functions are defined as

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}\big[\big(f(\mathbf{x}) - m(\mathbf{x})\big)\big(f(\mathbf{x}') - m(\mathbf{x}')\big)\big]$$

where $f(\mathbf{x})$ is the process. A Gaussian Process using formulas above can be denoted as

$$f(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big)$$

To fully understand GP it is crucial to talk about its structure – its mean and variance functions.

Mean function

Since a GP is fully expressed by its mean and a covariance functions or matrices, let's take a look into the mean function first. There exist methods of fitting a mean to the data at hand, however it is often omitted and left zero. The reason behind this is simply that the covariance function usually takes the mean into the account when computing the variance matrix. There exist, of course, applications where the data is inherently at a slope or has an angle, but a suited covariance function will usually cover the increasing/decreasing mean. This paper will therefore be using a mean function of 0 throughout the derivations and simulations.

Covariance function

Covariance (or kernel) functions contain most of the information about the model. Assuming the mean function to be zero results in covariance function defining the whole model by itself! Covariance functions always have one or more free parameters which decide its form. To accommodate the vast range of possible nonlinear models, a number of kernel functions have been developed. Both explicit and composite functions. These functions define one or more of the following aspects of the covariance functions: stationarity, isotropy, smoothness and periodicity.

Stationarity refers to a stochastic process whose unconditional joint probability distribution, mean, and variance do not change in time. It means that $\mathbf{x} - \mathbf{x}'$ only depends on the values of $\mathbf{x}$ and $\mathbf{x}'$ and not their position (in time?). Isotropy deals with the measurement of distance. If a function is only dependent on values and not the measurement direction, then the function is called isotropic. Smoothness is defined by the expected closeness (or similarity) between input-output pairs. If the expectancy is high, the resulting function will tend to favor a more rapidly changing model rather than a slower, *smoother* model.

There exist numerous kernels functions which have different properties and are best suited for a range of applications. Therefore, the choice of the best-suited kernel function is not arbitrary. The most common and maybe simplest kernel function is the Squared Exponential (SE)

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{M}(\boldsymbol{x} - \boldsymbol{x}')\right) + \sigma_n^2 \delta_{\mathbf{xx}'}$$

where σ_f^2 is the variance of the noise-free signal, σ_n^2 is the variance of the noise, and $\delta_{\mathbf{xx}'}$ is a Kronecker delta which is 1 if $\mathbf{x} = \mathbf{x}'$ and 0 otherwise. $\boldsymbol{M}$ is simply a symmetric matrix containing the characteristic length-scales. It might take one of the following forms, depending

on the values of hyperparameters:

$$M_1 = l^2 I, \qquad M_2 = diag(\boldsymbol{l})^{-2}, \qquad M_3 = \Lambda\Lambda^{\mathsf{T}} + diag(\boldsymbol{l})^{-2}$$

Parameter l in M_2 is called *characteristic length-scale*. It decides how fast pace of change the sample functions are to have. Low l values will yield a more rapidly changing functions, while greater values will tend to smooth out the functions. Now that the basics of GP structure are covered, let's take a look at the next step of GP regression – prior distribution.

Prior

When choosing hyperparameters manually, or picking the initial values for optimization later, it is wise to check the function pool for the general form of the functions being drawn, the so-called prior distribution. This distribution is a multivariate normal (Gaussian) distribution together with a covariance matrix generated by the chosen kernel function. Firstly, one needs to compute the $n \times n$ covariance matrix $\boldsymbol{K}_*$:

$$K(\mathbf{X}_*, \mathbf{X}_*) = \begin{bmatrix} k(\mathbf{x}_1^*, \mathbf{x}_1^*) & \cdots & k(\mathbf{x}_1^*, \mathbf{x}_\mathrm{n}^*) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_\mathrm{n}^*, \mathbf{x}_1^*) & \cdots & k(\mathbf{x}_\mathrm{n}^*, \mathbf{x}_\mathrm{n}^*) \end{bmatrix}$$

Note that the prior distribution is solely dependent on the training data. One can sample the desired number of sample functions from GP using the multivariate normal distribution

$$\mathbf{f}_* \sim \mathcal{N}\big(m(\mathbf{x}), K(\mathbf{X}_*, \mathbf{X}_*)\big)$$

Prior functions can be sampled using the following equation:

$$\mathbf{f}_{\text{prior}} = \text{cholesky}(K(\mathbf{X}_*, \boldsymbol{X}_*) + \sigma_n^2 I) \cdot \mathbf{b} \qquad (1)$$

where $\mathbf{b}$ is a $n_* \times N$ matrix of random numbers drawn from a normal (Gaussian) distribution with 0 mean and standard deviation σ_f. An example of such a sample of prior functions is shown in Figure 3 below.

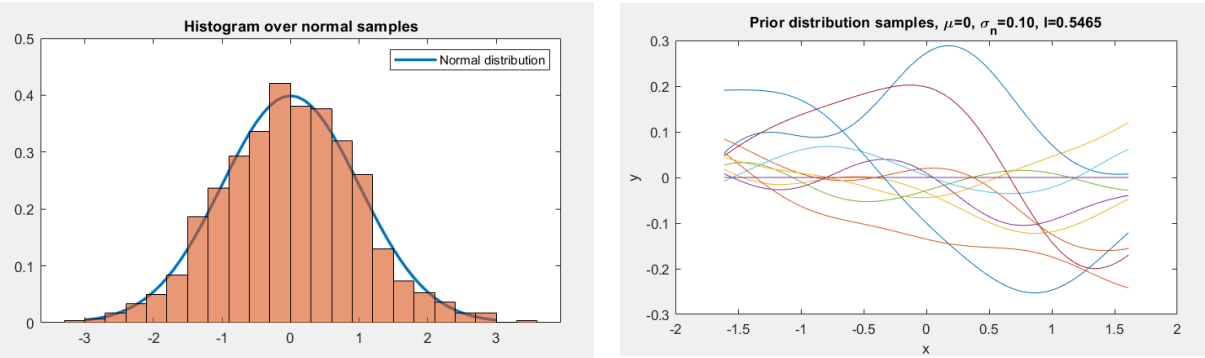


Figure 3. Example of multiple sample functions from the prior distribution.

Posterior

While prior has information about the prior functions which *might be* used in regression, posterior shows which functions *have been* used in the regression (Freitas, 2013):

$$\mathbf{f}_{\text{posterior}} = \bar{\mathbf{f}}_* + \text{cholesky}(K(\mathbf{X}_*, \boldsymbol{X}_*) + \sigma_n^2 I - L_k^\top L_k) \cdot \mathbf{b}$$

where $L_k = \text{cholesky}(K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I)^{-1} K(\mathbf{X}, \mathbf{X}_*)$ and $\boldsymbol{b}$ similar to (1). The posterior shows the uncertainty in the prediction very clearly.

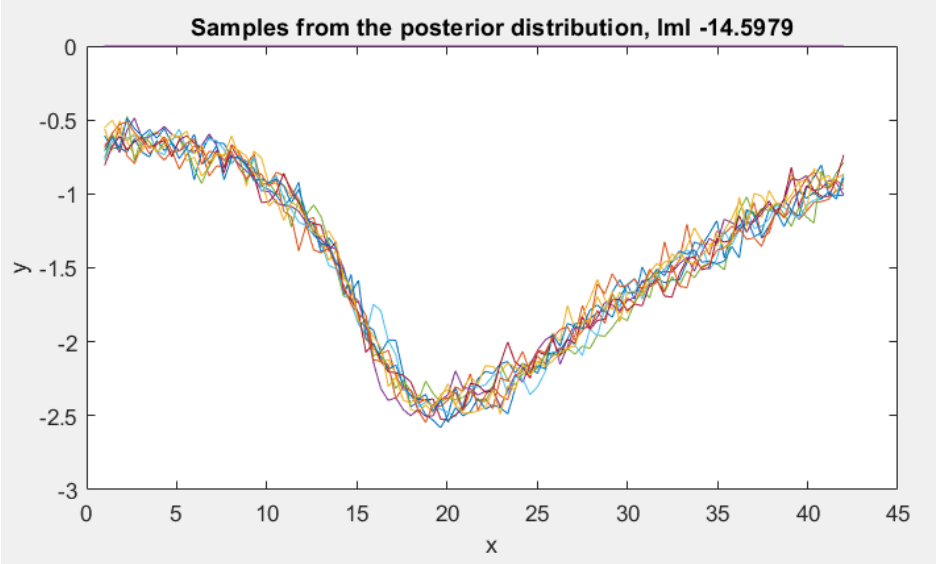


Figure 4. Posterior distribution of a rather noisy signal

Prediction

Realistic observations or measurements are never 100% accurate. Whether because of the measuring instrument noise, inaccuracies in computing or simply modelling errors, one must always account for some disturbances in the system. In robust control, the unknown disturbances are usually modelled as (white) noise. The prior on the observations should therefore be expressed (C. E. Rasmussen, 2006):

$$\text{cov}(\text{y}_\text{p}, \text{y}_\text{q}) = k(\mathbf{x}_\text{p}, \mathbf{x}_\text{q}) + \sigma_n^2 \delta_{pq} \; or \; \text{cov}(\mathbf{y}) = K(X, X) + \sigma_n^2 I$$

where δ_{pq} is a Kronecker delta which is 1 when $p = q$, otherwise 0. The joint distribution of the observations at the test points are (C. E. Rasmussen, 2006):

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}K(X,X)+\sigma_n^2 & K(X,X_*)\\ K(X_*,X) & K(X_*,X_*)\end{bmatrix}\right)$$

The fully derived conditional on Gaussian prior distribution is then

$$\bar{\mathbf{f}}_* = K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y}$$
$$\text{cov}(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)[K(X,X)-\sigma_n^2 I]^{-1}K(X,X_*)$$

Algorithm used in the simulations is the same as in (C. E. Rasmussen, 2006). The algorithm is denoted as (2.1) in the book. Given training input X, training targets $\mathbf{y}$, covariance function k, noise level σ_n^2, and test input $\mathbf{x}_*$. Define the following

$$L \coloneqq chol(K+\sigma_n^2 I)$$
$$\boldsymbol{\alpha} \coloneqq L^\top \backslash (L \backslash \mathbf{y})$$
$$\bar{f}_* \coloneqq \mathbf{k}_*^\top \boldsymbol{\alpha}$$
$$\mathbf{v} \coloneqq L \backslash \mathbf{k}_*$$
$$\mathbb{V}[f_*] \coloneqq diag\big(k(\mathbf{x}_*,\mathbf{x}_*)\big) - \mathbf{v}^\top\mathbf{v} + \sigma_n^2$$
$$\log p(\mathbf{y}|X) \coloneqq -\frac{1}{2}\mathbf{y}^\top\boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi \qquad (2)$$

where $chol(K+\sigma_n^2 I)$ returns the lower diagonal matrix L in $LL^\top = K+\sigma_n^2 I$. This is done to ensure stability since the inverse of the K matrix tends to be numerically unstable. The algorithm returns the predictive mean and variance for the noisy test data y_*. The three terms in (2) can be interpreted as penalties. The first term penalizes low the data-fit. The second term, which is only dependent on the covariance matrix K, is the complexity penalty. The model with high complexity are usually over-fitted. The last term is the normalization constant. The only way to control the data-fitting process is to change the hyperparameters of the covariance function.

Hyperparameters

The optimization of hyperparameters is rather tricky. The problem is non-linear and computationally costly. Therefore, a well-designed optimization algorithm is needed. To compute the hyperparameters using the maximization of marginal likelihood with respect to parameters (C. E. Rasmussen, 2006)

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|X,\boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^\top K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\mathbf{y} - \frac{1}{2}\text{tr}\left(K^{-1}\frac{\partial K}{\partial\theta_j}\right)$$

$$= \frac{1}{2}\mathrm{tr}\left((\boldsymbol{\alpha}\boldsymbol{\alpha}^{\top} - K^{-1})\frac{\partial K}{\partial \theta_j}\right)$$

where $\boldsymbol{\alpha} = K^{-1}\mathbf{y}$. Because of the inversion of matrix K, gradient based optimizers are advised, such that steepest descent or congujate gradient method.

The algorithm that (C. E. Rasmussen, 2006) propose to use with congujate gradient method is shown below:

input: X (inputs), $\mathbf{y}$ (± 1 targets), $\boldsymbol{\theta}$ (hypers), $p(\mathbf{y}|\mathbf{f})$ (likelihood function)
compute K — compute covariance matrix from X and $\boldsymbol{\theta}$
$(\mathbf{f},\ \mathbf{a}) := \mathrm{mode}\left(K, \mathbf{y}, p(\mathbf{y}|\mathbf{f})\right)$ — locate posterior mode using Algorithm 3.1
$W := -\nabla\nabla \log p(\mathbf{y}|\mathbf{f})$
$L := \mathrm{cholesky}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})$ — solve $LL^{\top} = B = I + W^{\frac{1}{2}} K W^{\frac{1}{2}}$
$\log Z := -\frac{1}{2}\mathbf{a}^{\top}\mathbf{f} + \log p(\mathbf{y}|\mathbf{f}) - \sum \log(\mathrm{diag}(L))$ — eq. (5.20)
$R := W^{\frac{1}{2}} L^{\top} \backslash (L \backslash W^{\frac{1}{2}})$ — $R = W^{\frac{1}{2}}(I + W^{\frac{1}{2}} K W^{\frac{1}{2}})^{-1} W^{\frac{1}{2}}$
$C := L \backslash (W^{\frac{1}{2}} K)$
$\mathbf{s}_2 := -\frac{1}{2}\,\mathrm{diag}\left(\mathrm{diag}(K) - \mathrm{diag}(C^{\top}C)\right)\nabla^3 \log p(\mathbf{y}|\mathbf{f})$ — } eq. (5.23)
for $j := 1 \ldots \dim(\boldsymbol{\theta})$ **do**
  $C := \partial K / \partial \theta_j$ — compute derivative matrix from X and $\boldsymbol{\theta}$
  $s_1 := \frac{1}{2}\mathbf{a}^{\top} C \mathbf{a} - \frac{1}{2}\mathrm{tr}(RC)$ — eq. (5.22)
  $\mathbf{b} := C\nabla \log p(\mathbf{y}|\mathbf{f})$
  $\mathbf{s}_3 := \mathbf{b} - KR\mathbf{b}$ — } eq. (5.24)
  $\nabla_j \log Z := s_1 + \mathbf{s}_2^{\top}\mathbf{s}_3$ — eq. (5.21)
end for
return: $\log Z$ (log marginal likelihood), $\nabla \log Z$ (partial derivatives)

Figure 5. Algorithm to compute log marginal likelihood and its partial derivatives.

Conjugate gradient (CG) method would then be used to optimize the hyperparameters (Andrew V. Knyazev, 2007). These methods usually are more computationally expensive than methods used in literature, i.e. kernel-alignment by (Nello Cristianini), grid search algorithms (Kupinski 2010), random search (Bergstra 2012), or Bayesian optimization-based search (Snoek 2012).

When dealing with data that is much greater than 1, it might be useful to normalize or standardize the data first. This should be done to both the training inputs and the training targets. Then the optimization can be started by initializing the hyperparameters: initialize length-scale and function variance to 1. Standardization of data is usually done with the following formula:

$$x_{standard} = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation of the data explored. The experience also

shows that higher initial values for the target function variance yield better performance when optimizing the other hyperparameters, even though you expect low noise. A small number of restarts of the optimization algorithm with different initial starting points is a possible remedy for when local optima is reached (Murray, 2008).

III Simulation

The software used in this project thesis is simply MATLAB prerelease version 2018a. The hardware is a Windows OS based laptop with Intel Core i7-7700 CPU @ 3.60 GHz and 8 GB memory.

Robot arm model

Friction coefficients in the robot joints is chosen to be simply $\begin{bmatrix} 10000 & 0 \\ 0 & 5000 \end{bmatrix}$. The behavior of the unforced system represents life-like behavior.

Model parameter	LBR iwwa 14's parameter	Value	Explanation
$\boldsymbol{l_1}$	D	$420\ mm$	Length of the 1st link
$\boldsymbol{l_2}$	$A - C - D$	$526\ mm$	Length of the 2nd link
$\boldsymbol{m_T}$	m_T	$29{,}9\ kg$	Total mass of the robot
$\boldsymbol{m_1}$	–	$9{,}6\ kg$	Mass of the 1st link
$\boldsymbol{m_2}$	–	$12{,}0\ kg$	Mass of the 2nd link
$\boldsymbol{I_1}$	–	$5640\ kgmm^2$	Inertia of the 1st link
$\boldsymbol{I_2}$	–		Inertia of the 2nd link
$\boldsymbol{g}$	–	$-9{,}81\ kgm/s^2$	Gravity constant

Table 1. Manipulator arm model parameters

Controller

PD-controller gains were found by tuning the controller manually. The final gains used in the simulation are shown in Table 2.

$\boldsymbol{k_p}$	$\begin{bmatrix} 100000 \cdot m_1 l_1 \\ 25000 \cdot m_2 l_2 \end{bmatrix}$
$\boldsymbol{k_d}$	$\begin{bmatrix} 15000 \cdot m_1 l_1 \\ 1000 \cdot m_2 l_2 \end{bmatrix}$

Table 2. Controller gains

The values of the gains might not be entirely feasible in a real-world application as they are relatively high. These gains yield an extremely fast convergence rate and fast error dynamics. Therefore, they are satisfactory based on the application of the current model.

The motion of the robot was chosen to be a circle with a diameter of the second robot arm link. This path of motion reveals a sufficient amount of robot dynamics to be analyzed by the GP.

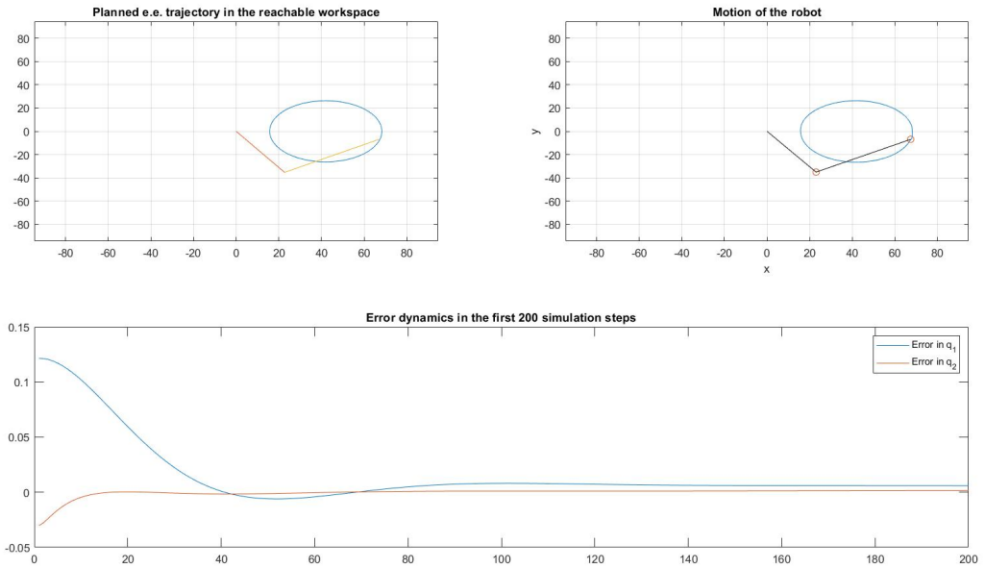


Figure 6. Motion trajectory, simulated motion and error dynamics

The error dynamics show fast convergence with minimal overshoot. They are shown in Figure 6 above.

The sampling values for the GP regression were chosen with an equal amount of spacing between the data points gathered from the robot motion. Only every 200th point of the data points was chosen, due to computational expensiveness. This sampling technique does not pose any real threats to the simulations since the motion is not prone to aliasing or really any other problems that oscillating motions have. The marginal likelihood function in GP regression did not perform as expected and has even achieved positive values, which should not be possible.

IV Results

The resulting plots are shown in figures below. The (sub-)optimal hyperparameter values were found using marginal likelihood and tuning the lengthscales values manually. Since the optimization of the hyperparameters did not work satisfactory it is left out of further discussion. The covariance function performance was very similar, even though the optimal

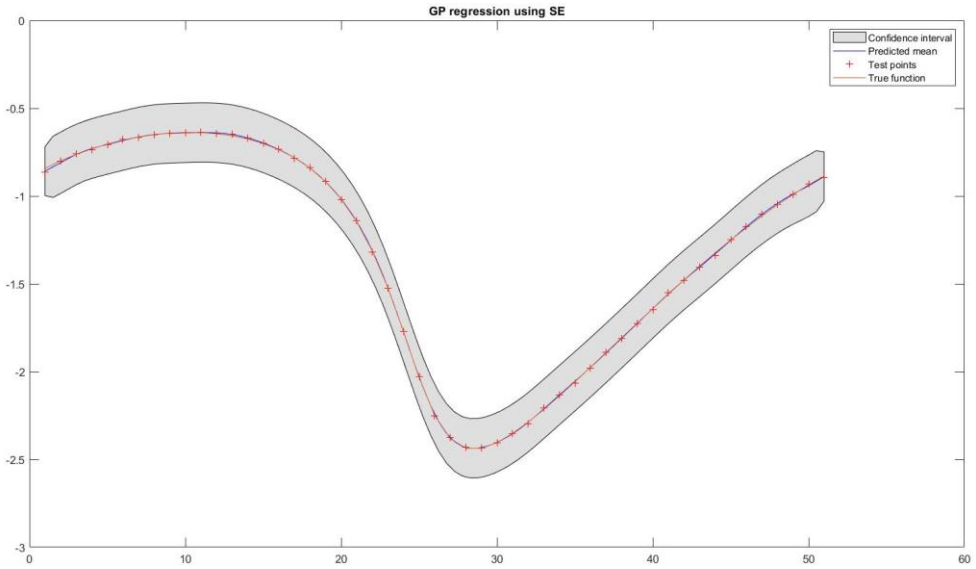


Figure 8. GP regression for joint q_2

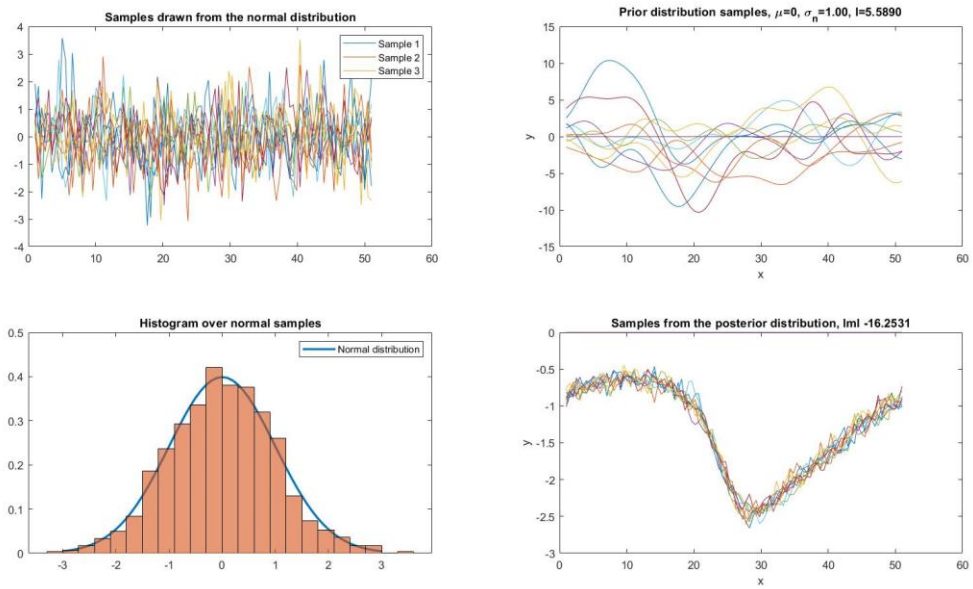


Figure 7. Prior and posterior for angular velocity of joint q_2

values were quite different. MacKay and Squared Exponential were the most promising ones yielding near-perfect regression. Some of the most interesting results are shown below. The

brown area surround the prediction is the uncertainty. It is 2 times the standard deviation, that is just above 95% confidence interval.

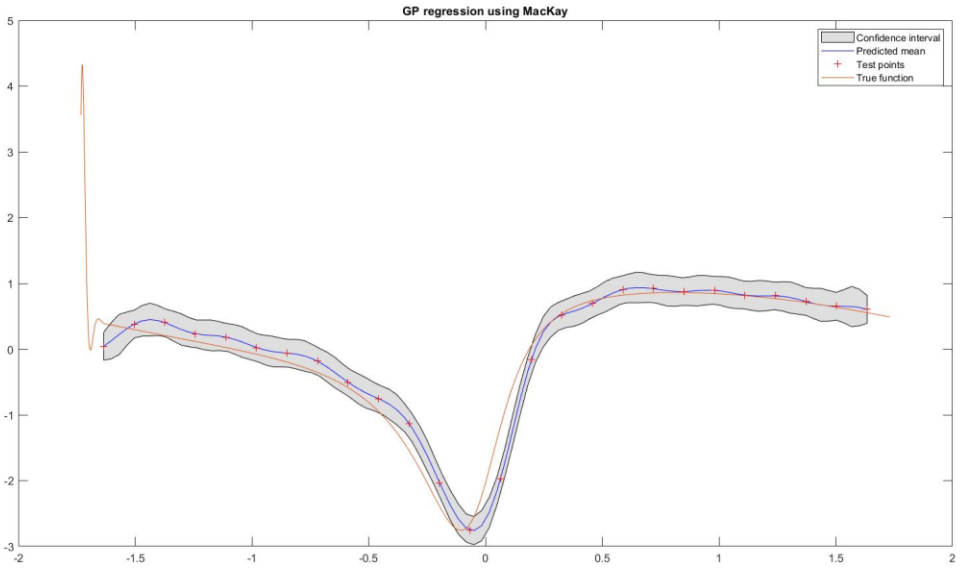


Figure 10. GP regression for the angular velocity of joint angle q_1

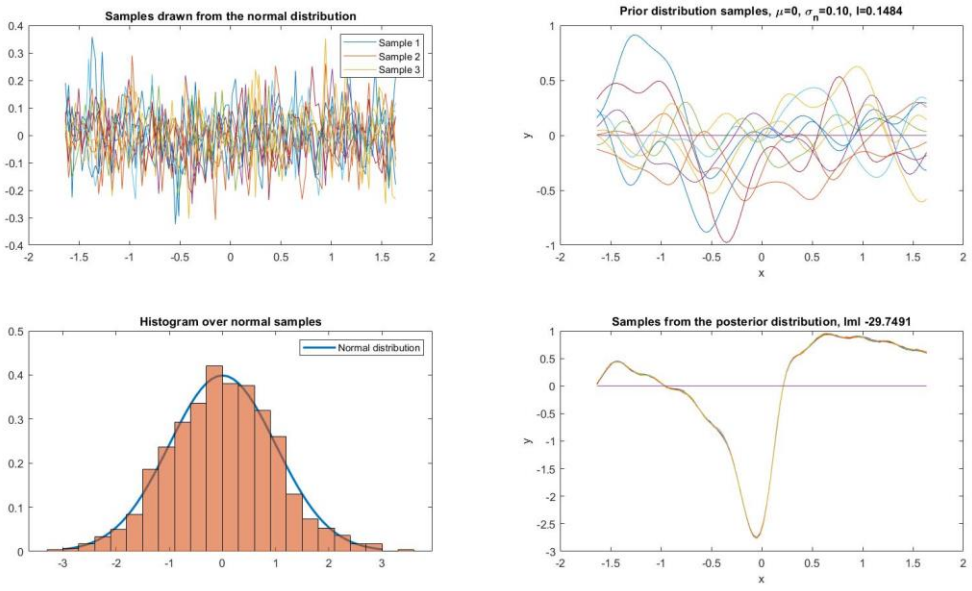


Figure 9. Prior and posterior for joint angle q_2

V Discussion

The results show great algorithm capabilities and versatility to adapt to new data. Surprisingly little effort is needed to achieve excellent results even when tuning the GP model manually. Even though the hyperparameter optimization did not work in time, the GP regression method has shown its strengths and application possibilities. Interestingly enough, failure to implement the optimizer, shows the great disadvantage with the method – its complexity. I have found numerous implementations of different parts of the GP regression and all of them seems to produce somewhat believable results. Which ones were the "correct" ones, is difficult to say. A strong statistical background is needed to understand and handle this method if it were used for control, whereas a simple PID controller is simple enough to be handled regardless of one's background.

VI Future Work

The further research ideas presented in this section are the framework for the master thesis following this paper. The first step is to implement gradient-based hyperparameter optimization and develop a sparse model. Online covariance function selector would further improve the data fit. Even though the GP is a very powerful method on its own, it would be very interesting to see it work together with some other controllers and models. Implementation of more sophisticated controllers, like (nonlinear) MPC controller, Neural Networks-based controller and other both industry and novel controllers. It would especially be interesting to see it work as a standalone model in a model reference control problem, where the requirement for the model update is of relative low frequency.

Decision theory for selecting new observations contra computational costs of the system. A bigger comparison of several techniques would be interesting. One promising technique is the Latin-hypercube sampling which maximizes the minimum distance between the sampling points (see Figure 11 below). Monte Carlo simulations and Orthogonal sampling have shown promising results and deserves some attention. Research on hyperparameter optimality conditions to check whether a true optimum has been reached or the optimizer is stuck in a local optimum.

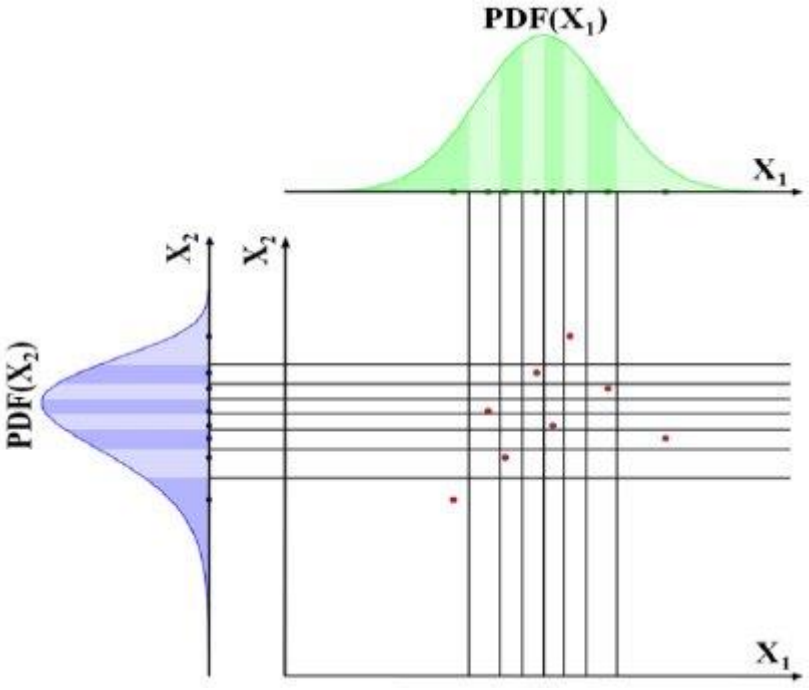


Figure 11. Latin Hypercube with two variables.

VII References

A smooth robust nonlinear controller for robot manipulators with joint stick-slip friction. **L. Cai, G. Song. 2002.** Atlanta : IEEE, 2002.
Andrew V. Knyazev, Ilya Lashuk. 2007. Steepest descent and conjugate gradient methods. [Online] April 2007. https://arxiv.org/pdf/math/0605767.pdf.
Artificial Intelligence: hype or reality? **Hopgood, A.A. 2003.** s.l. : IEEE, 2003.
C. E. Rasmussen, C. K. I. Williams. 2006. *Gaussian Processes for Machine Learning.* Massachusetts : the MIT press, 2006. pp. 13, 106, 114,.
Carl Edward Rasmussen, Chris Williams. 2018. Documentation for GPML Matlab Code version 4.2. [Online] June 11, 2018. http://www.gaussianprocess.org/gpml/code/matlab/doc/.
Cressie, Noel A. C. 2015. *Statistics for Spatial Data.* s.l. : Wiley, 2015.
Cressie, Noel Math Geol. 1990. *The origins of kriging.* s.l. : Kluwer Academic Publishers, 1990. Vol. 22: 239.
Egeland, Olav and Gravdahl, Jan Tommy. 2002. *Modeling and simulation for automatic control.* Trondheim : Marine Cybernetics, 2002. pp. 315, 320, 322, 323.
Freitas, Nando de. 2013. CPSC540 Gaussian Process. [Online] January 2013. https://www.cs.ubc.ca/~nando/540-2013/lectures/l6.pdf.
Gaussian Process - Wikipedia. [Online] https://en.wikipedia.org/wiki/Gaussian_process.
Gaussian process model based predictive control. **J. Kocijan, R. Murray-Smith, C. E. Rasmussen, A. Girard. 2004.** Boston : IEEE, 2004.
GmbH, KUKA Roboter. KUKA LBR iiwa Broschure - PDF, page 30. [Online] https://www.kuka.com/en-de/services/downloads?terms=Language:en:1;product_name:LBR%20iiwa%2014%20R820.
Harvey Motulsky, Arthur Christopoulos. 2004. *Fitting Models to Biological Data Using Linear and Nonlinear Regression.* New York : OXFORD University Press, 2004. 0-19-517179-9.
Hessmer, Dr. Rainer. 2009. Kinematics for Lynxmotion Robot Arm. [Online] October 2009. http://www.hessmer.org/uploads/RobotArm/Inverse%2520Kinematics%2520for%2520Robot%2520Arm.pdf.
Jazar, Reza N. 2010. *Theory of Applied Robotics. Kinematics, Dynamics and Control.* 2nd. s.l. : Springer US, 2010. p. 883.
Maximum-Likelihood Estimation With a Contracting-Grid Search Algorithm. **Jacob Y. Hesterman, Luca Caucci, Matthew A. Kupinski, Harrison H. Barrett, Lars R. Furenlid. 2010.** s.l. : IEEE, June 14, 2010. 11364921.
Murray, Iain. 2008. Introduction to Gaussian Processes, page 34. *University of Torronto.* [Online] 2008. https://www.cs.toronto.edu/~hinton/csc2515/notes/gp_slides_fall08.pdf.
Nello Cristianini, John Shawe-Taylor, Andre Elisseeff, Jaz Kondola. [Online] http://papers.nips.cc/paper/1946-on-kernel-target-alignment.pdf.
Practical Bayesian Optimization of Machine Learning Algorithms. **Jasper Snoek, Hugo Larochelle, Ryan P. Adams. 2012.** s.l. : Neural Information Processing Systems (NIPS), 2012.
Random serach for hyper-parameter optimization. **James Bergstra, Yoshua Bengio. 2012.** [ed.] Leon Bottou. Montreal : s.n., February 2012, Journal of Machine Learning Research 13.
Samy Youssef, Jeom K. Paik, Yang Seop Kim, Min Soo Kim, Fai Cheng. 2013. Probabilistic Selection of Ship-Ship Collision Scenarios. *Research gate.* [Online] June 2013. https://www.researchgate.net/publication/267607095_Probabilistic_Selection_of_Ship-Ship_Collision_Scenarios.
Scalable Variational Gaussian Process Classification. **James Hensman, Alexander G. de**

G. Matthews, Zoubin Ghahramani. 2015. 2015.
Technology, National Institue of Standards and. Engineering Statistics Handbook. [Online] https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm.
Wiener, Norbert. 1949. *Extrapolation, Interpolation and Smoothing of Stationary Time Series.* s.l. : MIT Press, 1949.

Appendix A

Derivation of the EOM for 2-link planer manipulator

Based on (Egeland, et al., 2002)

The planar manipulator has kinetic energy:

$$T = \frac{1}{2}m_1\vec{v}_{c1}\cdot\vec{v}_{c1} + \frac{1}{2}m_2\vec{v}_{c2}\cdot\vec{v}_{c2} + \frac{1}{2}\vec{w}_1\cdot\vec{M}_{\underline{1}}\cdot\vec{w}_1 + \frac{1}{2}\vec{w}_2\cdot\vec{M}_{\underline{2}}\cdot\vec{w}_2$$

This can be written as

$$T = \frac{1}{2}\boldsymbol{m_{11}}\dot{q}_1^2 + \boldsymbol{m_{12}}\dot{q}_1\dot{q}_2 + \frac{1}{2}\boldsymbol{m_{22}}\dot{q}_2^2$$

where

$$\boldsymbol{m_{11}} = I_{1z} + I_{2z} + m_1L_{c1}^2 + m_2(L_1^2 + L_{c2}^2 + 2L_1L_{c2}\cos q_2)$$
$$\boldsymbol{m_{12}} = \boldsymbol{m_{21}} = I_{2z} + m_2L_{c2}^2 + m_2L_1L_{c2}\cos q_2$$
$$\boldsymbol{m_{22}} = I_{2z} + m_2L_{c2}^2$$

are the elements of the inertia matrix and q_1 and q_2 are the angles between the horizontal plane and the robot arm 1 and 2 respectively. The potential energy in the system is given by

$$V = (m_1gL_{c1} + m_2gL_1)\sin q_1 + m_2gL_{c2}\sin(q_1 + q_2)$$

Then, from $\mathcal{L} = T - V$ partial derivatives are found to be

$$\frac{\partial\mathcal{L}}{\partial\dot{q}_1} = \frac{\partial T}{\partial\dot{q}_1} = \boldsymbol{m_{11}}\dot{q}_1 + \boldsymbol{m_{12}}\dot{q}_2$$
$$\frac{\partial\mathcal{L}}{\partial\dot{q}_2} = \frac{\partial T}{\partial\dot{q}_2} = \boldsymbol{m_{12}}\dot{q}_1 + \boldsymbol{m_{22}}\dot{q}_2$$
$$\frac{\partial\mathcal{L}}{\partial q_1} = -\frac{\partial V}{\partial q_1} = -(m_1L_{c1} + m_2L_1)g\cos q_1 - m_2L_{c2}g\cos(q_1 + q_2)$$
$$\frac{\partial\mathcal{L}}{\partial q_2} = \frac{\partial T}{\partial q_2} - \frac{\partial V}{\partial q_2} = \frac{1}{2}\frac{\partial\boldsymbol{m_{11}}}{\partial q_2}\dot{q}_1^2 + \frac{\partial\boldsymbol{m_{12}}}{\partial q_2}\dot{q}_1\dot{q}_2 - m_2L_{c2}g\cos(q_1 + q_2)$$

Recalling the chain rule expansion:

$$\frac{dz}{dt} = \frac{\partial z}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial z}{\partial y}\frac{\partial y}{\partial t}$$

the equations of motion are then found to be:

$$\tau_1 = \boldsymbol{m_{11}}\ddot{q}_1 + \boldsymbol{m_{12}}\ddot{q}_2 + \left(\frac{\partial\boldsymbol{m_{11}}}{\partial q_2}\dot{q}_2\right)\dot{q}_1 + \left(\frac{\partial\boldsymbol{m_{12}}}{\partial q_2}\dot{q}_2\right)\dot{q}_2 + \frac{\partial V}{\partial q_1}$$
$$\tau_2 = \boldsymbol{m_{21}}\ddot{q}_1 + \boldsymbol{m_{22}}\ddot{q}_2 + \left(\frac{\partial\boldsymbol{m_{21}}}{\partial q_2}\dot{q}_2\right)\dot{q}_1 - \left(\frac{\partial\boldsymbol{m_{21}}}{\partial\boldsymbol{q_2}}\dot{q}_1\right)\dot{q}_2 - \frac{1}{2}\left(\frac{\partial\boldsymbol{m_{11}}}{\partial q_2}\right)\dot{q}_1^2 + \frac{\partial V}{\partial q_2}$$

$$\tau_1 = \left(I_1 + I_2 + m_1 L_{c1}^2 + m_2 (L_1^2 + L_{c2}^2 + 2 L_1 L_{c2} \cos q_2)\right)\ddot{q}_1 + (I_2 + m_2 L_{c2}^2 + m_2 L_1 L_{c2} \cos q_2)\ddot{q}_2 - m_2 L_1 L_{c2} \sin q_2 \,(2\dot{q}_1 \dot{q}_2 + \dot{q}_2^2) + (m_1 L_{c1} + m_2 L_1) g \cos q_1 + m_2 L_{c2} g \cos(q_1 + q_2)$$

$$\tau_2 = (I_2 + m_2 L_{c2}^2 + m_2 L_1 L_{c2} \cos q_2)\ddot{q}_1 + (I_2 + m_2 L_{c2}^2)\ddot{q}_2 + (m_2 L_1 L_{c2} \sin q_2)\dot{q}_1^2 + m_2 L_{c2} g \cos(q_1 + q_2)$$

■

Inverse kinematics for 2-link planar manipulator

The derivation of these formulas is borrowed from (Jazar, 2010) and (Hessmer, 2009)

Start with forward kinematics formulas

$$\begin{pmatrix} x_e \\ y_e \end{pmatrix} = \begin{pmatrix} l_1 \cos q_1 + l_2 \cos(q_1 + q_2) \\ l_1 \sin q_1 + l_2 \sin(q_1 + q_2) \end{pmatrix} \qquad (1^*)$$

where all variables are according to Figure 2 on page 3.

Rewrite squares of the end effector position

$$\begin{pmatrix} x_e^2 \\ y_e^2 \end{pmatrix} = \begin{pmatrix} l_1^2 \cos^2 q_1 + l_2^2 \cos^2(q_1 + q_2) + 2 l_1 l_2 \cos q_1 \cos(q_1 + q_2) \\ l_1^2 \sin^2 q_1 + l_2^2 \sin^2(q_1 + q_2) + 2 l_1 l_2 \sin q_1 \sin(q_1 + q_2) \end{pmatrix}$$

Use Pythagorean identity

$$a\sin^2\theta + a\cos^2\theta = a^2$$

and rewrite

$$x_e^2 + y_e^2 = l_1^2 + l_2^2 + 2 l_1 l_2 [\cos q_1 \cos(q_1 + q_2) + \sin q_1 \sin(q_1 + q_2)]$$

Use the following identities

$$\sin(a \pm b) = \sin a \cos b \pm \cos a \sin b$$
$$\cos(a \pm b) = \cos a \cos b \mp \sin a \sin b$$

to prove

$$x_e^2 + y_e^2 = l_1^2 + l_2^2 + 2 l_1 l_2 [\cos q_1 (\cos q_1 \cos q_2 - \sin q_1 \sin q_2) + \sin q_1 (\sin q_1 \cos q_2 + \cos q_1 \sin q_2)]$$
$$x_e^2 + y_e^2 = l_1^2 + l_2^2 + 2 l_1 l_2 [\cos^2 q_1 \cos q_2 + \sin^2 q_1 \cos q_2]$$
$$x_e^2 + y_e^2 = l_1^2 + l_2^2 + 2 l_1 l_2 \cos q_2$$

From this follow

$$q_2 = \cos^{-1} \frac{x_e^2 + y_e^2 - l_1^2 - l_2^2}{2 l_1 l_2}$$

Since arcsin and arccos are inaccurate for small angles, use the atan2 function:

$$\begin{aligned}\theta_2 &= \text{atan2}(\sin\theta_2, \cos\theta_2)\\ &= \text{atan2}\left(\pm\sqrt{1-\cos^2\theta_2}, \cos\theta_2\right)\\ &= \text{atan2}\left(\pm\sqrt{1-\left(\frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2}\right)^2}, \frac{x^2+y^2-l_1^2-l_2^2}{2l_1l_2}\right)\end{aligned}$$

Next, use (1^*) to rewrite x and y:

$$x = k_1\cos\theta_1 - k_2\sin\theta_1$$
$$y = k_1\sin\theta_1 + k_2\cos\theta_1$$

where $k_1 = l_1 + l_2\cos\theta_2$ and $k_2 = l_2\sin\theta_2$.

Now, use the following:

$$r = \sqrt{k_1^2 + k_2^2}$$
$$\gamma = \text{atan2}(k_2, k_1)$$

This gives

$$k_1 = r\cos\gamma$$
$$k_2 = r\sin\gamma$$

This leads to

$$x = r\cos(\gamma + \theta_1)$$
$$y = r\sin(\gamma + \theta_1)$$

Finally apply atan2 function to find θ_1:

$$\gamma + \theta_1 \text{atan2}\left(\frac{y}{r}, \frac{y}{r}\right) = \text{atan2}(y, x)$$
$$\theta_1 = \text{atan2}(y, x) - \text{atan2}(k_2, k_1)$$

where $k_1 = l_1 + l_2\cos\theta_2$ and $k_2 = l_2\sin\theta_2$.

The final solution contains $\pm$ sign which can be thought of as *elbow-up* and *elbow-down* solutions. Plus sign yields the *elbow-up* solution, while the minus yields the opposite.

$$q(x_e, y_e) = \begin{bmatrix} \operatorname{atan2}(y_e, x_e) \pm \operatorname{atan} 2\left(\frac{l_2 \sin q_2}{l_1 + l_2 \cos q_2}\right) \\ \operatorname{atan2}\left(\pm\sqrt{1 - \left(\frac{x_e^2 + y_e^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)^2}, \frac{x_e^2 + y_e^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right) \end{bmatrix}$$

■

Appendix B

MATLAB code

Manipulator.m

```
%% Robot manipulator Vilius Ciuzelis

% Control parameters
sandbox = 0;
debug = 0;
h = 0.01;       % integration step length
time = 50;      % the length of simulation
hFig = figure();

% Define model constants
constants = getConstants(sandbox);

% Define trajectory
trajectory = getTrajectory(h, time, constants);
plotTraj(constants, trajectory, hFig);

% Simulate the trajectory using a planar 2-link robot
manipulator arm
data = simulate(h, trajectory, constants, sandbox);

% Present the results
animate(data, constants, hFig);
plotErrDyn(data, hFig);

% Export the data
output = struct('q_1', data.q(1,:), 'q_2', data.q(2,:),
'q_1_dot', ...
    data.omega(1,:),'q_2_dot', data.omega(2,:), 'u_1',
data.tau(1,:), 'u_2', data.tau(2,:));

%% Simulation
function res = simulate(h, traj, constants, sandbox)
    % Get variables
    [q, omega, omega_dot] = getInitialStates;
    [kp, kd] = getGains(constants, sandbox);
    f_coeff = getFCoeff(sandbox);
    storage = getStorage(length(traj));

    x_e = traj(:,1);
    y_e = traj(:,2);

    for i=1:1:length(traj)
```

```
        % -------------------------SENSE----------------
----------
        omega = omega + h*omega_dot;
        q = q + h*omega;
        [storage.l1_pos(:,i), storage.l2_pos(:,i),
storage.ee_pos(:,i)] = forwardKinematics(constants,q);
        [M, C, G] = getModel(constants, q, omega);
        % -------------------------PLAN-----------------
---------
        storage.q_r(:, i) =
inverseKinematics(constants,x_e(i),y_e(i));
        tau = kd*(-omega) + kp*(storage.q_r(:,i)-q) -
f_coeff*omega;
        % -------------------------ACT------------------
--------
        omega_dot=inv(M)*(tau-C*omega-G);
        % Storage
        storage.q(:,i) = q;
        storage.omega(:,i) = omega;
        storage.omega_dot(:,i) = omega_dot;
        storage.tau(:,i) = tau;
        storage.error(:,i) = storage.q_r(:,i)-q;
% error dynamics
    end

    res = struct('q', storage.q, 'omega',
storage.omega,...
        'omega_dot', storage.omega_dot, 'tau',
storage.tau,...
        'ee_pos', storage.ee_pos, 'l1_pos',
storage.l1_pos,...
        'l2_pos', storage.l2_pos, 'error', storage.error,
'q_r', storage.q_r);
end

%% Storage function
function res = getStorage(count)
    [tau, omega, omega_dot, ee_pos, l1_pos, l2_pos,
error, q_r] = deal( zeros(2, count) );
    q = zeros(2, count);
    res = struct( 'q', q, 'omega', omega, 'omega_dot',
omega_dot,...
        'tau', tau, 'ee', ee_pos, 'l1', l1_pos,...
        'l2', l2_pos, 'error', error, 'qr', q_r);
end

%% Storage of constants
```

```
function res = getConstants(sandbox)
    if sandbox
        l_1 = 1;
        l_2 = 0.7;
        m_1 = 1;
        m_2 = 1;
    else
        l_1 = 42;
        l_2 = 52.6;
        m_1 = 9.6156;
        m_2 = 12.0424;
    end

    l_c1 = l_1/2;
    l_c2 = l_2/2;
    g = -9.81;
    I_1 = m_1*l_1^2/12;
    I_2 = m_2*l_2^2/12;

    res = struct('l_1', l_1, 'l_c1', l_c1, 'l_2', l_2,...
        'l_c2', l_c2, 'm_1', m_1, 'm_2', m_2, 'g', g,...
        'I_1', I_1, 'I_2', I_2);
end

%% Manipulator model
function [M, C, G] = getModel(const, q, omega)
    l_1 = const.l_1;
    l_c2 = const.l_c2;
    m_2 = const.m_2;

    M_11 =
const.I_1+const.I_2+const.m_1*const.l_c1^2+m_2*(l_1^2+l_c
2^2+2*l_1*l_c2^2+2*l_1*l_c2*cos(q(2)));
    M_12 = const.I_2 + m_2*(l_c2^2+l_1*l_c2*cos(q(2)));
    M_21 = M_12;
    M_22 = const.I_2+const.m_2*l_c2^2;
    M =[M_11 M_12; M_21 M_22];

    C_11 = -m_2*l_1*l_c2*sin(q(2))*omega(2);
    C_12 = (-m_2*l_1*l_c2*sin(q(2)))*(omega(1)+omega(2));
    C_21 = m_2*l_1*l_c2*sin(q(2))*omega(1);
    C_22 = 0;
    C = [C_11 C_12; C_21 C_22];

    G_1 = -
const.g*((const.m_1*const.l_c1+m_2*l_1)*cos(q(1)) +
m_2*l_c2*cos(q(1)+q(2)));
```

```
    G_2 = -const.g*m_2*l_c2*cos(q(1)+q(2));
    G = [G_1; G_2];
end

%% Friction coefficients
function res = getFCoeff(sandbox)
    if sandbox
        res = [2 0; 0 1];
    else
        res = [10000 0; 0 5000];
    end
end

%% Initial states
function [q, omega, omega_dot] = getInitialStates
    q = [-1; pi/2];
    omega = [0; 0];
    omega_dot = [0; 0];
end

%% Forward kinematics
function [link1, link2, endEffector] =
forwardKinematics(const, q)
    link1 = [0;0];
    link2 = [const.l_1*cos(q(1)); const.l_1*sin(q(1))];
    endEffector =
[const.l_1*cos(q(1))+const.l_2*cos(q(1)+q(2));
        const.l_1*sin(q(1))+const.l_2*sin(q(1)+q(2))];
end

%% Inverse kinematics
function res = inverseKinematics(const, x,y)
    l_1 = const.l_1;
    l_2 = const.l_2;

    res = [atan2(y,x)-atan2(l_2*sin(atan2(sqrt(1-
((x^2+y^2-l_1^2-l_2^2)/(2*l_1*l_2))^2),...
    (x^2+y^2-l_1^2-l_2^2)/(2*l_1*l_2))), ...
    l_1+l_2*cos(atan2(sqrt(1-((x^2+y^2-l_1^2-
l_2^2)/(2*l_1*l_2))^2),...
    (x^2+y^2-l_1^2-l_2^2)/(2*l_1*l_2)))); ...
    atan2(sqrt(1-((x^2+y^2-l_1^2-l_2^2)/(2*l_1*l_2))^2),
(x^2+y^2-l_1^2-l_2^2)/(2*l_1*l_2))];
end

%% Controller gains
function [kp, kd] = getGains(const, sandbox)
```

```
    if sandbox
        k_d1 = 50;
        k_p1 = 500;
        k_d2 = 20;
        k_p2 = 200;
    else
        k_d1 = 30000*const.m_1*const.l_1/2;
        k_p1 = 100000*const.m_1*const.l_1;
        k_d2 = 1000*const.m_2*const.l_2;
        k_p2 = 50000*const.m_2*const.l_2/4;
    end
    kd = [k_d1 0; 0 k_d2];
    kp = [k_p1 0; 0 k_p2];
end

%% Motion trajectory
function [traj, count] = getTrajectory(h, steps, const)
    %% Define The Trajectory
    % Define a circle to be traced over the course of 10
seconds. This circle
    % is in the _xy_ plane with a radius of 0.15.
    t = (0:h:steps-h+1)'; % Time
    count = length(t);
    center = [const.l_1 0];
    radius = 1/2*const.l_2;
    theta = t*(2*pi/t(end));
    points = center + radius*[cos(theta) sin(theta)];

    traj = points;
end

%% Trajectory plotter
function plotTraj(const, traj, Hfig)
    l_1 = const.l_1;
    l_2 = const.l_2;

    [q,~,~] = getInitialStates;
    subplot(2,2,1)
    plot(traj(:,1), traj(:,2));
    grid on;
    title("Planned e.e. trajectory in the reachable
workspace");
    axis([-(l_1+l_2) (l_1+l_2) -(l_1+l_2) (l_1+l_2)]);
    hold on;
    plot([0 cos(q(1))*l_1],[0 sin(q(1))*l_1]);
    hold on;
```

```
    plot([cos(q(1))*l_1
cos(q(1))*l_1+cos(q(1)+q(2))*l_2],[sin(q(1))*l_1
sin(q(1))*l_1+sin(q(1)+q(2))*l_2]);
    hold off;
end

%% Animation
function animate(data, const, hFig)
    subplot(2,2,2);
    if nargin == 2
        hFig = figure();
    end
    d = 100;    % FPS
    j=1:d:length(data.q);

    ee_pos = data.ee_pos;
    l2_pos = data.l2_pos;

    for i=1:length(j)-1
        hold off
        plot(ee_pos(1,1:j(i)),ee_pos(2,1:j(i)), "-");
        hold on;
        plot([l2_pos(1,j(i)) ee_pos(1, j(i))],[l2_pos(2,
j(i)) ee_pos(2, j(i))],'o', ...
         [data.l1_pos(1) l2_pos(1,j(i))],[data.l1_pos(2)
l2_pos(2,j(i))],'k',... % first arm
        [l2_pos(1,j(i)) ee_pos(1, j(i))],[l2_pos(2,j(i))
ee_pos(2, j(i))],'k') % second arm
        hold on;

        title('Motion of the robot')
        xlabel('x')
        ylabel('y')
        axis([-const.l_1-const.l_2 const.l_1+const.l_2 -
const.l_1-const.l_2 const.l_1+const.l_2]);
        grid on;
        hold on;
        drawnow;
    end
end

%% Error dynamics plotter
function plotErrDyn(data, hFig)
    steps = 200;
    subplot(2,2,[3,4]);
    plot(1:steps, data.error(:,1:steps));
```

```
    txt = sprintf("Error dynamics in the first %d
simulation steps", steps);
    title(txt);
    legend("Error in q_1", "Error in q_2");
end
```

startup.m

```
% startup script to make Octave/Matlab aware of the GPML
package
%
% Copyright (c) by Carl Edward Rasmussen and Hannes
Nickisch 2018-08-01.

disp ('executing gpml startup script...')
mydir = fileparts (mfilename ('fullpath'));
% where am I located
addpath (mydir)
dirs = {'cov','doc','inf','lik','mean','prior','util'};
% core folders
for d = dirs, addpath (fullfile (mydir, d{1})), end
dirs =
{{'util','minfunc'},{'util','minfunc','compiled'}};     %
minfunc folders
for d = dirs, addpath (fullfile (mydir, d{1}{:})), end
addpath([mydir,'/util/sparseinv'])
```

GP_regression.m

```
%% GP Vilius Ciuzelis

run startup.m;                                          % For
gpml_randn()
run Manipulator.m

% Globals
global sigma_f sigma_n l gamma
sigma_f = 0.5;                                          %
standard deviation of the noise-free signal
sigma_n = 0.005;                                        %
standard deviation of the noise
l = 10.189;                                              %
Length-scale
gamma = 2;                                              % For use
in gammaExp cov. func. Value must be 2
rng('default');                                         % For
repeatability
```

```
rngseed = 5;
%% main
input = output.q_1;                                  % true
function
spacing = 100;

% Omitting the first 100 samples, due to very high
transients in u_1
Y = input(100:spacing:end)';                        % Target
data
y = Y + 0.005*gpml_randn(rngseed, length(Y) , 1 );  % Add
some noise

trainingData = 1:length(y);
X = trainingData;                                    %
Training data

m = 100;                                             % number
of test points
Ns = 10;                                             % number
of prior and posterior rnd samples
xs = linspace(X(1), X(end), m)';                     % Test
data

% Choose cov function between:
% 'SE', 'MacKay', 'Matern3/2', 'Matern5/2', 'gammaExp',
'exponential'
covFunc = 'SE';

% Sampling from prior
fPrior = getFPrior(xs, covFunc, Ns);

% GP regression
[mu, variance, lml] = getGP(X, xs, y, covFunc);

% Sampling from posterior
fPosterior = getFPost(X, xs, covFunc, mu, Ns, lml);

figure();
hold on; plot(xs,variance);hold off;
f = [mu+2*sqrt(variance); flipdim(mu-
2*sqrt(variance),1)];
fill([xs; flipdim(xs,1)], f, [7 7 7]/8);
hold on; plot(xs, mu, 'b'); plot(X, y, 'r+');
plot(1:length(Y) ...
    , Y); hold off;
```

```
legend('Confidence interval','Predicted mean', 'Test
points', 'True function');
txt = sprintf('GP regression using %s', covFunc);
title(txt);
%% Covariance function
function covariance = kernel(x, x_prime, covType)
    N = max(size(x,1),size(x,2));
    M = max(size(x_prime,1),size(x_prime,2));
    covariance = zeros(N,M);
    for i=1:1:N
        for j=1:1:M
            if nargin == 3
                switch covType
                    case 'SE'
                        covariance(i,j) =
SE(x(i),x_prime(j));
                    case 'MacKay'
                        covariance(i,j) =
MacKay(x(i),x_prime(j));
                    case 'Matern3/2'
                        covariance(i,j) =
Matern32(x(i),x_prime(j));
                    case 'Matern5/2'
                        covariance(i,j) =
Matern52(x(i),x_prime(j));
                    case 'gammaExp'
                        covariance(i,j) =
gammaExp(x(i),x_prime(j));
                    case 'exponential'
                        covariance(i,j) =
exponential(x(i),x_prime(j));
                    otherwise
                        disp('Invalid covariance function
given. Using SE instead');
                        covariance(i,j) =
SE(x(i),x_prime(j));
                end
            else
                disp('Too few arguments given. Exiting');
                return;
            end
        end
    end
end

%% Prior
function fPrior = getFPrior(X_star, covFunc, Ns)
```

```
    m = length(X_star);                                    %
number of test points
    hyp = getHyp;
    Kss = kernel(X_star, X_star, covFunc);             % m x
m
    Lss = chol(Kss+hyp.sn^2*eye(m), 'lower');          % m x
m
    b = normrnd(0, hyp.sf, m, Ns);                     % m x
Ns
    % Should I use Kss or Lss here? What's the
difference?
    fPrior = Kss*b;                                    % m x
Ns

    % Plotter functions
    figure()
    subplot(2,2,2);
    plot(X_star, fPrior);
    hold on;
    plot(X_star, 0*X_star);
    hold on;
    txt = sprintf("Prior distribution samples, \\mu=%d,
\\sigma_n=%.2f, l=%.4f",0,hyp.sf,hyp.l);
    title(txt);
    xlabel('x'); ylabel('y');

    subplot(2,2,1);
    plot(X_star, b);
    title("Samples drawn from the normal distribution");
    legend("Sample 1", "Sample 2", "Sample 3");
    hold on;

    subplot(2,2,3);
    samples = (-3:.1:3);
    norm = normpdf(samples,0,1);
    plot(samples, norm, 'LineWidth', 2);
    hold on;
    histogram(normalize(b), 'Normalization', 'pdf',
'BinMethod', 'auto');
    title("Histogram over normal samples");
    legend("Normal distribution");
    hold off;
end

%% GP regression
function [mean, variance, lml] = getGP(X, X_star, y,
covFunc)
```

```
    % This function uses algorithm 2.1 from
Rasmussen&Williams, 2006
    %
    fprintf("Using %s as covariance function for GP
regression\n", covFunc);
    n = length(X);
    hyp = getHyp;

    % Optimization of the hyperparameters
    % hyp = optimize(y, n, 'Iterations', 100, hyp);

    K = kernel(X,X, covFunc);
    Kss = kernel(X_star, X_star, covFunc);          % m x
m
    ks = kernel(X_star, X, covFunc);

    L = chol(K+hyp.sn^2*eye(n), 'lower');           % 2.
L:=cholesky(K+sigma_n^2*I)
    alfa = L'\(L\y);                                % 3.
alfa:=L^T\(L\y)
    mean = ks*alfa;                                 % 4.
f_star_bar:=k_star^T*alfa
    v = L\ks';                                      % 5.
v:=L\k_star
    variance = diag(Kss)-dot(v,v)' + hyp.sn^2;      % 6.
V[f_star_bar]:=k(x_star,x_star)-v^T*v
    dataFit = (-1/2)*y'*alfa;
    term1 = log(diag(L));
    complPenalty = (-1/2)*sum(term1(:));
    normConst = (-1/2)*n*log(2*pi);
    lml =  dataFit+complPenalty+normConst;          % 7.
log marginal likelihood;
    display(dataFit(1), 'Data fit');
    display(complPenalty(1), 'Complexity penalty');
    display(normConst(1), 'Normalization constant');
    display(lml(1), 'Log marginal likelihood');
end

%% Posterior
function FPost = getFPost(X, X_star, covFunc, mean, Ns,
lml)
    hyp = getHyp;
    n = length(X);                                  %
number of training points
    m = length(X_star);                             %
number of test points
```

```
    ks = kernel(X_star, X, covFunc);                % m x
n
    K = kernel(X,X, covFunc);                       % n x
n
    Kss = kernel(X_star, X_star, covFunc);          % m x
m

    L = chol(K+hyp.sn^2*eye(n), 'lower');
    Lk = L\ks';
    Lk = Kss+hyp.sn^2*eye(m)-Lk'*Lk;
    L = chol(Lk,'lower');

    b = normrnd(0, hyp.sf, m, Ns);                  % m x
Ns

    FPost = mean+L*b;

    subplot(2,2,4);
    plot(X_star, FPost);
    hold on;
    plot(X_star, 0*X_star);
    hold on;
    txt = sprintf("Samples from the posterior
distribution, lml %.4f", lml(1));
    title(txt);
    xlabel('x'); ylabel('y');
end

%% Squared Exponential covariance function (2.20) R&W
2006
function res = SE(arg1, arg2)
    hyp = getHyp;
    r = arg1^2+arg2^2-2*arg1*arg2;
    M = hyp.l^2;
    res = hyp.sf^2*exp((-1/2)*M\r)+hyp.sn^2*eq(arg1,
arg2);
end

%% MacKay covariance function, (4.31) R&W 2006
function res = MacKay(arg1, arg2)
    hyp = getHyp;
    r = arg1-arg2;

    res = exp(-2*inv(hyp.l^2)*sin(r/2).^2);
end
```

```
%% Matérn v=3/2 covariance function, (4.17) R&W 2006
function res = Matern32(arg1, arg2)
    hyp = getHyp;
    r = arg1-arg2;
    res = (1+sqrt(3)*r*inv(hyp.l))*exp(-
sqrt(3)*r*inv(hyp.l));
end

%% Matérn v=5/2 covariance function, (4.17) R&W 2006
function res = Matern52(arg1, arg2)
    hyp = getHyp;
    r = arg1-arg2;
    res =
(1+sqrt(5)*r*inv(hyp.l)+5*r^2*inv(3*hyp.l^2))*exp(-
sqrt(5)*r*inv(hyp.l));
end

%% Gamma-exponential covariance function, (4.18) R&W 2006
function res = gammaExp(arg1, arg2)
    hyp = getHyp;
    r = arg1-arg2;
    res = exp(-(r/hyp.l)^hyp.gamma);
end

%% Exponential covariance function
function res = exponential(arg1, arg2)
    hyp = getHyp;
    r = arg1-arg2;
    res = exp(-r*inv(hyp.l));
end

%% Hyperparameters
function res = getHyp
    global l sigma_f sigma_n gamma
    res = struct( 'l',l, 'sf', sigma_f , 'sn', sigma_n,
'gamma', gamma);
end
```

NTNU
Kunnskap for en bedre verden